UNIVERSITY OF TWENTE

FACULTY OF ELECTRICAL ENGINEERING, MATHEMATICS &
COMPUTER SCIENCE

Semantic Description of Explainable Machine Learning Workflows

Master Thesis

Patricia Inoue Nakagawa

Graduation Committee:

dr. L. Ferreira Pires
dr. F. A. Bukhsh
dr. J. L. Rebelo Moreira
dr. L. O. Bonino da Silva Santos

July 2021

Acknowledgments

First, I would like to express my sincere gratitude to my supervisors, dr. Luís Ferreira Pires, dr. João Moreira, and dr. Luiz Bonino, for their support, patience, constant feedback, and guidance during my project. Before starting my thesis, they discussed with me many possibilities and helped me choose the topic, and since the beginning of this project, they inspired me and contributed with extremely valuable ideas during our weekly meetings, introducing to me the world of ontologies and semantics, which I found very interesting and powerful, and motivated me to research this fascinating field that is explainable AI. I am also grateful for the support of dr. Faiza Bukhsh, who contributed with great feedbacks and guidance with her experience in Machine Learning experiments.

I would also like to thank dr. Núria Queralt for the great conversations about explainability, and her feedback and insights.

I would like to acknowledge my colleagues from the graduation support group, who every day supported each other in accomplishing tasks, sharing experiences, and motivating in this time of the pandemic.

I am grateful to my family for all their effort in providing me the best education, and for their love and support.

I would like to express my deepest gratitude to my husband, Carlos. Thank you for encouraging me years before starting my master's degree and for your continuous support during this process.

Lastly, I would like to thank the Orange Tulip Scholarship Program for supporting my studies.

Abstract

Machine learning algorithms have been extensively explored in many domains due to their success in learning and performing autonomous tasks. However, the best performing algorithms usually have high complexity, which makes it is difficult for users to understand how and why they achieved their results. Because of this, they are often considered black-boxes. Understanding the machine learning models is important not only to identify problems and make changes but also to increase trust in them, which can only be achieved by ensuring that the algorithms act as expected, not relying on bias or erroneous values in the data; and avoid ethical issues, not producing stereotypes, prejudiced or wrong conclusions. In this scenario, Explainable Machine Learning comprises methods and techniques that have a fundamental role in enabling users to better understand the machine learning functioning and results.

Semantic Web Technologies provide semantically interpretable tools that allow reasoning on knowledge resources, for this reason, they have been applied to make machine learning explainable. In this context, the contribution of this work is the development of an ontology that represents explainable machine learning experiments, allowing data scientists and developers to have a holistic view and better understanding of the machine learning process and the explanation process. We developed the ontology reusing already existing domain-specific ontology (ML-SCHEMA) and grounding it in the Unified Foundational Ontology (UFO), aiming at interoperability. The proposed ontology is structured in three modules: (1) the general module, which represents the general machine learning process; (2) the specific module, which specifies the machine learning process for supervised classification; (3) the explanation module, which represents the explanation process. The ontology was evaluated using a case study in the scenario of the COVID-19 disease, where we trained a Support Vector Machine to predict mortality of patients infected with COVID-19 and applied existing explanation methods to generate explanations from the trained model. The case study was used to populate the ontology with instances, thereafter, we queried the populated ontology to ensure that the retrieved information corresponds to the expected outputs and that the ontology fulfills its intended purpose.

Keywords: XAI, Machine Learning, Semantic Web Technologies, Ontology.

Contents

List of Acronyms

AI	Artificial Intelligence
COPD	Chronic Obstructive Pulmonary Disease
COVID-19	Coronavirus Disease
CQ	Competency Questions
DARPA	Defense Advanced Research Projects Agency
DL	Description Logics
DMOP	Data Mining OPtimization Ontology
DOLCE	Descriptive Ontology for Linguistic and Cognitive Engineering
GFO	General Formal Ontology
GOL	General Ontological Language
ICU	Intensive Care Unit
ILP	Inductive Logical Programming
KB	Knowledge Base
KG	Knowledge Graph
LIME	Local Interpretable Model-agnostic Explanations
MDP	Markov Decision Process
ML	Machine Learning
MLONTO	Machine Learning Ontology
MLS	ML-Schema
NN	Artificial Neural Network
ONTODM	Ontology of Data Mining
OWL	Web Ontology Language
RDF	Resource Description Framework
RDFS	RDF Schema
RIF	Rule Interchange Format
RIPPER	Repeated Incremental Pruning to Produce Error Reduction
RQ	Research Questions
RT-PCR	Reverse Transcription Polymerase Chain Reaction
SABiO	Systematic Approach for Building Ontologies
SP-LIME	Submodular Pick Module for Local Interpretable Model-agnostic Explanations
SVM	Support Vector Machine
SWRL	Semantic Web Rule Language
SWT	Semantic Web Technologies
Turtle	Terse RDF Triple Language
UFO	Unified Foundational Ontology
UML	Unified Modeling Language
URI	Universal Resource Identifiers
W3C	World Wide Web Consortium
WHO	World Health Organization
XAI	Explainable Artificial Intelligence

List of Figures

List of Tables

1. Introduction

Artificial intelligence (AI) and particularly Machine Learning (ML) have been extensively explored due to their success in learning and performing autonomous tasks, with the potential to achieve better results than humans [1] [2]. However, the algorithms usually are not transparent, generating predictions or classifications without a clear explanation of how they achieved these results. For this reason, they are often considered black-boxes [3].

To cope with this, Explainable Artificial Intelligence (XAI) is a term that refers to methods and techniques used to make the results of AI systems explainable, intelligible, transparent, interpretable, or comprehensible to humans [1]. The ML explainability is relevant because it allows the identification of necessary changes and optimization of the ML model used to generate the results, since being able to understand the model allows us to identify problems and make changes, ensuring that the system is acting adequately, improving trust and avoiding unethical issues [3].

Usually, ML models are evaluated by the accuracy of their results, but sometimes only the accuracy is not enough to choose the most suitable model. This can happen, for example, when the ML model predicts the risk of a patient having a disease by identifying the patient number as one of the characteristics that influence the result, creating spurious correlations. One model can produce lower accuracy than others but still be considered most suitable if it shows that the algorithm is not acting in an unreasonable way [3]. Thus, understanding the logic that led to such results is crucial to enable the user to select the most suitable model according to its goals and requirements.

In order to evaluate the ML model and verify if it is suitable for the task, it is important not only to understand its logic but also to have an overview of the whole ML process, since it consists of many components that influence the behavior and results of ML models. For example, the data used to train ML algorithms together with the preprocessing steps adopted to enhance the quality of the data have a significant impact on the model's performance, since ML algorithms rely on identifying patterns or regularities in data, leading these algorithms to follow bias existing in the data [4]. Also, the learning is always based on available data, and there may be differences between training data and real data [2]. Small changes in the input can make big differences in the output, which can lead to serious errors when the system is used in the real world. In addition, the input datasets are often noisy, biased, and sometimes contain incorrectly labeled samples, and without knowing that data have these kinds of problems, training the model is a tricky and challenging task [5].

Bias in data is any trend of deviation from the truth that can lead to false conclusions (Simundic, 2013 as cited by [5]). It can cause misinterpretation in ML models and for human experts, and is impossible, in practice, to gather all possible biased cases in the whole population. Usually in ML models, the population is sampled and sometimes the population cannot represent the whole scenario. For example, if a population of asthma patients that were hospitalized having pneumonia and never got any complications, the model can conclude that asthma prevents complications (Ambrosino et al., 1995 as cited by [5]). Hence, the input data need to be analyzed

to verify imbalances in data, that is, when the instances or features of one class outnumbers the other [6], and to identify unwanted correlations among features in the input dataset, such as unethical relationships or spurious correlations, which occurs when two variables are associated, but not causally related [7]. Therefore, the explainability has to be addressed from the input data step [8].

Furthermore, the evaluation of the ML implementation needs to use adequate measurements according to the task and the application, so the user can comprehend the evaluation method to correctly select the most suitable model. For example, diagnosis detection models should have high sensitivity, identifying most patients that truly have a condition, and high specificity, avoiding detecting a condition in patients that do not have it. Hence, an overview of the entire ML process, from the data input to the evaluation, would allow the user to verify if the ML model is adequate by having a better and complete understanding of the decision process and the reason why the ML model arrived at specific decisions, and identify where to make corrections and adjustments.

1.1. Semantic Web Technologies and XAI

Semantic Web Technologies (SWT) were initially introduced to make the Internet data machine-readable by encoding semantics with the data. In the scope of XAI, these techniques potentially can be applied to ML models and might enable the development of truly explainable AI-systems (Doran et al., 2017, Holzinger et al., 2017 and Holzinger et al., 2018 as cited by [1]), since they provide semantically interpretable tools and allow reasoning on knowledge resources that can help explain ML systems.

Existing solutions that aim to explain ML algorithms with SWT usually adopt these technologies as complementary sources of information in the form of ontologies, knowledge bases, and knowledge graphs, enriching the datasets with semantic knowledge and enabling the exploitation of the relationships between concepts and inference of new knowledge.

There are two main categories regarding how explanations are generated considering the part of the machine learning process the semantic resource is being used, namely (1) ante-hoc, which builds an intrinsic explainable model, using semantic resources during the machine learning training process, or (2) post-hoc, which builds a tool that applies a semantic resource after the prediction is generated from a black-box. These approaches present their explanations in diverse ways without standardized formats or information, having their advantages and disadvantages, which can be assessed in terms of effectiveness to the user's experience, coverage level regarding the instances of the dataset, and trustworthiness and fidelity to the underlying ML model.

1.2. Problem Definition

Even though there are XAI solutions that adopt SWT to make ML models explainable, current solutions usually limit their explanations to the logic of the results, especially post-hoc solutions, which try to explain black-boxes by considering only the output of the ML models or creating a

correspondence between input and output, but none of them describes or generates explanations of the other steps of the ML and explanation process.

The focus on the explanations generated by the XAI methods and the lack of information on the whole ML and explanation processes can restrict the understanding experience of the user, making it difficult to identify in which step of the ML process corrections and adjustments should take place. The overview of the ML process is important because many components influence the behavior and the results of ML models, such as the data quality, preprocessing steps, or parameter configurations of the ML model. Additionally, information from the explanation process is relevant so the user can be aware of how the explanations were obtained, if they cover only part of the instances or the whole dataset, or if they are faithful to the ML model, reflecting the real reasoning behind the model and enabling the user to truly trust in it.

A possible way to describe the main components of ML and explanation processes is to use an ontology, providing means that enable the user to have a holistic understanding of why the ML model arrived at such specific decisions. To our best knowledge, currently, there is no ontology to represent ML and explanation processes that can also provide ways to generate explanations, since existing semantic models for ML such as ML-Schema [9] have limited scope, but can be extended and specialized.

1.3. Research Questions

The main question of this study is: “Can we leverage ML post-hoc explainability to classification tasks by enabling the user to have a holistic view of ML and explanation processes using ontologies?”

The main question can be complemented with the following sub-questions for the ML and explanation process components:

- Data Input:
  - RQ1. Which data were used to train the model?
  - RQ2. How balanced are the data?
  - RQ3. How were the data preprocessed?
  - RQ4. What are the correlations of the input datasets?
- ML Algorithm and ML Model:
  - RQ5. What are the characteristics of the ML algorithm?
  - RQ6. What is the logic behind the ML model?
- Output:
  - RQ7. Why did the model generate this output?
- ML Model Evaluation:
  - RQ8. How was the ML model evaluated? What is the meaning of those metrics?
- Explanations:
  - RQ9. How were the explanations generated? How are the explanations presented to the user? How faithful are the explanations?

RQ10. How general are the explanations (do they apply to all instances)?

1.4. Research Goals

The goal of this research is to leverage ML post-hoc explainability for supervised learning, specifically classification tasks, by proposing an ontology that represents and provides a holistic overview of the entire ML and post-hoc explanation processes, which enables the user to have a better and complete understanding of those processes and complements post-hoc explanations that justify the reason why the ML model arrived at specific decisions.

The proposed ontology describes the metadata from the components of the ML process and the post-hoc explanation process that can affect the ML results, guaranteeing interoperability and common understanding by grounding the explanation process in foundational ontologies.

1.5. Methodology

This work was developed by first analyzing works related to SWT applied to Explainable ML and conducting a problem investigation to identify limitations and unexplored paths among the solutions. Then, in order to tackle the defined problem, we propose and design a solution that consists of an ontology to represent the main components of the ML process and explanation process. Our solution is then evaluated by applying it in a specific application scenario and verifying if it fulfills its intended purpose.

The construction of the ontology follows the guidelines of SABiO (Systematic Approach for Building Ontologies) [10], which is a systematic approach for ontology development that consists of five steps, which we identified as adequate to our project due to the alignment of the proposed steps with the main tasks we expected to perform. In the first step, we identify the purpose and requirements of the ontology by defining competency questions and modularization. Secondly, we perform ontology capture and formalization. In ontology capture, we select a foundational ontology and carry out knowledge acquisition, which in this project comprises identifying the components from the ML and explanation processes that need descriptions or explanations and then selecting an existing ontology of the ML domain to be reused and extended. In formalization, we develop a conceptual model by identifying and organizing relevant concepts and relations with a graphical model. In this step, we also ground the domain ontology in the foundational one, by defining the conceptual model using components of the foundational ontology. In the third and fourth steps, design and implementation, we generate an operational version of the ontology by transforming the conceptual model.

The last step consists of testing. In order to validate the proposed ontology through experimentation and examples, we develop a case study by generating an ML classification model to predict mortality using COVID-19 patients' data. We apply existing explanation methods in the ML components such as post-hoc methods. This scenario is then used as an example to create instances in the ontology. Based on this case study, necessary refinements are identified and the ontology is adjusted accordingly. Finally, we develop and run queries in the ontology to

answer competency questions, which in this project are also defined as the research questions. Figure 1 represents SABiO's proposed steps to generate and test the ontology.

Figure 1. SABiO's steps to generate and validate the proposed ontology

1.6. Structure

This report is organized as follows: Chapter 2 provides the background and definitions of useful concepts and terms concerning Machine Learning, ML Explainability, and Semantic Web Technologies. Chapter 3 presents the specification of the ontology regarding the methodology adopted for the ontology's development, the ontology purpose and requirements, and the knowledge acquisition process, where we define the explanation and description for each component of ML process and explanation process, and select the ontologies to be reused. Chapter 4 describes the ontology development, where the domain ontology is grounded in the foundational ontology and the conceptual models are generated. Chapter 5 specifies the application scenario and the application of explanation methods, populating the ontology with instances. Chapter 6 evaluates the proposed solution with queries to answer the research questions. Finally, Chapter 7 presents the final remarks of this work.

2. Background

This section presents the background and definitions of useful concepts and terms concerning Semantic Web Technologies, Machine Learning, and ML explainability.

2.1. Semantic Web Technologies

Semantic Web is defined by the World Wide Web Consortium (W3C's) as the Web of data [11], and was introduced to solve issues faced by the traditional Web that requires big effort for finding, retrieving, and exploiting information. Semantic Web solves these issues by facilitating machine interpretation of semantic information that is embedded in the Web content as metadata. Semantic Web Technologies (SWT) can help develop diverse other applications. Specifically in AI and ML systems, SWT are used to improve chatbots and intelligent assistants, to add background knowledge in areas where data are scarce, to improve accuracy and control, and to develop explainability on ML models [12].

The Semantic Web Architecture contains the concepts, technologies, and standards defined to support the development of the Semantic Web [13], and is structured in the Semantic Web Layer Cake depicted in Figure 2. In the sequel, we introduce its components, which are mentioned throughout this report.

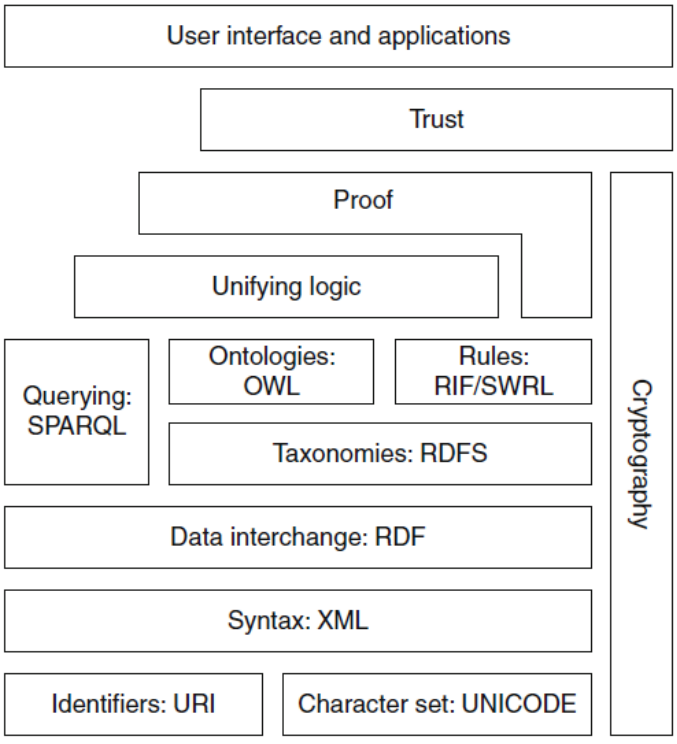

Figure 2. The Semantic Web Layer Cake [14]

We start from the bottom layer. **Unicode** is used to encode text, and **Universal Resource Identifiers (URIs)** are unique identifiers used to denote and identify concepts and the relationships between them. **XML** namespace and schema mechanisms provide syntactic descriptions of structured objects [14].

Resource Description Framework (RDF) is a data model that structures the semantic data embedded as metadata in the web content for semantically describing resources on the Web. This common structure to express knowledge allows data exchange. However, RDF has a much broader use as a generic data model for data management and reasoning [14]. It specifies a syntax with a linking structure to represent the relationships in the data model. This linking structure is defined by three components, namely subject, predicate, and object, and they can be represented as a directed labeled graph with nodes being the resources and the edges a named link between them.

The ontology languages are built on top of RDF. **RDF Schema (RDFS)** defines a simple ontology language with basic RDF statements, enabling the modeling of classes, properties, range restrictions, and hierarchies, representing a taxonomy. **Web Ontology Language (OWL)** is a family of languages to define ontologies, namely OWL Lite, OWL DL, and OWL Full. OWL extends, but it is not fully compatible with RDFS. For example, some valid RDF statements are not valid in OWL Lite or OWL DL, because DL does not support meta-statements, that is, statements over statements. In addition, RDFS is based on Logic Programming while OWL is based on Description Logics (DL), which have different semantics and data interpretation capabilities. Given this big gap, OWL2 was defined with three new languages, namely OWL2EL, OWL2QL, and OWL2RL, extending OWL with some new functionalities and relaxing some restrictions [15].

Rules are an alternative way of stating knowledge about concepts and data and specifying logical inferences, transforming or enriching data with additional specifications. **Rule Interchange Format (RIF)** is a general format to encode and exchange different kinds of rules, while **Semantic Web Rule Language (SWRL)** combines rule-based reasoning and OWL reasoning, as the union of rules and description logic, which could be seen as an attempt to approach the unifying logic [13].

Aside from these languages, **SPARQL** consists of a query language similar to SQL but applicable to RDF data models.

The upper layers have not been realized yet. The **Unifying Logic** represents the wish to bring together ontologies, rules, and queries, making them interoperable, with the logic supporting the inference of these concepts and formats. Once this logic is available, it should be possible to prove logical statements following semantic links and validate them, which is represented by the **Proof** component. The digital signatures (cryptography) together with the proof layer lead to **Trust**. On the top of the layer, we find **user interfaces and applications**, which make resources available to end-users [13].

2.1.1. Ontologies

Knowledge representation artifacts such as ontologies play a key role in the Semantic Web, providing the semantic vocabulary used to annotate websites in a way meaningful for machine interpretation. In the context of Artificial Intelligence, knowledge representation focuses on the formalization of knowledge in machine-interpretable forms allowing automated reasoning techniques to derive conclusions from it [16].

An **ontology** consists of a collection of related concepts that describes a particular domain, with definitions for objects and types of objects that provide a semantic vocabulary to define the meaning of things and can be used by applications to reason about the domain knowledge [14]. Domingue et al. [14] define an ontology as a formal, explicit specification of a shared conceptualization of a domain of interest, which is made machine-interpretable through knowledge representation techniques and can therefore be used by applications to base decisions on reasoning about domain knowledge. In other words, an ontology is a conceptual yet computable model of an application domain that is made machine-interpretable by knowledge representation techniques.

2.1.2. Semantic Data Sources

The Semantic Web generates rich sources of structured data regarding not only concepts but also facts of a domain that have well-defined meaning and are stored in systems. These concepts and facts are available as datasets that can be accessed and used by many applications [14]. Different Semantic Web concepts can be perceived as referring to data sources that are used by the solutions that apply SWT to XAI, especially Knowledge Bases and Knowledge Graphs.

A **Knowledge Base (KB)** can be seen as an ontology populated with instances or as an extension of ontologies, since ontologies consist not only of classes and properties but also instances (Daves et al., 2006, as cited by [17]). Therefore, a knowledge base differentiates ABox and TBox and contains the knowledge and an inference engine [18]. There are some possible distinctions between KB and ontologies. A KB captures information about a particular state of the domain, such as plain facts about the concrete instances, while ontologies capture the schema knowledge and interrelations, which is more general information about any possible situation. A KB can be also seen as a technical means for working with knowledge. A KB system loads specifications and instances of the ontology, which allows access and reasoning about the domain knowledge.

A **Knowledge Graph (KG)** represents knowledge about a certain domain by integrating various and heterogeneous information sources as (very large) semantic nets [19]. Sarker et al. [20] differentiate KG and ontologies, considering that the former is usually expressed using the RDF standard of triples that can be represented by graphs, while the latter attach type logic to these graphs and are usually expressed using OWL. Furthermore, a KG can also be distinguished from a KB because it has different architecture and structure [19]. A KG is less strict, without logical formulas nor separation between ABox and Tbox, sometimes with few or without assertions. Although the rigidly defined KB schema ensures data quality, maintenance, and storage

optimization, a KG allows effective and scalable data integration from various and heterogeneous sources.

2.2. Machine Learning

Machine learning consists of methods used in computers to make and improve outcomes or behaviors based on data [21]. ML is part of Artificial Intelligence, in which an intelligent system has the ability to learn and adapt to changes without being explicitly programmed, in a way that the system designer does not need to foresee all possible situations [22].

An ML algorithm uses statistics to build mathematical models, performing the core task that consists of making inferences on a sample. The model can be predictive, by making predictions in the future, descriptive, by gaining knowledge from data, or both. The model is built during the training phase that takes place so that the algorithm can learn, solving an optimization problem. The model is defined with some parameters, and the optimization of its performance criterion improves automatically through training data or past experience, which occurs by looking for data patterns and trying to make better decisions. Then, the learned model is used to carry out the inference [22].

Machine learning has outperformed humans in many areas [2], and that is why it has been adopted by a large variety of applications, such as computer vision, speech recognition, and robotics. Within these applications, ML can perform different tasks, like classification and predictions. Depending on whether or not there is feedback available to support the learning, the ML approaches are categorized as supervised, unsupervised, and reinforcement learning [22].

Supervised ML algorithms are those that learn with labeled examples, so it analyses the training data and infers a function that is used to determine the correct labels of unseen cases. The labels allow the model to compare its generated outcome with the correct one, and make changes accordingly. Classification and regression are commonly supervised ML tasks, performed by supervised ML algorithms, whose desired output is already known [22] [23].

In contrast, unsupervised ML algorithms are used when there are no labeled examples, where the aim is to find regularities in the input, performing what is called density estimation [22]. The tasks involved in unsupervised models are clustering, dimensionality reduction, and anomaly detection, among others. Examples of unsupervised algorithms are hierarchical clustering and some types of neural networks such as autoencoders [24].

Finally, reinforcement ML consists of a learning method suitable to applications that have actions as outputs and bases its behavior on a policy to maximize an expected reward, in which the policy is the sequence of correct or best actions to reach the goal. In this case, the rewards need to be defined for the agent to learn which actions are best. A basic reinforcement algorithm can be modeled as a Markov Decision Process (MDP), which defines a set of states, actions, rewards, and transition probabilities that consider specific time, action, and states. Games are common applications of reinforcement learning, which can use different algorithms such as SARSA or Q-learning [22].

2.3. Explainable Machine Learning

Explainable Artificial Intelligence (XAI) aims to make AI systems results more understandable to humans [25]. Although this term was conceived in 2004 by Van Lent et al. as cited by [25] to describe the ability of a system to explain the behavior of AI in simulation games application, the explainability problem has existed since researchers studied explanations for expert systems, in the mid-1970s (Swartout and Moore, 1988 as cited by [25]). The need for explanations occurs for many reasons. According to Keil et al. [26], explanations may highlight incompleteness, thus opaqueness can hinder optimization and evaluation, since being able to understand the model allows us to identify problems and make changes. Explanations can also improve trust in the ML model, increasing the acceptance of the results by convincing users and encouraging their use.

In addition, explanations are important because the available data can contain bias and erroneous values, producing stereotypes, prejudiced or wrong conclusions. The algorithm also cannot ensure that the data were obtained in ways that ensure privacy and were based on consent [4]. For instance, an algorithm can make use of unethical correlations for insurance companies or in the process of hiring candidates, which can cause unethical issues and misconceptions of realities. Within these circumstances, the European Union General Data Protection Regulation has decreed the citizens' right to explanation [27]. They encourage the combination of different disciplines such as machine learning and deep learning with symbolic approaches to improve the explainability of AI outcomes.

The United States Department of Defense also has XAI as one of the DARPA (Defense Advanced Research Projects Agency) programs expected to enable the "third-wave AI systems", where the context and environment are understood by machines, and they are able to explain their rationale, convey how they behave in the future, and characterize their strengths and weaknesses. The DARPA XAI program aims to pursue, during the years from 2017 to 2021, a portfolio of methods with a variety of techniques that produce explainable models that maintain the learning performance (prediction accuracy), enabling humans to understand and appropriately trust the models and their decisions, and providing future developers a range of XAI design options [28].

Figure 3 illustrates the XAI concept used in the DARPA XAI program. Today, with current ML models, users have difficulties in understanding the model and their decisions. XAI provides explanations to users that enable them to understand the decisions of the system, their overall strengths and weaknesses, convey how they will behave in future or different situations, and possibly permit users to correct the system's mistakes.

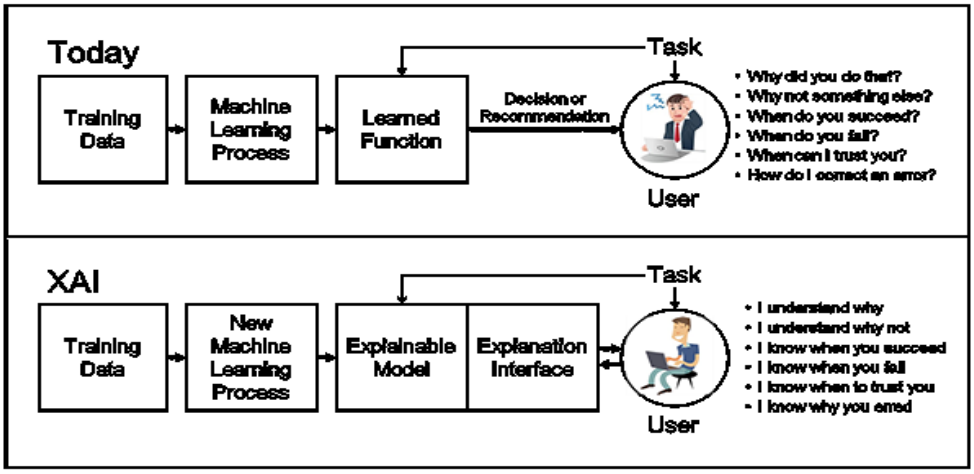

Figure 3. DARPA XAI concepts [28]

Among ML models, there are inherently intelligible algorithms, as opposed to inscrutable ones. Weld and Bansal [2] identify the first by the use of counterfactuals, which means the model is inherently intelligible to the degree that a human can predict how a change to a feature in the input can affect the output, which usually occurs in linear regression and Naive Bayes algorithms. For these algorithms, XAI and its benefits are more easily achieved. Inscrutable models can produce better results, but they are more complex and hard to explain, therefore it is more challenging to understand the reason for their results, such as complex neural networks or deep learning. In the literature, a trade-off between accuracy and interpretability of intrinsic explainable models is often mentioned [25]. This happens usually because the most accurate models are inscrutable and not very explainable, while the inherently intelligible are more interpretable but less accurate. However, this is not a static trade-off, but a dynamic target that researchers try to reach.

There are many terms in the literature related to explainable ML, such as intelligible, transparent, interpretable, or comprehensible to humans, but there is not a consensus in the literature when it comes to the definition of these terms. According to Adadi and Berrada [25], the terms explainable and interpretable are often used synonymously, while some authors also adopt understandability, comprehensibility of intelligible AI to refer to the same issue. Even though “explainable” is the keyword in the XAI, the term “interpretable” is more used in the ML community, so they differentiate both terms, defining interpretable systems as the ones that allow users to study the mathematical mapping from inputs to outputs while explainable ones provide an understanding of the logic. Explainability is closely related to interpretability, where interpretable systems are explainable if their operations can be understood by humans. The explanation for decisions, in turn, is the need for reasons or justifications for outcomes, rather than describing the inner workings and logic of reasoning behind the decision-making process. In this report, we follow the definition of Adadi and Berrada [25] in the sense that explanations focus on making the reason for the results understandable by humans.

According to Hoffman et al.[29], explanations are interactions that should enable users to quickly develop a suitable mental model that would permit the audience to develop appropriate trust and

perform well when using the system, considering the context of the ML system and their audience. Mental models, in cognitive psychology, are representations or expressions of how a person understands some sort of event, process, or system (Klein and Hoffman, 2008, as cited in [29]). In XAI, it is the user's understanding of the AI system [29]. The researcher that develops the XAI should learn what is useful about a user's mental model, and what limits the user's understanding. Many techniques can assess the effectiveness of the explanations. For example, users can be questioned about the steps or major components to check their understanding of the systems' functioning, probe questions can be used so they imagine circumstances or situations that could lead to errors, predictions tasks, and counterfactual reasoning to verify what they think that could happen next, diagramming to convey the understanding to the researcher, or self-explanation questions in which the users express their understanding or reasoning, allowing the researcher to directly access their mental model.

Arrieta et al. [30] emphasize the importance of the audience to explanations, encompassing the challenge of how to better present the explanation about how the result was obtained and for whom, in order to pass the message clearly and effectively. This comprises different manners to present the information and different purposes of explainability, considering the target audience as a key aspect when explaining ML models, taking into account the user needs, which prior knowledge they already have, their goals, and why they need explanations. In this context, [30] defines five types of audiences and identifies different goals for each of them. The first consists of domain experts such as medical doctors and insurance agents, who aim at explainability to trust the model and gain scientific knowledge, while the second, namely users affected by model decisions, want to understand their situation and verify fair decisions. For regulatory entities and agencies, it enables them to certify model compliance with the legislation and audits. For data scientists, developers, and product owners, it helps ensure or improve product efficiency, research, and new functionalities. Finally, for managers and executive board members, it allows assessing regulatory compliance and understanding corporate AI applications.

2.4. Explainable ML and Semantic Web Technologies

Many solutions that combine ML with SWT to generate explanations of results or to obtain explainable ML models can be found in the literature, where SWT are used as complementary sources of information that enriches the datasets with semantic knowledge, enabling the exploitation of the relationships between concepts and inference of new knowledge [1].

Ante-hoc and Post-hoc Methods

We can categorize the solutions taking into account which part of the ML process the semantic resource is being used, namely (1) ante-hoc and (2) post-hoc methods. Figure 4 depicts schematically both solutions, and Table 1 summarizes their advantages and disadvantages. Both methods aim to explain the outcomes of the learning models by answering questions such as "why does the model generate this outcome?" and "which are the features that are considered to make this decision?", but they answer them in different ways, in that explanations are based on results or generated considering the internal functioning of the learning model.

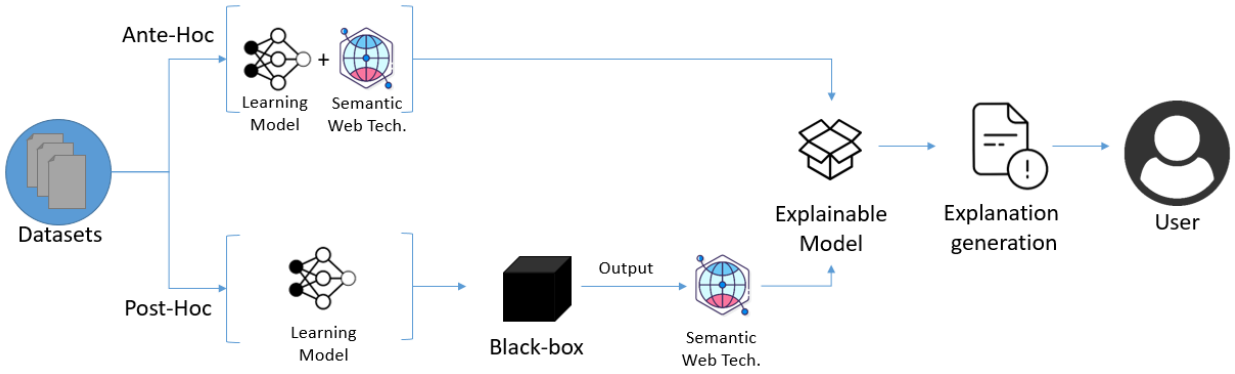

Figure 4. ML explanation approaches

Ante-hoc builds an intrinsic explainable model, using semantic resources during the ML training process to build explainable learning models that generate predictions together with explanations of its reasoning. In this case, the semantic source is integrated intrinsically to the ML algorithm to obtain explanations considering the internal functioning of the model, by mirroring the structure of knowledge graphs, using knowledge resources as embeddings, or exploring the ontology taxonomy, among others. This solution aims not only to express the reasons why certain outcomes were generated but also to develop interpretable models that are able to explain the internal mechanism and mathematical logic of the solution.

Ante-hoc solutions can generate intrinsic explainable models that facilitate the understanding of how the outcomes were obtained and enable the exploration of the internal functioning of the learning model. Although these solutions do not explain in detail all the steps taken to generate the outcome, the transparency of the models facilitates the understanding and enables adequate changes in the ML when necessary.

However, in ante-hoc solutions, each type of ML algorithm needs different adaptations to make it explainable. This happens usually due to changes in the algorithm necessary to incorporate the background knowledge or forced design choices, resulting in a bias towards explainability. Consequently, the solutions are often model-specific and sometimes also domain-specific. These changes can affect the performance of existing models regarding accuracy and efficiency, resulting in less appropriate and versatile outcomes. Moreover, these specific solutions are not always easily scalable and the efficiency and the performance can possibly be affected when compared to non-explainable models.

Post-hoc explainability consists of wrapping fully black-box trained models and adding an explainability layer [31]. SWT can be applied in the explainability layer to help explain the outputs after they are generated by the ML model. Here, the ML algorithm is normally run without any changes and the results go to another tool that maps them onto entities of a knowledge graph and generates explanations for those results, based on Inductive Logical Programming (ILP), local approximations, or on the KG relations between mapped items.

The biggest advantage in post-hoc solutions is that they are model-agnostic, that is, the explanations are separated from the ML model, thus no change is needed to the ML models so

that the solutions can be used across different models, as in the work of Ribeiro et al. [3] and Musto et al. [32]. In this case, the ML algorithms run without any interference, and the explanation tool can be reused to explain the results of different models. Post-hoc solutions explain the logic of the output by trying to justify the reason why the ML model generates the results. The use of SWT empowers the tools by expanding their knowledge without requiring prior experience, creating explanations for patterns or questions that go beyond the data analyzed.

Nonetheless, the major problem with these explanations is that they are not truthful to the underlying ML algorithm, raising concerns related to trust, reliability, and fidelity. This occurs because the explanations result from artifacts that mimic the behavior of the black-box, based on hypotheses that do not take into account the internal functioning of the ML such as node activations, nor the actual knowledge that the ML model gets from the data. Consequently, the explanations focus on how an output relates to some representations of interest (for example, which relation does the result have with the class it was classified), but do not present the behavior of the algorithm or how the ML model has been obtained in the learning phase. Furthermore, most post-hoc solutions focus on local explanations, that is, generate explanations for a single output. Few solutions focus on global explanations, which clarify the whole performance of the model.

Table 1. Summary of pros and cons related to post-hoc and ante-hoc types

Model type	Pros	Cons
Ante-hoc	● Develop intrinsic interpretable models ● Facilitate the understanding of how the outcomes have been obtained ● Enable the exploration of the internal functioning of the learning model	● Require changes and adaptations in the model ● Force design choices and bias towards explainability ● Changes can make algorithms less efficient ● Changes may affect the performance of existing models, possibly resulting in less capable and versatile outcomes ● Model-specific ● Might not be easily scalable
Post-hoc	● Explain the logic of the output ● Do not require changes in the learning model ● Can possibly achieve state-of-the-art results of inscrutable models ● Model-agnostic ● Can be reused to explain the results of different models	● Unfaithful and untrustworthy to the black-box model ● Generate unrealistic explanations, possibly leading to wrong conclusions and incorrect adaptations in the learning model ● Rely on hypotheses that consider outputs ● Usually focus on local explanation, not global

2.5. Explainable ML Tools

In this section, we present some tools that generate explanations from ML models. We focus especially on post-hoc solutions because they have the advantage of not requiring any change in the ML model to generate explanations. This enables the identification of a pattern in post-hoc explanation workflows and the design of a solution that can represent processes involving different methods. These solutions do not necessarily adopt SWT but are well-known among the scientific community.

2.5.1. RIPPER

Models based on decision rules in the format of IF-THEN statements are considered one of the most interpretable since this structure semantically resembles natural language [21]. Therefore, the opaqueness of inscrutable ML models can be remedied by extracting rules that mimic the black-box as closely as possible, since some insight is gained into the logical workings of the ML model by obtaining a set of rules that mimic the model's predictions [33].

The usefulness of a decision rule is defined by its coverage, that is, the percentage of instances to which the condition of a rule applies, and accuracy or confidence of a rule, which measures how accurate the rule is in predicting the correct class for the instances to which the condition of the rule applies, for example, one rule can predict the correct class for 80% of the instances covered by the rule [21].

One algorithm that can be used to obtain rules is RIPPER (Repeated Incremental Pruning to Produce Error Reduction) [34], which is a rule induction technique that learns rules directly from a set of training examples. According to Martens et al. [33], RIPPER can be used to extract human-comprehensible descriptions from opaque models. RIPPER learns rules by sequential covering, which is illustrated in Figure 5. In step 1, it learns one rule from the data. In step 2, it removes the data points that are covered by the rule. Then, as shown in step 3, the algorithm reiterates the remainder of the data. The learned rule needs to be highly accurate for predicting one class. If the accuracy of the rule is above a threshold, the rule is added to the rule set, otherwise, the algorithm terminates. The algorithm sorts the rules by accuracy to avoid overlapping rules.

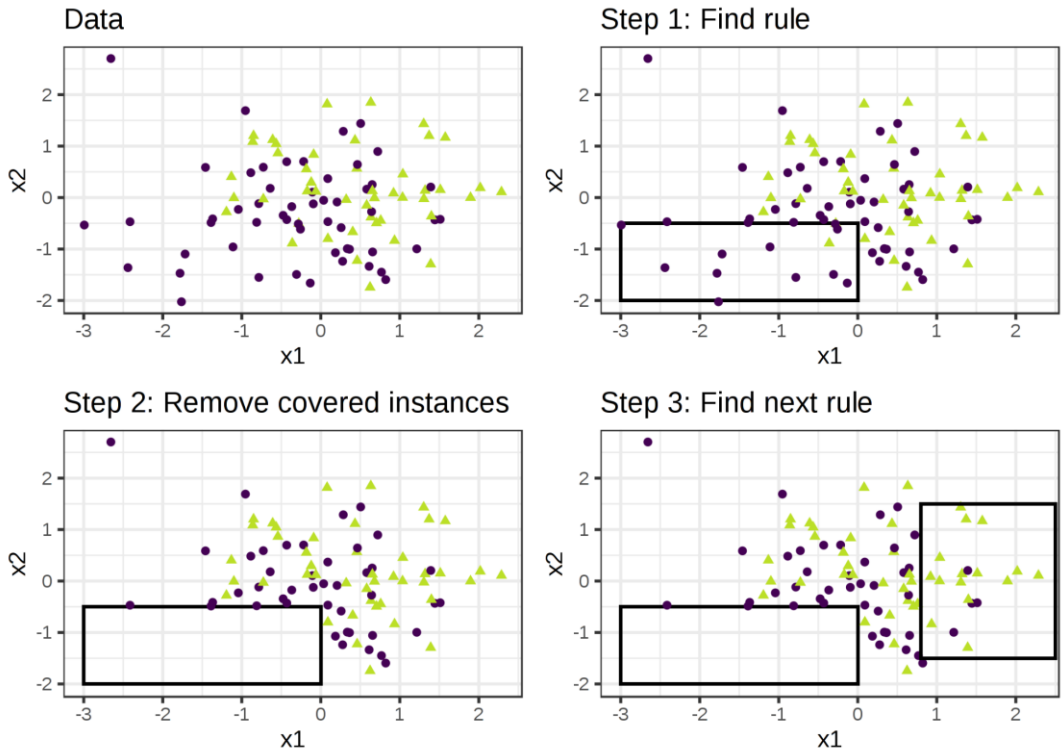

Figure 5. Illustration of how a sequential covering algorithm works to extract rules [21]

2.5.2. LIME

LIME (Local Interpretable Model-agnostic Explanations) [3] is one of the most popular solutions in the academic community to ML explainability. It consists of a post-hoc model-agnostic tool that identifies an interpretable model that is locally faithful to the black-box classifier.

For an original instance that is being explained, LIME samples uniformly at random instances around it and approximates a function to interpretable models, such as sparse linear models. This function represents the probability that the instances belong to a certain class. The approximation is done by minimizing a distance function, called locality-aware loss, which measures how unfaithful the interpretable model is in approximating the probability function in the locality. In this function, the samples are weighted by locality, such that samples in the vicinity of the original instance are assigned with higher weight and samples far from the original instance, with lower weights [3]. The intuition about how this solution works is presented in Figure 6(a), where the black-box function is represented with a blue and pink background. The instance to be explained is the bold red cross and the other crosses and circles are the sampled instances in the vicinity of the original instance, weighted by the proximity to it. The dotted line is the local learned explanation.

To determine the trustworthiness of the ML model, and not only of the instance, LIME introduces the Submodular Pick module (SP-LIME), which selects a set of representative instances and their explanations. These instances have to be non-redundant and globally representative.

For this, SP-LIME first creates an explanation matrix that represents the local importance of interpretable components for each instance. Features that explain many different instances have higher importance scores. For text applications, words that cover the maximum number of documents have the highest importance. This way, a global understanding of the model is achieved by explaining a set of individual instances, enabling a better selection between models, not depending only on accuracy. Figure 6(b) provides a visual representation of the SP-LIME matrix explanation for texts. Rows are the instances, in this case, the documents, while the columns correspond to the features (words). The column that represents feature 2 (f2) has the highest importance since it covers most of the documents and rows 2 and 5 would be selected by SP-LIME because they together cover all the features, except for f1.

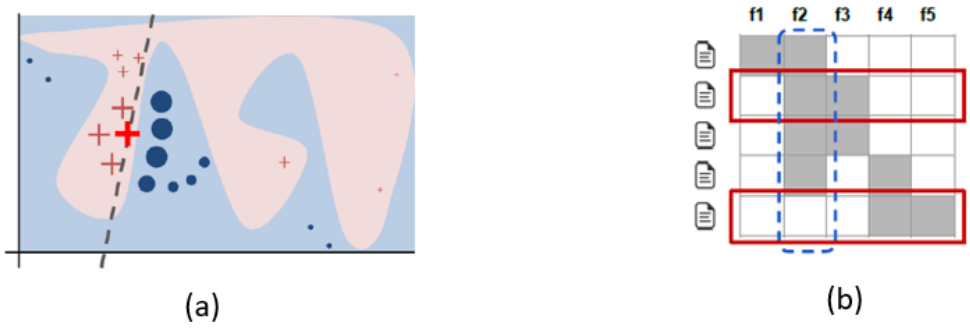

Figure 6. (a) Representation of the intuition for LIME (b) SP-LIME matrix explanation [3]

The experiments in [3] show that LIME generates faithful explanations to the model because it provides more than 90% recall, which is the fraction of retrieved relevant instances from the total relevant amount. LIME individual predictions are trustworthy since most of the predictions do not change when untrustworthy features are removed from the instance. In addition, the authors indicate that users could select with the help of LIME explanations the best model between two that have the same accuracy, but one presents spurious correlations, indicating that SP-LIME explanations are good indicators of generalization.

3. Ontology Specification

The first phase to build the ontology consists of the ontology specification, where we specify which methodology will be adopted as a guideline, as well as the goal, scope, and requirements for the ontology. This is a preparation stage before ontology development. During the specification, we also perform knowledge acquisition to find knowledge sources such as other ontologies aiming at the reuse of already established conceptualizations.

3.1. Overview of the Ontology Development Process

The ontology development of this project follows the guidelines of SABiO [10], which proposes a process for the development of domain ontologies based on foundational ontologies.

The SABiO development process consists of five main steps and supporting processes that are performed in parallel to the main development process, as depicted in Figure 7. The five steps are (1) purpose identification and requirements elicitation; (2) ontology capture and formalization; (3) design; (4) implementation; and (5) test. SABiO also distinguishes reference and operational ontologies, where reference ontologies are developed in the two first steps and the operational ontologies should follow the design and implementation steps of the process.

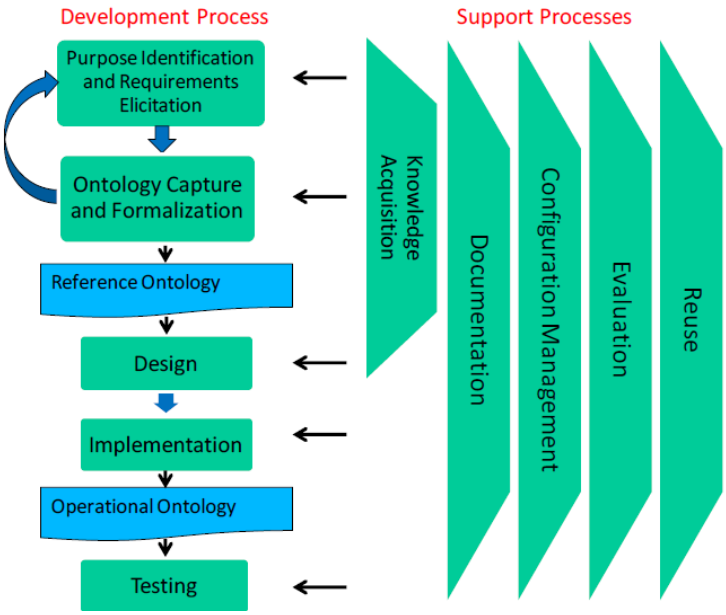

Figure 7. SABiO's Processes [10]

SABiO also proposes roles, which are considered in each step of the process. The main roles are the domain expert, who is the specialist in the domain; the ontology user; the ontology engineer, who is responsible for the reference ontology; the ontology designer, the ontology programmer, and the ontology testers, who are responsible for each of the last steps.

SABiO's first step, purpose identification and requirements elicitation, consists of defining the purpose and intended uses of the ontology. The requirements can be divided into functional and non-functional. The functional requirements are related to the content of the ontology and can be stated as questions that the ontology should be able to answer, known as competency questions (CQ). Non-functional requirements are aspects not related to the content of the ontology, such as usability and interoperability. In this step, modularization can be analyzed by identifying sub-ontologies if the domain is complex. The purpose and requirements of our ontology are defined in Section 3.2.

Knowledge acquisition is a supporting process that helps especially the first stages of ontology development by gathering knowledge from different sources, for example, from domain experts and other sources such as books and reference models. A foundational ontology needs to be selected, since SABiO suggests that the concepts and relations of the domain ontology should be analyzed considering a foundational ontology. Details of the knowledge acquisition process carried out by this project are presented in Section 3.3.

After the ontology specification, where we defined the methodology, goal, scope, requirements, and knowledge sources of the ontology, we moved to the second phase, ontology development. This phase comprises the ontology capture and formalization that generates the reference ontology and the design and implementation of the operational version of the ontology. Ontology capture and formalization, which is the second step of SABiO, consists of capturing the domain conceptualization based on the competency questions, which is strongly supported by the knowledge acquisition process, to generate the reference ontology. The concepts and relations can be identified and organized by adopting a graphical model, which supports communications and consensus among domain experts. The authors suggest the use of OntoUML, which is an ontology representation language suitable for reference ontologies and incorporates into the UML class diagram foundational distinctions of the Unified Foundational Ontology (UFO). In this step, axioms should be specified and later formalized. The ontology capture and formalization for this project are described in Sections 4.1 to 4.5.

Steps 3 and 4 aim to generate an operational version of the ontology. The objective of the design step is to bridge the gap between the conceptual modeling and code the operational ontology, thus it is necessary to complement the non-functional requirements with technological aspects and make definitions of the implementation environment, architectural and detailed design. The implementation step comprises implementing the ontology in the operational language. These steps for this project are detailed in Section 4.6.

Finally, testing consists of verifying and validating the ontology by instantiating data to the ontology and implementing competency questions as queries to the operational ontology. The instantiation of the ontology defined in this project is detailed in the case study of Chapter 5 and the evaluation with the queries is presented in Chapter 6.

3.2. Ontology Purpose and Requirements

The purpose of developing the domain-specific ontology in this project is to represent the entire ML process and post-hoc explanation process, enabling data scientists to have a holistic view and better understanding of those processes, aiming to complement and leverage the post-hoc explanations. The ontology captures the concepts of the domain and makes them machine-interpretable, making it possible to keep track of the steps from the processes and retrieve information from them.

The ontology must comply with functional and non-functional requirements. The functional requirements are related to the knowledge or content of the ontology, therefore can be stated as competency questions. We adopt the research questions of this project defined in Section 1.3 as the competency questions that the ontology should be able to answer. The definitions of the competency questions followed a top-down approach, stating first the main research question of this project and decomposing it into simpler ones that are applied to components of the ML and explanation process, which are represented in Figure 8.

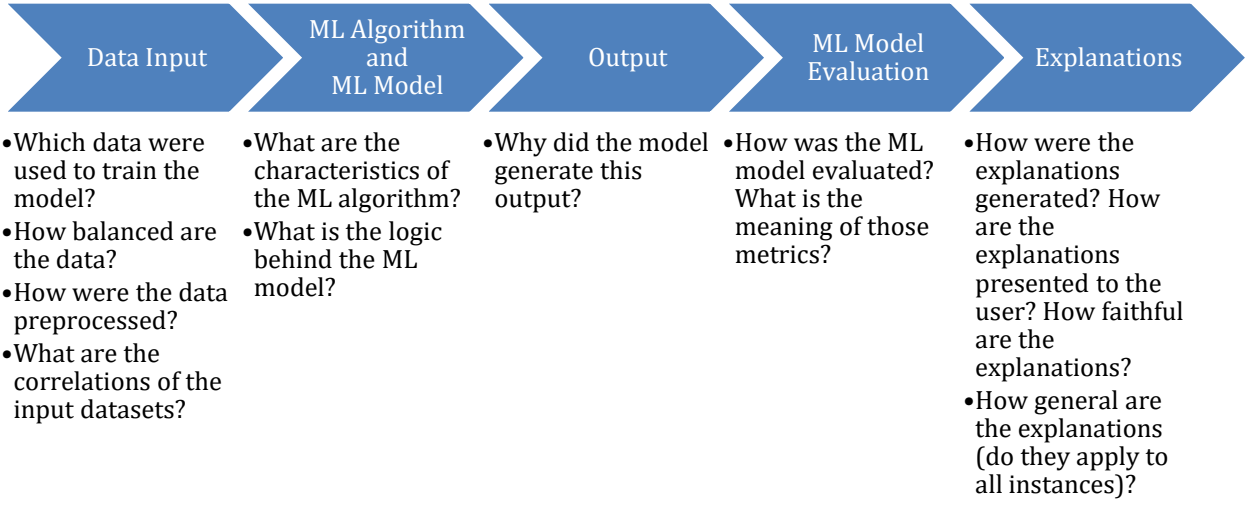

Figure 8. Competency questions related to components of ML process and explanation process to be addressed by the ontology

The non-functional requirements are related to characteristics, qualities, and general aspects not related to the content [10]. They can be divided into (i) ontology quality attributes, which refer to characteristics that an ontology should have as a software artifact, such as usability; (ii) project requirements derived from the ontology project, e.g., implementation requirements; (iii) requirements related to the intended uses of the ontology, such as interoperability. Considering these categories, we define the non-functional requirements of our ontology as follows:

(i) Ontology quality attributes:

REQ1. Guarantee usability to data scientists and developers, who want to understand the adequacy of the ML model and improve product efficiency, research, and new functionalities,

helping understanding the whole ML process and explanation process, possibly identifying where to make adaptations in the process;

REQ2. Guarantee extensibility by defining a more generic ML Ontology that represents ML processes that tackle different problems besides classification and can be further adapted or specialized.

(ii) Project requirements:

REQ3. Implementation in Protégé represented in OWL.

(iii) Intended uses-related requirements:

REQ4. Guarantee interoperability with already existing ontologies by grounding them into a foundational ontology.

In order to comply with non-function requirement REQ2 and considering the complexity of the ontology, we identify that the ontology can be structured into three modules: (1) a general module that represents general ML process independently of the task or the learning type performed; (2) a specific module for supervised classification; (3) an explanation module, which represents the post-hoc explanation process. Splitting the ontology into smaller parts allows the problems to be tackled one at a time [10].

3.3. Knowledge Acquisition and Reuse

Knowledge Acquisition and Reuse are auxiliary processes proposed by SABiO that assist ontology development. Usually, Knowledge Acquisition occurs in the initial stages of ontology development to gather knowledge from different sources, while Reuse can be adopted in many opportunities during the ontology development to reuse already established conceptualizations.

This project applies Reuse in the Knowledge Acquisition process by selecting already existing domain and foundational ontologies. We first define in Section 3.3.1 the ML and explanation processes with their components and what should be described. By identifying the main vocabulary necessary to represent the ML process, we select in Section 3.3.2 an existing domain ontology as the main reference to be reused and extended. Since SABiO proposes that the domain ontology should be analyzed in the light of a foundational ontology, the foundational ontology we have used is defined in Section 3.3.3.

3.3.1. ML Process and Explanation Process

In order to identify all components from the ML process and explanation process that can or should be described in an unambiguous way to complement the explanations generated by the post-hoc method, we define the vocabulary and represent the main components of both processes in Figure 9, and afterwards, we determine the objective of each description.

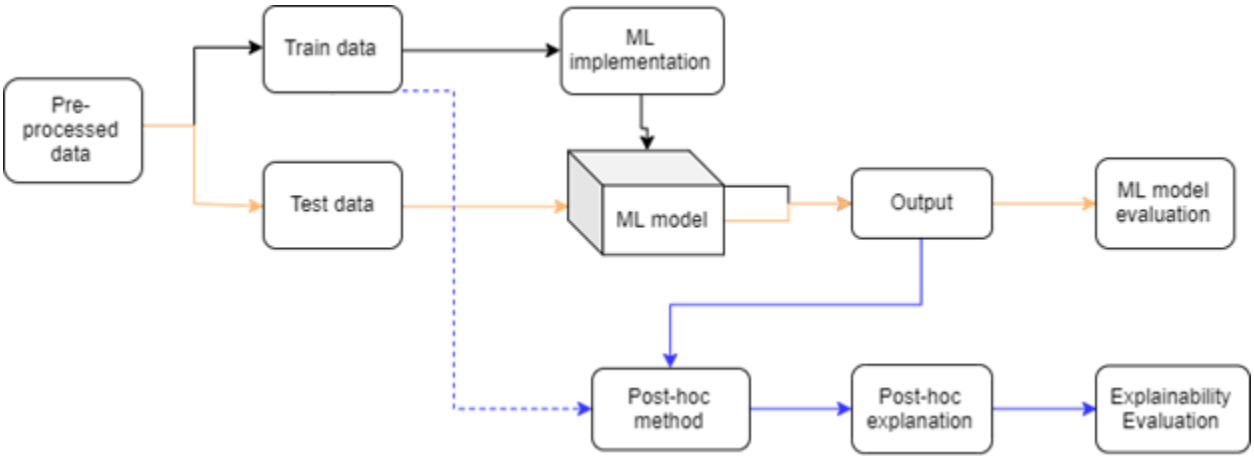

Figure 9. ML process and post-hoc explanation process

The ML process can be split into training and testing phases, represented in Figure 9 by black and orange arrows, respectively. The post-hoc explanation process is represented by blue arrows. The post-hoc method usually receives the output data from the ML model and some of the methods also use the input training data to generate the explanation (represented by a blue dashed arrow).

3.3.1.1. The ML Process

The ML process consists of preprocessing data, training phase, and testing phase.

The **preprocessed data** indicate the preprocessing steps, such as the cleaning process, the feature extraction or dimensionality reduction methods applied with respective parameters, and the split criteria between training and testing data. This information is important for the user to keep track of all process steps and methods applied that transform the raw initial data, which can affect the results. Given the different natures and particularities of the available datasets, which need diverse preprocessing steps to make it adequate for ML, we assume that the data are already preprocessed and only the preprocessing steps taken are modeled in the ontology without more details.

The training phase contains the training data, the ML implementation, the ML model, and the output. The **train data** represent the input data used to train the ML model and they can be described by identifying the input variables, or features, the mathematical correlations observed between the variables, and imbalances present in the data, which can introduce bias to the ML results.

The **ML Implementation** indicates the type and characteristics of the implemented algorithm, for example, if the model is a Support Vector Machine (SVM) or a Neural Network (NN), and the parameters used to train the model. The description of this step is related to the algorithm transparency, i.e., the understanding of how the algorithm works, but not for the specific model that is learned in the end, nor for how individual predictions are made, requiring only knowledge of the algorithm and not of the data or the learned model. For example, in the case of convolutional

neural networks used to image classification, we can describe that the algorithm learns edge detectors and filters on the lowest layers [21].

The **ML model (learned model)** has two types of desired descriptions. The first is the description of the inner workings, related to the “interpretable” concept, which is determined by the type of ML algorithm and enables the user to study and understand how inputs are mathematically mapped to outputs (Doran et. al., 2017 as cited by [25]). The second is the description of the explanation about the logic of reasoning behind the decision-making process in general [25], which enables the user to understand the work logic in ML algorithms and the patterns observed by the ML model in the data, being related to the “explainable” concept. For example, in the explanation method that extracts rules, the logic of reasoning is represented by the set of rules. Although many researchers often use both terms synonymously [25], in this project we distinguish them and focus on the second type of explanation desired from ML models, that is, “explainability”. However, the opacity of black-boxes hinders the direct generation of these explanations, making necessary the application of an XAI method that is able to generate explanations from them.

The description of the **outputs** obtained from the ML model is related to the explanation of a decision, which refers to the need for reasons that justify why a particular outcome was generated by the ML model. This explanation is particularly needed when unexpected decisions occur, ensuring also that there is an auditable and provable way to defend algorithmic decisions as being fair and ethical, leading to trust. Since the explanations of the results are also negatively affected by the opacity of the black-box, they also rely on XAI methods. Depending on the post-hoc method, the output explanation might depend on the logic of reasoning behind the ML model, that is, the ML model explanation. For example, in case the explanation of the ML model is represented by a set of rules, the output explanation should indicate which rule applies to this instance in order to explain the obtained result. In contrast, in the case of ILP (Inductive Logical Programming) post-hoc method, it explains by presenting semantic similarity (considering only “is a” relations between two terms) or relatedness (considering any relation between two terms) [35] with other resulting instances, creating a hypothesis that covers maximum positive and minimum negative examples based on a background knowledge source, not necessarily having an explanation for the whole ML model, since the explanations rely on instances. Another example of a post-hoc method that generates only explanations of the output without the ML model explanation is the DL-Learner tool [36], which generates explanations for image recognition of only one result by presenting semantic relatedness of the recognized components of the image with the assigned class.

The test phase comprises the descriptions of the testing data and the ML model evaluation. The **ML model evaluation** description is necessary because many metrics can be used to evaluate the ML models, such as accuracy, AUC, precision, recall. Choosing the best metrics to assess the ML model depends on the task that the ML model is performing and its application. Descriptions of the ML model evaluation could provide information about the metric and how the ML model performs in this metric.

3.3.1.2. The Explanation Process

The explanation process comprises the explanation method, the explanation generated by the method, and the explainability evaluation. This project focuses only on explanation processes that adopt post-hoc explanation methods, which do not require any change in the ML process.

The description of the **post-hoc method** indicates the post-hoc method adopted and its characteristics, for instance, the scope of the method (local or global explanations), the format of the explanation it generates (tree, rules, decision table, images, text highlight, natural language, etc.), if the explanation is iterative or static.

The **post-hoc explanation** is generated by the post-hoc method and explains the logic of reasoning behind the decision-making process, the patterns observed in the data, or provides means to justify the ML output. Depending on the method chosen, it is presented to the user in different formats and contains different information. For example, the method LIME [3] for image classification, highlights the most decisive pixels for the classification. The same method for tabular data indicates the weight of the impact that the input variable has on the classification. For rule extraction methods, it generates rules that explain the logic behind the decision of the ML model.

The **explainability evaluation** contains information about the assessment of the explanations, which can be evaluated in terms of effectiveness and user experience, for example, as subjective assessments such as A/B testing and surveys; quantitative metrics, for instance, the number of instances that the rules cover; and how faithful are the explanations to the underlying black-box.

3.3.2. Domain-Specific Ontology

One good practice to create an ontology is to take existing ontologies as starting points, assess their suitability for the proposed domain, and consider reusing them [10] [37]. This helps the knowledge acquisition process, speeds up the ontology development and guarantees interoperability to already existing applications that use the ontology [38]. Different domain-specific ontologies in the domain of ML are available. For example, MLOnto (Machine Learning Ontology) [39] defines an ontology to represent the knowledge around the ML discipline and DMOP (Data Mining OPtimization Ontology) [40] was developed to support decision-making in the data mining process, which is similar to an ML experiment.

Another well-known ontology for the ML domain is ML-Schema (MLS) [9], which provides a set of classes, properties, and restrictions to represent ML algorithms, with inputs, outputs, main steps, dependencies, their implementations, and executions. It aims to stimulate the development of standards, achieving interoperability and reproducible research, in order to cope and align with already existing ontologies that did not fully cover and support the needs of the ML area, such as Exposè [41], DMOP [40], and OntoDM [42]. It also reflects the data model of OpenML [43], which is an ML platform, being used to export all ML datasets, tasks, workflows, and runs as linked open data in RDF, allowing reuse and sharing of ML experiments by scientists.

MLS preserves the provenance of data and model, that is, metadata about their origin, derivation, or history. In ML workflows, it is useful to represent which data were used to train the ML model, where the data came from, and how they were preprocessed. Therefore, MLS allows us to track the creation, editing, publication, and future reuse of data. Since this ontology already is based on other well-known ontologies and is composed of many of the terms necessary to represent the ML process defined in Section 3.3.1.1, it is selected as the main reference to be reused and extended.

Figure 10 depicts the ML Schema core. Boxes represent classes of the schema, and the colors represent a taxonomy with three main classes: blue classes are processes, green are information entities, and yellow are qualities. Arrows without filled heads represent properties, arrows with empty heads represent subclass relations, and arrows with diamonds represent part-of relations [9].

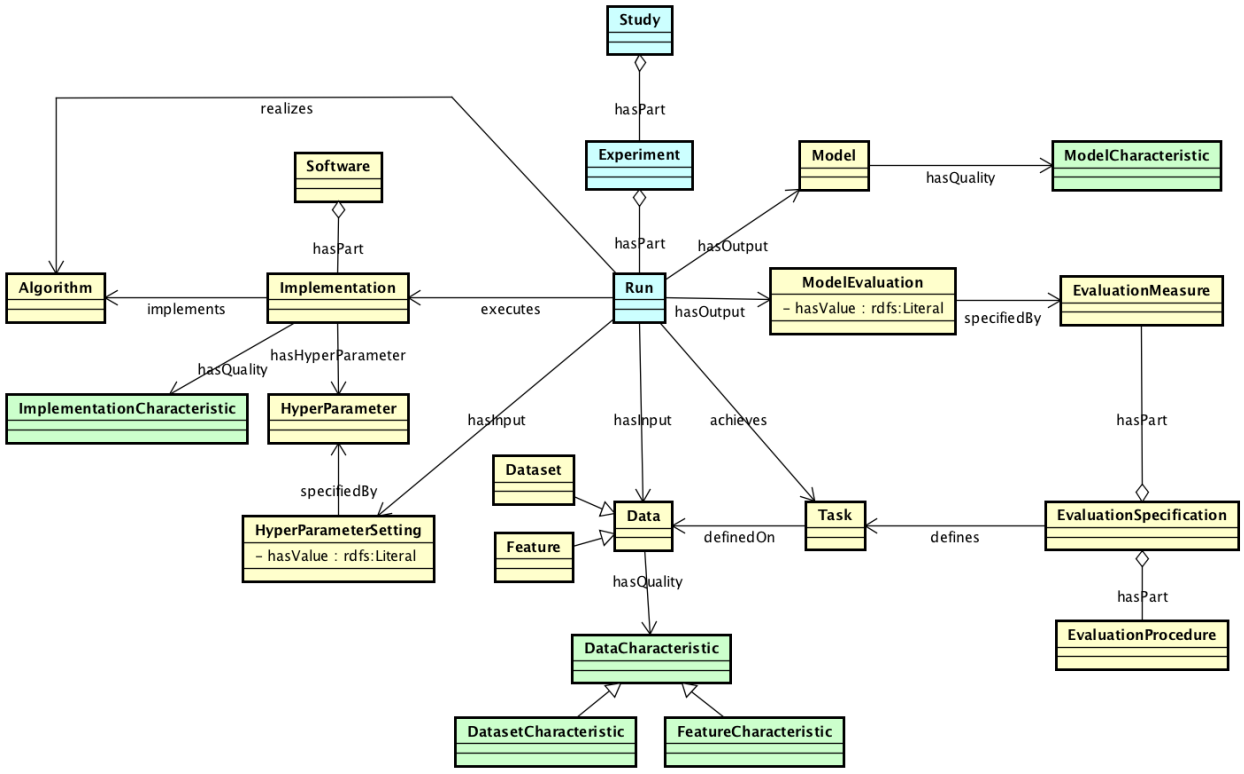

Figure 10. ML-Schema core configuration [9]

3.3.3. The Unified Foundational Ontology (UFO)

ML-Schema provides a standard to represent ML experiments and was conceived aiming to increase interoperability by preventing the proliferation of incompatible ML ontologies. However, it is not based on a foundational ontology, which usually improves the overall quality of the ontology by using principled design decisions and should facilitate interoperability with ontologies aligned to the same foundational ontology [44].

Foundational ontologies define the basic concepts upon which any domain-specific ontology is built [45]. They provide reusable information of high-level categorization about what will be represented in the ontology, modeling primitives for building ontologies in specific domains, such as processes and physical objects, relations, and attributes [44] [46]. By explicitly modeling the 'upper-level ontology', the top-level domain-independent ontological categories can be reused in domain-specific ontologies, guaranteeing semantic interoperability between them [45].

The Unified Foundational Ontology (UFO) [45] is based on two foundational ontologies, the GFO/GOL and OntoClean/DOLCE, with the goal to offer a general foundational ontology to applications in conceptual modeling. The General Formal Ontology (GFO) is based on the General Ontological Language (GOL) developed by the OntoMed research group, while OntoClean is based on the Descriptive Ontology for Linguistic and Cognitive Engineering (DOLCE) developed by the ISTC-CNR-LOA research group [45].

UFO makes a fundamental distinction between enduring and perduring individuals. Endurants, also known as continuants, are observed as concepts that endure in time and may change qualitatively while keeping their identity, for example, a person, a house, a car. Perdurants, or events, are individuals that happen in time, for instance, a business meeting, a soccer match, or an earthquake. Considering this distinction, UFO is divided into three sets: UFO-A, which is the UFO core, defines the things, sets, entities, individuals, and types. UFO-B defines the terms related to perdurants, such as events and states. UFO-C defines terms related to beliefs, desires, intentions, social roles, and linguistic things, extending UFO-B with concepts such as action, activity, and communication [45] [47].

UFO also defines a taxonomy of types to provide additional information about classes, and reflects the taxonomy of individuals, with abstract, relationship, endurant, event, and situation types. The taxonomy of endurant types is more detailed, qualifying how an endurant type applies to their instances. It considers if they apply necessarily to all its instances (rigid), or only part of the instances (anti-rigid), and if they carry a uniform principle of identity for their instances (sortal), or to individuals of different kinds (non-sortal). Based on this classification, type assumes the form of a category (non-sortal and rigid), kind (sortal and rigid), subkind (sortal, rigid, specializing the kind), phase, role (sortal and anti-rigid), among others [48].

OntoUML is a language for ontology-driven conceptual modeling based on UFO [49]. It is built as an extension of UML (Unified Modeling Language), which enables building conceptual models as fragments of UML class diagrams that are well-founded in UFO. In this work, we adopt OntoUML to develop conceptual models following SABiO's methodology suggestion, benefiting from the advantage of developing conceptual models and reference ontology already well-founded in UFO.

OntoUML offers tools, for example, the OntoUML plugin, which enables the development of conceptual models in a well-known platform for UML (Visual Studio), and facilitates the process of obtaining a well-founded operational ontology coded in OWL from the conceptual model by performing transformations. The plugin offers an automated transformation that maps OntoUML classes, associations, and attributes to OWL classes, object properties, and data properties, respectively. It also considers in the transformation the generalization sets, disjoint properties, and the model cardinalities [50]. The resulting operational ontology is well-founded in gUFO,

which is a lightweight implementation of UFO that contains a subset of UFO-A and minimal support for UFO-B, designed with the focus on guaranteeing computational properties [48].

4. Ontology Development

After the ontology specification, we perform ontology development. This phase consists of capturing and formalizing the ontology with its modules and metadata, as shown in Sections 4.1 to 4.5. Then, designing and implementing the ontology is carried out by a transformation of the conceptual model to the operational ontology, described in Section 4.6.

The ontology capture and formalization step starts with defining the conceptual model of the ontology, whose process in this project is represented in Figure 11. In this process, we reuse and extend an existing domain-specific ontology (MLS), which is grounded in the foundational ontology (UFO), using the OntoUML language. Then, we define the generic ML module that can represent ML processes that tackle different problems besides classification and can be further adapted or specialized. This module aligns the grounded ontology with the ML process defined in Section 3.3.1.1. Subsequently, we propose a specific ML module to represent the ML classification process with detailed operations, inputs, and outputs. Thereafter, the specific ML module is extended with an explanation module. Finally, the ontology is complemented with metadata to describe its components. After developing the conceptual model, SABiO recommends the generation of a dictionary of terms, which can be found in detail in Appendix A. Each step taken to develop the conceptual model is described below.

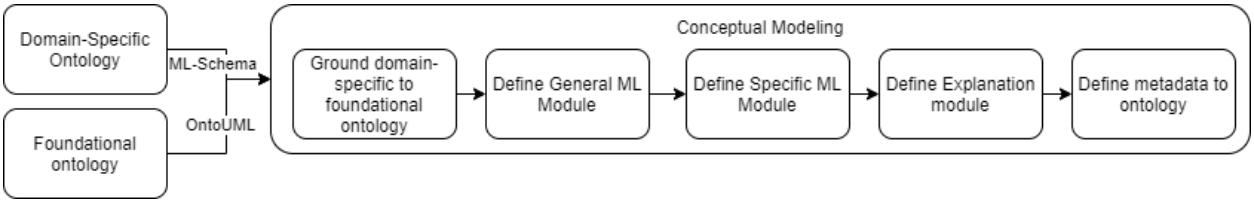

Figure 11. Process to develop the conceptual model of the proposed ontology

4.1. Grounding Domain-Specific Ontology in a Foundational Ontology

In this project, we ground the ML-Schema in gUFO [48], which is a lightweight implementation of UFO. The use of gUFO instead of UFO occurs for a technological reason because the tool that makes the transformation from the conceptual model to the operational ontology (OntoUML plugin) supports only gUFO, with the benefit of being simpler and retaining computational properties for the resulting OWL ontology [48]. However, gUFO imposed some limitations especially because of the lack of support to UFO-C, which could better represent *Actions, Goals,* and *Intentions*.

Grounding a domain ontology into a foundational requires them to be aligned, which leads to concerns such as how to overcome differences in expressiveness that can exist between the ontologies and how to accommodate for the different philosophies behind them [40].

In order to ground the MLS to gUFO, we first develop a conceptual model of MLS using OntoUML [49]. The conceptual model is developed by creating a class diagram using Visual Paradigm 16.3 and OntoUML plugin [51], which enable the use of OntoUML stereotypes in class diagrams to perform verification and transformation. By creating the class diagram and assigning OntoUML's class and relationship stereotypes to it, the OntoUML plugin allows a model transformation of the conceptual model into OWL with the support of gUFO.

Since OntoUML is a UML profile to represent ontologies grounded in UFO, there is a matching between the class and relationship stereotypes of OntoUML to UFO's structure. They both distinguish types from individuals, however, OntoUML supports only modeling types [49]. Thus, if we create a class in a class diagram and assign an OntoUML class stereotype to it, when the model transformation of the conceptual model to OWL occurs, it generates a gUFO ontology with corresponding classes in the taxonomy of types and the taxonomy of individuals. The transformation also converts the OntoUML relations to gUFO object properties in the ontology.

Therefore, to create the conceptual model in OntoUML, we selected concepts from MLS that are related to the ML process defined in Section 3.3.1.1, and by studying and analyzing the OntoUML stereotypes together with its matching in gUFO structure, we defined a mapping between them. The MLS qualities, processes, and information entities are grounded by creating classes in the class diagram and assigning the OntoUML class stereotypes to them according to the chosen components of the gUFO taxonomy of types. The MLS relations are analyzed and considering the types they connect, adequate relationships in OntoUML are selected, which are then transformed into adequate object properties in gUFO. Table 2 presents the correspondence of MLS components to gUFO and OntoUML, and Figure 12 shows the conceptual model of MLS using OntoUML.

Table 2. Correspondence of ML-SCHEMA components to gUFO and OntoUML

ML-SCHEMA Element	OntoUML stereotype	gUFO Type	gUFO Individual	Reason
Processes: • Study • Experiment • Run	Event	Event Type	Class Event	Run is a concrete individual that happens in time, Experiments are composed of runs, and Study is a collection of experiments. These components are represented as Events in gUFO but could be better represented by UFO-C.
Qualities: • DataCharacteristic • ImplementationCharacteristic • ModelCharacteristic	Quality	Kind	Class Quality	Intrinsic aspects that are measurable by some value spaces and may be used to compare individuals, e.g., number of features. For the taxonomy of types, they are kinds of existentially dependent aspects of objects.
Information Entities:	Collective	Kind	Class	A variable collection is a

• Algorithm • EvaluationProcedure			VariableCollection	complex object whose parts have a uniform structure and whose membership can vary. Algorithms and EvaluationProcedures are sets of instructions.
Information Entities: • EvaluationSpecification • Implementation • Software • HyperParameterSetting • HyperParameter • EvaluationMeasure • Data • Model • ModelEvaluation	Kind	Kind	Class FunctionalComplex	Rigid and sortal classes.
Properties: • hasInputs (domain: Run, range: Data) • executes (domain: Run, range: Implementation) • achieves (domain: Run, range: Task) • realizes (domain: Run, range: Algorithm)	Relation participation	Formal relation	Object property participatedIn (domain:Object, range:Event)	Identifies the event in which the object participated.
Property: • hasOutput (domain:Run, range: Model or ModelEvaluation)	Relation Creation	Formal relation	Object property wasCreatedIn (domain:Endurant, range:Event)	Identifies the event that brought the endurant into existence.
Properties: • hasHyperParameter (domain: Implementation, range: HyperParameter) • specifiedBy (domain: HyperParameterSetting, range: HyperParameter) • specifiedBy (domain: ModelEvaluation, range: EvaluationMeasure) • defines (domain: EvaluationSpecification, range: Task) • definedOn (domain:Task, range: Data) • implements (domain: Implementation, range: Algorithm)	Relator and relations mediation	Kind	Class Relator and sub-properties from mediates (domain: Relator, range: Endurant)	Extrinsic aspects that connect two or more concrete individuals.
Properties: • hasPart (domain:Study, range:Experiment) • hasPart (domain:Experiment, range:Run)	relation Participational	Formal relation	Object Property isEventProperPartOf (domain:Event, range:Event)	One event is part of another event.
Properties: • hasPart (domain:Software, range:Implementation) • hasPart(domain:EvaluationSpecification, range:EvaluationMeasure) • hasPart	Relation ComponentOf	Formal relation	Object property isComponentOf (domain:Object, range:FunctionalComplex)	Objects are components of Functional Complex classes.

(domain:EvaluationSpecification, range:EvaluationProcedure)				
Property: • hasQuality (domain:Model, range:ModelCharateristic)	Relation Characterization	Formal relation	Object property inheredIn (domain:Aspect, range: ConcreteIndividual)	Object property that represents the relation of inherence between Object and Quality.

The process *Run* of MLS is defined as an *Event* in OntoUML and gUFO because it consists of a process that happens in time. An *Experiment* is a collection of runs with no change in membership, since specific runs are part of an experiment, therefore, an *Experiment* is grounded also as an *Event*, with the relationship between run and experiment defined as *isEventProperPartOf* in gUFO and as *Participational* stereotype in OntoUML. A *Study* is a collection of *Experiments*, defined also as an *Event*. These *Events* could be better represented by components of UFO-C, such as *Actions* and *Intentions*, however, UFO-C is not supported by gUFO.

MLS qualities such as *DataCharacteristic*, *ImplementationCharacteristic*, and *ModelCharacteristic* are intrinsic aspects that are measurable by some value spaces and may be used to compare individuals, for example, the number of features a dataset has, or a characteristic of an Implementation that differentiates it from the others, such as the library that provides it, being defined as *Kind* in gUFO's taxonomy of types and *Qualities* in gUFO's taxonomy of individuals and OntoUML.

MLS information entities *Algorithm* and *EvaluationProcedure* are defined as *collective* in OntoUML, *kind* in gUFO taxonomy of types and *variable collection* in gUFO taxonomy of individuals. Collectives are complex objects whose parts have a uniform structure, in this case, *Algorithms* and *EvaluationProcedures* are sets of instructions. They are defined as *variable collections* in the taxonomy of individuals because the change in memberships is possible without creating a different collection, differently from *fixed collections* whose membership cannot vary. Although *Implementations* are composed of a set of implemented instructions, they also have parameters and other components, being defined as *FunctionalComplex*.

The remaining information entities are defined as *kind* in OntoUML and taxonomy of types because they are rigid and sortal and existentially-dependent aspects of objects. The kind stereotype is automatically transformed to *FunctionalComplex* in the taxonomy of individuals.

MLS properties are grounded as follows: *hasInputs* (domain: *Run*, range: *Data*), *executes* (domain: *Run*, range: *Implementation*), *achieves* (domain: *Run*, range: *Task*), and *realizes* (domain: *Run*, range: *Algorithm*) are defined as OntoUML relationship *participation* and as gUFO object property *participatedIn* (domain: *Object*, range: *Event*) because they identify the event in which the object participated. When creating these properties in OntoUML we also define the cardinalities of these relationships, which can be seen in Figure 12, for example, the participation of one or more *Data* in one *Run*. These cardinalities are reflected later in the operational ontology in gUFO.

The property *hasOutput* (domain: *Run*, range: *Model* or *ModelEvaluation*) is grounded in the object property *wasCreatedIn* (domain: *Endurant*, range: *Event*) in gUFO and *creation* relationship in OntoUML because it identifies the event that brought the endurant into existence.

Properties *hasHyperParameter* (domain: *Implementation*, range: *HyperParameter*), *specifiedBy* (domain: *HyperParameterSetting*, range: *HyperParameter*), *specifiedBy* (domain: *ModelEvaluation*, range: *EvaluationMeasure*), *defines* (domain: *EvaluationSpecification*, range: *Task*), *definedOn* (domain: *Task*, range: *Data*), and *implements* (domain: *Implementation*, range: *Algorithm*) are extrinsic aspects that connect two or more concrete individuals, hence we define them as *Relators* in OntoUML and gUFO, and use the gUFO object property *mediates* (domain: *Relator*, range: *Endurant*) and OntoUML relationship *mediation* to relate the individuals.

Properties *hasPart* with domain *Study* and range *Experiment*, and with domain *Experiment* and range *Run* are properties between *Events*, hence they are defined as *participational* in OntoUML, formal relation in the gUFO taxonomy of types and object property *isEventProperPartOf* (domain: *Event*, range: *Event*) in the gUFO taxonomy of individuals. The other *hasPart* properties (with domain *Software*, range *Implementation*; domain *EvaluationSpecification,* range *EvaluationMeasure*; domain *EvaluationSpecification*, range *EvaluationProcedure)* are components of functional complex classes, therefore they are defined as the relation *ComponentOf* in OntoUML, as a formal relation in the gUFO taxonomy of types, and as object property *isComponentOf* (domain: *Object*, range: *FunctionalComplex*) in the gUFO taxonomy of individuals.

Finally, *hasQuality* (domain: *Model*, range: *ModelCharateristic*) is defined as the gUFO object property *inheredIn* (domain: *Aspect*, range: *ConcreteIndividual*) and as the OntoUML relationship *characterization,* because it represents the relation of inherence between *Object* and *Quality*.

MLS components *feature, dataset, featureCharacteristic* and *datasetCharacteristic* were not considered in the grounding process since they are components of *Data* and *DataCharacteristic*, respectively, and represent the different granularity levels of *Data*. In MLS, the *dataset* is disjoint with the *feature*. However, a *feature* is an independent variable of a dataset, hence *features* should be components of the *dataset*.

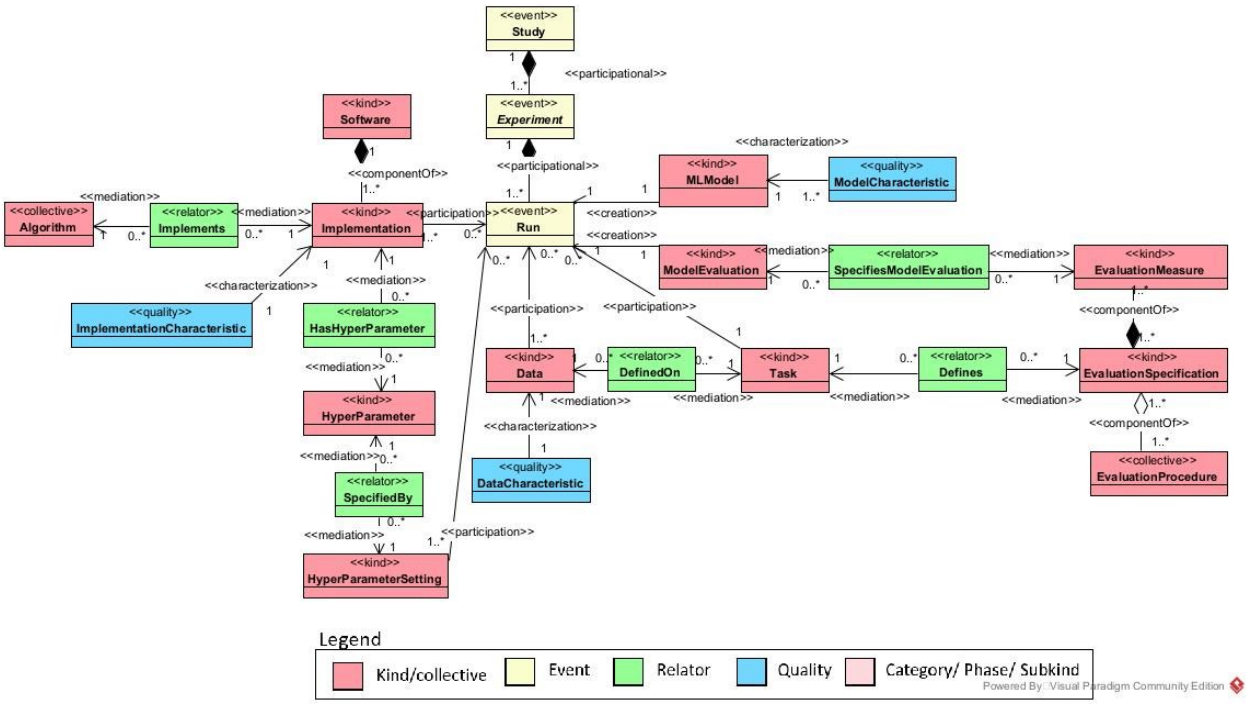

Figure 12. Conceptual Model of MLS in OntoUML using Visual Paradigm

After obtaining the conceptual model of Figure 12, we use the OntoUML plugin to transform the conceptual model to OWL, which can be manipulated in Protégé. Figure 13 depicts the MLS and the resulting MLS grounded in gUFO in Protégé.

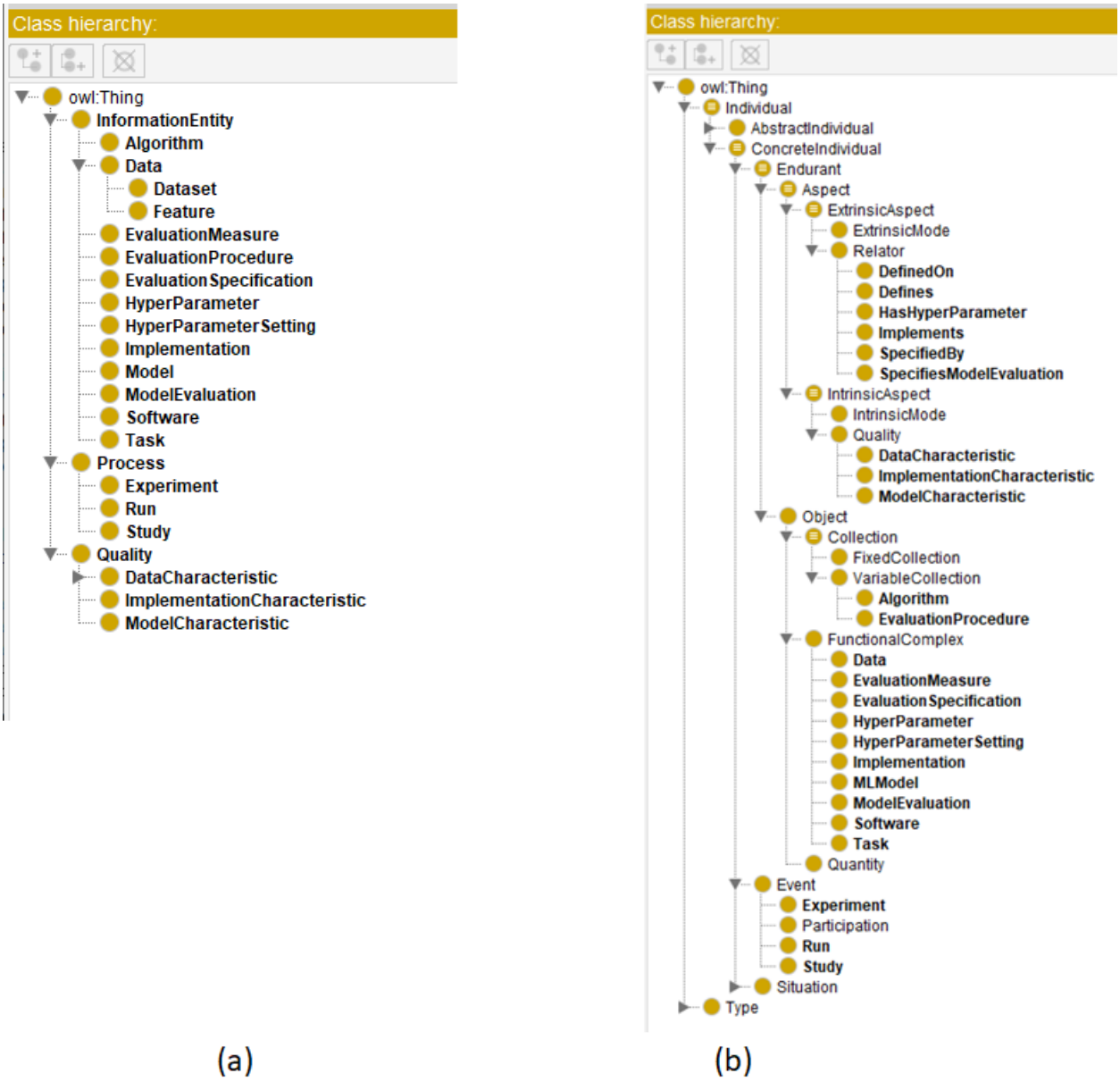

Figure 13. (a) ML-Schema in Protégé 5 (b) ML-Schema grounded in gUFO in Protégé 5

4.2. General ML Module

The general ML module of our ontology is developed by aligning the MLS grounded in gUFO with the ML process defined in Section 3.3.1.1. Then, we define the axioms and constraints of the module, followed by developing the conceptual model.

First, we verify how the MLS grounded in gUFO is aligned with the ML process. On the one hand, we can verify that some elements are aligned, such as *Implementation*, *ML model, ML Evaluation*. On the other hand, MLS focuses on the data generated in ML workflows, reflecting the OpenML structure, but does not represent the nature of ML processes, resulting in semantic gaps. In the

sequel, we discuss how to overcome these limitations and obtain an appropriate alignment of MLS with the ML process.

The first characteristic that can be identified in MLS is that a *Study* should have its *Purpose*. Similarly, an *Experiment* should define its main *Goal*, for example, classification. The *Goal* is addressed by the *MLAlgorithm,* e.g., Support Vector Machine (SVM) and Artificial Neural Network (NN) address classification tasks.

In MLS, the property *defines* between *EvaluationSpecification* and *Task* gives the notion that the *EvaluationSpecification*, containing the *EvaluationMeasures* and *EvaluationProcedure,* defines the *Task* when, in reality, we pose that the *Goal* of the *Experiment* defines the *EvaluationMeasures* with the adequate measurements to assess the *MLModel,* but it does not necessarily define the procedure to perform the evaluation. For example, for diagnosing a disease, sensitivity must be high, but there is no specification of which procedure we should use, if cross-validation or simple split training and testing data. We pose that the *EvaluationProcedure* can be considered as one type of *Algorithm* that is implemented and then executed by an *Operation*.

In MLS, we missed some concepts to organize the *Runs* into *Experiments*, arranging these *Runs* in a sequence to be executed. For this reason, our ontology incorporates some concepts of the Data Mining OPtimization Ontology (DMOP) [40], for example, experiment, workflow, operation, and their relations, but with some adaptations. In DMOP, a data mining experiment is a complex event composed of a series of operations and it executes a workflow that organizes the implementations sequentially. However, in Section 3.3.1 we stated that our experiments are composed of at least two different workflows, one related to the ML process and the other to the explanation process. Thus, to distinguish the run of different workflows, we introduce the event Workflow Execution (*WFExecution)*. The *WFExecutions* are processes that belong to an *Experiment,* are composed of a series of *Operations* and execute *Workflows*, which organize sequentially the implementations that are executed by these *Operations*.

Each *Operation* executes an *Implementation* and can be seen in MLS as *Run.* A *Task* in MLS is a piece of work that needs to be addressed in the ML process, which is already incorporated into each *Operation* in our ontology. Depending on the operation being performed, for example preprocessing, training, testing, and evaluating, different *Implementations* are executed. *Implementations* can have different parameters, which is the reason why we generalize the MLS *HyperParameter* to *Parameter*, the MLS *HyperParameterSetting* to *ParameterSetting,* and the relation *hasHyperParameter* to the relator *hasParameter*. Each *ParameterSetting* has only one *ParameterSettingCharacteristic*, which receives the value to set the parameter.

The lack of cardinality in MLS can generate misconceptions, for instance, indicating that the *Run* generates both the evaluation and the model. However, a dedicated *Run* usually outputs an ML model, and another *Run* dedicated to evaluating the model outputs the evaluation of the ML model in terms of the metrics. Other types of *Runs* generate other outcomes, such as preprocessed data or predictions. Hence, we included the component *Output* to generalize the output generated by the *Operations*, which can then be specialized into ML model, evaluation, etc.

Moreover, by considering *Feature* and *Dataset* as disjoint types of *Data* in MLS, the features cannot be considered as components of the dataset. However, a *feature* is an independent variable of a dataset, so we can model *Data* and consider the *Features* as components of *Data*.

The object property *mediates* and *participatedIn* are specified to determine exactly the domain and range for the relators. *Mediates* is specialized for each relator, for example, the relator *implements* uses the sub-property *implementsInvolvesAlgorithm* that mediates the relator to an *Algorithm*, and the sub-property *implementsInvolvesImplementation* mediates the relator to the *Implementation*. The object property *participatedIn* is also specialized into *executedBy* that connects the *Operation* to the *Implementation*.

These adaptations impact the taxonomy of types in UFO, where the relators are defined as *kinds*, and the *Output* is a *category* because it applies to more than one kind, considering the different kinds represented by the subclasses of *Output*, such as *MLModel*, *Result*, *ModelEvaluation*.

After properly aligning the MLS with the ML process and defining the concepts and relations of the generic ML module, we define the axioms and constraints necessary for this ontology module. We define restrictions concerning the cardinality of the properties between the concepts, disjoint classes, and the sequence that each operation needs to be executed. For example, we define that one *WFExecution* has at least one *Operation*, which defines the cardinality of the property *executedBy* between *WFExecution* and *Operation*. Each *Output,* such as *ModelEvaluation* and *MLModel*, is disjoint with each other, meaning that a *ModelEvaluation* cannot be at the same time an *MLModel*. To restrain the order that the operations can be executed, we specify that the *Preprocess* operation needs to occur before the *Train* operation. After specifying the axioms, we formalize them by writing in ontology language OWL2, which is a new version of OWL allowing new expressivity. The detailed axioms and their formalizations can be found in Appendix B.

The conceptual model of the general ML module is depicted in Figure 14. It extends and adapts the MLS grounded in UFO of Section 4.1, and it is also developed using OntoUML. The OntoUML diagrams express typed relations between components, cardinality constraints for the relations, and constraints related to which element can be connected to others, formalizing the above described specifications and axioms. In this diagram, we represent within the grey UML package the general ML module that could be reused to represent other ML processes and further adapted or specialized.

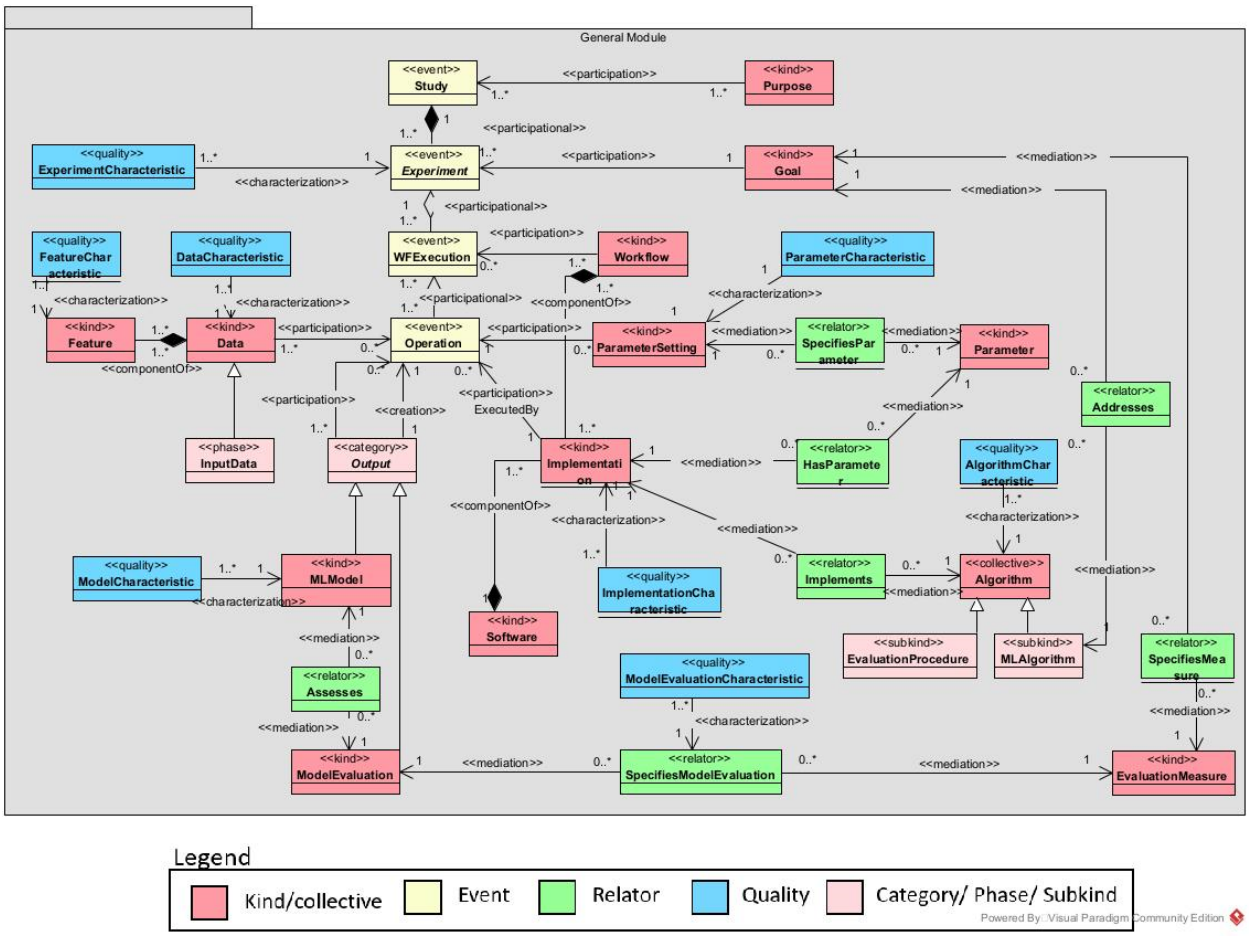

Figure 14. Conceptual model of the general ML module using OntoUML

4.3. Specific ML Module

The MLS adapted to UFO can be further specialized considering the different operations that are performed in the ML classification process, taking into account their different participants. This more detailed ontology is useful to model the characteristics of each operation and component, specifying the inputs, outputs, implementations, and algorithms. The conceptual model of the specific ML module is represented by the components within the yellow UML package of Figure 15.

The tasks involved in the classification process usually consist of preprocessing the data, training the ML model, testing the ML model, and finally evaluating the ML model. Each task is represented as a subclass of the *Operation* class in the specific ML module, with the participants and the artifacts that are involved in the operation.

First, the *Preprocess* operation takes place to make the input data suitable to train and test the ML model. This operation includes steps such as cleaning, transformation, feature extraction, dimensionality reduction, normalization, splitting the data into train and test sets. Given different natures and particularities of inputs that require diverse preprocessing steps to make them

adequate for ML, we model the preprocessing steps in the ontology, allowing the user to keep track of steps and methods applied that changed the initial data, but without further details such as changes in characteristics of the data that impact the results and the learned ML model. *Preprocess* receives as input the *InputData*, and executes the *PreprocessImplementation*, which in turn *implements* the *PreprocessingAlgorithm.* The output of this operation is the *PreprocessedData*, which can be specialized into the subclasses *TrainData* and *TestData*.

The distinction between *TrainData* and *TestData* is not represented in MLS nor the general ML ontology, however, they usually represent different parts of the data and can be used by different operations. *TrainData* participates in the *Train* operation, which fits the ML model, generating the fitted *MLModel*. For this, the *Train* operation executes the *MLImplementation* that implements the *MLAlgorithm*. The *MLAlgorithm* can be, for example, an SVM or a NN, and they can be implemented in the scikit-learn library in Python [52].

The *Test (Predict)* receives the *TestData* and executes the *PredictImplementation*. This implementation calls the fitted *MLModel* to predict the output *Results*. The prediction for each instance is represented by *ResultInstance*.

The *EvaluationModel* operation for labeled data receives as input the *Results* and compares them with the *TestData*. It executes *EvaluateModelImplementation* that implements the *EvaluationProcedure*, such as cross-validation or leave-one-out, taking into account the *EvaluationMeasures* that need to be evaluated. This operation generates the *ModelEvaluation* that contains the values for the measurements specified by *EvaluationMeasure*.

Besides specifying the operations and their participants, in order to align the ontology to the components of the ML process defined in Section 3.3.1.1, we add the *Correlates* relator that indicates the correlations between *Features.* Each *Correlates* relator has exactly one *CorrelationCharacteristic*, which receives the correlation value that defines how correlated the features are.

The new classes are instantiated in the UFO taxonomy of types by considering *Data* as a kind, and its subclasses *inputData* and *preprocessedData* as phases, since they represent different phases of the data, being sortal and anti-rigid. The *preprocessedData* is further divided into *TrainData* and *TestData*, which can be seen as roles, being also sortal and anti-rigid and assuming roles for different operations.

The above specifications are formalized as axioms applicable to this module, which can be found in detail in Appendix B.

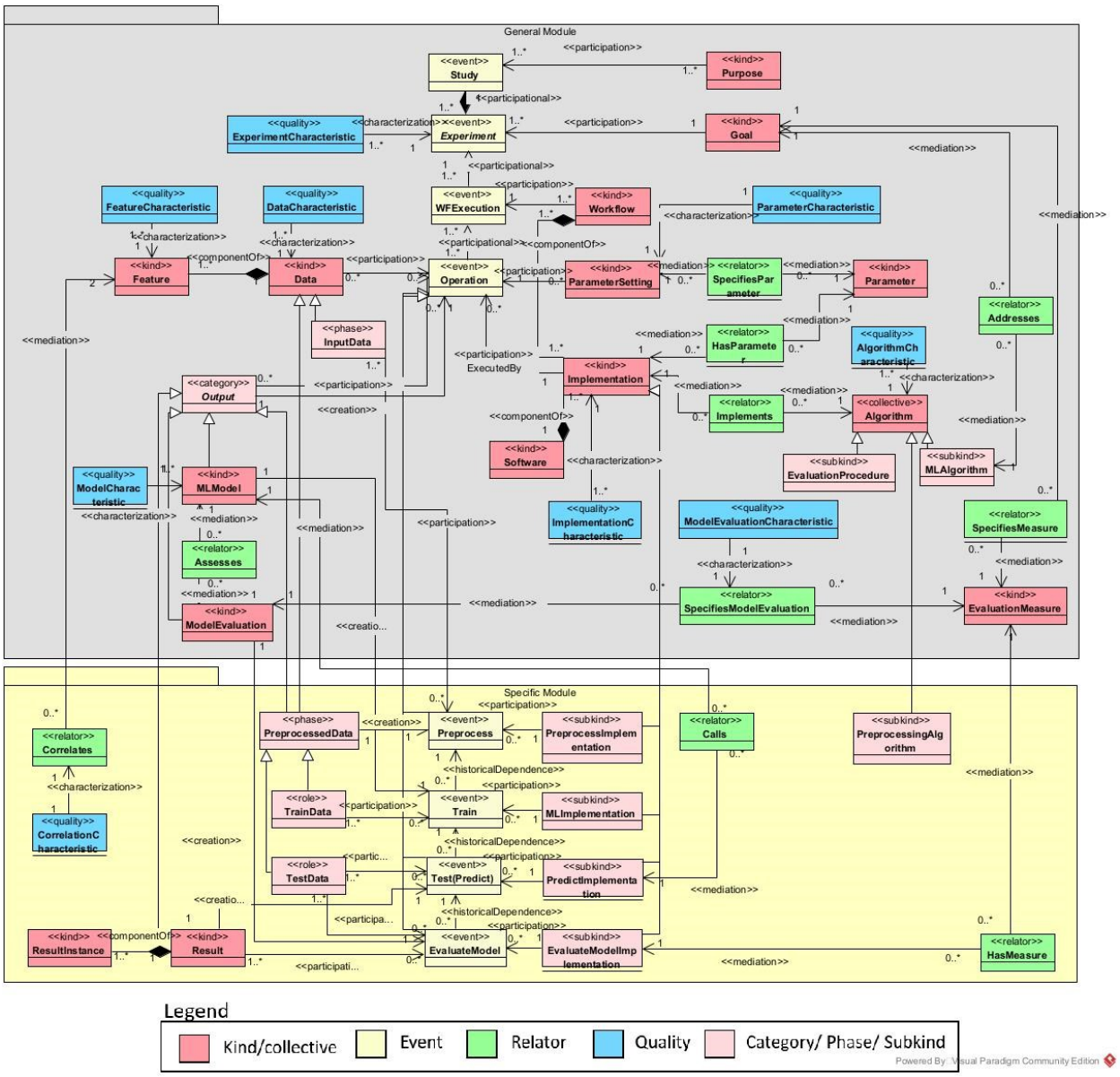

Figure 15. Conceptual model of the general ML module (grey area) and specific ML module (yellow area) using OntoUML

4.4. Explanation Module

We extend the ontology containing the generic and specific module by adding the explanation module, which models the explanation process defined in Section 3.3.1.2. The explanation module represents the post-hoc explanation process by adding the *Explain* and *EvaluateExplanation* operations with their corresponding participants, as represented by the green region in Figure 16.

The *Explain* operation aims to generate explanations by using the *Results* generated by the *Test (Predict)* operation. In some cases, it also uses the *PreprocessedData* to create a hypothesis that

maps the input to the output in order to explain the behavior of the ML model. It executes the *ExplainImplementation*, that is, the implementation of the *ExplainableAlgorithm* in a specific programming language. The *ExplainableAlgorithm* can be, for instance, the LIME explanation method or the rule set generator RIPPER. The output of this operation is the *Explanation*, which can be classified as *MLExplanation* if it aims to explain the ML model, or as *ResultExplanation*, if it explains only a resulting instance. The relator *ExplainsModel* between the *MLModel* and the *Explanation* allows the logic behind an ML model to be obtained after the post-hoc method is applied.

The *Explanation* has *ExplanationComponents*, as the *Explanation* rule set generated by RIPPER has rules and LIME has explanations for each instance. Independently of the type of the *Explanation*, that is, if it explains the *MLModel* or the *Result*, *ExplanationComponents* can explain *ResultInstances*. Since the explanation for the result is obtained after the post-hoc explanation method is applied, we relate the resulting instance and the explanation component using the relator *ExplainsResultInstance*.

The *EvaluateExplanation* operation contains information about the assessment of the explanations. It receives the *Explanation* generated by the *Explain* operation and executes the *EvaluateExplanationImplementation*. As a result, it generates the *ExplanationEvaluation*. The *ExplanationEvaluation* can evaluate the whole explanation, such as faithfulness, but can also evaluate specific *ExplanationComponents*. This happens in the case of coverage, which measures the number of instances covered by an *ExplanationComponent*. For example, one rule from the rule set generated by RIPPER covers many instances, while one explanation generated by LIME covers only one instance. In order to identify the correspondence between the coverage value generated by the *EvaluateExplanation* operation and the specific *ExplanationComponent*, we introduce the relator *EvaluatesExplanationComponent*.

The explanation module introduces the differentiation of *Explanation* into *ModelExplanation* and *ResultExplanation*, which characterize roles in the gUFO taxonomy of types. Following the SABiO guidance, we formalize these specifications and restrictions as axioms in OWL2, which can be found in detail in Appendix B.

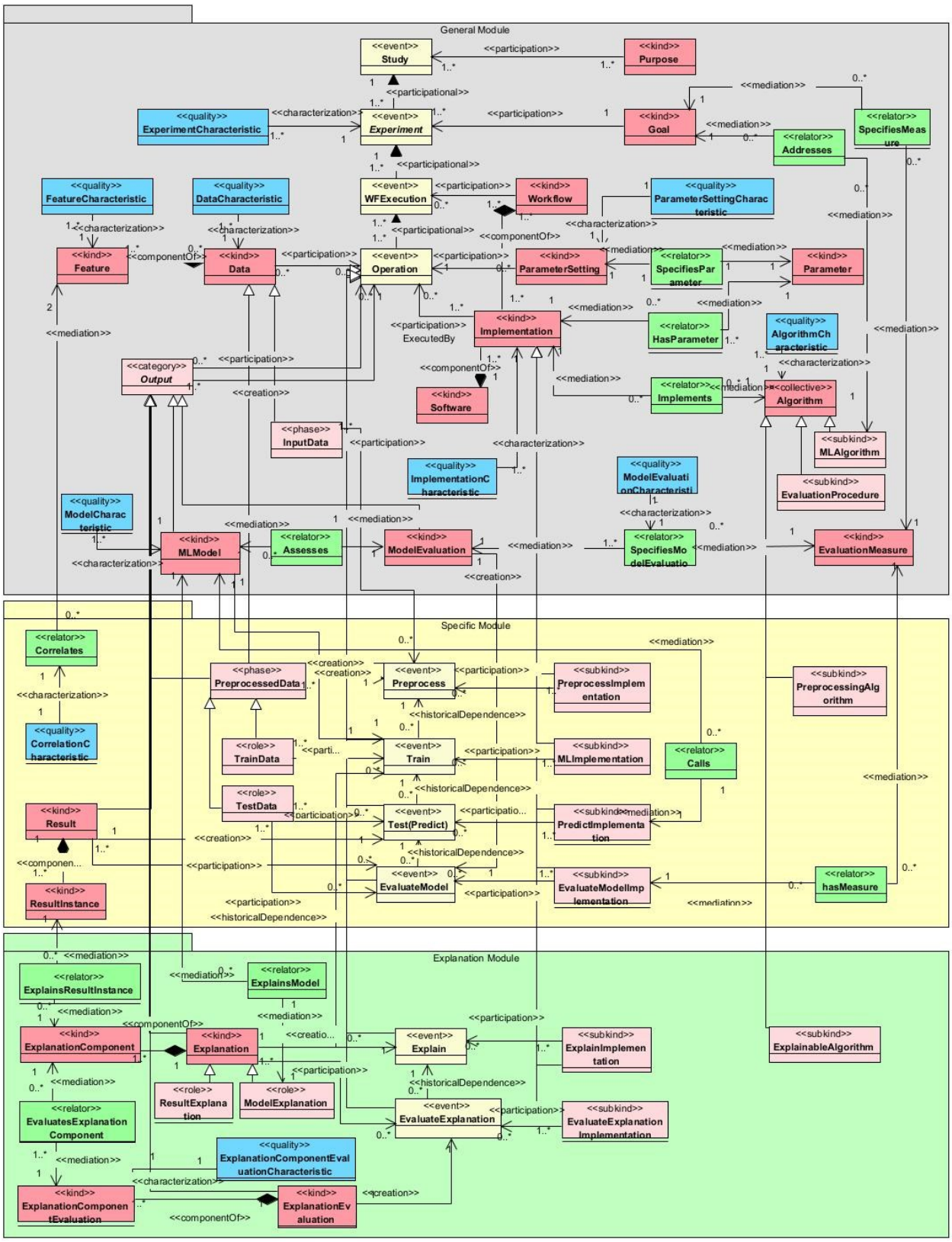
General Module
<<event>> Study
<<participation>>
<<kind>> Purpose
<<participational>>
<<quality>> ExperimentCharacteristic
<<characterization>>
<<event>> Experiment
<<participation>>
<<kind>> Goal
<<mediation>>
<<relator>> Addresses
<<relator>> SpecifiesMeasure
<<quality>> FeatureCharacteristic
<<quality>> DataCharacteristic
<<event>> WFExecution
<<kind>> Workflow
<<quality>> ParameterSettingCharacteristic
<<componentOf>>
<<kind>> Feature
<<kind>> Data
<<event>> Operation
<<kind>> ParameterSetting
<<relator>> SpecifiesParameter
<<kind>> Parameter
<<mediation>>
<<participation>> ExecutedBy
<<kind>> Implementation
<<relator>> HasParameter
<<quality>> AlgorithmCharacteristic
<<category>> Output
<<creation>>
<<kind>> Software
<<relator>> Implements
<<collective>> Algorithm
<<phase>> InputData
<<subkind>> MLAlgorithm
<<quality>> ImplementationCharacteristic
<<quality>> ModelEvaluationCharacteristic
<<subkind>> EvaluationProcedure
<<quality>> ModelCharacteristic
<<kind>> MLModel
<<relator>> Assesses
<<kind>> ModelEvaluation
<<relator>> SpecifiesModelEvaluation
<<kind>> EvaluationMeasure
Specific Module
<<relator>> Correlates
<<phase>> PreprocessedData
<<event>> Preprocess
<<subkind>> PreprocessImplementation
<<subkind>> PreprocessingAlgorithm
<<historicalDependence>>
<<quality>> CorrelationCharacteristic
<<role>> TrainData
<<event>> Train
<<subkind>> MLImplementation
<<relator>> Calls
<<role>> TestData
<<event>> Test(Predict)
<<subkind>> PredictImplementation
<<kind>> Result
<<event>> EvaluateModel
<<subkind>> EvaluateModelImplementation
<<relator>> hasMeasure
<<kind>> ResultInstance
Explanation Module
<<relator>> ExplainsResultInstance
<<relator>> ExplainsModel
<<kind>> ExplanationComponent
<<kind>> Explanation
<<event>> Explain
<<subkind>> ExplainImplementation
<<subkind>> ExplainableAlgorithm
<<role>> ResultExplanation
<<role>> ModelExplanation
<<relator>> EvaluatesExplanationComponent
<<event>> EvaluateExplanation
<<subkind>> EvaluateExplanationImplementation
<<quality>> ExplanationComponentEvaluationCharacteristic
<<kind>> ExplanationComponentEvaluation
<<kind>> ExplanationEvaluation
Legend
Kind/collective
Event
Relator
Quality
Category/ Phase/ Subkind
Powered By: Visual Paradigm Community Edition

Figure 16. Conceptual Model of the ML Explanation Ontology that is composed of the general ML module (grey region), the specific classification module (yellow region), and the explanation module that represents the post-hoc explanation process (green region).

4.5. Metadata

The metadata used to describe the ML process and explanation process are represented as qualities (blue boxes) in the diagram of Figure 16, being the semantic information that comprises qualities, reified quality values, and annotations in the ontology. In gUFO, qualities are intrinsic aspects that are measurable by receiving a literal, while reified quality values are abstract individuals that can use pre-defined data to provide the value of the quality, instead of literals. The reified quality values are instantiated in the gUFO taxonomy of types as abstract individual types and the qualities as kinds and subkinds. The characteristics of each concrete individual related to the ML process and explanation process are described below.

4.5.1. Metadata for the ML Process

The specific characteristics of concrete individuals related to the ML process that are included in the ontology are detailed in Table 3 and described as follows.

To define the information related to the creator and the creation date of the object we use the annotation properties *creator* and *date*, respectively. The duration time or execution time of *Experiments, WFExecutions,* and *Operations* are defined using the data properties *hasBeginPointInXSDDateTimeStamp* and *hasEndPointInXSDDateTimeStamp*, since they are properties related to events in gUFO.

The *Experiment* is enriched with a description as a comment value. The field of the experiment is a quality that receives, for example, the value “Healthcare”. The type of experiment is defined as a subclass of gUFO Quality, named *ExperimentType*. It receives a nominal value, for example, “ML experiment”.

The *Goal* specifies the required measurements for the experiment with the quality *SuitableMeasurement*.

MLAlgorithms have *MachineLearningTechnique* as a characteristic, which defines the type of technique that the algorithm uses, for example, classification or regression. Similarly, *PreprocessingAlgorithms* have *PreprocessingTechnique* to indicate if the algorithm performs dimensionality reduction, feature extraction, or split data, among others. Other nominal values used by *MLAlgorithms* are *LearningType*, which indicates the kind of learning performed by the ML algorithm (supervised or unsupervised learning); and *DataType*, which indicates the kinds of data they require. *MLAlgorithms* also have the *Transparency* intrinsic quality, which receives information about how the algorithm usually works.

Data has a description, the number of features, the number of instances, and the data source, with information related to the access date and the source. Each *Feature* present in the data has the description of its meaning and the reified quality value about the type of data, indicating, for example, if it is text, image, continuous, discrete, tabular, or categorical data. They also have

qualities related to the number of categories they represent in case of categorical data, the number of null values, and the number of instances for each category represented, which can help identify imbalances. *Features* might have correlations between them, which is represented by the relator *Correlates* and the quality *CorrelationCharacteristic*, which indicates the strength of the correlation.

Implementations receive optionally the library that implements the algorithm, for example, scikit-learn for Python [52]. They also have parameters set when running the operation, receiving the setting value through the quality *ParameterValue*.

The *MLModel* generated by the *Train* operation receives the *Interpretability* quality value, with nominal values “Inscrutable” or “Transparent”.

The measures to evaluate the ML model may receive descriptions that help clarify their meaning for the users. They also specify the *ModelEvaluation* that is generated by the *EvaluateModel* operation, having a relator that mediates them. The values obtained for the measure are expressed as a quality of this relator.

Table 3. Metadata of the ontology related to the ML process

Concrete Individual	**Metadata**	**Metadata Description**	**Characteristic Type**
All	Creator	Creator of the element	Annotation Creator
	Date	Date created	Annotation Date
Experiment, WFExecution, Operation	Date	Duration time or execution time of the individual	Dataproperties hasBeginPointInXSDDateTimeStamp and hasEndPointInXSDDateTimeStamp
Experiment	Experiment Description	Description of the experiment	Annotation comment
	Experiment Domain	Domain of the experiment, e.g. "Healthcare"	Quality
	Experiment Type	Type of the experiment, e.g. "ML experiment"	Quality
Goal	Suitable Measurement	Suitable measures for the experiment, e.g., "accuracy" and "sensitivity"	Quality
ML Algorithm	ML Technique	Type of technique used by ML Algorithm, e.g., classification or regression	Reified quality value
	Learning Type	Kind of learning performed by the ML Algorithm, e.g., supervised, unsupervised	Reified quality value

	Data Type	Type of data required by the ML Algorithm, e.g., categorical, continuous, discrete, image, tabular, text	Reified quality value
	Transparency	Description about how the algorithm usually works	Quality
Preprocessing Algorithm	Preprocessing Technique	Type of technique used by Preprocessing operation, e.g., dimensionality reduction, feature extraction, split data, etc.	Reified quality value
Data	Data Description	Description of the data	Annotation comment
	Number of Features	Number of features present in the data	Quality
	Number of Instances	Number of instances of the data	Quality
	Access Data	Access date to the data source	Quality
	Source	Source location, e.g., website link	Quality
	Source Name	Name of the source	Quality
Feature	Feature Description	Description of the meaning of the feature	Annotation comment
	Data Type	The data type of the feature, e.g., continuous, discrete, image, tabular, text	Reified quality value
	Number of categories	Number of categories present in the feature	Quality
	Number of Nulls	Number of null values present in the feature	Quality
	Category Quantity	Number of instances for each category represented by the feature	Quality
Correlates	Correlation Characteristic	The intensity of the correlation between features	Quality
Implementation	Implementation Library	Library that implements the algorithm, e.g., "scikit-learn"	Quality
Parameter	Parameter Value	Parameter value set when running the operation	Quality
ML Model	ML Interpretability	Interpretability of the ML model, e.g., inscrutable, transparent	Reified quality value
Evaluation Measure	Evaluation Measure Description	Description of the measure	Annotation comment

Specifies Model Evaluation	Measure Value	The value obtained for the measure that evaluates the ML model	Quality

4.5.2. Metadata for the Explanation Process

The components of the Explanation Module have the following characteristics:

- The explainable algorithm is characterized by the quality value *ExplanationScope*, which indicates if the algorithm approach is local or global to the number of instances they apply;
- The *ExplanationFormat* indicates the type of explanation it generates, for example, a decision tree, rules, etc.;
- *ExplanationInteraction* expresses whether the explanation is static or interactive;
- *ExplainedElement* points out if the ML model or the output result is explained;
- *ExplanationFidelity* indicates how faithful the explanation is to the underlying ML model, indicating, for example, local fidelity or unfaithful;
- *ExplainableMethodFlexibility* identifies if the post-hoc method is model-specific or model-agnostic.

The explanations have components with *ExplanationComponentCharacteristic* qualities, which receive the values obtained in the *Explain* operation, for example, each rule of the rule set generated using a rule extractor method.

The explanations are evaluated by the *EvaluateExplanation* operation, which has qualities such as *Fidelity* or *Coverage*. *Fidelity* is a quality that inheres in the whole *Explanation*. *Coverage* indicates the number of instances that are covered by the *ExplanationComponent* and applies only to global explanation methods because local explanation methods apply only to one instance. Since the coverage refers to one *ExplanationComponent*, a relator is defined between the *ExplanationComponentEvaluation* and the *ExplanationComponent*, with a quality that characterizes the relationship.

Table 4 shows the specific characteristics of concrete individuals related to the explanation process we included in the ontology.

Table 4. Metadata added to the ontology related to the Explanation Process

Concrete Individual	**Metadata**	**Metadata Description**	**Characteristic Type**
Explainable Algorithm	Explanation Scope	Scope of the explainable algorithm, e.g., local or global	Reified quality value
	Explanation Format	Type of explanation generated by the algorithm, e.g., rules, decision tree, image highlight, input variable weight, natural language	Reified quality value

	Explanation Interaction	Indicates if the explanation generated by the algorithm is static or interactive	Reified quality value
	Explained Element	The element that is explained by the explanation, i.e., if it explains the logic behind the ML model or the result	Reified quality value
	Explanation Fidelity	Indicates how faithful the explanation is to the underlying ML model with values, e.g., local fidelity or unfaithful	Reified quality value
	Explanation method Flexibility	Identifies if the post-hoc method is model-specific or model-agnostic	Reified quality value
Explanation Component	Explanation Component Characteristic	Receives explanation component values, e.g., each rule from the rule set	Quality
Evaluates Explanation Component	Coverage	Number of instances covered by the Explanation Component	Quality
Explanation Evaluation	Fidelity	How faithful is the explanation to the underlying ML model. It can receive an intensity value or indicate "local fidelity"	Quality
Explanation Evaluation	Accuracy	Accuracy of the post-hoc model in predicting the correct class for the instances	Quality

4.6. Ontology Design and Implementation

The ontology design and implementation steps have the objective of generating the operational version of the ontology. In the design step, we define the technical aspects of the ontology and the implementation environment. Since the OntoUML plugin is based on UFO and it already supports transforming the conceptual model to the implementation language, the gap between the conceptual models to the operational is shortened, so that fewer decisions are necessary for the design step. We adopt OWL as the implementation language of the ontology, considering that the plugin provides the model transformation into OWL supported by gUFO. In addition, the transformation requires the absence of verification errors among the exported entities, thus, the selection of all elements to be transformed should not introduce new violations to constraints.

The implementation step consists of implementing the ontology in the operational language. This is performed by executing a transformation of the OntoUML conceptual model by exporting it to OWL in Turtle (Terse RDF Triple Language) format, which represents the ontology in the RDF data model. We then open the exported file using Protégé and make further adjustments.

One of the limitations encountered using OntoUML is that the transformation to the operational ontology did not transform named relationships into sub-properties, being only possible to use

native relationships of OntoUML and their correspondents in gUFO. Therefore, we implemented manually the sub-properties in Protégé, creating *executedBy* as a sub-property of *participatedIn* and specializations of *mediates* with their specific domain and ranges, such as *implementsInvolvesImplementation* and *implementsInvolvesAlgorithm*, which have as domain the relator *Implements* and as range *Implementation* and *Algorithm*, respectively.

OntoUML allows the definition of detailed metadata, however, we chose to keep the conceptual model simpler to focus on the concepts of the ML process and explanation process and represent the semantic information only as generic characteristics, which can already be used to characterize the objects. Therefore, we include the metadata described in Section 4.5 directly in Protégé and associate them with the objects with the object properties *inheresIn* for qualities and with *hasReifiedQualityValue* for quality values.

5. Case Study

In this chapter, we present the application scenario that we used to conduct experiments and validate our ontology. The scenario adopted is the COVID-19 (Coronavirus disease), which is a highly contagious respiratory virus disease first reported in China in December 2019 that has spread rapidly around, being declared by the World Health Organization (WHO) in January 2020 as a pandemic with an international concern of public health emergency (Sohrabi C. et al., 2020 as cited by [53]). The academic and medical communities have been searching for new technology to identify infections, find the best clinical trials, and control the spread of the virus.

In this scenario, many studies have been conducted using ML and AI to learn about the disease and identify infected patients, with promising technologies for decision-support that enable better scale, processing power, and speed, outperforming humans in specific operations (Davenport T. et al., 2019 as cited by [53]). These studies have shown that there are many characteristics in clinical data that can influence the probability of infection, such as reported symptoms, risk factors, age, and gender. There are also many different ML classification models to predict the probability that a patient has the infection [54] or other related purposes, such as predicting mortality or the need for ventilation [55]. However, the comparison of these models usually relies only on the accuracy, not being clear about the logic of the model and the characteristics the model considers as more relevant to generate the results.

We adopt the COVID-19 scenario as a case study to illustrate and validate our ontology. We use supervised learning algorithms that perform classification operations to predict mortality among infected patients and existing explanation methods to generate explanations that intend to make the steps and the logic of the algorithm clear to data scientists to improve the model. Based on the results, instances are created to populate the ontology, validating it and refining it if necessary. The case study was developed in Python.

5.1. Data Description

In this project, we used an epidemiology dataset of people tested for COVID-19 in Mexico. These data are publicly available and reported daily by the Mexican government [56]. The dataset was also applied in [54] [55], but we used the same dataset as Muhammad et al. [54] available in [57], which was already preprocessed and translated to English. This is in accordance with the assumption that the ontology requires the data to be already preprocessed, and only the preprocessing steps are modeled in the ontology without specific details (Section 3.3.1.1).

The dataset has 566.602 instances with 23 features, containing demographic data such as age and gender of the patient, pre-existing conditions, for instance, diabetes, chronic obstructive pulmonary disease (COPD), asthma, immunosuppression, hypertension, obesity, pregnancy, chronic renal failure, other prior diseases, and whether the patient used tobacco. It indicates if the patient was hospitalized, had pneumonia, needed a ventilator, was treated in an intensive care unit (ICU), but also the result of the Reverse Transcription Polymerase Chain Reaction (RT-PCR) test, and the date when the patient deceased, if applicable. Figure 17 represents a fragment of the dataset.

Information of the dataset, such as its description, the source, the date when the source was accessed and the data period are included as metadata to the ontology. We also include all the features of the dataset, with the description of what they represent, their data type, their categories in case they are categorical features, the encoded label that the number of the category represents (for example, pneumonia with value 1 indicates that the patient had pneumonia, and value 2 indicates that he/she did not have), the number of instances for each category in the feature, and the number of null values.

	id	sex	patient_type	entry_date	date_symptoms	date_died	intubed	pneumonia	age	pregnancy	...	inmsupr	hypertension	other_disease
0	16169f	2	1	04/05/2020	02/05/2020	9999-99-99	97	2	27	97	...	2	2	2
1	1009bf	2	1	19/03/2020	17/03/2020	9999-99-99	97	2	24	97	...	2	2	2
2	167386	1	2	06/04/2020	01/04/2020	9999-99-99	2	2	54	2	...	2	2	2
3	0b5948	2	2	17/04/2020	10/04/2020	9999-99-99	2	1	30	97	...	2	2	2
4	0d01b5	1	2	13/04/2020	13/04/2020	22/04/2020	2	2	60	2	...	2	1	2
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
566597	01ff60	2	1	13/05/2020	03/05/2020	9999-99-99	97	2	58	97	...	2	2	2
566598	047cd1	1	1	07/04/2020	06/04/2020	9999-99-99	97	2	48	2	...	2	2	2
566599	1beb81	1	2	14/05/2020	01/05/2020	9999-99-99	2	1	49	2	...	2	2	2
566600	16fb02	1	1	31/05/2020	29/05/2020	9999-99-99	97	1	43	2	...	2	2	2
566601	0021c9	2	1	16/05/2020	16/05/2020	9999-99-99	97	1	65	97	...	2	1	2

566602 rows × 23 columns

Figure 17. Fragment of the COVID-19 dataset [57]

5.2. Experiments

Considering the application scenario and the available data, we define two machine learning experiments in the healthcare field. The goal of the first experiment is to classify mortality of confirmed cases using an ML model on the epidemiology dataset of tested people for COVID-19 in Mexico and to understand the ML model using the RIPPER rule extractor method. The second experiment is similar to the first, but it uses the LIME method to generate explanations, instead of RIPPER.

Each experiment involves two main workflows: the ML Workflow that consists of preprocessing the data, training, and testing the classifier, and the Explanation Workflow that comprises the generation and evaluation of explanations. The second experiment reuses the ML Workflow of the first experiment. The components of the ML Workflow are detailed in Section 5.3 and the Explanation Workflow in Section 5.4.

In order to describe the experiment using the ontology, we include instances containing the experiments and their characteristics, the experiment's goals, the workflows involved, and metrics to evaluate the ML models. The goal of classifying mortality is a binary classification, requiring the ML model to be evaluated using metrics such as accuracy, sensitivity, and specificity.

5.3. ML Workflow

The ML Workflow represents the workflow for the ML process and contains the steps of preprocessing the input dataset, training, and testing the ML model. In the ontology, the ML Workflow is executed by the ML Workflow execution and it contains sequences of implementations, which are each executed by the operations. These operations are components of the ML Workflow execution. The details of each step are described in the following subsections.

5.3.1. Data Preprocessing

The first step of the ML Workflow consists of preprocessing the dataset. From the dataset available we used a small sample of patients that entered the hospital between the first two days of June 2020 and that were diagnosed with positive results for COVID-19. Patients with unknown information were manually removed, keeping only the ones that indicated the presence or absence of conditions. The date of death was converted to a binary column indicating 1 if the patient was deceased and 0 if she/he recovered.

After importing the dataset into a Python environment, we removed the columns id (patient's identifier), the date that the patient started feeling symptoms, if the patient had contact with other COVID-19 cases, the result of the RT-PCR test (since they were already manually filtered), and pregnancy. The column 'inmsupr' was renamed to 'immunosuppression'. The final dataset remained with 9.451 cases and was split into 70% for training and 30% for testing. No feature extraction or dimensionality reduction methods were adopted.

The preprocessing steps performed in Python and the parameters used to the preprocessing functions are included in the ontology by adding instances of the implementation that were executed by the preprocessing operation. Manual steps are disregarded. Differences between the original dataset and the preprocessed can be identified, for example, as the difference between the number of instances of each dataset and, after deleting some columns from the original data, the presence of some features as components of the input dataset and their absence on the preprocessed datasets.

5.3.2. Data Analysis

We perform a dataset analysis in order to obtain a more detailed description of the datasets. For this, we chose to adopt an explainability tool available for ML that helps identify imbalances between features across classes and provides functions to identify correlations between features in the dataset, facilitating the data analysis. The application of functions from the explainability toolbox EthicalML-XAI [58] available in Python resulted in the identification of imbalances in the dataset (Figure 18a) and the identification of correlations between features (Figure 18b). This is an optional step, since there are many other ways to obtain information about feature correlation.

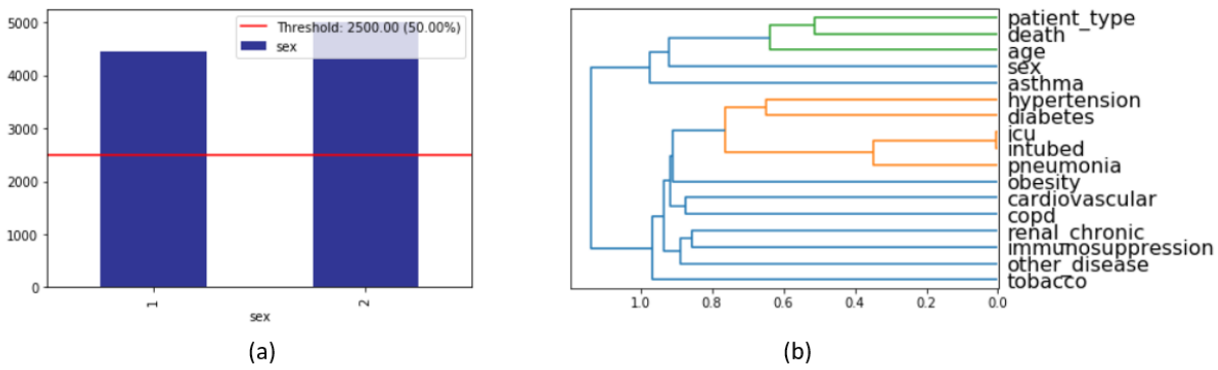

Figure 18. (a) Imbalances for the field gender in the dataset, where 1 defines female and 2 male (b) Highest correlations found in the training dataset

Figure 18 shows that mortality of COVID-19 cases is related to the type of patient (if they are hospitalized or attended in ambulatory) and their age. In addition, patients with hypertension often have diabetes. Patients that are intubated were usually in the ICU, and these cases are related to pneumonia. These views are useful to see correlations between variables and identify unexpected correlations, for example, if intubation were not related to ICU, it would be abnormal, leading the user to verify possible problems in the dataset.

The imbalances and correlations are included in the ontology as characteristics of features that belong to the training dataset. Imbalances are represented by including for each categorical feature their categories and the corresponding number of instances. For correlations, we include relators that connect two features, representing the correlation between them, and the values of the correlations are represented by qualities.

5.3.3. ML Model Training

The preprocessed training data is used to train a black-box model to classify the mortality of confirmed COVID-19 cases, similar to the study performed in [55]. We use SVM as the black-box algorithm, given its popularity in making classifications due to its ability to capture non-linearity [33], and its common application in COVID-19 detection in the literature, as found in [59] [60] [61] [62] [63].

The ML train operation is described in the ontology as the operation that receives the COVID-19 training dataset as input and generates the *SVMModel* as output. It executes the SVM implementation that implements the SVM algorithm provided by the scikit-learn library in Python [52], using a linear kernel type as a parameter. The ML algorithm is described in terms of the author of the algorithm; the algorithm transparency, defined by Molnar [21] as the description of how the algorithm usually works; the type of data adequate for this algorithm; the involved ML technique, in this case, classification; and the learning type, which in this example is supervised learning.

5.3.4. ML Model Evaluation

After the ML model is trained, we perform a step to evaluate it according to the metrics established by the goal. For this, we first classify the test set using the *Test (Predict)* operation that executes the *PredictImplementation*. This implementation calls the *SVMModel* function to generate the predictions. These predictions together with the original labels of the test set are then used as input for the *EvaluateModel*.

The *EvaluateModel* operation executes an implementation of the classification report provided by the scikit-learn library in Python [52], which already contains usual metrics for classification, including accuracy, specificity, and sensitivity. The SVM achieves an accuracy of 0.91 in the test set, specificity 0.92 (recall of the negative class), and sensitivity of 0.64 (recall of positive class), as shown in Figure 19. The metric values and the description of each metric are included in the ontology.

	precision	recall	f1-score	support
0	0.98	0.92	0.95	2707
1	0.28	0.64	0.39	129
micro avg	0.91	0.91	0.91	2836
macro avg	0.63	0.78	0.67	2836
weighted avg	0.95	0.91	0.93	2836

Figure 19. Metrics for the SVM in the test set. Class 0 represents the negative class with recovered patients and class 1 represents the positive class with deceased patients.

5.4. Explanation Workflow

The explanation workflow represents the explanation process, which consists of two main steps, namely the generation of explanations and the evaluation of these explanations. The objective of the first step is to generate explanations from the black-box model using post-hoc methods that identify the behavior of the ML model or try to explain results, indicating, for example, how the input variables impact the final results. The second step is to evaluate the generated explanations, which is necessary since post-hoc solutions generate hypotheses to explain the black-box model.

In the ontology, the explanation workflow (*ExplainWorkflow*) is executed by the explanation workflow execution (*ExplainWorkflow_exec)* and it contains sequences of implementations, which are each executed by the operations *Explain* and *EvaluateExplanation*. These operations are components of the explanation workflow execution.

In our proposal, the generation of explanations can be done by extracting rules with algorithms using the training dataset and the corresponding labels predicted by the fitted model, and by identifying the relevance of input variables to the classification. We explore the rule extraction method RIPPER in the first experiment. The impact of input variables on the corresponding class is analyzed using LIME in the second experiment. The representation of both methods is discussed in the sequel.

5.4.1. Rule Extraction with RIPPER

The SVM is a complex non-linear function that maps the input space into a higher dimensional feature space in which a hyperplane can separate two classes, in a way that the margin between the classes is maximized [33]. The opaqueness of SVM models can be remedied by extracting rules that mimic the black-box as closely as possible, since some insight is gained into the logical workings of the SVM by obtaining a set of rules that mimic the model's predictions [33]. Therefore, rule extraction was performed to understand the classifications of the SVM, opening up the black-box.

In order to extract rules from the SVM, we apply RIPPER (detailed in Section 2.5.1) in the dataset with labels predicted by the SVM. This algorithm extracts a rule set for the classification of the COVID-19 mortality class as depicted in Figure 20. The rule set is composed of disjunctive rules that lead to the classification of the positive class, in this case, mortality. Each line is a rule compound by a conjunction of clauses. We interpret the rule set as follows: if the first rule applies to the instance, for example, the patient has an age between 66 and 99 years (age=66-99), has pneumonia (pneumonia=1), has hypertension (hypertension=1), is hospitalized (hospitalized=2), and is a man (sex=2), then he will be classified with a high chance of mortality. If the patient does not fit in this first rule, we try the second, and so on, until the last rule. If none of the rules applies to the patient, he/she will be classified with the negative class, that is, recovery.

```
FINAL RULESET:
[[age=66-99 ^ pneumonia=1 ^ hypertension=1 ^ hospitalized=2 ^ sex=2] V
[pneumonia=1 ^ age=66-99 ^ icu=2 ^ intubed=1] V
[pneumonia=1 ^ age=66-99 ^ icu=2 ^ hypertension=1 ^ asthma=2 ^ copd=1] V
[pneumonia=1 ^ age=66-99 ^ icu=2 ^ hypertension=1] V
[pneumonia=1 ^ intubed=1 ^ icu=2] V
[pneumonia=1 ^ age=66-99 ^ icu=2 ^ sex=2] V
[renal_chronic=1 ^ pneumonia=1 ^ hospitalized=2 ^ sex=2] V
[pneumonia=1 ^ age=66-99 ^ intubed=1] V
[pneumonia=1 ^ age=66-99 ^ icu=2 ^ cardiovascular=1] V
[pneumonia=1 ^ age=66-99 ^ renal_chronic=1] V
[hospitalized=2 ^ renal_chronic=1 ^ other_disease=1] V
[pneumonia=1 ^ renal_chronic=1 ^ age=51-57] V
[icu=2 ^ age=66-99 ^ renal_chronic=1] V
[pneumonia=1 ^ hypertension=1 ^ age=57-66 ^ sex=2 ^ intubed=1] V
[age=57-66 ^ hypertension=1 ^ pneumonia=1 ^ diabetes=2 ^ sex=2 ^ tobacco=2] V
[intubed=1 ^ icu=2]]
```

Figure 20. The rule set extracted using RIPPER to classify mortality in COVID-19 cases

In the ontology, the RIPPER algorithm is described in terms of its author, the source reference paper, and further descriptions to indicate that it consists of a global model-agnostic method that explains the ML model and is unfaithful to the underlying model, because it extracts rules from the training examples, not directly from the ML model. The explanations generated by the algorithm are also described in terms of its format, i.e., the method generates static explanations in the format of rules. Its implementation in Python is provided by the Wittgenstein library [64].

The generated explanation is included in the ontology as an instance of *Explanation*, more specifically *ModelExplanation* because it explains the logic behind the whole ML model instead of explaining only one instance, and each of the rules instantiates *ExplanationComponents*. The

relator *ExplainsModel* indicates that the explanation explains the *SVMModel*, and the relator *ExplainsResultInstance* connects one prediction of the dataset to the rule that explains it.

5.4.2. LIME Explanations

The rules obtained from SVM with RIPPER could give the intuition behind the model but did not clearly indicate which input variables were more decisive to determine the classification. In order to generate a more complete view of the black-box model and identify the impact of each input variable on the classification, we use the post-hoc model-agnostic tool LIME[3] (detailed in Section 2.5.2). The implementation of this algorithm is easily accessible through the LIME package in the Python environment. In this work, SP-LIME is applied in the SVM model to generate the most representative explanations that indicate how the input variables contribute to each class, and we also use the LIME functions to generate explanations for single instances.

The application of SP-LIME generates explanations that show the positive and negative impact of the input variables for each class. Figure 21 shows the five most representative explanations for class 0 representing recovered and class 1 representing deceased. Because it is a binary classification, the positive impact of one variable on a class represents its negative impact on the other. The highest feature impacts are highlighted in red, if the impact for predicting the class specified in the first column is positive, and green, if the impact is negative.

In the same Figure, we can identify that intubation has the biggest impact on the classification. If the intubation has a value of 97, it means that the intubation is not applicable because the patient is not hospitalized, having a big chance of recovery. If it has value 2 (meaning that the patient is not intubated but he/she is hospitalized), or value 1 (patient is intubated), higher is the chance of death. ICU has also a high impact in classifying the patients. If the ICU value is 97, the patient is not hospitalized so the intensive care is not applicable, if the value is 2, the patient is hospitalized but not in intensive care, and if the value is 1, the patient needs intensive care. However, it is expected that the patient in ICU would have more impact on death than the patients not in ICU. Since SP-LIME gets the most representative instances, it is possible using the explanations to evaluate the impact of each variable in the classification and verify if the model behaves adequately as expected.

Classification	intubed=97	icu=97	pneumonia=2	patient_type=1	asthma=2	immunosuppression=2	age	hypertension=2	renal_chronic=2	sex=2	copd=2	cardiovascular=2	other_disease=2	diabetes=2	obesity=2	tobacco=2	intubed=2	icu=2	pneumonia=1	patient_type=2	other_disease=1	copd=1	hypertension=1	sex=1	diabetes=1	obesity=1	intubed=1	icu=1	tobacco=1	cardiovascular=1	renal_chronic=1	immunosuppression=1	asthma=1	exp number
0	0.76	-0.2	0.05	-0	-0	0.02	-0	0.02	0.01	-0	-0	-0	-0	-0	-0	-0																	0
0					-0	0.01	-0		0.02			-0				-0	-0.7	0.23	-0	0.03	0.03	0.02	-0	0.01	0	-0								1
0					-0	0.01	-0	0.02	0.08	-0	-0	-0	-0						-0.1	0.01					0.01	-0	-0.7	0.25	0.04					2
0							-0										-0.7	0.24	-0	0.02	0.02	0.01	0	0.01	0.01	0.01			0.02	0.04	-0	0.02	0.01	3
0			0.04		-0	-0	-0			-0	-0	-0	0	0.01	-0	-0	-0.7	0.24		0.03			-0								-0			4
1	-0.8	0.24	-0	0.03	0.03	-0	0.02	-0	-0	0.01	0.01	0.01	0.01	0	0	0																	0
1					0.03	-0	0.03		-0			0.02				0	0.73	-0.2	0.05	-0	-0	-0	0.01	-0	-0	0								1
1					0.02	-0	0.02	-0	-0.1	0	0	0.02	0.01						0.05	-0					-0	0.01	0.66	-0.2	-0					2
1							0.02										0.71	-0.2	0.05	-0	-0	-0	-0	-0	-0	-0			-0	-0	0.02	-0	-0	3
1			-0		0.02	0.03	0.02			0.01	0.01	0.03	-0	-0	0	0.02	0.72	-0.2		-0			0.02								0.01			4

Figure 21. Explanations generated by SP-LIME that show the impact of input variables on the classification problem

The same explainer used in SP-LIME to generate the most representative explanations for the ML model was used to generate explanations of specific instances, as shown in Figure 22. The explanation of a single instance shows the high prediction probability of recovery using the trained SVM and the impact of each variable on each class, with intubation having the highest impact on the recovery.

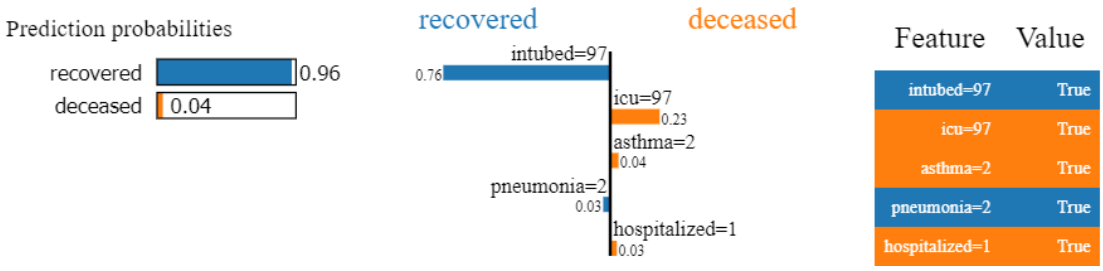

Figure 22. Explanation generated by LIME for one instance, indicating a higher probability of recovery and the weights of the most impacting features for each class

In the ontology, the LIME algorithm is described in terms of its authors, the source reference paper, and further descriptions to indicate that it is a local model-agnostic method that explains the ML model and the outputs, and that it is locally faithful to the underlying model. The explanations generated by the algorithm are also described regarding its format, i.e., the method generates static explanations in the format of the weight impact of the variables. Its implementation in Python is provided by the LIME library [3].

The generated explanations are included in the ontology as instances of *Explanation*. *SPLIMEExplanations* instantiates *ModelExplanation* because it explains the logic behind the whole ML model by using the most representative instance explanations, and each of the instances instantiates *ExplanationComponent*. *LIMEExplanation* instantiates *ResultExplanation* because it explains only one instance. The relator *ExplainsModel* indicates that the *SPLIMEExplanation* explains the *SVMModel*. The relator *ExplainsResultInstance* connects one prediction of the dataset to the rule that explains it.

5.4.3. Explanation Evaluation

The generated explanations are evaluated in the *EvaluateExplanation* operation, which executes the *EvaluationExplanationImplementation*, which is a procedure implemented in Python but not related to any already existing algorithm. The output of the *EvaluateExplanation* operation (*ExplanationEvaluation*) has components that describe each element of the explanation evaluation.

The RIPPER rules can be evaluated considering the coverage of the rules and the accuracy achieved by applying the rule set to predict the classes of a data set. Using the train set to identify the features and the coverage of each rule, we obtain the values in Figure 23, which are included in the ontology as characteristics of each component of the *ExplanationEvaluation*. Another component is the accuracy achieved by applying the rule set to determine the class of the test set, which has value 0.90.

The SP-LIME explanations can also be evaluated considering the coverage of the explanations. Using the train set, we obtain the number of instances covered by the explanation, which are included in the ontology as characteristics of each component of the *ExplanationEvaluation*.

```
['age', 'pneumonia', 'hypertension', 'hospitalized', 'sex'] 102
['pneumonia', 'age', 'icu', 'intubed'] 30
['pneumonia', 'age', 'icu', 'hypertension', 'asthma', 'copd'] 13
['pneumonia', 'age', 'icu', 'hypertension'] 165
['pneumonia', 'intubed', 'icu'] 76
['pneumonia', 'age', 'icu', 'sex'] 184
['renal_chronic', 'pneumonia', 'hospitalized', 'sex'] 29
['pneumonia', 'age', 'intubed'] 53
['pneumonia', 'age', 'icu', 'cardiovascular'] 22
['pneumonia', 'age', 'renal_chronic'] 15
['hospitalized', 'renal_chronic', 'other_disease'] 10
['pneumonia', 'renal_chronic', 'age'] 13
['icu', 'age', 'renal_chronic'] 16
['pneumonia', 'hypertension', 'age', 'sex', 'intubed'] 13
['age', 'hypertension', 'pneumonia', 'diabetes', 'sex', 'tobacco'] 28
['intubed', 'icu'] 82
```

Figure 23. Features of each rule of the rule set generated by RIPPER and the corresponding number of instances of the train set they cover

6. Evaluation

The last step of ontology development is ontology evaluation. SABiO's evaluation process proposes ontology verification, which aims to ensure that the ontology is built meeting specifications previously defined, such as the ontology requirements, and ontology validation, which aims to ensure that the ontology fulfills its intended purpose [10].

In order to verify the ontology, we first analyzed if the ontology meets the requirements defined in Section 3.2. For the ontology quality attributes, our ontology proposes a more generic module that can be further adapted and specialized, satisfying REQ2. For the project requirements, our operational ontology is implemented in Protégé represented in OWL, satisfying REQ3. Considering the intended uses-related requirements, our ontology is grounded in gUFO, complying with REQ4. With the aid of Protégé Reasoner and gUFO Protégé Plugin [65], we also checked the quality and correctness of the ontology implementation to assess if it meets language specifications in terms of having an ontology without inconsistencies and that satisfies rules of a gUFO-based ontology.

The first ontology quality attribute, REQ1, requires that the ontology is adequate for data scientists and developers to understand the adequacy of the ML model and make adaptations and improvements, and is related to ontology validation. Ontology validation is conducted by instantiating data of the ontology, implementing competency questions (CQ) as queries in the implementation environment, and checking if the obtained results are the expected outputs [10]. For this, the instances of the ontology are created based on the case study and we implement the CQ as queries using SPARQL, which consists of a query language applicable to RDF data models. The queries and the results obtained to answer the competency questions are described as follows.

6.1. Data Input

CQ1. Which data were used to train the model?

Figure 24 shows the SPARQL code used to query the ontology to obtain information about the data used to train the ML model in *Experiment1*. The same code was also applied to query the ontology for *Experiment2* by substituting the value "Experiment1" to "Experiment2".

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX gufo: <http://purl.org/nemo/gufo#>
PREFIX XMLo: <https://example.com#>
SELECT ?experiment ?WFExecution ?preprocessOperation ?inputdata ?characteristic ?property ?values
WHERE {
  {
    ?characteristic  ?property ?values.
    ?characteristic gufo:inheresIn ?inputdata.
    ?inputdata rdf:type XMLo:InputData.
    ?inputdata gufo:participatedIn ?preprocessOperation.
    ?preprocessOperation rdf:type XMLo:Preprocess.
    ?preprocessOperation  gufo:isEventProperPartOf  ?WFExecution.
    ?WFExecution gufo:isEventProperPartOf ?experiment.
    FILTER(?property not in (gufo:participatedIn, rdf:type, gufo:inheresIn)).
  }
  UNION{
    ?inputdata ?characteristic ?values.
    ?inputdata rdf:type XMLo:InputData.
    ?inputdata gufo:participatedIn ?preprocessOperation.
    ?preprocessOperation rdf:type XMLo:Preprocess.
    ?preprocessOperation  gufo:isEventProperPartOf  ?WFExecution.
    ?WFExecution gufo:isEventProperPartOf ?experiment.
    FILTER(?characteristic not in (gufo:participatedIn, rdf:type)).
  }
  FILTER(?experiment=XMLo:Experiment1).
  FILTER(?values != owl:NamedIndividual)
}
```

Figure 24. SPARQL query for CQ1 for Experiment1

The outputs of the query are depicted in Figure 25 and Figure 26 and represent the information of the same dataset (COVID-19) used in *Experiment1* and *Experiment2*. The images also show the same *MLWorkflow* execution for both experiments because the second experiment reused the workflow of the first experiment. The queries retrieved information about the data period, the source, when the source was accessed, the number of instances and features, and a description of the data.

experiment	WFExecution	preprocessOperation	inputdata	characteristic	property	
Experiment1	MLWorkflow_exec	PreprocessCovid_Exp1	Covid-19	Covid-19Period	hasBeginPointInXSDDate	"01-01-2020"^^<http://www.w3.org/2(
Experiment1	MLWorkflow_exec	PreprocessCovid_Exp1	Covid-19	Covid-19Period	hasEndPointInXSDDate	"29-06-2020"^^<http://www.w3.org/2(
Experiment1	MLWorkflow_exec	PreprocessCovid_Exp1	Covid-19	Covid-19Website	hasQualityValue	"https://www.gob.mx/salud/documen
Experiment1	MLWorkflow_exec	PreprocessCovid_Exp1	Covid-19	Covid-19AccessDate	hasQualityValue	"22-07-2020"^^<http://www.w3.org/2(
Experiment1	MLWorkflow_exec	PreprocessCovid_Exp1	Covid-19	Covid-19Features	hasQualityValue	"23"^^<http://www.w3.org/2001/XMLS
Experiment1	MLWorkflow_exec	PreprocessCovid_Exp1	Covid-19	Covid-19Instances	hasQualityValue	"9451"^^<http://www.w3.org/2001/XM
Experiment1	MLWorkflow_exec	PreprocessCovid_Exp1	Covid-19	Covid-19SourceName	hasQualityValue	"Dados Abiertos Mexico"^^<http://ww
Experiment1	MLWorkflow_exec	PreprocessCovid_Exp1	Covid-19	rdfs:comment		"Epidemiology dataset of tested peo

Figure 25. Output of the query for CQ1 related to Experiment1

experiment	WFExecution	preprocessOperation	inputdata	characteristic	property	values
Experiment2	MLWorkflow_exec	PreprocessCovid	Covid-19	Covid-19Period	hasBeginPointInXSDDat	"01-01-2020"^^<http://ww
Experiment2	MLWorkflow_exec	PreprocessCovid	Covid-19	Covid-19Period	hasEndPointInXSDDate	"29-06-2020"^^<http://ww
Experiment2	MLWorkflow_exec	PreprocessCovid	Covid-19	Covid-19Website	hasQualityValue	"https://www.gob.mx/salu
Experiment2	MLWorkflow_exec	PreprocessCovid	Covid-19	Covid-19AccessDate	hasQualityValue	"22-07-2020"^^<http://ww
Experiment2	MLWorkflow_exec	PreprocessCovid	Covid-19	Covid-19Features	hasQualityValue	"23"^^<http://www.w3.org
Experiment2	MLWorkflow_exec	PreprocessCovid	Covid-19	Covid-19Instances	hasQualityValue	"9451"^^<http://www.w3.(
Experiment2	MLWorkflow_exec	PreprocessCovid	Covid-19	Covid-19SourceName	hasQualityValue	"Dados Abiertos Mexico"
Experiment2	MLWorkflow_exec	PreprocessCovid	Covid-19	rdfs:comment		"Epidemiology dataset o

Figure 26. Output of the query for CQ1 related to Experiment2

CQ2. How balanced are the data?

The SPARQL code used to query the ontology to obtain information about the imbalances of the data used to train the ML model in the experiments is depicted in Figure 27.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX gufo: <http://purl.org/nemo/gufo#>
PREFIX XMLo: <https://example.com#>
SELECT ?experiment ?WFExecution  ?operation ?data   ?features ?feature_description ?featuresCharacteristic ?featurevalue
WHERE {
  ?features rdfs:comment ?feature_description.
  ?features gufo:isComponentOf ?data.
  ?data rdf:type XMLo:TrainData.
  ?data gufo:participatedIn ?operation.
  ?operation rdf:type XMLo:Train.
  ?operation  gufo:isEventProperPartOf  ?WFExecution.
  ?WFExecution gufo:isEventProperPartOf ?experiment.
  FILTER(?experiment=XMLo:Experiment1)
  OPTIONAL{
    ?featuresCharacteristic gufo:hasQualityValue ?featurevalue.
    ?featuresCharacteristic gufo:inheresIn ?features.
  }
}
ORDER BY ?features ?featuresCharacteristic
```

Figure 27. SPARQL query for CQ2 for Experiment1

Figure 28 shows a sample of the outputs of the query for *Experiment1*, containing the information of features of the dataset used in the *MLWorkflow*. This information is the same as in *Experiment2* because the second experiment reused the data of the first. The information comprises the features, their descriptions, the categories present in each feature in case of categorical data, and the number of instances of each category.

experiment	WFExecution	operation	data	features	feature_description	featuresCharacteristic	featurevalue
Experiment1	MLWorkflow_exec	TrainSVM	Covid-19TrainData_exp1	Asthma_Covid-19Train	"Asthma Identifies if the patient has a diag	Asthma_Covid-19Train_categories	"1 = Yes , 2 = No"^^<http://ww
Experiment1	MLWorkflow_exec	TrainSVM	Covid-19TrainData_exp1	COPD_Covid-19Train	"COPD Identifies if the patient has a diagn	COPD_Covid-19Train_1	"105"^^<http://www.w3.org/20
Experiment1	MLWorkflow_exec	TrainSVM	Covid-19TrainData_exp1	COPD_Covid-19Train	"COPD Identifies if the patient has a diagn	COPD_Covid-19Train_2	"6510"^^<http://www.w3.org/2
Experiment1	MLWorkflow_exec	TrainSVM	Covid-19TrainData_exp1	COPD_Covid-19Train	"COPD Identifies if the patient has a diagn	COPD_Covid-19Train_categories	"1 = Yes , 2 = No"^^<http://ww
Experiment1	MLWorkflow_exec	TrainSVM	Covid-19TrainData_exp1	Cardiovascular_Covid-19Train	"CARDIOVASCULAR Identifies if the patier	Cardiovascular_Covid-19Train_1	"132"^^<http://www.w3.org/20
Experiment1	MLWorkflow_exec	TrainSVM	Covid-19TrainData_exp1	Cardiovascular_Covid-19Train	"CARDIOVASCULAR Identifies if the patier	Cardiovascular_Covid-19Train_2	"6483"^^<http://www.w3.org/2
Experiment1	MLWorkflow_exec	TrainSVM	Covid-19TrainData_exp1	Cardiovascular_Covid-19Train	"CARDIOVASCULAR Identifies if the patier	Cardiovascular_Covid-19Train_categories	"1 = Yes , 2 = No"^^<http://ww
Experiment1	MLWorkflow_exec	TrainSVM	Covid-19TrainData_exp1	Death_Covid-19Train	"Indicates if the patient died or recovered."(	Death_Covid-19Train_1	"81"^^<http://www.w3.org/200
Experiment1	MLWorkflow_exec	TrainSVM	Covid-19TrainData_exp1	Death_Covid-19Train	"Indicates if the patient died or recovered."(	Death_Covid-19Train_2	"6534"^^<http://www.w3.org/2
Experiment1	MLWorkflow_exec	TrainSVM	Covid-19TrainData_exp1	Death_Covid-19Train	"Indicates if the patient died or recovered."(	Death_Covid-19Train_categories	"1 = Yes , 0 = No"^^<http://ww
Experiment1	MLWorkflow_exec	TrainSVM	Covid-19TrainData_exp1	Diabetes_Covid-19Train	"DIABETES Identifies if the patient has a di	Diabetes_Covid-19PTrain_categories	"1 = Yes , 2 = No"^^<http://ww
Experiment1	MLWorkflow_exec	TrainSVM	Covid-19TrainData_exp1	Diabetes_Covid-19Train	"DIABETES Identifies if the patient has a di	Diabetes_Covid-19Train_1	"956"^^<http://www.w3.org/20
Experiment1	MLWorkflow_exec	TrainSVM	Covid-19TrainData_exp1	Diabetes_Covid-19Train	"DIABETES Identifies if the patient has a di	Diabetes_Covid-19Train_2	"5659"^^<http://www.w3.org/2
Experiment1	MLWorkflow_exec	TrainSVM	Covid-19TrainData_exp1	Hospitalized_Covid-19Train	"Identifies it the patient returned home or w	Hospitalized_Covid-19Train_1	"4843"^^<http://www.w3.org/2
Experiment1	MLWorkflow_exec	TrainSVM	Covid-19TrainData_exp1	Hospitalized_Covid-19Train	"Identifies it the patient returned home or w	Hospitalized_Covid-19Train_2	"1772"^^<http://www.w3.org/2

Figure 28. Output sample of the query for CQ2

CQ3. How were the data preprocessed?

Figure 29 shows the SPARQL code used to query the ontology to obtain information about the preprocessing steps applied in the data, in terms of implementations and algorithms used to process the data. Since some implementations do not use well-known algorithms or are not pre-existing implementations provided by packages and libraries, we defined part of the code as optional.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX gufo: <http://purl.org/nemo/gufo#>
PREFIX XMLo: <https://example.com#>
SELECT  ?experiment ?WFExecution  ?operation  ?implementation  ?algorithm  ?operationtype ?parameter ?value
WHERE {
  ?implementation XMLo:executedBy ?operation.
  ?operation rdf:type XMLo:Preprocess.
  ?operation  gufo:isEventProperPartOf  ?WFExecution.
  ?WFExecution gufo:isEventProperPartOf ?experiment.
  FILTER(?experiment=XMLo:Experiment1)
  OPTIONAL{ #returns the algorithm related to the implementation, if applicable
    ?algorithm gufo:hasReifiedQualityValue ?operationtype .
    ?implementationrelation XMLo:implementsInvolvesAlgorithm ?algorithm.
    ?implementationrelation XMLo:implementsInvolvesImplementation ?implementation.
    ?hasParameter XMLo:hasParameterInvolvesImplementation ?implementation .
    ?hasParameter XMLo:hasParameterInvolvesParameter ?parameter.
    ?specifiesParameter XMLo:specifiesInvolvesParameter ?parameter.
    ?specifiesParameter XMLo:specifiesInvolvesParameterSetting ?paramSetting.
    ?characteristic gufo:inheresIn ?paramSetting.
    ?characteristic gufo:hasQualityValue ?value.
    ?paramSetting gufo:participatedIn ?operation.
  }
}
```

Figure 29. SPARQL query for CQ3 for Experiment1

The outputs of the query depicted in Figure 30 represent the information of the preprocessing steps in *Experiment1*, which consist of renaming columns, dropping columns, and splitting data. The preprocessing steps of *Experiment2,* obtained by substituting the value “Experiment1” to “Experiment2” in the query, are identical to *Experiment1*, since it reused the *MLWorkflow* of *Experiment1*. The result also shows the parameters used for each preprocessing step, being possible to keep track of changes in the dataset.

experiment	WFExecution	operation	implementation	algorithm	operationtype	parameter	
Experiment1	MLWorkflow_exec	PreprocessCovid_Exp1	RenameColumnPython	RenameColumn	TransformData	NewValueColumnParam	"[immunosuppression,h
Experiment1	MLWorkflow_exec	PreprocessCovid_Exp1	RenameColumnPython	RenameColumn	TransformData	RenameColumnParam	"[inmsupr,patient_type]"^
Experiment1	MLWorkflow_exec	PreprocessCovid_Exp1	SplitPython	Split	SplitData	TestSizeParam	"0.3"^^<http://www.w3.org
Experiment1	MLWorkflow_exec	PreprocessCovid_Exp1	RemoveColumnsPython	DropColumn	CleanData	DropColumnsParam	"['id', 'covid_res','entry_da

Figure 30. Output of the query for CQ3

CQ4. What are the correlations of the input datasets?

Figure 31 shows the SPARQL code used to query the ontology to obtain information about the correlations between features present in the data used to train the ML model.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX gufo: <http://purl.org/nemo/gufo#>
PREFIX XMLo: <https://example.com#>
SELECT ?experiment ?WFExecution  ?operation ?traindata    ?traindata  ?correlation
(SAMPLE(?features)  AS ?feature1)  (SAMPLE(?featurescorrelated) AS ?feature2) ?value
WHERE {
  ?correlationCharact gufo:hasQualityValue ?value.
  ?correlationCharact gufo:inheresIn ?correlation.
  ?correlation XMLo:correlates ?featurescorrelated.
  ?correlation XMLo:correlates ?features.
  ?features gufo:isComponentOf ?traindata.
  ?traindata rdf:type XMLo:TrainData.
  ?traindata gufo:participatedIn ?operation.
  ?operation rdf:type XMLo:Train.
  ?operation  gufo:isEventProperPartOf  ?WFExecution.
  ?WFExecution gufo:isEventProperPartOf ?experiment.
  FILTER(?experiment=XMLo:Experiment1) .
  FILTER(?featurescorrelated!=?features) .
}
GROUP BY  ?experiment ?WFExecution  ?operation ?traindata   ?traindata ?correlation ?value
ORDER BY DESC(?value)
```

Figure 31. SPARQL query for CQ4 for Experiment1

The outputs of the query depicted in Figure 32 represent the information of the correlations between features of the training data used in *Experiment1* in the *MLWorkflow*, after being preprocessed. Each line indicates one correlation between two features and the value of the correlation. For example, the biggest correlation is between *Intubated* and *ICU*, with a value of 0.99. The correlations values are the same for *Experiment2*.

experiment	WFExecution	operation	traindata	correlation	feature1	feature2	
Experiment1	MLWorkflow_exec	TrainSVM	Covid-19TrainData_exp1	IntubedCorrelatesICU	ICU_Covid-19Train	Intubed_Covid-19Train	"0.995625"^^
Experiment1	MLWorkflow_exec	TrainSVM	Covid-19TrainData_exp1	IntubedCorrelatesPneumonia	Pneumonia_Covid-19Train	Intubed_Covid-19Train	"0.658587"^^
Experiment1	MLWorkflow_exec	TrainSVM	Covid-19TrainData_exp1	DeathCorrelatesHospitalized	Death_Covid-19Train	Hospitalized_Covid-19Train	"0.489923"^^
Experiment1	MLWorkflow_exec	TrainSVM	Covid-19TrainData_exp1	HospitalizedCorrelatesAge	Hospitalized_Covid-19Train	Age_Covid-19Train	"0.402038"^^
Experiment1	MLWorkflow_exec	TrainSVM	Covid-19TrainData_exp1	HypertensionCorrelatesDiabetes	Diabetes_Covid-19Train	Hypertension_Covid-19Train	"0.364664"^^
Experiment1	MLWorkflow_exec	TrainSVM	Covid-19TrainData_exp1	DeathCorrelatesAge	Death_Covid-19Train	Age_Covid-19Train	"0.355090"^^
Experiment1	MLWorkflow_exec	TrainSVM	Covid-19TrainData_exp1	IntubedCorrelatesHypertension	Intubed_Covid-19Train	Hypertension_Covid-19Train	"0.253902"^^
Experiment1	MLWorkflow_exec	TrainSVM	Covid-19TrainData_exp1	HypertensionCorrelatesICU	Hypertension_Covid-19Train	ICU_Covid-19Train	"0.252025"^^
Experiment1	MLWorkflow_exec	TrainSVM	Covid-19TrainData_exp1	DiabetesCorrelatesICU	Diabetes_Covid-19Train	ICU_Covid-19Train	"0.245985"^^
Experiment1	MLWorkflow_exec	TrainSVM	Covid-19TrainData_exp1	IntubedCorrelatesDiabetes	Intubed_Covid-19Train	Diabetes_Covid-19Train	"0.245839"^^
Experiment1	MLWorkflow_exec	TrainSVM	Covid-19TrainData_exp1	DiabetesCorrelatesPneumonia	Pneumonia_Covid-19Train	Diabetes_Covid-19Train	"0.212427"^^
Experiment1	MLWorkflow_exec	TrainSVM	Covid-19TrainData_exp1	HypertensionCorrelatesPneumonia	Pneumonia_Covid-19Train	Hypertension_Covid-19Train	"0.210804"^^

Figure 32. Output of the query for CQ4

6.2. ML Algorithm and ML Model

CQ5. What are the characteristics of the ML algorithm?

The SPARQL code used to query the ontology to obtain information about the characteristics of the ML algorithm used in the experiments is depicted in Figure 33.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX gufo: <http://purl.org/nemo/gufo#>
PREFIX XMLo: <https://example.com#>
SELECT ?experiment ?WFExecution  ?operation     ?implementation  ?algorithm   ?characteristic ?value
WHERE {
  {
    ?characteristic gufo:hasQualityValue ?value.
    ?characteristic gufo:inheresIn ?algorithm .
    ?implementRelator XMLo:implementsInvolvesAlgorithm ?algorithm.
    ?implementRelator XMLo:implementsInvolvesImplementation ?implementation.
    ?implementation XMLo:executedBy ?operation.
    ?operation rdf:type XMLo:Train.
    ?operation  gufo:isEventProperPartOf  ?WFExecution.
    ?WFExecution gufo:isEventProperPartOf ?experiment.
  }
  UNION{
    ?algorithm?property ?value.
    ?implementRelator XMLo:implementsInvolvesAlgorithm ?algorithm.
    ?implementRelator XMLo:implementsInvolvesImplementation ?implementation.
    ?implementation XMLo:executedBy ?operation.
    ?operation rdf:type XMLo:Train.
    ?operation  gufo:isEventProperPartOf  ?WFExecution.
    ?WFExecution gufo:isEventProperPartOf ?experiment.
    OPTIONAL{
      ?value rdf:type ?type.
    }
    BIND ( IF(?property = gufo:hasReifiedQualityValue, ?type, ?property) AS ?characteristic).
  }
  FILTER(?experiment=XMLo:Experiment1) .
  FILTER(?characteristic not in (owl:NamedIndividual, rdf:type)) .
}
```

Figure 33. SPARQL query for CQ5 for Experiment1

Figure 34 shows the query results containing the characteristics of the ML algorithm. The SVM algorithm is described in terms of its author and transparency, which describes how the algorithm usually works. The results also indicate that the algorithm performs supervised learning and classification, which in this scenario requires tabular data.

experiment	WFExecution	operation	implement...	algorithm	characteristic	value
Experiment1	MLWorkflow_exec	TrainSVM	SVMPython	Support Vector Machine	SVMTransparency	"SVM maps training examples to points in space so as to maximise the
Experiment1	MLWorkflow_exec	TrainSVM	SVMPython	Support Vector Machine	LearningType	Supervised
Experiment1	MLWorkflow_exec	TrainSVM	SVMPython	Support Vector Machine	dc:date	"1963"^^<http://www.w3.org/2001/XMLSchema#date>
Experiment1	MLWorkflow_exec	TrainSVM	SVMPython	Support Vector Machine	dc:creator	"Vladimir Vapnik"^^<http://www.w3.org/2001/XMLSchema#string>
Experiment1	MLWorkflow_exec	TrainSVM	SVMPython	Support Vector Machine	DataType	Tabular
Experiment1	MLWorkflow_exec	TrainSVM	SVMPython	Support Vector Machine	MLTechnique	Classification
Experiment1	MLWorkflow_exec	TrainSVM	SVMPython	Support Vector Machine	rdfs:label	"Support Vector Machine"^^<http://www.w3.org/2001/XMLSchema#string

Figure 34. Output of the query for CQ5

CQ6. What is the logic behind the ML model?

Figure 35 shows the SPARQL code used to query the ontology to obtain information about the logic behind the ML model in *Experiment1*. This code was also applied to query the ontology for the second experiment by substituting the value “Experiment1” to “Experiment2”.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX gufo: <http://purl.org/nemo/gufo#>
PREFIX XMLo: <https://example.com#>
SELECT ?experiment   ?WFExecution  ?operation ?explanation  ?explanationComponent  ?value
WHERE {
  ?relator XMLo:explainsModelInvolvesModelExplanation ?explanation.
  ?explanationCharacteristic gufo:hasQualityValue ?value.
  ?explanationCharacteristic gufo:inheresIn ?explanationComponent.
  ?explanationComponent gufo:isComponentOf ?explanation.
  ?explanation gufo:wasCreatedIn ?operation.
  ?operation rdf:type XMLo:Explain.
  ?operation  gufo:isEventProperPartOf  ?WFExecution.
  ?WFExecution gufo:isEventProperPartOf ?experiment.
  FILTER(?experiment=XMLo:Experiment1) .
}
ORDER BY (?explanationComponent)
```

Figure 35. SPARQL query for CQ6 for Experiment1

The outputs of the query depicted in Figure 36 represent the information of the rules generated by the RIPPER method that aim to explain the logic behind the ML model.

Figure 37 represents the explanations generated by SP-LIME with explanations of the most relevant instances. In both figures, it is also possible to identify the different workflows and operations used to generate RIPPER rules and LIME explanations.

experiment	WFExecution	operation	explanation	explanationCo...	value
Experiment1	ExplainWorkflow_exec_exp1	ExplainSVMRIPPER	RipperExplanation	RipperRule01	"[age=66-99 ^ pneumonia=1 ^ hypertension=1 ^ hospitalized=2 ^ sex=2]"^^<http://ww
Experiment1	ExplainWorkflow_exec_exp1	ExplainSVMRIPPER	RipperExplanation	RipperRule02	"[pneumonia=1 ^ age=66-99 ^ icu=2 ^ intubed=1]"^^<http://www.w3.org/2001/XMLSch
Experiment1	ExplainWorkflow_exec_exp1	ExplainSVMRIPPER	RipperExplanation	RipperRule03	"[pneumonia=1 ^ age=66-99 ^ icu=2 ^ hypertension=1 ^ asthma=2 ^ copd=1]"^^<http
Experiment1	ExplainWorkflow_exec_exp1	ExplainSVMRIPPER	RipperExplanation	RipperRule04	"[pneumonia=1 ^ age=66-99 ^ icu=2 ^ hypertension=1]"^^<http://www.w3.org/2001/XI
Experiment1	ExplainWorkflow_exec_exp1	ExplainSVMRIPPER	RipperExplanation	RipperRule05	"[pneumonia=1 ^ intubed=1 ^ icu=2]"^^<http://www.w3.org/2001/XMLSchema#string>
Experiment1	ExplainWorkflow_exec_exp1	ExplainSVMRIPPER	RipperExplanation	RipperRule06	"[pneumonia=1 ^ age=66-99 ^ icu=2 ^ sex=2]"^^<http://www.w3.org/2001/XMLSchem
Experiment1	ExplainWorkflow_exec_exp1	ExplainSVMRIPPER	RipperExplanation	RipperRule07	"[renal_chronic=1 ^ pneumonia=1 ^ hospitalized=2 ^ sex=2]"^^<http://www.w3.org/20
Experiment1	ExplainWorkflow_exec_exp1	ExplainSVMRIPPER	RipperExplanation	RipperRule08	"[pneumonia=1 ^ age=66-99 ^ intubed=1]"^^<http://www.w3.org/2001/XMLSchema#s
Experiment1	ExplainWorkflow_exec_exp1	ExplainSVMRIPPER	RipperExplanation	RipperRule09	""^^<http://www.w3.org/2001/XMLSchema#string>
Experiment1	ExplainWorkflow_exec_exp1	ExplainSVMRIPPER	RipperExplanation	RipperRule10	"[pneumonia=1 ^ age=66-99 ^ renal_chronic=1]"^^<http://www.w3.org/2001/XMLSche
Experiment1	ExplainWorkflow_exec_exp1	ExplainSVMRIPPER	RipperExplanation	RipperRule11	"[hospitalized=2 ^ renal_chronic=1 ^ other_disease=1]"^^<http://www.w3.org/2001/XI
Experiment1	ExplainWorkflow_exec_exp1	ExplainSVMRIPPER	RipperExplanation	RipperRule12	"[pneumonia=1 ^ renal_chronic=1 ^ age=51-57]"^^<http://www.w3.org/2001/XMLSche
Experiment1	ExplainWorkflow_exec_exp1	ExplainSVMRIPPER	RipperExplanation	RipperRule13	"[icu=2 ^ age=66-99 ^ renal_chronic=1]"^^<http://www.w3.org/2001/XMLSchema#stri
Experiment1	ExplainWorkflow_exec_exp1	ExplainSVMRIPPER	RipperExplanation	RipperRule14	"[pneumonia=1 ^ hypertension=1 ^ age=57-66 ^ sex=2 ^ intubed=1]"^^<http://www.w3
Experiment1	ExplainWorkflow_exec_exp1	ExplainSVMRIPPER	RipperExplanation	RipperRule15	"[age=57-66 ^ hypertension=1 ^ pneumonia=1 ^ diabetes=2 ^ sex=2 ^ tobacco=2]"^^<
Experiment1	ExplainWorkflow_exec_exp1	ExplainSVMRIPPER	RipperExplanation	RipperRule16	"[intubed=1 ^ icu=2]"^^<http://www.w3.org/2001/XMLSchema#string>

Figure 36. Output of the query for CQ6 related to Experiment1

experiment	WFExecution	operation	explanation	explanationComponent	value
Experiment2	ExplainWorkflow_exec_exp2	ExplainSVMLIME	SPLimeExplanation	SPLimeExplanation01	"{'intubed=97': -0.7609351179630631, 'icu=97': 0.23133388669953558, 'pne
Experiment2	ExplainWorkflow_exec_exp2	ExplainSVMLIME	SPLimeExplanation	SPLimeExplanation02	"{'intubed=2': 0.7272498521085603, 'icu=2': -0.2421863611446646, 'pneum
Experiment2	ExplainWorkflow_exec_exp2	ExplainSVMLIME	SPLimeExplanation	SPLimeExplanation03	"{'intubed=1': 0.644651523416289, 'icu=1': -0.22907602901127197, 'other_c
Experiment2	ExplainWorkflow_exec_exp2	ExplainSVMLIME	SPLimeExplanation	SPLimeExplanation04	"{'intubed=97': -0.7485790835552334, 'icu=97': 0.2424409361424163, 'pneu
Experiment2	ExplainWorkflow_exec_exp2	ExplainSVMLIME	SPLimeExplanation	SPLimeExplanation05	"{'intubed=2': 0.7277132635958735, 'icu=2': -0.24741837384588292, 'pneur

Figure 37. Output of the query for CQ6 related to Experiment2

6.3. Output

CQ7. Why did the model generate this output?

Figure 38 shows the SPARQL code used to query the ontology to obtain information about the reason why did the ML model generate such outputs. This query aims to retrieve all predicted instances that have a connection with an explanation.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX gufo: <http://purl.org/nemo/gufo#>
PREFIX XMLo: <https://example.com#>
SELECT  ?experiment ?WFExecution ?operation   ?result   ?resultInstance  ?explanationComponent  ?value
WHERE {
  ?WFexec gufo:isEventProperPartOf ?experiment.
  ?operation2 gufo:isEventProperPartOf ?WFexec.
  ?explanation gufo:wasCreatedIn ?operation2.
  ?explanationComponent gufo:isComponentOf ?explanation.
  ?explanationCharacteristic gufo:hasQualityValue ?value.
  ?explanationCharacteristic gufo:inheresIn ?explanationComponent.
  ?explainResultInstanceRelation XMLo:explainsInvolvesExplanationComponent ?explanationComponent.
  ?explainResultInstanceRelation XMLo:explainsInvolvesResultInstance ?resultInstance.
  ?resultInstance gufo:isComponentOf ?result.
  ?result gufo:wasCreatedIn ?operation.
  ?operation rdf:type XMLo:Predict.
  ?operation  gufo:isEventProperPartOf  ?WFExecution.
  ?WFExecution gufo:isEventProperPartOf ?experiment.
  FILTER(?experiment=XMLo:Experiment1) .
}
```

Figure 38. SPARQL query for CQ7 for Experiment1

The outputs of the query depicted in Figure 39 show the predicted instances of the test set and the corresponding rules generated by the RIPPER method that are applicable to the instances in *Experiment1*. For example, result instance 15 can be explained by rule 7. Figure 40 depicts the predicted instances using the test set and their corresponding LIME explanations in *Experiment2*.

experiment	WFExecution	operation	result	resultInstance	explanationCo...	value
Experiment1	MLWorkflow_exec	PredictTest_exp1	TestPrediction_exp1	ResultInstance15_exp1	RipperRule07	"[renal_chronic=1 ^ pneumonia=1 ^ hospitalized=2 ^ sex=2]"^^<http://www.
Experiment1	MLWorkflow_exec	PredictTest_exp1	TestPrediction_exp1	ResultInstance28_exp1	RipperRule06	"[pneumonia=1 ^ age=66-99 ^ icu=2 ^ sex=2]"^^<http://www.w3.org/2001/X
Experiment1	MLWorkflow_exec	PredictTest_exp1	TestPrediction_exp1	ResultInstance46_exp1	RipperRule01	"[age=66-99 ^ pneumonia=1 ^ hypertension=1 ^ hospitalized=2 ^ sex=2]"^^

Figure 39. Output of the query for CQ7 concerning Experiment1

experiment	WFExecution	operation	result	resultInstance	explanationCompo...	value
Experiment2	MLWorkflow_exec	PredictTest	TestPrediction	ResultInstance01	LimeExplanation01	"[('intubed=2', 0.7151164489962197), ('icu=2', -0.2364075681141168), ('ag
Experiment2	MLWorkflow_exec	PredictTest	TestPrediction	ResultInstance02	LimeExplanation02	"[('intubed=2', 0.7039602887283274), ('icu=2', -0.23217606341882432), ('r
Experiment2	MLWorkflow_exec	PredictTest	TestPrediction	ResultInstance03	LimeExplanation03	"[('intubed=2', 0.7132545189983044), ('icu=2', -0.25052834811586927), ('r
Experiment2	MLWorkflow_exec	PredictTest	TestPrediction	ResultInstance00	LimeExplanation00	"[('intubed=97', -0.7475507216476007), ('icu=97', 0.25243822941504745),

Figure 40. Output of the query for CQ7 concerning Experiment2

6.4. ML Model Evaluation

CQ8. How was the ML model evaluated? What is the meaning of those metrics?

The SPARQL code used to query the ontology to obtain information about the metrics used to evaluate the ML model of the first experiment is depicted in Figure 41. Similar to all previous queries, this code was also applied to query the ontology for the second experiment by substituting the value “Experiment1” to “Experiment2”.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX gufo: <http://purl.org/nemo/gufo#>
PREFIX XMLo: <https://example.com#>
SELECT ?experiment ?WFExecution ?operation ?ModelEvaluation   ?measures ?comment ?value
WHERE{
  ?MeasureValue gufo:hasQualityValue ?value.
  ?MeasureValue gufo:inheresIn ?ModelEvaluationRelator.
  ?measures rdfs:comment ?comment.
  ?ModelEvaluationRelator XMLo:specifiesInvolvesEvaluationMeasure ?measures.
  ?ModelEvaluationRelator XMLo:specifiesInvolvesModelEvaluation ?ModelEvaluation.
  ?ModelEvaluation gufo:wasCreatedIn ?operation.
  ?operation rdf:type XMLo:EvaluateModel.
  ?operation  gufo:isEventProperPartOf  ?WFExecution.
  ?WFExecution gufo:isEventProperPartOf ?experiment.
  FILTER(?experiment=XMLo:Experiment1).
}
```

Figure 41. SPARQL query for CQ8 for Experiment1

The outputs of the query depicted in Figure 42 represent the metrics used to evaluate the ML model and the obtained value for the metric. In addition, the query result contains the description of the measure as comments.

experiment	WFExecution	operation	ModelEvaluation	measures	comment	value
Experiment1	MLWorkflow_exec	EvaluateSVM	SVMEvaluation	Sensitivity	"Sensitivity (True Positive rate) measures the pr	"0.79"^^<http://www.w3.org/2001/XMLSchema#decimal>
Experiment1	MLWorkflow_exec	EvaluateSVM	SVMEvaluation	Specificity	"Specificity (True Negative rate) measures the p	"0.9"^^<http://www.w3.org/2001/XMLSchema#decimal>
Experiment1	MLWorkflow_exec	EvaluateSVM	SVMEvaluation	Accuracy	"Accuracy is the proportion of correct predictions	"0.9"^^<http://www.w3.org/2001/XMLSchema#decimal>

Figure 42. Output of the query for CQ8

6.5. Explanation

CQ9. How were the explanations generated? How are the explanations presented to the user? How faithful are the explanations?

Figure 43 shows the SPARQL code used to query the ontology to obtain information about the explanations generated in the experiments.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX gufo: <http://purl.org/nemo/gufo#>
PREFIX XMLo: <https://example.com#>
SELECT ?experiment  ?WFExecution   ?operation  ?implementation ?algorithm    ?type ?characteristics
WHERE{
  ?characteristics rdf:type ?type.
  ?algorithm gufo:hasReifiedQualityValue ?characteristics.
  ?implemetsRelator XMLo:implementsInvolvesAlgorithm ?algorithm.
  ?implemetsRelator XMLo:implementsInvolvesImplementation ?implementation.
  ?implementation XMLo:executedBy ?operation.
  ?operation rdf:type XMLo:Explain.
  ?operation  gufo:isEventProperPartOf  ?WFExecution.
  ?WFExecution gufo:isEventProperPartOf ?experiment.
  FILTER(?experiment=XMLo:Experiment1) .
  FILTER(?type != owl:NamedIndividual).
}
```

Figure 43. SPARQL query for CQ9 for Experiment1

The outputs of the query depicted in Figure 44 indicate the implementation and algorithm used to generate explanations (RIPPER) in *Experiment1*, and the characteristics of the method. It is a global model-agnostic method that aims to explain the ML model, and it generates static explanations in the form of rules. The query also indicates that the explanation is unfaithful, so the user can be aware of how faithful the explanation is to the underlying ML model.

Figure 45 describes the LIME explanation method used in *Experiment2*, which is model-agnostic and locally faithful to the underlying classifier. It generates static local explanations that indicate the impact of the input variables by giving weights to them. LIME explanations explain outputs, but the module SP-LIME samples representative instances as a way to explain the whole ML model.

experiment	WFExecution	operation	implementati...	algorithm	type	characteristics
Experiment1	ExplainWorkflow_exec	ExplainSVM	RipperPython	RIPPER	ExplanationInteraction	Static
Experiment1	ExplainWorkflow_exec	ExplainSVM	RipperPython	RIPPER	ExplainableMethodFlexibility	ModelAgnostic
Experiment1	ExplainWorkflow_exec	ExplainSVM	RipperPython	RIPPER	ExplanationScope	Global
Experiment1	ExplainWorkflow_exec	ExplainSVM	RipperPython	RIPPER	ExplainedElement	ExplainsMLModel
Experiment1	ExplainWorkflow_exec	ExplainSVM	RipperPython	RIPPER	ExplanationFormat	Rules
Experiment1	ExplainWorkflow_exec	ExplainSVM	RipperPython	RIPPER	ExplanationFidelity	Unfaithful

Figure 44. Output of the query for CQ9 concerning Experiment1

experiment	WFExecution	operation	implement...	algorith...	type	characteristics
Experiment2	ExplainWorkflow_exec_exp2	ExplainSVMLIME	LIMEPython	LIME	ExplanationInteraction	Static
Experiment2	ExplainWorkflow_exec_exp2	ExplainSVMLIME	LIMEPython	LIME	ExplanationFidelity	LocalFidelity
Experiment2	ExplainWorkflow_exec_exp2	ExplainSVMLIME	LIMEPython	LIME	ExplanationScope	Local
Experiment2	ExplainWorkflow_exec_exp2	ExplainSVMLIME	LIMEPython	LIME	ExplanationFormat	InputVariableWeight
Experiment2	ExplainWorkflow_exec_exp2	ExplainSVMLIME	LIMEPython	LIME	ExplainableMethodFlexibility	ModelAgnostic
Experiment2	ExplainWorkflow_exec_exp2	ExplainSVMLIME	LIMEPython	LIME	ExplainedElement	ExplainedElementOutput
Experiment2	ExplainWorkflow_exec_exp2	ExplainSVMLIME	LIMEPython	LIME	ExplainedElement	ExplainedElementMLModel

Figure 45. Output of the query for CQ9 concerning Experiment2

CQ10. How general are the explanations (do they apply to all instances)?

Figure 46 shows the SPARQL code used to query the ontology to obtain information about the coverage of the explanations generated in the experiments.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX gufo: <http://purl.org/nemo/gufo#>
PREFIX XMLo: <https://example.com#>
SELECT ?experiment ?WFExecution ?operation  ?evaluation   ?evaluationComponent  ?type ?characteristics  ?value
WHERE {
  ?characteristics rdf:type ?type.
  ?characteristics gufo:hasQualityValue ?value.
  ?characteristics gufo:inheresIn ?evaluationComponent.
  ?evaluationComponent gufo:isComponentOf ?evaluation.
  ?evaluation gufo:wasCreatedIn ?operation.
  ?operation rdf:type XMLo:EvaluateExplanation.
  ?operation  gufo:isEventProperPartOf  ?WFExecution.
  ?WFExecution gufo:isEventProperPartOf ?experiment.
  FILTER(?type != owl:NamedIndividual).
  FILTER(?experiment=XMLo:Experiment1) .
}
```

Figure 46. SPARQL query for CQ10 for Experiment 1

Figure 47 depicts the result of the SPARQL to answer CQ10 for *Experiment1*. It shows the coverage of each RIPPER rule regarding the number of instances of the train set to which the rule is applicable. In addition, the query retrieved the accuracy of 0.9 obtained by applying the rules to predict the classes of instances from the test set.

Figure 48 shows each explanation of SP-LIME and the number of instances to which the explanations can be applied. For LIME, it generates explanations that explain the output, therefore, the explanation always covers only one instance.

experiment	WFExecution	operation	evaluation	evaluationCompon...	type	characteristics	value
Experiment1	ExplainWorkflow_exe	EvaluateRIPPER	RipperEvaluation	EvaluationRipperRul	Coverage	CoverageRipperRule13	"16"^^<http://www.w3
Experiment1	ExplainWorkflow_exe	EvaluateRIPPER	RipperEvaluation	EvaluationRipperRul	Coverage	CoverageRipperRule01	"102"^^<http://www.w
Experiment1	ExplainWorkflow_exe	EvaluateRIPPER	RipperEvaluation	EvaluationRipperRul	Coverage	CoverageRipperRule05	"76"^^<http://www.w3
Experiment1	ExplainWorkflow_exe	EvaluateRIPPER	RipperEvaluation	EvaluationRipperRul	Coverage	CoverageRipperRule08	"53"^^<http://www.w3
Experiment1	ExplainWorkflow_exe	EvaluateRIPPER	RipperEvaluation	EvaluationRipperRul	Coverage	CoverageRipperRule16	"82"^^<http://www.w3
Experiment1	ExplainWorkflow_exe	EvaluateRIPPER	RipperEvaluation	EvaluationRipperRul	Coverage	CoverageRipperRule04	"165"^^<http://www.w
Experiment1	ExplainWorkflow_exe	EvaluateRIPPER	RipperEvaluation	EvaluationRipperRul	Coverage	CoverageRipperRule07	"29"^^<http://www.w3
Experiment1	ExplainWorkflow_exe	EvaluateRIPPER	RipperEvaluation	EvaluationRipperRul	Coverage	CoverageRipperRule12	"13"^^<http://www.w3
Experiment1	ExplainWorkflow_exe	EvaluateRIPPER	RipperEvaluation	EvaluationRipperRul	Coverage	CoverageRipperRule03	"13"^^<http://www.w3
Experiment1	ExplainWorkflow_exe	EvaluateRIPPER	RipperEvaluation	EvaluationRipperRul	Coverage	CoverageRipperRule15	"28"^^<http://www.w3
Experiment1	ExplainWorkflow_exe	EvaluateRIPPER	RipperEvaluation	EvaluationRipperRul	Coverage	CoverageRipperRule11	"10"^^<http://www.w3
Experiment1	ExplainWorkflow_exe	EvaluateRIPPER	RipperEvaluation	EvaluationRipperRul	Coverage	CoverageRipperRule02	"30"^^<http://www.w3
Experiment1	ExplainWorkflow_exe	EvaluateRIPPER	RipperEvaluation	EvaluationRipperRul	Coverage	CoverageRipperRule14	"13"^^<http://www.w3
Experiment1	ExplainWorkflow_exe	EvaluateRIPPER	RipperEvaluation	EvaluationRipperRul	Coverage	CoverageRipperRule06	"184"^^<http://www.w
Experiment1	ExplainWorkflow_exe	EvaluateRIPPER	RipperEvaluation	EvaluationAccuracy	Accuracy	RipperRulesAccuracy	"0.90"^^<http://www.w
Experiment1	ExplainWorkflow_exe	EvaluateRIPPER	RipperEvaluation	EvaluationRipperRul	Coverage	CoverageRipperRule10	"15"^^<http://www.w3
Experiment1	ExplainWorkflow_exe	EvaluateRIPPER	RipperEvaluation	EvaluationRipperRul	Coverage	CoverageRipperRule09	"22"^^<http://www.w3

Figure 47. Output of the query for CQ10 related to Experiment1

experiment	WFExecution	operation	evaluation	evaluationComponent	type	characteristics	value
Experiment2	ExplainWorkflow_exec_exp2	EvaluateSPLIME	SPLimeEvaluation	EvaluationSPLimeExplar	Coverage	CoverageSPLimeExplanation0	"1408"^^<http://www.w3.org/20
Experiment2	ExplainWorkflow_exec_exp2	EvaluateSPLIME	SPLimeEvaluation	EvaluationSPLimeExplar	Coverage	CoverageSPLimeExplanation0	"1"^^<http://www.w3.org/2001/x
Experiment2	ExplainWorkflow_exec_exp2	EvaluateSPLIME	SPLimeEvaluation	EvaluationSPLimeExplar	Coverage	CoverageSPLimeExplanation0	"1"^^<http://www.w3.org/2001/x
Experiment2	ExplainWorkflow_exec_exp2	EvaluateSPLIME	SPLimeEvaluation	EvaluationSPLimeExplar	Coverage	CoverageSPLimeExplanation0	"1"^^<http://www.w3.org/2001/x
Experiment2	ExplainWorkflow_exec_exp2	EvaluateSPLIME	SPLimeEvaluation	EvaluationSPLimeExplar	Coverage	CoverageSPLimeExplanation0	"5"^^<http://www.w3.org/2001/x

Figure 48. Output of the query for CQ10 related to Experiment2

7. Final Remarks

In this chapter, we present the final remarks of this work, by first summarizing the main conclusions, followed by the contributions, limitations, and future work.

7.1. General Conclusions

This project provided an overview of the concepts related to Semantic Web Technologies, Machine Learning, and Explainable AI, focusing on how SWT are being applied to make ML explainable, and categorizing the existing solutions as ante-hoc and post-hoc methods. Focusing on post-hoc methods that generate explanations after the ML model is trained, we proposed an ontology to describe the main components of the ML process and post-hoc explanation process, providing means that enable a user to have a holistic understanding of why the ML model arrived at such specific results. We developed the ontology following the SABiO methodology and in the knowledge acquisition phase, we selected ML-SCHEMA as the domain-specific ontology and we grounded it in the foundational ontology gUFO, aiming at interoperability.

The process of grounding the domain ontology presented some challenges because it required the alignment of ontologies that follow different philosophies and deep knowledge of their concepts. To overcome this challenge, we studied each concept thoroughly to determine where to allocate each element, but the complexity of having two taxonomies in gUFO with many elements made it difficult to be certain whether the element was properly allocated. Hence, besides studying their concepts, we developed a conceptual model using OntoUML, whose functionalities showed up fundamental to facilitate the grounding process. It helped overcome the alignment difficulties by enabling transforming the conceptual model to the gUFO ontology, in a way that the transformation mapped the OntoUML stereotypes to both the taxonomy of individuals and the taxonomy of types. Cardinalities and restrictions in the conceptual model were also considered in the transformation. For example, cardinality restrictions were transformed into axioms, as well as descriptions of each component. However, specifying sub-properties of mediator in OntoUML was not possible, which was done manually in Protégé with the definition of domain and ranges in the gUFO-based ontology. OntoUML also made further maintenance difficult after we included instances to the ontology using Protégé because instances were lost in each new transformation.

After grounding MLS in gUFO, we structured our ontology using modules. The general ML module proposed a more generic ontology to represent ML processes that could be adapted or specialized to other ML processes besides classification. We developed this module by aligning the grounded ontology with the ML process, solving some issues encountered in MLS. For example, the lack of cardinality indicated that one Run of MLS generated both the ML model and the evaluation, when in reality they are usually generated in separate Runs. The specific ML module proposed a more detailed ML ontology to describe the ML process for classification, specifying each operation and its inputs and outputs. The explanation module extended the ontology to describe the ML explanation process, more specifically post-hoc explanation. We

finally added to the ontology the descriptors for each element that we considered necessary to describe the processes.

The ontology was then evaluated first by using the OntoUML plugin for Visual Paradigm, which was adopted to build the conceptual model and also provided verification, giving more assurance about the quality of the ontology. After the transformation of the conceptual model to the gUFO-based operational ontology, we used Protégé to create individuals according to a case study to validate the ontology.

We defined the case study considering the scenario of the COVID-19 disease, and trained an SVM model with data of infected patients to predict the mortality according to demographic data, such as age and gender, and comorbidities. We then applied existing explanation methods to get feature correlations among the training data, and post-hoc explanation methods to generate explanations concerning the behavior of the ML model, by generating rules with RIPPER and obtaining the impact of each variable on the result with LIME. The ontology was then populated with instances that describe the case study, which helped identify necessary changes. The reasoner and the UFO plugin available to Protégé also assisted the evaluation process by verifying consistency and ensuring that the rules of our gUFO-based ontology were satisfied.

The validation of the ontology consisted not only in populating the ontology with instances but also querying it, retrieving information for each CQ to ensure that the obtained results were the expected outputs and that the ontology fulfilled its intended purpose. The CQs correspond to the RQs of this project, which were answered as follows:

RQ1. Which data were used to train the model?

We used the ontology to describe and retrieve information about the data used to train the model, indicating the description of the data, the source, the period, the date it was accessed, and the number of instances and features.

RQ2. How balanced are the data?

We created individuals in the ontology to represent the features present in the data and, in case of categorical features, we included the number of instances of each category of the feature. This way, it was possible to represent and query imbalances present in categorical data.

RQ3. How were the data preprocessed?

The ontology was used to describe the steps taken to preprocess the data, for example, renaming columns, dropping columns, and splitting data. For each preprocessing step, we represented the parameters used, being possible to keep track of changes in the dataset. However, we assumed that the data were already preprocessed in the experiment, creating individuals in the ontology to represent the steps taken without more details, and including only information about the preprocessing programmatically performed, disregarding manual updates.

RQ4. What are the correlations of the input datasets?

We used the ontology to represent the strongest correlations between features in the dataset obtained by applying the XAI toolbox [58], indicating which were the features involved in the correlation and its value.

RQ5. What are the characteristics of the ML algorithm?

The ontology was used to describe and retrieve information about the type of the ML algorithm used in the experiment, the transparency containing the description of how the algorithm usually works, the author and origin of the algorithm (for example, a scientific paper), the type of learning applicable such as supervised learning, the type of task performed, and the type of data it required.

RQ6. What is the logic behind the ML model?

We used the ontology to represent the explanations generated by the two post-hoc methods, LIME and RIPPER, indicating the impact of each feature on the result and explaining the logic behind the model using rules, complementing each other. The ontology also indicated if the explanation explained the ML model or explained one instance of the output.

RQ7. Why did the model generate this output?

With a relation between the explanation obtained from the explanation method and the instance of the output result, we represented the predicted instance of the test set and the corresponding rule generated by the RIPPER method that was applicable to the instance, justifying why the ML model generated specifically this output. We also indicated the predicted instances and connected them to their corresponding LIME explanations.

RQ8. How was the ML model evaluated? What is the meaning of those metrics?

We used the ontology to describe the metrics that evaluated the ML model and the obtained value for the metric. The description of the measure could also be included in the ontology and queried.

RQ9. How were the explanations generated? How are the explanations presented to the user? How faithful are the explanations?

The results for queries of CQ9 retrieved information about the implementation and the algorithm used to generate explanations, together with their characteristics. For RIPPER, the query result indicated that it is a global model-agnostic method that aims to explain the ML model, and it generates static explanations in the format of rules. The query also returned the characteristic that the explanation is unfaithful, so the user could be aware of how faithful the explanation is to the underlying ML model. For LIME, the retrieved information described the LIME explanation method, which is model-agnostic and locally faithful to the underlying classifier. It generates static local explanations that indicate the impact of the input variables by giving weights to them. LIME explanations explain outputs, but the module SP-LIME samples representative instances as a way to explain the whole ML model.

RQ10. How general are the explanations (do they apply to all instances)?

In order to describe the coverage or generality of the explanations, we evaluated the explanations by implementing in the case study functions to count the number of instances covered by the RIPPER rules and the accuracy obtained by predicting the correct class using the rules. The coverage was then included in the ontology and could be retrieved in the queries for CQ10. The queries returned the coverage of each RIPPER rule regarding the number of instances of the train set that the rule is applicable. In addition, they retrieved the accuracy of 0.9 obtained by applying the rules to predict the classes of instances from the test set.

Similarly, for SP-LIME, we identified the number of instances on which each of the explanations could be applied. For LIME, it generates explanations that explain the output, therefore, the explanation always covers only one instance.

Research question: Can we leverage ML post-hoc explainability to classification tasks by enabling the user to have a holistic view of ML and explanation processes using ontologies?

We verified that the ontology described information about components of the ML classification process and post-hoc explanation process. It could be queried, retrieving the intended information and answering the competency questions, which correspond to the research questions of this project. By answering them, we ensure that the ontology fulfills its intended purpose.

This way, we understand that the user could also perform these queries to gather the desired information from the components of the processes, which gives a holistic view of them. Furthermore, obtaining information besides the explanations generated by post-hoc methods, such as the correlations between input features, the preprocessing steps, the characteristics of algorithms, and the provenance of the data, complement the post-hoc explanations leveraging the ML explainability.

7.2. Contributions

The main contribution of this work is an ontology that represents ML and ML explanation experiments, allowing data scientists and developers to have a holistic view and better understanding of the whole ML and explanation process, and assuring provenance, enabling them to keep track of the steps of the process. In the literature, we can find different ontologies for data mining and ML, but none of them covers the explanation process or aims at explainability. Our ontology describes components of the processes that can affect the ML result, starting from the input data until the evaluation of the explanation. The ontology can be used to store metadata about these experiments in a central repository, and the stored metadata can be queried to get details of each step of learning and explanation.

The information that could be retrieved from our ontology during the validation process showed that the ontology was capable of answering all competency questions, consequently, answering the research questions defined in this project, leveraging the post-hoc explainability. We verified that our ontology was capable of describing the data used to train the ML model (with their source, instances, and features), detailing the imbalances of categorical features in the data, representing correlations between features, and keeping track of the preprocessing steps applied to the data.

Our ontology also described the ML algorithm and the trained ML model with their characteristics and the logic behind the model, represented the reason why the ML generated specific outputs, and described the metrics used to evaluate the ML model, providing details of the meaning of those metrics. In terms of describing the explanation process, the ontology was used to describe post-hoc explanation methods and the generated explanations, connecting the explanations to the ML model or predictions they explain, and describing the evaluation of these explanations in terms of accuracy, faithfulness and coverage.

The ontology is based on already existing ontologies, but it also considered different points of view from domain experts. MLS, which is the main domain-specific ontology that inspired the development of our ontology, was aligned with the ML process and grounded to a foundational ontology, becoming interoperable with existing ontologies that follow UFO. The development of the ontology followed best practices adhering to the SABiO methodology and explored the use of existing technologies for ontology engineering, for instance, the OntoUML plugin and the UFO plugin for Protégé, being developed in OWL.

Our ontology allows the description of different kinds of ML experiments, with different ML algorithms, and it is modularized in a way that the general ML module can be extended and used for other purposes. It also can be used to describe different post-hoc explanation methods. Furthermore, the information that describes ML algorithms, explanation methods, metrics, etc. can be easily reused.

7.3. Limitations

One of the limitations of the developed ontology concerns the lack of involvement of users, such as data scientists and developers to validate the use of the ontology as a tool that complements the explanations and helps understand the adequacy of the ML Model. The validation was mainly carried out by instantiating the ontology and using it to retrieve information to answer the competency questions.

The ontology was evaluated only with textual datasets, but not with images or other different kinds of data, neither integrated datasets, which combines data from multiple technologies. Additionally, considering that we focused on classification tasks, we represented only the imbalances of categorical features of the data. However, some limitations can be introduced to REQ2 in situations that use the ontology to represent imbalances for continuous features.

We also assumed that the dataset was preprocessed, which can impose some limitations on REQ3. Although we can represent the steps taken in preprocessing the data, it is only possible to model the input and output states of the features and the steps taken in the preprocessing operation, but we did not consider manual preprocessing steps nor represent in detail the preprocessing steps that are machine learning models themselves, for example, when applying ML to execute dimensionality reduction or feature extraction.

In addition, the correlations between features detected using the explainability toolbox EthicalML-XAI [58] were identified in the training data, but the ontology does not keep track of these correlations in the subsequent steps. In order to be able to identify the correlations and their

influence on the results or to keep track of them in each step of the process, it is necessary to have an explanation method that can detect this influence. Moreover, the ontology represented the explanations by modeling the components of the explanation generated by a post-hoc explanation method. Nonetheless, the explanation as a mapping between the inputs to the output was not explicitly represented in the ontology, keeping the ontology more generic for different types of explanations generated by post-hoc explanation methods.

Another limitation concerns the use of the lightweight version of UFO, gUFO. On the one hand, gUFO aims to guarantee computational properties and it is supported by OntoUML, which was used to develop the conceptual models and transform them into gUFO-based operational ontology, facilitating the grounding process and the design and implementation steps. On the other hand, the lightweight version lacks the support of UFO-C, whose elements could be used in this ontology to represent, for example, actions and goals. The components suitable to be modeled with components of UFO-C were modeled considering only the available classes in UFO-A and B, for example, with events and objects.

Furthermore, manually feeding the ontology with instances was laborious and time-consuming, especially when including characteristics of features from the data, such as the number of categories and the number of instances of each category. Automation of this step should facilitate and encourage the use of the ontology.

7.4. Future Work

As future work, we propose automating the process of feeding the ontology with instances by adopting technologies that can create individuals in the ontology while conducting the experiments. The automation would gather information from the data and the workflows and automatically feed the ontology. However, some information will still need to be included manually, for example, the characteristics of algorithms.

Since this project focuses on tackling post-hoc explanation methods for supervised learning, more specific classification, the ontology can be further tested with other ML models for other purposes and with other types of data, verifying if they have peculiarities that should be modeled or extending the vocabulary. Likewise, it can be tested using other post-hoc explanation methods besides RIPPER and LIME.

Finally, considering the novelty of exploiting the post-hoc explanation methods and representing them using ontologies, ante-hoc approaches could also be evaluated in terms of whether they can be generically represented by a single ontology, taking into account the diverse approaches that require changes in the implementation of ML algorithms. If this is the case, the ontology presented in this work can be analyzed regarding its suitability to be extended to cover also ante-hoc approaches, or if it is necessary to develop a new ontology.

Appendix A. Dictionary of Terms

Table 5. Dictionary of Terms

Module	Term	Definition
General	Experiment	An experiment is a complex process composed of sub-processes that belong together to do some kind of analysis on its results. This analysis can be general or very specific (e.g. a hypothesis test), having a specific goal.
General	WFExecution	Workflow execution (WFExecution) is a complex process composed of series of operations. It executes the implementations of operations by executing Workflows, which organize sequentially the implementations of these operations.
General	Workflow	Workflow is a complex structure that is composed of series of implementations that need to be completed sequentially. One experiment may have, for example, a workflow for the ML process and a workflow for the explanation.
General	Operation	The operation, similarly as in DMOP [40], is an execution of an implementation that accomplishes a specific piece of work that needs to be addressed. It is limited in time (has a start and end point), can be successful or failed.
General	Goal	The main goal of the experiment, for example, is to diagnose patients with COVID-19.
General	Data	Data are facts and statistics collected together for reference or analysis. They can be images, tables, texts, etc.
General	InputData	Input data are the data used in the experiment. It is a phase of the data, representing the raw data before the preprocessing step.
General	Feature	Features are measurable properties or characteristics of a phenomenon. In a dataset, they can be the input variables.
General	Output	Artifact created in the Operation, which is the event that brought the output into existence. It can be a preprocessed dataset, a machine learning model, an evaluation, etc.
General	ParameterSetting	ParameterSetting is an entity that connects a parameter and its value that is being set before an implementation execution.
General	Parameter	Prior parameter of implementation, i.e., a parameter that is set before the implementation execution. For example, a Hyperparameter when training an ML model.
General	Implementation	Implementation is an executable implementation of an algorithm. It is versioned and sometimes belongs to a library (e.g. scikit-learn).
General	Software	Software is implemented in computer programs, procedures, scripts, or rules with associated documentation, possibly constituting an organized environment, stored in read/write memory to be executed within a computer system [9].

General	MLModel	Machine Learning Model (MLModel) is a generalization of a set of training data able to predict values for unseen instances. It is an output from the Train operation, which executes an ML algorithm implementation. ML models have a dual nature. They can be treated as data structures and as such represented, stored, and manipulated. On the other hand, they act as functions and are executed, taking as input data examples and giving as output the result of applying the function to a data example. Models can also be divided into global or local ones. A global model has global coverage of a data set, i.e., it generalizes the whole data set. A local model, such as a pattern set, is a set of local hypotheses, i.e. each applies to a limited region of the data set [9].
General	ModelEvaluation	ModelEvaluation is a setting of a value of the performance measure specified by the evaluation specification. It connects an evaluation measure with its value.
General	Algorithm	The algorithm regardless of software implementation. An algorithm can be a machine learning algorithm, a preprocessing algorithm, an evaluation procedure, etc.
General	MLAlgorithm	Machine Learning Algorithm (MLAlgorithm) is a specific type of algorithm that is used to perform machine learning, such as NN, SVM, Decision Trees, Logistic Regression.
General	EvaluationProcedure	EvaluationProcedure evaluates ML models, being considered a specific type of algorithm, and can be implemented in different programming languages. Examples are cross-validation and leave-one-out.
General	EvaluationMeasure	EvaluationMeasure is a measure to assess the performance of the ML model generated by the train operation. Examples are accuracy or f-measure.
General	SpecifiesParameter	Relator that indicates that the ParameterSetting is specified by the Parameter.
General	HasParameter	Relator that indicates that the Implementation may have Parameter(s).
General	Implements	Relator that indicates that the Implementation implements an Algorithm.
General	Addresses	Relator that indicates that the ML Algorithm addresses the goal of the study.
General	SpecifiesMeasure	Relator that indicates that the Goal of the study specifies which evaluation measures are adequate to evaluate the ML model.
General	SpecifiesModelEvaluation	Relator that indicates that the evaluation measures specify the model evaluation.
General	Assesses	Relator that indicates that the model evaluation assesses the Machine Learning Model.
Specific	Preprocess	A specific type of operation that focuses on preprocessing the input data to make them more adequate to train and test the ML model.
Specific	Train	A specific type of operation that trains the ML model, executing the ML Algorithm implementation.
Specific	Test(Predict)	A specific type of operation that executes the PredictImplementation, generating results from the testing data.
Specific	EvaluateModel	A specific type of operation that focuses on evaluating the ML model. It executes the implementation of the evaluation procedure, generating the values for the measurements that assess the ML model.
Specific	PreprocessedData	Type of output that is generated by the preprocessing operation. It

		represents the phase of the data after the preprocessing step.
Specific	TrainData	It is part of preprocessed data that is usually used by the Train operation but can be used by Predict operation to generate explanations.
Specific	TestData	It is part of preprocessed data that is usually used by the Test (predict) operation.
Specific	PreprocessImplementation	A specific type of implementation that implements preprocessing algorithms.
Specific	MLImplementation	A specific type of implementation that implements ML Algorithms.
Specific	PredictImplementation	A specific type of implementation that implements the code to generate predictions/results by calling the MLModel as a function.
Specific	EvaluateModelImplementation	A specific type of implementation that implements the evaluation procedure.
Specific	PreprocessingAlgorithm	A specific type of algorithm that focuses on preprocessing the input data making them more adequate to train and test the ML model.
Specific	Result	Type of output that is generated by the test (predict) operation. It has the results or predictions generated using the fitted MLModel.
Specific	ResultInstance	One ResultInstance represents one result or prediction from the total results.
Specific	Correlates	Relator that indicates that two features from the data are mathematically correlated.
Specific	Calls	Relator that indicates that the implementation used to generate results or predictions calls the MLModel, which can act as functions and are executed.
Specific	hasMeasure	Relator that indicates that the implementation of the evaluation procedure takes into account the evaluation measurements.
Explanation	Explain	A specific type of operation that focuses on generating the explanations from the MLModel.
Explanation	EvaluateExplanation	A specific type of operation that focuses on evaluating the explanations generated by the Explain operation.
Explanation	ExplainImplementation	A specific type of implementation that implements the explainable algorithm.
Explanation	EvaluateExplanationImplementation	A specific type of implementation that implements the evaluation of the explanation.
Explanation	ExplainableAlgorithm	A specific type of post-hoc method that focuses on generating explanations from the MLModel.
Explanation	Explanation	Type of output that is generated by the explain operation, which contains the explanation generated by the post-hoc method.
Explanation	ExplanationComponent	Components of the Explanation, which can represent the explanation of a single instance or one single rule.
Explanation	ResultExplanation	A specific type of explanation, which focuses on explaining the result (predictions) obtained by applying the MLModel function to the data.
Explanation	ModelExplanation	A specific type of explanation, which can focus on explaining the fitted MLModel by expressing the logic behind it.

Explanation	ExplanationEvaluation	Type of output that is generated by the EvaluateExplanation operation. It indicates, for example, the fidelity of the Explanation or the coverage of its components.
Explanation	ExplanationComponentEvaluation	A component of the explanation evaluation that evaluates explanation components (e.g., coverage).
Explanation	ExplainsModel	Relator that indicates that the Model Explanation explains the ML model, instead of explaining only one instance of the result.
Explanation	ExplainsResultInstance	Relator that indicates that the explanation component explains the result instance, enabling describing the explanation of a specific instance of the result.
Explanation	EvaluatesExplanationComponent	Relator that indicates that the component of the explanation evaluation evaluates explanation components (e.g., coverage).

Appendix B. Axioms

Table 6. Axioms of the ontology

Module	Axiom	Formalization in OWL
General	An Experiment is part of a Study	SubClassOf(ObjectSomeValuesFrom(a:isEventProperPartOf a:Experiment) a:Study)
General	An Experiment has at least one WFExecution	SubClassOf(ObjectMinCardinality(1 a:isEventProperPartOf a:WFExecution) a:Experiment)
General	Every Experiment has one Goal	SubClassOf(ObjectExactCardinality(1 a:participatedIn a:Goal) a:Experiment)
General	A WFExecution has at least one Operation	SubClassOf(ObjectMinCardinality(1 a:IsEventProperPartOf a:Operation) a:WFExecution)
General	A Workflow has at least one Implementation	SubClassOf(ObjectMinCardinality(1 a:IsComponentOf a:Implementation) a:Workflow)
General	The Goal defines the EvaluationMeasures	SubclassOf(a:specifiesMeasure a:relator) SubObjectPropertyOf(a:specifiesMeasureInvolvesGoal a:mediates) ObjectPropertyDomain(a:specifiesMeasureInvolvesGoal a:SpecifiesMeasure) ObjectPropertyRange(a:specifiesMeasureInvolvesGoal a:Goal) SubObjectPropertyOf(a:specifiesMeasureInvolvesMeasure a:mediates) ObjectPropertyDomain(a:specifiesMeasureInvolvesMeasure a:SpecifiesMeasure) ObjectPropertyRange(a:specifiesMeasureInvolvesMeasure a:EvaluationMeasure)
General	An MLAlgorithm addresses a Goal	SubclassOf(a:Addresses a:relator) SubObjectPropertyOf(a:addressesInvolvesAlgorithm a:mediates) ObjectPropertyDomain(a:addressesInvolvesAlgorithm a:Addresses) ObjectPropertyRange(a:addressesInvolvesAlgorithm a:Algorithm) SubObjectPropertyOf(a:addressesInvolvesGoal a:mediates) ObjectPropertyDomain(a:addressesInvolvesGoal a:Addresses) ObjectPropertyRange(a:addressesInvolvesGoal a:Goal)
General	MLAlgorithm is an Algorithm	SubClassOf(a:MLAlgorithm a:Algorithm)
General	EvaluationProcedure is an Algorithm	SubClassOf(a:EvaluationProcedure a:Algorithm)
General	executedBy implies participatedIn	SubObjectPropertyOf(a: executedBy a:participatedIn)
General	One Operation executes one or more Implementation	SubClassOf(ObjectMinCardinality(1 a:ExecutedBy a:Implementation) a:Operation)
General	One Implementation implements Algorithms	SubclassOf(a:Implements a:relator) SubObjectPropertyOf(a:implementsInvolvesAlgorithm a:mediates) ObjectPropertyDomain(a:implementsInvolvesAlgorithm a:Implements) ObjectPropertyRange(a:implementsInvolvesAlgorithm a:Algorithm) SubObjectPropertyOf(a:implementsInvolvesImplementation a:mediates) ObjectPropertyDomain(a:implementsInvolvesImplementation a:Implements) ObjectPropertyRange(a:implementsInvolvesImplementation a:Implementation)
General	One Implementation is a component of a Software	SubClassOf(ObjectExactlyCardinality(1 a:IsComponentOf a:Implementation) a:Software)

General	One Implementation has Parameters	SubclassOf(a:HasParameter a:relator) SubObjectPropertyOf(a:hasParameterInvolvesImplementation a:mediates) ObjectPropertyDomain(a:hasParameterInvolvesImplementation a:HasParameter) ObjectPropertyRange(a:hasParameterInvolvesImplementation a:Implementation) SubObjectPropertyOf(a:hasParameterInvolvesParameter a:mediates) ObjectPropertyDomain(a:hasParameterInvolvesParameter a:HasParameter) ObjectPropertyRange(a:hasParameterInvolvesParameter a:Parameter)
General	Parameters are set when executing the Operation.	SubClassOf(ObjectExactlyCardinality(1 a:ParticipatedIn a:ParameterSetting) a:Operation)
General	Operations generate at least one Output	SubClassOf(ObjectExactlyCardinality(1 a:wasCreatedIn a:Output) a:Operation) SubClassOf(ObjectSomeValuesFrom(ObjectInverseOf(a:wasCreatedIn) a:Operation) a:Output)
General	An MLModel is an Output	SubClassOf(a:MLModel a:Output)
General	Each Output (Explanation, MLModel) is disjoint with each other	DisjointClasses(a:Explanation a:MLModel a:ExplanationEvaluation a:ModelEvaluation a:Result a:PreprocessedData)
General	A ModelEvaluation is a specialization of an Output	SubClassOf(a:ModelEvaluation a:Output)
General	Features are components of Data	SubClassOf(ObjectExactlyCardinality(1 a:IsComponentOf a:Feature) a:Data)
General	Data are generalizations of InputData	SubClassOf(a:InputData a:Data)
General	One Feature has at least one FeatureCharacteristic	SubClassOf(ObjectMinCardinality(1 ObjectInverseOf(a:inheresIn) a:Feature) a:FeatureCharacteristic)
General	Data have at least one DataCharacteristic	SubClassOf(ObjectMinCardinality(1 ObjectInverseOf(a:inheresIn) a:Data) a:DataCharacteristic)
General	MLModel may have ModelCharacteristics	SubClassOf(ObjectSomeValuesFrom(ObjectInverseOf(a:inheresIn) a:MLModel) a:ModelCharacteristic)
General	One Implementation may have ImplementationCharacteristics	SubClassOf(ObjectSomeValuesFrom(ObjectInverseOf(a:inheresIn) a:Implementation) a:ImplementationCharacteristic)
General	One ModelEvaluation has one or more ModelEvaluationCharacteristic	SubClassOf(ObjectMinCardinality(1 ObjectInverseOf(a:inheresIn) a:ModelEvaluation) a:ModelEvaluationCharacteristic)
General	One Algorithm can have more than one characteristic	SubClassOf(ObjectMinCardinality(1 ObjectInverseOf(a:inheresIn) a:Algorithm) a:AlgorithmCharacteristic)
General	All objects are disjoint to others except for Output and Data	DisjointClasses(a:Algorithm a:Parameter a:Feature a:ResultInstance a:Implementation a:Goal a:ExplanationComponentEvaluation a:Workflow a:Software a:Data a:ExplanationComponent a:EvaluationMeasure a:ParameterSetting)
General	One Characteristic (quality) is disjoint with other Characteristics	DisjointClasses(a:AlgorithmCharacteristic a:CorrelationCharacteristic a:DataCharacteristic a:ExperimentCharacteristic a:ExplanationComponentCharacteristic a:ExplanationComponentEvaluationCharacteristic a:FeatureCharacteristic a:GoalCharacteristic a:ImplementationCharacteristic a:ModelCharacteristic a:ModelEvaluationCharacteristic a:ParameterSettingCharacteristic)

General	One ParameterSetting can have only one characteristic, which is the value set to the parameter	SubClassOf(ObjectExactlyCardinality(1 ObjectInverseOf(a:inheresIn) a:ParameterSetting) a:ParameterSettingCharacteristic)
General	Preprocess operation occurs before Train	SubClassOf(ObjectSomeValuesFrom(ObjectInverseOf(a:historicallyDependsOn) a:Preprocess) a:Train)
General	Test(Predict) operation is executed after Train	SubClassOf(ObjectMinCardinality(1 a:historicallyDependsOn a:Predict) a:Train)
General	EvaluateModel is executed after Test(Predict) operation	SubClassOf(ObjectMinCardinality(1 a:historicallyDependsOn a:EvaluateModel) a:Predict)
Specific	Operation is a generalization of Preprocess, Train, Predict, and EvaluateModel	SubClassOf(a:Preprocess a:Operation) SubClassOf(a:Train a:Operation) SubClassOf(a:Predict a:Operation) SubClassOf(a:EvaluateModel a:Operation)
Specific	Each operation is disjoint to the others	DisjointClasses(a:EvaluateExplanation a:EvaluateModel a:Explain a:Predict a:Preprocess a:Train)
Specific	PreprocessingAlgorithm is an Algorithm	SubClassOf(a:PreprocessingAlgorithm a:Algorithm)
Specific	Preprocess operation receives as input the InputData	SubClassOf(ObjectMinCardinality(1 ObjectInverseOf(a:participatedIn) a:Preprocess) a:InputData)
Specific	Preprocess operation generates PreprocessedData	SubClassOf(ObjectExactlyCardinality(1 a:wasCreatedIn a:PreprocessedData) a:Preprocess)
Specific	Preprocess executes PreprocessImplementation	SubClassOf(ObjectMinCardinality(1 ObjectInverseOf(a:ExecutedBy) a:Preprocess) a:PreprocessImplementation)
Specific	Train operation executes MLImplementation	SubClassOf(ObjectMinCardinality(1 ObjectInverseOf(a:ExecutedBy) a:Train) a:MLImplementation)
Specific	Train operation receives TrainData	SubClassOf(ObjectMinCardinality(1 ObjectInverseOf(a:participatedIn) a:Train) a:TrainData)
Specific	Train operation generates MLModel	SubClassOf(ObjectExactlyCardinality(1 a:wasCreatedIn a:MLModel) a:Train)
Specific	Test(Predict) operation receives PreprocessedData	SubClassOf(ObjectMinCardinality(1 ObjectInverseOf(a:participatedIn) a:Predict) a:PreprocessedData)
Specific	Test(Predict) operation executes PredictImplementation	SubClassOf(ObjectMinCardinality(1 ObjectInverseOf(a:ExecutedBy) a:Predict) a:PredictImplementation)
Specific	PredictImplementation calls MLModel as a function	SubclassOf(a:Calls a:relator) SubObjectPropertyOf(a:callInvolvesMLModel a:mediates) ObjectPropertyDomain(a:callInvolvesMLModel a:Calls) ObjectPropertyRange(a:callInvolvesMLModel a:MLModel) SubObjectPropertyOf(a:callInvolvesPredictImplementation a:mediates) ObjectPropertyDomain(a:callInvolvesPredictImplementation a:Calls) ObjectPropertyRange(a:callInvolvesPredictImplementation a:PredictImplementation)
Specific	Test (predict) operation generates Result	SubClassOf(ObjectExactlyCardinality(1 a:wasCreatedIn a:Result) a:Predict)
Specific	The result is compound by	SubClassOf(ObjectExactlyCardinality(1 a:IsComponentOf a:ResultInstances)

	ResultInstances	a:Result)
Specific	Evaluate operation receives TestData and Result	SubClassOf(ObjectMinCardinality(1 ObjectInverseOf(a:participatedIn) a:EvaluateModel) a:TestData) SubClassOf(ObjectMinCardinality(1 ObjectInverseOf(a:participatedIn) a:EvaluateModel) a:Result)
Specific	EvaluateModel executes EvaluateModelImplementation	SubClassOf(ObjectMinCardinality(1 ObjectInverseOf(a:ExecutedBy) a:EvaluateModel) a:EvaluateModelImplementation)
Specific	EvaluationModelImplementation has EvaluationMeasures	SubclassOf(a:HasMeasure a:relator) SubObjectPropertyOf(a:hasMeasureInvolvesEvaluateModelImplementation a:mediates) ObjectPropertyDomain(a:hasMeasureInvolvesEvaluateModelImplementation a:HasMeasure) ObjectPropertyRange(a:hasMeasureInvolvesEvaluateModelImplementation a:EvaluateModelImplementation) SubObjectPropertyOf(a:hasMeasureInvolvesMeasure a:mediates) ObjectPropertyDomain(a:hasMeasureInvolvesMeasure a:HasMeasure) ObjectPropertyRange(a:hasMeasureInvolvesMeasure a:EvaluationMeasure)
Specific	EvaluateModel operation generates ModelEvaluation	SubClassOf(ObjectExactlyCardinality(1 a:wasCreatedIn a:ModelEvaluation) a:EvaluateModel)
Specific	ModelEvaluation is specified by the EvaluationMeasure	SubclassOf(a:SpecifiesModelEvaluation a:relator) SubObjectPropertyOf(a:specifiesInvolvesModelEvaluation a:mediates) ObjectPropertyDomain(a:specifiesInvolvesModelEvaluation a:SpecifiesModelEvaluation) ObjectPropertyRange(a:specifiesInvolvesModelEvaluation a:ModelEvaluation) SubObjectPropertyOf(a:specifiesInvolvesEvaluationMeasure a:mediates) ObjectPropertyDomain(a:specifiesInvolvesEvaluationMeasure a:SpecifiesModelEvaluation) ObjectPropertyRange(a:specifiesInvolvesEvaluationMeasure a:EvaluationMeasure)
Specific	Feature can have correlations to another Feature	SubclassOf(a:Correlates a:relator) SubObjectPropertyOf(a:correlates a:mediates) ObjectPropertyDomain(a:correlates a:Correlates) ObjectPropertyRange(a:correlates a:Feature) ObjectMinCardinality(2 a:correlates a:Feature)
Specific	Correlates relator has only one CorrelationCharacteristic, which indicates the correlation value	SubClassOf(ObjectExactlyCardinality(1 a:inheresIn a:CorrelationCharacteristic) a:Correlates)
Specific	PreprocessData are Data	SubClassOf(a:PreprocessData a:Data)
Specific	PreprocessData are Outputs	SubClassOf(a:PreprocessData a:Output)
Specific	TestData are PreprocessedData	SubClassOf(a:TestData a:PreprocessedData)
Specific	TrainData are PreprocessedData	SubClassOf(a:TrainData a:PreprocessedData)
Specific	TestData are disjoint with TrainData	DisjointClasses(a:TestData a:TrainData)
Specific	PreprocessedData are created in a Preprocess operation	SubClassOf(ObjectExactlyCardinality(1 a:wasCreatedIn a:ProcessedData) a:Preprocess)

Specific	PreprocessedData are disjoint with InputData	DisjointClasses(a:PreprocessedData a:InputData)
Specific	PreprocessImplementation is an Implementation	SubClassOf(a:PreprocessImplementation a:Implementation)
Specific	MLImplementation is an Implementation	SubClassOf(a:MLImplementation a:Implementation)
Specific	PredictImplementation is an Implementation	SubClassOf(a:PredictImplementation a:Implementation)
Specific	EvaluateModelImplementation is an Implementation	SubClassOf(a:EvaluateModelImplementation a:Implementation)
Specific	Implementations are disjoint with each other	DisjointClasses(a:EvaluateExplanationImplementation a:EvaluateModelImplementation a:ExplainImplementation a:PredictImplementation a:PreprocessImplementation a:MLImplementation)
Explanation	Explain is executed after Train operation	SubClassOf(ObjectExactlyCardinality(1 a:historicallyDependsOn a:ExplainModel) a:Train)
Explanation	Explain is an operation	SubClassOf(a:Explain a:Operation)
Explanation	EvaluateExplanation is an operation	SubClassOf(a:EvaluateExplanation a:Operation)
Explanation	Explain receives Results as inputs	SubClassOf(ObjectMinCardinality(1 ObjectInverseOf(a:participatedIn) a:Explain) a:Result)
Explanation	Explain executes ExplainImplementation	SubClassOf(ObjectMinCardinality(1 ObjectInverseOf(a:ExecutedBy) a:Explain) a:ExplainImplementation)
Explanation	Explain operation generates Explanation	SubClassOf(ObjectExactlyCardinality(1 a:wasCreatedIn a:Explanation) a:Explain)
Explanation	Explanation can be a ModelExplanation or a ResultExplanation	SubClassOf(a:ModelExplanation a:Explanation) SubClassOf(a:ResultExplanation a:Explanation)
Explanation	An explanation cannot be a ModelExplanation and a ResultExplanation at the same time (disjoint)	DisjointClasses(a:ModelExplanation a:ResultExplanation)
Explanation	ModelExplanation explains MLModel	SubclassOf(a:ExplainsModel a:relator) SubObjectPropertyOf(a:explainsModelInvolvesModel a:mediates) ObjectPropertyDomain(a:explainsModelInvolvesModel a:ExplainsModel) ObjectPropertyRange(a:explainsModelInvolvesModel a:MLModel) SubObjectPropertyOf(a:explainsModelInvolvesModelExplanation a:mediates) ObjectPropertyDomain(a:explainsModelInvolvesModelExplanation a:ExplainsModel) ObjectPropertyRange(a:explainsModelInvolvesModelExplanation a:ModelExplanation)
Explanation	Explanation has one or more ExplanationComponents	SubClassOf(ObjectMinCardinality(1 ObjectInverseOf(a:IsComponentOf) a:Explanation) a:ExplanationComponent)

Explanation	One ExplanationComponent can explain one ResultInstance	SubclassOf(a:ExplainsResultInstance a:relator) SubObjectPropertyOf(a:explainsInvolvesResultInstance a:mediates) ObjectPropertyDomain(a:explainsInvolvesResultInstance a:ExplainsResultInstance) ObjectPropertyRange(a:explainsInvolvesResultInstance a:ResultInstance) SubObjectPropertyOf(a:explainsInvolvesExplanationComponent a:mediates) ObjectPropertyDomain(a:explainsInvolvesExplanationComponent a:ExplainsResultInstance) ObjectPropertyRange(a:explainsInvolvesExplanationComponent a:ExplanationComponent)
Explanation	EvaluateExplanation operation receives Explanation	SubClassOf(ObjectMinCardinality(1 ObjectInverseOf(a:participatedIn) a:EvaluateExplanation) a:Explanation)
Explanation	EvaluateExplanation executes EvaluateExplanationImplementation	SubClassOf(ObjectMinCardinality(1 ObjectInverseOf(a:ExecutedBy) a:EvaluateExplanation) a:EvaluateExplanationImplementation)
Explanation	EvaluateExplanation generates ExplanationEvaluation	SubClassOf(ObjectExactlyCardinality(1 a:wasCreatedIn a:ExplanationEvaluation) a:EvaluateExplanation)
Explanation	ExplanationEvaluation is compound of one or more EvaluateExplanationComponent	SubClassOf(ObjectMinCardinality(1 ObjectInverseOf(a:IsComponentOf) a:ExplanationEvaluation) a:EvaluateExplanationComponent)
Explanation	Each ExplanationComponentEvaluation has exactly one characteristic that contains the value of the evaluation	SubClassOf(ObjectExactlyCardinality(1 ObjectInverseOf(a:inheresIn) a:ExplanationComponentEvaluation) a:ExplanationComponentEvaluationCharacteristic)
Explanation	EvaluateExplanationComponent evaluates one ExplanationComponent	SubclassOf(a:EvaluatesExplanationComponent a:relator) SubObjectPropertyOf(a:evaluatesInvolvesExplanationComponent a:mediates) ObjectPropertyDomain(a:evaluatesInvolvesExplanationComponent a:EvaluatesExplanationComponent) ObjectPropertyRange(a:evaluatesInvolvesExplanationComponent a:ExplanationComponent) SubObjectPropertyOf(a:evaluatesInvolvesExplanationComponentEvaluation a:mediates) ObjectPropertyDomain(a:evaluatesInvolvesExplanationComponentEvaluation a:EvaluatesExplanationComponent) ObjectPropertyRange(a:evaluatesInvolvesExplanationComponentEvaluation a:ExplanationComponentEvaluation)
Explanation	ExplainableAlgorithm is an Algorithm	SubClassOf(a:ExplainableAlgorithm a:Algorithm)

References

[1] A. Seeliger, M. Pfaff and H. Krcmar, "Semantic Web Technologies for Explainable Machine Learning Models: A Literature Review," *PROFILES/SEMEX@ISWC,* 2019.

[2] D. Weld and G. Bansal, "The Challenge of Crafting Intelligible Intelligence," *Commun. ACM,* vol. 62, no. 6, pp. 70-79, 2019.

[3] M. Ribeiro, S. Singh and C. Guestrin, "Why Should I Trust You?: Explaining the Predictions of Any Classifier," *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16),* vol. 22, p. 1135–1144, 2016.

[4] V. Dignum, Responsible Artificial Intelligence: How to Develop and Use AI in a Responsible Way, Springer, 2019.

[5] A. Mikolajczyk, M. Grochowski and A. Kwasigroch, "Towards explainable classifiers using the counterfactual approach - global explanations for discovering bias in data," *CoRR,* vol. abs/2005.02269, 2020.

[6] G. Jacobusse and C. Veenman, "On Selection Bias with Imbalanced Classes," in *Calders T., Ceci M., Malerba D. (eds) Discovery Science. DS 2016. Lecture Notes in Computer Science, vol 9956.* , Springer, 2016.

[7] H. Simon, "Spurious Correlation: A Causal Interpretation," *Journal of the American Statistical Association,* vol. 49, pp. 467-479, 1954.

[8] A. Chander and R. Srinivasan, "Creation of User Friendly Datasets: Insights from a Case Study concerning Explanations of Loan Denials," *CoRR,* vol. abs/1906.04643, 2019.

[9] D. Esteves, A. Ławrynowicz, P. Panov, L. Soldatova, T. Soru and J. Vanschoren, "ML Schema Core Specification," 17 October 2016. [Online]. Available: http://ml-schema.github.io/documentation/ML%20Schema.html. [Accessed 01 February 2021].

[10] R. A. Falbo, "SABiO: Systematic approach for building ontologies," 2014.

[11] W3C, "Semantic Web," 2020. [Online]. Available: https://www.w3.org/standards/semanticweb/#:~:text=The%20term%20%E2%80%9CSemantic%20Web%E2%80%9D%20refers,SPARQL%2C%20OWL%2C%20and%20SKOS.. [Accessed 2020].

[12] GraphDB, "GraphDB," [Online]. Available: https://graphdb.ontotext.com/documentation/standard/index.html.

[13] A. Dörthe, "Notation3 as the Unifying Logic for the Semantic Web," 2019.

[14] J. Domingue, D. Fensel and J. Hendler, Handbook of Semantic Web Technology, Springer, 2011.

[15] W3C, "OWL 2 Web Ontology Language Document Overview (Second Edition)," 11 12 2012. [Online]. Available: https://www.w3.org/TR/owl2-overview/#Relationship_to_OWL_1. [Accessed 27 01 2021].

[16] R. Studer, S. Grimm and A. Abecker, Semantic Web Services: Concepts, Technologies and Applications, Springer, 2007.

[17] L. Ehrlinger and W. Wöß, "Towards a Definition of Knowledge Graphs," *International Conference on Semantic Systems - SEMANTiCS,* 2016.

[18] R. Akerkar and P. Sajja, Knowledge-Based Systems, Jones & Bartlett Learning, 2010.

[19] D. Fensel, U. Simsek, K. Angele, E. Huaman, E. Kärle, O. Panasiuk, I. Toma, J. Umbrich and A. Wahler, Knowledge Graphs: Methodology, Tools and Selected Use Cases, Springer, 2020.

[20] M. Sarker, N. Xie, D. Doran, M. Raymer and P. Hitzler, "Explaining Trained Neural Networks with Semantic Web Technologies: First Steps," *ArXiv, abs/1710.04324,* 2017.

[21] C. Molnar, Interpretable Machine Learning: A Guide for Making Black Box Models Explainable., 2021.

[22] E. Alpaydin, Introduction to Machine Learning, The MIT Press, 2010.

[23] A. Singh, N. Thakur and A. Sharma, "A review of supervised machine learning algorithms," *3rd International Conference on Computing for Sustainable Global Development (INDIACom),* pp. 1310-1315, 2016.

[24] A. Ng, J. Ngiam, C. Foo, Y. Mai, C. Suen, A. Coates, A. Maas, A. Hannun, B. Huval, T. Wang and S. Tandon, "Deep Learning Tutorial," UFLDL Stanford, [Online]. Available: http://ufldl.stanford.edu/tutorial/. [Accessed 2020].

[25] A. Adadi and M. Berrada, "Peeking Inside the Black-Box: A Survey on Explainable Artificial Intelligence (XAI)," *IEEE Access 6,* pp. 52138-52160, 2018.

[26] F. Keil, L. Rozenblit and C. Mills, "What lies beneath? Understanding the limits of understanding," *Thinking and seeing: Visual metacognition in adults and children,* 2004.

[27] European Comission, "White Paper On Artificial Intelligence - A European approach to excellence and trust.," 2020.

[28] D. Gunning and D. W. Aha, "DARPA's Explainable Artificial Intelligence Program," *AI MAGAZINE,* vol. 40, pp. 44-58, 2019.

[29] R. Hoffman, S. Mueller, G. Klein and J. Litman, "Metrics for Explainable AI: Challenges and Prospects," *arXiv:1812.04608v2,* 2018.

[30] A. Arrieta , N. Díaz-Rodríguez, J. Ser, A. Bennetot , S. Tabik , A. Barbado, S. García , S. Gil-López , D. Molina , R. Benjamins , R. Chatila and F. Herrera, "Explainable Artificial Intelligence (XAI): Concepts, Taxonomies, Opportunities and Challenges toward Responsible AI," *ArXiv abs/1910.10045,* 2020.

[31] L. Longo, R. Goebel , F. Lecue , P. Kieseberg and A. Holzinger , "Explainable Artificial Intelligence: Concepts, Applications, Research Challenges and Visions," *olzinger A., Kieseberg P., Tjoa A., Weippl E. (eds) Machine Learning and Knowledge Extraction. CD-MAKE 2020. Lecture Notes in Computer Science,* vol. 12279, 2020.

[32] C. Musto, F. Narducci, P. Lops, M. Gemmis and G. Semeraro, "ExpLOD: A Framework for Explaining Recommendations based on the Linked Open Data Cloud," *Proceedings of the 10th ACM Conference on Recommender Systems (RecSys '16). Association for Computing Machinery,* p. 151–154, 2016.

[33] D. Martens, J. Huysmans, R. Setiono and J. Vanthienen, "Rule Extraction from Support Vector Machines: An Overview of Issues and Application in Credit Scoring," in *Studies in Computational Intelligence*, vol. 80, Springer, 2008.

[34] W. W. Cohen, "Fast effective rule induction," in *Proceedings of the Twelfth International Conference on Machine Learning*, Tahoe City, 1995.

[35] T. Pedersen , S. V. Pakhomov , S. Patwardhan and C. G. Chute , "Measures of semantic similarity and relatedness in the biomedical domain," *J Biomed Inform.,* 2007.

[36] A. Bühmann, J. Lehmann and P. Westphal, "DL-Learner - A Framework for Inductive Learning on the Semantic Web," *eb Semantics: Science, Services and Agents on the World Wide Web,* 2016.

[37] N. F. Noy and D. L. McGuinness, "Ontology Development 101: A Guide to Creating Your First Ontology," 2001.

[38] M. Fernández, A. Gómez-Pérez and N. Juristo, "METHONTOLOGY: from ontological art towards ontological engineering," *AAAI Technical Report,* 1997.

[39] J. Braga, J. L. R. Dias and F. Regateiro, "A Machine Learning Ontology," 2020.

[40] C. M. Keet, A. Ławrynowicz, C. d'Amato, A. Kalousis, P. Nguyen, R. Palma, R. Stevens and M. Hilario, "The Data Mining OPtimization Ontology," *Journal of Web Semantics,* vol. 32, pp. 43-53, 2015.

[41] J. Vanschoren and L. Soldatova, "Exposé: An ontology for data mining experiments," *Third Generation Data Mining Workshop at ECML PKDD 2010,* 2010.

[42] P. Panov, S. Džeroski and L. Soldatova, "OntoDM: An Ontology of Data Mining," *2008 IEEE International Conference on Data Mining Workshops,* pp. 752-760, 2008.

[43] Open Machine Learning Community, "Open ML," [Online]. Available: https://www.openml.org/. [Accessed 20 February 2021].

[44] M. Keet, "Foundational Ontologies," 7 July 2020. [Online]. Available: https://eng.libretexts.org/@go/page/6436. [Accessed 8 April 2021].

[45] G. Guizzardi and G. Wagner, "A Unified Foundational Ontology and some Applications of it in Business Modeling," in *CAiSE'04 Workshops in connection with The 16th Conference on Advanced Information Systems Engineering, Knowledge and Model Driven Information Systems Engineering for Networked Organisations*, Riga, 2004.

[46] L. Schneider, "How to Build a Foundational Ontology," in *KI 2003: Advances in Artificial Intelligence. KI 2003. Lecture Notes in Computer Science*, vol. 2821, Springer, 2003.

[47] G. Guizzardi, G. Wagner, R. de Almeida Falbo, R. Guizzardi and J. Almeida, "Towards Ontological Foundations for the Conceptual Modeling of Events," in *Conceptual Modeling*, Springer, 2013.

[48] J. P. A. Almeida, R. A. Falbo, G. Guizzardi and T. P. Sales, "gUFO: A Lightweight Implementation of the Unified Foundational Ontology (UFO)," [Online]. Available: http://purl.org/nemo/doc/gufo. [Accessed 10 April 2021].

[49] OntoUML Community, "OntoUML Specification," [Online]. Available: https://ontouml.readthedocs.io/en/latest/intro/ontouml.html.

[50] J. Guerson, T. P. Sales, G. Guizzardi and J. P. A. Almeida, "OntoUML Lightweight Editor," in *19th IEEE Enterprise Computing Conference* , 2015.

[51] C. M. Fonseca, T. P. Sales, L. Bassetti and V. Viola, "OntoUML Plugin for Visual Paradigm," May 2021. [Online]. Available: https://github.com/OntoUML/ontouml-vp-plugin. [Accessed 12 May 2021].

[52] Pedregosa et al., "Scikit-learn: Machine Learning in Python," *JMLR 12,* pp. 2825-2830, 2011.

[53] S. Lalmuanawma, J. Hussain and L. Chhakchhuak, "Applications of machine learning and artificial intelligence for COVID-19 (SARS-CoV-2) pandemic: A review," *Chaos Solitons Fractals,* 2020.

[54] L. J. Muhammad, E. A. Algehyne, S. Usman, A. Ahmad, C. Chakraborty and I. A. Mohammed , "Supervised Machine Learning Models for Prediction of COVID-19 Infection using Epidemiology Dataset," *SN Comput Sci.,* 2021.

[55] S. Wollenstein-Betech, C. G. Cassandras and I. C. Paschalidis, "Personalized Predictive Models for Symptomatic COVID-19 Patients Using Basic Preconditions," *International Journal of Medical Informatics,* vol. 142, 2020.

[56] Secretaría de Salud, "Datos Abiertos Dirección General de Epidemiología," [Online]. Available: https://www.gob.mx/salud/documentos/datos-abiertos-152127. [Accessed 9 March 2021].

[57] M. R. Franklin, "Kaggle: Mexico COVID-19 clinical data," 6 May 2020. [Online]. Available: https ://www.kaggle.com/marianarfranklin/mexico-COVID19-clinical-data/metadata. [Accessed 9 March 2021].

[58] Ethical Institute, "XAI - An eXplainability toolbox for machine learning," [Online]. Available: https://github.com/EthicalML/xai. [Accessed 10 March 2021].

[59] A. Khanday, S. Rabani, Q. R. Khan, N. Rouf and M. M. U. Din , "Machine learning based approaches for detecting COVID-19 using clinical text data," *International Journal of Information Technology volume,* vol. 12, p. 731–739, 2020.

[60] C. An, H. Lim, D. Kim, J. H. Chang, Y. J. Choi and S. W. Kim, "Machine learning prediction for mortality of patients diagnosed with COVID-19: a nationwide Korean cohort study," *Scientific Reports volume,* vol. 10, 2020.

[61] D. Brinati, . A. Campagner, D. Ferrari , M. Locatelli, G. Banfi and F. Cabitza , "Detection of COVID-19 Infection from Routine Blood Exams with Machine Learning: A Feasibility Study," *J Med Syst,* 2020.

[62] J. Lu, R. Jin, E. Song, M. Alrasho, K. N. Al-Mutib and M. S. Al-Rakhami, "An Explainable System for Diagnosis and Prognosis of COVID-19," *IEEE Internet of Things Journal,* 2020.

[63] M. M. Ahamad, S. Aktar, M. Rashed-Al-Mahfuz, S. Uddin, P. Liò, H. Xu, M. A. Summers, J. M. W. Quinn and M. A. Moni , "A machine learning model to identify early stage symptoms of SARS-Cov-2 infected patients," *Expert systems with applications,* vol. 160, 2020.

[64] I. Moscovitz, "Wittgenstein," 19 May 2020. [Online]. Available: https://github.com/imoscovitz/wittgenstein. [Accessed 5 February 2021].

[65] L. C. Barcellos, J. O. Batista and J. P. A. Almeida, "UFO validation for Protégé," 23 November 2020. [Online]. Available: https://github.com/nemo-ufes/ufo-protege-plugin. [Accessed 16 April 2021].