Modeling Quality and Machine Learning Pipelines through Extended Feature Models

Giordano d'Aloisio
giordano.daloisio@graduate.univaq.it
University of L'Aquila
L'Aquila, Italy

Antinisca Di Marco
antinisca.dimarco@univaq.it
University of L'Aquila
L'Aquila, Italy

Giovanni Stilo
giovanni.stilo@univaq.it
University of L'Aquila
L'Aquila, Italy

ABSTRACT

The recently increased complexity of Machine Learning (ML) methods, led to the necessity to lighten both the research and industry development processes. ML pipelines have become an essential tool for experts of many domains, data scientists and researchers, allowing them to easily put together several ML models to cover the full analytic process starting from raw datasets. Over the years, several solutions have been proposed to automate the building of ML pipelines, most of them focused on semantic aspects and characteristics of the input dataset. However, an approach taking into account the new quality concerns needed by ML systems (like fairness, interpretability, privacy, etc.) is still missing.

In this paper, we first identify, from the literature, key quality attributes of ML systems.Further, we propose a new engineering approach for quality ML pipeline by properly extending the Feature Models meta-model. The presented approach allows to model ML pipelines, their quality requirements (on the whole pipeline and on single phases), and quality characteristics of algorithms used to implement each pipeline phase. Finally, we demonstrate the expressiveness of our model considering the classification problem.

CCS CONCEPTS

• **Software and its engineering** → **Extra-functional properties**; • **Computing methodologies** → *Machine learning*; **Model development and analysis**.

KEYWORDS

machine learning pipeline, software quality, feature models, product line architectures, model-driven

ACM Reference Format:
Giordano d'Aloisio, Antinisca Di Marco, and Giovanni Stilo. 2022. Modeling Quality and Machine Learning Pipelines through Extended Feature Models. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Machine Learning (ML) systems are increasingly becoming a used instrument, applied to all application domains and affecting our real life. Such systems can be defined as a set of one or more *ML pipelines* that take as input raw (unprocessed) data and return actionable answers to questions in the form of machine learning models [3].

Such pipelines usually require a good knowledge of the underlying ML domain to choose the best techniques and models to solve the targeted problem. For this reason, many methods have been developed in the last years to automate some phases of pipeline development [22, 28, 46]. However, these techniques are mainly focused on semantic characteristics of the input dataset, ignoring the new essential *quality properties* introduced by such ML systems, such as dataset's bias reduction, model's interpretability, and fairness improvement [7, 38, 41]. Indeed, if we consider the impact that ML applications have in our lives, it is clear how assuring that these quality properties are satisfied is of paramount importance. The importance of having *high-quality* ML systems is also highlighted by some of the 17 sustainable development goals proposed by the United Nations [42]. In particular, to accomplish goals 5 (gender equality) or 10 (reduced inequalities) on a large scale, we will possibly rely on information systems. If those information systems include some ML pipelines (for classification or recommendation problems), it will be essential to properly manage and improve properties as *Fairness* or *Interpretability*.

In this paper, we present the first step on implementing MANILA (Model bAsed developmeNt of ml pIpeLines with quAlity), a novel approach for engineering high-quality ML pipelines. First, we identify key quality attributes in ML systems by selecting the more adopted properties in the literature. Next, we discuss how ML pipelines can be modeled as Product Line Architectures [10], in which the variation points are the developed algorithms. For our aims, we use and extend the Feature Model formalism and meta-model[32]. The choice of which algorithms will be executed in the pipeline, case by case, is driven by the functional and quality requirements specified by the ML designer.

In this work, we focus on the modeling and specification of ML pipelines with involved quality attributes. The pipelines are modeled through *Quality and Feature Models*, which extend Feature Models [32] with quality properties of features. Models can also contain functional and quality requirements, specified by ML designers, that the approach will use to generate the final ML system.

The *Quality and Feature Models* are complaint to the MANILA *Quality and Feature Meta-Model* (detailed in Section 4.1), that is an extension of the Feature Meta-Model [9]. In addition, on top of the defined meta-model, we implement a graphical editor which permits: *i)* to create Quality and Feature models, representing the Product-Line Architecture of the ML pipelines, enriched by Quality aspects. This modeling is in charge of ML experts; *ii)* to specify

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

functional and quality requirements that the final ML system must satisfy. This modeling task is in charge of ML designers.

This paper is organized as follows: in section 2, we discuss related works describing first the papers that are related to identify and engineering quality attributes in ML pipelines; next we discuss other approaches that uses Feature Models to model quality of software systems or other kinds of ML pipelines. In section 3, we make an overview of the problem of Quality Assurance in ML pipelines: we first discuss the selected quality attributes and how they affect ML pipelines, next we describe MANILA, a more comprehensive approach that aims to define an innovative model-driven framework that guides ML designers in developing ML pipelines assuring quality requirements. Section 3.3 describes the modeling framework that we have developed on this work: we start by describing the Quality and Feature Meta-Model, and next we describe the implemented graphical editor to create Quality and Feature models and specify quality requirements. Section 5 is dedicated to a proof of concept of the developed modeling framework by reproducing a case study. Finally, section 6 presents some discussions, describes future work, and wraps up the paper.

2 RELATED WORK

The problem of quality assurance in machine learning systems has gained much relevance in the last years. Many articles highlight the needing of defining and formalize new standard quality attributes for machine learning systems [7, 15, 21, 38, 41, 52]. Most of the works in the literature focus either on the identification of the most relevant quality attributes for ML systems or on the formalization of them in the context of pipelines development.

Concerning the identification of quality attributes in ML systems, the authors of [34, 54] identify three main components in which quality attributes can be found: **Training Data**, **ML Models** and **ML Platforms**. The quality of **Training Data** is usually evaluated with properties such as *privacy*, *bias*, *number of missing values*, *expressiveness*. For **ML Model**, the authors mean the trained model used by the system. The quality of this component is usually evaluated by *fairness*, *explainability*, *interpretability*, *security*. Finally, the **ML Platform** is the implementation of the system, which is affected mostly by *security* and *computational complexity*. Muccini et al. identify in [41] a set of quality properties as stakeholders' constraints and highlight the needing of considering them during the *Architecture Definition* phase. The quality attributes are: *data quality*, *ethics*, *privacy*, *fairness*, *ML models' performance*, etc. Martinez-Fernàndez et al. also highlight in [38] the needing of formalizing quality properties in ML systems and to update the software quality requirements defined by ISO 25000 [30]. The most relevant properties highlighted by the authors concern: *ML safety*, *ML ethics*, and *ML explainability*.

Many solutions have been proposed to formalize and model standard quality attributes in ML systems. *CRISP_ML* is a process model proposed by Studer et al. [49], extending the more known *CRISP_DL* [37] process model to machine learning systems. They identify a set of common phases for the building of ML systems namely: *Business and Data understanding*, *Data preparation*, *Modeling*, *Evaluation*, *Deployment*, *Monitoring and Maintenance*. For each phase, the authors identify a set of functional quality properties to guarantee the quality of such systems. Similarly, the *Q4AI* consortium proposed a set of guidelines [25] for the quality assurance of ML systems for specific domains: *generative systems*, *operational data in process systems*, *voice user interface system*, *autonomous driving* and *AI OCR*. For each domain, the authors identify a set of properties and metrics to ensure quality. Concerning the modelling of quality requirements, Azimi et al. proposed a layered model for the quality assurance of machine learning systems in the context of IoT [5]. The model is made of two layers: *Source Data* and *ML Function/Model*. For the *Source Data*, a set of quality features are defined: *completeness*, *consistency*, *conformity*, *accuracy*, *integrity*, *timeliness*. Machine learning models are instead classified into *predictors*, *estimators* and *adapters* and a set of quality features are defined for each of them: *accuracy*, *correctness*, *completeness*, *effectiveness*, *optimality*. Each system is then influenced by a subset of quality characteristics based on the type of ML model and the required data. Ishikawa proposed, instead, a framework for the quality evaluation of an ML system [29]. The framework defines these components for ML applications: *dataset*, *algorithm*, *ML component* and *system*, and, for each of them, proposed an argumentation approach to assess quality. Finally, Siebert et al. [47] proposed a formal modelling definition for quality requirements in ML systems. They start from the process definition in [37] and build a meta-model for the description of quality requirements. The meta-model is made of the following classes: *Entity* (which can be defined at various levels of abstraction, such as the whole system or a specific component of the system), *Property* (also expressed at different levels of abstraction), *Evaluation* and *Measure* related to the property. Starting from this meta-model, the authors build a tree model to evaluate the quality of the different components of the system. To the best of our knowledge, this is the first attempt at formalizing the quality of ML systems using a model-driven approach.

From this analysis, we can conclude that there is a robust research motivation in formalizing and defining new quality attributes for ML systems. Many attempts have been proposed to solve these issues, and several quality properties, metrics and definitions of ML systems can now be derived from the literature. However few issues are still not fully been addressed:

(1) a mapping between quality attributes and components of an ML pipeline;
(2) a modeling of how each quality attribute may influence the development of an ML pipeline;
(3) a modeling of the influence each quality property can have on other attributes;

In this paper, we aim to solve these concerns by proposing a novel model-driven approach which will allow ML designers to model ML pipelines with quality attributes. In particular, we extend the meta-model of the Feature Models to allow the specification of *Quality and Feature Models*. The Quality and Feature model comprises features and quality properties which are related to or can be implemented by the features themselves. In addition, the presented meta-model allows the specification of functional and quality requirements by selecting from the Quality and Feature model only the quality properties that are needed and by specifying attributes related to features.

The proposed approach is similar to the one proposed by Asadi et al. in [4]. In their work, the authors present a framework for the automatic configuration of feature models based on non-functional requirements. They start from the extension of feature models to allow the definition of quality attributes associated to features. Then, they use the stakeholders requirements (defined as a set of functional and non-functional requirements with relative constraints among them) and the extended feature model to build a Hierarchical Task Network Planning problem which is finally solved to derive the final configuration that satisfies both functional and non-functional requirements. However, in their work the authors consider the traditional quality properties as defined in the ISO 25000. Instead, the new quality properties introduced by ML systems requires a different formulation in terms of modeling. For example, some quality attributes (e.g., fairness) are directly implemented by features in the model which must be selected if the implemented property is required. In our work, we consider these new quality properties introduced by ML systems and make a new modeling of them taking into account their new intrinsic characteristics. Finally, a similar approach of using Feature Models to model ML pipelines has been used by Di Sipio et al. in [16]. In their work the authors use Feature Models to model ML pipelines for Recommender Systems, however they do not consider quality attributes in their approach.

3 QUALITY ASSURANCE IN ML PIPELINES

In this section, we make an overview of Quality Assurance (QA) in ML pipelines. In particular, in section 3.1 we define *quality* in ML pipelines by selecting the quality properties that are more frequent and relevant in the literature. Next, in section 3.2 we describe a generic ML pipeline and highlight how the chosen quality properties may influence each phase of the process. Finally, in section 3.3 we propose our broaden vision on engineering ML pipelines with quality constraints leveraging on model-driven generative techniques.The modeling framework we present in this paper (detailed in Section 4) is the foundational part of our generative approach and it leverages on the . definitions given in these sections.

3.1 Considered Quality Attributes

In software engineering, a Quality Requirement is a requirement that specifies criteria that can be used to quantify or qualify the operation of a system, rather than to specify its behaviors [12].

To analyze an ML pipeline from a qualitative perspective, we have to determine the Quality Attributes (QA) that we can use to judge the operation of the pipeline, and that can influence the ML designers' decisions. To identify the QA to consider, we refer to the literature for ML systems [21, 34, 41, 52].

Figure 1 summarizes the selected QA and their mutual influence. In the figure, white circles represent the quantitative QA (attributes that can be measured using one or more metrics), while grey ones represent the qualitative QA (attributes that can not be measured with a specific metric). Arrows mean influence, for example an arrow from QA A to QA B models the influences of A to B. In some cases, as for the *Computational Complexity* and the *Prediction Correctness*, the impact is mutual; in other cases, like *Fairness* and the *Interpretability*, it is mono-directional. In addition, while

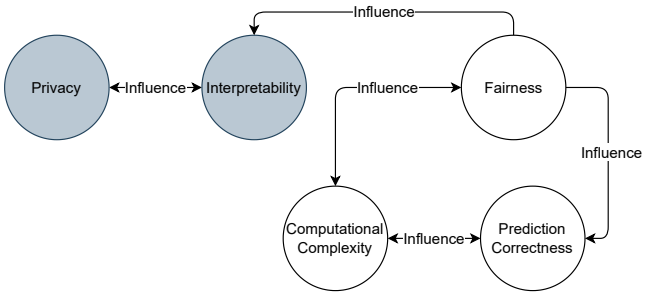

Figure 1: Quality attributes in ML pipelines and their influence

some properties can be estimated a-priori (i.e. without adding additional computational tasks to the pipeline) and associated with features (e.g., *Interpretability* or *Computational Complexity*), others (e.g., *Fairness* or *Prediction Correctness*) can be evaluated only executing the actual pipeline implementation enriched by specific computational steps that quantifies them.

In the following, we describe the selected QA in more detail.

Computational Complexity. This quantitative QA defines the computational complexity of the final ML pipeline at production time as the pair *Space Complexity (SC)* and *Time Complexity (TC)*, where the first indicates the amount of memory required by a pipeline to perform all its phases, and the second indicates the time required by the pipeline to complete the whole task [13]. This attribute can influence various pipeline stages (see section 3.2) and other QA such as *Prediction Correctness*, or *Fairness*. In fact, ML methods that are more accurate in their predictions are also more complex from a computational point of view (e.g., Neural Networks). The same holds for *Fairness*, in fact if the ML pipeline is required to be fair, then enhancing fairness methods must be included in the pipeline and this will increase the overall complexity of the process.

Prediction Correctness. This quantitative QA is used to define how good the model must be in predicting outcomes. There are different metrics in the literature, each addressing a different goal of the user to compute the prediction correctness of an ML model. Among the most common metrics, we cite *Precision*: fraction of true positives (TP) with respect to the total positive predictions [8]; *Recall*: fraction of TP to the total positive items of the dataset [8]; *F1 Score*: harmonic mean of *Precision* and *Recall* [50]; *Accuracy*: fraction of True Positives (TP) and True Negatives (TN) above the total of predictions [44]. This QA could impact all the steps of the pipeline regarding the training and testing of an ML model (see section 3.2) and can influence the *Computational Complexity* QA.

Interpretability. *Interpretability* can be defined as the ability of a system to enable user-driven explanations of how a model reaches the produced conclusion [11]. *Interpretability* is one of the quality attributes that can be estimated without executing an actual ML pipeline. Interpretability is a very strong property that can hold only for white-box approaches (such as decision trees). Instead, black-box methods (such as neural networks) requires the addition of explainability enhancing methods to have their results interpretable [36]. In this paper, we consider interpetability while we leave, for future work, the treatment of explainability for black-box learning

approaches. Interpretability can influence (i.e., it can reduce) the *Privacy* of the pipeline and can be affected by the *Fairness* QA. In fact, fairness enhancing methods change the value of (some) attributes of the dataset and this can reduce the interpretability of the model [39].

Privacy. *Privacy* can be defined as a qualitative QA allowing sensitive information of a dataset to be hidden, changing the value of some attributes [20]. Recent works have shown how privacy can affect the ML model during the deployment phase [45]. This QA directly impacts the *Interpretability* since a higher level of privacy necessarily causes less interpretability of the model.

Fairness. A ML model can be defined as *fair* if it has no prejudice or favoritism towards an individual or a group based on their inherent or acquired characteristics, identified by the so-called *sensitive variables* [39]. This quantitative QA influences the *Interpretability* (since some methods for bias mitigation require changing the label of some sensitive attributes), the *Computational Complexity* (since bias mitigation methods add an extra step to the pipeline), and the *Prediction Correctness* (since mitigating the bias typically reduces the performance of the predictions).

3.2 ML Pipelines with Quality Attributes

"ML pipelines formalize and implement processes to accelerate the development, management, deploy and reuse of ML models" [26]. Figure 2, inspired by [3, 49], reports a generic ML pipeline

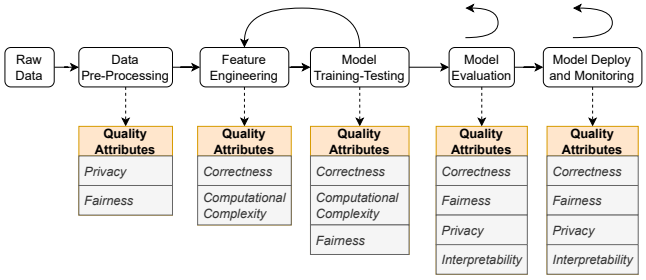

Figure 2: ML pipeline with involved quality attributes (inspired by [3, 49])

(on the top of the figure) together with the selected QA affecting the pipeline steps (on the bottom of it). We say that a QA affects a pipeline's step if, in presence of a Quality Requirement (QR) specifying a constraint on that QA, such a QR has an impact on the development of the step or, in other words, it imposes restriction in step's implementation. For instance, in the figure, *Computational Complexity* has an impact on the *Model Training-Testing* and *Feature selection* steps, while *Privacy* can affect *Model Evaluation* and *Model Deploy and Monitoring* steps.

Usually, a ML pipeline takes as input raw (unprocessed) data and, if needed, computes some pre-processing transformations on it. Next, it uses the data to train and test an ML model. This model could be further evaluated, also with human intervention, and finally deployed and continuously monitored. As shown in figure 2 the pipeline workflow is an iterative process, meaning that most of the depicted phases can roll back to previous ones. Each of the described steps are affected by at least two of the QA described in section 3.1.

The *Data Pre-Processing* step, which manipulates data, could be affected by *Privacy* and *Fairness* since they are strongly related to data [20, 39]. The *Feature Engineering* step aims at selecting the best dataset's features used to train the ML model. Since this phase is fundamental for the definition of a model able to make good predictions, its implementation is influenced by *Prediction Correctness*. In addition, the *Computational Complexity* is also involved in this phase since the selection of some features can improve or decrease the computational complexity of a model [33].

After selecting the best features, the ML model is trained and tested to verify the correctness of the implemented model. In this step, QA affecting its implementation are, besides *Prediction Correctness*, the *Computational Complexity*, and the *Fairness*. In case associated QRs are not satisfied, the ML pipeline rolls back to the *Feature Engineering* to retry the feature selection.

The last two steps, namely *Model Evaluation* and *Model Deploy and Monitoring*, are affected by the same QAs: *Correctness*, *Fairness*, *Privacy*, and *Interpretability*.

It is worth to note that the described steps are not mandatory and some of them can be skipped if not needed in the specific use case. For instance, if the dataset does not need to be manipulated before training the model, then the *Data Pre-processing* phase is skipped.

Another crucial aspect to consider when evaluating the quality of ML pipelines is that the depicted QAs are not atomic entities but they can influence each other. For example, if the system requires high fairness (e.g., legal reasons [6, 18, 35]), fairness enhancing components must be included during the *Data Pre-Processing* and the *Model Training and Testing* phases [39]. However, as explained in section 3.1 this has a negative influence on the predictions' correctness [19, 31]. So, if the system is required to have also a high *Prediction Correctness*, the pipeline development becomes more complicated and the complexity of the problem grows as the number of QAs and QRs increases.

To conclude, the identification and formalization of quality attributes and requirements, and their handling in the development of ML pipelines are mandatory and complex tasks. Our approach takes these challenges and in this paper, as first results, we present the modeling framework that enable the development of quality ML pipelines.

3.3 The MANILA Approach

As explained in section 3.2, the quality assurance of ML pipelines is a complex task that must consider numerous variables, such as the impact of quality requirements in each pipeline step and the influence among quality attributes.

The work presented in this paper is part of MANILA (Model bAsed developmeNt of ml pIpeLines with quAlity), a more comprehensive methodology that aims to define an innovative model-driven approach and relative framework that guides ML experts and designers in developing ML pipelines assuring quality requirements. Figure 3 depicts the high-level architecture of such a framework and highlights, by means of bold line, the modeling framework presented in this paper. As discussed earlier (see Figure 2), ML pipelines

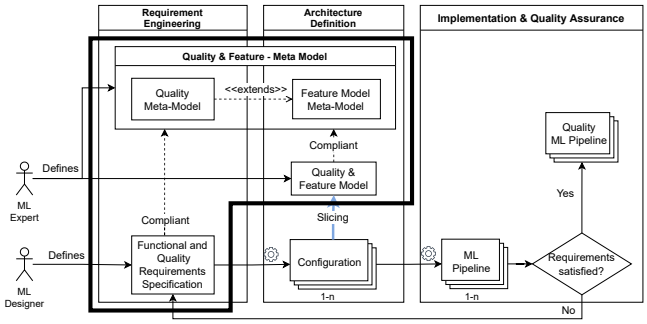

Figure 3: High-level view of MANILA

are made of typical phases [3, 49] that embed a set of standard components identified by the system's functional requirements (like the ML model suited for an ML goal such as classification) and a set of variability points. These variability points are represented by different methods that implement the functional requirement (e.g., we can have Decision Trees [14], Logistic Regression [40], KNN [23], Neural Networks [24], and other methods that realize classification task). Which one of these will be implemented in the pipeline depends on the specific quality requirements to satisfy. For instance, suppose we consider multi-class classification problems [2] and fairness quality property. In this case, variability points are represented by ML models suitable for the multi-class classification task and methods to enhance fairness, which must be included in the pipeline if we want to achieve the given quality requirement. However, while some fairness methods can be applied to multi-class datasets (e.g., *Exponentiated Gradient* [1], or *Blackbox* [43]) others can not (e.g., *Reweighing* [31]), or *DIR* [19]), hence, they have to be excluded in the pipeline's implementation.

Product-Line Architectures, specified by Feature Models, [32, 51] represent a suitable model to formalize ML pipelines with variability. Feature Models allow us to define a template for families of software products with standard features and a set of variability points that differentiate the final systems. However, they miss adequate means to specify quality attributes and requirements. In fact, they do not allow to specify thresholds, metrics, or quality properties related to features. To address this issue, in MANILA approach (see Figure 3), we propose to extend the Feature Models meta-model to enable:

(1) the creation, by the machine learning experts and designers, of an enriched feature model with associated quality attributes for each variability (like done in [4]);
(2) the specification, by the ML designer, of functional and quality requirements that the implemented pipeline (that is the one with all variability points solved) satisfies.

During the *Requirement Engineering* phase, the end-user (named ML designer in figure 3) specifies a set of functional and quality requirements compliant with the defined meta-model. These requirements are used during the *Architecture Definition* step to automatically generate, from the extended feature model provided by the machine learning expert, a set of ML pipeline configurations able to satisfy the defined functional requirements (the *Configuration* boxes in the figure). The configurations are defined by removing from the feature model all the components (and their relative specification) not suitable to meet the specified requirements or against some constraints specified in the model. The generated configurations are given as input to the *Implementation & Quality Assurance* step that aims to:

(1) generate for each configuration a python script implementing the *ML Pipeline*;
(2) verify that at least one generated pipeline satisfies the quality requirements (*Quality ML Pipeline*).

The framework returns the set of Quality ML Pipelines, satisfying quality constraints, if any, or demands the ML designer to relax quality requirements and repeat the process.

As highlighted in figure 3, the work presented in this paper implements the *Requirement Engineering* and part of the *Architecture Definition* phases. In particular, we extend the Feature Models meta-model with the possibility of associating quality properties to features and specifying functional and quality requirements. To enact the modeling, we developed on top of the extended meta-model. In addition, in section 5, we show the potentialities of MANILA modeling approach.

4 MANILA MODELING FRAMEWORK

This section describes the MANILA modeling framework which is made of two main components: *i)* the Quality and Feature meta-model, and *ii)* a graphical editor to create the quality and feature models. Using them, it is possible to specify:

- a feature model for each ML problem. Each feature model represents the ML pipeline for the specific ML problem, where the variability points are the set of alternative methods that could be used in the pipeline steps. This modeling action is done by the ML expert once. The produced models are stored in a repository for future uses of ML designers or for models updates from the experts;
- the quality characteristics, possessed by the methods modelled as variability points. This task is still in charge of the ML expert and can be specified only for QA quantifiable a-priori;
- the functional and quality requirements of the specific ML problem to solve. This modeling task is a duty of ML designer and it is repeated every time a new ML system must be implemented.

All the artifacts, models and meta-models, are available at the following link: https://github.com/giordanoDaloisio/manila-framework.

In the following, we detail the *Quality and Feature meta-model* and the editor we implement to perform modeling tasks, while an example of modeling is showed, as proof of concept, in section 5.

4.1 Quality and Feature Meta-Model

The *Quality and Feature meta-model* allows the definition of an extended Feature Model with quality properties and the specification of functional and quality requirements. The QAs identified in section 3.1 are used to specify both quality properties of features and quality requirements of the whole pipeline. Quality properties and requirements refer to QAs and show the same form, but the former models *descriptive characteristics* of variability points whereas the latter models the *prescriptive characteristic* of the whole

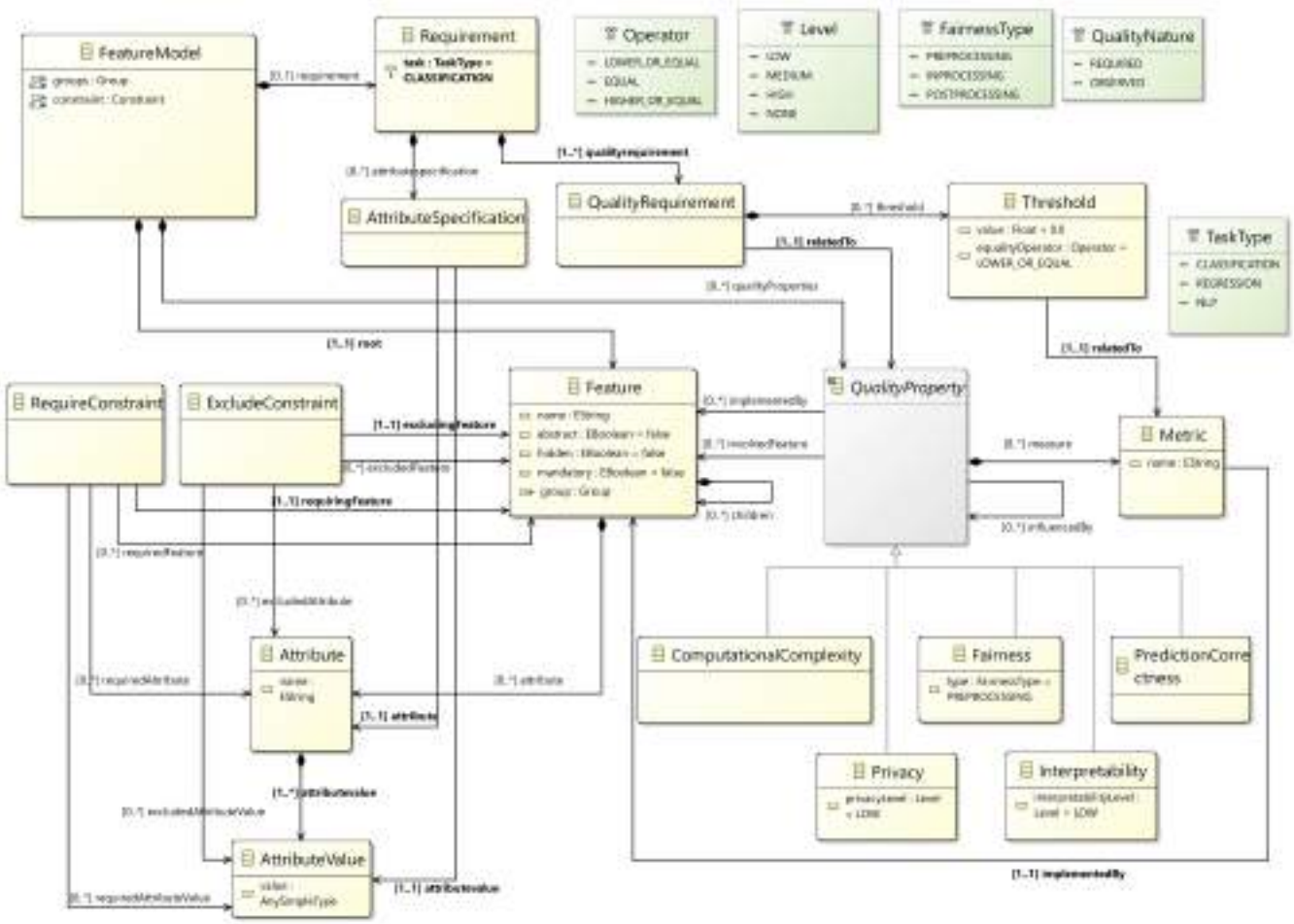

Figure 4: Quality and Feature meta-model

ML pipeline. The combination of features, showing specific quality characteristics can (or not) satisfy the quality requirements of the ML pipeline. MANILA approach aims to select a combination of features that all together implement the ML pipeline and satisfy the quality requirements.

The meta-model has been implemented using the established EMF Framework [48] and is depicted in figure 4.

As already discussed in section 3.3, Feature Models are a suitable model to formalize ML pipelines where variability points models the concrete methods that can use to implement a pipeline's step. Hence, MANILA meta-model is an extension of the meta-model proposed in [9].

The root entity of the meta-model is the *Feature Model*, which represents the model itself and is composed by a *Feature* entity, representing the whole ML pipeline, and composed itself by a set of children, i.e. (sub-)`Features`.

A *Feature* entity represents a building block of the ML pipeline. Features have a name and it can be abstract (i.e., they do not have a concrete implementation in the final system), hidden (i.e., they are not shown in the model) and mandatory (i.e., they must be selected in every configuration) [32]. If a feature does not show the abstract attribute, it means that it will be concrete. A *Feature* can be a composition of other Features. Variability points are represented by the *Feature*s as well, but at a lower level in the composition.

In the composition, Features can belong to two groups: *OrGroup* and *AltGroup*[1]. *OrGroup* entities represent an inclusive relationship, meaning that at least one child must be selected if the father is selected. *AltGroup* entities represent instead an exclusive relationship, meaning that exactly one child must be selected if the father is selected.

Finally, Features can have *Attributes*. Each *Attribute* has a name and contains a set of *Attribute Value*, predefined by the ML expert, that represent the set of possible values an *Attribute* can have.

Feature, *Attribute*, and *AttributeValue* entities can be part of a logical *Constraint*. There are two types of logic constraints: *RequireConstraint* and *ExcludeConstraint*. The first means that, if a feature involved in the RequiredConstraint is selected in a configuration,

[1]To enhance the visualization, we omitted from figure 4 these entities. However, a full picture of the meta-model is available here https://github.com/giordanoDaloisio/manila-framework/blob/main/assets/metamodel.png

then also the other Features or Attributes in the logical constraint must be considered in the final implementation. The second means instead that, if a feature under the ExcludeContraint is selected, then the other entity must not be selected in the configuration.

Note that, we extended the original Feature Meta-model to include in the Constraints also Attributes and AttributeValues. This extension can help to automatically select suitable components starting from, for example, the characteristic of the Dataset, which might have several attributes (e.g. the number of classes in the case of a classification tasks or the number of sensitive variables when dealing with fairness [39]) whose values are not compatible with the characteristics of components that hence must be excluded during the generation of the ML pipeline configuration.

A Feature Model can contain a set of *QualityProperty*. Following the definition of quality we made in section 3.1, each quality property refers to one of the attributes we defined earlier: *Computational Complexity*, *Privacy*, *Fairness*, *Interpretability*, and *Prediction Correctness*. Measurable quality properties are composed by one or more *Metric* entities, each implemented by a feature.

Quality property itself can be *implemented by* one or more Features and it can involve other Features. The distinction between *implemented by* and *involved feature* is needed since some features directly implement some quality properties (e.g., fairness is straight provided by some methods able to mitigate the bias of an ML method), otherwise others are an intrinsic quality property without requiring an extra computational step (e.g., the computational complexity is an intrinsic feature of ML algorithms). Finally, quality properties can also be influenced by each other (modelled as *influenced by* relation).

Finally, the functional and quality requirements are represented by the *Requirement* entity. In particular, a Feature Model can include a *Requirement* related to a specific ML *task* (classification, regression, natural language processing, and so on). Each *Requirement* is composed by a set of *Attribute Specification*, modeling the functional requirement, and a set of *Quality Requirement*, modeling the quality requirements on the ML *task*. Each *Attribute Specification* is related to an *Attribute* and an *Attribute Value*. Each *Quality Requirement* is associated with a *Quality Property* previously defined in the model and can contain one or more *Thresholds* related to a specific *Metric*.

4.2 Graphical Editor

After defining the meta-model, we implemented a graphical editor that allows users to easily model ML pipelines and specify functional and quality requirements. The editor has been implemented using *Sirius*[53], a well-known tool for building graphical editors based on the EMF framework.

Figure 5 shows an example of a model complaint to the *Quality and Feature Meta-model* built with our editor. Note that, in order to keep the model simpler and focus only on its semantic, we named each feature with abstract label such as *Child* that must be interpreted as components. In the figure, *Child2.1* and *Child2.2* represent specific ML pipeline components involved in the *Interpretability* quality property, whereas *Child1* and his children model additional computational components implementing fairness. A real example of modeling can be seen in section 5.

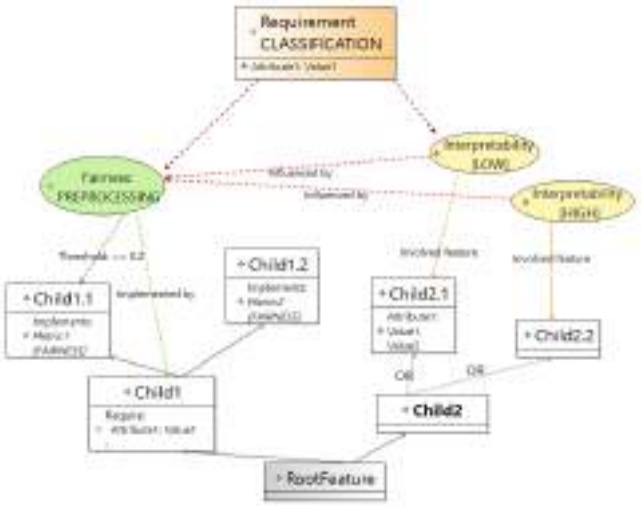

Figure 5: Example of Quality and Feature model with a given requirement

In the devised editor, features are represented by boxes: white boxes model concrete features, while grey boxes represent abstract features [51]. Mandatory features are instead highlighted with a boldface name. Attributes of a feature are defined inside the box, below its name, with a label following the pattern: `<Attribute Name>: <Attribute Values>`. Concerning the example in figure 5, *Root Feature* is an abstract feature while all the others are concrete ones. Furthermore, *Child2* is mandatory, and *Child2.1* has an attribute named *Attribute1* with two possible values: *Value1* and *Value2*.

Following the original definition of Feature Models, each feature can have one or more children, and children can belong to logical *OrGroup* or *AltGroup* [32]. In the graphical editor, this information is represented using dotted edges labeled with `OR` or `ALT` keyword. If a child feature does not belong to any group, the edge is not labelled.

In addition, as already said in section 4.1, features can also require or exclude other features or attributes. This constraint is expressed using, inside the box of the requiring/excluding feature, a specification following the form: `<Require|Exclude>: <Feature Name|Attribute Name|Attribute Name: Attribute Values>`. In the example shown in figure 5, *Child2.1* and *Child2.2* belong to an *OR* group, while *Child1*, if selected, requires *Attribute1* equal to *Value1*.

QAs are represented with ellipses distinguished by color according to the property type (i.e., fairness is green, interpretability is yellow, and so on).

Quality Properties (ellipses) are linked to the feature that implement them using a green edge. The editor use a yellow edge to link Quality Properties (ellipses) to the involved features. To keep the model clearer, we have decided not to represent metrics but just to label the features implementing a given metric with a label following the pattern: `Implements: <Metric Name> [<Quality Property>]`.

The influence among quality properties is modelled by means of a red edge (*influenced by*) that starts from the influenced quality (ellipse) and points to the influencing property (ellipse). In the example of figure 5, we have two quality properties: fairness (of *PRE-PROCESSING* type [39]) and interpretability. Fairness requires extra computational task is implemented by the feature *Child1*, while interpretability, being an a-priori quality does not require an extra feature and involves *Child2.1* and *Child2.2*. In particular, *Child2.1* is involved with a low level of interpretability, while *Child2.2* is involved with a high level of interpretability. Interpretability could be influenced by fairness (as discussed in section 3.1). This information is modelled by means two red edges starting from each *Interpretability* node and pointing to the *Fairness* node. Finally, *Child1.1* and *Child1.2* are two features implementing two metrics for fairness, namely *Metric1* and *Metric2*, respectively.

Requirements are represented with orange boxes. The name of the box follows this pattern: `Requirement <ML Task>`. In the Requirement box, there is a list of possible attribute specifications. Each specification follows the pattern: `<Attribute Name>: <Attribute Value>`. Quality requirements are represented with red edges connecting the requirement to the related quality property. Finally, thresholds are represented with brown edges that connect the quality property with a feature implementing the relative metric. The value of the threshold is reported with a label on the edge. Concerning figure 5, there is a requirement for the classification task, setting *Attribute1* equal to *Value1* and requiring both fairness and interpretability quality properties. In particular, there is a requirement for a low level of interpretability and a fairness value equal to or lower than 0.2 under *Metric1*.

5 PROOF OF CONCEPT

This section describes the proof of concept we conducted in our work. In particular, we want to ask the following research questions:

RQ1. Are our meta-model and editor able to properly represent an actual ML pipeline specification with involved quality requirements?
RQ2. Does the built model enable the generation of ML pipeline configurations?

To answer these questions, we specify, using the implemented editor, a Quality and Feature model that reproduces a ML pipeline for multi-class classification problems, subject to fairness and prediction correctness quality constraints. In particular, in this example we focus on the Pre-Processing fairness, which consists on applying fairness enhancing methods to the dataset before using it as an input to train the classifier [39]. Hence, recalling the figure 2, the pipeline steps involved in this example are *Data Pre-Processing* and *Model Training-Testing*. We do not represent the *Feature Engineering* phase to keep model clearer and more readable.

Concerning **RQ1**, figure 6 shows the implemented model to represent the multi-class classification pipeline[2]. The root feature is the *ML Pipeline*, which is an abstract mandatory feature. Describing its children, going from left to right, the first is the *Dataset* feature, which has two attributes regarding the type of label (binary or multiclass) and the number of sensitive variables (i.e., variables affected by bias). Following, there are the features related to fairness. In particular, these features are distinguished between methods to implement fairness and metrics to measure it. Concerning methods, we have included the most common pre-processing methods to improve fairness: *Reweighing* [31], *DIR* [19], *EG* [1], and *Blackbox* [43]. Since *Reweighing* and *DIR* can only be applied to datasets with a binary label and a single sensitive variable, we added a *Require* logical constraint requiring the *Label* attribute to be equal to *binary* and the *Number of sensitive variables* attribute to be equal to *single*. In addition, the abstract *Fairness* feature requires, if selected, to specify the *Number of sensitive variables* attribute.

The *Fainess Metrics* entity is in relationship with features implementing metrics usable to measure fairness, which are: *Statistical Parity (SP)* [35], *Disparate Impact (DI)* [18], and *Equalized Odds (EO)* [27]. These features belong to an *OR* group, meaning that at least one metric must be selected if the father is selected.

Features implementing the metrics for the *Prediction Correctness* are *Precision* [8], *Recall* [8], *Zero One Loss* [17], and *Accuracy* [44]. These features are children of an abstract father feature named *Prediction correctness* and belong to an *OR* group.

Finally, the model contains features representing ML classifiers. In this model, we have included the most used ML methods for classification: *KNN* [23], *Logistic Regression* [40], *Neural Networks* [24], and *Decision Trees* [14]. These features are children of an abstract *Classifier* feature, which is mandatory, and they are members of an *ALT* group, meaning that only one of them can be selected inside a configuration. The *Classifier* is involved in the *Prediction Correctness* quality property, meaning that all the children of this feature will be also involved in this quality attribute. Fairness and prediction correctness can influence each other, so there are two (red) edges indicating their mutual influence.

The presented model specifies the requirements of the needed ML system that must execute multi-class classification on datasets with more than one sensitive variable. The ML system must be fair and have high prediction correctness:

(1) the Prediction correctness must consider *Accuracy* metric, whose value must be higher or equal to 0.85, and *Zero-One Loss* metric that must be lower or equal to 0.2;
(2) Fairness must use *Disparate Impact* metrics, whose value must be higher or equal to 0.8, and *Statistical Parity* metrics that must be lower or equal to 0.2.

These functional and quality requirements are modeled through a *Requirement* entity, which specifies in the *Dataset* feature, the value *multi-class* for the attribute *Label* and the value *multiple* for the attribute *Number of sensitive variables*.

The *Requirement* is connected to the two involved quality properties: *Prediction correctness* and *Fairness*. The two involved properties are then related to the features implementing the metrics used in the thresholds. In particular, *Prediction Correctness* is connected to *Accuracy* and *Zero One Loss* through edges labeled with the threshold's value. Same for *Fairness*, which is connected to *SP* and *DI*.

The model also specifies the *Interpretability* property that is particular important for classification problems even if it is not

[2]The same picture with a higher resolution is available here https://github.com/giordanoDaloisio/manila-framework/blob/main/assets/quality-features-model.jpg

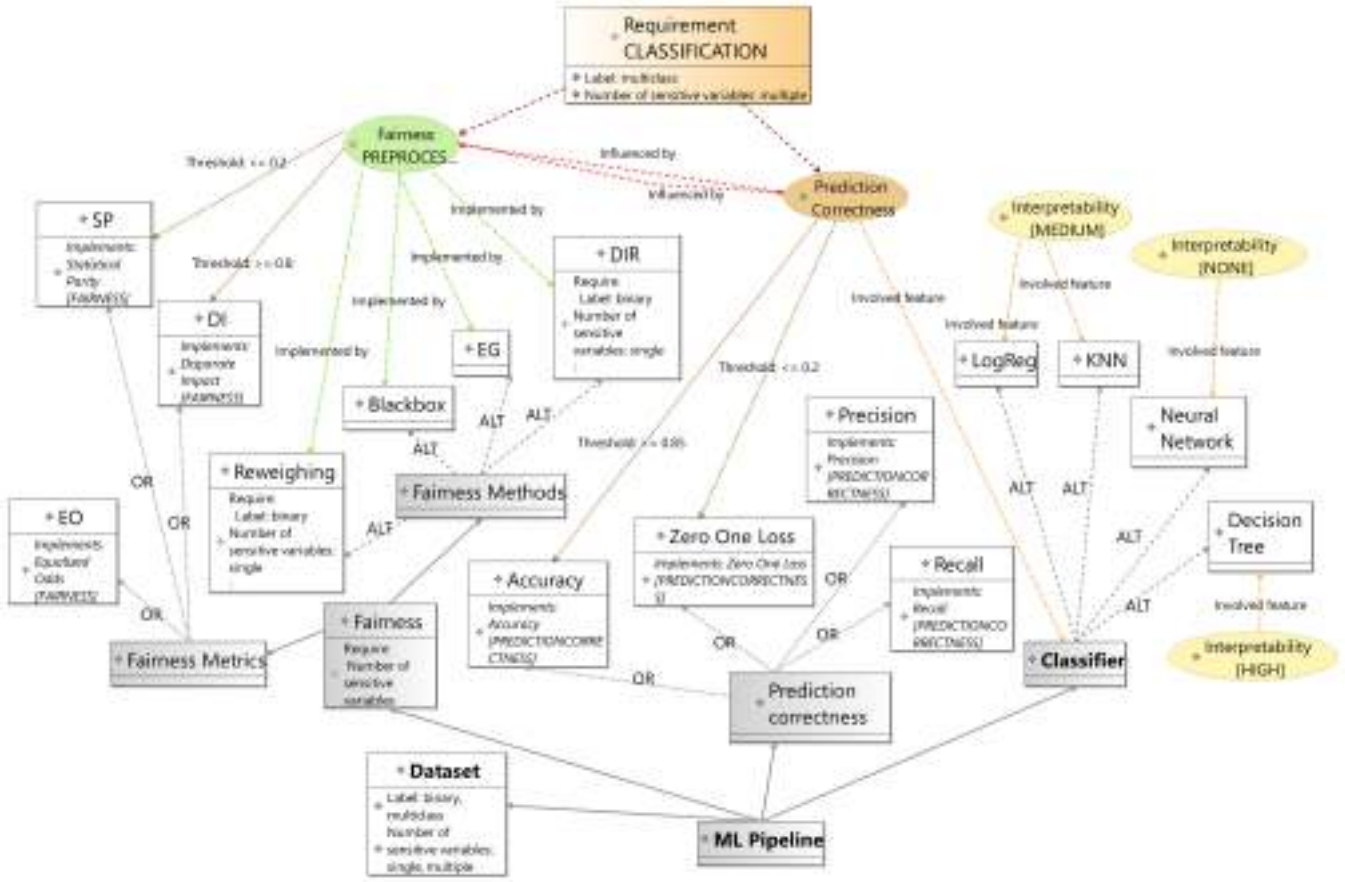

Figure 6: Implemented Quality and Feature model for multi-class classification task with given requirements

targeted by the showed requirement. But, in general, another ML system specification could ask for it.

> **RQ1**. The presented meta-model is able to represent an ML pipeline with involved quality properties. In particular, relying on the concept of feature models, we can represent any step of an ML pipeline using features and constraints. Each feature can be *involved in* or *can implement* one or more quality properties. These properties, along with features' attributes, are then used to specify a requirement. The requirement, along with the constraints defined in the model, can be used to generate a set of ML pipeline configurations by cutting off the features that are not needed and those against a particular constraint.

Concerning **RQ2**, the requirement of our example defines a multi-class dataset with more than one sensitive variable. This specification automatically excludes *Reweighing* and *DIR* from the features to consider to improve fairness since they require a dataset with a binary label and one sensitive variable. In addition, *Precision*, *Recall*, and *Equalized Odds* are metrics not considered in the given quality specification, so their implementations can be omitted from the set of configurations. Finally, concerning the classifier methods, they are not involved in any specification or constraint, so all of them must be considered. However, since they belong to an *ALT* group, they must be included in different configurations. Table 1 summarises the set of valid configurations that can be defined from this requirement. In particular, each row of the table depicts a valid ML pipeline configuration in which only the required features are selected. It can be noted how some features (like the dataset or the fairness and correctness metrics) are always selected. Other features, like the different ML classifiers or the fairness enhancing methods varies between configurations. Finally, there are features that are not present in any configuration, because they are against the functional and quality requirements defined above.

> **RQ2**. Underlying logical conditions can be derived from the requirement and constraints specified by the designed model. The logical condition automatically defines a set of pipeline configurations by cutting off the features that are not needed.

Table 1: ML pipeline configurations derived from the requirement specification

		Classifier				Prediction Correctness Metrics				Fairness Methods				Fairness Metrics		
	Dataset	KNN	LogReg	Neural Net	Decision Tree	Zero One Loss	Accuracy	Precision	Recall	Blackbox	EG	Reweighing	DIR	DI	SP	EO
1	x	x				x	x			x				x	x	
2	x	x				x	x				x			x	x	
3	x		x			x	x			x				x	x	
4	x		x			x	x				x			x	x	
5	x			x		x	x			x				x	x	
6	x			x		x	x				x			x	x	
7	x				x	x	x			x				x	x	
8	x				x	x	x				x			x	x	

6 CONCLUSION AND FUTURE WORK

In this paper, we have presented a novel approach to model and specify ML pipelines with quality requirements. First, we have identified the most influential quality properties in ML pipelines by selecting the quality attributes that are most cited in the literature. We have shown how these quality properties may impact the pipeline's steps. Next, we have presented MANILA, a novel model-driven approach that will guide ML designers in developing ML pipelines assuring quality requirements. Finally, we described the modeling framework implemented in our work and proven its usefulness by reproducing a real use case scenario.

In particular, in order to model ML pipelines with quality attributes, we relied on the concept of Feature Models. We extended the Feature Models meta-model to create a Quality and Features meta-model, which allows associating quality attributes to features and specifying functional and quality requirements. We have demonstrated the expressiveness of our meta-model by reproducing an ML pipeline with quality attributes. In addition, we implemented a graphical editor which allows the creation of Quality and Feature models and the specification of the requirements.

In the future, we plan first to conduct a user evaluation of our graphical editor to evaluate its usability and to integrate other quality properties (for instance, *Explainability* [36]) in our meta-model. Next, we will continue the development of MANILA framework by first transforming the derived ML pipeline configurations into actual implementations and further by evaluating the real quality of the implemented pipelines through their execution in a test environment. This last step will allow the ML designer to select the most promising ML system among the ones that satisfy all the requirements.

Acknowledgments. This work is partially supported by Territori Aperti a project funded by Fondo Territori Lavoro e Conoscenza CGIL CISL UIL and by SoBigData-PlusPlus H2020-INFRAIA-2019-1 EU project, contract number 871042.

REFERENCES

[1] Alekh Agarwal, Alina Beygelzimer, Miroslav Dudik, John Langford, and Hanna Wallach. 2018. A Reductions Approach to Fair Classification. In *Proceedings of the 35th International Conference on Machine Learning*. PMLR, 60–69. https://proceedings.mlr.press/v80/agarwal18a.html ISSN: 2640-3498.

[2] Mohamed Aly. 2005. Survey on multiclass classification methods. *Neural Netw* 19, 1-9 (2005), 2.

[3] Saleema Amershi, Andrew Begel, Christian Bird, Robert DeLine, Harald Gall, Ece Kamar, Nachiappan Nagappan, Besmira Nushi, and Thomas Zimmermann. 2019. Software engineering for machine learning: A case study. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 291–300.

[4] Mohsen Asadi, Samaneh Soltani, Dragan Gasevic, Marek Hatala, and Ebrahim Bagheri. 2014. Toward automated feature model configuration with optimizing non-functional requirements. *Information and Software Technology* 56, 9 (Sept. 2014), 1144–1165. https://doi.org/10.1016/j.infsof.2014.03.005

[5] Shelernaz Azimi and Claus Pahl. 2020. A Layered Quality Framework for Machine Learning-driven Data and Information Models.. In *ICEIS (1)*. 579–587.

[6] Richard Berk, Hoda Heidari, Shahin Jabbari, Michael Kearns, and Aaron Roth. 2018. Fairness in Criminal Justice Risk Assessments: The State of the Art. *https://doi.org/10.1177/0049124118782533* 50, 1 (2018), 3–44. https://doi.org/10.1177/0049124118782533 Publisher: SAGE PublicationsSage CA: Los Angeles, CA.

[7] Jan Bosch, Helena Holmström Olsson, and Ivica Crnkovic. 2021. Engineering AI Systems: A Research Agenda. https://doi.org/10.4018/978-1-7998-5101-1.ch001 ISBN: 9781799851011 Pages: 1-19 Publisher: IGI Global.

[8] Michael Buckland and Fredric Gey. 1994. The relationship between recall and precision. *Journal of the American society for information science* 45, 1 (1994), 12–19. Publisher: Wiley Online Library.

[9] Johannes Bürdek, Timo Kehrer, Malte Lochau, Dennis Reuling, Udo Kelter, and Andy Schürr. 2016. Reasoning about product-line evolution using complex feature model differences. *Automated Software Engineering* 23, 4 (Dec. 2016), 687–733. https://doi.org/10.1007/s10515-015-0185-3

[10] Rafael Capilla, Jan Bosch, Pablo Trinidad, Antonio Ruiz-Cortés, and Mike Hinchey. 2014. An overview of Dynamic Software Product Line architectures and techniques: Observations from research and industry. *Journal of Systems and Software* 91 (2014), 3–23. https://doi.org/10.1016/j.jss.2013.12.038

[11] Diogo V Carvalho, Eduardo M Pereira, and Jaime S Cardoso. 2019. Machine learning interpretability: A survey on methods and metrics. *Electronics* 8, 8 (2019), 832.

[12] Lianping Chen, Muhammad Ali Babar, and Bashar Nuseibeh. 2013. Characterizing Architecturally Significant Requirements. *IEEE Software* 30, 2 (2013), 38–45. https://doi.org/10.1109/MS.2012.174

[13] Stephen A Cook. 1983. An overview of computational complexity. *Commun. ACM* 26, 6 (1983), 400–408.

[14] GM Cramer, RA Ford, and RL Hall. 1976. Estimation of toxic hazard—a decision tree approach. *Food and cosmetics toxicology* 16, 3 (1976), 255–276.

[15] Elizamary de Souza Nascimento, Iftekhar Ahmed, Edson Oliveira, Márcio Piedade Palheta, Igor Steinmacher, and Tayana Conte. 2019. Understanding development process of machine learning systems: Challenges and solutions. In *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. IEEE, 1–6.

[16] Claudio Di Sipio, Juri Di Rocco, Davide Di Ruscio, and Dr. Phuong Thanh Nguyen. 2021. A Low-Code Tool Supporting the Development of Recommender Systems. In *Fifteenth ACM Conference on Recommender Systems*. ACM, Amsterdam Netherlands, 741–744. https://doi.org/10.1145/3460231.3478885

[17] Pedro Domingos and Michael Pazzani. 1997. On the Optimality of the Simple Bayesian Classifier under Zero-One Loss. *Machine Learning* 29, 2 (Nov. 1997), 103–130. https://doi.org/10.1023/A:1007413511361

[18] Cynthia Dwork, Moritz Hardt, Toniann Pitassi, Omer Reingold, and Richard Zemel. 2012. Fairness through awareness. *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference* (2012), 214–226. https://doi.org/10.1145/2090236.2090255

[19] Michael Feldman, Sorelle A. Friedler, John Moeller, Carlos Scheidegger, and Suresh Venkatasubramanian. 2015. Certifying and Removing Disparate Impact. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, Sydney NSW Australia, 259–268. https://doi.org/10.1145/2783258.2783311

[20] Benjamin C. M. Fung, Ke Wang, Rui Chen, and Philip S. Yu. 2010. Privacy-Preserving Data Publishing: A Survey of Recent Developments. *ACM Comput. Surv.* 42, 4, Article 14 (June 2010), 53 pages. https://doi.org/10.1145/1749603.1749605

[21] Görkem Giray. 2021. A software engineering perspective on engineering machine learning systems: State of the art and challenges. *Journal of Systems and Software* (2021), 111031.

[22] P. M. Goncalves Jr. and R. S. M. Barros. 2011. Automating Data Preprocessing with DMPML and KDDML. In *2011 10th IEEE/ACIS International Conference on Computer and Information Science*. 97–103. https://doi.org/10.1109/ICIS.2011.23

[23] Gongde Guo, Hui Wang, David Bell, Yaxin Bi, and Kieran Greer. 2003. KNN model-based approach in classification. In *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*. Springer, 986–996.

[24] Martin T Hagan, Howard B Demuth, and Mark Beale. 1997. *Neural network design*. PWS Publishing Co.

[25] Koichi Hamada, Fuyuki Ishikawa, Satoshi Masuda, Tomoyuki Myojin, Yasuharu Nishi, Hideto Ogawa, Takahiro Toku, Susumu Tokumoto, Kazunori Tsuchiya, Yasuhiro Ujita, et al. 2020. Guidelines for Quality Assurance of Machine Learning-based Artificial Intelligence.. In *SEKE*. 335–341.

[26] Hannes Hapke and Catherine Nelson. 2020. *Building Machine Learning Pipelines*. "O'Reilly Media, Inc.". Google-Books-ID: H6_wDwAAQBAJ.

[27] Moritz Hardt, Eric Price, Eric Price, and Nati Srebro. 2016. Equality of Opportunity in Supervised Learning. In *Advances in Neural Information Processing Systems*, Vol. 29. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2016/hash/9d2682367c3935defcb1f9e247a97c0d-Abstract.html

[28] Xin He, Kaiyong Zhao, and Xiaowen Chu. 2021. AutoML: A survey of the state-of-the-art. *Knowledge-Based Systems* 212 (2021), 106622. https://doi.org/10.1016/j.knosys.2020.106622

[29] Fuyuki Ishikawa. 2018. Concepts in quality assessment for machine learning-from test data to arguments. In *International Conference on Conceptual Modeling*. Springer, 536–544.

[30] ISO. 2011. *ISO/IEC 25010:2011*. Technical Report. https://www.iso.org/cms/render/live/en/sites/isoorg/contents/data/standard/03/57/35733.html

[31] Faisal Kamiran and Toon Calders. 2012. Data preprocessing techniques for classification without discrimination. *Knowledge and Information Systems* 33, 1 (Oct. 2012), 1–33. https://doi.org/10.1007/s10115-011-0463-8

[32] Kyo C Kang, Sholom G Cohen, James A Hess, William E Novak, and A Spencer Peterson. 1990. *Feature-oriented domain analysis (FODA) feasibility study*. Technical Report. Carnegie-Mellon Univ Pittsburgh Pa Software Engineering Inst.

[33] Michael J Kearns. 1990. *The computational complexity of machine learning*. MIT press.

[34] Fumihiro Kumeno. 2019. Sofware engneering challenges for machine learning applications: A literature review. *Intelligent Decision Technologies* 13, 4 (2019), 463–476.

[35] Matt J Kusner, Joshua Loftus, Chris Russell, and Ricardo Silva. 2017. Counterfactual Fairness. In *Advances in Neural Information Processing Systems* (2017), Vol. 30. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2017/hash/a486cd07e4ac3d270571622f4f316ec5-Abstract.html

[36] Pantelis Linardatos, Vasilis Papastefanopoulos, and Sotiris Kotsiantis. 2021. Explainable AI: A Review of Machine Learning Interpretability Methods. *Entropy* 23, 1 (2021), 18.

[37] Fernando Martínez-Plumed, Lidia Contreras-Ochando, Cesar Ferri, Jose Hernandez Orallo, Meelis Kull, Nicolas Lachiche, Maréa José Ramírez Quintana, and Peter A Flach. 2019. CRISP-DM twenty years later: From data mining processes to data science trajectories. *IEEE Transactions on Knowledge and Data Engineering* (2019).

[38] Silverio Martínez-Fernández, Justus Bogner, Xavier Franch, Marc Oriol, Julien Siebert, Adam Trendowicz, Anna Maria Vollmer, and Stefan Wagner. 2022. Software Engineering for AI-Based Systems: A Survey. *ACM Transactions on Software Engineering and Methodology* 31, 2 (April 2022), 1–59. https://doi.org/10.1145/3487043 arXiv:2105.01984 [cs].

[39] Ninareh Mehrabi, Fred Morstatter, Nripsuta Saxena, Kristina Lerman, and Aram Galstyan. 2021. A Survey on Bias and Fairness in Machine Learning. *Comput. Surveys* 54, 6 (July 2021), 1–35. https://doi.org/10.1145/3457607

[40] Scott Menard. 2002. *Applied logistic regression analysis*. Vol. 106. Sage.

[41] Henry Muccini and Karthik Vaidhyanathan. 2021. Software Architecture for ML-based Systems: What Exists and What Lies Ahead. In *2021 IEEE/ACM 1st Workshop on AI Engineering - Software Engineering for AI (WAIN)*. 121–128. https://doi.org/10.1109/WAIN52551.2021.00026

[42] United Nations. [n.d.]. THE 17 GOALS | Sustainable Development. https://sdgs.un.org/goals

[43] Preston Putzel and Scott Lee. 2022. Blackbox Post-Processing for Multiclass Fairness. *arXiv:2201.04461 [cs]* (Jan. 2022). http://arxiv.org/abs/2201.04461 arXiv:2201.04461.

[44] G.H. Rosenfield and K. Fitzpatrick-Lins. 1986. A coefficient of agreement as a measure of thematic classification accuracy. *Photogrammetric Engineering and Remote Sensing* 52, 2 (1986), 223–227. http://pubs.er.usgs.gov/publication/70014667

[45] Victor Ruehle, Robert Sim, Sergey Yekhanin, Nishanth Chandran, Melissa Chase, Daniel Jones, Kim Laine, Boris Kopf, James Teevan, Jim Kleewein, and Saravan Rajmohan. 2021. Privacy Preserving Machine Learning: Maintaining confidentiality and preserving trust. https://www.microsoft.com/en-us/research/blog/privacy-preserving-machine-learning-maintaining-confidentiality-and-preserving-trust/

[46] Mauno Rönkkö, Jani Heikkinen, Ville Kotovirta, and Venkatachalam Chandrasekar. 2015. Automated preprocessing of environmental data. *Future Generation Computer Systems* 45 (2015), 13–24. https://doi.org/10.1016/j.future.2014.10.011

[47] Julien Siebert, Lisa Joeckel, Jens Heidrich, Adam Trendowicz, Koji Nakamichi, Kyoko Ohashi, Isao Namba, Rieko Yamamoto, and Mikio Aoyama. 2021. Construction of a quality model for machine learning systems. *Software Quality Journal* (2021), 1–29.

[48] Dave Steinberg, Frank Budinsky, Marcelo Paternostro, and Ed Merks. 2008. *EMF: Eclipse Modeling Framework, 2nd Edition* (2nd ed.). Addison-Wesley Professional. https://www.informit.com/store/emf-eclipse-modeling-framework-9780321331885

[49] Stefan Studer, Thanh Binh Bui, Christian Drescher, Alexander Hanuschkin, Ludwig Winkler, Steven Peters, and Klaus-Robert Müller. 2021. Towards CRISP-ML (Q): a machine learning process model with quality assurance methodology. *Machine Learning and Knowledge Extraction* 3, 2 (2021), 392–413.

[50] Abdel Aziz Taha and Allan Hanbury. 2015. Metrics for evaluating 3D medical image segmentation: analysis, selection, and tool. *BMC Medical Imaging* 15, 1 (Aug. 2015), 29. https://doi.org/10.1186/s12880-015-0068-x

[51] Thomas Thum, Christian Kastner, Sebastian Erdweg, and Norbert Siegmund. 2011. Abstract features in feature modeling. In *2011 15th International Software Product Line Conference*. IEEE, 191–200.

[52] Hugo Villamizar, Tatiana Escovedo, and Marcos Kalinowski. 2021. Requirements Engineering for Machine Learning: A Systematic Mapping Study. In *SEAA*. 29–36.

[53] Vladimir Viyović, Mirjam Maksimović, and Branko Perisić. 2014. Sirius: A rapid development of DSM graphical editor. In *IEEE 18th International Conference on Intelligent Engineering Systems INES 2014*. 233–238. https://doi.org/10.1109/INES.2014.6909375 ISSN: 1543-9259.

[54] Jie M Zhang, Mark Harman, Lei Ma, and Yang Liu. 2020. Machine learning testing: Survey, landscapes and horizons. *IEEE Transactions on Software Engineering* (2020).