Democratizing Quality-Based Machine Learning Development through Extended Feature Models *

Giordano d'Aloisio[0000−0001−7388−890X], Antinisca Di Marco[0000−0001−7214−9945], and Giovanni Stilo[0000−0002−2092−0213]

University of L'Aquila, Italy
giordano.daloisio@graduate.univaq.it
{antinisca.dimarco,giovanni.stilo}@univaq.it

Abstract. ML systems have become an essential tool for experts of many domains, data scientists and researchers, allowing them to find answers to many complex business questions starting from raw datasets. Nevertheless, the development of ML systems able to satisfy the stakeholders' needs requires an appropriate amount of knowledge about the ML domain. Over the years, several solutions have been proposed to automate the development of ML systems. However, an approach taking into account the new quality concerns needed by ML systems (like fairness, interpretability, privacy, and others) is still missing.
In this paper, we propose a new engineering approach for the quality-based development of ML systems by realizing a workflow formalized as a Software Product Line through Extended Feature Models to generate an ML System satisfying the required quality constraints. The proposed approach leverages an experimental environment that applies all the settings to enhance a given Quality Attribute, and selects the best one. The experimental environment is general and can be used for future quality methods' evaluations. Finally, we demonstrate the usefulness of our approach in the context of multi-class classification problem and fairness quality attribute.

Keywords: Machine Learning System · Software Quality · Feature Models · Software Product Line · Low-code development

1 Introduction

Machine Learning (ML) systems are increasingly becoming used instruments, applied to all application domains and affecting our real life. The development

* This work has been partially supported by EMELIOT national research project, which has been funded by the MUR under the PRIN 2020 program (Contract 2020W3A5FY) and by European Union – Horizon 2020 Program under the scheme "INFRAIA-01-2018-2019 – Integrating Activities for Advanced Communities", Grant Agreement n.871042, "SoBigData++: European Integrated Infrastructure for Social Mining and Big Data Analytics" (http://www.sobigdata.eu)

of ML systems usually requires a good knowledge of the underlying ML approaches to choose the best techniques and models to solve the targeted problem. Many methods have been developed in the last years to automate some ML systems development phases and help non-technical users [61,31,34]. However, these techniques do not consider the *quality properties* essential for ML systems, such as dataset's Privacy, model's Interpretability, Explainability, and Fairness [50,46,12]. 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 (look for instance at some of the 17 sustainable development goals proposed by the United Nations [51]).

In this paper, we present MANILA (**M**odel b**A**sed developme**N**t of mach**I**ne **L**earning systems with qu**A**lity), a novel approach which will democratize the quality-based development of ML systems by means of a low-code platform [62]. The goal of our approach is to provide an environment for the automatic configuration of experiments that automatically selects the ML System (i.e., ML Algorithm and quality enhancing method) better satisfying a given quality requirement. The requirement is satisfied by finding the best trade-off among the involved quality attributes. This will simplify the work of the data scientist and will make the quality-based development of ML systems also accessible to non-technical users (in other words, *democratize*).

Hence, the main contributions of this paper are the following:

- The identification of key quality attributes in ML systems by selecting the more adopted ones in the literature;
- The specification and realization of a general workflow for the quality-based development of ML systems. This workflow is derived from our experience in the quality-based development of ML systems. It leverages an experimental environment that evaluates all the methods to enhance a given quality attribute, and selects the one performing better. Such workflow can be modelled as a Software Product Line (SPL);
- The specification of an Extended Feature Models (ExtFM) [38,9] that implements the SPL, where the variation points are identified by all the components needed to generate a quality experiment. The ExtFM guides the data scientist through a low-code workflow configuration;
- The generation, from the workflow configuration, of an actual Python implementation of the experiment to find the ML System that better satisfies a given quality constraint. The generated experimental environment is general and can be used in the future to evaluate other methods to enhance a given quality property.

This paper is organized as follows: in section 2 we discuss related works related to quality engineering of ML systems.In section 3, we present the selected quality attributes and discuss how they affect ML systems. Section 4 is devoted to presenting a general workflow to choose the ML system achieving the best-given quality attributes. This general workflow has been the motivating scenario for MANILA. In section 5, we present MANILA by describing in detail the implemented ExtFM and explaining each step of the quality-based development

of ML systems. Section 6 is dedicated to a proof of concept of the developed modelling framework by reproducing a case study. Section 7 describes some threats to validity, and finally, section 8 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 formalizing new standard quality attributes for machine learning systems [30,65,70] [50,12,46]. 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 ML systems development.

Concerning the identification of quality attributes in ML systems, the authors of [40,72] 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 *performance reliability and availability*. Muccini et al. identify in [50] 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 [46] the needing of formalizing quality properties in ML systems and to update the software quality requirements defined by ISO 25000 [36]. The most relevant properties highlighted by the authors concern: *ML safety*, *ML ethics*, and *ML explainability*. In our work, we focus on quality properties that arises during the development of ML systems such as, *fairness*, *explainability*, *interpretability*, and dataset's *privacy*, while we leave other quality properties (e.g., *performance*) that arises during other phases (e.g., deployment) for future works.

Many solutions have been proposed to formalize and model standard quality assurance process in ML systems. Amershi et al., have been the first authors to identify a set of common steps that identify each ML system development [5]. In particular, each ML system is identified by nine stages that go from data collection and cleaning, to model training and evaluation, and finally to the deployment and monitoring of the ML model. Their work has been the foundation of many subsequent papers on quality modelling of ML systems. *CRISP_ML* (*Cross-Industry Standard Process model for Machine Learning*) is a process model proposed by Studer et al. [66], extending the more known *CRISP_DL* [45] process model to ML 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 *Quality for Artificial Inteligence* (*Q4AI*) consortium proposed a set of guidelines [32] 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 Internet of Things (IoT) [7]. The model is made of two layers: *Source Data* and *ML Function/Model*. For the *Source Data*, a set of quality attributes are defined: *completeness*, *consistency*, *conformity*, *accuracy*, *integrity*, *timeliness*. Machine learning models are instead classified into *predictors*, *estimators* and *adapters* and a set of quality attributes 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 [35]. 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. [64] proposed a formal modelling definition for quality requirements in ML systems. They start from the process definition in [45] 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. 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 extracted from the literature. However a framework that actually guides the data scientist through the development of a ML systems satisfying quality properties is still missing.In this paper, we aim to solve these concerns by proposing MANILA, a novel approach which will democratize the quality-based development of ML systems by means of a low-code platform. In particular, we model a general workflow for the quality-based development of ML systems as a SPL through the ExtFM formalism. Next, we demonstrate how it is possible to generate an actual implementation of such workflow from a low-code experiment configuration and how this workflow is actually able to find the best methods to satisfy a given quality requirement. Recalling the ML development process of [5], MANILA focuses on the *model training* and *model evaluation* development steps by guiding the data scientist in selecting the ML system (i.e., ML algorithm and quality-enhancing method) better satisfying a given quality attribute.

Concerning the adoption of Feature Models to model ML systems, a similar approach has been used by Di Sipio et al. in [24]. In their work, the authors use Feature Models to model ML pipelines for Recommender Systems. The variation points are identified by all the components needed to implement a recommender system (e.g., the ML algorithm to use or the python libraries for the implementation). However, they do not consider quality attributes in their approach.

Finally, concerning assessing quality attributes in ML systems, there is an intense research activity primarily related to the fairness-testing domain [20]. In general, the problem of fairness assurance can be defined as a search-based problem among different ML algorithms and fairness methods [20]. Many tools have been proposed for the automatic fairness test, such as [18,63,69] to cite a few. However, these tools tend to require programming skills and thus are unfriendly to nontechnical stakeholders [20]. In our work, we aim to fill this gap by proposing a low-code framework that, generating and executing suitable experiments, supports (also not expert) users in the quality-based development of ML systems, by returning the trained ML model with best quality.

3 Considered Quality Attributes

In software engineering, a quality requirement specifies criteria that can be used to quantify or qualify the operation of a system rather than to specify its behaviours [19]. To analyse an ML system from a qualitative perspective, we must determine the Quality Attributes (QA) that we can use to judge the system's operation, influencing the ML designers' decisions. We refer to the literature for ML systems to identify the QA to consider [46,50,30,40,70]. In this work, we consider a sub-set of the identified QA, i.e., *Effectiveness*, *Fairness*, *Interpretability*, *Explainability*, and *Privacy*.

Effectiveness. This QA is used to define how good the model must be in predicting outcomes [13]. There are different metrics in the literature to address the Effectiveness of an ML model. Among the most common metrics, we cite *Precision*: fraction of true positives (TP) to the total positive predictions [14]; *Recall*: fraction of TP to the total positive items in the dataset [14]; *F1 Score*: harmonic mean of *Precision* and *Recall* [67]; *Accuracy*: fraction of True Positives (TP) and True Negatives (TN) above the total of predictions [60]. This attribute can be considered crucial in developing an ML system and must always be accounted in the quality evaluation of ML systems [72,13].

Fairness. A ML model can be defined *fair* if it has no prejudice or favouritism towards an individual or a group based on their inherent or acquired characteristics identified by the so-called *sensitive variables* [47]. Sensitive variables are variables of the dataset that can cause prejudice or favouritism towards individuals having a particular value of that variable (e.g., *sex* is a very common sensitive variable, and *women* can be identified as the unprivileged group [47,16,42]). Several metrics can assess the discrimination of an ML system towards sensitive groups (*group fairness metrics*) or single individuals (*individual-fairness metrics*) [47,16].

Interpretability. *Interpretability* can be defined as the ability of a system to enable user-driven explanations of how a model reaches the produced conclusion [15]. Interpretability is one QA that can be estimated without executing an actual ML system. Indeed, ML methods are classified as *whitebox*, i.e., interpretable (e.g., Decision Trees or linear models), and *black-box*, i.e., not interpretable (e.g., Neural Networks) [49]. 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) require the addition of *explainability*-enhancing methods to have their results interpretable [43].

Explainability. *Explainability* can be defined as the ability to make black-box methods' results (which are not interpretable) interpretable [43]. Enhancing the Interpretability of black-box methods has become crucial to guarantee the trustworthiness of ML systems, and several methods have been implemented for this purpose [43]. The quality of explanations can be measured with several metrics that can be categorised as *application-grounded* metrics, which involve an evaluation of the explanations with end-users, *human-grounded* metrics, which include evaluations of explanations with non-domain-experts, and *functionally-grounded* metrics, which use proxies based on a formal definition of interpretability [73].

Privacy. *Privacy* can be defined as the susceptibility of data or datasets to revealing private information [21]. Several metrics can assess the ability to link personal data to an individual, the level of detail or correctness of sensitive information, background information needed to determine private information, etc [71].

4 Motivating Scenario

Today, a data scientist, required to realize an ML system satisfying a given quality constraint, has no automatic support in the development process. Indeed, she follows and manually executes a general experiment workflow aiming at evaluating a set of ML systems obtained by assembling quality assessment and improvement algorithms with the ones solving the specific ML tasks. By running the defined experiment, she aims to find the optimal solution satisfying a given QA constraint.

Algorithm 1 reports the pseudo-code of a generic experiment to assess a generic QA during the development of an ML system. This code has been derived from our previous experience in the quality-based development of ML Systems and by asking researchers studying ML development and quality assessment how they evaluate such properties during ML systems development.

The first step in the experiment workflow is selecting the dataset to use (in this work, we assume that the dataset has already been preprocessed and is ready to train the ML model). Next, the data scientist selects the ML algorithms, the methods enhancing a QA, and the appropriate quality metrics for the evaluation. Then, for each of the chosen ML algorithms, she applies the selected quality methods accordingly to their type, there can be the following options:

Algorithm 1: Quality-evaluation experiment pseudo-code

```
1  select dataset d;
2  select set of ML Algorithms;
3  select set of QA Methods and Metrics;
4  for m ∈ ML Algorithms do
5      for q ∈ QA Methods do
6          if q works on d then
7              apply q on d;
8          if q works on m before training then
9              apply q on m;
10         f = train m;
11         if q works on f then
12             apply q on f;
13         compute selected metrics on f;
14 choose report technique;
15 evaluate the results;
16 Q = best QA Method;
17 M = best ML Algorithm;
18 F = train M with full dataset applying Q;
19 return F
```

- if the quality method works on the training set, it has to be applied to the dataset before training the ML algorithm;
- if the quality method works on the ML algorithm before training, then it has to be applied to the ML algorithm before the training phase;
- if the method works on the trained ML algorithm (i.e., f in the code), then it has to be applied after the training of the ML algorithm.

Finally, the data scientist computes the selected metrics for the specific pair of ML and QA methods. After repeating the process for all the selected methods, she chooses a report technique (e.g., table or chart), evaluates the obtained results collected in the report and trains with the entire dataset the ML algorithm performing better by applying the quality method that better achieves the QA. If the data scientist has a threshold to achieve, then she can verify if at least one of the ML and quality methods combinations satisfies the constraint. If so, one of the suitable pair is selected. Otherwise, she has to relax the threshold and repeat the process again.

The workflow described in Algorithm 1 can be generalized as a process of common steps describing any experiment in the considered domain. Figure 1 sketches such a generalization. First, the data scientist selects all the features of the experiment, i.e., the dataset, the ML Methods, the methods assuring a specific QA and the related metrics. we call such a step *Features Selection*. Next, she runs the quality methods using the general approach described in algorithm 1 and evaluates the results (namely, *Experiment Execution*). If the results are

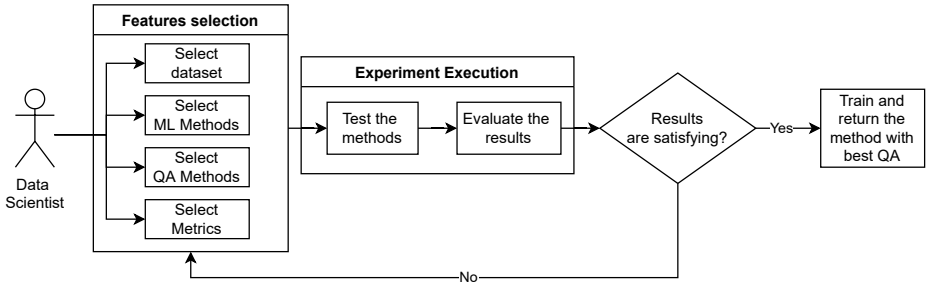


Fig. 1: Manual execution of the quality experiment workflow

satisfying (i.e., they satisfy the quality constraints), then the method with the best QA is returned. Otherwise, the data scientist have to repeat the process.

The described workflow is the foundation of MANILA that aims to formalise and democratise it by providing a SPL and ExtFM-based low-code framework that supports her in development of quality ML systems.

5 MANILA Approach

In this section, we describe MANILA, a framework to formalise and democratise the quality-based development of ML systems. This work is based on the quality properties and the experiment workflow described in sections 3 and 4, respectively.

Our approach aims to automate and ease the quality-based development of ML systems. We achieve this goal by proposing a framework to automatically generate a configuration of an experiment to find the ML system (i.e., ML algorithm and quality enhancing method) better satisfying a given QA. This framework will accelerate the quality-based development of ML systems making it accessible also to not experts.

Recalling the experimental workflow described in section 4, the set of ML models, quality methods and metrics can be considered *variation points* of each experiment, differentiating them from one another. For this reason, we can think of this family of experiments as a Software Product Line (SPL) specified by a Feature Model [6]. Indeed, Feature Models allow us to define a template for families of software products with standard features (i.e., components of the final system) and a set of variability points that differentiate the final systems [38,29]. Features in the model follow a tree-like parent-child relationship and could be *mandatory* or *optional* [29]. Sibling features can belong to an *Or-relationship* or an *Alternative-relationship* [29]. Finally, there could be *Cross-tree* relationships among features not in the same branch. These relationships are expressed using logical propositions [29]. However, traditional Feature Models do not allow associating attributes to features, which are necessary in our case to represent a proper experiment workflow (for instance, to specify the label of

the dataset or the number of rounds in a cross-validation [58]). Hence, we relied on the concept of Extended Feature Models [38,9] to represent the family of experiments workflows.

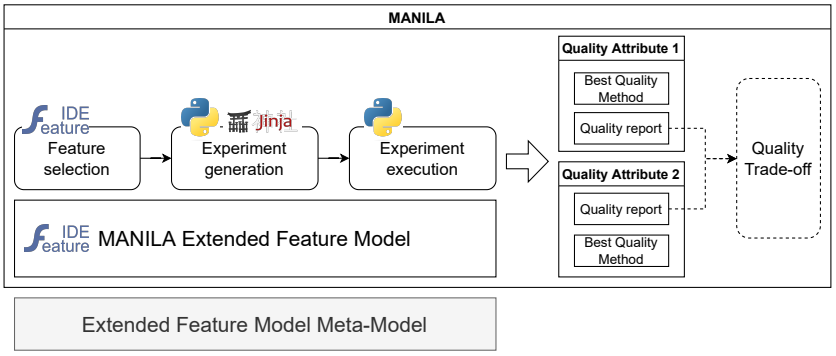


Fig. 2: MANILA approach

Figure 2 details a high-level picture of MANILA, where each rounded box represents a step in the quality-driven development process, while square boxes represent artefacts. Dotted blocks represent steps which have not been implemented yet and will be considered in future works.

The basis of MANILA is the Extended Feature Model (ExtFM), based on the existing ExtFM Meta-Model. The ExtFM is the template of all possible experiments a data scientist can perform and guides her through the quality-based development of an ML system. The first step in the development process is the features selection, in which the data scientist selects all the components of the quality-testing experiment. Next, a Python script implementing the experiment is automatically generated from the selected features. Finally, the experiment is executed, and for each QA selected, it returns:

1. a quality report reporting for each quality method and ML algorithm the related metrics;
2. the ML algorithm with the applied quality enhancing method that better performs with the given QA, trained and ready for production.

In the future, MANILA will analyse the quality reports of each selected QA in order to find the best trade-off among them (for instance, by means of Pareto-front functions). The architecture of MANILA makes it easy to extend. In fact, adding a new method or metric to MANILA just translates to adding a new feature to the ExtFM and adding the proper code implementing it.

Near each step, we report the tools involved in its implementation. The source code of the implemented artefacts is available on Zenodo [23], and GitHub [22]. In the following, we detail the ExtFM and each process step.

5.1 Extended Feature Model

As already mentioned, the ExtFM is the basis of MANILA approach since it defines the template of all possible experiments a data scientist can generate. It has been implemented using *FeatureIDE*, an open-source graphical editor which allows the definition of ExtFMs [68]. Figure 3 shows a short version of the im-

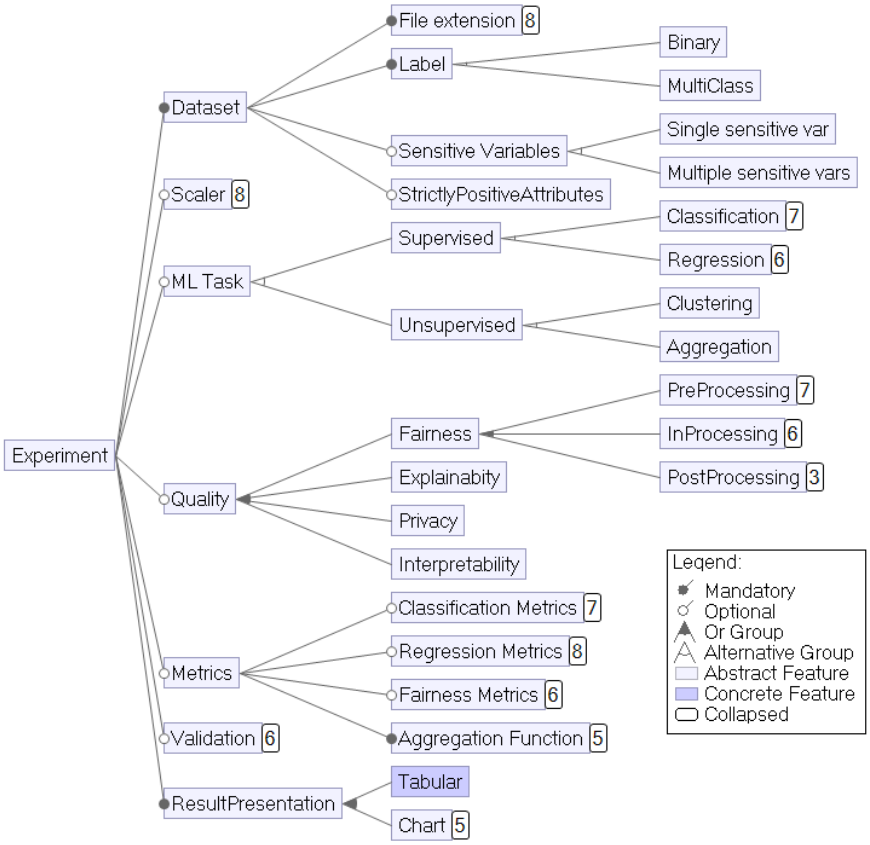


Fig. 3: Short version of the implemented Extended Feature Model

plemented ExtFM[1]. In particular, each experiment is defined by seven macro features, which are then detailed by children's features.

The first mandatory feature is the Dataset. The Dataset has a file extension (e.g., CSV, EXCEL, JSON, and others), and a Label which can be Binary or Multi-Class. The Label feature has two attributes specifying his name and

[1] The whole picture can be downloaded here https://anonymous.4open.science/r/manila-101D/imgs/feature-model.png

the positive value (used to compute fairness metrics). The Dataset could also have one or more sensitive variables that identify sensitive groups subject to unfairness [47]. The sensitive variables have a set of attributes to specify their name and the privileged and unprivileged groups [47]. Finally, there is a feature to specify if the Dataset has only positive attributes. This feature has been included to define a cross-tree constraint with a scaler technique that requires only positive attributes (see table 1). All these features are modelled as abstract since they do not have a concrete implementation in the final experiment. The next feature is a Scaler algorithm, which is not mandatory and can be included in the experiment to scale and normalize the data before training the ML model [54]. Different scaler algorithms from the *scikit-learn* library [55] are listed as concrete children of this feature. Next, there is the macro-feature representing the ML Task to perform. This feature has not been modelled as mandatory since there are two fairness methods (i.e. Gerry Fair and Meta Fair [39,17]) that embed a fair classification algorithm and so, if they are selected, the ML Task can not be specified. However, we included a cross-tree constraint requiring the selection of ML Task if any of these two methods are selected ($\neg$ Gerry Fair $\wedge$ $\neg$ Meta Fair $\Rightarrow$ ML Task). An ML Task could be Supervised or Unsupervised. A Supervised task could be a Classification task or a Regression task and has an attribute to specify the size of the training set. These two abstract features are then detailed by a set of concrete implementations of ML methods selected from the *scikit-learn* library [55]. The Unsupervised learning task could be a Clustering or an Aggregation task. At this stage of the work, these two features have not been detailed and will be explored in future works. Next is the macro feature representing the system's Quality Attributes. This feature is detailed by the four quality attributes described in section 3. Effectiveness is not included in these features since it is an implicit quality of the ML methods and does not require adding other components (i.e. algorithms) in the experiment. At the time of this paper, the Fairness quality has been detailed, while the other properties will be deepened in future works. In particular, Fairness methods can be *Pre-Processing* (i.e. strategies that try to mitigate the bias on the dataset used to train the ML model [47,37,27]), *In-Processing* (i.e. methods that modify the behaviour of the ML model to improve fairness [47,3]), and *Post-Processing* (i.e. methods that re-calibrate an already trained ML model to remove bias [47,56]). These three features are detailed by several concrete features representing fairness-enhancing methods. In selecting such algorithms, we selected methods with a solid implementation, i.e., algorithms integrated into libraries such as *AIF360* [8] or *Fairlearn* [11] or algorithms with a stable source code such as *DEMV* [26] or *Blackbox* [56]. All these quality features have been implemented with an *Or-group* relationship. Forward, the macro feature represents the Metrics to use in the experiment. Metrics are divided among Classification Metrics, Regression Metrics and Fairness Metrics. Each metric category has a set of concrete metrics selected from the *scikit-learn* library [55] and the *AIF360* library [8]. Based on the ML Task and the Quality Attributes selected, the data scientist must select the proper metrics to assess Correctness and the other Quality Attributes. This

constraint is formalized by cross-tree relationships among features (see table 1). In addition, a set of Aggregation Functions must be selected if more than one metric is selected. The aggregation function combines the value of the other metrics to give an overall view of the method's behaviour. Forward, there is the optional macro feature identifying the Validation function. Validation functions are different strategies to evaluate the Quality Attributes of an ML model [57]. Several Validation functions are available as children features, and there is an attribute to specify the number of groups in case of cross-validation [57]. The last macro-feature is related to the presentation of the results. Recalling the experiment workflow described in section 4, the results are the metrics' values derived from the execution of the experiment. The results can be presented in a tabular way or using proper charts. Different chart types are available as concrete children features. Finally, table 1 lists the cross-tree constraints defined

Table 1: Extended Feature Model cross-tree constraints

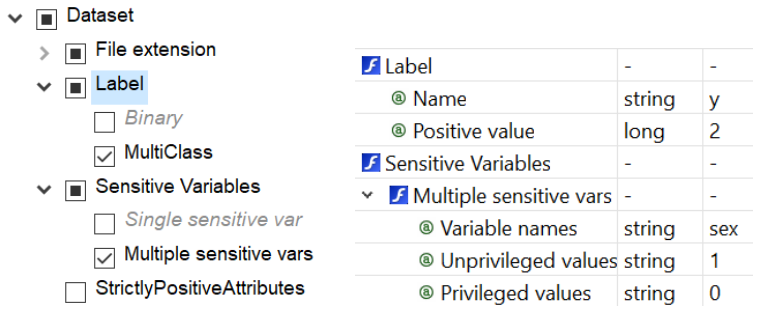
Cross-tree constraints
Single Sensitive Var $\Rightarrow \neg$ Sampling $\wedge \neg$ Blackbox $\wedge \neg$ DIR
Fairness $\Rightarrow$ Sensitive Variables
MultiClass $\Rightarrow \neg$ Reweighing $\wedge \neg$ DIR $\wedge \neg$ Optimized Preprocessing $\wedge \neg$ LFR $\wedge \neg$ Adversarial Debiasing $\wedge \neg$ Gerry Fair $\wedge \neg$ Meta Fair $\wedge \neg$ Prejudice Remover $\wedge \neg$ Calibrated EO $\wedge \neg$ Reject Option
Regression $\Rightarrow \neg$ PostProcessing $\wedge \neg$ Reweighing $\wedge \neg$ DIR $\wedge \neg$ DEMV $\wedge \neg$ Optimized Preprocessing $\wedge \neg$ LFR $\wedge \neg$ Adversarial Debiasing $\wedge \neg$ Gerry Fair $\wedge \neg$ Meta Fair $\wedge \neg$ Prejudice Remover
Exponentiated Gradient $\vee$ Grid Search $\Rightarrow \neg$ MLP Classifier $\wedge \neg$ MLP Regressor
$\neg$ GerryFair $\wedge \neg$ MetaFair $\Rightarrow$ ML Task
Classification $\iff$ Classification Metrics $\wedge \neg$ Regression Metrics
Classification Metrics $\iff \neg$ Regression Metrics
Regression $\iff$ Regression Metrics $\wedge \neg$ Classification Metrics
Fairness $\Rightarrow$ Fairness Metrics
Box Cox Method $\Rightarrow$ Strictly Positive Attributes

in our model. These constraints are useful to guide the data scientist through selecting proper fairness-enhancing methods or metrics based on the Dataset's characteristics (i.e., label type or the number of sensitive variables) or the ML Task.

5.2 Features selection

From the depicted ExtFM, the data scientist can define her experiment by specifying the needed features inside a *configuration file*. A *configuration file* is an XML file describing the set of selected features and the possible attribute values. The constraints among features defined in the ExtFM will guide the data

scientist in the selection by not allowing the selection of features that are in contrast with already selected ones. The editor used to implement the ExtFM [68] provides a GUI for the specification of configuration files, making this process accessible to non-technical users.



(a) Feature selection (b) Attribute specification

Fig. 4: Feature selection and attribute specification process

Figure 4 depicts how the features selection and attribute specification processes are done in MANILA. In particular, figure 4a details how the features of the Dataset are selected inside the configuration. Note how features in contrast with already selected ones are automatically disabled by the system (e.g., the *Binary* feature is disabled since the *MultiClass* feature is selected). This automatic cut of the ExtFM guides the data scientist in defining configurations that always lead to valid (i.e., executable) experiments. Figure 4b details how attributes can be specified during the definition of the configuration. In particular, the rightmost column in figure 4b displays the attribute value specified by the data scientist (e.g., the name of the label is *y*, and the positive value is 2). During the *experiment generation* step, a process will automatically check if all the required attributes (e.g., label name) have been defined. Otherwise, it will ask the data scientist to fill them.

5.3 Experiment generation

From the XML file describing an experiment configuration, it is possible to generate a Python script implementing the defined experiment.

```
<feature automatic="selected" manual="undefined" name="
    Dataset"/>
<feature automatic="selected" manual="undefined" name="Label"
    >
  <attribute name="Positive value" value="2"/>
  <attribute name="Name" value="contr_use"/>
</feature>
```

```
<feature automatic="unselected" manual="undefined" name="
    Binary"/>
<feature automatic="undefined" manual="selected" name="
    MultiClass"/>
```

Listing 1.1: Portion of configuration file

Listing 1.1 shows a portion of the configuration file derived from the feature selection process. In particular, it can be seen how the Dataset and the Label features have been automatically selected by the system (features with `name="Dataset"` and `name="Label"` and `automatic="selected"`), the MultiClass feature has been manually selected by the data scientist (feature with `name="MultiClass"` and `manual="selected"`), and the Binary feature was not selected (feature with `name="Binary"` and both `automatic` and `manual` unselected). In addition, the name and the value of two Label attributes (i.e., `Positive value` equal to 2 and `Name` equal to `contr_use`) are reported.

The structure of the configuration file makes it easy to be parsed by a proper script. In MANILA, we implemented a Python parser that reads the configuration file given as input and generates a set of scripts implementing the defined experiment. The parser can be invoked using the Python interpreter with the following command shown in listing 1.2.

```
$ python generator.py -n <CONFIGURATION FILE PATH>
```

Listing 1.2: Python parser invocation

In particular, the parser first checks if all the required attributes (e.g., the label's name) are set. If some of them are not set, it asks the data scientist to fill them in before continuing the parsing. Otherwise, it selects all the features with `automatic="selected"` or `manual="select"` and uses them to fill a Jinja2 template [53]. The generated quality-evaluation experiment follows the same structure of algorithm 1. It is embedded inside a Python function that takes as input the dataset to use (listing 1.3). An example of a generated file can be accessed on the GitHub [22] or Zenodo [23] repository.

```
def experiment(data):
    # quality evaluation experiment
```

Listing 1.3: Quality-testing experiment signature

In addition to the `main` file, MANILA generates also a set of Python files needed to execute the experiment and an `environment.yml` file containing the specification of the *conda* [1] environment needed to perform the experiment. All the files are generated inside a folder named `gen`.

5.4 Experiment execution

The generated experiment can be invoked directly through the Python interpreter using the command given in listing 1.4. Otherwise, it can be called through

a REST API or any other interface such as a desktop application, or a Scientific Workflow Management System like *KNIME* [44,10]. This generality of our experimental workflow, makes it very flexible and suitable to many use-cases.

```
$ python main.py -d <DATASET PATH>
```

Listing 1.4: Experiment invocation

The experiment applies each ML algorithm with each quality method and returns a report using the adequate selected metrics along with the method achieving the best QA. It is worth noting how each quality method is evaluated individually on the selected ML algorithm, and for each QA, a corresponding report is returned by the system. Figure 5 reports an example of how the quality

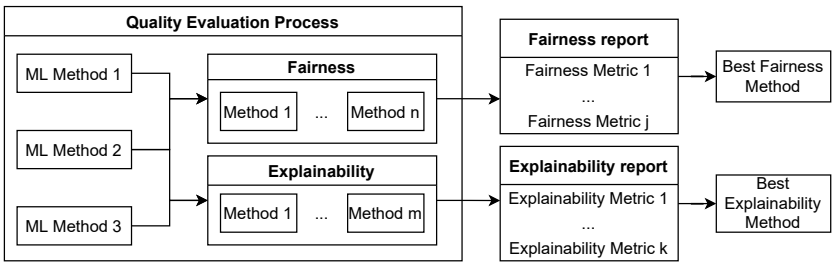


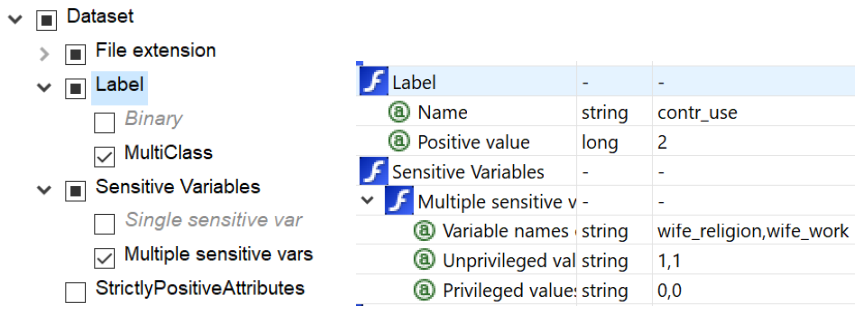
Fig. 5: Quality evaluation process example

evaluation process is done in MANILA. In this example, the data scientist has selected three ML algorithms and wants to assure Fairness and Explainability. She has selected n methods to assure Fairness and m methods to assure Explainability. In addition, she has selected j metrics for Fairness and k metrics for Explainability. Then, the testing process performs two parallel set of experiments. In the first, it applies the n fairness methods to each ML algorithm accordingly and computes the j fairness metrics. In the second, it applies the m Explainability methods to the ML algorithms and computes the k Explainability metrics. Finally, the process returns two reports synthesising the obtained results for Fairness and Explainability along with the ML algorithms with the best Fairness and Explainability, respectively. If the data scientist chooses to see the results in tabular form (i.e., selects the **Tabular** feature in the ExtFM), then the results are saved in a CSV file. Otherwise, the charts displaying the results are saved as PNG files. The ML algorithm returned by the experiment is instead saved as a *pickle* file [2]. We have chosen this format since it is a standard format to store serialized objects in Python and can be easily imported in other scripts.

Finally, it is worth noting how the generated experiment workflow is written in Python and can be customised to address particular stakeholders' needs or evaluate other quality methods.

6 Proof of Concept

To prove the ability of MANILA in supporting the quality-based development of ML systems, we implemented with MANILA a fair classification system to predict the frequency of contraceptive use by women, using a famous dataset in the Fairness literature [42]. This use case is reasonable since fairness has acquired much importance in recent years, partly because of the sustainable goals of the UN [51]. The first step in the quality development process is feature



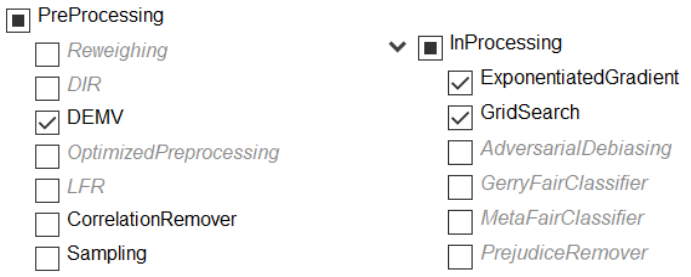
(a) Selected features of the Dataset (b) Attributes of the Dataset

Fig. 6: Dataset specification

selection. The ML task to solve is a multi-class classification problem [4], hence in the ExtFM we selected the feature *MultiClass* for the *Label* and we specified its name and the positive value to consider for the fairness evaluation (*long-term use*). We will use a CSV dataset file, so we specified this feature in the configuration. Finally, accordingly to the literature [42], we specified that the dataset has multiple sensitive variables to consider for fairness, and we specified their names and privileged and unprivileged values. Figure 6 reports the selected features of the Dataset and the attributes specified.

Next, we specified that we want to use a *Standard Scaler* algorithm to normalize the data and we selected the following ML algorithms for classification: *Logistic Regression*[48], *Support Vector Classifier*[52], and *Gradient Boosting Classifier*[28]. Figure 7 reports the Fairness methods we want to test. Note how many methods have been automatically disabled by the system based on the features already selected[2]. Further, we specified the metrics we want to use to evaluate Fairness and Effectiveness: *Accuracy* [60], *Zero One Loss* [25], *Disparate Impact* [27], *Statistical Parity* [41], and *Equalized Odds* [33], and the *Harmonic Mean* as aggregation function (we have chosen this aggregation function since it is widely used in the literature). Finally, we specified that we want to perform a *10-fold* cross validation [59] and that we want the results in tabular form without the

[2] In particular, these methods have been disabled because they do not support multi-class classification or multiple sensitive variables



(a) Pre-processing methods (b) In-processing methods

Fig. 7: Selected Fairness methods

generation of a chart. From the given configuration, MANILA generates all the python files needed to run the quality-assessment experiment. In particular, the generated experiment trains and tests all the selected ML algorithm (i.e., *Logistic Regression*, *Support Vector Classifier*, and *Gradient Boosting Classifier*) applying all the selected fairness methods properly (i.e., *DEMV*, *Exponentiated Gradient*, and *Grid Search*). Finally, it computes the selected metrics on the trained ML algorithms and returns a report of the metrics along with the fully trained ML algorithm with the best fairness. All the generated files are available on Zenodo [23] and Github [22]. The generated experiment was executed di-

Table 2: Generated results

Fairness Method	**ML Model**	**Stat Par**	**Eq Odds**	**ZO Loss**	**Disp Imp**	**Accuracy**	**HMean**
demv	svm	0.004	0.216	0.273	0.708	0.546	0.705
demv	gradient	-0.006	0.197	0.276	0.689	0.561	0.702
demv	logreg	0.003	0.193	0.225	0.676	0.511	0.7
grid	gradient	-0.057	0.21	0.167	0.749	0.443	0.694
eg	gradient	-0.09	0.183	0.309	0.658	0.546	0.685
grid	logreg	-0.012	0.241	0.26	0.815	0.445	0.679
eg	svm	-0.109	0.16	0.337	0.549	0.546	0.652
eg	logreg	-0.107	0.218	0.35	0.543	0.509	0.617
grid	svc	-0.197	0.273	0.295	0.197	0.435	0.301

rectly from the python interpreter, and the obtained results are available in table 2. In the table are reported the Fairness enhancing methods, the ML algorithms and all the metrics computed. The table has been automatically ordered based on the given aggregation function (i.e., the rightmost column *HMean*). From the results, we can see that the Support Vector Classifier (i.e., *svc* in the table) and the *DEMV* fairness method can achieve the best Fairness and Effectiveness trade-off, since they have the highest *HMean* value (highlighted in green in table 2). Hence, the ML algorithm returned by the experiment is the Support Vector Classifier, trained with the full dataset after the application of the DEMV algorithm.

7 Threats to Validity

Although the QA considered in MANILA are the most relevant and the most cited in the literature, there could be other QA highly affecting the environment/end users of the ML system that are not focused prominently by existing papers. In addition, the proposed experimental workflow is based on the considered QA; there could be other QA not considered at the time of this paper that should be evaluated differently.

8 Conclusion and Future Work

In this paper, we have presented MANILA, a novel approach to democratize the quality-based development of ML systems. First, we have identified the most influential quality properties in ML systems by selecting the quality attributes that are most cited in the literature. Next, we have presented a general workflow for the quality-based development of ML systems. Finally, we described MANILA in detail by first explaining how the general workflow can be formalized through an ExtFM. Next, we detailed all the steps required to develop a quality ML system using MANILA. We started from the low-code configuration of the experiment to perform; we described how a Python implementation could be generated from such a configuration. Finally, we showed how the execution of the experiment could identify the method better satisfying a given quality requirement. We have demonstrated the ability of MANILA in guiding the data scientists through the quality-based development of ML systems by implementing a *fair* multi-class classification system to predict the use of contraceptive methods by women.

In future, we plan to improve MANILA by extending the ExtFM with additional methods enhancing other quality attributes and by implementing in the framework the trade-off analysis that combines the different quality attribute evaluations when required by means of Pareto-front functions. MANILA appears to be easy to use and very general, able to embed different quality attributes that are quantitatively measured. To demonstrate our intuition, we will conduct a user evaluation of MANILA, to evaluate its usability by involving experts and not experts of the quality ML system development. Some groups we aim to involve are: master students in computer science and applied data science (i.e., non-expert users), data scientists working in industries, and researchers studying ML development and quality assessment (i.e., expert users). In addition, since MANILA supports the configuration of an experiment by running all possible combinations of the selected features, a limit of the proposed approach can be its complexity and the time needed to obtain the results. Such limitation is mitigated by the feature selection step, which demands the user to choose which features to include in the experiment. As future work, to enlarge the MANILA usage, we will better study such aspects and provide guidelines to the users on how to mitigate such potential limitations.

References

1. Conda website, https://docs.conda.io/
2. Pickle documentation, https://docs.python.org/3/library/pickle.html
3. Agarwal, A., Beygelzimer, A., Dudik, M., Langford, J., Wallach, H.: A Reductions Approach to Fair Classification. In: Proceedings of the 35th International Conference on Machine Learning. pp. 60–69. PMLR (Jul 2018), https://proceedings.mlr.press/v80/agarwal18a.html, iSSN: 2640-3498
4. Aly, M.: Survey on multiclass classification methods. Neural Netw **19**(1-9), 2 (2005)
5. Amershi, S., Begel, A., Bird, C., DeLine, R., Gall, H., Kamar, E., Nagappan, N., Nushi, B., Zimmermann, T.: Software Engineering for Machine Learning: A Case Study. In: 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP). pp. 291–300. IEEE, Montreal, QC, Canada (May 2019). https://doi.org/10.1109/ICSE-SEIP.2019.00042, https://ieeexplore.ieee.org/document/8804457/
6. Apel, S., Batory, D., Kästner, C., Saake, G.: Feature-oriented software product lines. Springer (2016)
7. Azimi, S., Pahl, C.: A layered quality framework for machine learning-driven data and information models. In: ICEIS (1). pp. 579–587 (2020)
8. Bellamy, R.K., Dey, K., Hind, M., Hoffman, S.C., Houde, S., Kannan, K., Lohia, P., Martino, J., Mehta, S., Mojsilović, A., et al.: Ai fairness 360: An extensible toolkit for detecting and mitigating algorithmic bias. IBM Journal of Research and Development **63**(4/5), 4–1 (2019)
9. Benavides, D., Segura, S., Ruiz-Cortés, A.: Automated analysis of feature models 20 years later: A literature review. Information Systems **35**(6), 615–636 (Sep 2010). https://doi.org/10.1016/j.is.2010.01.001, https://www.sciencedirect.com/science/article/pii/S0306437910000025
10. Berthold, M.R., Cebron, N., Dill, F., Gabriel, T.R., Kötter, T., Meinl, T., Ohl, P., Thiel, K., Wiswedel, B.: Knime - the konstanz information miner: Version 2.0 and beyond. SIGKDD Explor. Newsl. **11**(1), 26–31 (Nov 2009). https://doi.org/10.1145/1656274.1656280, https://doi-org.univaq.clas.cineca.it/10.1145/1656274.1656280
11. Bird, S., Dudík, M., Edgar, R., Horn, B., Lutz, R., Milan, V., Sameki, M., Wallach, H., Walker, K.: Fairlearn: A toolkit for assessing and improving fairness in AI. Tech. Rep. MSR-TR-2020-32, Microsoft (May 2020), https://www.microsoft.com/en-us/research/publication/fairlearn-a-toolkit-for-assessing-and-improving-fairness-in-ai/
12. Bosch, J., Olsson, H.H., Crnkovic, I.: Engineering AI Systems: A Research Agenda (2021). https://doi.org/10.4018/978-1-7998-5101-1.ch001, https://www.igi-global.com/chapter/engineering-ai-systems/www.igi-global.com/chapter/engineering-ai-systems/266130, iSBN: 9781799851011 Pages: 1-19 Publisher: IGI Global
13. Braiek, H.B., Khomh, F.: On testing machine learning programs. Journal of Systems and Software **164**, 110542 (2020). https://doi.org/https://doi.org/10.1016/j.jss.2020.110542, https://www.sciencedirect.com/science/article/pii/S0164121220300248
14. Buckland, M., Gey, F.: The relationship between recall and precision. Journal of the American society for information science **45**(1), 12–19 (1994), publisher: Wiley Online Library

15. Carvalho, D.V., Pereira, E.M., Cardoso, J.S.: Machine learning interpretability: A survey on methods and metrics. Electronics **8**(8), 832 (2019)
16. Caton, S., Haas, C.: Fairness in machine learning: A survey (2020)
17. Celis, L.E., Huang, L., Keswani, V., Vishnoi, N.K.: Classification with fairness constraints: A meta-algorithm with provable guarantees. In: Proceedings of the conference on fairness, accountability, and transparency. pp. 319–328 (2019)
18. Chakraborty, J., Majumder, S., Yu, Z., Menzies, T.: Fairway: A way to build fair ml software. In: Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. pp. 654–665 (2020)
19. Chen, L., Ali Babar, M., Nuseibeh, B.: Characterizing architecturally significant requirements. IEEE Software **30**(2), 38–45 (2013). https://doi.org/10.1109/MS.2012.174
20. Chen, Z., Zhang, J.M., Hort, M., Sarro, F., Harman, M.: Fairness Testing: A Comprehensive Survey and Analysis of Trends (Aug 2022), http://arxiv.org/abs/2207.10223, arXiv:2207.10223 [cs]
21. Clifton, C.: Privacy Metrics. In: LIU, L., ÖZSU, M.T. (eds.) Encyclopedia of Database Systems, pp. 2137–2139. Springer US, Boston, MA (2009). https://doi.org/10.1007/978-0-387-39940-9_272, https://doi.org/10.1007/978-0-387-39940-9_272
22. d'Aloisio, G., Marco, A.D., Stilo, G.: Manila github repository (Jan 2023), https://github.com/giordanoDaloisio/manila
23. d'Aloisio, G., Marco, A.D., Stilo, G.: Manila zenodo repository (Jan 2023). https://doi.org/10.5281/zenodo.7525759, https://doi.org/10.5281/zenodo.7525759
24. Di Sipio, C., Di Rocco, J., Di Ruscio, D., Nguyen, D.P.T.: A Low-Code Tool Supporting the Development of Recommender Systems. In: Fifteenth ACM Conference on Recommender Systems. pp. 741–744. ACM, Amsterdam Netherlands (Sep 2021). https://doi.org/10.1145/3460231.3478885, https://dl.acm.org/doi/10.1145/3460231.3478885
25. Domingos, P., Pazzani, M.: On the Optimality of the Simple Bayesian Classifier under Zero-One Loss. Machine Learning **29**(2), 103–130 (Nov 1997). https://doi.org/10.1023/A:1007413511361, https://doi.org/10.1023/A:1007413511361
26. d'Aloisio, G., D'Angelo, A., Di Marco, A., Stilo, G.: Debiaser for Multiple Variables to enhance fairness in classification tasks. Information Processing & Management **60**(2), 103226 (Mar 2023). https://doi.org/10.1016/j.ipm.2022.103226, https://www.sciencedirect.com/science/article/pii/S0306457322003272
27. Feldman, M., Friedler, S.A., Moeller, J., Scheidegger, C., Venkatasubramanian, S.: Certifying and Removing Disparate Impact. In: Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 259–268. ACM, Sydney NSW Australia (Aug 2015). https://doi.org/10.1145/2783258.2783311, https://dl.acm.org/doi/10.1145/2783258.2783311
28. Friedman, J.H.: Stochastic gradient boosting. Computational statistics & data analysis **38**(4), 367–378 (2002), publisher: Elsevier
29. Galindo, J.A., Benavides, D., Trinidad, P., Gutiérrez-Fernández, A.M., Ruiz-Cortés, A.: Automated analysis of feature models: Quo vadis? Computing **101**(5), 387–433 (May 2019). https://doi.org/10.1007/s00607-018-0646-1, http://link.springer.com/10.1007/s00607-018-0646-1
30. Giray, G.: A software engineering perspective on engineering machine learning systems: State of the art and challenges. Journal of Systems and Software p. 111031 (2021)

31. Goncalves Jr., P.M., Barros, R.S.M.: Automating data preprocessing with dmpml and kddml. In: 2011 10th IEEE/ACIS International Conference on Computer and Information Science. pp. 97–103 (2011). https://doi.org/10.1109/ICIS.2011.23
32. Hamada, K., Ishikawa, F., Masuda, S., Myojin, T., Nishi, Y., Ogawa, H., Toku, T., Tokumoto, S., Tsuchiya, K., Ujita, Y., et al.: Guidelines for quality assurance of machine learning-based artificial intelligence. In: SEKE. pp. 335–341 (2020)
33. Hardt, M., Price, E., Price, E., Srebro, N.: Equality of Opportunity in Supervised Learning. In: Advances in Neural Information Processing Systems. vol. 29. Curran Associates, Inc. (2016), https://proceedings.neurips.cc/paper/2016/hash/9d2682367c3935defcb1f9e247a97c0d-Abstract.html
34. He, X., Zhao, K., Chu, X.: Automl: A survey of the state-of-the-art. Knowledge-Based Systems **212**, 106622 (2021). https://doi.org/https://doi.org/10.1016/j.knosys.2020.106622, https://www.sciencedirect.com/science/article/pii/S0950705120307516
35. Ishikawa, F.: Concepts in quality assessment for machine learning-from test data to arguments. In: International Conference on Conceptual Modeling. pp. 536–544. Springer (2018)
36. ISO: ISO/IEC 25010:2011. Tech. rep. (2011), https://www.iso.org/cms/render/live/en/sites/isoorg/contents/data/standard/03/57/35733.html
37. Kamiran, F., Calders, T.: Data preprocessing techniques for classification without discrimination. Knowledge and Information Systems **33**(1), 1–33 (Oct 2012). https://doi.org/10.1007/s10115-011-0463-8, http://link.springer.com/10.1007/s10115-011-0463-8
38. Kang, K.C., Cohen, S.G., Hess, J.A., Novak, W.E., Peterson, A.S.: Feature-oriented domain analysis (foda) feasibility study. Tech. rep., Carnegie-Mellon Univ Pittsburgh Pa Software Engineering Inst (1990)
39. Kearns, M., Neel, S., Roth, A., Wu, Z.S.: An empirical study of rich subgroup fairness for machine learning. In: Proceedings of the conference on fairness, accountability, and transparency. pp. 100–109 (2019)
40. Kumeno, F.: Sofware engneering challenges for machine learning applications: A literature review. Intelligent Decision Technologies **13**(4), 463–476 (2019)
41. Kusner, M.J., Loftus, J., Russell, C., Silva, R.: Counterfactual fairness. In: Advances in Neural Information Processing Systems. vol. 30. Curran Associates, Inc. (2017), https://proceedings.neurips.cc/paper/2017/hash/a486cd07e4ac3d270571622f4f316ec5-Abstract.html
42. Lim, T.S., Loh, W.Y., Shih, Y.S.: A comparison of prediction accuracy, complexity, and training time of thirty-three old and new classification algorithms. Machine learning **40**(3), 203–228 (2000), publisher: Springer
43. Linardatos, P., Papastefanopoulos, V., Kotsiantis, S.: Explainable ai: A review of machine learning interpretability methods. Entropy **23**(1), 18 (2021)
44. Liu, J., Pacitti, E., Valduriez, P., Mattoso, M.: A survey of data-intensive scientific workflow management. Journal of Grid Computing **13**(4), 457–493 (2015)
45. Martínez-Plumed, F., Contreras-Ochando, L., Ferri, C., Orallo, J.H., Kull, M., Lachiche, N., Quintana, M.J.R., Flach, P.A.: Crisp-dm twenty years later: From data mining processes to data science trajectories. IEEE Transactions on Knowledge and Data Engineering (2019)
46. Martínez-Fernández, S., Bogner, J., Franch, X., Oriol, M., Siebert, J., Trendowicz, A., Vollmer, A.M., Wagner, S.: Software Engineering for AI-Based Systems: A Survey. ACM Transactions on Software Engineering and Methodology **31**(2), 37e:1–37e:59 (Apr 2022). https://doi.org/10.1145/3487043, https://doi.org/10.1145/3487043

47. Mehrabi, N., Morstatter, F., Saxena, N., Lerman, K., Galstyan, A.: A Survey on Bias and Fairness in Machine Learning. ACM Computing Surveys **54**(6), 1–35 (Jul 2021). https://doi.org/10.1145/3457607, https://dl.acm.org/doi/10.1145/3457607
48. Menard, S.: Applied logistic regression analysis, vol. 106. Sage (2002)
49. Molnar, C.: Interpretable machine learning. Lulu. com (2020)
50. Muccini, H., Vaidhyanathan, K.: 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). pp. 121–128 (May 2021). https://doi.org/10.1109/WAIN52551.2021.00026
51. Nations, U.: THE 17 GOALS | Sustainable Development, https://sdgs.un.org/goals
52. Noble, W.S.: What is a support vector machine? Nature biotechnology **24**(12), 1565–1567 (2006), publisher: Nature Publishing Group
53. PalletsProject: Jinja website, https://jinja.palletsprojects.com/
54. Patro, S., Sahu, K.K.: Normalization: A preprocessing stage. arXiv preprint arXiv:1503.06462 (2015)
55. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine learning in Python. Journal of Machine Learning Research **12**, 2825–2830 (2011)
56. Putzel, P., Lee, S.: Blackbox Post-Processing for Multiclass Fairness. arXiv:2201.04461 [cs] (Jan 2022), http://arxiv.org/abs/2201.04461, arXiv: 2201.04461
57. Refaeilzadeh, P., Tang, L., Liu, H.: Cross-validation. Encyclopedia of database systems **5**, 532–538 (2009)
58. Refaeilzadeh, P., Tang, L., Liu, H.: Cross-Validation, pp. 1–7. Springer New York, New York, NY (2016). https://doi.org/10.1007/978-1-4899-7993-3_565-2
59. Refaeilzadeh, P., Tang, L., Liu, H.: Cross-Validation. In: Encyclopedia of Database Systems, pp. 1–7. Springer New York, New York, NY (2016). https://doi.org/10.1007/978-1-4899-7993-3_565-2
60. Rosenfield, G., Fitzpatrick-Lins, K.: A coefficient of agreement as a measure of thematic classification accuracy. Photogrammetric Engineering and Remote Sensing **52**(2), 223–227 (1986), http://pubs.er.usgs.gov/publication/70014667
61. Rönkkö, M., Heikkinen, J., Kotovirta, V., Chandrasekar, V.: Automated preprocessing of environmental data. Future Generation Computer Systems **45**, 13–24 (2015). https://doi.org/https://doi.org/10.1016/j.future.2014.10.011, https://www.sciencedirect.com/science/article/pii/S0167739X14002040
62. Sahay, A., Indamutsa, A., Di Ruscio, D., Pierantonio, A.: Supporting the understanding and comparison of low-code development platforms. In: 2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA). pp. 171–178. IEEE (2020)
63. Saleiro, P., Kuester, B., Hinkson, L., London, J., Stevens, A., Anisfeld, A., Rodolfa, K.T., Ghani, R.: Aequitas: A bias and fairness audit toolkit. arXiv preprint arXiv:1811.05577 (2018)
64. Siebert, J., Joeckel, L., Heidrich, J., Trendowicz, A., Nakamichi, K., Ohashi, K., Namba, I., Yamamoto, R., Aoyama, M.: Construction of a quality model for machine learning systems. Software Quality Journal pp. 1–29 (2021)
65. de Souza Nascimento, E., Ahmed, I., Oliveira, E., Palheta, M.P., Steinmacher, I., Conte, T.: Understanding development process of machine learning systems: Challenges and solutions. In: 2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM). pp. 1–6. IEEE (2019)

66. Studer, S., Bui, T.B., Drescher, C., Hanuschkin, A., Winkler, L., Peters, S., Müller, K.R.: Towards crisp-ml (q): a machine learning process model with quality assurance methodology. Machine Learning and Knowledge Extraction **3**(2), 392–413 (2021)
67. Taha, A.A., Hanbury, A.: Metrics for evaluating 3D medical image segmentation: analysis, selection, and tool. BMC Medical Imaging **15**(1), 29 (Aug 2015). https://doi.org/10.1186/s12880-015-0068-x, https://doi.org/10.1186/s12880-015-0068-x
68. Thüm, T., Kästner, C., Benduhn, F., Meinicke, J., Saake, G., Leich, T.: Featureide: An extensible framework for feature-oriented software development. Science of Computer Programming **79**, 70–85 (2014)
69. Tramer, F., Atlidakis, V., Geambasu, R., Hsu, D., Hubaux, J.P., Humbert, M., Juels, A., Lin, H.: Fairtest: Discovering unwarranted associations in data-driven applications. In: 2017 IEEE European Symposium on Security and Privacy (EuroS&P). pp. 401–416. IEEE (2017)
70. Villamizar, H., Escovedo, T., Kalinowski, M.: Requirements engineering for machine learning: A systematic mapping study. In: SEAA. pp. 29–36 (2021)
71. Xu, R., Baracaldo, N., Joshi, J.: Privacy-Preserving Machine Learning: Methods, Challenges and Directions. arXiv:2108.04417 [cs] (Sep 2021), http://arxiv.org/abs/2108.04417, arXiv: 2108.04417
72. Zhang, J.M., Harman, M., Ma, L., Liu, Y.: Machine learning testing: Survey, landscapes and horizons. IEEE Transactions on Software Engineering (2020)
73. Zhou, J., Gandomi, A.H., Chen, F., Holzinger, A.: Evaluating the Quality of Machine Learning Explanations: A Survey on Methods and Metrics. Electronics **10**(5), 593 (Jan 2021). https://doi.org/10.3390/electronics10050593, https://www.mdpi.com/2079-9292/10/5/593, number: 5 Publisher: Multidisciplinary Digital Publishing Institute