When DevOps Meets Meta-Learning: A Portfolio to Rule them all

Benjamin Benni, Mireille Blay-Fornarino, Sébastien Mosser, Frédéric Precioso, Günther Jungbluth

When DevOps meets Meta-Learning: A portfolio to rule them all

Benjamin Benni*, Mireille Blay-Fornarino*, Sébastien Mosser†, Frédéric Précioso*, Günther Jungbluth*
*Université Côte d'Azur, CNRS, I3S, France
{benni, blay, precioso, jungblunth}@i3s.unice.fr
†Université du Québec à Montréal, Canada
mosser.sebastien@uqam.ca

***Abstract*—The *Machine Learning* (ML) world is in constant evolution, as the amount of different algorithms in this context is evolving quickly. Until now, it is the responsibility of data scientists to create ad-hoc ML pipelines for each situation they encounter, gaining knowledge about the adequacy between their context and the chosen pipeline. Considering that it is not possible at a human scale to analyze the exponential number of potential pipelines, picking the *right* pipeline that combines the proper preprocessing and algorithms is a hard task that requires knowledge and experience. In front of the complexity of building a right ML pipeline, algorithm portfolios aim to drive algorithm selection, learning from the past in a continuous process. However, building a portfolio requires that (i) data scientists develop and test pipelines and (ii) portfolio maintainers ensure the quality of the portfolio and enrich it. The firsts are the developers, while the seconds are the operators. In this paper, we present a set of criteria to be respected, and propose a pipeline-based meta-model, to support a DevOps approach in the context of Machine Learning Pipelines. The exploitation of this meta-model, both as a graph and as a logical expression, serves to ensure continuity between *Dev* and *Ops*. We depict our proposition through the simplified study of two primary use cases, one with developer's point-of-view, the other with ops'.**

***Index Terms*—Machine Learning Pipeline, portfolio, generation, composition, meta-learning**

I. Introduction

Recent advances in *Machine Learning* (ML) have brought new solutions for the problems of prediction, decision, and identification. ML is impacting almost all domains of science or industry, but determining the right ML pipeline for a given problem remains a crucial question. Schematically, data processing in ML consists of 6 phases [14]: (1) understanding business objectives, (2) understanding data, (3) data preparation (*e.g.*, feature extraction, data cleaning), (4) modeling to train and calibrate ML algorithms, (5) evaluation to determine the best possible choices based on criteria to be defined, (6) deployment to operate the built pipeline. The first 5 phases, grouped under the term *development phase*, are repeated to evaluate the different possibilities and produce a family of models from which a predictive model is chosen. The deployment phase is also critical; it involves online evaluation, monitoring, and possibly model maintenance.

Faced with this complexity of building a right ML Pipeline, different complementary solutions are proposed: the DataOps movement to help with experiments, meta-learning to predict performances or dedicated environments such as algorithm portfolios to learn from the past in a continuous process.

One of the benefits of a portfolio is to have short production and feedback cycles. However, to build a portfolio, it is necessary to build ML pipelines whose evaluation results will be used to build a selection model, possibly by meta-learning. The evaluation of ML pipelines drives the meta-learning approach itself, whose input data contains the performance of the previously tested pipelines. Thus *data scientists develop* new pipelines that they test and compare on different datasets. *Portfolio managers* (often data-scientists themselves) add to the portfolio, algorithms, dataset and ML pipelines at the cost of numerous experiments and evaluations to implement meta-learning.

Complex mechanisms of quality validation, learning and feedback loops support this "reflexive" approach to portfolio construction, development, and integration. It is therefore essential to automate as many tasks as possible without losing knowledge in order to save not only time (experimentations can take several days) but also computing resources.

We propose to consider pipelines as first-class models and to use these models to build tools that facilitate a continuous and efficient process of solving a problem specific to continuous meta-learning. In this context:

- the *devs* are data scientists who focus on algorithms and their performance characteristics. They focus on on-time and quality delivery [6]. The challenge is to provide new and more "efficient" pipelines;
- The *ops* are those who maintain the portfolio; they are responsible for monitoring the portfolio itself. The challenge here is to put in place the tools to manage massive feedback from independent experiments, mainly to acquire new knowledge.

In this article, we have chosen to focus on pipeline construction and testing in a non-predefined context. *"... so far there is only little work on discovering new pipeline building blocks. Auto-sklearn uses a predefined set of preprocessings and classifiers in a fixed order. An efficient way to also come up with new pipelines would be helpful"*[1]. We show how we used the models to support related activities in the development of

[1]https://www.kdnuggets.com/2017/01/current-state-automated-machine-learning.html

the ROCKFlows [10] project, which aims to facilitate the automatic selection of ML pipelines according to a given problem. It is, of course, a question of statistics and machine learning, but it is, even more, a problem of *Software Engineering* (SE) practices including metamodelling. The contributions in this article focus on (1) the characterization of some criteria related to portfolio construction in this context, and (2) the use of a portfolio meta-model, the main element of which is the pipeline, to support these different criteria.

Section II positions this work and explains the concept of pipelines portfolio. Section III explains the use cases chosen to highlight the roles of Dev and Ops and to explain some of the requirements with which the portfolio must comply. The meta-model used as the basis for the portfolio is presented in section IV. We show in section V how it is used to meet the different criteria before concluding with our perspectives in section VI.

II. Related work

A. Automating Learning

Last years have seen an increasing effort from the big data companies (Amazon AWS, Microsoft Azure, Google AutoML...) to provide any user with simple platforms for designing tailored ML pipelines, allowing non-experts users to benefit from ML potential. However, none of these solutions consider the design of ML pipeline as a generic process intending to capture common processing patterns between pipelines (even through pipelines targeting different application contexts). These platforms either propose a set of dedicated solutions for given classes of problem (*i.e.*, AutoML Vision, AutoML natural language, AutoML Translation...) or propose a recipe to build your ML pipeline from scratch (*i.e.*, MS Azure Machine Learning studio, RapidMiner). However, to determine the right ML workflow for a given problem, numerous parameters have to be taken in account: the kind of data, expected predictions (error, accuracy, time, memory space), the choice of the algorithms and their judicious composition [16], [19]. In front of the complexity of choosing the "right" assembly, meta-learning offers an attractive solution, learning from the problems of the past. The algorithm selection problem is one of its applications [15]: given a dataset, identify which learning algorithm (and which hyperparameter setting) performs best on it.

B. Algorithm Portfolio

Figure 1, taken from [11], sketches a model for selecting algorithms in the line of Rice's work [15]. The selection model S is constructed using automatic learning techniques. The data for the model comes from the algorithms $A \in \mathcal{A}$, and the problems $x \in \mathcal{P}$, characterized by *meta-features*. S is created using learning data containing the *performance* of algorithms on a subset of the problems in the problem space. The S model predicts a specific algorithm A from a problem x. This algorithm is then used to solve the problem.

Algorithm Portfolio generalizes the problem and automates the construction of selection models [8]. The immediate goal is the same: to predict the results of the algorithms on a given problem without executing them. Even if, in the portfolio, meta-learning builds some selection models [4], the purpose is different: it is the systematic acquisition of knowledge about the algorithms it contains that drives its construction. The research then focuses on the quality and the return of knowledge, the acquisition process itself, and the construction of selection models over time.

However, these solutions focus on recommending a single algorithm, while it has been widely recognized that the quality of the results can be markedly improved by selecting the right workflows, *i.e.*, a complete chain of pre-processing operators and algorithms [17]. One of the additional challenges is then the growth of the search space. The capability of learning a relationship between data and a suitable algorithm is the premise of meta-learning and portfolio approaches. For this, it is essential to learn from past experiences such as, for example, databases of experiments (*e.g.*, OpenML [18]). The accuracy of the selection models depends then on the coverage of the problem area.

However, the space of problems and solutions presents a very great diversity even within a single class of problem like classification [5]. Also, the resources required for ML experiments are massive (time, memory, energy). Moreover, as the ML domain is particularly productive, the portfolio must be able to evolve to integrate new algorithms. To cope with the mass of data, the transformation of experimental results into knowledge requires the implementation of automatic analysis procedures.

III. Selected uses cases

The role of data preprocessing in ML is fundamental. Choosing the right combination of data preprocessing and predictive algorithms represents a significant amount of time in the development of predictive models [12]. The concern is to choose and apply, sequentially, different types of preprocessing on the same data (*e.g.*, missing value imputation, formatting data) before applying a predictive algorithm. Because there is no optimal solution for all problems [19], there are many different data preprocessing and predictive algorithms, grouped in different libraries (*e.g.*, scikit-learn, weka) and many ways to arrange them in an ML pipeline.

In the context of the portfolio, we are interested in two types of actors: (i) developers who define new *Machine Learning Pipelines* (MLPs), test them, evaluate them (cf. III-A); (ii) operational peoples who support the platform, (cf. III-B). This section relies on three personas to reify these actors: one on the *development* side (Lucas) and one on the *operational* side (Fred and Günther). Based on the needs of these personas, we identify thirteen criteria qualifying a model-based portfolio solution dedicated to ML.

Luca is a Master Data Sciences student. He seeks to compare several classification pipelines to analyze a dataset and is not an expert in SE. He mainly knows Python and R as programming languages and implements scripts.

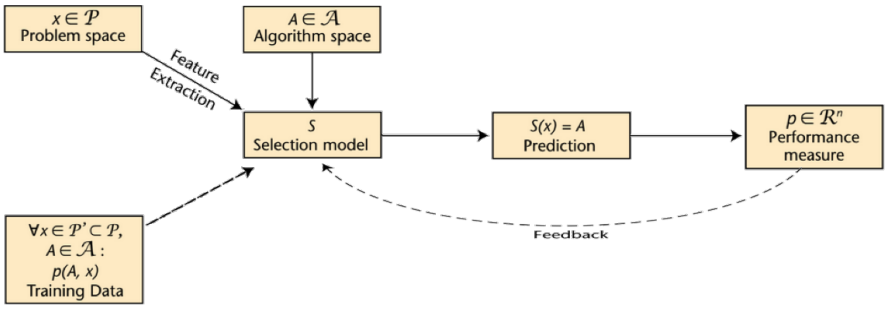

Fig. 1. Algorithm Selection Model illustrated by L. Kotthoff [11]

Fred is Professor, interested in building the portfolio, in facilitating access to the pipelines and in finding new properties on the pipelines. He has excellent expertise in machine learning but very little time. He must have confidence in the portfolio to be able to rely on it in his research results. He regularly explores new directions based on his research and the results obtained in the ML community. Even if he has a proper funding basis in his lab, he does not have the resources of a GAFA, and he is sensitive to energy problems.

Günther is an engineer, who maintains the portfolio. He regularly receives requests from the team of researchers and students to add new algorithms to the portfolio, new kinds of experiments. He is far from being an expert in ML and only works part-time on the portfolio.

A. Devs in action: as a data scientist, ...

a) Luca wants to build a pipeline to learn from a given dataset: and only pipelines *compatible* with the given dataset should be considered (c1). While the number of possible pipelines increases exponentially with the number of ML algorithms, Luca expects some help to eliminate *inefficient* pipelines according to his dataset (c2). Moreover, from energy consumption and time point of view, the choice of a pipeline should not require its execution. Luca has neither the time nor the resources to do so (c3). Finally, a pipeline might integrate ML components implemented in different languages. However, since Luca is not an expert in Java, he would like to avoid manipulating this language (c4).

b) Luca wants to evaluate a pipeline on a given dataset: The evaluation of a pipeline must include different performance measures (*e.g.*, predictive accuracy, the area under the ROC curve) (c5). He is also interested in having measurements on resources from a non-functional point of view (*e.g.*, run_memory, *_cpu_time) (c6). Finally, experiments must be reproducible (c7).

B. Ops in action: as a porfolio manager, ...

a) Günther wants to integrate a new algorithm in the portfolio: His goal when adding a new algorithm is to make it available to developers like Luca. To do this, he must collect knowledge about the algorithm. The portfolio should automatically launch experiments on selected data sets and automate the collection of results (c8). For this purpose, the system must select the data sets on which an algorithm can be applied. Unlike approaches that freeze all pipelines to a few known compositions, we want to be able to adapt the pipelines tested according to prior knowledge but also by testing new compositions (c9). The addition of a new prediction or preprocessing algorithm must, therefore, lead to the determination of all *consistent* pipelines and then to the execution of a subset of them on all compatible datasets.

b) Fred wants to capitalize on knowledge (meta-learning): Fred's objective is to use the experiments conducted by Günther and the devs to build a Selection Model (cf. Figure 1) by meta-learning and analysis of past experiment results. The choice of meta-features used by meta-learning is a challenge in itself: algorithms are sensitive to different meta-features, some are linked, others are expensive to calculate. Fred needs to be able to define new ones and exploit them easily (c10). The set of meta-features should evolve according to the knowledge acquired [1] (c11), and the knowledge acquired must be of high quality, based on trust in experiments (c12). The vast number of experiments requires (1) to automate the validation of experiments, (2) to trace experiments and their validation. Defining the validity of an experiment (*e.g.*, it did not fail, the comparisons respect criteria such as working on the same fold decompositions) is beyond the scope of this article. However, the validation operation itself is expected to evolve according to the knowledge acquired, and the biases observed. Finally, to reduce search spaces, it is necessary to automatically identify properties such as inefficient algorithm compositions, commutativity (c13).

IV. Pipelines Modeling

A coarse-grained description of pipeline meta-model is depicted in Figure 2 using the class-diagram formalism.

A. Pipelines

The Pipeline part of this meta-model is directly inspired by ADORE Meta-model [13], and by transitivity by the BPEL language grammar expressiveness (classically used in the 00's to compose web services)

We define a *Machine Learning Pipeline* (`Pipeline`) as a sequence of ML algorithm calls (`Activity`). The last step

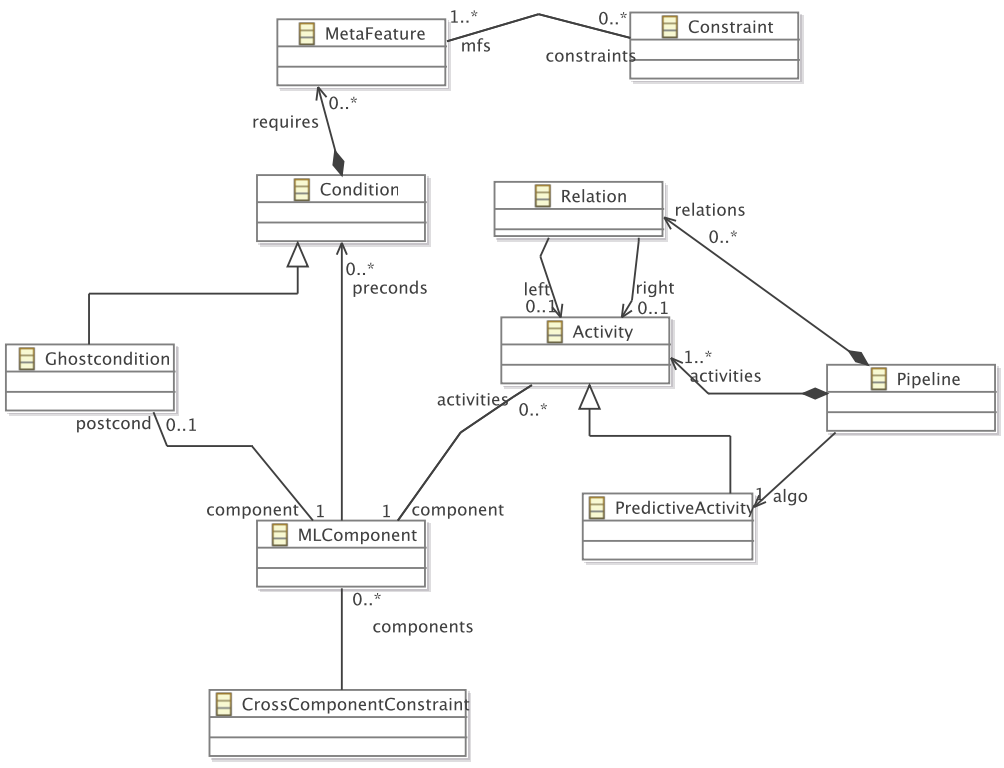

Fig. 2. Extract of the Pipeline meta-model

of a `Pipeline` refers to a `PredictiveActivity`. Some algorithms may not be directly suitable for a given dataset, *e.g.*, some classification algorithms cannot handle datasets containing missing values. The role of such preprocessing is to prepare the dataset and make it ready for the analysis by a predictive algorithm. A pipeline usually includes data cleaning, data imputation, and data transformation as preprocessing activities. In a portfolio, we need to distinguish the available `MLComponents` from calls to these components (`Activity`), like function calls are distinguished from function definition in classical programming languages. We simplify the relationship by considering a bijection between the calls and the components to lighten the formal model.

A pipeline is formally defined as: $mlp = pp_1; ...; pp_n; a$, where a is a predictive algorithm (a function taking as input a dataset and producing a model), pp_i is a preprocessing (a function taking as input a dataset and producing a new one as output), and ; a sequencing operator (meaning that the input to its right operand is the output of its left one).

B. Meta-features

To meet the criterion (c10), it is necessary to characterize a dataset by meta-features (*e.g.*, the number of classes, percentage of binary attributes). The choice of meta-features is itself critical, (1) because they can take a long time to calculate, and (2) pipelines are not equally sensitive to all meta-features. The choice of meta-features themselves evolves with our knowledge [1]. Considering a dataset denoted as d, we denote as f_d the extracted meta-features.

From the modeling perspective, a meta-feature corresponds to a variable. By considering meta-features as logical variables, we will show how to infer conditions on datasets to determine those to which a pipeline can be applied (c8). A meta-feature may be free, *i.e.*, not bound to a specific value.

C. Preconditions

An algorithm might not be compatible with a given dataset. For example, the `NaiveBayesMultinomial` algorithm defined within the Weka library cannot handle datasets that are not complete.

To model this notion, we rely on preconditions associated with any `MLComponent`. In the metamodel, `Conditions` define constraints on meta-features. The preconditions associated with an ML component are extracted from the code libraries (*e.g.*, by statically analyzing the *capabilities* defined in the Weka library as Java annotations) or defined by the developer of the algorithm. In the long term, we plan to complete preconditions based on failures on experiments. The modelling of preconditions supports criteria (c1), (c8).

A precondition p_i is defined by a set of constrained meta-features denoted as $requires(p_i)$. A precondition p_i is verified by a set of meta-features f_d if $checks(p_i, f_d)$ is true. By construction, $checks(p_i, requires(p_i))$ is verified.

Let a a `MLComponent`, $preconds(a) = \{p_1...p_n\}$ is the set of preconditions associated to a. Then, a is said *compatible* with a meta-feature set f_d if $compatible(a, f_d)$ is verified, *i.e.*, for all the preconditions $checks(p_i, f_d)$ is true.

The function `requires` associated with a ML component returns the meta-features required by the component *i.e.*, $requires(a) = \bigcup_{i=1}^{n}(requires(p_i))$. By construction, we assume $compatible(a, requires(a))$ is always verified.

D. Postconditions

The cost of executing an ML component can be high, as can the size of the dataset. It is therefore essential not to have to run all pipelines to determine which ones are compatible with a given dataset (c3). At the same time, if we do not want to freeze the possible compositions (c9), we must be able to work on the specific effects of the components. We have introduced the notion of postcondition in the form of "ghost condition". A ghost condition is a condition that can emulate the normal execution of a program by calculating meta-features. The function `predict` associated with a `GhostCondition` returns the meta-features that should be modified when executing the linked component. Only one postcondition is associated with a component. It can be statically defined (*e.g.*, for preprocessing that replaces all missing values with the median value, the meta-features concerning the missing values are forced to take the value 0). In the case where the constraints depend on the input meta-features, the prediction function calculates the new set of meta-features (*e.g.*, Feature Selection Techniques induce at least that the number of features of the output dataset is lower than the number of features of the input dataset).

Emulating a preprocessing a on a set of meta-features f_d means to apply ghost condition prediction function and returning a new set of meta-features, denoted as $result(a, f_d) = predict(postcond(a), f_d)$.

E. Pipeline compatibility with a given dataset

Based on the previous definitions, we can now establish the compatibility of a dataset with a pipeline as follows, by generalizing the notion of compatibility defined at the activity level.

A pipeline $mlp = pp_1; ...; pp_n; a$ is *compatible* with a set of meta-features f_{d_0} if : $\forall i \in [1..n]$ $result(pp_i, f_{d_{i-1}}) = f_{d_i}$, $compatible(pp_i, f_{d_{i-1}})$ and $compatible(a, f_{d_n})$ are verified.

A pipeline mlp is *consistent* if it is compatible with the metafeatures required by its first preprocessing step (*i.e.*, $requires(pp_1)$).

F. Cross-Component Constraints

It is common knowledge that some combinations of preprocessing activities are useless or inefficient. For instance, the following sequence is useless: `nominal->numeric; numeric->nominal`, as it brings back the pipeline to square one at a syntactical level. At a business level, putting the preprocessing `attribute-selection` at the end is considered as bad practice by ML experts. So a pipeline must respect cross-component constraints such as:

- A given preprocessing *"is always at the beginning"*;
- A given preprocessing *"is always/never at the end"*;
- The preprocessing pp_x *"is applied after"* the preprocessing pp_y;
- The preprocessing pp_x *"is never used in the same pipeline than"* the preprocessing pp_y.

These constraints are required to get efficient pipelines. Machine learning experts define some of them based on their expert knowledge. The analysis of the pipeline graph correlated with the results of past experiments supports the obtention of the others. The aim here is to build knowledge of the compositions themselves. For example, we are currently interested in identifying similarities, and cliques/stables that correspond to algorithms that statistically should always or never be used together. These constraints contribute to criteria (c2) and (c13).

A pipeline is said *efficient* if it is *consistent* and all cross-component constraints are respected (c2).

V. Modelling the Portfolio

A coarse-grained description of the Portfolio meta-model is depicted in Figure 3 using the class-diagram formalism.

A. Architecture Elements

We describe here how the proposed meta-model can supports both *devs* and *ops* based on the requirements identified in section III.

The entry point of the portfolio is the datasets registered, and the ML components available. A registry stores their definitions, and an experiment database stores the result of meta-learning experiments used to enrich the knowledge of the portfolio contents.

1) Registries: The `MLComponentRegistry` stores the algorithms (preprocessing and predictive algorithms) as turnkey images. To support the multi-language criteria (c4), we used the Docker container technology to encapsulate an algorithm into a black-box image. These images contain the evaluation tools necessary to evaluate the algorithm (*e.g.*, performance measurement (c6)), and their intrinsic definition supports the reproducibility of an experiment (c7).

It is possible to associate to certain `MLComponent` a meta-learning model (`MLModel`) that support the prediction of the efficiency of a new pipeline for a given dataset. The selection model described in Fig. 1 is built thanks to these models.

2) Experiments and Pipelines: The pipelines known by the system, as well as the meta-learning experiments made on these pipelines are stored in a service modeled by the `XPBase` concept. It supports the querying of the pipeline set, and its evolution (*e.g.*, the addition of a new pipeline). We attach to each experiment `Justifications` [7], which assess the different artifacts that were used to reach the stored conclusions, in order to achieve the explainability of the results (c12).

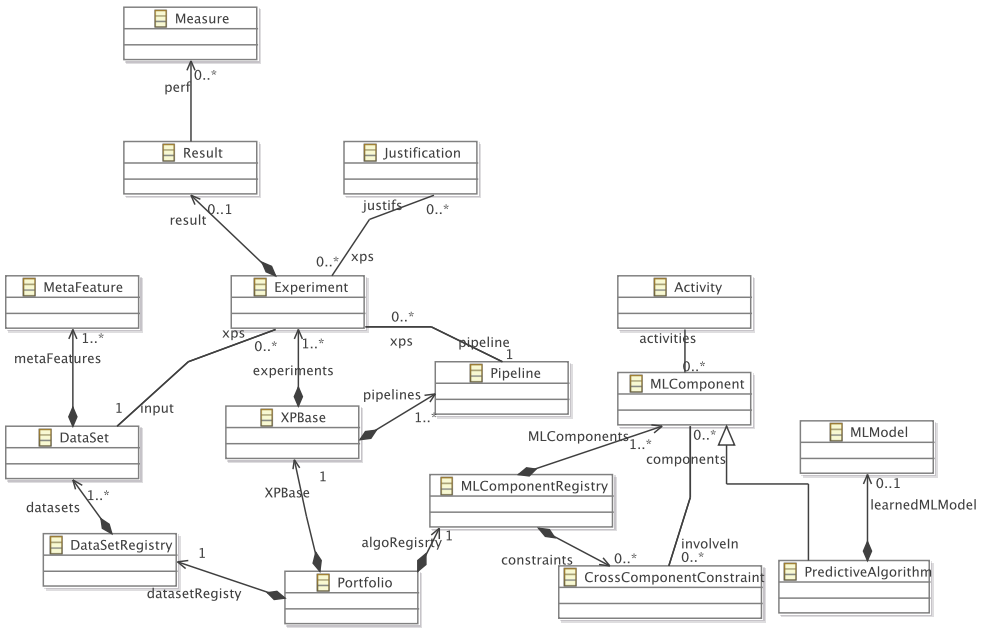

Fig. 3. Extract of the Portfolio meta-model

B. Supporting Ops

1) Integrating a new algorithm:

Composition Step : Given a new algorithm a, the portfolio will build all the efficient pipelines containing a. Cross-Component constraints play a crucial role in reducing this space of the composition. If this step does not result in at least one efficient pipeline, the new algorithm is not integrated in the portfolio as experiments are necessary to validate the newly available pipelines.

Experiment Preparation Step: If none of the known datasets in the portfolio are compatible with any of the pipelines produced in the previous step, then the new algorithm is also not integrated. It will be up to the data scientist to evaluate if there is an error in the definition of the algorithm in terms of pre- and post- conditions or if it is a problem with the dataset space that is not suitable.

The cartesian product of the compatible pipelines and the available datasets represents the space of experiments to be made to enrich the portfolio.

Experiment Selection Step: As the portfolio runs on limited resources, the experiments to be performed must be selected carefully from the set of available ones. The idea here is to select experiments to perform in priority those that are potentially the most valuable [9]. The description of a smart selection algorithm is a contribution by itself and is out of the scope of this paper. Instead, we consider here that all the experiments are required (*i.e.*, the selection step returns the whole space).

Experiment Execution Step : We use a controlled environment to launch each selected experiment in isolation, and the results are stored automatically. Tests are automatically triggered to check the accuracy of the experiments. These are of different natures such as the absence of errors, or the stability of execution times. It should be noted that these automatic tests are critical as they take place at all levels (*e.g.*, decomposition into folds, cleaning, learning, collection of measurements) and ensure the trustability of the portfolio. These different steps meet the criteria (c8), (c9) and contribute to (c12).

2) Meta-learning: The activation of meta-learning depends on the data collected by the experiments. For the moment it is triggered on demand, for example, because enough information has been collected for a new predictive algorithm, to update an ML model if new information has been collected, or because we have chosen a new way of learning. Meta-features are extracted from the experiments by dedicated programs, associated with the meta-features definitions. To reduce the costs of these calculations and improve the quality of predictions, the selection of meta-features is a work in progress [1]. The use of JSON as a pivotal model gives us confidence in our ability to evolve the tools.

These different points contribute to address the criteria (c10) and (c11). The tests and justifications produced during the experiments (cf. V-B1) are used to limit meta-learning to successful experiments only, contributing to (c12). Meta-learning is then based on meta-features extracted from valid experiments and produces regression models.

3) Analysis: We are considering different analyses to analyze the pipeline set, the regression models produced, and the results of experimentation, including failures. In particular, we seek to identify properties that allow cross-component constraints to be deduced based on the detection of commutativity, failure, or inefficiency of specific compositions. The results are automatically notified to the portfolio managers. This work

TABLE I
AMOUNT OF PIPELINES ACCORDING TO GIVEN DATASETS AND GIVEN CONSTRAINTS

Dataset	With pre/post conditions	With all constraints
empty	3,653,644	39,694
speed-dating [4]	3,419,170	36,360
texture [5]	256,286	5,688
iris [6]	105,892	3,190

aims to achieve criterion (c13).

C. Supporting Devs

1) Building a pipeline to learn from a given dataset: From the dev's point of view, given a dataset d, all pipelines that are compatible with d will be extracted from the portfolio (1) by looking for those that are compatible by browsing the pipeline graph and (2) by predicting for the pipelines their performance from models learned by meta-learning (c1). In this way we achieve criteria (c2) and (c3).

As dedicated docker components encapsulate `MLComponent`, pipelines can be composed of algorithms implemented in different languages (cf. criteria (c4)). The developer has then the choice of selecting the pipelines that suit him or her, taking into account the predictions obtained according to different criteria, which may be the performance or resources required.

2) Evaluating a pipeline on a given dataset: The evaluation of a pipeline is then carried out based on the containers. It can be done either in the platform itself, which it automatically feeds, or outside. Containers support pipeline evaluation (choice of evaluation method), performance reporting, and resource monitoring. To support the replication of an experiment, we define a composite container that contains the pipeline as a turn-key artifact. In this way, the criteria (c5), (c6), and (c7) are all verified.

D. First Results

We automatically extracted the pre-conditions of the Weka ML algorithms by using the provided capabilities[2]. We chose ten preprocessing algorithms from the Weka library and set their post-conditions according to their semantics. We extracted 94 algorithms with different enhancers[3] for a total of 483 algorithms. Enhancers are strategies to set hyper-parameters according to the input dataset.

The amount of potential pipelines for our extracted algorithms is 670,758,800. This number does not take in consideration pre/post conditions.

We observe a high variation of compatible pipelines for given datasets. The differences between the datasets partially explain this. For example, the *iris* dataset has no missing values, and the *speed-dating* has missing values, reducing by a factor the amount of pipelines.

[2]http://weka.sourceforge.net/doc.dev/weka/core/Capabilities.html

[3]http://weka.sourceforge.net/doc.dev/weka/classifiers/SingleClassifierEnhancer.html

[4]https://www.openml.org/d/40536

[5]https://www.openml.org/d/40499

[6]https://www.openml.org/d/61

VI. CONCLUSION AND PERSPECTIVES

We focused this contribution on the application of the *DevOps* paradigms to meta-learning, where *devs* are data scientists and *ops* algorithms portfolio maintainers. We proposed to generate Machine Learning Pipelines based on a set of preprocessing and learning algorithms without executing them when possible, saving time and resources. In order to limit this generation to pipelines that are consistent, we defined pre- and post-conditions as constraints on meta-features. We also defined cross-constraints to limit the amount of consistent pipelines. We verified the consistency of our approach on a set of algorithms extracted from the Weka library. Considering the identified use cases, we described how a model-driven approach that relies on the definition of carefully chosen abstractions can support both devs and ops in this domain. The approach is implemented, and a prototype is available [10].

More generally, we think that models are crucial elements in the DevOps paradigm, which are often neglected or hidden. In the context of machine learning, contrarily to the black-box approaches that only exposes a result, relying on models supports the explainability of the decisions to the user, supporting devs (who know why a given pipeline was chosen) and ops (who knows why an experiment is more valuable than another and should be prioritized). This approach brings to the meta-learning ecosystem the benefits of the DevOps paradigm, such as change reactivity (new algorithms produced by devs as well as experiments performed by the ops are enriching themselves), reliability (thanks to the reproducibility of the experiments and the shipment of the pipelines as turn-key containers), automation (the quality of the knowledge is ensured by automated measurement process).

At a software engineering level, one of our long-term objectives is to explore how models can be used in the context of DevOps to support the justification of decisions taken. In the context of machine learning, we showed that pre- and post-conditions highly reduce the space of consistent pipelines and are crucial to support the scalability of the approach. These conditions depend on the number of meta-features and can be improved by adding more meta-feature types and extending constraints expressiveness. In particular, one of our short-term objective is to look at how dataset generation strategies might allow improving the pre- and post-conditions of preprocessings [3] and extending them to automatically take new algorithms into account generating their pre and post-conditions. Cross-constraints also highly reduce the number of pipelines. We are currently working on machine learning environments such as OpenML [18] to automatically extract cross-constraints (*e.g.*, dominated algorithms [2]). To facilitate the addition of such cross-constraints by humans, we aim to build a Domain Specific Language dedicated to ML experts.

REFERENCES

[1] Besim Bilalli, Alberto Abelló, and Tomàs Aluja-Banet. On the predictive power of meta-features in OpenML. *International Journal of Applied Mathematics and Computer Science*, 27(4), 2017.

[2] Bernd Bischl, Pascal Kerschke, Lars Kotthoff, Marius Thomas Lindauer, Yuri Malitsky, Alexandre Fréchette, Holger H Hoos, Frank Hutter, Kevin Leyton-Brown, Kevin Tierney, and Joaquin Vanschoren. ASlib: {A} benchmark library for algorithm selection. *Artif. Intell.*, 237:41–58, 2016.

[3] Erwan Brottier, Franck Fleurey, Jim Steel, Benoit Baudry, and Yves Le Traon. Metamodel-based test generation for model transformations: An algorithm and a tool. In *Proceedings - International Symposium on Software Reliability Engineering, ISSRE*, 2006.

[4] Cécile Camillieri, Luca Parisi, Mireille Blay-Fornarino, Frédéric Precioso, Michel Riveill, and Joël Cancela Vaz. Towards a software product line for machine learning workflows: Focus on supporting evolution. In Tanja Mayerhofer, Alfonso Pierantonio, Bernhard Schätz, and Dalila Tamzalit, editors, *Proceedings of the 10th Workshop on Models and Evolution co-located with ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems (MODELS 2016), Saint-Malo, France, October 2, 2016.*, volume 1706 of *CEUR Workshop Proceedings*, pages 65–70. CEUR-WS.org, 2016.

[5] Manuel Fernández Delgado, Eva Cernadas, Senén Barro, and Dinani Gomes Amorim. Do we need hundreds of classifiers to solve real world classification problems? *Journal of Machine Learning Research*, 15(1):3133–3181, 2014.

[6] Erik Dornenburg. The Path to DevOps. *IEEE Software*, 2018.

[7] C. Duffau, C. Camillieri, and M. Blay-Fornarino. Improving confidence in experimental systems through automated construction of argumentation diagrams. In *ICEIS 2017 - Proceedings of the 19th International Conference on Enterprise Information Systems*, volume 2, 2017.

[8] Carla P Gomes and Bart Selman. Algorithm portfolios. *Artif. Intell.*, 126(1-2):43–62, 2001.

[9] Hans Degroote, Bernd Bischl, Lars Kotthoff and Patrick De Causmaecker. Reinforcement Learning for Automatic Online Algorithm Selection - an Empirical Study. In *ITAT 2016 Proceedings, CEUR Workshop Proceedings Vol. 1649*, pages 93–101, 2016.

[10] I3S. The ROCKFlows platform. http://rockflows.i3s.unice.fr, 2017.

[11] Lars Kotthoff. Algorithm Selection for Combinatorial Search Problems: {A} Survey. In Christian Bessiere, Luc De Raedt, Lars Kotthoff, Siegfried Nijssen, Barry O'Sullivan, and Dino Pedreschi, editors, *Data Mining and Constraint Programming - Foundations of a Cross-Disciplinary Approach*, volume 10101 of *Lecture Notes in Computer Science*, pages 149–190. Springer, 2016.

[12] Manuel Martin Salvador, Marcin Budka, and Bogdan Gabrys. Towards automatic composition of multicomponent predictive systems. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2016.

[13] S. Mosser and M. Blay-Fornarino. "Adore", a logical meta-model supporting business process evolution. *Science of Computer Programming*, 78(8):1035–1054, 2013.

[14] Giang Nguyen, Stefan Dlugolinsky, Martin Bobák, Viet Tran, Álvaro López García, Ignacio Heredia, Peter Malík, and Ladislav Hluchý. Machine Learning and Deep Learning frameworks and libraries for large-scale data mining: a survey. *Artificial Intelligence Review*, jan 2019.

[15] John R. Rice. The Algorithm Selection Problem. *Advances in Computers*, 15(C):65–118, 1976.

[16] Floarea Serban, Joaquin Vanschoren, Jörg-Uwe Kietz, and Abraham Bernstein. A survey of intelligent assistants for data analysis. *ACM Computing Surveys*, 2013.

[17] Jan N. Van Rijn and Joaquin Vanschoren. Sharing RapidMiner workflows and experiments with OpenML. In *CEUR Workshop Proceedings*, volume 1455, pages 93–103. CEUR-WS, 2015.

[18] Joaquin Vanschoren, Jan N. van Rijn, Bernd Bischl, and Luis Torgo. OpenML: Networked science in machine learning. *ACM SIGKDD Explorations Newsletter*, 2013.

[19] David H. Wolpert and William G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1997.