Evolvable SPL management with partial knowledge: an application to anomaly detection in time series

Yassine El Amraoui, Mireille Blay-Fornarino, Philippe Collet, Frédéric Precioso, Julien Muller

Evolvable SPL management with partial knowledge: an application to anomaly detection in time series

Yassine El Amraoui
Université Côte d'Azur, CNRS, I3S
EZAKO
Sophia Antipolis, France
yassine.elamraoui@ezako.com

Mireille Blay-Fornarino
Université Côte d'Azur, CNRS, I3S
Sophia Antipolis, France
Mireille.blay@univ-cotedazur.fr

Philippe Collet
Université Côte d'Azur, CNRS, I3S
Sophia Antipolis, France
philippe.collet@univ-cotedazur.fr

Frédéric Precioso
Université Côte d'Azur, Inria, CNRS, I3S
Sophia Antipolis, France
frederic.precioso@univ-cotedazur.fr

Julien Muller
EZAKO
Sophia Antipolis, France
julien.muller@ezako.com

ABSTRACT

In Machine Learning (ML), the resolution of anomaly detection problems in time series presents a great diversity of practices as it can correspond to many different contexts. These practices cover both grasping the business problem and designing the solution itself. By practice, we designate explicit and implicit steps toward resolving a problem, while a solution corresponds to a combination of algorithms selected for their performance on a given problem. Two related issues arise. The first one is that the practices are individual and not explicitly mutualized. The second one is that choosing one solution over another is all the more difficult to justify because the space of solutions and the evaluation criteria are vast and evolve rapidly with the advances in ML. To solve these issues and tame the evolving diversity in ML, a Software Product Line (SPL) approach can be envisaged to represent the variable set of solutions. However, this requires characterizing an ML business problem through an explicit set of criteria and justifying one ML solution over all others. The resolution of anomaly detection problems is thus different from finding the best configuration workflow from past configurations but lies more in guiding the configuration towards a solution that may never have been studied before. This paper proposes an SPL approach that capitalizes on past practices by exploiting a variability-aware representation to detect new criteria and constraints when practices adopt different solutions to seemingly similar problems. We report on the evaluation of our approach using a set of applications from the literature and an ML software company. We show how the analysis of practices makes it possible to consolidate the knowledge contained in the SPL.

SPLC '22, September 12–16, 2022, Graz, Austria

ACM ISBN 978-1-4503-9443-7/22/09…$15.00
https://doi.org/10.1145/3546932.3547008

CCS CONCEPTS

• **Software product line**; • **Machine learning**; • **Configurations**;

KEYWORDS

Software Product Line, Machine Learning, Evolution, Metrics

ACM Reference Format:
Yassine El Amraoui, Mireille Blay-Fornarino, Philippe Collet, Frédéric Precioso, and Julien Muller. 2022. Evolvable SPL management with partial knowledge: an application to anomaly detection in time series. In *26th ACM International Systems and Software Product Line Conference - Volume A (SPLC '22), September 12–16, 2022, Graz, Austria.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3546932.3547008

1 INTRODUCTION

Building learning systems are increasingly complex, as industry data, human and organizational factors, and application domains define different contexts that require tailored practices[11]. By practices, for the Machine Learning (ML) community, we mean the entire process of producing ML workflows, from analyzing the customer's data, business goals, and constraints to delivering the ML model built by composing algorithms. To address this variability of contexts, data scientists are developing a great deal of expertise, including developing dedicated algorithms within companies and tracking the evolution of theory and practice through literature and collaborations with researchers. However, with the profusion of algorithms and the diversity of industry problems, connecting problems to appropriate practices requires increasing capabilities and resources.

To make this connection between real-world problems and undiscovered scientific knowledge, we chose to focus on time series anomaly detection, such as stock price outlier detection, which presents a wide variety of challenges and practices [11, 25]. Scientific knowledge in this area remains to be discovered as the data and application domains require developing new solutions. While building an ML model involves a composition of algorithms that takes a long time to design and test, the available experiments only partially cover the large variability of the domain, especially in industrial applications (see Section 2.1).

In this paper, we argue that a Software Product Line (SPL) approach allows for linking partial configurations of ML problems with appropriate workflows. The originality of the approach is to exploit past configurations to enrich the knowledge captured by the SPL. In this context, we identify the following functional requirements.

R1- Identifying similarities between partially described problems. Looking for similarities and differences with previous problems is a natural first thought for data scientists but remains a difficult task. Indeed, the nature of the source data often makes their characterization challenging, especially since precisely defining an anomaly in a time series can be difficult, including for the customer who delivered the data. Being able to deal with partially characterized source data is therefore mandatory, including proposing algorithms that will be able to manage the variability of the data sets, for example, when the type of anomaly is not known (singular points, global anomalies, or pattern anomalies [12]).

R2-Consolidating knowledge according to the evolution of practices. Considering new solutions (*i.e.*, new ML workflow) from the literature requires characterizing the boundaries of the problems targeted by these solutions. Thus, it is not only a matter of selecting new algorithms but also new criteria such as evaluation metrics or business requirements. [8, 17]. For instance, detecting anomalies in scarce resource environments such as IoT embedded systems, has an impact on the entire ML model production chain [10, 43]. To consolidate the knowledge, we must be able to compare applications. By *application*, we mean not only the solutions and their performances, but also the targeted *problems*, *i.e.*, the data and the business requirements. In particular, we want to identify data sets and business requirements that appear similar but have different solutions to highlight new criteria or the obsolescence of some past criteria. For example, in the literature, when, for the same dataset, two different compositions of algorithms work well, it is interesting to identify, if possible, which requirement criteria could distinguish them.

To highlight these requirements, we propose three scenarios in which Lucile, a data scientist persona, uses our framework named **ROCK'n RWL**[1](RRW).

Scenario 1:Lucile uses RRW to search for a solution to a new anomaly detection problem over a given dataset. Through a dedicated interface, Alice indicates the business requirements and some additional information about the dataset. RRW narrows the solution space to the suitable components and selects the most relevant ones. In addition, if previous *applications* match the same criteria, RRW helps Alice compare and browse them as her analysis evolves.

Scenario 2: Lucile wishes to enrich the SPL with a new set of applications that she considers interesting. RRW analyses the information related to these *applications*. After checking that it does not contradict previous knowledge, RRW makes them available to other data scientists.

Scenario 3: Lucile wants to evaluate the SPL.. RRW informs her of the equivalences between the descriptions of the data sets, the business requirements, and the solutions of the various registered *applications*. RRW can then draw her attention to various issues. For example, RRW points out *applications* that address similar problems, i.e., seemingly identical business requirements and dataset characteristics, but use different solutions. It also identifies criteria that are never used or always used. These warnings are intended to help identifying new data spaces to be tested, for example, new criteria for comparing problems, and solution updates.

Research Vision and approach. The exploratory nature of machine learning makes an exhaustive analysis of the domain difficult, if not impossible. According to Drummond [17] and from our own experience, *"any advantage indicated by a simple scalar measure may be illusory if it hides situation-dependent performance differences."* Despite the generalization power of ML and the substantial evolution of the field, we advocate that SPLs are well suited to explore this complexity of dependencies between data, business goals, and algorithm composition. Our contribution then concerns:

- Exploiting the SPL to guide the data scientist in narrowing the solution space and more easily pinpointing past solutions that solved similar problems.
- Leveraging the SPL to reason about past solutions, making new knowledge explicit, and exploiting it to consolidate the SPL.

The principle is then the following. The configurations of the *applications* in the SPL incrementally capture our partial knowledge of the problems and solutions. The SPL progressively supports capitalizing on what is not explicit by reasoning about these configurations. Then the main issue is not to determine the configuration workflow that best suits the actors according to the previous configurations [45], but to guide them in composing a solution for an unprecedentedly studied problem. Concurrently, it is not a question of generating random samples [23], whose relevance could not be precisely verified (*e.g.*, stuffing the SPL with all the available algorithms and pre-processing components from the literature). Instead, it is more a matter of enriching our knowledge by systematically studying new validated configurations.

This paper shows how we apply these principles in constructing an SPL for anomaly detection in time series. We explain the difficulties specific to this domain of ML and why we consider that an adapted SPL can help in its analysis (see Section 2). Then we present the principles of the SPL, particularly the patterns used to identify knowledge from past *applications* (see Section 3). We validate this proposal on the first three steps of the SPL construction. The first phase consists in building the SPL proactively by domain analysis. Then we enrich the SPL with practices extracted from the partner company, and a third phase adds some *applications* of OpenML [53] to the SPL. We then discuss the limits and perspectives of the approach (see Section 5) before concluding this paper.

2 FROM PARTIAL KNOWLEDGE TO AN SPL

Designing ML workflow has become essential to almost any scientific field with Deep Learning advances in the past decade. The field increased fast and in many directions based on different models, yet it is not a rich and shared knowledge enough to be organized and made accessible to non-experts. Most of the knowledge one can hold is partial, shared through non-conventional channels such as personal blogs of other data scientists, webinars, and tricks shared

[1] *Request* your *Own Convenient Knowledge* flow and *Run* your ML *WorkfLows*

orally during conferences. This situation does not help to adopt these new techniques efficiently, particularly in companies. From our point of view, software engineering should play a more vital role in solving this central issue.

2.1 ML Workflows for anomaly detection in time series

Defining anomalies in a given business context requires business goals and constraints to be precised. Depending on the anomaly detection problem, it remains challenging to construct appropriate workflows [4] because the interactions between the current data, the composition of algorithms, and the business requirements are substantial and not always well understood. Sculley et al. summarise these interactions as follows: *"changing anything, changes everything"* [47].

Events predicted as statistically abnormal by the model may not be relevant anomalies for the end-user, if they are unrelated to business requirements, as for instance sensor failures. In order for the model to distinguish relevant from non-relevant anomalies, selecting the proper data preparation algorithm is then part of the final ML workflow solution and often a crucial part of it.

Testing the variability of workflows is all the more problematic as the resources required to train models can be very important in terms of time, memory, computation, but also in terms of human investment, and not only from data scientists. Indeed, in the absence of a normality reference or threshold, decisions on whether values are abnormal or not have to be made by end-users, which is time-consuming. Therefore, it is essential to reduce the solution space to those more appropriate to solve the problem. However, identifying the problem itself can also be complex and resource-consuming. Thus, even if deep learning-based solutions were the solution to all problems [19], it would still be necessary to take into account the variability of upstream processing to prepare the data and downstream processing to maintain the models in production.

2.2 On meta-learning and AutoML

Automated machine learning methods (AutoML) have been proposed and are focusing the efforts of many industries and research teams [21]. However, most AutoML algorithms aim only at solving a specific problem on specific datasets and do not provide end-users with the ability to acquire reasoned knowledge. Therefore, these systems are high-value solution components that we have introduced into the SPL to solve specific problems. More generally, many SE4AI[2] works addressed the issue of classification workflow selection in a generic way like in the work of Martínez-Fernández et al. [34] or by using a meta-learning-based portfolio like in the work of Kerschke et al. [29] and Degroote et al. [14]. Furthermore, these automated approaches entail massive needs for computation, memory, and time resources. One standard solution is to limit the solution space on which to train: measure choice, algorithms, or composition of algorithms.

However, we aim at the contrary at enriching our SPL by regularly adding new algorithms, new business requirements criteria, etc. We also aim to help in the evaluation measure choice, according to the prediction performance [24] and ensure that the solutions deployed will scale in production [48], in particular, in anomaly detection, automation is difficult since end-users must validate anomalies, as explained in the previous section. We are advocating a reverse approach that is drastically less computationally, memory-intensive, and more suitable for scientific and reasoned knowledge acquisition directed by and for humans.

2.3 Towards an evolvable SPL

This work is based on a collaboration between academic researchers in software engineering and data scientists from a company providing ML workflows for business customers. The work thus targets various applications, involving diverse industrial datasets and business problems.

Despite the constantly evolution of the domain (not to say the volatility of the domain), *everything changes ... in an unpredictable way* [44], choosing to capitalize on the different solutions designed by data scientists through an SPL seemed to be the best option. Our interviews revealed that, based on their experience, the domain experts had already identified most of the main dimensions of variabilities and commonalities of their domain. However, the domain analysis quickly showed that a proactive adoption scenario was not suitable, it is not yet possible to strictly separate domain analysis from the practices of data scientists seeking to solve new problems. Therefore, we have opted for a "reactive adoption strategy" for the product line and managed its evolution by integrating the practices of data scientists [30].

Effectively managing the evolution of variant-rich software involves bridging the gap between software solutions and the capture of domain variability [28]. However, in our case, while data variability is partially identifiable automatically [8], the context variability is not entirely identifiable from software solutions. Our goal is therefore to obtain this non-explicit information with a minimum of manual effort. To this end, we introduce into the round-trip engineering process proposed by Promote-pl [30], a feedback phase on the content of the SPL itself. This consists in identifying, when integrating new *applications*, those that can provide new information by comparison with past *applications*. To meet this objective, we started from the following assumption: "Any customer can generate the software they want, as long as they can describe it in the SPL"[3]. We use this assumption as a postulate to investigate the practices, and to hopefully identify new knowledge. The principle is that if two equivalent descriptions of problems correspond to two different solutions, then it is not the same problem; otherwise, we would not know which software to generate. Thus integrating an *application* to the SPL involves interactions with data scientists only when it is not possible to distinguish the contexts that led to two different solutions.

Evolving an SPL in an *ad hoc* manner is error-prone because the configuration space is large and involves taking into account many interdependent artefacts. Thus, the definition of a reactive approach integrates the need to foresee *"a typical pattern for maintaining and evolving a product line during its lifetime"* [5]. In [49], while defining safe evolution the authors state *"that the resulting SPL must be able to generate products that behaviorally match all of the original SPL products"*. We are in a slightly simpler application

[2]Software Engineering for AI

[3]Loosely adapted from Henry Ford.

context since we aim at composing only a unique stated version of each algorithm. It is not the products' behavior that changes, but the logic of assembling the workflows that evolves. Our objective is, thus, to detect configurations that are no longer valid due to changes in the SPL, possibly because of a past error.

2.4 From the requirements to the SPL paradigm

According to newly identified practices, the only artefacts that evolve in our SPL are the feature model (*i.e.*, CUD[4] operations on features, feature groups, and constraints) and the assets by addition or removal of algorithms and workflows (*i.e.*, CD operations). The configuration knowledge does not evolve as such. It consists only of bijections between the algorithms' code and the corresponding feature. Analyzing the impact of the evolution operations on past configurations is part of our perspectives inspired by the preliminary work of Nieke et al. [36].

Besides, given the very high variability of the domain and its constant evolution, it is neither possible, at least for the time being, to consider sampling techniques for workflows on which to learn [27], nor to support the configuration process in an optimal way [37].

To answer the requirements stated in the introduction, we reformulate them into technical requirements (RT) for an SPL dedicated to the composition of ML workflows in the context of anomaly detection in time series. To meet *R1*, the solution search corresponds to configuring a feature model, producing a valid configuration, partial in the problem specification and complete in the solution definition (*RT1.1*). The configurations must be comparable on the subspaces: the dataset description, the business requirements, and the solution components (*RT1.2*). The evolution of knowledge-driven by practices (*R2*) requires that the criteria of the domain analysis evolve without impacting the solution space (*RT2.1*). We must set up comparison patterns among configurations corresponding to the *applications* to guide the discovery of new knowledge (*RT2.2*).

3 DESIGNING AN EVOLVABLE SPL WITH PARTIAL KNOWLEDGE

The RRW SPL defines a set of ML practices with well-defined variabilities and commonalities. A combination of features (*i.e.*, configuration) identifies each product, and results in an *application*, *i.e.*, an ML workflow with its performance, its evaluation strategy, its deployment environment, etc. The format of the *applications* varies from notebooks, references to runs in OpenML, and references in the company tool. The set of valid feature combinations is specified in a feature model (FM) whose structure aims at facilitating the SPL evolution. Implementation artefacts are essentially references to algorithms and workflow models expressed in BPMN [39]. Mappings express the relationships between solution features and these artefacts. The SPL supports the generation of BPMN workflows based on the selected workflow model and algorithms. Since the construction of the SPL depends on the new *applications* created, we set up different mechanisms to control its evolution.

We developed tools to validate the overall process: (i) configuration and search of past *applications* (configurator), (ii) generation of ML workflows (generator), (iii) integration of *applications* in the SPL (integrator), (iv) reconfiguration of past configurations (reconfigurator), (v) evaluation of the knowledge carried by the SPL concerning the recorded *applications* (analyzer). We only present the concepts in this article. The configurator dynamically reads the feature model and a CSV file with helpful information for presenting the features (question, description, links to external elements). To help data scientists understand the feature selection, we associate descriptions with the constraints related to the selected features during the configuration. The possibility to import/export configurations allows proceeding by enrichment and, in the case of reconfiguration, manually adapting the past problematic configurations. The reusable artifacts are then the previous experiments (*codes* and configurations to adapt), a set of algorithms and workflow models. The reusable artifacts are, therefore, the previous experiments (*codes* and *configurations* to adapt), a set of algorithms, and workflow models.

3.1 FM structure to tame the SPL evolution

We structure the knowledge captured by the FM according to six main concepts, which organized the top of the FM hierarchy as depicted in figure 1. Information about the data sources, the business requirements, and the solution are mandatory as they are required to identify new applications. Information sources, states, and applications help the user in her analysis; they are optional. The numbers correspond in our case study to the number of features under each branch. The following paragraphs detail the content of the FM.

Data set properties. `InitialData` branch of the FM characterizes the space of datasets containing time series. Some properties are automatically extracted from the dataset (the sampling frequency, the time series number of dimensions, the stationarity verdict,...), while we can only get others through interacting with the business expert, such as how to interpret the missing values. No outgoing constraints from this branch to another branch are allowed since the dataset properties do not inherently imply algorithms or business requirements (*RT2.1*). For example, in our case study, this branch under the feature `InitialData` contains 28 other features.

Business requirement characteristics. `BusinessRequirements` branch captures requirements, such as limited memory usage to comply with hardware constraints or the solution's ability to provide explanations.

Solution components & states. The `Solution` branch groups and structures the algorithms used for solving anomaly detection problems and the types of workflows used in learning and deployment. The `states` branch represents the states through which the data passes. We express preconditions and the impact of a solution component through constraints relative to a state. Therefore, we forbid solution components to refer directly to the features of the initial data set but only to the corresponding state (*RT2.1*). For instance, an algorithm can require the state of the data to be scaled but not need that to be the initial state of the data.

Application & sources. The `Application` branch is only used in the configurator to facilitate direct access to past *applications* by filtering them according to the initial problem or the components of the Solution used. Similarly, the `Sources` branch helps remember

[4] create (C), update (U) or delete (D) operation [33]

from which literature article some features and constraints have been extracted and who are the authors of the *applications*.

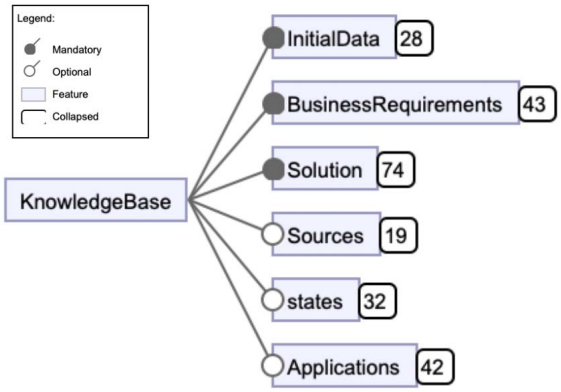

Figure 1: Feature Model Structure

3.2 Capturing knowledge through configuration management

To build and develop the SPL from past *applications*, we memorize the valuable elements to find the associated codes and the context in which they were defined.

Configurations. Configurations are our primary tool for determining *application* context. As we work on *applications* whose context is difficult to define and the SPL evolves, the configurations associated with the *applications* may be partial (*RT1.1*). Therefore, we consider any feature neither selected nor deselected as "unknown."

A partial configuration c is defined as a set of *selected*, *deselected*, and *undefined* features. The intersection is empty between these three subsets.
Let $[\![FM]\!]$ be the set of valid configurations of a feature model FM. A partial configuration c of a feature model FM is valid iff $\forall f_i \in c, f_i \in FM \wedge \exists c_k \in [\![FM]\!], selected(c) \subseteq selected(c_k) \wedge deselected(c) \subseteq deselected(c_k)$. By extension, we note $c \in [\![FM]\!]$. A configuration is complete relatively to a set of features F, when $\forall f_i \in F, f_i \in selected(c) \cup deselected(c)$. Thus, in RRW , configurations must be complete only relative to the `Solution` branch since we know whether the solution components are part of the *application* or not (*RT1.1*). In the following we refer to configurations, even for partial configurations.

To evaluate the evolution of our knowledge, we preserve the information on the manual or automatic selection/deselection. So we denote a configuration as a set of pairs: $(feature, status)$ where $status \in \{ms, md, as, ad, u\}$, where m for manual, a for automatic, s for selected, d for deselected, u for undefined. For example,
$c = \{(f1, ms), (f2, as), (f3, u), (f4, ad)\}$,
$selected(c) = \{f1, f2\}, deselected(c) = \{f4\}$

Data sets. For each dataset involved in an *application*, we preserve the associated partial configuration relative to the branch `initialData` (see Figure1). This record supports a consolidation mechanism. New *applications* dealing with known datasets should describe them in compliance with previous records and further complete them (*RT2.2*).

Required information about applications. The information associated with an *application* is its name (used as a reference), its initial configuration, the feature model used to define it, a reference to the dataset, its author, and the associated codes. The connection to the author makes it possible to identify the practices and preferences of the data scientists and allows a data scientist to reduce the space of the *applications* by authors. The reference to codes corresponds, to Jupyter NoteBooks, to runs in OpenML or to a workflow in the industrial partner platform.

3.3 Pattern detection based on the premise: a problem has a unique solution

In the same way as Tornhill [50], we seek to identify "hotspots" to narrow our study of *applications* to a few critical patterns that are most likely to guide us in the extraction of new knowledge (*RT2.2*). Therefore, the principle is that if two similar problems correspond to different solutions, then it is not the same problem.
We defined the first pattern: *2 problems evaluated as equivalent have a different solution.* This pattern allows us to detect different situations. (i) One of the solutions is not adapted to the problem, and in this case, in retrospect, the data scientist should not have used it. We must enrich the feature model to prohibit it. (ii) The two problems are different, but we had not yet identified these discriminative criteria in the feature model; we must enrich the feature model with these new criteria.

Because the configurations partially characterize the problems, we define a second pattern: *2 problems evaluated as unifiable have a different solution.* It can indeed be two similar problems partially filled in. In this case, we expect the same solution as before. But, the data scientist may also have designed a different solution to address the lack of information about the problem, such as not knowing the anomaly types.

The fact that several problems have the same solution can also induce an insensitivity of the solution to certain features. Despite the small number of *applications*, this situation allowed us to identify a feature that we apprehended at a level too detailed to be discriminating. Detecting these patterns occurs in Scenario 3 and meets the requirement *RT2.2*.

We now specify the notions of *equivalence classes* and their *unifiability*. To explain these concepts, we use the feature model presented in Figure 2 and the configurations described in Table 1. Table 3 shows the identified equivalence classes.

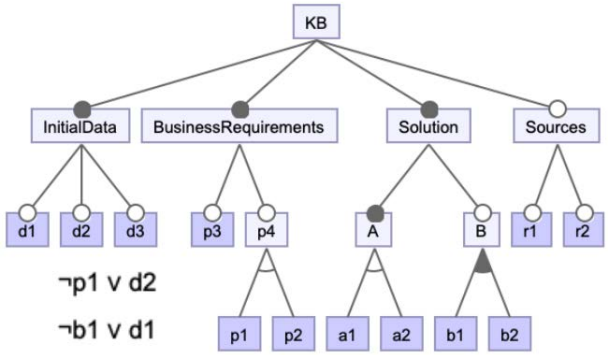

Figure 2: Feature model for explaining metrics

Table 1: Examples of *application* configurations.

XP Name	d1	d2	d3	p3	*p*4	p1	p2	a1	a2	b1	b2	*Sources*	r1	r2
app1	as	u	u	u	as	ad	ms	md	as	ms	md	u	u	u
app2	as	u	u	u	as	ad	ms	ad	ms	ms	md	u	u	u
app3	as	as	u	u	as	ms	ad	ad	ms	ms	md	u	u	u
app4	as	u	u	u	as	ad	ms	ms	ad	ms	md	as	ms	u

s=selected, d=deselected, u=undefined, a=automatic, m=manual

Table 2: Metrics related to the FM in fig. 2 and its configurations (tab. 1)

NoF	20	NoA	4
N_{leaf}	12	Cov	66 %
NoC	2	$NoEC$	3
$CTCR$	20 %	Com	37,5 %

Let a feature model FM and A a set of valid partial configurations of FM, $A \subseteq [\![FM]\!]$.

3.3.1 Equivalence Classes in A. An equivalence class on a subset of features F of FM is defined as a set of valid configurations $[c1] = \{c_1, ...c_k\}$, $c_i \in A$, such as $\forall f_i \in F$,
$f_i \in \cap_1^k selected(c_j) \bigcup \cap_1^k deselected(c_j) \bigcup \cap_1^k undefined(c_j)$.
In Table 3, two equivalence classes are identified on the `Initial-Data` subtree. `CDS1`={app1, app2, app4}. `CDS1` can also be noted: $\{(d1, s), (d2, u), (d3, u)\}$ and `CDS2`: $\{(d1, s), (d2, s), (d3, u)\}$

Problem equivalence classes are defined on the sub-features of `InitialData` and `BusinessRequirements`.
For example `CP1`={app1, app2, app4}.
$CP1 : \{(d1, s), (d2, u), (d3, u), (p1, d), (p2, s), (p3, u), (p4, s)\}$

Solution equivalence classes are defined on the sub-features of `Solution`.

Two configurations are equivalent in FM if they are member of the same Problem and Solution equivalence classes. In our example, `app1` and `app2` are equivalent.

We note $NoEC$ the number of equivalent classes.

3.3.2 Unifiable classes. Two equivalence classes $[c_1]$ and $[c_2]$ defined on a same set of features F are unifiable if $\forall f_i \in F$,
$f_i \in \cap_1^2(selected(c_j) \cup undefined(c_j))$
$\bigcup \cap_1^2(deselected(c_j) \cup undefined(c_j))$
In the example, CDS1 and CDS2 are unifiable.

3.4 Metrics for evaluating the SPL

To explain the following metrics, we use the feature model presented in Figure 2 and the configurations presented in table 1.

3.4.1 Feature model metrics. We selected some standard metrics [18] to assess the state of the feature model and, by comparison, its evolution.

The number of Features (NoF) and the number of features with no children (N_{leaf}) are a way to measure the scope of the SPL. Our objective is to integrate new solutions while identifying better and better the problems solved. We analyze these metrics in the different spaces. In our example in Figure 2 , the number of leaves is twelve overall, and it is four in the Solution subtree (Solution Space). The evolution of the number of cross-constraints (NoC), together with the tree-cross-constraint ratio ($CTCR$)[5], gives a numerical indication of the identified interactions. In our example, four features are involved in constraints, so the $CTCR$ is 20%. The theoretical number of possible configurations is not only not calculable but also does not provide any information. Indeed, it is likely that a part of the valid configurations does not correspond to suitable solutions. Moreover, this partial knowledge of the domain combined with the very high cost of ML workflows evaluations does not allow to `test` the SPL by generating examples, except to consume a lot of resources without any assurance of a real gain.

We now propose to evaluate the feature model FM according to the set of valid configurations $A \subseteq [\![FM]\!]$ that correspond to *applications* integrated in the SPL.

3.4.2 Feature-level metrics based on past configurations. Commonality of feature ($Com(f)$) indicates the selection ratio (manual or automatic) of a feature f in A. This ratio identifies the *"unused variability"* smell (*i.e.*, feature always selected, *e.g.*, $Com(b1) = 1$)[5]. The *rate of deselection* ($Des(f)$ identifies the *"unused feature"* smell (*i.e.*, feature always deselected, *e.g.*, $Com(b2) = 0, Des(b2) = 1$)[5]. We also compute the *rate of undefined occurrences*($Und(f)$) that may identify an obscure feature that is not well related to the scope of the SPL (*e.g.*, $Und(d3) = 1, Des(d3) = 0, Com(d3) = 0$).

We globalize these metrics to all feature model leaves, which in our case study characterize practices.

3.4.3 Feature Model Coverage. The Feature Model Coverage rate (Cov) measures (in percentage) the degree of leaf selections in a set of configurations A.
*Feature model coverage rate = number of feature leaves selected in A/number of feature leaves * 100.*
In our example, eight leaves are selected at least once, $Cov = 66\%$. For the Solution subtree, as three leaves were chosen at least one time, $Cov_{Solution} = 75\%$

Feature Model Coverage does not correspond to *t-wise* coverage [22]. Unlike the latter, it only provides a measure of the feature selection rate in a given set of configurations, it does not allow to assess the coverage of feature interactions. Nevertheless, it has the advantage of not requiring to compute the number of possible configurations.

3.4.4 Feature Model Commonality Rate. The Feature Model Commonality Rate (Com) measures (in percentage) the selection ratio of leaf selections in A.
*Feature model Commonality Rate = number of selection of feature leaves in A / number of feature leaves * #A * 100*, where $\#A$ denotes the cardinality of the set A.

In our example, 18 selections of leaves for 4 configurations and 12 leaves, $Com = 37,5\%$. $Com_{Solution} = 50\%$
Intuitively, the confidence in the feature model suggestions is proportional to its commonality rate.

[5]number of distinct features involved in cross-tree constraints and divides them through the total number of features in the feature model

Table 3: Equivalence classes and pattern detection.

App Name	InitialDSClass	InitialPBClass	SolutionClass	EquivalentApp	Warning	SameSolution
app1	CDS1	CP1	CS1	{app2}	{app4}	{app3}
app2	CDS1	CP1	CS1	{app1}	[]	[]
app4	CDS1	CP1	CS2	[]	{app1}	[]
app3	CDS2	CP2	CS1	[]	[]	{app1}

app1, *app2* and *app4* handles the same equivalence classes of dataset (CDS1) and problem (CP1). *app1* and *app2* are equivalent. *app3* deals with the same equivalence class of solutions (CS1) than *app1* (and therefore *app2*). While *app4* handles the same equivalence class of problem than *app1*, it proposes a new solution, a warning is raised. According to Table 1, CDS1 and CDS2 are unifiable.

3.5 Systematically mastering the evolutions of the SPL

To promote safe reactive development of the SPL, we suggest the following process.

At step T, the SPL is coherent, *i.e.*, all *applications* correspond to valid configurations of the feature model. We create new *applications* with the `configurator`. We integrate them to the SPL to make them available to other users (scenario 2, see Section 3.5.1). At step $T+1$, we update the feature model by adding, renaming, and removing features and constraints. It is then necessary to ensure that the configurations related to the previously developed *applications* remain valid and do not contradict the new knowledge captured by the feature model (see Section 3.5.2). We use the new feature model to build solutions to new problems. We use the patterns presented above, coupled with metrics to evaluate the SPL and detect new knowledge (see Section 3.5.3). We use the same means to analyze the evolution of the SPL (see Section 3.5.4).

3.5.1 Making applications identifiable in the configurator. Integrating in the feature model an *application* named app on a dataset named d and defined by a valid configuration c consists in adding, in the `Applications` branch of the feature model, the features d and app if they are not already there. Then, the minimal constraints[6] linking d to the selected and deselected features of the `initialData` space are added starting from the manually selected and unselected leaves. The constraint $app \Rightarrow d$ is then added. We then proceed in the same way to link app to the rest of the feature model, starting with the problem space. When the dataset d is already present in the feature model, there should be no contradiction with its constraints. However, they can be completed. This step corresponds to scenario two and is essential in scenario one to identify datasets or *applications* with the same features as the current configuration.

For example, adding the *application* $app5$ created by *John* on dataset $ds5$ and defined by the following configuration $\{(d2, as), (d3, ms), (p1, ms), (p2, ad), (a1, ms), (b1, md), (b2, ms), ...\}$ adds the features $app5$, $ds5$ and $John$ in the branch `Applications` and the following constraints: $ds5 \Rightarrow d3, app5 \Rightarrow ds5, app5 \Rightarrow p1, app5 \Rightarrow b2 \wedge \neg b1 \wedge a1, app5 \Rightarrow John$

3.5.2 Application-preserving refactoring against practice evolution. Refactorings of the feature model may lead to past *applications* (*i.e.*, their related configurations) being detected as conflicting with the current feature model [5]. To promote a safe evolution, a reconfiguration step is performed on all past configurations. For now, reconfiguring a c_s configuration into a c_t configuration with respect to a new feature model FM consists in (i) renaming in c_t some of the features of c_s, (ii) omitting the features that disappeared in FM with a warning if they were selected or deselected in c_s, (iii) adding in c_t the new features of FM whose value is known, (iv) copying in c_t the other features, then (v) replaying c_t in FM to obtain a new valid configuration or to raise an error in the contrary situation. If past configurations cannot be rendered valid in FM, RTFS excludes them with a warning. The new valid configurations related to *applications* can then be integrated into the FM using the previous operation.

For example, if we add the constraint $\neg b1 \vee \neg d2$ in FM of Figure 2, the configuration corresponding to $app3$ is no more valid, while all the other configurations are automatically updated with $(d2, ad)$.

3.5.3 Knowledge extraction driven by SPL assessment. Regarding scenario 3, the identification of the patterns presented above and the associated metrics allow us to evaluate the SPL to extract new knowledge and orient future evolutions, notably according to the spaces covered or not by the *applications*.

3.5.4 Knowledge extraction driven by SPL evolution assessment. The metrics and the detection of patterns also make it possible to evaluate the evolution of the SPL.

Have more features been used? Do unifiable problems become equivalent? Conversely, does the enrichment now allow us to distinguish previously equivalent problems? Both of these cases can occur when the addition of constraints affects previously undefined features.

4 APPLICATION

We now report on the first three steps of the SPL's construction, showing how the practices contributed to its enrichment. Figure 3 summarizes this construction process. The configurations and the results of the analyses are accessible online [7]

4.1 First three steps of the SPL construction process

In each of the steps presented below we have integrated the *applications* into the FM, which did not raise any significant issue.

Initial product line version from literature study. Following a first analysis of the domain, we built the SPL's initial version (SPL_{T^0}). The feature model (FM_{T^0}) integrates some solutions from the literature dealing with the detection of anomalies in time series. The

[6] features automatically selected or deselected during the configuration are not involved in new constraints

[7] https://anonymous.4open.science/r/RFTS-SPLC2022-D508/

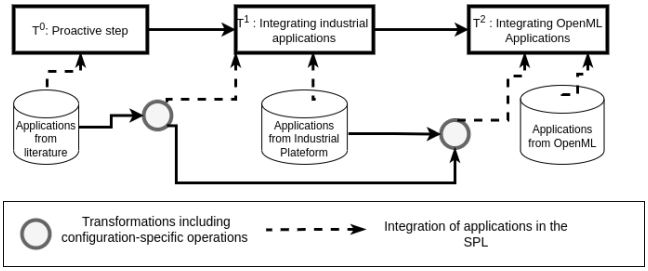

Figure 3: Three main steps of our SPL construction

applications correspond to experiments carried out on reference data sets [13].

Enrichment of the product line through industrial practices. At T^1, we leverage the practices of the partner company's data scientists to build SPL_{T^1} by enriching SPL_{T^0}. The interest in exploiting industrial *applications* is to broaden the scope of the SPL to the processing of industrial data. The industrial partner uses a custom tool to summarize all the *applications* on their customer's data. We update the feature model ($FM_{T^0} \rightarrow FM_{T^1}$) by including company-specific solution components, new initial dataset properties relevant to analyzing customer datasets, and new features necessary to describe the customer business requirements. Then we collect *applications* conducted by the company's data scientists, keeping only the solutions from deployed workflows and solving customers' anomaly detection problems. We have thus selected six workflows whose resulting product models are in production. The production of these workflows can take several months for the data scientists. We have generated partial configurations containing information about the dataset and solution based on automatic solution extraction and data analysis tools. We used these partial configurations to initialize the configurator. We then completed the source data and business requirements parts via a discussion with the *application* authors.

Consolidation by extraction of OpenML workflows. At T^2, we extract some practices from the OpenML platform. OpenML is an automated machine learning environment [53], from which ML practices can be downloaded and uploaded *i.e.*, solutions (runs and flows in OpenML) to a given problem (task and dataset in OpenML). The interest in exploiting OpenML's practices is to analyze the impact of upgrading the SPL with external sources. In OpenML, we selected time-series datasets and associated tasks of type *Supervised Learning* and *Unsupervised learning* since anomaly detection is supervised or unsupervised learning with unbalanced classes. We only had four datasets that matched these criteria. We kept 4 tasks of *Supervised learning* that had runs associated with them. Among these runs, we selected only the best runs on F1-score evaluation criterion as evaluations on other measures such as user CPU-time were not available for these runs. We preferred the runs using the scikitlearn library when we had the choice. We then extracted the associated flows and generated the associated partial configurations for each run. We had already studied in T^0 the meta-features proposed by OpenML to characterize datasets, so we only updated the feature model($FM_{T^1} \rightarrow FM_{T^2}$) by adding new solution components.

4.2 Knowledge extraction driven by SPL assessment

We explain in the following subsections how we exploit pattern and metric analysis in our use case.

Two different solutions for the same problem: algorithms side effects. At T^0, we encountered the following scenario. For two equivalent problems, the solutions used two different scaling techniques in each workflow, min-max scaler and robust scaler [40]. This equivalence of problems and not solutions raised a warning. We analyzed workflows for both experiments and observed that for the second workflow, the robust scaler results were equivalent to the min-max scaler results due to the data properties. In this scenario, we were able to confirm that the main particularity of the robust scaler was not required[8]. Therefore only the first application with min-max scaler was kept. We added a constraint to the selection of this algorithm to prevent the error from being repeated. *i.e.*, data without outliers will not anymore be scaled with robust scaler.

Two different solutions for the same problem: Data Scientist preferences impact. The data scientist's preferences bias her choice of the solution components. At T^1, we identified two applications that presented different solutions to equivalent problems. The two authors could not justify the difference in the choice of Solution components other than by their expertise in selected algorithms. Therefore, we have kept these two applications distinguishable by their author, with a warning for possible future treatment.

Two problems same solution: factorizing unnecessary variability. At T^0, two problems differ only in acquisition sampling; data acquisition sampling is in seconds for one and in microseconds for the second. Otherwise, the data are similar, and the anomaly detection requirements are equivalent. After the detection of this pattern, we checked the impact of acquisition sampling on the algorithms and factorized all four regular sampling features into `regularSampling` for the SPL at T^1.

4.3 Knowledge extraction driven by SPL evolution assessment

We also exploited the analysis of the evolution of patterns and metrics as another source of information. We use Pb_α and Pb_β to refer to the problem part of the configurations (*i.e.*, the features of the `InitialData` and `BusinessRequirements` branches) and S_α and S_β to refer to the solution part.

4.3.1 Pattern evolution and knowledge consolidation. At T^0, Pb_α and Pb_β are equivalent, but solved by two different clustering models[9], *kmeans* [26] on the one hand and *Dbscan* [46] on the other. At T^0, we did not know which to delete; we kept both *applications*. At T^1, we reconfigured the configurations to align with the new feature model, which now incorporates features detailing business expert insights into possible outliers in the data[10]. The feature model also includes associated constraints expressing compatibility between

[8]Usage of the robust scaler is interesting only if outliers are within the values of the time series
[9]Solution workflows vary according to machine learning algorithms
[10]The data scientists can decide whether outliers are anomalies or not in the context of the experiment

Table 4: Metrics Evolution in times and spaces

		NoF	N_{leaf}	*Cov*	*Com*	*NoEC*	*NoA*	*NoC*	*CTCR*
T^0	InitialData	23	16	37%	19%	5	-	-	-
	BusinessRequirements	33	24	41%	21,25 %	7	-	-	-
	Solution	51	25	52%	16.8 %	7	-	-	-
	Global*	156	96	35,04 %	14,68 %	10	10	25	21,19%
T^1	InitialData	28	19	42,10 %	18,94 %	9	-	-	-
	BusinessRequirements	43	33	54,55 %	17 %	13	-	-	-
	Solution	67	37	48,64 %	8,64 %	11	-	-	-
	Global*	194	124	40,32 %	10,86 %	14	15	31	21,76%
T^2	InitialData	28	19	57,9 %	18,00 %	14	-	-	-
	BusinessRequirements	43	33	57,6 %	17,24 %	17	-	-	-
	Solution	74	42	57,14 %	7,89 %	15	-	-	-
	Global*	203	131	47,32 %	10,41 %	18	19	32	21,78%

*The difference between the global figures and the figures of the 3 spaces corresponds to the branches `Sources` and `states`.
The feature model hierarchy is six levels deep for the `Solution` branch, and four for the `InitialData` and `BusinessRequirements` branches.

solution components and these new features. The reconfiguration made it possible to distinguish the two problems and the adequacy of the two different solutions.

4.3.2 Pattern evolution and knowledge extraction. Pb_α and Pb_β are equivalent in T^0, S_α includes a dimension reduction process through PCA [1] while S_β skips this step. Like in the previous example, we kept both *applications*. At T^1, we extended the `InitialData` space with features to explicit time series dimensionalities and automated their evaluation by dataset analysis. The reconfiguration step indicated that in Pb_α, the time series were multivariate. In contrast, Pb_β's time series were uni-variate [2]. This unique change in configuration highlighted the link between *PCA* and *time series dimensionalities*.

4.4 Exploiting the metrics

In sections 4.2 and 4.3, we established that the analysis of equivalence classes on both the problems and the solutions helps to trace the *applications* and their common points. We will now describe how the metrics defined in section 3.4 help us assess the evolution of the practices in each space.

`InitialData`. The coverage rate (*Cov*) increased from T^0 to T^1, while the number of features (*NoF*) also increased. This increase indicates that the industrial *applications* cover different data set properties from the first *applications* on benchmark datasets. Between T^1 and T^2 the coverage increased while the number of features did not change. New *applications* did involve new features of the InitialData. We rely on commonality analysis to better understand the variations between industrial and benchmark datasets. It shows that at T^0 all the features related to `Missingvalues` were *unused features*[11] which means that the datasets did not have missing values of any type. At T^1 `MCARMV`[12], and `StructuralMV`[13] features had a $com(f) > 1$, which means that the new datasets were exhibiting these two types of missing values. Similarly, we identify the emergence of irregular sampling time series at T^1.

`BusinessRequirements`. Within this feature space, we sought to identify the questions that experts answered the least. These questions may need rephrasing. The principle is then to identify the most undefined features of the penultimate level. We did not meet such a case yet, which was confirmed by the data scientists.

The coverage and commonality analysis highlight the requirements of industrial *applications* for memory, CPU, or energy consumption optimization. The features representing these hardware constraints are either undefined or deselected at T^0 and T^2. They are selected at T^1 only.

`Solution`. . The coverage rate decreases at step T^1 and increases at T^2, while the number of features increases strictly. The evolution of these two metrics indicates: (i) on the one hand, that industrial *applications* use new solution components; (ii) and on the other hand, that the *applications* we integrated at T^2 consolidate our SPL by reusing existing solution components. The commonality rate decreases to reach 7.89%. However, a detailed analysis of the number of selections by feature indicates that some algorithms are used in several solutions, while others are never used. For instance, we observe that each of `LSTMAE (LSTM Auto-encoder)` and `MAE (mean absolute error)` have been used 5 times out of 19, while `padding`, `FrontFill` and others have not been used. Therefore, correlated with broader coverage of problem space, this metric should help identify some of the preferences of data scientists and maybe some bias. Indeed, it is natural to think that data scientists generally rely on the algorithms they are comfortable with, sometimes maybe at the expense of the solution.

5 DISCUSSION

In this section, we relate our findings to existing work, and discuss potential threats to validity and current limitations.

Usability. . While we are confident that our SPL approach helps narrowing the problem, reducing the solution space, and identifying similar *applications*, these points remain to be proven through controlled experiments. To facilitate the use of the configurator, we rely on visualization techniques [41] since recommendation systems are not yet applicable [42, 51]. Yet, due to the increasing size and complexity of the feature model, one threat is that the configurator might become cumbersome to use because it exposed

[11] always deselected
[12] Missing value completely at random
[13] Missing values of structural nature

too many questions and too many possible solutions. Controlling the evolution of the feature model is therefore essential to avoid irrelevant questions and poorly fitting solution components. Metrics and patterns are part of the proposed solution to reduce this risk. Nevertheless, detecting patterns, especially those related to unification, can pose scalability issues on which we are currently working. Another threat to usability is related to the actual maintenance of the SPL in response to metrics and patterns analysis. These tasks were performed by the SPL modelers, interacting with the data scientists. This point does not challenge the relevance of the approach, but we still need to demonstrate that the tools allows autonomous maintenance by data scientists and collaborative FM updating [31].

Practice-driven feature modeling. To address the different perceptions of domain concepts, we not only unified domain terminology with descriptive feature names, but also provided descriptions and sources that are accessible directly from the configurator. However, we specified requirements only qualitatively (with propositional FM) using an ordinal scale when necessary, instead of their scalar values (*e.g.*, available memory greater than/lower than 1 GB) [7]. To automate and ensure reproducibility of reasoning between stakeholders, we scripted a mapping between time series metadata values and features. So far, these approximations have not hampered knowledge acquisition. Therefore, we did not need attributed feature models for which pattern detection has yet to be designed.

Practice-driven evolution. . Our work follows a reactive SPL adoption process [20, 28], using different techniques to locate features [16]. However, identifying the variations between workflows does not always enable us to understand the variations of the problem. The feature model then plays a crucial role in revealing undefined elements of the problem from the known constraints on the solutions. It is therefore essential that the FM be rich enough. We have demonstrated through our case study that we can enrich it with pattern detection. Yet, other complementary avenues still need to be explored to identify the relationships between solution components and source datasets. We are currently working on extracting the preconditions and effects of the algorithms by analyzing different techniques and ML environments [6, 35, 38, 52].

Quality assurance. . When the feature model is modified, we check, through automatic reconfigurations [51], that the previous configurations are preserved or even enhanced. These systematic checks have already allowed us to identify errors in the definition of new constraints. They participate in non-regression testing. However, SPL testing [15] and ML testing [55] are inherently difficult activities that we do not yet address; Many algorithms built into SPL are too resource-intensive (CPU, memory, and time) to consider sampling techniques [23]. Nonetheless, we believe that some work on SPL configurations opens up new opportunities to help build portfolios for automatic algorithm selection [29]. For example, configuration similarity analysis should help analyze the coverage of the problem space [3, 15, 27], while modeled features provide additional information to the metadata usually considered in meta-learning [32].

Generalizability. . External validity concerns the ability to generalize the results to other environments [54]. Our study has been developed in the context of one company, taking into account industrial applications. However, we have collected *applications* from three different sources, which mitigates the risk of dependency on the company's applications. Pattern detection relies on our ability to distinguish between the problem space and the solution space, the essence of any SPL. However, we decided to showcase our work on the particular context of this SPL (*i.e.*, focused on specific types of ML applications, with scientific knowledge yet to be discovered and with a small set of configurations) because it can be exploited industrially as is. We could generalize this approach to other systems as one of our most prized contributions is to build and evaluate an incremental SPL. However, t the particular context of this SPL (*i.e.*, focused on specific types of ML applications, with scientific knowledge yet to be discovered and with a small set of configurations) does not allow us to state that our contribution is generalizable. Nevertheless, several subdomains of ML at least present the same characteristics.

6 CONCLUSION

Recent technological advances have made possible to collect a large amount of data over time. The purpose of time series data mining is to enable classification, clustering, or outlier detection [9]. Our study focuses on this last task. In this paper, we have proposed a practice-driven approach to build an SPL as a first step toward allowing the design of generic solutions to detect anomalies in time series, while capturing new knowledge and capitalizing on the existing one.

The incrementality in the acquisition of knowledge and the instability of the domain [44] are supported by the SPL through its structuring and the exploitation of partial configurations associated with past *applications*. As far as we know, this is the first case of application of the SPL paradigm in such a context, and with a knowledge acquisition objective. We argue that using this paradigm to record and analyze practices will enable advances in the selection of ML workflows that are much less energy-intensive than meta-learning techniques, while assisting scientific knowledge production. By capturing practices in partial configurations, we obtain the abstractions to reason about datasets, solutions, and business requirements. The SPL is then used both to produce new solutions and compare them to past solutions, as well as to identify knowledge that was not explicit. The growing abstraction supported by the SPL also brings other benefits. In mentoring junior data scientists, we have observed a shift in the approach to creating ML workflows, focusing on analyzing problems before looking for similar applications, especially in choosing evaluation metrics. It is rather difficult for data scientists to explain the precise reasons for their choice. We observed that focusing only on particular cases identified as patterns makes the relevant criteria explicit.

This preliminary work paves the way for new software engineering contributions to ML. Our SPL is now evolving through the various works of data scientists to enrich the knowledge of anomaly detection in time series. We are working on visualization tools to facilitate the exploitation of practices, and thus the SPL maintenance. Distinguishing the users of the SPL from those who maintain it is also part of our future plan in order to obtain an empirical validation.

REFERENCES

[1] Hervé Abdi and Lynne J Williams. 2010. Principal component analysis. *Wiley interdisciplinary reviews: computational statistics* 2, 4 (2010), 433–459.

[2] Patrick Aboagye-Sarfo, Qun Mai, Frank M Sanfilippo, David B Preen, Louise M Stewart, and Daniel M Fatovich. 2015. A comparison of multivariate and univariate time series approaches to modelling and forecasting emergency department demand in Western Australia. *Journal of biomedical informatics* 57 (2015), 62–73.

[3] M Al-Hajjaji, T Thüm, J Meinicke, M Lochau … Software Product Line …, and undefined 2014. 2014. Similarity-based prioritization in software product-line testing. *dl.acm.org* 1 (sep 2014), 197–206. https://doi.org/10.1145/2648511.2648532

[4] Saleema Amershi, Andrew Begel, Christian Bird, Robert DeLine, Harald Gall, Ece Kamar, Nachiappan Nagappan, Besmira Nushi, and Thomas Zimmermann. 2019. Software Engineering for Machine Learning: A Case Study. In *Proceedings - 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice, ICSE-SEIP 2019*. IEEE, Montreal Quebec Canada, 291–300. https://doi.org/10.1109/ICSE-SEIP.2019.00042

[5] S Apel, D Batory, C Kästner, and G Saake. 2016. *Feature-oriented software product lines*. Springer. https://link.springer.com/content/pdf/10.1007/978-3-642-37521-7.pdf

[6] Benjamin Benni, Mireille Blay Fornarino, Sebastien Mosser, Frederic Precisio, and Gunther Jungbluth. 2019. When DevOps meets meta-learning: A portfolio to rule them all. In *Proceedings - 2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion, MODELS-C 2019*. Institute of Electrical and Electronics Engineers Inc., 605–612. https://doi.org/10.1109/MODELS-C.2019.00092

[7] Thorsten Berger, Steven She, Rafael Lotufo, Andrzej Wasowski, and Krzysztof Czarnecki. 2013. A Study of Variability Models and Languages in the Systems Software Domain. *IEEE Transactions on Software Engineering* 39, 12 (2013), 1611–1640. https://doi.org/10.1109/TSE.2013.34

[8] Besim Bilalli, Alberto Abelló, and Tomàs Aluja-Banet. 2017. On the predictive power of meta-features in OpenML. *International Journal of Applied Mathematics and Computer Science* 27, 4 (2017), 697–712.

[9] Ane Blázquez-García, Angel Conde, Usue Mori, and Jose A. Lozano. 2021. A Review on Outlier/Anomaly Detection in Time Series Data. *ACM Computing Surveys (CSUR)* 54, 3 (feb 2021), 33. https://doi.org/10.1145/3444690

[10] Sérgio Branco, André G Ferreira, and Jorge Cabral. 2019. Machine learning in resource-scarce embedded systems, FPGAs, and end-devices: A survey. *Electronics* 8, 11 (2019), 1289.

[11] Mikel Canizo, Isaac Triguero, Angel Conde, and Enrique Onieva. 2019. Multi-head CNN–RNN for multi-time series anomaly detection: An industrial case study. *Neurocomputing* 363 (2019), 246–260.

[12] Varun Chandola, Arindam Banerjee, and Vipin Kumar. 2009. Anomaly detection: A survey. *ACM computing surveys (CSUR)* 41, 3 (2009), 1–58.

[13] Hoang Anh Dau, Anthony J Bagnall, Kaveh Kamgar, Chin-Chia Michael Yeh, Yan Zhu, Shaghayegh Gharghabi, Chotirat Ann Ratanamahatana, and Eamonn J Keogh. 2018. The UCR Time Series Archive. *CoRR* abs/1810.07758 (2018), 1–12. http://arxiv.org/abs/1810.07758

[14] Hans Degroote, Bernd Bischl, Lars Kotthoff, and Patrick De Causmaecker. 2016. Reinforcement Learning for Automatic Online Algorithm Selection - an Empirical Study. In *ITAT 2016 Proceedings, CEUR Workshop Proceedings Vol. 1649 (CEUR Workshop Proceedings, Vol. 1649)*, Brona Brejová (Ed.). CEUR-WS.org, 93–101. http://ceur-ws.org/Vol-1649/93.pdf

[15] Xavier Devroey, Gilles Perrouin, Axel Legay, Pierre Yves Schobbens, and Patrick Heymans. 2015. Covering SPL behaviour with sampled configurations: An initial assessment. In *Proceedings of the Ninth International Workshop on Variability Modelling of Software-Intensive Systems*. ACM Press, Hildesheim, Germany, 59–66. https://doi.org/10.1145/2701319.2701325

[16] Bogdan Dit, Meghan Revelle, Malcom Gethers, and Denys Poshyvanyk. 2013. Feature location in source code: A taxonomy and survey. *Journal of software: Evolution and Process* 25, 1 (jan 2013), 53–95. https://doi.org/10.1002/SMR.567

[17] Chris Drummond. 2006. Machine learning as an experimental science (revisited). In *AAAI workshop on evaluation methods for machine learning*. AAAI Press, Phoenix, Arizona USA, 1–5. http://www.aaai.org/Library/Workshops/ws06-06.php

[18] Sascha El-Sharkawy, Nozomi Yamagishi-Eichler, and Klaus Schmid. 2019. Metrics for analyzing variability and its implementation in software product lines: A systematic literature review. *Information and Software Technology* 106 (feb 2019), 1–30. https://doi.org/10.1016/j.infsof.2018.08.015

[19] Dennis Elbrächter, Dmytro Perekrestenko, Philipp Grohs, and Helmut Bölcskei. 2019. Deep neural network approximation theory. *CoRR* abs/1901.02220 (2019), 1–43. http://arxiv.org/abs/1901.02220

[20] Stefan Fischer, Lukas Linsbauer, Roberto E. Lopez-Herrejon, and Alexander Egyed. 2015. The ECCO Tool: Extraction and Composition for Clone-and-Own. In *Proceedings - International Conference on Software Engineering*. IEEE, Florence, Italy, 665–668. https://doi.org/10.1109/ICSE.2015.218

[21] Xin He, Kaiyong Zhao, and Xiaowen Chu. 2021. AutoML: A survey of the state-of-the-art. *Knowledge-Based Systems* 212 (2021), 106622.

[22] Christopher Henard, Mike Papadakis, Gilles Perrouin, Jacques Klein, Patrick Heymans, and Yves Le Traon. 2014. Bypassing the combinatorial explosion: Using similarity to generate and prioritize t-wise test configurations for software product lines. *IEEE Transactions on Software Engineering* 40, 7 (2014), 650—-670. https://ieeexplore.ieee.org/abstract/document/6823132/

[23] Ruben Heradio, David Fernandez-Amoros, José A. Galindo, David Benavides, and Don Batory. 2022. Uniform and scalable sampling of highly configurable systems. *Empirical Software Engineering* 27, 2 (mar 2022), 44. https://doi.org/10.1007/s10664-021-10102-5

[24] Mohammad Hossin and Md Nasir Sulaiman. 2015. A review on evaluation metrics for data classification evaluations. *International journal of data mining & knowledge management process* 5, 2 (2015), 1.

[25] Jianglin Huang, Yan-Fu Li, and Min Xie. 2015. An empirical analysis of data preprocessing for machine learning-based software cost estimation. *Information and software Technology* 67 (2015), 108–127.

[26] Xiaohui Huang, Yunming Ye, Liyan Xiong, Raymond YK Lau, Nan Jiang, and Shaokai Wang. 2016. Time series k-means: A new k-means type smooth subspace clustering for time series data. *Information Sciences* 367 (2016), 1–13.

[27] Christian Kaltenecker, Alexander Grebhahn, Norbert Siegmund, Jianmei Guo, Sven Apel, and Michael Felderer. 2019. Distance-based sampling of software configuration spaces. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE Press, 1084–1094. https://doi.org/10.1109/ICSE.2019.00112

[28] T Kehrer, T Thüm, A Schultheis … Conference on Software …, and Undefined 2021. 2021. Bridging the gap between clone-and-own and software product lines. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)*. IEEE, 21–25. https://ieeexplore.ieee.org/abstract/document/9402254/

[29] Pascal Kerschke, Holger H. Hoos, Frank Neumann, and Heike Trautmann. 2018. Automated algorithm selection: Survey and perspectives. *Evolutionary Computation* 27, 1 (2018), 3–45. https://doi.org/10.1162/evco_a_00242 arXiv:1811.11597

[30] Jacob Krüger, Wardah Mahmood, and Thorsten Berger. 2020. Promote-pl: a round-trip engineering process model for adopting and evolving product lines. In *Proceedings of the 24th ACM Conference on Systems and Software Product Line: Volume A-Volume A*, Vol. Part F1642. Association for Computing Machinery, 263–273. https://doi.org/10.1145/3382025.3414970

[31] Elias Kuiter, Sebastian Krieter, Jacob Krüger, Gunter Saake, and Thomas Leich. 2021. variED: an editor for collaborative, real-time feature modeling. *Empirical Software Engineering* 26, 2 (mar 2021), 1–47. https://doi.org/10.1007/S10664-020-09892-X

[32] Luc Lesoil, Hugo Martin, Mathieu Acher, Arnaud Blouin, and Jean-Marc Jézéquel. 2022. Transferring Performance between Distinct Configurable Systems : A Case Study. *Proceedings of the 16th International Working Conference on Variability Modelling of Software-Intensive Systems* 6 (feb 2022), 1–6. https://doi.org/10.1145/3510466.3510486

[33] Maíra Marques, Jocelyn Simmonds, Pedro O. Rossel, and María Cecilia Bastarrica. 2019. Software product line evolution: A systematic literature review. , 190–208 pages. https://doi.org/10.1016/j.infsof.2018.08.014

[34] Silverio Martínez-Fernández, Justus Bogner, Xavier Franch, Marc Oriol, Julien Siebert, Adam Trendowicz, Anna Maria Vollmer, and Stefan Wagner. 2021. Software Engineering for AI-Based Systems: A Survey. *CoRR* abs/2105.0 (may 2021), 54. arXiv:2105.01984 https://arxiv.org/abs/2105.01984v1http://arxiv.org/abs/2105.01984

[35] Hoan Anh Nguyen, Robert Dyer, Tien N. Nguyen, and Hridesh Rajan. 2014. Mining preconditions of APIs in large-scale code corpus. In *Proceedings of the ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE)*. ACM, Hong Kong, China, 166–177. https://doi.org/10.1145/2635868.2635924

[36] Michael Nieke, Gabriela Sampaio, Thomas Thüm, Christoph Seidl, Leopoldo Teixeira, and Ina Schaefer. 2022. Guiding the evolution of product-line configurations. *Software and Systems Modeling* 21 (jul 2022), 225–247. https://doi.org/10.1007/S10270-021-00906-W/TABLES/5

[37] Lina Ochoa, Juliana Alves Pereira, Oscar González-Rojas, Harold Castro, and Gunter Saake. 2017. A survey on scalability and performance concerns in extended product lines configuration. In *Proceedings of the Eleventh International Workshop on Variability Modelling of Software-intensive Systems*. Association for Computing Machinery, 5–12. https://doi.org/10.1145/3023956.3023959

[38] Pascal Olz, Conny and Biundo, Susanne and Bercher. 2021. Revealing Hidden Preconditions and Effects of Compound HTN Planning Tasks–A Complexity Analysis. In *35th AAAI Conference on Artificial Intelligence (AAAI). AAAI Press*. AAAI Press, Virtual Event, 1903–11912. https://www.aaai.org/AAAI21Papers/AAAI-655.OlzC.pdf

[39] OMG. 2006. *Business Process Modeling Notation (BPMN) Specification, Final Adopted Specification*. Technical Report. Object Management Group (OMG).

[40] S. Gopal Krishna Patro and Kishore Kumar Sahu. 2015. Normalization: A Preprocessing Stage. *CoRR* abs/1503.06462 (2015), 1–3. http://arxiv.org/abs/1503.06462

[41] Juliana Alves Pereira, Sebastian Krieter, Jens Meinicke, Reimar Schröter, Gunter Saake, and Thomas Leich. 2016. FeatureIDE: Scalable product configuration of variable systems. In *International Conference on Software Reuse, Lecture Notes in*

Computer Science, Vol. 9679. Springer Verlag, 397–401. https://doi.org/10.1007/978-3-319-35122-3_27

[42] Juliana Alves Pereira, Pawel Matuszyk, Sebastian Krieter, Myra Spiliopoulou, and Gunter Saake. 2018. Personalized recommender systems for product-line configuration processes. *Computer Languages, Systems and Structures* 54 (2018), 451–471. https://doi.org/10.1016/j.cl.2018.01.003

[43] Nelishia Pillay, Rong Qu, Dipti Srinivasan, Barbara Hammer, and Kenneth Sorensen. 2018. Automated design of machine learning and search algorithms [guest editorial]. *IEEE Computational intelligence magazine* 13, 2 (2018), 16–17.

[44] Klaus Pohl, Günter Böckle, and Frank J van der Linden. 2005. *Software Product Line Engineering: Foundations, Principles and Techniques.* Springer-Verlag.

[45] Belén Ramos-Gutiérrez, Ángel Jesús Varela-Vaca, José A. Galindo, María Teresa Gómez-López, and David Benavides. 2021. Discovering configuration workflows from existing logs using process mining. *Empir. Softw. Eng.* 26, 1 (jan 2021), 11. https://doi.org/10.1007/s10664-020-09911-x

[46] Erich Schubert, Jörg Sander, Martin Ester, Hans Peter Kriegel, and Xiaowei Xu. 2017. DBSCAN revisited, revisited: why and how you should (still) use DBSCAN. *ACM Transactions on Database Systems (TODS)* 42, 3 (2017), 1–21.

[47] D. Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, and Michael Young. 2015. Machine Learning: The High Interest Credit Card of Technical Debt. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, Ghahramani Zoubin, Max Welling, Corinna Cortes, Neil D. Lawrence, and Kilian Q. Weinberger (Eds.). MIT Press, Montreal, Canada, 2503–2511. https://ai.google/research/pubs/pub43146

[48] Marina Sokolova, Nathalie Japkowicz, and Stan Szpakowicz. 2006. Beyond accuracy, F-score and ROC: a family of discriminant measures for performance evaluation. In *Australasian joint conference on artificial intelligence*. Springer, Springer, Berlin, Heidelberg, 1015–1021.

[49] Leopoldo Teixeira, Rohit Gheyi, and Paulo Borba. 2020. Safe Evolution of Product Lines Using Configuration Knowledge Laws. In *Brazilian Symposium on Formal Methods, Lecture Notes in Computer Science*, Vol. 12475 LNCS. Springer, Cham, 210–227. https://doi.org/10.1007/978-3-030-63882-5_13

[50] A Tornhill. 2015. *Your Code as a Crime Scene.* Pragmatic Bookshelf. https://books.google.fr/books?id=l7dDnQAACAAJ

[51] Mathias Uta, Alexander Felfernig, Viet Man Le, Andrei Popescu, Thi Ngoc Trang Tran, and Denis Helic. 2021. Evaluating recommender systems in feature model configuration. In *Proceedings of the 25th ACM International Systems and Software Product Line Conference*, Vol. Part F1716. ACM, New York, NY, USA, 58–63. https://doi.org/10.1145/3461001.3471144

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

[53] Joaquin Vanschoren, Jan N. van Rijn, Bernd Bischl, and Luis Torgo. 2013. OpenML: Networked Science in Machine Learning. *SIGKDD Explorations* 15, 2 (2013), 49–60. https://doi.org/10.1145/2641190.2641198 arXiv:1407.7722

[54] C Wohlin, P Runeson, M Höst, MC Ohlsson, and B Regnell. 2012. *Experimentation in software engineering.* Springer. 1–236 pages. https://books.google.com/books?hl=fr&lr=&id=QPVsM1_U8nkC&oi=fnd&pg=PR5&dq=Experimentation+in+Software+Engineering.&ots=GPx7rciRCu&sig=KyBLRUIbGY48ZlXMyE9nRVCbP_o

[55] Jie M. Zhang, Mark Harman, Lei Ma, and Yang Liu. 2022. Machine Learning Testing: Survey, Landscapes and Horizons. *IEEE Transactions on Software Engineering* 48, 01 (jan 2022), 1–36. https://doi.org/10.1109/TSE.2019.2962027 arXiv:1906.10742