Taming the Diversity of Computational Notebooks

Yann Brault, Yassine El Amraoui, Mireille Blay-Fornarino, Philippe Collet, Florent Jaillet, Frédéric Precioso

Taming the Diversity of Computational Notebooks

Yann Brault
yann.brault@etu.univ-cotedazur.fr
Université Côte d'Azur, CNRS, I3S
Sophia Antipolis, France

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

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

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

Florent Jaillet
florent.jaillet@univ-cotedazur.fr
Université Côte d'Azur, CNRS, I3S
Sophia Antipolis, France

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

ABSTRACT

In many applications of Computational Science and especially Data Science, notebooks are the cornerstone of knowledge and experiment sharing. Their diversity is multiple (problem addressed, input data, algorithm used, overall quality) and is not made explicit at all. As they are heavily reused through a clone-and-own approach, the tailoring process from an existing notebook to a specific problem is cumbersome, error-prone, and particularly uncertain. In this paper, we propose a tooled approach that captures the different dimensions of variability in computational notebooks. It allows one to seek an existing notebook that suits her requirements, or to generate most parts of a new one.

CCS CONCEPTS

• **Software and its engineering** → **Software product lines**; • **Computing methodologies** → *Machine learning*.

KEYWORDS

computational science, software variability, clone-and-own

1 INTRODUCTION

The development of Machine Learning (ML) brings multiple new challenges to the scientific community, including the discovery, reuse, and production of ML models [26, 41]. Designing a new ML model to solve an industrial ML task involves a series of iterations. These iterations consist in collecting data and business requirements from the domain expert, analyzing the data, transforming them accordingly to the domain expert's insights, then evaluating different workflow configurations [4, 40]. Besides code-sharing platforms such as GitHub or GitLab, many supporting environments aim to rationalize the ML life cycle, particularly by promoting the sharing of ML models. Thus, widely used platforms such as OpenML [37] or ModelDB [25] allow the specific sharing and discovery of ML models by scientists. Notebook sharing, more specifically, is widely supported by popular platforms such as Kaggle[1] (over 400K notebooks today), Google Colab[2], Jupyter Notebook, JupyterLab[3], or Baidu AI Studio[4].

[1]https://kaggle.com/
[2]https://colab.research.google.com/
[3]https://jupyter.org/
[4]https://aistudio.baidu.com/aistudio/index

Building effective data science systems is a challenging sociotechnical endeavor that involves both technical and human work [28]. Designing a universal solution that can work for any Machine Learning (ML) problem specification is not possible due to the high combinatorial diversity of both data [7] and business requirements [15]. Additionally, the pool of available ML solutions is continuously growing, making it increasingly difficult for data scientists to navigate this "wilderness" of possibilities [12, 41]. Consequently, a data scientist may, for the same problem, try several preprocessing libraries, several types of models (e.g., decision trees and neural networks), and even multiple frameworks for the same type of model (e.g., TensorFlow and PyTorch) [41]. A recent study [22] highlights the importance of code reuse in notebook production, with 18% of participants' time spent searching for code samples online. Duplication and reuse of code rely on tutorials or APIs and various sources because solutions are problem sensitive, and the field is evolving rapidly.

To the best of our knowledge, search options to retrieve notebooks are mostly based on a textual search (on domain keywords, benchmarks/competitions, machine learning artifacts, language, author, or dataset), possibly using ontologies as in the case of ModelDB [25]. Such search options could lead to various types of errors. For example, in the case of searching notebooks by keywords, this can result in an overwhelming number of notebooks, some of which are not even related to the field, without the problem being addressed. The key issue is then to reduce the solution space to the suitable solutions only. In this context AutoML-like approaches and recommender systems will learn from the past runs to find one *best* solution only according to the datasets, while this solution may not fit the current requirements [15]. Instead, we believe that a data science tool support could provide guidance in finding an appropriate set of solutions taking into account both datasets and requirements. It should provide a comprehensive view of the data science landscape, including the solutions already developed and their applicability to different problem specifications. Consequently, it should help them identify the effective solution set for their problems.

Facing the issues of clone-and-own in computational notebook usage (*cf.* section 2) this paper describes emerging ideas along with a possible tooled approach for resolving them. It captures the different dimensions of variability in computational notebooks and distinguishes two layers (*cf.* section 3). The upper layer focuses

on the problem's configuration (including data and business requirements), and the lower layer handles the solution retrieval. The adequacy of a notebook to a new problem is then more related to the problem encountered than to the keywords that define it. Our approach thus allows covering scenarios inspired by real cases (*cf.* section 4), such as retrieving a notebook if it exists or generating a new one depending on the configuration matching.

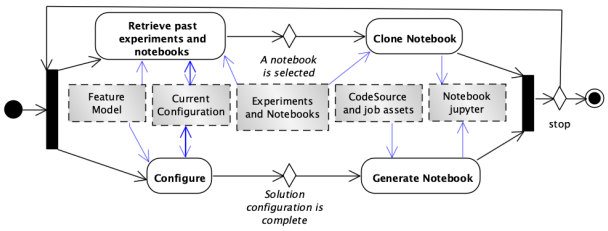


Figure 1: Functional overview of our approach

2 CONTEXT

The term *Data analysis* corresponds to an iterative and exploratory process aiming to extract information from data. Small changes in how data is collected, annotated, cleaned, or processed may lead to different results. Data scientists must document their data analysis and processing stages. This is even more crucial if someone else has to understand, maintain, or trust the initial work [33]. In this context, notebooks have clear benefits by providing storytelling through cells of code intertwined with inline documentation. Data scientists have widely adopted computational notebooks, even at the heart of the internal data analysis infrastructures of companies such as Netflix or IBM.

This success is due to a combination of code, result visualizations, and textual documentation, all in a single document. The most famous implementation is the Jupyter Notebook, an open-source project, motivating share and reuse [30]. However, the freedom offered by notebooks and the iterative and exploratory nature of building ML workflows affect their reusability. Several reasons have been devised, such as a need for more documentation on the workflow [21, 38], the difficulty of replicating the experiment, [39] and the diversity of problems. To tackle this, data scientists only clone the relevant parts and use them in their own workflows [22], thus following the *clone-and-own* practice [11, 20, 23, 32]. While data scientists find the same advantages in this approach as in previous empirical studies [11], the experimental nature of ML makes the approach even more compelling.

Cloning saves time, especially by helping to prepare data and fix parameters. Therefore, the ability to clone a notebook is crucial. The autonomy provided by cloning becomes essential in ensuring adaptability to new problem domains. Despite the advantages of the autonomy offered by cloning, managing the integration of cloned artifacts, such as cells or code segments, can be challenging, as it can result in an inappropriate combination of algorithms and impede the reuse process. Creating appropriate workflows remains challenging [3], as the interactions between current data, algorithm composition, and business requirements are substantial and not always well-understood. Passi and Sengers observed in [29] that *"building data science systems and making them work requires enormous subjective judgment"*. Data scientists rely on their familiarity with specific algorithms to choose the solution component compositions, even though identifying the preconditions for using these compositions can be challenging. Sculley et al. summarize these interactions as *"changing anything changes everything"* [34]. Therefore, it is not about defining configuration workflows as in [1, 2], as they do not align with the practices and needs. Instead, our approach aims to address the solution search process as a whole.

In summary, the development of computational notebooks by clone-and-own is a natural, necessary, and at the same time, a complex approach [22] involving heavy, error-prone, and particularly uncertain activities.

3 TAMING THE DIVERSITY

The field of ML is constantly evolving and highly diverse, which makes comprehensive domain engineering practically impossible. *Therefore, we represent the variability of the domain and reason about it while recognizing that our knowledge is inevitably incomplete, both in analyzing the domain and in identifying the problem to be solved.*

When facing a new problem, the suitability of a composition should be determined based on the triplet of data, business requirements, and ML artifacts compositions; a single requirement can render a composition unsuitable, while a composition can often be adapted to solve problems in different domains. *Therefore, it is essential to maintain relations and constraints between the notebooks (products) and their initial experiment specifications (variability configurations) while considering that the product space continually evolves. An existing notebook (product) may only sometimes match new specifications (configurations).*

3.1 Process overview

The main steps of our approach are summarized in Figure 1. The data scientist plays an active role in problem-solving by creating a configuration and exploring the repository of past experiments aligned with the problem's specifications. During the configuration process, the data scientist specifies the characteristics of her dataset and her business requirements, and selects the relevant ML artifacts that match her problem. When the configuration is complete, she can either clone an experiment (and notebook) with a similar problem specification or generate a new notebook if such a product does not exist. In the following, we detail our approach and refer to Figure 2 to explain the relationships between the spaces. We use italics to emphasize examples.

3.2 Domain variability model

The configuration of the domain variability model is done by the user during the step of `Configuration`, in the Figure 1. However, it is the user's choice to define in which order she wants to configure the domain. In any case, the retrieval of an experiment or a notebook is performed simultaneously (*cf.* Figure 1 as each user selection is propagated in the constraint system in order to reduce the space of suitable solutions).

*We rely on simplified examples of feature models in the domain of anomaly detection in time series to illustrate our points (*cf. *Figure 4). Some constraints connecting these spaces are illustrated in Figure 4e, with the corresponding arrows in Figure 2 denoted by circled numbers.*

3.2.1 Initial data. `InitialData` submodel characterizes the space of initial datasets. The data scientist configures this submodel through semi-automatic data analysis and manual selection of properties that depend on expert knowledge, such as the nature of missing values.

Figure 4a illustrates this submodel emphasizing the availability of data labels or annotations, the data type (which could comprise image data such as sound spectrograms) and the normalization status of the data prior to analysis.

3.2.2 Business requirements. `BusinessRequirements` submodel captures requirements, such as limited memory usage to comply with hardware constraints. The data scientist configures this submodel through a manual selection of requirements. While the initial data space does not impose restrictions on other feature models, specificities regarding the initial data, such as sampling rates, can be included as part of the business requirements. This dependency is illustrated by *arrow 1* in Figure 2.

Figure 4b illustrates this submodel highlighting three significant considerations. Firstly, the deployment platform during production should be taken into account, especially if it involves microcontrollers for embedded models. Secondly, an understanding of potential deviations between anomalies observed in the production environment and those represented in the training data is crucial. Lastly, experts' interest in the types of anomalies, such as outliers or single point amongst others should not be overlooked.

3.2.3 ML artifacts & states. `MLArtifacts` submodel organizes the algorithmic hierarchy and specifies the types of workflows used during the learning and deployment phases (*e.g.*, active learning workflow). On its side, the `States` submodel represents the different stages that the data passes through, with preconditions and the impact of ML artifacts expressed through constraints relative to each state. For example, an algorithm may require the data to be in a scaled state without necessarily needing that state to be the initial one. The `MLArtifacts` submodel is manually configured by the data scientist based on their specific problem and preferences. In contrast, the `States` submodel is only configured through the propagation of information about the data and ML artifact choices.

The separation of concerns in the ML artifacts model and Business Requirements model (*arrow 4* in Figure 2) facilitates this management of evolution. Additionally, the state of the data depends not only on its initial state (*arrow 2*) but also on its transformations and processing throughout the ML pipeline (*arrow 3*). The intermediate `States` model plays a critical role in managing this variability and ensuring the overall effectiveness of the ML solution.

Figure 4c illustrates the `MLArtifacts` *submodel, while Figure 4d illustrates the* `States` *submodel.* `MLArtifacts` *submodel comprises three primary types of solution components. Preprocessing elements such as mathematical transformations of time series, from time domain to frequency. Next, the algorithm components consist of features such as neural networks, support vector machines, and tree-based models. Finally, the postprocessing component exclusively includes the Quantizing algorithm, which serves to reduce the model size and facilitates embedding in microcontrollers.*

3.3 Variability of realized products and assets

We use the vocabulary of Idowu et al. in [17] to present the assets that enable us to find notebooks or generate a primitive version of a new one[5]. We structure the application space into two main areas: firstly, the code source and job assets, which represent possible products (notebooks), and secondly, the already realized products that we organize into variability subspaces `ExperimentProducts` and `NotebookProducts` for easy retrieval and potential cloning.

3.3.1 Code source and job assets. Code Source and Job Assets are the core of the generation (see Figure 1). They are mapped to the `MLArtifacts` submodel, in order to maintain a mapping between model and implementation. These assets do not contain the datasets as they are not used in the generation process. Technical details on generation are given in section 3.5.

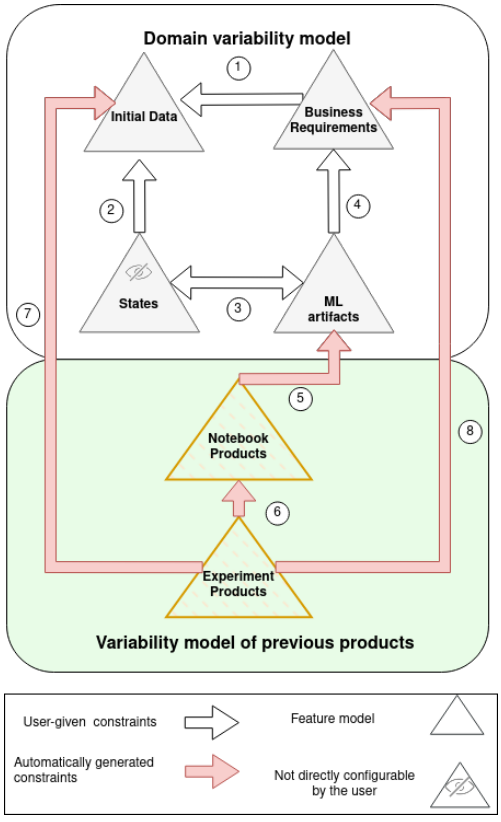


Figure 2: Feature model spaces

[5]Some of the assets we use, such as the authors of the notebooks and the source scientific articles, are not presented even if they help with the selection of notebooks They are, however, not decisive to establish a matching between the specification of a problem and the past ML artifacts, which is the focus of this paper.

3.3.2 Experiments. An experiment artifact refers to a specific configuration that defines the problem to be solved and the notebook that implements the solution. Our platform automatically extracts a representation of these artifacts and adds it to the variability submodel `ExperimentProducts`. Further details on this aspect will be given in section 3.4. This model enables us to capture and organize the products produced by data scientists and facilitates the search for past experiments. The `ExperimentProducts` variability submodel is available from the configuration phase and allows for quick identification of experiments addressing similar problems.

3.3.3 Notebooks. Data scientists can save updated notebooks along with the experiments they are associated with. We capture a simplified representation of each notebook in the `NotebookProducts` variability submodel, which consists mainly of constraints (*cf.* section 3.4). This submodel is accessible from the configuration phase onwards. It enables rapid identification of notebooks that could be cloned, or accelerates the configuration process by selecting the most appropriate notebook for the problem at hand and by automatically selecting ML artifacts. The constraints depicted by *arrows 5, 6, 7, and 8* support the identification of past products and ML artifacts based on Domain variability model configuration.

All constraints that drive this interplay between the four FMs capturing domain variability are provided by data scientists (indicated by white arrows in Figure 2)[6] All other constraints, from the two FMs capturing previous products' variability, are automatically generated (indicated by arrows 5, 6, 7, and 8 in Figure 2).

3.4 Formal definition of the interactions

We provide here a formal explanation of how interactions with the realized product variability model are defined.

Domain variability model. Configurations are our primary tool for determining *experiment* context. Let $[\![FM_D]\!]$ be the set of valid configurations of the domain variability model FM_D. In line with the domain modeling, a configuration c is composed of three subsets : $c \in [\![FM]\!], c = initData(c) \cup busReq(c) \cup mlArt(c) \cup states(c)$.

Partial configurations to specify ML problems. As we work on *experiments* whose context is difficult to define and the SPL evolves, some features about the initial data or business requirements can be neither selected nor deselected, they are simply "unknown." A set of features s in a configuration can then be defined as three subsets $s = selected(s) \cup deselected(s) \cup undefined(s)$. The intersection is empty between these three subsets.

Experiments. An *experiment* e is defined by a name n_e, the reference to a notebook nb_e and a valid configuration c_e, we note $e = (n_e, nb_e, c_e)$ where $c_e \in [\![FM_D]\!]$
The subsets $initData(c_e)$ and $busReq(c_e)$ may have a non-empty subset of undefined features. In contrast, $mlArt(c) \cup states(c)$ is complete; all the features are selected or deselected.

Variability model of realized products. The domain of realized products is defined by a set of experiments, E. The constraint system uses the experiment's name and the notebook's name to select or deselect them, depending on the other features that characterize the problem to be solved. The defined interactions are as follows.

$\forall e_i \in E, e_i = (n_{e_i}, nb_{e_i}, c_{e_i})$,
(arrow 6) $n_{e_i} \Rightarrow nb_{e_i}$,
(arrow 5) $nb_{e_i} \Rightarrow \Lambda\, selected(mlArt(c_{e_i}))$,
(arrow 7) $n_{e_i} \Rightarrow \Lambda\, selected(initData(c_{e_i})) \cup deselected(initData(c_{e_i}))$,
(arrow 8) $n_{e_i} \Rightarrow \Lambda\, selected(busReq(c_{e_i})) \cup deselected(busReq(c_{e_i}))$.

The constraint corresponding to arrow 5 deselects a notebook when an artifact used by the notebook can no longer be selected and, therefore, the experimentation associated with it (arrow 6). By restricting this constraint to the selected elements, it is possible to keep a notebook as an aid to clone even though it lacks components. The constraint (arrow 6) separates the deselection of the experimentation relative to the notebook from the deselection of the notebook. The two last constraints establish the correspondence between the experiment, the specification of the business requirements, and the initial data. Adding experimentation builds all the above constraints automatically.

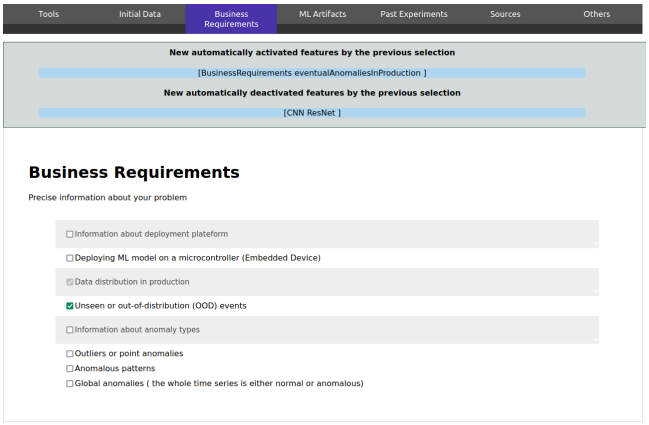


Figure 3: Tool UI for Business Requirements configuration

3.5 Tooling overview

Figure 3 depicts the tool interface developed based on Elias Kuiter's feature-configurator[7]. Data scientists use the *Initial Data*, *Business Requirements*, and *ML Artifacts* tabs to configure their experiments. Visualization, including questions, is achieved by associating features with questions through a CSV file. The tool provides access to past experiments and notebooks through the *Past Experiments* tab. The generation step is accessible through the *Tools* Tab. Technically, the generation of a totally new notebook is handled as such: Once the user requests from the tool to generate the notebook, the *generator* is only given the `MLArtifacts` part of the configuration. It then uses the mapping of the concerned submodel to `Job Assets` to find the requested product implementations. Those implementations are then copied into a data structure that represents the notebook cells. This data structure is provided by Jupyter Python library designed to programmatically generate Jupyter Notebook. It must be noted that the choice of algorithms and ML workflows is considered part of the configuration, allowing data scientists to select established

[6] Some of these constraints have been learned through the study of past experiments, but always in interaction with the data scientist, while others related to states correspond to the application of patterns upon the data scientist's request.

[7] https://github.com/ekuiter/feature-configurator

algorithms [18]. This configuration at a level usually considered as low [10] then automatically drives the configuration of business requirements.

4 ILLUSTRATION

To evaluate our approach, we discuss two illustrative scenarios, each one corresponding to a real-world case that has been simplified to clarify the propagation principles. Still, they enable us to demonstrate the capabilities of our approach. A reproduction package of our tool covering the two scenarios is available as a Zenodo artifact [9].

As explained in section 3.2 and illustrated in Figure 4c, the solution space comprises three primary types of solution components (preprocessing elements, algorithm components, and postprocessing components). As depicted in Figure 4d, the state space structure is similar to the one of the initial data space. However, it captures the current state of the data, which differs from the former in terms of data properties that may change over time. For instance, converting data from time series to a collection of spectrograms alters the data type to images, thereby enabling the utilization of solution components that may have been restricted before.

We then limit the space of realized products to two experiments with their associated notebooks. `XP1` is an experiment conducted on non-labeled and non-normalized time series data of motor vibrations to detect acquisition errors in the data, specifically the occurrence of single-point anomalies. The associated notebook includes a normalization algorithm (*e.g.*, `MinMaxScaler`) and an `LSTMAE`-type neural network. However, due to the non-quantizable nature of `LSTMAE`, this experiment cannot be embedded.

`XP2`, which refers to `NB2`, pertains to an experiment conducted on partially labeled and non-normalized time series data of sounds to detect identifiable malfunction anomalies in motor sounds with abnormal patterns (*e.g.*, `patternAnomaly`). The notebook associated with this experiment employs a spectrogram calculation to transform data into the time-frequency domain, enabling better detection of anomalies via (*e.g.*, `SFFT_createSpectrograms`), normalization via `MinMaxScaler`, and a `CNNAE`-type neural network. This model was not embedded in a `microcontroller`.

4.1 Scenario 1: Retrieve and clone notebook

Objectives. With this scenario, we want to illustrate how our approach differentiates itself from current practices. To determine whether a solution to a similar problem already exists, the user relies on her expert's insight. Once the constraints are applied, suitable solutions are proposed, enabling the user to choose an experiment or ML components.

Unfolding the scenario: Lea, a data scientist, is tasked with solving an anomaly detection problem on a dataset of motor vibrations. She must produce a solution that raises alarms when abnormal patterns indicate a defect in the motor bearing:

Lea first configures the `InitialData` FM (*cf.* Figure 4a), specifying the data type is `TimeSeries`, `PartiallyLabelled`, and not `NormalizedData`. *The given specification is sufficient to exclude Experiment* `XP1`*, which dealt with motor vibration but was performed on unlabeled data. However, this finding does not contradict the associated notebook, which employs an algorithm capable of handling partially labeled data. The constraints represented by arrow 8 in Figure 2 were applied.* Lea selects in the `BusinessRequirements` FM (*cf.* Figure 4b) `NovelAnomaliesEmergeInProd` and identifies the anomalies to be detected as `patternAnomaly`. *Based on the current state of the configuration,* `CNN` *and* `Resnet` *algorithms cannot be selected anymore, as they are not suitable for handling new anomalies in production (according to arrow 4 in Figure 2,* i.e.*, constraints 6 and 7 in Figure 4e).* At this stage, several experiments are compatible with the current configuration. Lea then clones the experiment `XP2` to work with it. *As a result, she has been able to retrieve several past experiments and notebooks by giving the system her configuration. Thanks to the constraints model she has been able to clone one that matches her problem.*

4.2 Scenario 2: Generate a new notebook

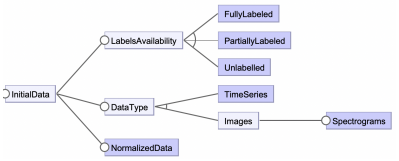
Objectives. With this scenario, we want to illustrate the correctness of the generative process, from the proposed code artifacts to the generated notebook. In the case where none of the experiments nor notebooks are reusable, we want to validate that our approach leverages constraint system propagation to reduce the number of unsuitable component compositions.

Unfolding the scenario. Lea is tasked with solving another anomaly detection problem. This time she has to use a collection of inline process control measurements from various sensors during the processing of silicon wafers for semiconductor fabrication. She must build a solution that raises the alarm for faulty wafers.

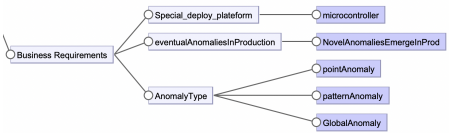
Lea informs the data properties as being `TimeSeries` and `FullyLabelled`. She also deselects `NormalizedData` as the data are not normalized. *No experiments are compatible with her data configuration.* As Lea knows that no new anomalies will arise during the production use, she then deselects `NovelAnomaliesEmergeInProd` in the business requirements. Following a discussion with the expert, she is able to determine that the series is irregular. Thus she checks `globalAnomaly`. *The two still available notebooks do not fully meet the new requirements. The tool suggests available algorithms that Lea can select.* To compare the two models, Lea generates two notebooks that share a `MinMax` scaling preprocessing, one notebook featuring a `CNN` classifier and the other featuring a `Resnet` classifier. *She chose to conduct a generative process by picking machine learning components. Only configuration-suitable features were provided to her. In order to test several classifiers, Lea had to generate as many notebooks.*

5 CONCLUSION

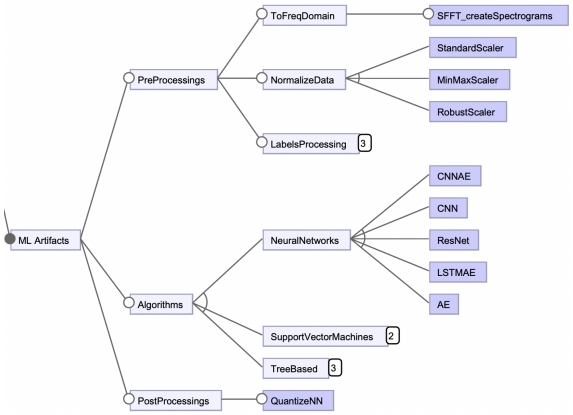
In this work, we proposed a first tooled approach to tackle the problem of diversity in computational notebooks. To help data scientists tailor a notebook to their own problem, our approach uses several variability models coupled with a constraint system. This allows the user to get feedback according to the data and business requirements of the new problem. The presented scenarios demonstrated that the user can retrieve past experiments based on problem configuration, or can totally tailor a new one. These scenarios acknowledge the potential of the proposed solution. Furthermore, an ongoing case study with an industrial partner will assess the usefulness of the configuration process, and gives us



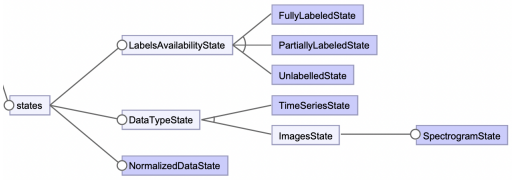
(a) FM on Initial Data



(b) FM on Business Requirements



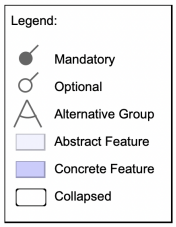
(c) FM on ML Artifacts



(d) FM on States

(1) CNNAE ⟶ ImagesState ③
(2) CNN ⟶ (ImagesState ∧ FullyLabeledState) ③
(3) ResNet ⟶ (ImagesState ∧ FullyLabeledState) ③
(4) LSTMAE ⟶ TimeSeriesState ③
(6) CNN ⟶ ¬NovelAnomaliesEmergeInProd ④
(7) ResNet ⟶ ¬NovelAnomaliesEmergeInProd ④
(10) QuantizeNN ⟶ microcontroller ④
(11) FullyLabeledState ⟶ (FullyLabeled ∨ (PartiallyLabeled ∧ augmentation)) ②&③
(14) NormalizedDataState ⟶ (NormalizedData ∨ (¬NormalizedData ∧ NormalizeData)) ②&③

(e) Domain variability model set of constraints



(f) Legend

Figure 4: Features and constraints

confidence that our findings will be soon applicable to various machine learning subdomains, considering a much broader domain modeling.

We are pursuing two different complementary validation approaches to strengthen our results. First, we plan to enhance the knowledge base through an empirical study by analyzing popular notebook repositories to characterize similarities at various levels and across different spaces, extracting the necessary information for managing the SPL [19, 27]. To validate the approach's effectiveness in reducing the solution space, we are conducting comparisons with autoML-like approaches and compared the results of novice data scientists using the approach to solve given exercises in an academic setting.

In the longer term, we aim to evolve the SPL by involving data scientists in the enrichment process. This includes using automated reasoning techniques based on past configurations [5, 14, 24, 35] and exploring model evolution [2, 6, 13, 16, 36]. Additionally, we plan to support multiple implementations within ML components using the Multi-Level Feature Trees approach [10, 31]. Improving feature visualization during configuration [8] is also a key aspect. We expect these future results to advance SPL management and enhance configuration processes in machine learning.

REFERENCES

[1] Ebrahim Khalil Abbasi, Arnaud Hubaux, and Patrick Heymans. 2011. A toolset for feature-based configuration workflows. In *2011 15th International Software Product Line Conference*. IEEE, 65–69.

[2] Mathieu Acher, Philippe Collet, Philippe Lahire, and Robert France. 2010. Managing Variability in Workflow with Feature Model Composition Operators. In *9th International Conference on Software Composition(SC'10) (Software Composition, Vol. LNCS)*. Springer, Malaga, Spain, 16.

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

[4] Saleema Amershi, Maya Cakmak, William Bradley Knox, and Todd Kulesza. 2014. Power to the People: The Role of Humans in Interactive Machine Learning. *AI Magazine* 35, 4 (Dec. 2014), 105–120. https://doi.org/10.1609/aimag.v35i4.2513

[5] 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 *Proc. of the 26th ACM*

International Systems and Software Product Line Conference - Volume A (Graz, Austria) *(SPLC '22)*. ACM, New York, NY, USA, 222–233. https://doi.org/10.1145/3546932.3547008

[6] Thorsten Berger, Ralf Rublack, Divya Nair, Joanne M Atlee, Martin Becker, Krzysztof Czarnecki, and Andrzej Wąsowski. 2013. A survey of variability modeling in industrial practice. In *Proceedings of the seventh international workshop on variability modelling of software-intensive systems*. ACM, New York, USA, 1–8.

[7] 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.

[8] Goetz Botterweck, Steffen Thiel, Daren Nestor, Saad bin Abid, and Ciarán Cawley. 2008. Visual tool support for configuring and understanding software product lines. In *2008 12th International Software Product Line Conference*. IEEE, Limerick, Ireland, 77–86.

[9] Yann Brault, Yassine El Amraoui, Mireille Blay-Fornarino, Philippe Collet, Florent Jaillet, and Frédéric Precioso. 2023. *SPLC'23 Reproduction Package*. https://doi.org/10.5281/zenodo.8013518

[10] Deepak Dhungana, Dominik Seichter, Goetz Botterweck, Rick Rabiser, Paul Grunbacher, David Benavides, and Jose A Galindo. 2011. Configuration of multi product lines by bridging heterogeneous variability modeling approaches. In *2011 15th International Software Product Line Conference*. IEEE, 120–129.

[11] Yael Dubinsky, Julia Rubin, Thorsten Berger, Slawomir Duszynski, Martin Becker, and Krzysztof Czarnecki. 2013. An exploratory study of cloning in industrial software product lines. In *Proceedings of the European Conference on Software Maintenance and Reengineering, CSMR*. IEEE, Genova, Italy, 25–34. https://doi.org/10.1109/CSMR.2013.13

[12] Manuel Fernández-Delgado, Eva Cernadas, Senén Barro, and Dinani Amorim. 2014. Do we need hundreds of classifiers to solve real world classification problems? *The Journal of Machine Learning Research* 15, 1 (Jan. 2014), 3133–3181. https://jmlr.org/papers/v15/delgado14a.html

[13] José A Galindo, Deepak Dhungana, Rick Rabiser, David Benavides, Goetz Botterweck, and Paul Grünbacher. 2015. Supporting distributed product configuration by integrating heterogeneous variability modeling approaches. *Information and Software Technology* 62 (2015), 78–100.

[14] Eddy Ghabach, Mireille Blay-Fornarino, Franjieh El Khoury, and Badih Baz. 2018. Clone-and-Own software product derivation based on developer preferences and cost estimation. In *Proceedings - International Conference on Research Challenges in Information Science*, Vol. 2018-May. IEEE Computer Society, 1–6. https://doi.org/10.1109/RCIS.2018.8406682

[15] Khan Mohammad Habibullah and Jennifer Horkoff. 2021. Non-functional requirements for machine learning: understanding current use and challenges in industry. In *2021 IEEE 29th International Requirements Engineering Conference (RE)*. IEEE, 13–23.

[16] Herman Hartmann and Tim Trew. 2008. Using Feature Diagrams with Context Variability to Model Multiple Product Lines for Software Supply Chains. In *SPLC'08*. IEEE, 12–21.

[17] Samuel Idowu, Daniel Struber, and Thorsten Berger. 2021. Asset Management in Machine Learning: A Survey. In *Proceedings - International Conference on Software Engineering*. IEEE, Virtual Event Spain, 51–60. https://doi.org/10.1109/ICSE-SEIP52600.2021.00014

[18] Michael I Jordan and Tom M Mitchell. 2015. Machine learning: Trends, perspectives, and prospects. *Science* 349, 6245 (2015), 255–260.

[19] Cory Kapser and Michael W Godfrey. 2003. Toward a taxonomy of clones in source code: A case study. *Evolution of large scale industrial software architectures* 16 (2003), 107–113.

[20] Timo Kehrer, Thomas Thüm, Alexander Schultheiß, and Paul Maximilian Bittner. 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.

[21] Mary Beth Kery, Marissa Radensky, Mahima Arya, Bonnie E. John, and Brad A. Myers. 2018. The Story in the Notebook: Exploratory Data Science using a Literate Programming Tool. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems (CHI '18)*. Association for Computing Machinery, New York, NY, USA, 1–11. https://doi.org/10.1145/3173574.3173748

[22] Andreas P. Koenzen, Neil A. Ernst, and Margaret-Anne D. Storey. 2020. Code Duplication and Reuse in Jupyter Notebooks. In *2020 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, Dunedin, New Zealand, 1–9. https://doi.org/10.1109/VL/HCC50065.2020.9127202

[23] Jacob Krüger and Thorsten Berger. 2020. An empirical analysis of the costs of clone-and platform-oriented software reuse. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ACM, 432–444.

[24] Wardah Mahmood, Daniel Struber, Thorsten Berger, Ralf Lammel, and Mukelabai Mukelabai. 2021. Seamless variability management with the virtual platform. In *Proceedings - International Conference on Software Engineering*. ACM, 1658–1670.

[25] Robert A. McDougal, Thomas M. Morse, Ted Carnevale, Luis Marenco, Rixin Wang, Michele Migliore, Perry L. Miller, Gordon M. Shepherd, and Michael L. Hines. 2017. Twenty years of ModelDB and beyond: building essential modeling tools for the future of neuroscience. *Journal of Computational Neuroscience* 42, 1 (feb 2017), 1–10. https://doi.org/10.1007/S10827-016-0623-7

[26] Nadia Nahar, Haoran Zhang, Grace Lewis, Shurui Zhou, and Christian Kästner. 2023. A Meta-Summary of Challenges in Building Products with ML Components–Collecting Experiences from 4758+ Practitioners. *arXiv preprint 2304.00078* (2023), 1–15. https://doi.org/10.48550/arXiv.2304.00078

[27] Luca Negrini, Guruprerana Shabadi, and Caterina Urban. 2023. Static Analysis of Data Transformations in Jupyter Notebooks. In *Proc. of the 12th ACM SIGPLAN International Workshop on the State Of the Art in Program Analysis*. 8–13.

[28] Samir Passi and Phoebe Sengers. 2020. Making data science systems work. *Big Data & Society* 7, 2 (2020), 1–13. https://doi.org/10.1177/2053951720939605

[29] Samir Passi and Phoebe Sengers. 2020. Making data science systems work. *Big Data and Society* 7 (7 2020). Issue 2. https://doi.org/10.1177/2053951720939605

[30] Jeffrey M. Perkel. 2018. Why Jupyter is data scientists' computational notebook of choice. *Nature* 563, 7729 (Oct. 2018), 145–146. https://doi.org/10.1038/d41586-018-07196-1

[31] M-O Reiser and Matthias Weber. 2006. Managing highly complex product families with multi-level feature trees. In *Requirements Engineering, 14th IEEE International Conference*. IEEE, 149–158.

[32] Julia Rubin, Krzysztof Czarnecki, and Marsha Chechik. 2013. Managing cloned variants: a framework and experience. In *Proceedings of the 17th International Software Product Line Conference*. 101–110.

[33] Adam Rule, Aurélien Tabard, and James D. Hollan. 2018. Exploration and Explanation in Computational Notebooks. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems* (Montreal QC, Canada) *(CHI '18)*. Association for Computing Machinery, New York, NY, USA, 1–12. https://doi.org/10.1145/3173574.3173606

[34] D. Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-Francois Crespo, and Dan Dennison. 2015. Hidden Technical Debt in Machine Learning Systems. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2* (Montreal, Canada) *(NIPS'15)*. MIT Press, Cambridge, MA, USA, 2503–2511.

[35] Leopoldo Teixeira, Rohit Gheyi, and Paulo Borba. 2020. Safe evolution of product lines using configuration knowledge laws. In *Formal Methods: Foundations and Applications: 23rd Brazilian Symposium, SBMF 2020, Ouro Preto, Brazil, November 25–27, 2020, Proceedings 23*. Springer, 210–227.

[36] Thomas Thum, Don Batory, and Christian Kastner. 2009. Reasoning about edits to feature models. In *2009 IEEE 31st International Conference on Software Engineering*. IEEE, 254–264.

[37] 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

[38] April Yi Wang, Dakuo Wang, Jaimie Drozdal, Xuye Liu, Soya Park, Steve Oney, and Christopher Brooks. 2021. What Makes a Well-Documented Notebook? A Case Study of Data Scientists' Documentation Practices in Kaggle. In *Extended Abstracts of the 2021 CHI Conference on Human Factors in Computing Systems (CHI EA '21)*. Association for Computing Machinery, New York, NY, USA, 1–7. https://doi.org/10.1145/3411763.3451617

[39] Jiawei Wang, Tzu-yang Kuo, Li Li, and Andreas Zeller. 2020. Restoring Reproducibility of Jupyter Notebooks. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Companion Proceedings* (Seoul, South Korea) *(ICSE '20)*. Association for Computing Machinery, New York, NY, USA, 288–289. https://doi.org/10.1145/3377812.3390803

[40] Rüdiger Wirth and Jochen Hipp. 2000. CRISP-DM: Towards a standard process model for data mining. In *Proceedings of the 4th international conference on the practical applications of knowledge discovery and data mining*, Vol. 1. Manchester, 29–39.

[41] Matei Zaharia, Andrew Chen, Aaron Davidson, Ali Ghodsi, Sue Ann Hong, Andy Konwinski, Siddharth Murching, Tomas Nykodym, Paul Ogilvie, Mani Parkhe, Fen Xie, and Corey Zumar. 2018. Accelerating the machine learning lifecycle with MLflow. *IEEE Data Engineering Bulletin* 41, 4 (2018), 39–45.