Taming the Diversity of Computational Notebooks
Yann Brault, Yassine El Amraoui, Mireille Blay-Fornarino, Philippe Collet, Florent Jaillet, Frédéric Precioso

To cite this version:

HAL Id: hal-04247860
https://hal.science/hal-04247860
Submitted on 18 Oct 2023

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers. L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
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

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


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 (over 400K notebooks today), Google Colab, Jupyter Notebook, JupyterLab, or Baidu AI Studio.

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.

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. 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 submodule 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 submodule, which consists mainly of constraints (cf. section 3.4). This submodule 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). 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 $ce$, we note $e = (n_e, nb_e, ce)$ where $ce \in [FM_D]$. The subsets initData($ce$) and busReq($ce$) may have a non-empty subset of undefined features. In contrast, mlArt($ce$) $\cup$ states($ce$) 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.

$$\forall e_i \in E, e_i = (n_{e_i}, nb_{e_i}, ce_i),$$

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

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.

3.5 Tooling overview

Figure 3 depicts the tool interface developed based on Elias Kuiter’s feature-configurator. 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

---

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.

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 LSTMÆ-type neural network. However, due to the non-quantizable nature of LSTMÆ, 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 4c). 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
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


