Towards a Software Product Line for Machine Learning Workflows: Focus on Supporting Evolution

Cécile Camillieri
camillie@i3s.unice.fr

Luca Parisi
parisi@i3s.unice.fr

Mireille Blay-Fornarino
blay@i3s.unice.fr

Frédéric Precioso
precioso@i3s.unice.fr

Michel Riveill
riveill@i3s.unice.fr

Joël Cancela Vaz
joel.cancelavaz@gmail.com

Université Côte d'Azur, CNRS, I3S
Bat Templier, 930 Route des Colles
Sophia Antipolis, France

ABSTRACT

The purpose of the ROCKFlows project is to lay the foundations of a Software Product Line (SPL) that helps the construction of machine learning workflows. Based on her data and objectives, the end user, who is not necessarily an expert, should be presented with workflows that address her needs in the "best possible way". To make such a platform durable, data scientists should be able to integrate new algorithms that can be compared to existing ones in the system, thus allowing to grow the space of available solutions. While comparing the algorithms is challenging in itself, Machine Learning, as a constantly evolving, extremely complex and broad domain, requires the definition of specific and flexible evolution mechanisms. In this paper, we focus on mechanisms based on meta-modelling techniques to automatically enrich a SPL while ensuring its consistency.

Keywords

Software Product Line, Machine Learning Workflow, Evolution

1. INTRODUCTION

The answer to the question "What Machine Learning (ML) algorithm should I use?" is always "It depends." It depends on the size, quality, and nature of the data. It also depends on what we want to do with the answer [21].

The industry of cloud-based machine learning (*e.g.*, IBM's Watson Analytics, Amazon Machine Learning, Google's Prediction API) provides tools to learn "from your data" without having to worry about the cumbersome pre-processing and ML algorithms. To address such a challenge they propose fully automated solutions to some classical learning problems such as classification. Some other actors like Microsoft, with the Azure's Machine Learning platform, allow users to build much more complex ML workflows, in a graphical editor that is targeted towards ML experts.

The common point between these solutions is that they chose to select only a few algorithms, in comparison to the hundreds that are available. However data scientists know that the best algorithm will not be the same for each dataset [22]. Moreover, new algorithms are regularly proposed by data scientists for dealing with more or less specific problems and improving performances and accuracy [6]. Thus, in order to help users who want to build ML workflows, we have to propose a system that can present a large variety of algorithms to users, while helping them in their choices based on their data and objectives. At the same time, we should be able to extend the supported solutions at least by incorporating new algorithms. The challenge is to hide the complexity of the choices to the end user and to revise our knowledge with each addition: an algorithm can become less efficient compared to a new one, while the introduction of new pre-processing operations can extend the reach of algorithms already present.

The contribution of this paper is thus to describe a tool-supported approach, responding to this challenge: the ROCKFlows project[1].

The remainder of the paper is organized as follows. We discuss in the next section challenges we face and some related works. Section 3 describes the architecture that supports the project and two usage scenarios focusing each on a different user of ROCKFlows. We detail the evolution process and the correlated artefacts in Section 4. Section 5 concludes the paper and briefly discusses future work.

2. TOWARDS A SPL FOR MACHINE LEARNING WORKFLOWS

The purpose of the ROCKFlows project is to lay the foundations of a software platform that helps the construction of ML workflows. This task is highly complex because of the increasing number and variability of available algorithms and the difficulty in choosing the suitable and parametrized algorithms and their combinations. The problem is not only on choosing the proper algorithms, but the proper transformations to apply on the input data. It is a trade-off between many requirements (*e.g.*, accuracy, execution and training time).

Since Software Product Line (SPL) engineering is concerned with both variability and systematically reusing development assets in an application domain [5], we have based our project on SPL and model-driven techniques. The SPL engineering separates two processes: *domain engineering* for defining commonality and the variability of the product line and *application engineering* for deriving product line appli-

[1]ROCKFlows stands for Request your Own Convenient Knowledge Flows.

cations [15]. Similarly, the ROCKFlows project requires on one hand to build a consistent SPL, allowing end users to get reliable workflows, and on the other hand to allow evolution of this SPL to integrate new algorithms and pre-processing treatments.

Based on these requirements, we have identified the following challenges, addressing the needs for building and evolving a SPL in a domain as complex and changing as ML, ensuring a global consistency of the knowledge and scalability of the system.

C1: Exploratory project in a complex environment.

Making a selection among the high number of data mining algorithms is a real challenge: more than hundreds of algorithms exist that can tackle a single ML problem such as classification. While work exists to try and rank their performance [6], it only gives an overview of which algorithms are best in average, not for a given specific problem and not according to different pre-processing pipelines.

Data scientists often approach new problems with a set of best practices, acquired through experience. However, there is few scientific evidence as to why an algorithm or pre-processing technique leads to better results than another and in which case. Thus, one of the biggest challenge for this project is so to find a proper way to characterise algorithms and to compare them, relatively to the very broad spectrum of user needs and data representations.

Collaboration between SPL developers and data scientists induces a complex software ecosystem [13] where some mathematical results may or not find a correspondence at end user problem level. Heterogeneity of formalisms induces that new evolutions are regularly discussed between the different stakeholders.

Given these domain requirements, meta-model driven engineering provides an efficient and powerful solution to address the complexity of the ecosystem through support of separation of concerns and collaborations. In order to operationalize it, we chose to consider this environment as a set of components relying on different meta-models for which the evolution mechanisms are exposed through services.

C2: SPL building in a constantly evolving environment.

While the number of ML algorithms and techniques constantly grows, the fundamental understanding of ML internal mechanisms is not stable enough to allow us to set any knowledge in stone. Both the domain and our understanding of it evolve quickly, forcing constant evolution of the SPL.

The line evolves in particular through the addition of new algorithms and pre-processings. We run experiments to identify dataset patterns leading to similar behavior of algorithms on different concrete datasets. The high number of possible combinations (variability of compositions and algorithms) as well as the frequent changes in ML require evolution mechanisms that are both incremental and loosely coupled with the elements presented to the end user.

As of today, ROCKFlows' SPL contains roughly 300 features, and 5000 constraints, representing 70 different ML algorithms, 5 pre-processing workflows, and is mostly focused on classification problems.

Evolution in SPLs has been a challenge for many years [15]. In particular, several works exist on evolution of Feature Models (FM) [8, 1]. They propose different mechanisms for maintaining consistency of evolving FMs. In our case, we rely on these operations to update our models, but we had to encapsulate them in business oriented services.

Moreover, despite the huge variability of the system, we have decided to propose the end user only choices that can lead to a proper result. Hence, it should not be possible to build a configuration for which we would not be able to generate a workflow. For instance, it will not be possible for a user to select, relatively to a given dataset, a performance value for which we have no algorithm that can reach such requirements. It is necessary for us to ensure a consistent configuration process [20].

Contrary to the works allowing several users to modify a model in contradictory manners and aiming to reconcile those [3], here we are in a setting where only consistent evolutions are possible. Thus we did not have to handle co-evolution problems. Like the approach used in SPLEMMA [17], "Maintenance Services" define the semantics of evolution operations on the SPL ensuring its consistency. However, the analogy between meta-elements manipulated in ROCKFlows and SPLEMMA is hard to establish, especially because our solution and problem spaces are in a constant evolution. Thus the associated meta-models are not stable, implying intraspatial second degree evolutions [19] *i.e.*, several spaces and mappings are simultaneously modified; *e.g.*, Adding a non-functional property kind, due to some improvement of the experiment meta-model, involves to extend the FM and corresponding end user representation (problem space) and generation tools to take into account this new feature.

C3: User Centered SPL.

We identify three stakeholders, each leveraging challenges.
- **SPL users** are the end users of the SPL: it can be a neophyte who is looking for a solution to extract information from a dataset as well as an expert who wants to check or learn dependencies among dataset properties, algorithms, targeted platforms and user objectives. They want to use a system helping them to master the variability, *i.e.*, to express their requirements and get their envisioned ML workflow. Both of these do not know about FMs and may need complementary information like examples of uses of the algorithms, details about the algorithm author or implementation, etc. Thus, the visualization of a FM as in standard tool is not adapted. Creating a user interface dedicated to the SPL is also problematic knowing the changing nature of the system. At the same time, in a agile process, we need to test the SPL with users in order to align it with their needs, which are hard to identify a priori.

Some flowcharts have been designed to give users a bit of a rough guide on how to approach ML problems [16, 14]. ROCKFlows wants this approach to be operational. So, we do not aim for the construction of workflows by assembly, but the automatic production of these workflows, without a direct contribution of the user in this construction process. Works such as these are however potential targets for the generation of the workflows where proposed optimization could then be used automatically. Moreover, faced with the multitude of such systems (Clowdflows [10], MLbase [11], Weka [9]), it is right to allow the user to select her execution target(s) so that the production is limited only to the ML workflows implemented by these platforms.
- **External Developer** are domain experts who contribute

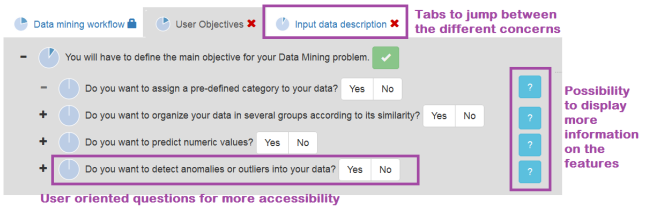

Figure 2: Screenshot of ROCKFlows' GUI.

to the SPL. They do not have all the knowledge of the system and contribute by adding new algorithms. They have to be able to contribute separately with minimal interference.
- **Internal developers** are leaders of the SPL. They have the knowledge of the global architecture and manage contributions of external developers to integrate them. They have to be able to maintain the platform and to ensure the consistency of all products despite the evolution of the ecosystem.

3. ARCHITECTURE FOR ROCKFlows

3.1 ROCKFlows Big Picture

Figure 1 represents the current proposed architecture for ROCKFlows on the component level. On the left of the central vertical line are the components that will be necessary for the end user to configure her worfklow. On the right, components enabling users to add their own ML algorithm in the system is described. Relationships between the different components are relying on meta-models.

We now describe this architecture through the description of two scenarios:

3.1.1 *Scenario 1: Configuration of a ML Workflow*

A user willing to configure a new ML workflow will do so through a web-based configuration interface[2]. The process requires at least the following steps, as visible on the left of figure 1:

(a) The Graphical User Interface (`GUI`) requests display metadata on the Feature Model. Metadata associate each unique feature of the model with descriptions, references or other artefacts aiming to help non-experts in their choices. Figure 2 shows a screenshot of our GUI. Here, user is presented with the choice of its main objective, in the form of questions.

(b) Once the FM is loaded and displayed properly with the metadata, the user configures the underlying FM by responding to questions. The `Feature Model` component in the figure exposes a web-service that allows for configuration on any FM, through the use of SPLAR's API [12].

(c) Once a valid and complete configuration has been defined, it is sent to the `Workfow` component. The configuration is then transformed into a Platform Independent workflow model, that can in a second step be used to generate executable code for different target platforms.

(d) The Generator may require access to the base of algorithms handled by the system in order to be able to produce the proper code.

[2]The interface is accessible at http://rockflows.i3s.unice.fr

Though it is not described here, depending on user's preference, the generated workflow would either be provided to the user or directly executed by the target platform.

3.1.2 *Scenario 2: Submission of a new algorithm*

We will now focus on the introduction of new ML algorithms in the SPL. Such an action has impacts on several parts of the system:
- **SPL:** At least a new feature representing the algorithm should be added in the FM;
- **GUI**: Display metadata should be updated to reflect this new feature in the GUI;
- **Generation:** If we can execute the algorithm, the generator should be updated to allow it either through code or a reference to the corresponding element in target platform(s);
- **Experiments** should be made with this new algorithm in order to compare it to the other known algorithms. Currently, the properties that are considered are accuracy of the results, execution time of the workflow, and memory usage. If experiments can be achieved, *i.e.*, `Experiments` module has access to algorithm execution and results, the SPL needs to be updated with performance information for this algorithm but also for all the algorithms whose ranking has changed.

The central component `SPLConsistencyManager`'s role is to ensure that all required changes are made across the whole system. Through the present scenario, we describe how this component handles the impacts mentioned above.

(1) As an External Developer wants to add a new algorithm, the `GUI` presents her with proper information that she needs to provide. Because this information is meant to change as the system encompasses more possibilities of Machine Learning, it should be easy to change. Hence, the `SPLConsistencyManager` provides the GUI with the information needed to add a new algorithm in the tool. The GUI presents then a generated form to the user.

(2) Once the External Developer has provided all information on the algorithm, it is sent to the Consistency Manager and dispatched among the other components.

(3) If the External Developer provided code to execute the algorithm, the `Experiments` component is requested to run tests on the algorithms, in order to find its performance. This component is described more precisely in the next section.

(4) Once experiments are finished, the manager analyses the results to find whether any inconsistency was found between the information provided by the user and the results. If not, the algorithm can be added both to the Feature Model and base of currently supported algorithms.

(5) Finally, once the feature has been added to the FM, its display metadata can be filled with the information provided by the expert.

3.2 Component for experiments on algorithms

As we want even non-expert users to be able to get the appropriate algorithm for their need, we chose to express higher level goals such as *best* accuracy or *quickest* execution time in the FM. This representation allows to filter out a number of algorithms by offering users with trade-offs over

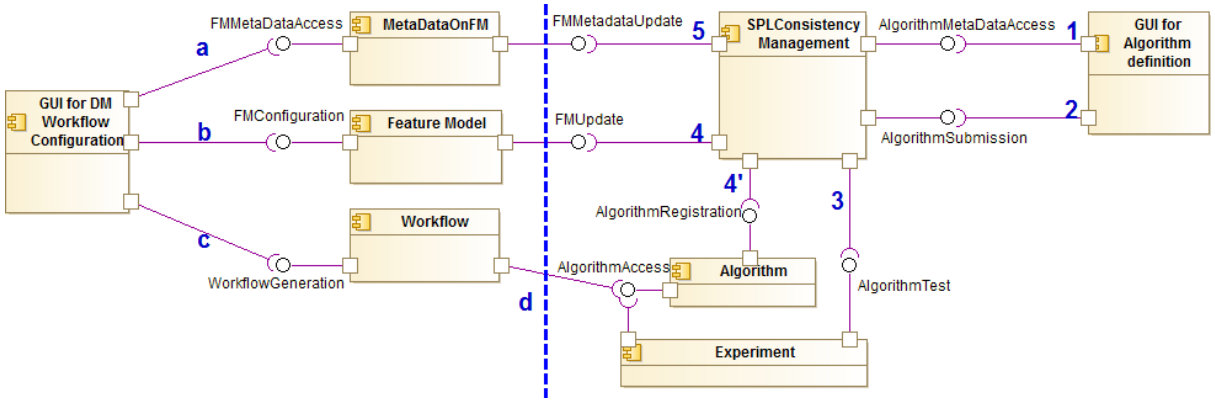

Figure 1: High level architecture for ROCKFlows.

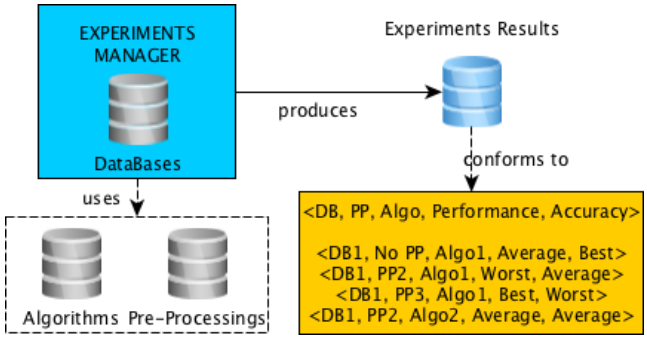

Figure 3: Experiments

these goals. In a second step, or a more advanced mode, using other models such as requirements engineering Goal Models is considered. In combination with our FM, it could allow during configuration to present the user with more precise expected values for her non functional goals [2].

To be able to express knowledge such as "best", "average" or "worst" accuracy, we need an appropriate way to compare algorithms on a given problem. This ranking among algorithms is computed by our `Experiment` module. The component runs each known algorithm on the available compatible datasets and stores its performance on the different properties for each dataset. On top of that, it will transform the datasets with a set of available pre-processing operations and test again each algorithm with those new sets.

Algorithms results on similar datasets are then compared to one another to get a result similar to the one visible on figure 3. Though it will not be discussed here, we have defined an algorithm based on classical ML and statistical methods to pull off this comparison and regroup datasets in so-called dataset patterns. This knowledge can then be pushed in the FM in the form of constraints linking a functional objective, an algorithm, a dataset pattern and the ranking for each of the properties, also depending on the possible preprocessings for this dataset.

As presented in section 3.1.2, the `Experiment` component is driven by the `SPL Consistency Manager`, that is charged to start experiments, gather and validate results before incorporating them in the SPL. As new algorithms are added, all experiments do not have the need to be executed again, however the ranking of the algorithms must be updated to take in account the newest algorithm.

4. ARTEFACTS TO SUPPORT EVOLUTION

This section discusses how we allow users to provide information about new algorithms, how we use it to update the system, and how we can ensure consistency despite the multiple impacts of these changes.

4.1 Meta-models

Figure 4 shows an excerpt of the meta-elements that deal with evolution of the SPL. Each component described earlier corresponds to a meta-model, and the `SPLConsistencyManager` maintains consistency among them.

4.1.1 SPL: Feature Model and Configuration

Our feature model is represented in the SXFM (Simple XML Feature Model)[3]. The rest of our SPL handling is also made through SPLOT's FM reasoning library, SPLAR[4].

4.1.2 Addressing non-expert users

In order to make the system as accessible as possible, additional information on features must be set, such as descriptions or examples, as well as closed questions that will be asked to the end user during configuration. This metadata on the practical features is handled in a dedicated meta-model `AlgorithmDescriptionMM` and used to build the GUIs that are presented to the end users and external developers. The model is briefly described in subsection 4.2.

4.1.3 Handling the results of Experiments

Information such as the accuracy ranking of algorithms according to dataset patterns is managed by the `Experiment` component. The expected format of this data is designed in a dedicated meta-model `ExperimentPropertyMM`. It evolves as we gain knowledge and experience.

4.2 Metadata on Algorithms

[3]http://ec2-52-32-1-180.us-west-2.compute.amazonaws.com:8080/SPLOT/sxfm.html

[4]An excerpt of the meta-model used for both FM and configuration definition in the library, can be found in https://github.com/FMTools/sxfm-ecore/blob/master/plugins/sxfm/model/sxfm.png

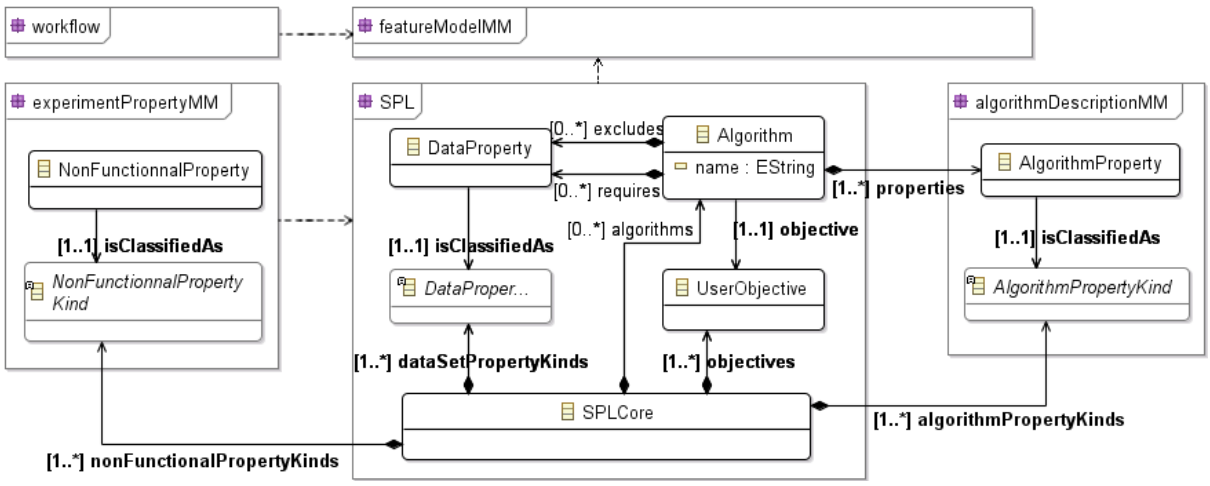

Figure 4: SPL Core Meta-models excerpt

Data scientists adding their algorithms in the system need to provide at least:
- **The high level ML objective** the algorithm can be used for: classification of data, prediction of numerical values (regression), anomaly detection, etc.;
- **Properties** of the algorithm in regards to **input data**: which data types the algorithm supports, can the algorithm handle missing values in the data, etc.;
- **A description** of the algorithm, examples of its use, references to publications or web pages describing the algorithm. As described in section 4.1.2, those will be displayed in the configuration interface;
- If possible, **code** that will allow us to run the algorithm in our tool, so that we can both compare it to the others algorithms and provide executable workflows to end users.

Through the definition of theses elements in the `AlgorithmDescriptionMM` meta-model, a form is automatically generated and presented to the domain expert. Thus, the model can be extended to add new properties for the algorithms and will be automatically handled by the GUI. However the impact of such changes on the FM still has to be handled by the SPL manager. We do not know if tools such as the one described in [7] could help because those changes mostly impact code.

4.3 Domain driven tooled approach to manage Feature Model evolution

Even though we have defined a single FM for ROCK-Flows, we put a focus on separating concerns in it. The tree is currently separated in 4 sub-trees handling: input data description, user objectives, the processing algorithms, and expected properties of the generated workflow. This separation of concerns provides a first level of modularity for the model.

Linking Domain artefact and FM structure. Metadata external to the FM itself defines particular points in the FM where feature can be inserted. Only the sub-trees that need to be modified are considered. In our example, we add all algorithms responding to the classification problem in the same sub-tree. This mechanism enables us to extend the model cleanly, and abstract ourselves from the exact hierarchy of the features. So, adding new class of algorithms or modifying the structure of the FM can easily be achieved. Once the feature for the algorithm has been created, additional constraints need to be defined between the algorithm and other features, in particular those describing input data.

Generating domain constraints. Only certain types of constraints must be defined among those different sub-trees. For instance algorithms can define constraints towards input data, such as "SVM *implies* Numerical Data" but never the other way around. However such a constraint only applies in a workflow if no pre-processing is used. So, this constraint has to be transformed to express a constraint depending on the pre-processings that can be applied. The complexity of these cases, their multitude and the frequency of evolution lead us to encapsulate the generation of those constraints in dedicated operators, working on given ensembles (*e.g.*, pre-processing set that returns numerical Data). They also introduce features that are hidden to the end user, allowing us to tame this complexity.
It is interesting to note that, depending of the semantics associated to the features, different constraints should be generated. For instance, if an algorithm cannot deal with missing values, a constraint "algo *excludes* missing values" needs to be generated. In the other case, a constraint "algo *implies* missing values" should never be generated because such an algorithm can still be used even if no missing value is present in the input data. Like previously, this higher level knowledge of the features is defined in our metadata. It allows us to ensure that all necessary constraints are properly defined for all algorithms we add into the SPL.

5. CONCLUSION AND FUTURE WORK

In this paper, we have outlined some of the difficulties related to building and evolving a SPL for ML workflows. To handle the line's complexity and evolution, we have proposed an architecture organized around a set of meta-models and transformations encapsulated within services, A large number of complementary perspectives are considered, both on mechanisms to build the SPL and on the business approach of ML.

The complexity we are facing requires an agile and pragmatic approach in which users are given the opportunity to provide feedback during the early stages of the project.

The necessity for evolution of the system leads us today to consider moving towards approaches focused on the reuse of existing components [18]. This would allow for a necessary control over the system's evolution.

The domain of ML is currently booming with propositions for algorithms, distributivity of execution and the opening to a larger audience. We consider to extend our approach to integrate the necessary parameters for distributing workflows' execution as well as proposing deep learning workflows. A longer term question is to target Scientific Workflow Management systems, allowing so to explore data driven execution through transformations targeting the specification interchange language called WISP [4].

Finally, we wish to allow the end user to express her needs in business terms, through an approach similar to IBM Watson Analytics[5]. It is a question of integrating the state of the art practices in our modelisation, without losing the power of our evolutionary approach, driven by experiments.

6. REFERENCES

[1] M. Acher, P. Collet, P. Lahire, and R. France. Separation of Concerns in Feature Modeling: Support and Applications. In *AOSD'12*. ACM, 2012.

[2] O. Alam, J. Kienzle, and G. Mussbacher. *Concern-Oriented Software Design*, pages 604–621. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[3] A. Anwar, S. Ebersold, B. Coulette, M. Nassar, and A. Kriouile. A Rule-Driven Approach for composing Viewpoint-oriented Models. *Journal of Object Technology*, 9(2):89–114, 2010.

[4] B. F. Bastos, R. M. M. Braga, and A. T. A. Gomes. WISP: A pattern-based approach to the interchange of scientific workflow specifications. *Concurrency and Computation: Practice and Experience*, 2016.

[5] P. Clements and L. M. Northrop. *Software Product Lines : Practices and Patterns*. Addison-Wesley Professional, 2001.

[6] M. Fernández-Delgado, E. Cernadas, S. Barro, and D. Amorim. Do we Need Hundreds of Classifiers to Solve Real World Classification Problems? *Journal of Machine Learning Research*, 15:3133–3181, 2014.

[7] S. Getir, M. Rindt, and T. Kehrer. A generic framework for analyzing model co-evolution. In *Proceedings of the Workshop on Models and Evolution co-located with MoDELS 2014, Valencia, Spain, Sept 28, 2014.*, pages 12–21, 2014.

[8] J. Guo, Y. Wang, P. Trinidad, and D. Benavides. Consistency maintenance for evolving feature models. *Expert Systems with Applications*, 39(5):4987–4998, 2012.

[9] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The weka data mining software: An update. *SIGKDD Explor. Newsl.*, 11(1):10–18, Nov. 2009.

[10] J. Kranjc, V. Podpečan, and N. Lavrač. Clowdflows: a cloud based scientific workflow platform. In *Machine Learning and Knowledge Discovery in Databases*, pages 816–819. Springer Berlin Heidelberg, 2012.

[11] T. Kraska, A. Talwalkar, J. Duchi, R. Griffith, M. J. Franklin, and M. Jordan. MLbase: A Distributed Machine-Learning System. In *CIDR*, 2013.

[5]https://www.ibm.com/analytics/watson-analytics/

[12] M. M. Mendonça, M. Branco, D. Cowan, and M. Mendonca. S.P.L.O.T.: software product lines online tools. In *OOPSLA*, pages 761–762. ACM Press, 2009.

[13] T. Mens, M. Claes, P. Grosjean, and A. Serebrenik. Studying Evolving Software Ecosystems based on Ecological Models. In *Evolving Software Systems*, pages 297–326. Springer, 2014.

[14] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, and al. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011. http://scikit-learn.org/stable/tutorial/machine_learning_map/.

[15] K. Pohl, G. Böckle, and F. J. van der Linden. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer-Verlag, 2005.

[16] B. Rohrer. Machine learning algorithm cheat sheet for Microsoft Azure Machine Learning Studio. https://azure.microsoft.com/en-us/documentation/articles/machine-learning-algorithm-cheat-sheet/.

[17] D. Romero, S. Urli, C. Quinton, M. Blay-Fornarino, P. Collet, L. Duchien, and S. Mosser. SPLEMMA: a generic framework for controlled-evolution of software product lines. In *International Workshop on Model-driven Approaches in SPL (MAPLE)*, volume 2013, pages 59–66, 2013.

[18] M. Schöttle, O. Alam, J. Kienzle, and G. Mussbacher. On the modularization provided by concern-oriented reuse. In *Proceedings of MODULARITY'16*, pages 184–189, New York, NY, USA, 2016. ACM.

[19] C. Seidl, F. Heidenreich, U. Aßmann, and U. Aßmann. Co-evolution of Models and Feature Mapping in Software Product Lines. In *Proceedings of SPLC'12*, pages 76–85, New York, NY, USA, 2012. ACM.

[20] S. Urli, M. Blay-fornarino, and P. Collet. Handling Complex Configurations in Software Product Lines : a Tooled Approach. In ACM, editor, *Proceedings of SPLC'14*, pages 112–121, Florence, Italy, 2014.

[21] I. H. Witten, E. Frank, and M. a. Hall. *Data Mining: Practical Machine Learning Tools and Techniques (Google eBook)*. 2011.

[22] D. H. Wolpert. The lack of a priori distinctions between learning algorithms. *Neural Comput.*, 8(7):1341–1390, Oct. 1996.