Explicit or Implicit? On Feature Engineering for ML-based Variability-intensive Systems

Paul Temple, Gilles Perrouin

Explicit or Implicit? On Feature Engineering for ML-based Variability-intensive Systems

Paul Temple
paul.temple@irisa.fr
Univ Rennes, CNRS, Inria, IRISA
Rennes, France

Gilles Perrouin
gilles.perrouin@unamur.be
PReCISE, NaDI, University of Namur
Namur, Belgium

ABSTRACT

Software variability engineering benefits from Machine Learning (ML) to learn *e.g.,* variability-aware performance models, explore variants of interest and minimize their energy impact. As the number of applications of combining variability with ML grows, we would like to reflect on what is the core to the configuration process in software variability and inference in ML: *feature engineering*. These disciplines previously managed features explicitly, easing graceful combinations. Now, deep learning techniques derive automatically obscure but efficient features from data. Shall we give up explicit feature management in variability-intensive systems to embrace machine learning advances?

KEYWORDS

feature, machine learning, software variability

ACM Reference Format:
Paul Temple and Gilles Perrouin. 2023. Explicit or Implicit? On Feature Engineering for ML-based Variability-intensive Systems. In *17th International Working Conference on Variability Modelling of Software-Intensive Systems (VaMoS 2023), January 25–27, 2023, Odense, Denmark*. ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

To deal with the inherent engineering complexity of variability-intensive systems, machine learning (ML) techniques are helpful. Applications include specifying software product lines [1, 20] and predicting systems' performance [18, 19]. Conversely, variability management techniques can support the design of machine learning models [8].

Core to the cross-fertilization of software variability and ML is the process of *feature engineering*. While the term "feature" has many meanings [4, 6] in the software variability and relates to various types of data processed by ML algorithms, feature engineering is essential for relevant configurations (variability) and accurate decisions (ML). In this paper, we show that though variability management relies on *explicit* feature engineering, ML transitioned from explicit to implicit feature engineering, notably enhancing the performance of deep learning models. Embracing these advances via implicit feature engineering may imply giving up the essence of variability management by losing control over the features. Should we go there?

Section 2 presents explicit feature engineering in ML and variability. Section 3 describes implicit feature management in deep learning. Finally, Section 4 discusses the implications of implicit feature management for software variability practice.

2 EXPLICIT FEATURE ENGINEERING

Software variability features. From a user-oriented perspective, we find multiple definitions of "feature" in software product lines textbooks: "a feature is a characteristic or end-user-visible behavior of a software system" [3]; "a set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way." [7]. Rashid *et al.* define a feature as "characteristic of a product that is visible to the end user in some way" [15]. Thus, from a user perspective, a feature can be anything (requirements, design element, software artifact) as long as it helps distinguish one product from another.

ML features for image processing. On the other hand, ML models also try to distinguish products (*i.e.,* data) based on their feature descriptions. ML features are supposed observable characteristics that, if appropriately selected, should help decide which group data belong to. Their definition remains difficult and historically relies on domain expertise. In image processing, Harris *et al.* [10] defined corners as the most relevant parts of images helping to differentiate them. Schmid *et al.* [16] improved this approach before the introduction of the Scale Invariant Feature Transform (SIFT) descriptors [13]. In this context, the literature defines features as visually explicit elements: corners and boundaries.

ML features for tabular data. The Iris dataset [1] is a toy example for pedagogical purposes. It contains only 150 instances separated into three classes, all described by four features (or attributes). They are the length and width of the sepals and petals. The ML model task is to analyze the attributes of the sepals and petals to understand what makes one instance belongs to a class or another. According to *domain knowledge*, these four characteristics *should be sufficient* to classify irises. Taking a different example, the Postoperative dataset [2] tries to determine where patients in a postoperative recovery area should go next (home, intensive care unit, or general hospital floor). Domain knowledge informs us that hypothermia is an important risk after surgery. Thus, the identified features that

[1] https://archive.ics.uci.edu/ml/datasets/iris
[2] https://archive.ics.uci.edu/ml/datasets/Post-Operative+Patient

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

describe the situation (*i.e.,* overall state of patients) focus on body temperature.
All these examples have defined their features according to domain knowledge that documents how experts analyze data to decide. Because of this human and somewhat manual aspect, features carry some explicit, explainable semantics. It allows direct mapping from the decisions to a combination of these features.

Using software variability features as ML features. Regarding software variability and feature models, concrete features (abstract features help structure information) also relate to the decisions users make when building their products. Here again, the semantics of features is explicit, allowing users to understand the impact of their choices on the resulting products. However, despite using software variability features as directly as ML features (*e.g.,* [1, 2, 9, 20, 21]), some difficulties arise. For instance, ML models often require homogeneous features (*i.e.,* over the same range of values and preventing the use of a list of elements in a single string). It demands changing the representation of enumerations. Usually, one turns literals into integers. Yet, ML models consider a spurious order between literals. One thus flattens enumerations into a set of Boolean features, for example: "is choice X selected?". The Pandas framework, in Python, calls this decomposition "dummification"[3]. While this causes the number of features artificially increase (which may cause other problems for ML, *e.g.,* the curse of dimensionality [11]) and makes them independent, the enumeration semantics remains.

3 IMPLICIT FEATURE ENGINEERING

The rise of deep learning (DL) has forced models to learn themselves which features are relevant to their tasks. Doing so, DL models have shown incredible performances that continue to be improved every day.

Embedding in Deep Learning. A first step (called embedding) transforms data into an intermediate homogeneous representation that accounts for relations between elements of input data (*e.g.,* close meaning or frequent co-occurrences). One turns raw input data into vectors of real-value numbers. This vector defines a high-dimension space that groups data points of close meaning or frequent co-occurrence. Conversely, there is a greater distance among points having different meanings or infrequent occurrences. While this transformation may take as input the previously mentioned characteristics, the fact that dimensions are now a non-trivial combination of these different elements renders the interpretation of each dimension difficult, if not impossible. However, the new representation is prone to define a continuous space allowing the use of, for instance, gradient-based methods for optimization (and especially to converge when learning the task of the DL model). The creation and selection of such dimensions is fully automatic. Hence, embedding and feature extraction do not have to rely on the knowledge of domain experts to guide the selection or definition of such features. Yet, there is no guarantee of meeting all the expected properties of the resulting encoding space.

[3]https://pandas.pydata.org/docs/reference/api/pandas.get_dummies.html

Interpretation and control over DL features. Automated embedding and feature extraction techniques raise the problem of building humanly understandable models (*i.e.,* explainable AI), in which we seek to understand why the model took a decision. Also, multiple software engineering tasks rely on a deep understanding of the system's behavior. For instance, trying to debug a system because of unexpected behavior is facilitated by clear code structures. With DL models, understanding the model decision or why it did not activate the expected neuron is much more difficult because the features do not have concrete meaning. As an example of an application, Pierazzi *et al.* [14] used some gradient-based techniques to modify programs such that they become malicious. Yet, the goal of the transformation is to insert a malicious behavior in the program while retaining its basic functionality (*i.e.,* the program still behaves correctly, in addition, it retrieves and leaks information about the system). This work first transforms the code of the application into an intermediate representation. This representation supports an optimal introduction of malicious behavior. Yet, the transformation needs to be bidirectional since we need to produce back the modified source code to run tests. Such a property is desirable for domain expert intervention and control.

Balancing performance and control. Deep learning can achieve far better performances in various tasks than older ML models and relies on techniques that demand fewer human actions to optimize the data representation. The current trend is to use embedding to provide an efficient intermediate representation to employ gradient-based optimization techniques. Unfortunately, this prevents domain experts from controlling how the model behaves and, if necessary, modifying it accordingly. On the other hand, their intuitions on which characteristics to consider to solve a problem may be wrong, leading to ML models of lower performance. In the end, it seems that there is a trade-off to find between the level of control and understanding experts need to have to pursue their different activities, such as model debugging, and the performance of the models.

4 EXPLICIT OR IMPLICIT?

So the dilemma in this paper is the following: *either we acknowledge that explicit feature engineering promoted by the variability modeling community is crucial, implying non-deep models of lower performance, or we do fully embrace the deep learning advances, allowing ML models to transform the variability space of our systems automatically in ways that domain experts may not understand.*

Giving up on three decades of feature management research since the definition of feature models [12] for performance is not easy. First, it means denying in part the achievements members of the variability management community contributed to, in this conference or other venues. They successfully offered techniques to model, design, test, and verify variability-intensive systems with myriad options. Second, there are categories of critical systems where features need to be interpretable by humans, should they be involved in medical decisions or justice ones. On the other hand, it leads to missing automation and performance opportunities that only deep learning can bring. Thus, our community have to explore trade-offs between explicit and implicit feature engineering. We

also think that this exploration could lead to a new *notion of synthetic feature*, issued from reversible transformations of the original feature space.

Such a graceful combination is required to address complex challenges such as predicting the performance or energy consumption of variability-intensive systems [17]. We think now is the time to do it, as prompt-enabled language models generate code and documentation for features automatically [5].

ACKNOWLEDGMENTS

Gilles Perrouin is an FNRS Research Associate. This work was partly funded by the EOS-VeriLearn, project number 30992574 of the Fonds de la Recherche Scientifique (F.R.S-FNRS) in Belgium.

REFERENCES

[1] Mathieu Acher, Paul Temple, Jean-Marc Jézéquel, José A Galindo, Jabier Martinez, and Tewfik Ziadi. 2018. Varylatex: Learning paper variants that meet constraints. In *Proceedings of the 12th International Workshop on Variability Modelling of Software-Intensive Systems*. 83–88.

[2] Benoit Amand, Maxime Cordy, Patrick Heymans, Mathieu Acher, Paul Temple, and Jean-Marc Jézéquel. 2019. Towards learning-aided configuration in 3D printing: Feasibility study and application to defect prediction. In *Proceedings of the 13th International Workshop on Variability Modelling of Software-Intensive Systems*. 1–9.

[3] Sven Apel, Don Batory, Christian Kästner, and Gunter Saake. 2016. *Feature-oriented software product lines*. Springer.

[4] Thorsten Berger, Daniela Lettner, Julia Rubin, Paul Grünbacher, Adeline Silva, Martin Becker, Marsha Chechik, and Krzysztof Czarnecki. 2015. What is a Feature? A Qualitative Study of Features in Industrial Software Product Lines. In *Proceedings of the 19th International Conference on Software Product Line* (Nashville, Tennessee) *(SPLC '15)*. Association for Computing Machinery, New York, NY, USA, 16–25. https://doi.org/10.1145/2791060.2791108

[5] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating Large Language Models Trained on Code. https://doi.org/10.48550/ARXIV.2107.03374

[6] Andreas Classen, Patrick Heymans, and Pierre-Yves Schobbens. 2008. What's in a Feature: A Requirements Engineering Perspective. In *Fundamental Approaches to Software Engineering*, José Luiz Fiadeiro and Paola Inverardi (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 16–30.

[7] Paul Clements and Linda M. Northrop. 2001. *Software Product Lines : Practices and Patterns*. Addison-Wesley Professional, Boston, USA.

[8] Salah Ghamizi, Maxime Cordy, Mike Papadakis, and Yves Le Traon. 2019. Automated Search for Configurations of Convolutional Neural Network Architectures. In *Proceedings of the 23rd International Systems and Software Product Line Conference-Volume A*. 119–130.

[9] Jianmei Guo, Krzysztof Czarnecki, Sven Apel, Norbert Siegmund, and Andrzej Wąsowski. 2013. Variability-aware performance prediction: A statistical learning approach. In *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 301–311.

[10] Chris Harris, Mike Stephens, et al. 1988. A combined corner and edge detector. In *Alvey vision conference*, Vol. 15. Manchester, UK, 10–5244.

[11] Piotr Indyk and Rajeev Motwani. 1998. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*. 604–613.

[12] Kyo C Kang, Sholom G Cohen, James A Hess, William E Novak, and A Spencer Peterson. 1990. *Feature-oriented domain analysis (FODA) feasibility study*. Technical Report. Carnegie-Mellon Univ Pittsburgh Pa Software Engineering Inst.

[13] David G Lowe. 2004. Distinctive image features from scale-invariant keypoints. *International journal of computer vision* 60, 2 (2004), 91–110.

[14] Fabio Pierazzi, Feargus Pendlebury, Jacopo Cortellazzi, and Lorenzo Cavallaro. 2020. Intriguing Properties of Adversarial ML Attacks in the Problem Space. In *2020 IEEE Symposium on Security and Privacy (SP)*. 1332–1349. https://doi.org/10.1109/SP40000.2020.00073

[15] Awais Rashid, Jean-Claude Royer, and Andreas Rummler. 2011. *Aspect-Oriented, Model-Driven Software Product Lines The AMPLE Way*. Cambridge University Press. 470 pages. https://hal.archives-ouvertes.fr/hal-00620981 ISBN: 9780521767224.

[16] Cordelia Schmid, Roger Mohr, and Christian Bauckhage. 2000. Evaluation of interest point detectors. *International Journal of computer vision* 37, 2 (2000), 151–172.

[17] Norbert Siegmund, Johannes Dorn, Max Weber, Christian Kaltenecker, and Sven Apel. 2022. Green Configuration: Can Artificial Intelligence Help Reduce Energy Consumption of Configurable Software Systems? *Computer* 55, 3 (2022), 74–81.

[18] Norbert Siegmund, Alexander Grebhahn, Sven Apel, and Christian Kästner. 2015. Performance-influence models for highly configurable systems. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*. 284–294.

[19] Norbert Siegmund, Sergiy S Kolesnikov, Christian Kästner, Sven Apel, Don Batory, Marko Rosenmüller, and Gunter Saake. 2012. Predicting performance via automated feature-interaction detection. In *2012 34th International Conference on Software Engineering (ICSE)*. IEEE, 167–177.

[20] Paul Temple, José A Galindo, Mathieu Acher, and Jean-Marc Jézéquel. 2016. Using machine learning to infer constraints for product lines. In *Proceedings of the 20th International Systems and Software Product Line Conference*. 209–218.

[21] Paul Temple, Gilles Perrouin, Mathieu Acher, Battista Biggio, Jean-Marc Jézéquel, and Fabio Roli. 2021. Empirical assessment of generating adversarial configurations for software product lines. *Empirical Software Engineering* 26, 1 (2021), 1–49.