WASP – Software Engineering and Cloud Computing
Assignment Module 2: Software Engineering

Student: Willem Meijer (willem.meijer@liu.se)
June 12, 2025

Introduction

The overarching goal of my research is to improve quality assurance methods in systems that integrate machine learning (ML) components. In my context, this is predictive ML, so not generative ML, like LLMs. This type of software is increasingly present in modern systems and is expected to grow substantially in the future. However, even though companies and individuals *want* to integrate ML components and applicable ML technologies are generally there, *how* they integrate them effectively is generally underexplored, while challenging. This places my research in the field of SE4ML, using software engineering principles to a ML context.

ML algorithms commonly make very stringent assumptions about their data, like formats, distributions, or the type of information represented in the data. However, although most ML algorithms will fairly trivially run on data that does not conform to these assumptions (either during construction or during deployment), the algorithm's effectiveness will generally be negatively affected. This could be obvious, like the prediction quality dropping substantially, however, this impact is generally much more subtle (e.g., a gradual decline over time) or only present in a particular subset of your data (e.g., people of a particular age group or data generated by specific devices). Specifically, our project addresses if and how ML engineers currently address these issues, develop tooling to support them in this process, and ultimately integrate this tooling into development pipelines.

At a high level, our research questions are how to do these three things: What are the development and integration issues faced by ML engineers? How can tooling detect these issues while writing ML code? And how can such tooling be integrated into ML engineering pipelines to detect regressions? Overall, this will allow companies, institutions, etc. that deploy systems with ML components to better safeguard the systems' *internal* quality.

Lecture principles

Because my background is in software engineering, with a great interest in the human aspects of software engineering and some formal education in psychology, most of the topics discussed were not new to me. However, some of the testing methods presented that could help when verifying/validating your ML components/algorithms did prove quite insightful. Specifically, invariant/metamorphic and anchor testing, and data slicing in the context of intersectional biases in data/ML algorithms; a topic I will likely soon start a new collaboration on, but have not yet had the time to read up on. Invariant and anchor testing allow us to evaluate whether an algorithm does or does not latch onto features of interest. In the context of fairness and discrimination in ML algorithms, this is incredibly important as it allows us to identify whether the algorithm classifies according to, for example, sensitive properties (gender, socio-economic status, etc.) if these are present in raw data (addressing the entire ML pipeline or ML-including system as a black-box). Additionally, if a party (company, institute, etc.) makes a case that particular sensitive feature is imperative to their use-case (e.g., as income is to bank loans), we could potentially leverage an anchor-like approach to systematically evaluate whether this is indeed one of the core features of the model, or merely an auxiliary one, and judge whether its use is warranted. Finally, as was mentioned in the lecture, data slicing could be useful to identify whether a ML algorithm performs better on cases with particular properties compared to others, giving us the opportunity to, for example, identify edge cases which we could emphasize during future training or use as a candidate population for additional data collection.

Guest lecture principles

Although not entirely new, one idea I took from the guest lectures is Per Lenberg's description of risk-averse decision making when introducing new components into (existing) systems. Obviously, his case is very specific to aeronautics, which is a highly regulated discipline where it is inherently challenging to verify the quality and correctness of your system from a technical perspective. However, the importance of social factors, to which ML components are especially subject, such as distrust of these new systems or the fact that they challenge the status quo, is easy to overlook when designing a new and helpful system. In some sense, this affects my work twofold. First, because I will develop a tool that could be adopted in company settings, which will be subject to such resistance. Second, because the tool provides a series of guarantees about components that might be integrated into the larger system, making them more trustworthy. Because ML components are inherently harder to trust, the latter could reduce this — at least to a subset of people involved in building the system.

Data scientists versus software engineers

Data scientists and software engineers. Although I am certain it holds for some people, I think the book's description of data scientists and software engineers is quite reductive. It is certainly not sustainable as, for example, the number of (university) programs on data science increasingly teach a combination of software engineering and data courses. Anecdotally, I took courses in both, which was already three years ago. I am sure there are globally fundamental differences; there have to be, as *"software engineer"* and *"data scientist"* are two extremely broad terms. The definition of a software engineer includes web (and cloud) engineering (which is more or less what the book uses) and embedded systems engineering (which the book disregards completely). Similarly, a data scientist could do anything from reinforcement learning to pure math to basic statistics. Consequently, I think that comparisons between these classes of people are not very useful when they are not placed in context. If people choose to specialize in their respective domains, the differences will clearly be great. However, if they do not, the difference is more or less meaningless. It is only a simple decision and a matter of priorities. However, at that point, we will call them a data pipeline operator, RLOps developer, applied statistician, or ML product owner.

Role evolution. Whether roles will evolve depends greatly on context as well. While everybody does everything in start-ups, you could specialize in your favorite niche when you work at a huge organization. Sure, once operationalization and deployment of ML software become mainstream, you will need people with a different set of skills. A company that toys around with ML pipelines will not need an ML monitoring expert, whereas a company with 30 live ML services certainly does. In my experience, after talking with several companies, the maturity of ML systems differs wildly right now. However, as these systems generally mature, new data engineers will increasingly need specialized skills, which inevitably involves learning from *"the other side."* An ML monitoring expert will, for example, need to be able to interpret trends in both ML-specific and software-specific performance metrics, like classification error and CPU utilization, and design interventions where necessary.

Paper analysis

Data smells. The work of Recupito et al. (2024), which builds on the work of Foidl et al. (2022), lies close to my work, as it proposes a taxonomy of *"data smells"* in learning software. These are errors in ML pipelines' input data, which they argue can have negative consequences on the system's quality. In this context, they relate these smells to dataset quality, measured using metrics of uniqueness, consistency, readability, and completeness. To some extent, these indicate the amount of information contained in the dataset and how clean the data is. They find a total of 50 smells, comprising eight overarching themes that range from low-level issues, like storing data in the wrong format or missing values, to higher-level issues, like outliers, correlated features, and inclusion of sensitive features. They continue to identify how frequently these smells occur in datasets, and, ultimately, whether the presence of the top five smells affects dataset quality. Unsurprisingly, these smells relate to dataset quality; however, they affect it in different ways.

This paper takes a very data-oriented perspective on ML engineering — which is completely fair. However, my work (Meijer, 2024) builds upon it by evaluating whether such smells are present in ML pipelines, and, more importantly, whether they are dealt with. For example, if you remove some of the correlated features from your dataset, it does not matter whether they were originally present, as you dealt with them accordingly. Specifically, we currently work on a static analysis tool for learning software, evaluating incompatibilities between datasets and algorithms.

Static analysis. Various solutions that claim to detect these issues exist already. For example, the static analyzer shared in the lecture (Quaranta et al., 2024), or the second CAIN paper I want to discuss (Shivashankar and Martini, 2025). The work of Shivashankar and Martini (2025) proposes a linter for ML smells, capturing 76 smells distributed across five popular ML libraries by analyzing the abstract syntax tree (AST) of ML pipelines written in Python. They performed a small-scale evaluation, finding that its precision is 1 for all classified notebooks and its recall ranges between 0.75 and 1, suggesting a limited number of false flags (which static analyzers are notorious for). Further, their small-scale user study suggests some benefits of such a tool in practice (scoring between 3 and 4 with 0.3 to 0.8 variance on a Likert scale of 1 to 5).

One major drawback of these tools is that they generally do not comprehensively address the relationship between algorithms and data, but instead re-implement primitive (and long-known) rules, like placing your dependencies at the top of your script (Quaranta et al., 2024), or make strong assumptions about the the relationship between different function calls, for example, using whether the number of unique elements in any feature of a dataset is printed as an indicator of class imbalance (Shivashankar and Martini, 2025). Consequently, in my experience, the tests done are either not specific to ML at all or yield an unnecessary number of false reports/misses due to unruly assumptions. Although the reported evaluation of the last tool is positive, I am not convinced in the slightest that it will translate well to real development environments, as printing unique values is by no means an indicator of the imbalance of your classes. Arguably, it is a good practice in any scenario that could lead to the healthy conclusion that there is no class imbalance. I zoom in on class imbalance, as I consider it is a fundamental issue in many ML algorithms, however, I noticed similar issues with their rules regarding improper feature selection, which draw a strange relationship with validation strategies, rules regarding large dataset loading, which require users to print the file size to enable the rule, and the generally very strong assumptions that rules make about used variable and file names (for example, any file that has a name starting with `test_` or `helper` will never violate a rule).

Adaptation. Both of these lines of research (Recupito et al., 2024; Shivashankar and Martini, 2025) are essential to correct ML development. In *"regular software"* it is already easy to make mistakes, in my opinion, this is only amplified in the context of ML due to its inherent complex relationship between code and datasets — especially when you consider that a large portion of *"bugs"* in ML software do not cause crashes but present themselves as bias, poor generalization, etc., which exclusively occur after you have trained your model. Obviously, training is a very time consuming and costly task, making the detection of these issues prior to training economically interesting; beyond the social benefits they give by preventing the deployment of buggy ML software.

Consequently, because of its importance and the limited depth of existing solutions, my work (Meijer, 2024) tries to push the envelope by actively involving data in static analysis for ML code. Going back to the class imbalance example, this will allow us highlight this problem exclusively when such an imbalance is actually present in the data. This allows us to test the problems given by Recupito et al. (2024), but also allows us to go much further, testing for grouping factors, distribution assumptions, cardinality, dimensionality, etc. as well. This allows us to find functional incorrectness in ML software, but also non-functional issues as some algorithms *"only"* become much less efficient given particular input data.

Going into a mild amount of detail, we use ASTs of ML code written in Python (like Shivashankar and Martini (2025)) and translate it to a construct similar to to control-/data-flow graphs. These graphs have real datasets at its roots, that can be accessed to calculate data properties (data ranges, means, correlation, etc.; essentially, any property you can think of). Then, for each ML method (e.g., the `fit` on `sklearn`'s `LogisticRegressionClassifier`), we can identify its input data and hyperparameters, what properties we expect the data to have with respect to those hyperparameters, and how they have been transformed throughout the pipeline to see if the data still has those properties. Of course, this process is still subject to assumptions and will inevitably cause false classifications; however, compared to existing solutions (Shivashankar and Martini, 2025; Quaranta et al., 2024), our assumptions are at the very least inherent to ML and are directly connected to the thing we make the assumptions about: the data.

Research ethics and synthesis reflection

I am not quite sure what is expected in this section. However, my search strategy was fairly straightforward. Because my research is closely related to the things published in CAIN, I chose to pick papers that are relevant to my work, and emphasized recently published ones. First, I searched through the papers in my reference manager, specifically the ones I cited previously. Through Meijer (2024), where I cited various CAIN papers, I found Recupito et al. (2024). After, I sieved the titles of the 2025 Research and Experience Papers[1] for relevant works, finding Shivashankar and Martini (2025), which is the only one directly related to my work. Having said that, the work by Vonderhaar et al. (2025) is likely closely related to a potential future study as well.

[1]https://conf.researchr.org/track/cain-2025/cain-2025-call-for-papers#Papers-Accepted

Although I did not read all abstracts, I did not spot any misleading titles/abstracts. However, I think that Shivashankar and Martini (2025) is not a great paper (as might have been clear from my critical reflection earlier).

Beyond Grammarly as a spelling/grammar checker, I have not used LLMs for this assignment whatsoever.

References

Foidl, H., Felderer, M., and Ramler, R. (2022). Data smells: categories, causes and consequences, and detection of suspicious data in ai-based systems. In *Proceedings of the 1st International Conference on AI Engineering: Software Engineering for AI*, pages 229–239.

Meijer, W. (2024). Contract-based validation of conceptual design bugs for engineering complex machine learning software. In *Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems*, MODELS Companion '24, page 155–161, New York, NY, USA. Association for Computing Machinery.

Quaranta, L., Calefato, F., and Lanubile, F. (2024). Pynblint: A quality assurance tool to improve the quality of python jupyter notebooks. *SoftwareX*, 28:101959.

Recupito, G., Rapacciuolo, R., Di Nucci, D., and Palomba, F. (2024). Unmasking data secrets: An empirical investigation into data smells and their impact on data quality. In *Proceedings of the IEEE/ACM 3rd International Conference on AI Engineering-Software Engineering for AI*, pages 53–63.

Shivashankar, K. and Martini, A. (2025). Mlscent a tool for anti-pattern detection in ml projects. In *Proceedings of the 4th international conference on AI engineering: software engineering for AI*.

Vonderhaar, L., Elvira, T., and Ochoa, O. (2025). Generating and verifying synthetic datasets with requirements engineering. In *Proceedings of the 4th international conference on AI engineering: software engineering for AI*.