Towards Declarative Systems for Data-Centric Machine Learning

Stefan Grafberger [1] **Bojan Karlaš** [2] **Paul Groth** [1] **Sebastian Schelter** [1]

Abstract

We argue for a declarative approach to simplify the application of data-centric ML in real-world scenarios, and present our prototypical system MLWHATIF, which takes a first step in this direction.

1. Introduction

The process of developing machine learning (ML) applications has recently been going through a fundamental shift. An emerging set of *data-centric operations* is becoming indispensable for making ML models perform better and for understanding the causes of their failure (Liang et al., 2022). Examples of such operations (along with some prominent *methods* for conducting them) include: data debugging (Jia et al., 2019; Northcutt et al., 2021), slice discovery (Chung et al., 2019) and data pruning (Sorscher et al., 2022). Given the large number of promising methods developed by the community along with recent benchmarking efforts (Mazumder et al., 2022), the field is well-positioned to impact real-world ML development. However, there are two key challenges.

Real-world challenges. First, in real-world ML settings, input data is typically spread across different data sources (e.g., data lakes, data warehouses, REST APIs, ...), is regularly updated, and must be integrated and converted to features in order to be consumable by ML systems (Sculley et al., 2015; Polyzotis et al., 2018; Schelter et al., 2018). However, most existing data-centric methods do not account for this situation, but instead expect a single, often static, already prepared input dataset. Because of this discrepancy, current data-centric methods are either not directly applicable to real-world settings, or integrating them forces data scientists and ML engineers to implement custom one-off solutions for each use case.

[1]AIRLab, University of Amsterdam, Amsterdam, The Netherlands [2]Department of Biomedical Informatics, Harvard University, Boston, US. Correspondence to: Stefan Grafberger <s.grafberger@uva.nl>.

Proceedings of the Data-Centric Machine Learning Research workshop (DMLR) at ICML, 2023.

Managing this increased complexity only adds to the high operational overhead in the job of data scientists, who already spend more than 60% of their time on data preparation tasks (Anaconda.com, 2020).

Second, for each data-centric operation, there is a growing number of applicable methods. However, given that the success of these methods often varies on a case-by-case basis, it is unclear beforehand which one to choose for a given scenario. Consequently, developers need to repeatedly and manually re-write their experimental code (e.g., via messy Jupyter notebooks) to try out different methods (and orchestrate the repeated data access and model inference or retraining), which is time-consuming and error-prone.

Towards automation and declarativity. We argue that we need to increase the level of automation of data-centric ML development to improve this situation. We should enable end users to *declaratively* state which data-centric operation they want to apply, while shielding them from the complexity of interacting with different methods and their implementations.

Furthermore, there should be an underlying system that can *automate* the process of selecting a well-performing method. Declarativity and automation are at the core of many successful computing approaches from the last decades: for example, deep learning engines like Pytorch and Tensorflow let users declaratively define the computational graph of their model and automate the subsequent model training on accelerator hardware. Analogously, users should be able to apply data-centric ML (e.g., cleaning label errors) with a simple declarative configuration and a few lines of code.

Contribution. The purpose of this abstract is to make the community aware of this challenge (Section 2) and to present our prototypical system MLWHATIF (Grafberger et al., 2023) available open source at `https://github.com/stefan-grafberger/mlwhatif`. Based on a declarative configuration, MLWHATIF can automatically rewrite ML code (written with libraries such as pandas, sklearn or keras) to execute multiple data-centric methods, and returns interpretable reports which allow data scientists to identify a well-performing method for a given case (Section 3).

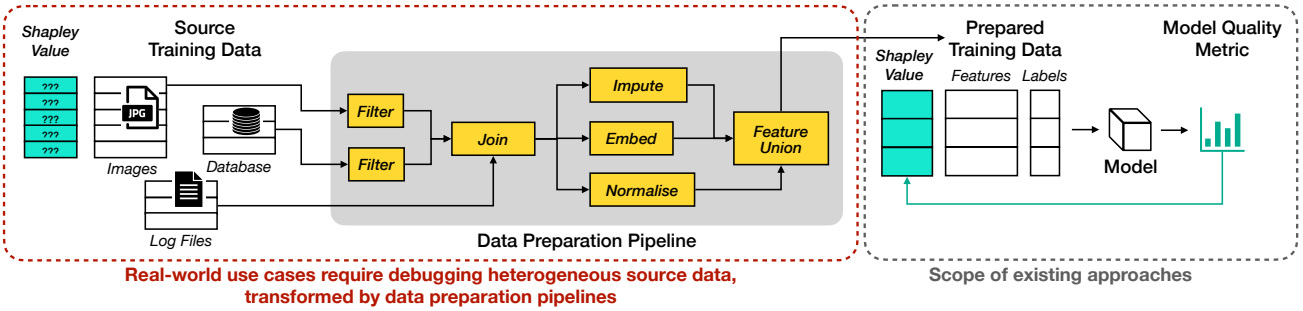

Figure 1. Development challenges of train set debugging real-world scenarios.

2. Example – Training Set Debugging

We discuss unsolved challenges of developing data-centric ML applications for real-world scenarios on the task of training set debugging (Mazumder et al., 2022), where the goal is to identify mislabeled samples that negatively impact the performance of a model.

Implementation overheads. First of all, there are various methods to identify label errors (Jia et al., 2019; Northcutt et al., 2021) and to fix them (e.g., by removing mislabeled samples, auto-repairing them or asking humans to relabel them). It is not obvious which method works best in a real use case, therefore data scientists have to integrate several of these methods into their ML code to make a decision.

Identifying mislabeled source data. Secondly, as depicted in Figure 1, the data to debug in a real-world setup is typically not a single static dataset, but originates from many different heterogenous data sources and may be multi-modal (e.g., contain images, text and relational data). While existing methods are designed to debug featurized, already prepared training data, we are ultimately interested in debugging a particular source dataset, which is combined with other source data and transformed into featurized training data by a data preparation pipeline. The featurized training data is usually not understandable for human experts when relabeling records. Furthermore, label errors need to be fixed in the original source data, not the derived prepared training data. This requires mapping training data matrix entries to their corresponding entries in the original data sources, and, importantly, also requires propagating updates to the original data sources through preprocessing code.

3. Overview of MLWHATIF

We give an overview of our system MLWHATIF (Grafberger et al., 2023), which tackles the implementation overhead challenge.

Data preparation pipelines as a first-class citizen. MLWHATIF reasons about the data preparation operations in ML code by modeling a data preparation pipeline as a dataflow computation (Grafberger et al., 2023), inspired by the relational algebra from data management. We treat a classification task as a dataflow computation from several source datasets as input to a set of ML-specific matrices as output, e.g., for features, labels, and predictions. Furthermore, we model all pipeline operations with operators from the positive relational algebra (Abiteboul et al., 1995). MLWHATIF can extract such a dataflow plan from declaratively written ML code with common Python libraries via code instrumentation (Grafberger et al., 2022).

Automatic application of data-centric analyses to existing ML code. Given this dataflow plan, MLWHATIF can identify different pipeline operations (such as accessing data sources, relational preprocessing, feature extraction, model training, and label encoding) and re-execute them on changed inputs. It leverages so-called "pipeline patches" as a formal framework to re-write ML pipelines. These patches describe changes to the dataflow plan and are the basis for applying existing data-centric methods automatically to existing code without the need for manual integration.

Example. We provide an example notebook with training set debugging applied to a simple computer vision pipeline, which resembles the example in Figure 1, and consumes images, relational data and logfiles. The notebook is available at `https://github.com/stefan-grafberger/mlwhatif/blob/1877736feb3fb71acd38a61a37cb0216f7595523/demo/dc_demo/dc-demo.ipynb` and shows how to declaratively and automatically run label error detection and repair with *kNN-Shapley* (Jia et al., 2019; Karlaš et al., 2022) and *cleanlab* (Northcutt et al., 2021) on the code:

```
debugging = TrainsetDebugging([KNN_SHAPLEY, CLEANLAB])
report = mlwhatif
  .on_pipeline("cv-pipeline.py") \
  .run(debugging)
print(report)
```

MLWHATIF automatically rewrites the ML code to execute different label error detection and cleaning methods. It outputs a report, which details how these methods impact the model performance, based on which the data scientist can decide which data-centric ML method to finally choose.

```
+-------------+-----------+----------+
| method      | cleaning  | accuracy |
+-------------+-----------+----------+
| None        | None      | 0.964    |
| knn-shapley | remove    | 0.951    |
| knn-shapley | repair    | 0.982    |
| cleanlab    | remove    | 0.960    |
| cleanlab    | repair    | 0.965    |
+-------------+-----------+----------+
```

Future work. We will extend the number of data-centric methods supported by MLWHATIF and integrate it with frameworks for data preparation such as *mlflow recipes* (Databricks, 2023). Furthermore, we will also tackle the challenge of mapping the analysis results back to source data items, by building on our work on fine-grained data provenance over data preparation pipelines (Schelter et al., 2022).

Acknowledgements. This work was supported by Ahold Delhaize. All content represents the opinion of the authors, which is not necessarily shared or endorsed by their respective employers and/or sponsors.

References

Abiteboul, S., Hull, R., and Vianu, V. *Foundations of Databases*. Addison-Wesley, 1995. ISBN 0-201-53771-0.

Anaconda.com. The State of Data Science. https://www.anaconda.com/state-of-data-science-2020, 2020.

Chung, Y., Kraska, T., Polyzotis, N., Tae, K. H., and Whang, S. E. Slice finder: Automated data slicing for model validation. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pp. 1550–1553. IEEE, 2019.

Databricks. Mlflow Recipes. https://mlflow.org/docs/latest/recipes.html, 2023.

Grafberger, S., Groth, P., Stoyanovich, J., and Schelter, S. Data distribution debugging in machine learning pipelines. *VLDBJ*, 2022.

Grafberger, S., Groth, P., and Schelter, S. Automating and optimizing data-centric what-if analyses on native machine learning pipelines. *SIGMOD*, 2023.

Jia, R., Dao, D., Wang, B., Hubis, F. A., Gurel, N. M., Li, B., Zhang, C., Spanos, C. J., and Song, D. Efficient task-specific data valuation for nearest neighbor algorithms. *VLDB*, 2019.

Karlaš, B., Dao, D., Interlandi, M., Li, B., Schelter, S., Wu, W., and Zhang, C. Data debugging with shapley importance over end-to-end machine learning pipelines. *arXiv preprint arXiv:2204.11131*, 2022.

Liang, W., Tadesse, G. A., Ho, D., Fei-Fei, L., Zaharia, M., Zhang, C., and Zou, J. Advances, challenges and opportunities in creating data for trustworthy ai. *Nature Machine Intelligence*, 4(8):669–677, 2022.

Mazumder, M., Banbury, C., Yao, X., Karlaš, B., Rojas, W. G., Diamos, S., Diamos, G., He, L., Kiela, D., Jurado, D., et al. Dataperf: Benchmarks for data-centric ai development. *arXiv preprint arXiv:2207.10062*, 2022.

Northcutt, C., Jiang, L., and Chuang, I. Confident learning: Estimating uncertainty in dataset labels. *JAIR*, 70, 2021.

Polyzotis, N., Roy, S., Whang, S. E., and Zinkevich, M. Data lifecycle challenges in production machine learning: a survey. *SIGMOD Record*, 47(2), 2018.

Schelter, S., Biessmann, F., Januschowski, T., Salinas, D., Seufert, S., and Szarvas, G. On challenges in machine learning model management. *IEEE Data Engineering Bulletin*, 2018.

Schelter, S., Grafberger, S., Guha, S., Sprangers, O., Karlaš, B., and Zhang, C. Screening native ml pipelines with "arguseyes". 2022.

Sculley, D., Holt, G., Golovin, D., Davydov, E., Phillips, T., Ebner, D., Chaudhary, V., Young, M., Crespo, J.-F., and Dennison, D. Hidden technical debt in machine learning systems. *Advances in neural information processing systems*, 28, 2015.

Sorscher, B., Geirhos, R., Shekhar, S., Ganguli, S., and Morcos, A. Beyond neural scaling laws: beating power law scaling via data pruning. *Advances in Neural Information Processing Systems*, 35:19523–19536, 2022.