Towards Data-Centric What-If Analysis for Native Machine Learning Pipelines

Stefan Grafberger
s.grafberger@uva.nl
AIRLab, University of Amsterdam

Paul Groth
p.groth@uva.nl
University of Amsterdam

Sebastian Schelter
s.schelter@uva.nl
University of Amsterdam

ABSTRACT

An important task of data scientists is to understand the sensitivity of their models to changes in the data that the models are trained and tested upon. Currently, conducting such *data-centric what-if analyses* requires significant and costly manual development and testing with the corresponding chance for the introduction of bugs. We discuss the problem of data-centric what-if analysis over whole ML pipelines (including data preparation and feature encoding), propose optimisations that reuse trained models and intermediate data to reduce the runtime of such analysis, and finally conduct preliminary experiments on three complex example pipelines, where our approach reduces the runtime by a factor of up to six.

ACM Reference Format:
Stefan Grafberger, Paul Groth, and Sebastian Schelter. 2022. Towards Data-Centric What-If Analysis for Native Machine Learning Pipelines. In *Data Management for End-to-End Machine Learning (DEEM'22), June 12, 2022, Philadelphia, PA, USA*. ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3533028.3533303

1 INTRODUCTION

An important task of data scientists is to understand the sensitivity of their models to changes in the data that the models are trained and tested upon. What if my testing data has more errors than my training data? What if there are high amounts of missing data? Currently, these *data-centric what-if analyses* require significant and costly manual development and testing with the corresponding chance for the introduction of bugs.

Such analyses are becoming increasingly relevant with the current push towards *data centric AI* and are closely related to recently proposed benchmark tasks [3] in this direction. Examples for such data-centric what-if analyses include the investigation of the *robustness of ML models against data errors* in train and test data [11, 13], *data cleaning on a budget*, where one has to select a small numbers of data points to clean in order to improve model performance [9, 10], or understanding the *impact of different data preprocessing operations on the fairness* of the resulting ML model [5]. In contrast to this and other existing work which is either limited to relational queries [7] or models with a static input dataset [15], we aim to enable data scientists to automatically run such data centric analysis tasks over whole ML pipelines (which include data preparation and feature encoding operations). In particular, we focus on *natively written* ML pipelines, e.g., pipelines that use code from established libraries from the data science ecosystem, such as pandas, scikit-learn or keras, to enable data scientists to easily apply our techniques without having to manually annotate their code or rewrite it to a new language or library. The automation and optimisation of data-centric what-if analyses impacts not just the efficiency of the execution, but also the coverage of potential problems that may have substantial business and social impact, as well as better reuse and sharing of these experiments.

In this paper, we discuss the problem of data-centric what-if analysis over ML pipelines using a code example from the healthcare domain in Section 2, and describe how a data scientist would manually rewrite the code to conduct the analysis. Next, we propose directions on how to optimise the execution of the what-if analysis on such pipelines in Section 3 by modeling these pipelines as dataflow computations and reusing trained models and intermediate data. Subsequently, we conduct preliminary experiments for two types of data-centric what-if analysis (robustness against data errors and sample selection strategies for data cleaning on a budget) in Section 4. We evaluate the runtime benefits of our proposed optimised execution on three complex pipelines. Finally, we discuss the next steps for the proposed research direction. In summary, this paper provides the following contributions:

- We discuss the problem of data-centric what-if analysis over ML pipelines (Section 2).
- We propose optimisations that reuse trained models and intermediate data to reduce the runtime of such analysis (Section 3).
- We conduct preliminary experiments on two kinds of what-if analyses and three complex example pipelines, where our approach reduces the runtime by a factor of up to six (Section 4).

2 PROBLEM STATEMENT

We discuss automating and optimising the execution of data centric what-if analysis on ML pipelines with a running example.

Running example. Imagine a data scientist in health care has developed the pipeline shown in Listing 1 to create a classifier that predicts potential complications for patient treatments. The resulting pipeline integrates patient and treatment data (Lines 1-6), conducts a temporal split in Lines 8 & 9 and encodes the following features for prediction in Lines 11-16: the `age` and `weight` of patients, information on whether they `smoke` or follow a `vegetarian` diet, and textual `notes` from the treating doctor. Finally, the ML pipeline trains and evaluates a logistic regression model to predict potential complications for patients (Lines 18-21).

DEEM'22, June 12, 2022, Philadelphia, PA, USA

ACM ISBN 978-1-4503-9375-1/22/06…$15.00
https://doi.org/10.1145/3533028.3533303

What-if analysis. Imagine that our data scientist is now concerned about the robustness of the resulting model to potential errors in the data. What if a larger number of patients chose not to report their weight? What if doctors start to only write very short notes (or have a lot of typos in them) in stressful periods? The data scientist would like to evaluate what happens to the model predictions in these cases. Also, she deems it important to determine whether these errors would negatively affect the fairness of the classifier, e.g., would the prediction quality be impacted for certain age groups? Furthermore, the data scientist would like to explore whether it is beneficial to simulate such errors at training time and augment the data with them.

```
# Data loading
patients = pd.read_csv("s3://...")
treatments = pd.read_csv("s3://...")
# Integration and filtering
histories = patients.join(treatment, on="patient_id")
histories = histories[patients.consent==True]
# Temporal train/test split
train = histories[histories.date<2019]
test = histories[histories.date>=2019]
# Declaratively defined (nested) feature encoding pipeline
pipeline = Pipeline([
  ('features', ColumnTransformer([
    (StandardScaler(), ["age", "weight"]),
    (Pipeline([SimpleImputer(), OneHotEncoder()])),
      ["smoker", "vegetarian"]),
    (HashingVectorizer(n_features=100), "notes")])),
  # ML model for learning
  ('learner', LogisticRegression())])
# Train and evaluate model
model = pipeline.fit(train, train.had_complications)
print(model.score(test, test.had_complications))
```

Listing 1: Example pipeline for health care use case.

In summary the data scientist would like to quantify the effect of the following four scenarios on the accuracy and a given fairness metric of the model:

- **(S1)** *What-if there were a high number of missing values in the* `weight` *attribute at inference time?*
- **(S2)** *What-if there were a high number of typos in the* `notes` *attribute at inference time?*
- **(S3)** *What-if there were a high number of missing values in the* `weight` *attribute at inference time, but the model saw similar errors at train time?*
- **(S4)** *What-if there were a high number of typos in the* `notes` *attribute at inference time, but the model saw similar errors at train time?*

Manually implementing the what-if analysis. Our data scientist would likely have to spend several hours to manually implement the what-if analysis for the pipeline. She would start to write custom code for evaluating the scenarios, and copy and re-use code from the existing pipeline (which is tedious and dangerous as code bases might easily become out of sync). During the implementation, it would become clear that there are several ways to accelerate the execution of all these scenarios. For example, (S1) and (S2) both use the same model trained on the original clean train data, and have to evaluate this model on different corrupted versions of the test data. The data scientist could persist the trained model (e.g., via `pickle.dumps(model)`) and as well as the test data (e.g., via the `to_csv` function of pandas). In the experimental code, the data scientist then loads both the persisted model and test data, uses

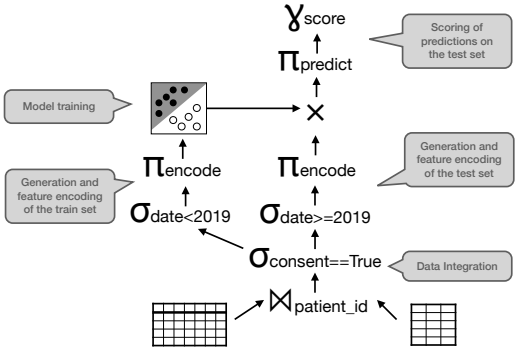


Figure 1: Classification pipeline for our running example modeled as a dataflow computation.

the Python library *Jenga* [13] to simulate data corruptions in the `weight` and `notes` attribute, and handcodes the fairness metric evaluation by retrieving the confusion matrix for the predictions on the dirty data via scikit-learn's `metrics` API. The implementation of the scenarios (S3) and (S4) is more complicated because the data scientists needs to corrupt the training data and train a separate custom model for both scenarios. The test data, however, is corrupted analogously to the scenarios (S1) and (S2). That means that this data could be re-used if the data scientist thought about this and persisted it. In the end, the data scientist would have to compute predictions on the corrupted test data with the custom models and reuse the custom code to evaluate the fairness metrics

Automation and optimisation potential. Instead of forcing the data scientist to spend a long time on manually implementing these what-if analyses, we should provide means to automate them. Our vision is to allow data scientists to declaratively specify a given what-if scenario, and to automatically generate and execute the required workloads based on their existing pipeline. Such a method would reduce the analysis effort of the data scientist and the time-to-deployment of the model. At the same time, it would avoid potential errors that can occur during manual copying and editing of pipeline code. Furthermore, the execution of the what-if analysis tasks should be jointly optimised for all scenarios, for example by identifying and leveraging re-use opportunities for intermediate models and data. Finally, several tasks, such as the computation of fairness metrics, can also be automated via data provenance techniques [8, 12].

3 APPROACH

In the following, we describe our preliminary ideas to automate and optimise the data centric what-if analysis for ML pipelines as outlined in the previous section.

Modeling ML pipelines as dataflow computations. We base our approach on a recently introduced model of treating ML pipelines for classification tasks as dataflow computations [8, 12]. This allows us to reason about the operations and intermediate results in such pipelines as well as to modify them. We model a classification pipeline as a dataflow computation from several input relations in a star schema to a set of ML-specific matrices, e.g., for features,

labels, and predictions. The data integration stage of a classification pipeline combines the individual input datasets into a single table by conducting a series of joins. The subsequent data cleaning and filtering operations can then be modeled with selections and (extended) projections to remove tuples and remove/recompute attributes. The final feature encoding stage turns the data into matrix form with operations such as one-hot-encodings, which we again treat as extended projections. Formally, all these stages can be modelled with operations from the positive relational algebra (SPJU; selection, projection, join, union).

Figure 1 illustrates the corresponding dataflow graph for our running example from the previous section. The pipeline integrates and filters the input datasets by joining them on the `patient_id` and removing all records where a patient has not given `consent` to data analysis with a selection. Afterwards, the pipeline conducts a temporal split to generate train and test data with a selection on the `date` attribute. Next, the train data is encoded with extended projections and the model training is executed. The pipeline analogously encodes the test data with extended projections and applies the model to the result to generate predictions. Finally, the predictions are scored (e.g., to compute the model's accuracy on the test set) via an aggregation.

Optimised what-if workload execution. A naive baseline approach to execute what-if analysis tasks over this pipeline would be to generate all required different variants of the pipeline, and execute them sequentially. However, as discussed in the previous section, there is a lot of potential for reuse of models and data. In an optimised execution, we should avoid repeated model training by reusing models that are shared between different variants of the pipeline, and at the same time avoid the re-execution of shared data preparation operations between different variants of the pipeline. Instead, we want to materialise and re-use intermediate results.

Figure 2 illustrates four such proposed optimisations for our running example from the previous section: (*i*) The train and test set (e.g., the results of the selections $\sigma_{\text{date<2019}}$ and $\sigma_{\text{date>=2019}}$) can be reused by all four scenarios; (*ii*) Scenarios (S1) and (S2) share the same model trained on clean training data; (*iii*) Scenarios (S1) and (S3) share the same corrupted test data (missing values in the `weight` attribute); (*iv*) Analogously, the scenarios (S2) and (S4) share the test data corrupted with typos in the `notes` attribute. Furthermore, the optimised execution can additionally apply common query optimisation techniques, e.g., pushing the filter $\sigma_{\text{consent=True}}$ down through the join on `patient_id`. Note that such optimisations can be automatically derived, e.g., by generating a single plan for all naive pipeline variants and identifying and rewriting shared sub-plans via common subexpression elimination [14].

4 PRELIMINARY EXPERIMENTS

Overview. In the following, we experiment with two what-if analysis tasks on three complex exemplary ML pipelines. In each case, we compare the runtime of naive baseline execution (sequentially executing different pipeline variants) to the runtime of optimised execution, where we implement a hardcoded pipeline variant with our proposed optimisations. The source code for our preliminary experiments is available at https://github.com/stefan-grafberger/deem22-what-if-experiments.

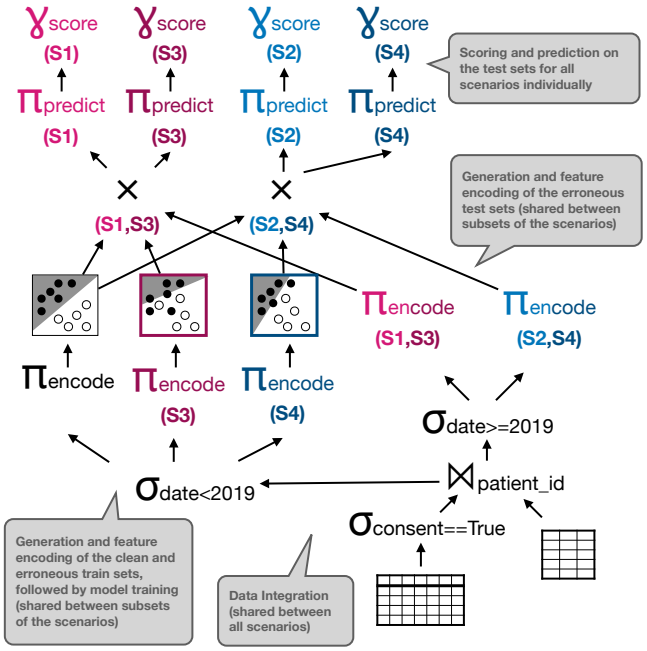
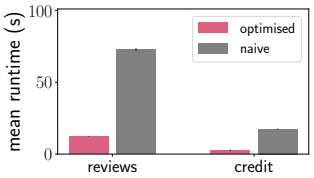


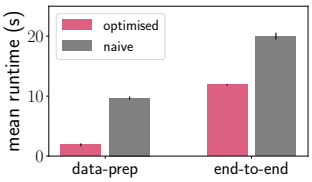
Figure 2: Dataflow of the optimised classification pipeline for what-if analysis with respect to four scenarios on our running example. The optimised pipeline reuses the integrated input data, corrupted test and train sets, as well as learned models for different scenarios.

Example pipelines. We evaluate our proposed optimisations on three different example pipelines, which contain data preparation, feature encoding and model training operations. Their source code is available in the github repository. *Helpful Reviews* identifies helpful reviews for video games from a dataset of customer reviews from Amazon [2]. This pipeline serves as example for pipelines with complex data preprocessing (e.g., containing several joins). *Credit* predicts the income level of a person based on demographic and workplace attributes from the *adult income* dataset [1]. This is a proxy task for decision making about loan applications, which is commonly used in fairness research [6]. This pipeline serves as an example for pipeline with relatively simple data preparation operations. *Product Images* is a pipeline training a convolutional neural network that distinguishes images of sneakers from ankle boots in the *Fashion MNIST* dataset [4]. The purpose of this pipeline is to show that our proposed optimisations are also beneficial for pipelines with non-relational data, employing neural networks.

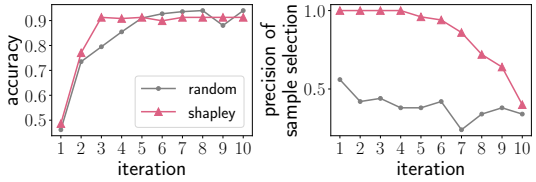
What-If-Analysis 1: Robustness against data errors. We run a what-if analysis for the robustness of a classifier against data errors. We use the Jenga libary [13] to inject data corruptions into randomly chosen rows of the test data of our pipelines and evalute the predictive performance on the resulting model. We repeat this for a setting where we also inject the same type of data corruptions into the training data to see if that helps the model learn to handle such erroneous data. We leverage the reviews pipeline, where we analyse the impact of three different data corruptions: scaling a random fraction of values in the numeric `star_rating` attribute, generating



(a) Runtime comparison for the "robustness against data errors" experiments on the reviews and credit pipelines. The optimised execution reduces the mean end-to-end runtime by more than 5x.



(b) Runtime comparison for the "data cleaning on a budget" experiment on the product images pipeline. The optimised execution reduces the runtime for data preparation operations by more than 5x.



(c) (Left) Accuracy over time for the "data cleaning on a budget" experiment on the product images pipeline. Selecting samples to clean via Shapley values reaches a high accuracy level faster and produces more stable results; (Right) corresponding precision of the mislabeled sample selection, where selection based on Shapley values clearly dominates random selection.

Figure 3: Runtime comparison for the benefits of optimised execution for two what-if analyses (left), and predictive performance for the data cleaning on budget analysis (right).

a distribution shift in the categorical `verified_purchase` attribute, and injecting typos in the textual `review_headline` attribute. For the credit pipeline, we scale values in the numeric `age` attribute, apply a distribution shift in the categorical `workclass` attribute, and inject missing values in the categorical `education` attribute. We apply the selected data corruption to 20%, 50% and 90% of the values.

Results and discussion. We plot the resulting mean runtime of the optimised and naive execution for the what-if analysis in Figure 3(a). For the review pipeline, we observe a runtime decrease by a factor of more than six for the optimised execution (12.12 seconds) compared to naive execution (72.73 seconds). Similarly, we observe a more than 6x runtime decrease for the credit pipeline, where the optimised execution reduces the mean runtime from 17.41 seconds to 2.70 seconds. We attribute this to the fact the optimised version only needs to load the csv input files once in the beginning, and that it does not need to retrain the model for the cases where only the test data is corrupted. Furthermore, our optimised variant does not featurise all columns every time but selectively generates corrupted versions of them. For that, we corrupt 100% of the data once and then sample from this data (for the missing values corruption, we can even generate the corrupted data on the fly). We summarise interesting findings from the what-if analysis. For the reviews pipeline, we observe the strongest negative impact (up to 5% decline in AUC) for scaling corruptions on the `star_rating` attribute, and training on erroneous data does not fix this. For the credit pipeline, we additionally look at the fairness of the model in terms of the difference in false negative rates between white and non-white persons. The accuracy of the model is relatively robust against corruptions in attributes other than `age`. The fairness impact is often uncorrelated with the accuracy impact, e.g., we find a low impact on accuracy for a high error rate in the `workclass` attribute, but a strong negative impact on the fairness.

What-If-Analysis 2: Data cleaning on a budget. Next, we run a what-if analysis for a scenario, where one has limited means to clean the training data for an ML pipeline. We leverage the product images pipeline for this experiment. We generate dirty, mislabeled training data by flipping the target label of 50% of the 1,000 training images, chosen at random. Next, we select 50 samples to clean with a predefined strategy, simulate manual data cleaning (by correcting their label if it was wrong), retrain the convoluational neural network in the pipeline, and measure its accuracy on the test set. We repeat this process for 10 iterations, and evaluate two different strategies for choosing the samples to clean: (*i*) we naively choose the samples at random, (*ii*) we choose the samples based on their estimated Shapley value [9], which denotes their utility for the classifier.

Results and discussion. The reduction in the mean runtime is depicted in Figure 3(b). The end-to-end runtime is decreased from 20.01 seconds for naive execution to 11.89 seconds with optimised execution. Note that roughly 10 seconds of this runtime are for repeated model training, which cannot be optimised but is not required for computing estimated shapley values. Without model retraining, our optimisations reduce the runtime by a factor of more than five from 9.67 seconds to 1.88 seconds. We attribute this to the fact that the optimised variant only needs to load the data from disk once, and, more importantly, only needs to featurise the train and test set once, which saves all but one execution of the preprocessing operations. The resulting featurised train and test data can then be used for all subsequent iterations. Figure 3(c) illustrates the results of the what-if analysis for the two sample selection strategies under comparison. The left plot shows that Shapley value-based selection is more beneficial than random selections, as the model reaches a high accuracy level faster, and its predictive performance is more stable. Additionally, as plotted on the right, Shapley value-based selection identifies a much higher fraction of the mislabeled samples in each iteration than random selection.

5 SUMMARY & NEXT STEPS

We discussed the problem of data-centric what-if analysis for natively written ML pipelines and showcased the runtime benefits provided by reusing trained models and intermediate results. In followup work, we will explore how to automatically rewrite user-defined pipelines to apply these optimisations. We intend to build this rewriting machinery on top of our recently introduced work for instrumenting the Python code of ML pipelines and extracting a "logical plan" of the pipeline operations [8, 12]. Furthermore, we intend to generalise the covered what-if scenarios, to base the optimisations on a cost model, and to experiment with parallelising the execution using different backends.

REFERENCES

[1] [n.d.]. Adult income dataset. https://archive.ics.uci.edu/ml/datasets/adult.

[2] [n.d.]. Amazon Reviews dataset. https://s3.amazonaws.com/amazon-reviews-pds/readme.html.

[3] [n.d.]. Data Centric AI Benchmark. https://www.datacentricai.cc/benchmark/.

[4] [n.d.]. Fashion MNIST dataset. https://github.com/zalandoresearch/fashion-mnist.

[5] Sumon Biswas and Hridesh Rajan. 2021. Fair preprocessing: towards understanding compositional fairness of data transformers in machine learning pipeline. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 981–993.

[6] Irene Chen, Fredrik D Johansson, and David Sontag. 2018. Why is my classifier discriminatory? *Advances in Neural Information Processing Systems* 31 (2018).

[7] Daniel Deutch, Zachary G Ives, Tova Milo, and Val Tannen. 2013. Caravan: Provisioning for What-If Analysis.. In *CIDR*.

[8] Stefan Grafberger, Paul Groth, Julia Stoyanovich, and Sebastian Schelter. 2022. Data distribution debugging in machine learning pipelines. *The VLDB Journal* (2022), 1–24.

[9] Bojan Karlaš, David Dao, Matteo Interlandi, Bo Li, Sebastian Schelter, Wentao Wu, and Ce Zhang. 2022. Data Debugging with Shapley Importance over End-to-End Machine Learning Pipelines. https://doi.org/10.48550/ARXIV.2204.11131

[10] Sanjay Krishnan, Jiannan Wang, Eugene Wu, Michael J Franklin, and Ken Goldberg. 2016. Activeclean: Interactive data cleaning for statistical modeling. *Proceedings of the VLDB Endowment* 9, 12 (2016), 948–959.

[11] Peng Li, Xi Rao, Jennifer Blase, Yue Zhang, Xu Chu, and Ce Zhang. 2019. Cleanml: A benchmark for joint data cleaning and machine learning [experiments and analysis]. *ICDE* (2019).

[12] Sebastian Schelter, Stefan Grafberger, Shubha Guha, Olivier Sprangers, Bojan Karlaš, and Ce Zhang. 2022. Screening Native ML Pipelines with "ArgusEyes". *CIDR* (2022).

[13] Sebastian Schelter, Tammo Rukat, and Felix Biessmann. 2021. JENGA-A Framework to Study the Impact of Data Errors on the Predictions of Machine Learning Models.. In *EDBT*. 529–534.

[14] Evan R Sparks, Shivaram Venkataraman, Tomer Kaftan, Michael J Franklin, and Benjamin Recht. 2017. Keystoneml: Optimizing pipelines for large-scale advanced analytics. In *2017 IEEE 33rd international conference on data engineering (ICDE)*. IEEE, 535–546.

[15] James Wexler, Mahima Pushkarna, Tolga Bolukbasi, Martin Wattenberg, Fernanda Viegas, and Jimbo Wilson. 2019. The What-If Tool: Interactive Probing of Machine Learning Models. *IEEE Transactions on Visualization and Computer Graphics* PP (08 2019), 1–1. https://doi.org/10.1109/TVCG.2019.2934619