MLWHATIF: What If You Could Stop Re-Implementing Your Machine Learning Pipeline Analyses Over and Over?

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

Shubha Guha
University
of Amsterdam
s.guha@uva.nl

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

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

ABSTRACT

Software systems that learn from data with machine learning (ML) are used in critical decision-making processes. Unfortunately, real-world experience shows that the pipelines for data preparation, feature encoding and model training in ML systems are often brittle with respect to their input data. As a consequence, data scientists have to run different kinds of *data centric what-if analyses* to evaluate the robustness and reliability of such pipelines, e.g., with respect to data errors or preprocessing techniques. These what-if analyses follow a common pattern: they take an existing ML pipeline, create a pipeline variant by introducing a small change, and execute this variant to see how the change impacts the pipeline's output score.

We recently proposed `mlwhatif`, a library that enables data scientists to declaratively specify what-if analyses for an ML pipeline, and to automatically generate, optimize and execute the required pipeline variants. We demonstrate how data scientists can leverage `mlwhatif` for a variety of pipelines and three different what-if analyses focusing on the robustness of a pipeline against data errors, the impact of data cleaning operations, and the impact of data preprocessing operations on fairness. In particular, we demonstrate step-by-step how `mlwhatif` generates and optimizes the required execution plans for the pipeline analyses. Our library is publicly available at https://github.com/stefan-grafberger/mlwhatif.

PVLDB Reference Format:
Stefan Grafberger, Shubha Guha, Paul Groth, and Sebastian Schelter. MLWHATIF: What If You Could Stop Re-Implementing Your Machine Learning Pipeline Analyses Over and Over?. PVLDB, 16(12): 4002 - 4005, 2023.
doi:10.14778/3611540.3611606

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/shubhaguha/mlwhatif-demo.

1 INTRODUCTION

Software systems that learn from data with machine learning (ML) are used in critical decision-making processes [14]. Unfortunately, real-world experience shows that the pipelines for data preparation, feature encoding and model training in ML systems are often brittle with respect to issues in the data they process [9, 10, 14].

Proceedings of the VLDB Endowment, Vol. 16, No. 12 ISSN 2150-8097.
doi:10.14778/3611540.3611606

Data centric what-if analysis with `mlwhatif`. Data scientists play a critical role in ensuring the robustness and fairness of these pipelines by performing *data-centric what-if analyses*. These analyses focus on understanding the sensitivity of the pipelines to small changes to their input data or pipeline operators [2]. Such analyses, for example, focus on (*i*) the robustness against data errors [13], asking *what-if the input data to a pipeline had certain errors like missing values or outliers?*, (*ii*) the impact of preprocessing operators on the pipeline's fairness [1], asking *what-if the pipeline filtered or featurized the training data differently?*, and (*iii*) the impact of data cleaning operations [7], asking *what-if the pipeline applied a particular error detection and cleaning technique?*

Performing what-if analyses on ML pipelines poses several technical challenges. Many analysis techniques are designed for single input datasets and are not easily integrated with existing pipeline code. Additionally, the repeated execution of pipeline variants incurs significant overhead, making it difficult for data scientists to iterate quickly during development.

We recently proposed `mlwhatif` [2] in response to these problems. `mlwhatif` *enables data scientists to declaratively specify what-if analyses for an ML pipeline, and to automatically generate, optimize and execute the required pipeline variants.* Our approach builds on dataflow representations for ML pipelines from previous work [3, 4, 11, 12].

Demonstration Details. We showcase `mlwhatif` in three scenarios, each highlighting a different what-if analysis, which `mlwhatif` can automatically apply to existing pipeline code. Attendees will be able to experiment with these analyses on pipelines from different domains, implemented using popular data science libraries like scikit-learn and pandas. We provide a web-based user interface for the attendees to experience `mlwhatif` from the perspective of a data scientist. In particular, they can experiment with what-if analyses focusing on the robustness of a pipeline against data errors, the impact of preprocessing operators on the pipeline's fairness, and the impact of data cleaning operations.

Our web-based user interface additionally allows attendees to explore the inner workings of `mlwhatif`: we show how `mlwhatif` extracts a dataflow plan from the original pipeline, and how it leverages so-called "pipeline patches" to create pipeline variants, according to the attendee's configuration of the what-if analyses.

Furthermore, we visualize `mlwhatif`'s' multi-query optimization process step-by-step, as it applies different optimization rules and merges all optimized pipeline variants into one joint execution plan.

We provide the fully working web-based user interface for our demonstration, along with all example pipelines and datasets, at https://github.com/shubhaguha/mlwhatif-demo.

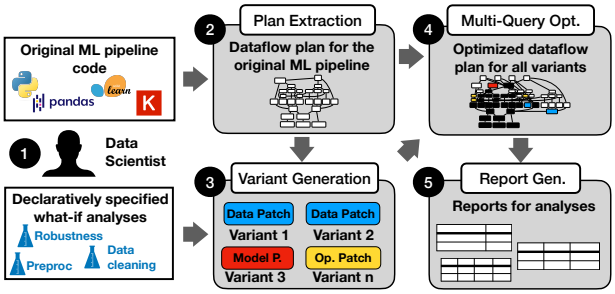

Figure 1: Overview of `mlwhatif`: The data scientist provides the code for an ML pipeline and declaratively specifies the what-if analyses to run on that pipeline ❶. Next, `mlwhatif` extracts a dataflow plan from the original pipeline via code instrumentation ❷. Based on the specified what-if analyses and the extracted plan, `mlwhatif` generates different pipeline variants with "pipeline patches" ❸, and merges the variants into a single joint dataflow plan. Next, `mlwhatif` applies various multi-query optimization rules to re-use shared intermediates (illustrated as black operators) between variants ❹. Finally, `mlwhatif` executes the optimized plan, and generates a report per analysis, which details the variants and their output scores ❺.

2 SYSTEM OVERVIEW

In the following, we provide a brief overview of `mlwhatif` and refer to the original publication [2] for further details.

Core ideas. Figure 1 provides an overview of how `mlwhatif` works internally. ❶ `mlwhatif` only requires the user to provide the source code of their pipeline and declaratively specified what-if analyses to run. ❷ Then, `mlwhatif` starts by extracting logical query plans, modeled as directed acyclic graphs (DAGs) of preprocessing operators, from ML pipelines that use popular libraries like pandas and scikit-learn. It supports code that combines relational operations on dataframes and estimator/transformer pipelines on matrix data for feature encoding [3, 4]. Additionally, `mlwhatif` extracts all function call arguments necessary for "replaying" a plan operator on different input data, and thus operates on a fully re-executable plan, which can be re-written and re-executed as needed.

With this plan as starting point, `mlwhatif` offers an interface to express what-if analyses. These what-if analyses follow a common pattern: they take an existing ML pipeline, create a pipeline variant by introducing a small change, and execute this pipeline variant to see how the change impacts the pipeline's output score. Each such analysis is written by an expert (and provided to other data scientists later). Data scientists using `mlwhatif` only need to configure these what-if analyses as they wish.

❸ The what-if analyses model their changes to the original pipeline plan via a declarative abstraction called *pipeline patches*. Based on a configuration from the user, what-if analyses generate patches to create the required pipeline variants. ❹ Given the resulting pipeline variants, `mlwhatif` then applies multi-query optimization to compute and execute a joint query plan for all pipeline variants. ❺ Finally, `mlwhatif` generates a detailed report with the output scores of different variants originating from the analyses for the user.

Modeling what-if analyses with "pipeline patches". Internally, `mlwhatif` generates different pipeline variants for what-if analyses by applying *pipeline patches* to the dataflow plan of the original ML pipeline. Such a patch defines how to change the plan of the original pipeline. `mlwhatif` currently supports three types of pipeline patches: *model patches*, *operator patches*, and *data patches*. Model patches denote that a pipeline variant should use a different model, operator patches specify the removal or replacement of a particular operator in the plan, and data patches specify that a particular operation should be applied to a column of an input data source. In contrast to the other patches, data patches are declarative as they only specify the semantics of the operation to apply to the input column, but no plan location to change.

Multi-query optimization on ML pipeline variants. `mlwhatif` optimizes the joint execution of all generated ML pipeline variants, via subsumption-based optimization rules. The goal of these rules is to increase the shared work between all variants. These optimizations focus on the patches for the original plan. Because all subexpressions until the first patch location are already shared between variants, `mlwhatif` only needs to consider rewrites to move the patches further up in the plan to potentially re-use more subexpressions. Besides common subexpression elimination, `mlwhatif` currently applies four optimization rules: projection push-up, filter addition push-up, filter removal push-up, and UDF split-reuse, and uses cost-based heuristics to decide when to apply them. The first three optimizations try to push-up projections and filters as high as possible, while UDF split-reuse aims to optimize the execution of expensive UDFs, which are repeatedly applied to large fractions of the data.

3 DEMONSTRATION DETAILS

We demonstrate `mlwhatif` with a web-based interface (illustrated in Figure 2), which allows attendees to configure three what-if analyses [1, 6–8, 13] , and automatically apply them to existing pipeline code from different domains (healthcare, product reviews, census data). Attendees will analyze the source code of the pipelines, and inspect reports about the results of the analyses. Furthermore, they can leverage `mlwhatif`'s runtime estimation to tailor the analysis configuration to their time budget. Additionally, the interface visualizes the internal multi-query optimization steps that `mlwhatif` applies to reduce the runtime of the analyses.

We make the web interface, datasets and pipeline code publicly available at https://github.com/shubhaguha/mlwhatif-demo.

In particular, we demonstrate each what-if analysis as follows:

(1) Attendees choose one of our provided ML pipelines, and we briefly introduce them to the ML pipeline code and the what-if analysis.
(2) Attendees use `mlwhatif`'s runtime estimation feature to estimate the runtime of different analysis configurations.
(3) Attendees select an analysis configuration and execute the pipeline analysis. Afterwards, they inspect and discuss the resulting reports with us and other attendees.

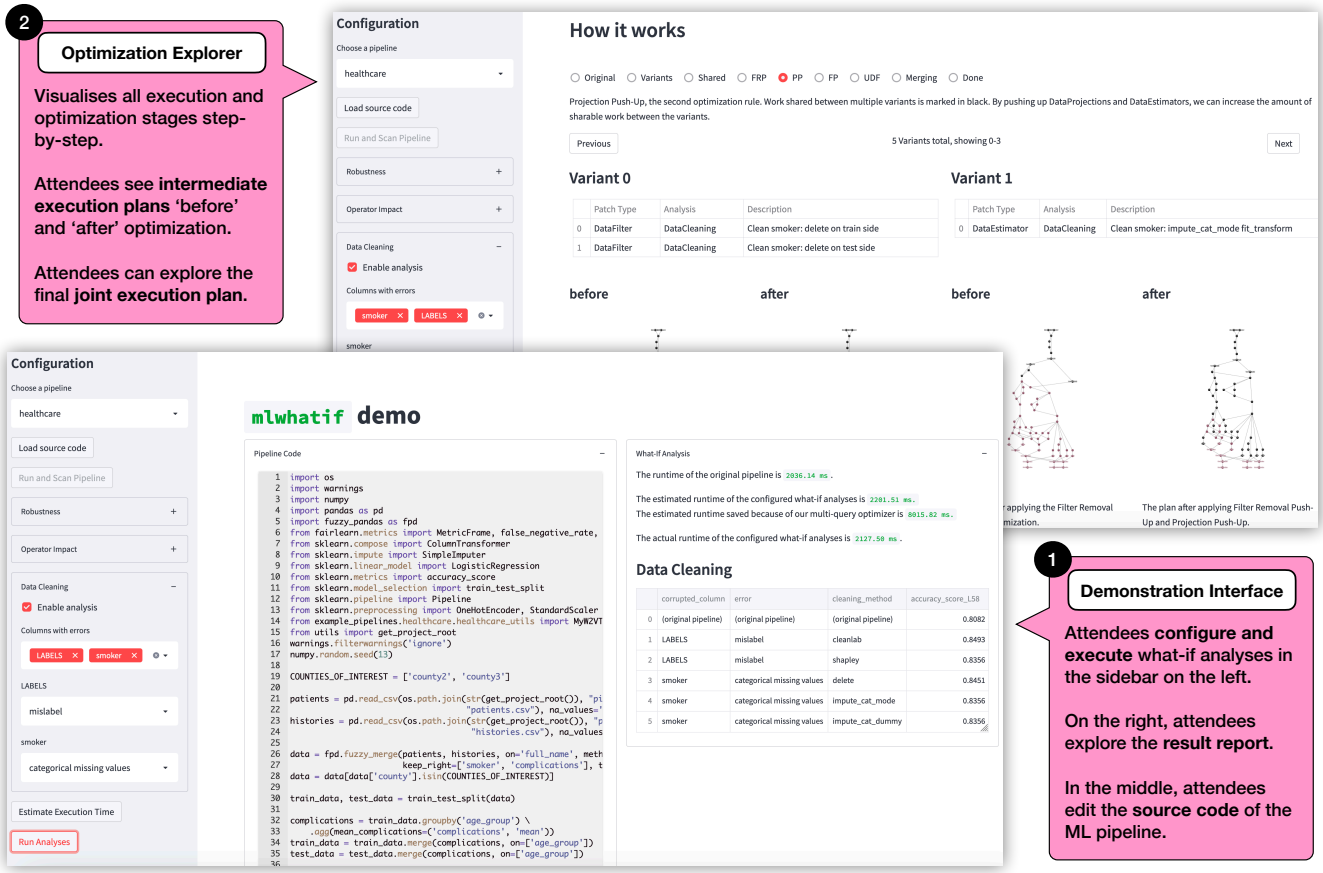

Figure 2: The web-based user interface for our demonstration. First, attendees experience `mlwhatif` from the perspective of a data scientist ❶. Afterwards, they explore how `mlwhatif` executes the configured what-if analyses, and follow the multi-query optimization process step-by-step ❷.

(4) To provide attendees with a deeper understanding of how `mlwhatif` generates, optimizes, and executes analyses internally, our web-based frontend provides step-by-step visualizations of the execution and optimization process. Attendees can freely explore those visualizations. First, they will observe how the dataflow plan is extracted from the original pipeline and how the what-if analyses use pipeline patches to create the pipeline variants for their configured what-if analysis.
(5) Then, they can follow how the generated pipeline variants get optimized and merged into one joint execution plan. Attendees will interactively explore the execution plans in different stages of our multi-query optimizer.

In detail, we demonstrate the following three scenarios:

What-If Analysis 1: Robustness against data errors. This analysis allows attendees to test the robustness of ML pipelines against data errors [13], asking *what-if the input data to a pipeline had certain errors like missing values or outliers?* Before deploying an ML pipeline in production, it is important to analyze how robust it is against potential data quality problems. For example, in healthcare an ML model might make predictions based on patient data and notes from a doctor. *What-if* a doctor makes a lot of typos in a stressful period? *What-if* patients enter their weight via a web-form and mix up the separator symbol for decimals, resulting in a sudden change of scale?

Interactivity. Our web-interface allows attendees to select which columns to corrupt and how to corrupt them (e.g., with missing values, categorical shifts, random scaling of numerical attributes, and introducing broken characters to text columns). Additionally, it allows attendees to specify different fractions of rows to corrupt to test the level of robustness of the pipelines. Optionally, attendees can introduce these corruptions not just to the test data, but also to the train data, to see if encountering similar errors at training time already helps the pipeline deal with particular errors better.

Analysis results. After running the what-if analysis, attendees are presented with a report, which describes the type and magnitude of the introduced data corruptions, as well as the corresponding pipeline output scores on the test set. Optionally, `mlwhatif` also reports the change to the output score if both the train and test set were corrupted. Based on this report, attendees explore how robust the respective pipeline is against different data errors at inference

time. These findings give them indications on ways to increase the robustness of their pipeline, e.g., by augmenting the training data with corrupted examples for a particular error type.

Optimization opportunities. Additionally, attendees can get a close look under the hood of `mlwhatif`'s multi-query optimizer. As the robustness analysis corrupts only the test side of a pipeline, our optimizer reuses large portions of work between the variants, e.g., all operations on the train side, including featurization and model training. For the test side, `mlwhatif` can push up the corruption operations below the featurization, and optimize the repeated application of expensive UDFs.

What-If Analysis 2: Impact of data cleaning operations. This analysis allows attendees to test the impact of data cleaning operations [6–8], asking *what-if the pipeline applied a particular error detection and cleaning technique?* Data quality problems like outliers are easy to miss but can significantly impact the performance of an ML model consuming the data.

There are many data cleaning techniques to choose from for particular error types. E.g., outliers can be detected with, e.g., standard deviations, percentiles, or isolation forests. Potential outliers can be imputed with methods like replacing them with the median, the mean, or the most frequent value. Unfortunately, it is often unclear in advance which data cleaning techniques are likely to help most. Thus, practitioners often have to experiment with many different methods [5]. `mlwhatif` automates this tedious and time consuming repetitive process for existing pipelines.

Interactivity. The what-if analysis for this only requires attendees to specify which data quality problem they expect in which parts of the data. The analysis will automatically try a pre-defined list of cleaning methods for each data quality problem, e.g., identifying and cleaning label errors with cleanlab [8] or kNN-Shapley [6].

Analysis results. `mlwhatif` generates a report on how particular cleaning methods affect the output scores of all variants of the pipeline code. For each variant, the report details how a data quality problem in a column was addressed, which cleaning method was applied, and how this impacts the output score.

Optimization opportunities. Attendees can again explore how much work is re-useable between different pipeline variants thanks to our optimization rules. The data cleaning analysis always patches both the train and test side of the pipeline, and requires a model re-training per variant. Therefore, the optimizer focuses on preprocessing optimizations here, such as projection push-up and filter push-up optimizations.

What-If Analysis 3: Impact of preprocessing on fairness. The third analysis allows attendees to measure the impact of preprocessing operators on the output scores of a pipeline, e.g., fairness metrics [1], asking *what-if* the pipeline filtered or featurized the training data differently? Sometimes, even inconspicuous preprocessing operations like removing rows with missing values via `dropna` in pandas introduce technical bias in ML pipelines [3]. Biswas et al. [1] proposed measuring the fairness impact of data transformers in ML pipelines by removing them or replacing them with a reference operation. `mlwhatif` can apply such analyses automatically to existing pipelines.

Interactivity. Attendees only need to specify which kind of preprocessing operators they want to analyze; they can select both filters and featurizers. Next, `mlwhatif` will generate a variant for each preprocessing operator to analyze, where the operator is dropped or replaced with a reference operation. The resulting pipeline scores are then compared to the original pipeline scores, to measure the impact of a particular preprocessing choice.

Analysis results. Attendees are presented with a report, describing each measured operator, its reference operation (e.g., the removal of the operator), and the resulting change in the pipeline output scores. The report allows attendees to discover particular operations that introduce technical bias, and to remove or change them accordingly in the original pipeline. One such example for a potentially problematic operation is the aforementioned `dropna` function.

Optimization opportunities. Repeatedly dropping or replacing preprocessing operations such as filters presents interesting optimization opportunities. Optimization rules like filter-removal push-up help `mlwhatif` to optimize across multiple different variants, which systematically remove or replace one operator at a time. Attendees will again be able to follow step-by-step how our multi-query optimizer leverages such optimization opportunities.

ACKNOWLEDGMENTS

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

[1] Sumon Biswas and Hridesh Rajan. 2021. Fair preprocessing: towards understanding compositional fairness of data transformers in machine learning pipeline. *ESEC/FSE* (2021).
[2] Stefan Grafberger, Paul Groth, and Sebastian Schelter. 2023. Automating and Optimizing Data-Centric What-If Analyses on Native Machine Learning Pipelines. *SIGMOD* (2023).
[3] Stefan Grafberger, Paul Groth, Julia Stoyanovich, and Sebastian Schelter. 2022. Data distribution debugging in machine learning pipelines. *VLDBJ* (2022).
[4] Stefan Grafberger, Shubha Guha, Julia Stoyanovich, and Sebastian Schelter. 2021. MLINSPECT: A Data Distribution Debugger for Machine Learning Pipelines. *SIGMOD* (2021).
[5] Shubha Guha, Falaah Arif Khan, Julia Stoyanovich, and Sebastian Schelter. 2023. Automated Data Cleaning Can Hurt Fairness in Machine Learning-based Decision Making. *ICDE* (2023).
[6] Ruoxi Jia, David Dao, Boxin Wang, Frances Ann Hubis, Nezihe Merve Gurel, Bo Li, Ce Zhang, Costas J Spanos, and Dawn Song. 2019. Efficient task-specific data valuation for nearest neighbor algorithms. *VLDB* (2019).
[7] Peng Li, Xi Rao, Jennifer Blase, Yue Zhang, Xu Chu, and Ce Zhang. 2019. Cleanml: A benchmark for joint data cleaning and machine learning. *ICDE* (2019).
[8] Curtis Northcutt, Lu Jiang, and Isaac Chuang. 2021. Confident learning: Estimating uncertainty in dataset labels. *JAIR* 70 (2021).
[9] Neoklis Polyzotis, Sudip Roy, Steven Euijong Whang, and Martin Zinkevich. 2018. Data lifecycle challenges in production machine learning: a survey. *SIGMOD Record* 47, 2 (2018).
[10] Sebastian Schelter, Felix Biessmann, Tim Januschowski, David Salinas, Stephan Seufert, and Gyuri Szarvas. 2018. On challenges in machine learning model management. *IEEE Data Engineering Bulletin* (2018).
[11] Sebastian Schelter, Stefan Grafberger, Shubha Guha, Bojan Karlaš, and Ce Zhang. 2023. Proactively Screening Machine Learning Pipelines with ArgusEyes. *SIGMOD* (2023).
[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. *EDBT* (2021).
[14] Julia Stoyanovich, Bill Howe, Serge Abiteboul, H.V. Jagadish, and Sebastian Schelter. 2022. Responsible Data Management. *Commun. ACM* (2022).