Demonstrating PIPE-X: Supporting Iterative Pipeline Development Through Explanations

Nadja Geisler
nadja.geisler@cs.tu-darmstadt.de
Technical University of Darmstadt
Darmstadt, Germany

Benjamin Hättasch
benjamin.haettasch@dfki.de
DFKI,
Technical University of Darmstadt
Darmstadt, Germany

Carsten Binnig
carsten.binnig@cs.tu-darmstadt.de
Technical University of Darmstadt,
DFKI
Darmstadt, Germany

Abstract

Building good data processing systems from an end-to-end perspective is as important as it is complex. While explainable AI (XAI) approaches help users to understand the behavior of trained models, choices in preprocessing and their effects are not yet part of explanations. We have recently proposed PIPE-X (**P**reprocessing **I**mpact and **P**ipeline **Ex**planations) to explain data preprocessing pipelines by calculating the impact of each step on the model's behavior. In this paper, we demonstrate how data scientists can leverage PIPE-X through an interactive graphical interface to gain insights into their pipeline and improve the end-to-end system accordingly. Users can provide their data, pipeline, and model to obtain the impacts for individual model outputs or an overview of the effect throughout the model. They can leverage various interactive outputs (graphical and numerical) to gain the best result for their specific use case.

Keywords

Machine Learning, Tabular Data, XAI, Data Cleaning, Preprocessing Pipeline, Impact

1 Introduction

The preprocessing of tabular training data is crucial for AI systems. It heavily influences the quality, robustness, cost, unintended bias and speed of the resulting model. This impact dictates that developers need to understand as much as possible about the effects of their pipeline and the interaction with other components of the system.

A Predictive Maintenance Example. Consider a newly hired full-stack data scientist who takes over a predictive maintenance system. Labeled sensor data passes through a pipeline of preprocessing steps before a classifier is trained to identify instances with a high likelihood of failure. The data scientist aims to reduce the number of costly misclassifications by analyzing all components of the system. Explainable AI (XAI) techniques can help to investigate the trained model. However, regarding the preprocessing pipeline, the data scientist must answer the question: "How does each preprocessing step change the model's behavior?"

XAI needs to cover data preprocessing. We propose expanding XAI research to cover data preprocessing as well. It is just as critical as model architectures and hyperparameters, skewing data distribution, handling data characteristics like missing values, and changing how features are considered during training (e.g., continuous/discrete values, scaling of value ranges). This leads to different models with different performances. To investigate the impact of data preprocessing in the context of AI systems, there

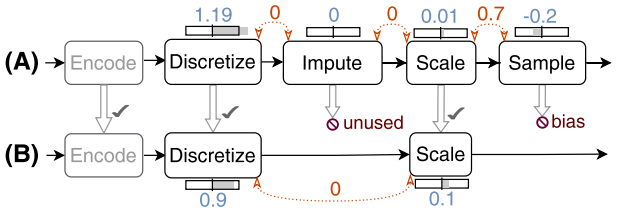

Figure 1: (A) Preprocessing pipeline with impacts ■ for each step and selected interaction indicators ■. Discretizer impact is high, imputer has no impact, scaler has little impact and sampler has negative impact. Scaler and Sampler have high interaction. (B) Redesigned pipeline after leveraging preprocessing explanations. Imputer is removed (no missing values, therefore no impact). Sampler is removed (class balancing introduces unintended shift). Scaler is retained despite low impact to benefit training time.

are currently no established tools. At the same time, deciding which preprocessing steps are well-suited is difficult. They might interact with the model architecture, hyperparameters, data and each other. A scaler might be crucial for achieving feasible training times for SVMs while other models are not affected by value ranges at all. With a neural network architecture, removing a scaler may have little to no effect if the values are already within a suitable range. In the case of other value ranges, the model might not converge.

Quantifying the importance of preprocessing steps. Suppose our data scientist knew how big each individual step's contribution to the resulting model was, and whether it brought the model closer to the desired performance. This would enable them to judge the benefit of each step, detect undesirable effects and interactions, as well as make different pipelines easier to compare. Using such a system on their pipeline as shown in Fig. 1 (A), our data scientist observes: (1) The impact of the missing value imputation is negligible (near zero). (2) The impact of the scaler on its own is minimal. However, (3) the interaction with the oversampling step is strong. (4) The scaler heavily influences the training time of the resulting model. (5) The impact vector of the oversampler is inverted compared to the rest of the pipeline (negative impact, see step five in Fig. 2 as an example). They draw the following conclusions: Following up on (1), missing value imputation is no longer necessary. It can be eliminated since there are no longer missing values in the data. As the data scientist eliminates the scaler based on (2), they discover from (3) that the oversampling suffers. It is kNN-based and, therefore, sensitive to scaling. Upon inspecting the oversampler following (5), they discover that its effect is not in line with the desired effect of the pipeline. It is intended to reduce the number of misclassifications, as the training data is very imbalanced, but

EDBT '26, Tampere (Finland)

10.48786/edbt.2026.64

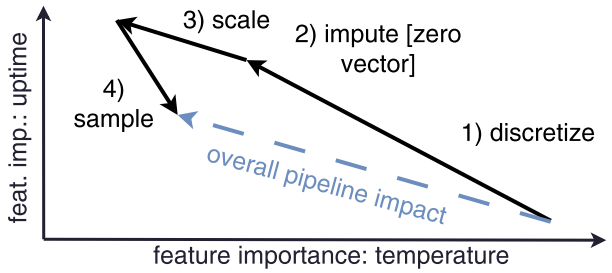

Figure 2: *Immediate impact* of each step in the pipeline from Figure 1 (A) in feature importance vector space. Steps sum up to the impact of the full pipeline. Note that the impact of step 2, the imputer which does nothing, is the zero vector.

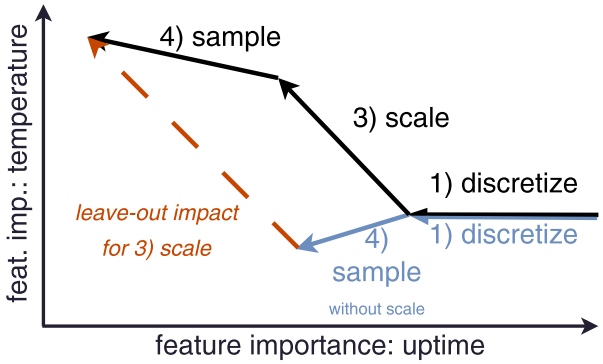

Figure 3: Full pipeline ■ vs. without scaler ■. *Leave-out impact* for scaler ■ is the difference between pipelines.

in this use case, false negatives are of much higher consequence than false positives, and class balancing skews the model toward these costly mistakes. Consequently, the data scientist removes the oversampler but retains the scaler, as the SVM architecture benefits greatly in terms of training time, as noted in (4). These changes result in the optimized pipeline visualized in Fig. 1 (B).

PIPE-X: Impact metrics as explanations for preprocessing steps. PIPE-X enables users to explain preprocessing pipelines by calculating the impact of each step on the model's behavior. PIPE-X computes two related impact metrics for each step: *leave-out impact* (What do we lose by leaving out this step?) and *immediate impact* (What do we gain by applying this step right now?).

Our Demonstration: Investigating Pipeline Impact and Interaction. In this demonstration, we introduce an easy-to-use browser-based application building on PIPE-X that allows practitioners to apply PIPE-X to their use case, to filter, visualize, and analyze outputs, incrementally improve their pipeline, and review these iterations before settling on a final configuration. Especially in the context of comparatively small models and fast training cycles, developers often intuitively apply strategies like omitting individual preprocessing steps and observing the impact on model performance. We provide them with a tool to automate, speed up and expand that process while also receiving more information on the actual effects of their pipeline.

This application improves on the manual process in several ways: Through automated, systematic variation of the pipeline, the process is quicker, users do not need to remember what has been tried, more variations are investigated, and models/datasets are reused wherever possible, improving runtime. Traditionally, accuracy or similar metrics are used to judge which model is better. By investigating the impact on model behavior, in the form of feature importance, the user gets more information on the effects of steps, in addition to model performance metrics and runtime, in one interface. Several types of graphical output, with options to filter, provide a clearer overview of the results, especially tailored for the properties of structured data. The computed metrics summarize information effectively even for longer pipelines and allow for automated flagging of potential issues the user might want to investigate (impact near zero, negative impact, high variation of impacts in different parts of the dataset, etc.). They also provide detailed information on the interaction of steps, which is often difficult to obtain in the manual process.

Video/Artifacts. A video demonstrating our system can be found at https://link.tuda.systems/PIPE-X_Demo. The source code of PIPE-X as well as the interactive application have been published at https://link.tuda.systems/PIPE-X.

2 Overview of PIPE-X

PIPE-X aims to provide additional insights into data preprocessing for data scientists, enabling them to utilize their expertise in building the optimal pipeline for each use case. In this section, we will give an overview of PIPE-X and introduce the different metrics [2]. Please refer to our supplementary material and other publications for more details, particularly on the computation of metrics: https://link.tuda.systems/PIPE-X.

System architecture. PIPE-X accepts any tabular dataset, a preprocessing pipeline, and a model architecture. In a setup phase, it automatically derives a number of systematic variations from the original pipeline. With each pipeline variation, a training dataset is processed and a model trained. In the explanation phase, PIPE-X retrieves the feature importances for each model, as a representation of the internal model behavior. According to the metrics described below, meaningful pairs of models are selected and their feature importances compared. The resulting impact vector, the differences in feature importance, is normalized to represent the contribution to the original preprocessing pipeline and its associated impact. PIPE-X provides the numerical impact for each step, giving an impression of how significant that step's impact on the model behavior was. We compute two different but related metrics that capture the contribution of each step to the pipeline and choose the pipelines to compare accordingly.

Immediate Impact. To answer the question, "What do we gain immediately when using this step at this time?" PIPE-X computes *immediate impact*. It represents the process of going through the pipeline one step at a time and observing the changes each step caused in the resulting model. Therefore, we compare the pipeline until right before the step and the pipeline until right after the step. The resulting number quantifies how much of the way of the entire pipeline this particular step takes the model. The *immediate impacts* of all steps of a pipeline sum up to one, which represents the difference the original pipeline makes. Each number's sign indicates whether the step aligns with the full pipeline's impact (+) or partly counteracts it (-).

Leave-out Impact. To answer the question "What do we lose when removing this step from the pipeline?" PIPE-X computes

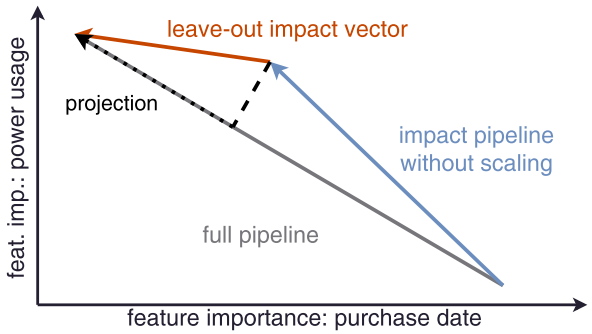

Figure 4: Vector calculation to compute impact. ■ is the full pipeline, ■ excludes scaler; ■ is the difference, ■■ dotted vector is projected onto the full pipeline, the length/direction is the resulting *leave-out impact*.

leave-out impact. It represents the process of removing one step from the pipeline and observing the changes caused in the end-to-end system. So for *leave-out impact*, we compare the original pipeline to the pipeline without the step in question. This process results in a number that quantifies how far away from the result of the full pipeline a pipeline without this step takes us. Figure 3 shows an example. The *leave-out impacts* of all steps do not sum to a fixed constant; however, a value of one still represents the impact of the full pipeline. A step with a *leave-out impact* of 0.5 would therefore have half as much impact as the full pipeline.

Step Interactions. When *Leave-out impacts* and *immediate impacts* for one pipeline diverge beyond scaling, this indicates an interaction of the step with another step. For example, consider the pipeline shown in Fig. 1 (A), where the sampler is a kNN-based oversampler and the model architecture is a decision tree. We would get a value near zero for the *immediate impact* of the scaler and a significantly larger *leave-out impact* for the same step. This occurs because in the computation of the *immediate impact* of the scaler, the sampler is not considered. In the computation of the *leave-out impact* of the scaler, however, the sampler is included. The sampler behaves very differently in conjunction with the scaler, therefore *leave-out impact* is high.

Metric Computation. We represent the behavior of a trained model as the importance of each feature to create a vector space. This can be achieved using any feature attribution method, such as LIME [4]. We can then compute the preprocessing pipeline as a vector that moves the model behavior from one point in that space to another (compare Figure 2), hopefully in a direction desired by the developer (often judged via accuracy, F_1 score, fairness metrics, resource usage or similar considerations). The difference between the vectors of two meaningfully chosen pipelines then represents the impact of that sole difference between the pipeline variations used to train those models. Our goal is a single number for each of these that is interpretable and retains as much information as possible. We want to consider the importance of each step in relation to the original pipeline, as the absolute value is not meaningful across different pipelines. Therefore, we calculate the vector for the original pipeline and project the impact vector onto it (see Fig. 4). That leaves us with a relative length ("How much of the way there does this step take us?") and a direction that is either in line or opposite to the pipeline. We then normalize the length, so the impact value is expressed as a percentage of the length of the pipeline vector, and the sign indicates the direction; e.g., *1* would cover precisely the length of the pipeline, a negative value would signify counteracting its effect.

3 Designing PIPE-X

Previous work on the transparency of preprocessing has been limited by the types of preprocessing steps supported (only one type of transformation) or the complexity of the pipeline (one step). Additionally, the effects of the preprocessing on the model are not reflected at all or in limited ways, as approaches either only consider changes on the data level, disregarding interactions with the model [1] entirely, or only considering changes in the model output, disregarding internal model behavior [3] and how outputs were produced (a key area for XAI research). As a further challenge, the effects of preprocessing steps may substantially differ across areas in the data (depending on the completeness and correctness of the samples, but also on the values for input and target themselves). These effects may be obscured when examining the overall model, hindering developers from optimizing the pipeline for subgroups, even though this may significantly contribute to the quality or fairness of the resulting system. Therefore, our approach is local-first and allows for the analysis of the pipeline for a given instance and its surroundings. We developed PIPE-X to provide impact metrics for each step of a data preprocessing pipeline that reflect the interaction of steps in their impacts. Additionally, PIPE-X considers the complexity of the full system, including behavior beyond model output. It works on any given dataset, pipeline, and model, as each AI system has different requirements and supports a diverse group of preprocessing steps, including parts of data wrangling, data cleaning, feature engineering, dimensionality reduction, and more. PIPE-X supports long, complex pipelines, including conditional steps and nesting, is independent of model architecture as a post-hoc explainer, provides local explanations, as well as regional or global ones with explicit confidence and supports users in an interactive process, valuing exploration with open results. PIPE-X is the only system of this kind.

4 Demonstration

This demonstration focuses on our browser-based application, which we designed to be useful quickly and without specific previous knowledge. The interface supports an iterative process where the user retains full control but has access to more insights than in the typical development of a preprocessing pipeline.

Usage Scenarios. The interface offers a low threshold to experimentation. Users can choose a dataset, pipeline, and model. We provide examples for each. The pipeline can be edited directly in the interface in code, aided by a visualization. Users can either explain an individual data sample or get averaged results over multiple samples, and see metrics for each step of the pipeline, joint with runtime measurements and model performance scores. Additional visualizations using dimensionality reduction illustrate the interplay between steps. Particularly, we show the impact vectors, giving users more information on the length and angle between it and the full pipeline.

The provided examples are oriented along several use cases: A suboptimal pipeline contains unnecessary steps and needs pruning. A new and old version of a dataset demonstrate data drift that necessitates changes in the pipeline. A minimal pipeline benefits from expanding to serve the model optimally. Finally, a runtime optimization task is heavily influenced by the chosen implementations of the steps in the pipeline, and the model

architecture. Participants may also create their own scenarios, comprising data, pipeline, and model architecture, and receive results within seconds, allowing them to interactively modify the pipeline from within the application.

Runtime Measurements and Scalability. In order to make sure this interface is practical, we evaluated runtime on 43 different pipelines with up to 13 steps on datasets with up to 41.000 samples. We provide 7 different model architectures in the demonstration: A Multilayer Perceptron/Neural Network (NN), a Logistic Regression Classifier (LR), a Decision Tree (DT), Gradient-boosted Trees (GBT), a K-Nearest-Neighbor Classifier (KNN), a Support Vector Machine (SVC), and a Histogram-based Gradient Boosting Classification Tree (HGB). On a consumer notebook with a quad-core CPU, explaining one sample takes a second or less for all these scenarios. When explaining 10, 100, or 500 samples to receive an aggregated global explanation, model training and dataset preprocessing, the time intensive steps, do not need to be repeated, as they are retained between explanations. The interface also connects with the computation backend via network, so even in the demonstration, computation can be run in parallel on a server for more complex use cases.

Showcases. In the context of this demo, participants may also explore two prepared showcases where we introduced artificial noise in the dataset or include a step dropping informational content in the pipeline. Participants can modify the probability of these factors to observe the changes in impact metrics aggregated over a large number of samples. This allows them to explore the expressiveness of the metrics and visualizations.

Interface. PIPE-X focuses on a human-in-the-loop approach, aiming to enable change through understanding rather than fully automatic optimization of a single metric. This is important because of the many optimization goals a user may have, the importances of which vary with each use case. The graphical interface, therefore, facilitates iterative improvement through a pipeline editor, a history feature that allows users to navigate their previous pipeline changes, and interactive visualizations. Users can choose between aggregating results across a number of chosen or randomly selected samples to explain average impacts, or providing a detailed explanation of one specific model output. The interface is shown in Figure 5. The application guides users to select a dataset, pipeline, model, and sample(s). The application will display the dataset and pipeline information, making it easy to verify inputs. After computation, the application will provide an overview diagram of all steps and their impacts in both a diagram and a table. This includes indicators of abnormalities the user might want to investigate: Negligible or negative impacts, high runtimes, drops in accuracy or high interaction among them. Depending on the specified use case, various textual, numerical, and visual displays of insights follow. Help texts serve as interpretation help or reminder. A run configuration can be modified easily for the next iteration, and an integrated editor facilitates changes to the pipeline. The history feature enables users to revisit previous runs. Users can select from various run configurations based on their current use case. An aggregated run provides an initial impression of the system across a number of (optionally random) samples. In addition to the overview diagram, users can choose which steps to investigate more closely through average alignment (the distribution of positive vs. negative impacts for a step) and average consistency (the degree to which impacts vary between samples for a step). A localized run,

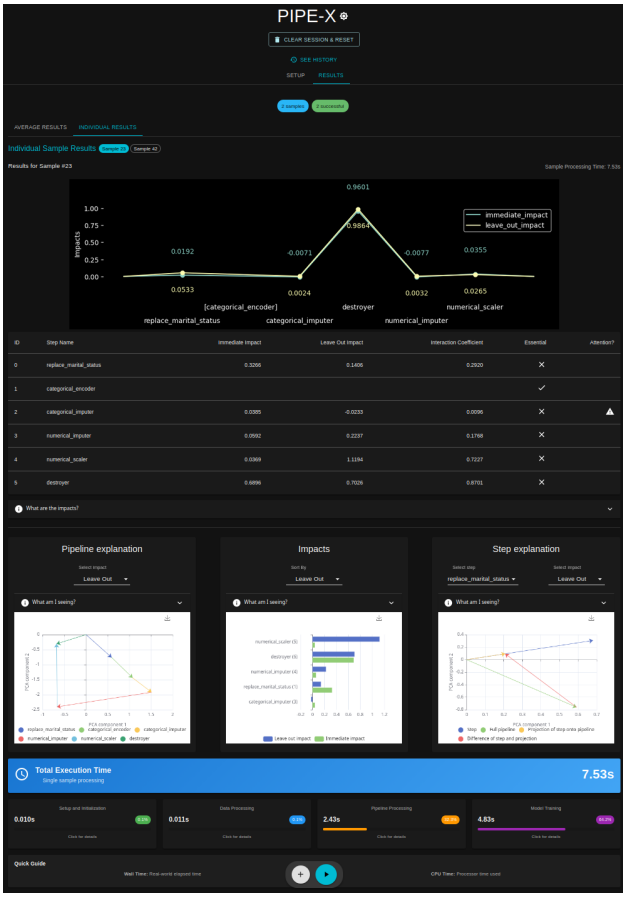

Figure 5: Excerpt from the output of a run as it is represented in the application interface.

which explains the model output for a single data sample, enables users to delve deeper into analyzing and identifying specific issues. Additionally, they can choose which metric to visualize in vector space, as seen in Figure 2, (via dimensionality reduction if needed) or what step to leave out and compare the vectors visualizing the interaction with all other steps, as seen in Figure 3.

Acknowledgments

This research and development project is/was partially funded by the German Federal Ministry of Education and Research (BMBF) within the "The Future of Value Creation – Research on Production, Services and Work" program (funding number 02L19C150). The authors are responsible for the content of this publication.

References

[1] Tamraparni Dasu and Ji Meng Loh. 2012. Statistical distortion: consequences of data cleaning. *Proc. VLDB Endow.* 5, 11 (jul 2012), 1674–1683. doi:10.14778/2350229.2350279

[2] Nadja Geisler and Carsten Binnig. 2024. Towards Extending XAI for Full Data Science Pipelines. In *Proceedings of the 2024 Workshop on Human-In-the-Loop Data Analytics* (Santiago, AA, Chile) *(HILDA 24)*. Association for Computing Machinery, New York, NY, USA, 1–7. doi:10.1145/3665939.3665967

[3] Carlos Vladimiro Gonzalez Zelaya. 2019. Towards Explaining the Effects of Data Preprocessing on Machine Learning. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. Institute of Electrical and Electronics Engineers (IEEE), Macao, China, 2086–2090. doi:10.1109/ICDE.2019.00245

[4] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. "Why Should I Trust You?": Explaining the Predictions of Any Classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (San Francisco, California, USA) *(KDD '16)*. Association for Computing Machinery, New York, NY, USA, 1135–1144. doi:10.1145/2939672.2939778