Proactively Screening Machine Learning Pipelines with ArgusEyes

Sebastian Schelter[†] Stefan Grafberger[†] Shubha Guha[†] Bojan Karlaš[††] Ce Zhang[*]

[†]University of Amsterdam [††]Harvard University [*]ETH Zürich
{s.schelter,s.grafberger,s.guha}@uva.nl bojan_karlas@hms.harvard.edu ce.zhang@inf.ethz.ch

ABSTRACT

Software systems that learn from data with machine learning (ML) are ubiquitous. ML pipelines in these applications often suffer from a variety of data-related issues, such as data leakage, label errors or fairness violations, which require reasoning about complex dependencies between their inputs and outputs. These issues are usually only detected in hindsight after deployment, after they caused harm in production. We demonstrate ArgusEyes, a system which enables data scientists to *proactively screen their ML pipelines for data-related issues as part of continuous integration.* ArgusEyes instruments, executes and screens ML pipelines for declaratively specified pipeline issues, and analyzes data artifacts and their provenance to catch potential problems early before deployment to production. We demonstrate our system for three scenarios: detecting mislabeled images in a computer vision pipeline, spotting data leakage in a price prediction pipeline, and addressing fairness violations in a credit scoring pipeline.

ACM Reference Format:
Sebastian Schelter, Stefan Grafberger, Shubha Guha, Bojan Karlaš, Ce Zhang. 2023. Proactively Screening Machine Learning Pipelines with ArgusEyes. In *Companion of the 2023 International Conference on Management of Data (SIGMOD-Companion '23), June 18–23, 2023, Seattle, WA, USA.*

1 INTRODUCTION

Software systems that learn from data with machine learning (ML) are ubiquitous. The behavior of such applications very much depends on their input data, and experience shows that it is difficult to ensure that all data-centric operations are implemented correctly [1, 7–10, 13].

Data-related issues in ML pipelines. ML pipelines in real-world applications often suffer from a variety of data-related issues. Examples for such hard-to-detect issues include *data leakage* [7] from train to test data and *label errors* [8] in training samples, which negatively impact the accuracy of a pipeline, as well as *fairness violations* [2] which result in biased predictions for particular demographic groups. In practice, such pipeline issues are usually only detected in hindsight after deployment, after they already caused harm in production (such as faulty predictions), and need to be urgently addressed in an ad-hoc fashion, requiring a lot of time, expertise and extra code. Unfortunately, existing ML management platforms like mlflow [14] lack support for detecting and debugging such issues, because they do not support record-level provenance and do not understand the semantics of operations in ML pipelines.

Proactive pipeline screening with ArgusEyes. We postulate that data scientists require system support to be able to proactively screen their ML pipelines in an automated fashion. This enables them to catch data-related issues early before deployment, and reduce the incurred manual effort for fixing them. For that, we demonstrate the recently proposed ArgusEyes[1] system [11]. As we detail in Section 2, our system enables data scientists to *proactively screen their ML pipelines for data-related issues as part of continuous integration.* They can declaratively specify a variety of pipeline issues that they are concerned about. Subsequently, ArgusEyes instruments, executes and screens the pipeline for the configured pipeline issues, to catch potential problems early before deployment to production. In contrast to existing ML platforms, ArgusEyes tracks record-level provenance [5] and understands the semantics of individual pipeline operations, which enables it to reason about complex dependencies between the inputs and outputs of ML pipelines. ArgusEyes leverages mlinspect [4] to instrument ML pipelines for supervised learning, which are natively written in Python and contain known operations from popular data science libraries such as pandas, scikit-learn or keras. Note that mlinspect focuses on local, operator-centric data debugging, while ArgusEyes enables holistic pipeline debugging, to detect issues which require reasoning about complex interactions between the inputs and the model of an ML pipeline (e.g., detecting label errors).

Demonstration details. We demonstrate ArgusEyes in three different scenarios, with pipelines created from real-world code from the keras [3], OpenML [12] and mlflow [14] projects, consuming both tabular and image data. Each scenario focuses on a particular issue to detect. During the demonstration, we will explain the pipeline, show how to detect the underlying issue with ArgusEyes, and walk attendees through a retrospective analysis of the pipeline artifacts captured by ArgusEyes to fix the pipeline issue (Section 3). In particular, we demonstrate how to detect mislabeled images in a computer vision pipeline, how to spot data leakage in a price prediction pipeline and how to detect fairness violations in a credit scoring pipeline. We provide a fully working implementation of the pipelines and screening, integrated with Github workflows at https://github.com/amsterdata/arguseyes-demo.

[1] https://github.com/amsterdata/arguseyes

SIGMOD-Companion '23, June 18–23, 2023, Seattle, WA, USA

ACM ISBN 978-1-4503-9507-6/23/06.
https://doi.org/10.1145/3555041.3589682

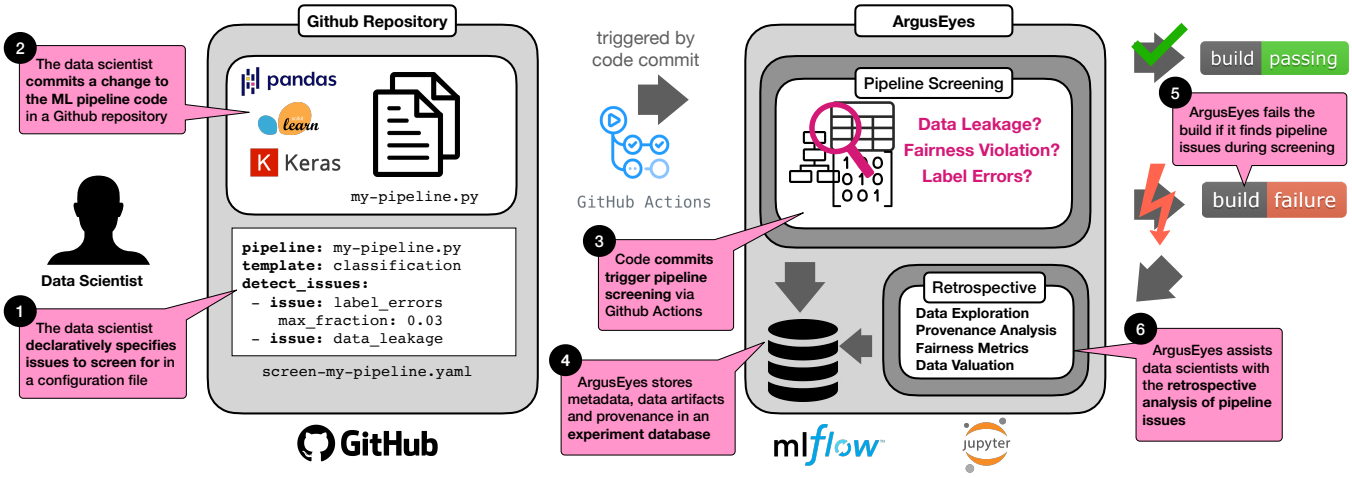

Figure 1: Proactive pipeline screening with ArgusEyes as part of a continuous integration workflow on Github.

2 SYSTEM OVERVIEW

ArgusEyes [11] is implemented in Python and builds upon our previous work *mlinspect* [4], a library to instrument ML pipelines. ArgusEyes handles ML pipelines for supervised learning, implemented as Python scripts leveraging known operations from popular data science libraries such as pandas, sklearn, or keras. ArgusEyes models ML pipelines as a dataflow computation from relational inputs to matrix outputs for the features, labels and predictions of the model trained by the pipeline, based on predefined pipeline "templates" for classification and regression tasks [4, 11]. It executes and instruments ML pipelines, and captures input data and intermediates, as well as their corresponding record-level provenance, and stores these artifacts in the experiment database *mlflow* [14]. Its core novelty over existing ML management platforms is to analyze record-level "why-provenance" [5] (e.g., the information which input records were used to compute a particular output) to detect a wide variety of issues in ML pipelines[2], which arise from dependencies between pipeline artifacts such as relational inputs, computed feature matrices, labels and model predictions.

Pipeline screening as part of continuous integration. The main use case of ArgusEyes is to be run as part of continuous integration, in order to alert data scientists and data engineers early (before the deployment of a pipeline change to production) in case of detected errors. Subsequently, it allows them to analyze the root cause of detected issues retrospectively based on captured intermediates and metadata from the pipeline. Figure 1 gives an overview over the integration of ArgusEyes into a data science development workflow: ❶ We assume that the data science team already leverages a Github repository for their ML pipeline code. They can integrate ArgusEyes into their build process via Github workflows[3] and declaratively specify their pipeline and issues to screen for in a `.yaml` configuration file (without having to change the pipeline code). ❷ Due to the integration with Github workflows, code commits to the repository will automatically trigger ArgusEyes; ❸ Next, ArgusEyes runs within a Github action: it instruments and executes the specified ML pipeline, screens it for the configured issues, and ❹ stores metadata, data artifacts and provenance information in the experiment database mlflow [14]. If ArgusEyes encounters issues during screening ❺, it will automatically fail the Github build. Subsequently, ArgusEyes assists the data scientists with a retrospective analysis of the pipeline run (and stored artifacts) within a Jupyter notebook ❻ with debugging functionality tailored to the detectable issues.

3 DEMONSTRATION DETAILS

Approach. In order to demonstrate ArgusEyes, we design three scenarios, each containing a particular ML pipeline based on existing code and focusing on a particular issue to detect. We setup a Github repository at https://github.com/amsterdata/arguseyes-demo, which contains the source code and data of the pipelines for the scenarios, and integrates the screening of these pipelines with ArgusEyes via a Github workflow. We demonstrate each scenario as follows:

(1) We briefly introduce the ML pipeline for the given scenario and the pipeline issue that occurs.
(2) Next, we run ArgusEyes to screen the pipeline and showcase that it detects the given issue.
(3) We demonstrate how to conduct a retrospective analysis of the pipeline run with ArgusEyes and Jupyter to determine the root cause of issue. For that, ArgusEyes allows us to interactively explore the execution plan and intermediate data from the pipeline (as illustrated in Figure 3). Additionally, it provides custom code for analyzing particular issues like label errors, data leakage or fairness violations.
(4) Finally, we showcase how to change the pipeline code (based on insights from the retrospective analysis) to fix the detected issue in the pipeline.

[2]https://github.com/amsterdata/arguseyes/tree/main/arguseyes/issues
[3]https://docs.github.com/en/actions/using-workflows/about-workflows

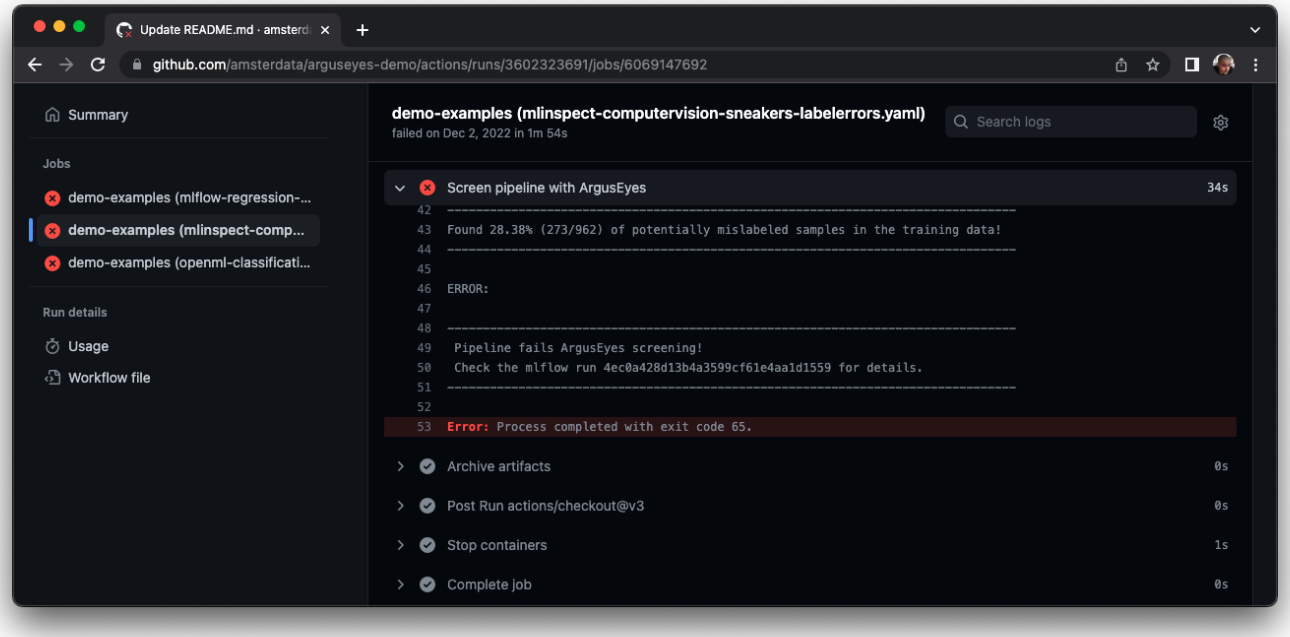

Figure 2: Github integration – Pipeline screening during the build process executed with a github workflow. ArgusEyes fails the build as it detects a high amount of label errors in the data.

In detail, we demonstrate the following three scenarios:

Scenario 1: Mislabeled images in a computer vision pipeline. Our first scenario evolves around a computer vision pipeline based on code from keras [3]. The pipeline trains a convolutional neural network to identify fashion products, in particular to distinguish between images of sneakers and ankle boots from the *FashionMNIST* dataset. The pipeline consumes three input tables, filters the data based on the product category, and applies common image-specific preprocessing operations such as normalization of the pixel values.

Issue to detect. The training images for the pipeline contain a large amount of label errors [8], e.g., images which are labeled with the wrong product category. These mislabeled training samples have a strong negative impact on the accuracy of the learned neural network. We showcase how ArgusEyes identifies such label errors by computing Shapley values [6] for the featurized training samples.

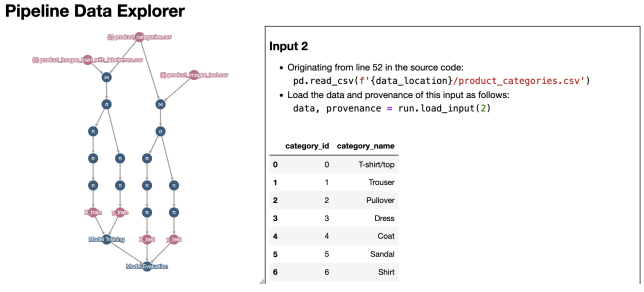

Figure 3: Interactive exploration of the intermediate data from a pipeline for retrospective analysis.

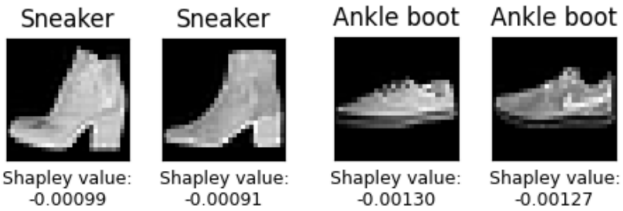

Figure 4: Label errors – ArgusEyes automatically detects mislabeled product images in a computer vision pipeline (left: ankle boots mislabeled as sneakers, right: sneakers mislabeled as ankleboots).

Retrospective analysis and fix. Next, we detail how to retrospectively analyze the pipeline and identify the mislabeled images. ArgusEyes allows attendees to retrieve a copy of the input tuples stored in mlflow, annotated with the corresponding Shapley values, which denote their estimated "value" for the final classifier. By selecting and plotting the input images with negative Shapley values (which indeed have the wrong category information, as shown in Figure 4), attendees can confirm the label errors and identify the images to relabel. Finally, we show that rerunning the pipeline with cleaned inputs results in an improvement in the accuracy of the neural network.

Scenario 2: Data leakage in a price prediction pipeline. Our second scenario concerns a pipeline that trains a regression model for predicting the price of taxi rides in New York, based on code from the mlflow [14] project. The pipeline consumes a single input dataset about the times and locations of taxi rides, filters it and computes several new features. Next, it splits the data into train and testset based on the drop-off times in the data, and trains and evaluates a regression model to predict the price of rides.

Issue to detect. The pipeline suffers from data leakage [7]: several training samples are accidentally included in the test set as well, leading to an unrealistic, overly optimistic accuracy score of the classifier. This is due to a faulty predicate used for conducting the train/test split, which has the effect that the samples from the last day of the train set are also included in the test set.

```
run = PipelineRun(run_id='bc07e7b4c8c54ee694078030860649b2')
retrospective = DataLeakageRetrospective(run)

leaked_data = retrospective.compute_leaked_tuples()
leaked_data
```

	tpep_pickup_datetime	tpep_dropoff_datetime	trip_distance	fare_amount	pickup_zip	dropoff_zip
37	2016-02-15 01:45:11	2016-02-15 01:48:40	0.60	4.5	10153	10065
57	2016-02-15 08:51:02	2016-02-15 09:06:27	5.50	17.0	11371	11379
73	2016-02-15 23:03:19	2016-02-15 23:38:15	11.40	35.0	11371	10011
77	2016-02-15 16:41:54	2016-02-15 17:38:20	18.43	52.0	11422	10011
144	2016-02-15 12:44:51	2016-02-15 13:07:39	3.48	16.5	10011	10022
...	...	...	...	...	...	...

Figure 5: Data leakage – ArgusEyes allows attendees to identify and analyze input tuples of a pipeline that were accidentally leaked from the train to the test set.

Retrospective analysis and fix. ArgusEyes allows data scientists to retrospectively analyze the run of the price prediction pipeline. We demonstrate how to leverage ArgusEyes to compute the set of leaked input tuples, and interactively analyze them (Figure 5. This exploratory analysis shows that they all originate from the same drop-off day of the ride. Based on that, attendees can identify and fix the faulty predicate in the pipeline code, and verify with ArgusEyes that the leakage problem goes away after that.

Scenario 3: Fairness violations in a credit scoring pipeline. Our third scenario evolves around a credit scoring pipeline, based on code from the OpenML project [12]. The pipeline operates on demographic information about income, stored in seven different tables, joins and filters this data based on a predefined set of employment types, and trains a decision tree model to predict whether a person has a high or low income (as a proxy for creditworthiness).

Issue to detect. We demonstrate how to leverage ArgusEyes to detect fairness violations with respect to different groups in the data (e.g., male compared to non-male persons). ArgusEyes allows attendees to define such groups, and specify a fairness metric and a threshold for the maximum difference in the metric. ArgusEyes will automatically compute the fairness metric for the defined groups and fail the build if the difference in the metric is larger than the configured threshold. In our demonstration, we show that the pipeline produces unfair predictions for female persons with respect to the "equal opportunity" metric, which measures the difference in recall between the demographic groups.

```
retrospective = FairnessRetrospective(run)
retrospective.show_confusion_matrix('sex', 'male')
```

Confusion matrix for the privileged group (sex=male)

	positive prediction	negative prediction	
positive label	TP=179	FN=206	385
negative label	FP=166	TN=813	979
	345	1019	

Confusion matrix for the disadvantaged group (sex!=male)

	positive prediction	negative prediction	
positive label	TP=141	FN=151	292
negative label	FP=121	TN=421	542
	262	572	

Figure 6: Fairness analysis – ArgusEyes allows attendees to inspect the confusion matrices for demographic groups in the input data, based on the pipeline's predictions.

Retrospective analysis and fix. We showcase how attendees can leverage ArgusEyes to retrieve fairness statistics about the pipeline run, compute detailed confusion matrices for the predictions per group (as shown in Figure 6), and to compute and plot different fairness metrics. Furthermore, we show that the fairness violation can be fixed by increasing the training data, in particular by including samples from a more diverse set of employment types.

Interactivity. Our demonstration will be executed with a code editor for the ML pipelines, a shell to run ArgusEyes and an interactive notebook for the retrospective analysis. Attendees can interact with our demonstration in several ways: (*i*) they can suggest changes to the pipeline code to introduce or fix certain pipeline issues and rerun ArgusEyes to screen for these issues; (*ii*) they can suggest and explore changes to the screening configuration in ArgusEyes, e.g., to modify the tolerable amount of label errors or to define additional groups for fairness analysis, and (*iii*) during the retrospective analysis of pipeline runs in a notebook, attendees can suggest and try different analysis steps. Moreover, attendees with their own laptop can clone our github repository and run and change the demonstration scenarios themselves.

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

[1] Anaconda.com. 2020. The State of Data Science 2020. https://www.anaconda.com/state-of-data-science-2020.
[2] Irene Chen et al. 2018. Why is my classifier discriminatory? *NeurIPS* (2018).
[3] François Chollet et al. 2015. Keras. https://keras.io.
[4] Stefan Grafberger et al. 2022. Data distribution debugging in machine learning pipelines. *VLDB Journal* (2022).
[5] Todd J Green et al. 2007. Provenance semirings. *PODS* (2007).
[6] Ruoxi Jia et al. 2019. Efficient task-specific data valuation for nearest neighbor algorithms. *VLDB* (2019).
[7] Sayash Kapoor et al. 2022. Leakage and the Reproducibility Crisis in ML-based Science. *arXiv preprint arXiv:2207.07048* (2022).
[8] Curtis G Northcutt et al. 2021. Pervasive label errors in test sets destabilize machine learning benchmarks. *NeurIPS* (2021).
[9] Neoklis Polyzotis et al. 2017. Data management challenges in production machine learning. *SIGMOD* (2017).
[10] Sebastian Schelter et al. 2015. On challenges in machine learning model management. *IEEE Data Engineering Bullettin* (2015).
[11] Sebastian Schelter et al. 2022. Screening Native ML Pipelines with "ArgusEyes". *CIDR* (2022).
[12] Joaquin Vanschoren et al. 2014. OpenML: networked science in machine learning. *KDD* (2014).
[13] Doris Xin et al. 2021. Production machine learning pipelines: Empirical analysis and optimization opportunities. *SIGMOD* (2021).
[14] Matei Zaharia et al. 2018. Accelerating the machine learning lifecycle with MLflow. *IEEE Data Engineering Bullettin* 41, 4 (2018), 39–45.