Plutus: Understanding Data Distribution Tailoring for Machine Learning

Jiwon Chang
jchang38@ur.rochester.edu
University of Rochester
Rochester, USA

Christina Dionysio
dionysio@tu-berlin.de
Technsiche Universität Berlin
Berlin, Germany

Fatemeh Nargesian
fnargesian@rochester.edu
University of Rochester
Rochester, USA

Matthias Boehm
mb7@dams.tu-berlin.de
Technsiche Universität Berlin
Berlin, Germany

ABSTRACT

Existing data debugging tools allow users to trace model performance problems all the way to the data by efficiently identifying slices (conjunctions of features and values) for which a trained model performs significantly worse than the entire dataset. To ensure accurate and fair models, one solution is to acquire enough data for these slices. In addition to crowdsourcing, recent data acquisition techniques design cost-effective algorithms to obtain such data from a union of external sources such as data lakes and data markets. We demonstrate Plutus, a tool for human-in-the-loop and model-aware data acquisition pipeline, on SystemDS, as an open source ML system for the end-to-end data science lifecycle. In Plutus, a user can efficiently identify problematic slices, connect to external data sources, and acquire the right amount of data for these slices in a cost-effective manner.

KEYWORDS

data acquisition; model debugging; distribution tailoring

ACM Reference Format:
Jiwon Chang, Christina Dionysio, Fatemeh Nargesian, and Matthias Boehm. 2024. Plutus: Understanding Data Distribution Tailoring for Machine Learning. In *Companion of the 2024 International Conference on Management of Data (SIGMOD-Companion '24), June 9–15, 2024, Santiago, AA, Chile.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3626246.3654745

1 INTRODUCTION

In machine learning, a common assumption is that the training data accurately represents the data encountered in production. However, in practice, this assumption is hard to verify and, more importantly, hard to satisfy. The reason is three-fold. First, it is often impossible to obtain the training data by randomly sampling from the distribution of the "full" production data. For instance, the data collected by a survey is highly skewed toward those people who responded. Second, even when random sampling is possible the underlying distribution may not be known in advance. Third, even in scenarios where representative samples can be obtained for training, to ensure model performance, we may need to train with data in which some parts of data are intentionally over- or under-represented. This distribution mismatch between training data and production data has led to numerous severe societal issues [20].

Model Debugging: The goal of ML model debugging is to monitor, analyze, and track the validity of train/validation data and model characteristics. The issues to consider include data errors, lack of model generalization (e.g., due to overfitting or test data leakage), as well as model fairness [19]. In particular, existing work [6, 22] aim to find the top-K data slices, conjunctions of features and values such as (`gender='female' and degree='PhD'`), where a model performs significantly worse than for the entire dataset. Finding such problematic slices is useful for understanding and debugging the distribution of a training dataset and subsequent model bias [6, 22].

Data Acquisition: For model improvement, existing data acquisition techniques aim to obtain new data from other data sources and enrich the training data in a way that it satisfies some desired distribution requirements [4, 15, 18]. While acquiring data for slices that have insufficient representation may reduce errors for those slices, it may also influence the model errors on other slices [23], and thus, the overall loss. This issue arises when slices are correlated and do not benefit the model equally. Thus, naïvely acquiring equal amounts of data per slice is not optimal. This imbalance is especially important, because data acquisition is often limited by budget: acquisition from crowdsourcing, data markets, and data lakes is often associated with monetary or computation costs.

Fairness: Another challenge is the fairness-accuracy trade-off. When some notion of fairness is enforced, then usually the predictive accuracy suffers. This trade-off, however, depends on the data and fairness measures [8]. In the case of a perfectly balanced dataset and a perfect model, there is no trade-off between demographic parity and accuracy. For equality of opportunity, given any dataset and a perfect classifier, no trade-off exists. However, often models are far away the optimal frontier. Therefore, the fairness-accuracy trade-off is often treated based on the requirements of applications and use cases. When fairness is prioritized, we would like to acquire data such that the models are similarly accurate for different slices, which may induce lower overall accuracy, and vice versa. Effective data acquisition requires interpreting and debugging models and understanding the intricate distribution characteristics of training data and its synergies with the model performance.

Contributions: We demonstrate Plutus, a human-in-the-loop data distribution understanding and tailoring pipeline, built upon SystemDS [3] (an open-source ML system). The main components of Plutus include 1) SliceLine [22], an exact and efficient enumeration algorithm for finding the top-K problematic data slices, where an ML model performs worse than overall, 2) Distribution Tailoring (Dt) [18], a suite of algorithms for cost-effectively acquiring data,

This work is licensed under a Creative Commons Attribution International 4.0 License.

SIGMOD-Companion '24, June 9–15, 2024, Santiago, AA, Chile
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0422-2/24/06
https://doi.org/10.1145/3626246.3654745

with distribution requirements, from a union of external sources, and 3) a visualization module, for exploring the accuracy-fairness-cost tradeoffs. The integration of these components on SystemDS allows users to efficiently iterate over model training, model debugging, and distribution tailoring, as they gain more insights on problematic slices of data and the impact of distribution on various measures. In our demonstration scenario, the attendees impersonate a data scientist. They can specify their own training datasets and data acquisition sources. The participants observe first-hand how Plutus recommends problematic slices and how it applies various data tailoring techniques to obtain data for desired slices. We proceed to discuss related work (Section 2), a solution sketch (Section 3), and a detailed outline of our demonstration (Section 4).

2 RELATED WORK

Plutus is related to model debugging and data acquisition, as well as tools for their joint, exploratory evaluation.

Model Debugging: Besides traditional model debugging via confusion matrices and their extensions [12], various metrics (e.g., area under ROC curve), interactive model training [9], and basic visualizations (e.g., after dimensionality reduction into 2D), there is a trend toward exploratory model debugging. First, explanation techniques such as occlusion-based explanations [17, 24], sparse linear explanations (LIME) [21], layer-wise relevance propagation [14], and Shapley additive explanations (SHAP) [16] are utilized to better understand (often visually) the reasons of predictions and related errors. Second, finding groups with special properties has also proven very helpful. Existing techniques include data coverage analysis [13], explanation tables [11], slice finding [6, 7, 22], and slice tuning [23]. However, good tooling support for understanding interactive slice improvements is still lacking.

Data Acquisition: Li et al. consider enhancing model accuracy using data markets with a strategy that estimates utilities of providers' data, then allocates budget accordingly, and a sequential predicate selection strategy that focuses on statistically promising data segments for model improvement [15]. The underlying assumption is that the market data follows the same distribution as the desired distribution of the train data. Unlike Dt, Chi et al. propose to first union data sources, cluster the data, then iteratively select and evaluate mini-batches from these clusters, and use multi-armed bandit and reinforcement learning solutions to refine the cluster selection criteria based on evaluation feedback [4]. Fallah et al. consider the problem of acquiring data from privacy-sensitive users for estimating a parameter [10] and establish minimax bounds for the estimation error at varying privacy levels.

3 SOLUTION SKETCH

Figure 1 depicts the overall architecture of Plutus. In the following, we describe a summary of the main components of Plutus (model training, slice finding via SliceLine, distribution tailoring), and the implementation of this iterative pipeline on top of SystemDS.

3.1 Slice Finding

After model training and scoring, we obtain an error vector $\mathbf{e}$ (*independent* of the used ML model) that indicates mispredictions on the train or validation sets (e.g., 0/1 for classification, or mean squared error for regression). Subsequently, we apply SliceLine [22] in order to find problematic slices (conjunctions of attributes, where the model underperforms) for guiding data discovery and sampling.

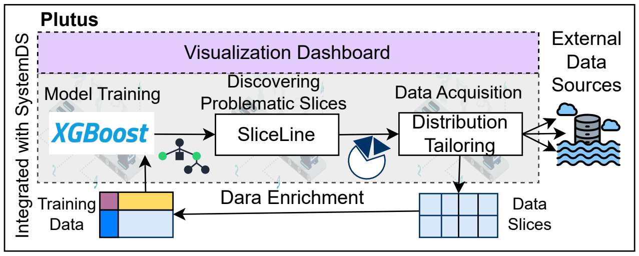


Figure 1: Plutus Architecture.

Problem Formulation: We aim to find the top-K worst slices by maximizing the following scoring function (subject to a minimum-support threshold σ and only positive scores), which balances large errors $\overline{se}$ (problematic subset) and large sizes $|S|$ (coverage):

$$sc = \alpha \left(\frac{\overline{se}}{\bar{\mathbf{e}}} - 1 \right) - (1 - \alpha) \left(\frac{|\mathbf{X}|}{|S|} - 1 \right)$$

where $\alpha \in (0, 1]$ is a weight parameter for slice errors. The compelling properties of this scoring function are that the original data $\mathbf{X}$ has a score of 0, any score larger than 0 is interesting, the error/size terms are balanced for $\alpha = 0.5$ (a slice S with twice the average error $\frac{\overline{se}}{\bar{\mathbf{e}}}$ but half the size of $\mathbf{X}$ has the same score), and the scoring function is amenable to pruning by monotonicity.

Efficient Slice Finding: Given a dataset with m features and d_j distinct items per feature, the lattice of all slices is exponentially large. In order to efficiently find the top-K worst slices, we prune and evaluate each lattice level via a single, (ultra-)sparse matrix multiplication. First, inspired by Frequent Itemset Mining, a slice can only be frequent ($|S| \geq \sigma$) if all its parents (i.e., subsets of predicates) are frequent. Accordingly, in SliceLine [22] on SystemDS [3], we prune by these sizes, but also by upper-bound scores (negative or less than current top-K set). Second, we enumerate all remaining slices $\mathbf{S}$, of a lattice level L, and efficiently evaluate slices via $(\mathbf{X} \odot \mathbf{S}^{\top}) = L$ (matrix multiplication and comparison for all matching predicates), from which we can again derive slice scores and errors. The iterative nature of Plutus allows for additional optimizations (i.e., reusing state), which we do not leverage yet.

3.2 Data Distribution Tailoring

Once the user finds and selects problematic slices, Dt algorithms can assist with cost-effective data acquisition.

Problem Definition: Dt aims to enable integrating data from multiple sources to construct a target dataset. The input query to Dt specifies quotas over some groups. In Plutus, groups are the slices identified by SliceLine. Count requirements are user-specified. Another input to Dt is a collection of sources and their corresponding costs. Data sources can be external, such as crowdsources and data markets, or data views that are defined by a project-join queries over a database or a data lake. We assume tuple-at-a-time access for a fee to each source. Acquiring samples from sources is associated with amortized costs per query, whether monetary, annotation, computation, or integration. For instance, obtaining a source with the same schema as the target schema may involve data discovery and integration. In particular, if the views are expensive to compute, advanced query sampling techniques may be considered [2]. Currently, Plutus assumes sources have the same schema as the target and training data schemas. Given a collection $\mathcal{L} = \{D_1, \ldots, D_n\}$ of data sources and their costs $\{C_1, \ldots, C_n\}$, Dt aims to build a target dataset with the count distribution $\{Q_1, \ldots, Q_m\}$ on slices $\{\mathcal{G}_1, \ldots, \mathcal{G}_m\}$, both decided by the user. The key strategy is to query adaptively, in a sequential manner, to collect samples that fulfill the distribution description, while the expected total cost is minimized.

Algorithms: Dt proposes sampling strategies for scenarios in which data distribution of sources may or may not be known.

RatioColl is an adaptive sampling strategy that chooses a priority slice G^* based on the expected cost of satisfying the counts of each slice independently. Then, the algorithm samples from the data source D^* which minimizes the expected cost to sample it. Let $P_{i,j}$ be the probability to sample G_j from D_i. The priority slice and its corresponding cost-effective source are chosen as:

$$G^* = \underset{j \in [m], Q_j > 0}{\operatorname{argmax}} \left(Q_j \cdot \min_{i \in [n]} \left(\frac{C_i}{P_{i,j}} \right) \right), D^* = \underset{D_i \in \mathcal{D}}{\operatorname{argmin}} \left(\frac{C_i}{P_{i,*}} \right).$$

This forces Dt to prioritize slices that are rare (small $P_{i,j}$), expensive (large C_i), and has a high query count (large Q_j).

When distribution of sources are not available, Dt is mapped to the stochastic multi-arm bandit (MAB) problem, where every data source is an arm and we would like to select arms sequentially.

ExploreExploit is a simple and efficient algorithm with two phases: exploration and exploitation. The time horizon is at most some unknown constant multiple of the total count requirement, denoted Q. As such, Plutus crudely approximates $T \approx Q$ Then Dt sets the exploration phase to be $\alpha T^{2/3}$ iterations for some tunable parameter α, during which ExploreExploit uniformly explores each source. This phase is batched and parallelized. During exploitation, it calls RatioColl as a subroutine, with a modification to estimate $P_{i,j}$ with increasing accuracy as it samples data sources.

Cost Analysis: RatioColl has a tight asymptotic expected cost for the case of two groups and sources, and outperforms various baselines. Since ExploreExploit batches and parallelizes exploration at the beginning, it is often more cost and runtime efficient than asymptotically optimal algorithms. It achieves sublinear regret if the time horizon is accurately estimated. In practice, both strategies outperform random source selection baseline. We refer interested readers to [18] for detailed description and analysis.

3.3 Implementation

Configuration: Plutus is configurable with a config file. The required fields include database connection string(s) for the initial dataset and additional data sources, name and type of input and prediction columns, and bin borders for consistent binning over multiple iterations. On startup, Plutus reads the config file and fetches the initial training data.

Frontend: The dashboard is implemented in Plotly Dash [1]. User inputs trigger callback functions that update the state of global variables to track model performance, slices found, and additional data acquired over iterations. Callbacks also visualize model performance and DT cost breakdown.

Middleware: The frontend issues API calls to a Python middleware which calls database queries, handles data processing, and orchestrates the three phases' main algorithms.

Training: The model training utilizes XGBoost, a popular gradient boosting library that performs well on tabular data [5].

SliceLine: We utilize the SystemDS Python API to find the problematic slices using the data and errors from model training. Under the covers, the DSL-based SliceLine function is compiled to runtime plans according to data and cluster characteristics. Plutus allows exploring different parameters for the individual datasets, including the number of slices k as well as slice size and error weighing α.

Data Tailoring: We implement RatioColl, ExploreExploit, and a random baseline. Priority sources are computed efficiently using matrices. For ExploreExploit, exploration parameter $\alpha = 1/2$ works well in practice. We simulate external data sources by implementing them as PostgreSQL queries.

4 DEMONSTRATION DESCRIPTION

We demonstrate Plutus using the Flights[1] and NYPD-stop&question[2] datasets. We describe intended user interactions in four steps (Figure 2) using the Flights dataset as our running example[3].

Step 1. Model Training: Suppose the goal is to accurately predict the arrival delay of a flight. A user can use xgboost for training boosted trees. The user can verify the validation loss, compare it to train loss. Plutus visualizes the validation loss for existing slices.

Step 2. Model Debugging with SliceLine: To identify problematic slices, a user sets parameters such as scoring parameters α, number of slices to find, the maximum level of lattice to search, and minimum number of support required for each slice. Upon running SliceLine, the problematic slices are displayed and the user can set the desired number of data points for each slice.

Step 3. Data Acquisition and Enrichment with Dt: Plutus displays the distribution of slices in each connected data source for explainability. Next, the user can run RatioColl and ExploreExploit algorithms and compare them with a random baseline. For better understanding of Dt algorithms, the system visualizes the ratio of samples the algorithms have collected from each source. This feedback allows a user to visually investigate the trade-off of cost and data diversity. Finally, the user can enrich the training data with the acquired data and return to the training phase.

Step 4. Model Training and Accuracy-Fairness Trade-off Visualization: The user can iterate through the pipeline with various slice distribution hypotheses and investigate the accuracy-fairness trade-off. The interactivity of the human-in-the-loop Plutus facilitates experimentation and finding the right balance for a usecase.

5 ACKNOWLEDGEMENTS

This work was partially supported by the German Federal Ministry of Education and Research grant BIFOLD24B and, in part, by the National Science Foundation grant 2107050.

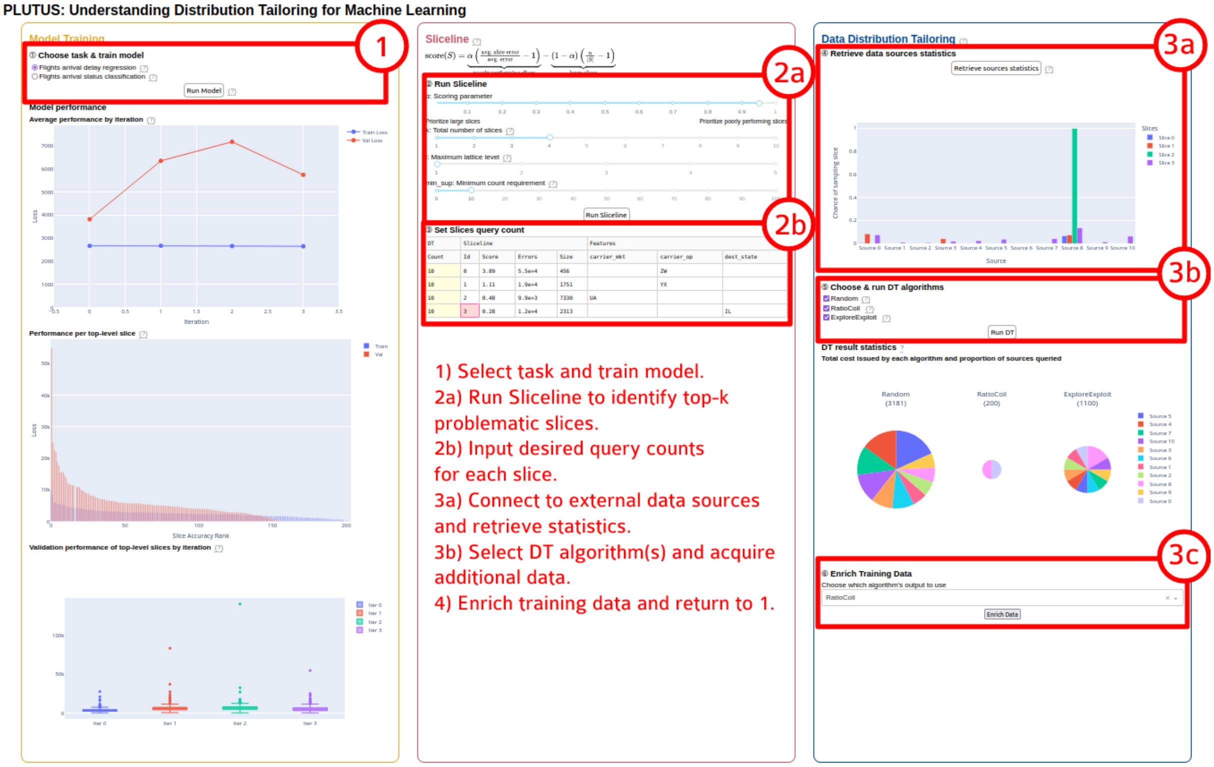
REFERENCES

[1] [n. d.]. Dash Enterprise: The Premier Data App Platform for Python. https://plotly.com/dash.

[1] https://www.transtats.bts.gov/

[2] https://www.nyc.gov/site/nypd/stats/reports-analysis/stopfrisk.page

[3] https://www.youtube.com/watch?v=HDnFFwTULwo


PLUTUS: Understanding Distribution Tailoring for Machine Learning

Model Training

① Choose task & train model

● Flights arrival delay regression

○ Flights arrival status classification

Sliceline

$$\text{score}(S) = \alpha\left(\frac{\text{avg. slice error}}{\text{avg. error}} - 1\right) - (1-\alpha)\left(\frac{n}{|S|} - 1\right)$$

② Run Sliceline

α: Scoring parameter

k: Total number of slices

ℓ: Maximum lattice level

min_sup: Minimum count requirement

③ Set Slices query count

DT	Sliceline				Features		
Count	Id	Score	Errors	Size	carrier_mkt	carrier_op	dest_state
10	0	3.89	5.5e+4	456		ZW	
10	1	1.11	1.9e+4	1751		YX	
10	2	0.48	9.9e+3	7330	UA		
10	3	0.28	1.2e+4	2313			IL

1) Select task and train model.

2a) Run Sliceline to identify top-k problematic slices.

2b) Input desired query counts for each slice.

3a) Connect to external data sources and retrieve statistics.

3b) Select DT algorithm(s) and acquire additional data.

4) Enrich training data and return to 1.

Data Distribution Tailoring

④ Retrieve data sources statistics

⑤ Choose & run DT algorithms

☑ Random

☑ RatioColl

☑ ExploreExploit

DT result statistics

Total cost issued by each algorithm and proportion of sources queried

⑥ Enrich Training Data

Choose which algorithm's output to use

RatioColl

Figure 2: Plutus Demonstration Scenario.

[2] Abolfazl Asudeh and Fatemeh Nargesian. 2022. Towards Distribution-aware Query Answering in Data Markets. *PVLDB* 15, 11 (2022), 3137–3144.

[3] Matthias Boehm et al. 2020. SystemDS: A Declarative Machine Learning System for the End-to-End Data Science Lifecycle. In *CIDR*.

[4] Chengliang Chai, Jiabin Liu, Nan Tang, Guoliang Li, and Yuyu Luo. 2022. Selective Data Acquisition in the Wild for Model Charging. *PVLDB* 15, 7 (2022), 1466–1478.

[5] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. In *SIGKDD*. ACM, San Francisco California USA, 785–794. https://doi.org/10.1145/2939672.2939785

[6] Yeounoh Chung, Tim Kraska, Neoklis Polyzotis, Ki Hyun Tae, and Steven Euijong Whang. 2019. Slice Finder: Automated Data Slicing for Model Validation. In *ICDE*.

[7] Yeounoh Chung, Tim Kraska, Neoklis Polyzotis, Ki Hyun Tae, and Steven Euijong Whang. 2020. Automated Data Slicing for Model Validation: A Big Data - AI Integration Approach. *Trans. Knowl. Data Eng.* 32, 12 (2020), 2284–2296. https://doi.org/10.1109/TKDE.2019.2916074

[8] Sam Corbett-Davies, Emma Pierson, Avi Feller, Sharad Goel, and Aziz Huq. 2017. Algorithmic decision making and the cost of fairness. In *SIGKDD*. 797–806.

[9] Andrew Crotty, Alex Galakatos, Emanuel Zgraggen, Carsten Binnig, and Tim Kraska. 2015. Vizdom: Interactive Analytics through Pen and Touch. *PVLDB* 8, 12 (2015), 2024–2027. https://doi.org/10.14778/2824032.2824127

[10] Alireza Fallah, Ali Makhdoumi, Azarakhsh Malekian, and Asuman E. Ozdaglar. 2022. Optimal and Differentially Private Data Acquisition: Central and Local Mechanisms. In *EC*. 1141.

[11] Kareem El Gebaly, Parag Agrawal, Lukasz Golab, Flip Korn, and Divesh Srivastava. 2014. Interpretable and Informative Explanations of Outcomes. *PVLDB* 8, 1 (2014), 61–72. https://doi.org/10.14778/2735461.2735467

[12] Jochen Görtler et al. 2022. Neo: Generalizing Confusion Matrix Visualization to Hierarchical and Multi-Output Labels. In *CHI*. 408:1–408:13. https://doi.org/10.1145/3491102.3501823

[13] Zhongjun Jin, Mengjing Xu, Chenkai Sun, Abolfazl Asudeh, and H. V. Jagadish. 2020. MithraCoverage: A System for Investigating Population Bias for Intersectional Fairness. In *SIGMOD*. 2721–2724. https://doi.org/10.1145/3318464.3384689

[14] Sebastian Lapuschkin, Alexander Binder, Grégoire Montavon, Klaus-Robert Müller, and Wojciech Samek. 2016. Analyzing Classifiers: Fisher Vectors and Deep Neural Networks. In *CVPR*. 2912–2920. https://doi.org/10.1109/CVPR.2016.318

[15] Yifan Li, Xiaohui Yu, and Nick Koudas. 2021. Data Acquisition for Improving Machine Learning Models. *PVLDB* 14, 10 (2021), 1832–1844.

[16] Scott M. Lundberg and Su-In Lee. 2017. A Unified Approach to Interpreting Model Predictions. In *NeurIPS*. 4765–4774.

[17] Supun Nakandala, Arun Kumar, and Yannis Papakonstantinou. 2019. Incremental and Approximate Inference for Faster Occlusion-based Deep CNN Explanations. In *SIGMOD*. 1589–1606. https://doi.org/10.1145/3299869.3319874

[18] Fatemeh Nargesian, Abolfazl Asudeh, and H. V. Jagadish. 2021. Tailoring Data Source Distributions for Fairness-aware Data Integration. *PVLDB* 14, 11 (2021).

[19] Neoklis Polyzotis, Sudip Roy, Steven Euijong Whang, and Martin Zinkevich. 2017. Data Management Challenges in Production Machine Learning. In *SIGMOD*.

[20] Propublica. [n. d.]. Machine Bias Series. https://www.propublica.org/series/machine-bias.

[21] Marco Túlio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. "Why Should I Trust You?": Explaining the Predictions of Any Classifier. In *SIGKDD*. 1135–1144. https://doi.org/10.1145/2939672.2939778

[22] Svetlana Sagadeeva and Matthias Boehm. 2021. SliceLine: Fast, Linear-Algebra-based Slice Finding for ML Model Debugging. In *SIGMOD*. ACM, 2290–2299.

[23] Ki Hyun Tae and Steven Euijong Whang. 2021. Slice Tuner: A Selective Data Acquisition Framework for Accurate and Fair Machine Learning Models. In *SIGMOD*. 1771–1783.

[24] Matthew D. Zeiler and Rob Fergus. 2014. Visualizing and Understanding Convolutional Networks. In *ECCV*, Vol. 8689. 818–833. https://doi.org/10.1007/978-3-319-10590-1_53