Actionable Data Insights for Machine Learning

Manuel Bähr
Apple
Heidelberg, Germany
mbaehr@apple.com

Nils Braun
Apple
Heidelberg, Germany
nbraun@apple.com

Katrin Honauer
Apple
Heidelberg, Germany
khonauer@apple.com

Ming-Chuan Wu
Apple
Seattle, WA, USA
ming-chuan.wu@apple.com

Abstract

Artificial Intelligence (AI) and Machine Learning (ML) have made tremendous progress in the recent decade and have become ubiquitous in almost all application domains. Many recent advancements in the ease-of-use of ML frameworks and the low-code model training automations have further reduced the threshold for ML model building. As ML algorithms and pre-trained models become commodities, curating the appropriate training datasets and model evaluations remain critical challenges. However, these tasks are labor-intensive and require ML practitioners to have bespoke data skills. Based on the feedback from different ML projects, we built ADIML (**A**ctionable **D**ata **I**nsights for **ML**) – a holistic data toolset. The goal is to democratize data-centric ML approaches by removing big data and distributed system barriers for engineers. We show in several case studies how the application of ADIML has helped solve specific data challenges and shorten the time to obtain actionable insights.

Keywords: data-centric machine learning, interactive diagnostics, data insights

ACM Reference Format:
Manuel Bähr, Nils Braun, Katrin Honauer, and Ming-Chuan Wu. 2023. Actionable Data Insights for Machine Learning. In *3rd Workshop on Machine Learning and Systems (EuroMLSys '23), May 8, 2023, Rome, Italy*. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3578356.3592581

1 Introduction

Applied ML is a highly iterative experimentation process with intertwined loops of data preparation, training tuning, and failure analysis. With advancements in hardware and software, many efforts have contributed to training acceleration [10] and training automation [9], leading to fast loops of model training. However, the data activities, including the initial training dataset curation, post-training failure analysis, and model evaluation, remain a time-consuming manual process. Continuous monitoring and data updates require a certain level of automation to guarantee a satisfactory model quality over time. Data activities often require cross-disciplinary expertise of distributed system and data science literacy, which further prolongs turnaround time.

Post-training failure analysis is another data intensive task. Training metrics need to be analyzed at a subpopulation level to reveal possible limitations or biases. ML engineers need to slice and dice the high-dimensional feature space of failure cases so as to identify hidden patterns where models perform poorly. Often time, they also need to cross reference distributions of different feature dimensions between failure and success learning samples. The complexity of analytical queries and the accessibility of data pose barriers that prevent ML engineers from obtaining actionable data insight promptly. The ever-growing volume of data further exacerbates the challenges and prolongs the turnaround time of failure analysis.

Based on interview feedback from internal ML teams, we derived the requirements for ADIML (Actionable Data Insights for ML) to democratize data-centric methods for ML. ADIML is aimed at enabling data-centric ML [4] approaches in a simple and intuitive way so that data scientists and ML engineers can focus on troubleshooting data and model performance. The high-level declarative programming interface allows users to fully leverage the rich and familiar Python ML ecosystem without an additional steep learning curve. Interactive data diagnostics allows ML engineers to experiment data improvements with rapid feedback. The main contributions include the following.

- We summarize the data challenges and pain points based on the interviews of various ML teams.
- We present the design of ADIML that allows ML engineers and data scientists to directly explore, prepare, drill-in, and diagnose data without the prolonged turnaround time due to the data and system barriers.
- We present real-world case studies on how ADIML helps improve ML productivity. The case studies range from actionable model analysis, to sophisticated training data curation, to rapid prototyping of novel multimodal models, to reliable dataset reports.

The rest of the paper is organized as follows. In Section 2, we describe the motivation of this work based on user challenges and pain points. In Section 3, we discuss the high-level

EuroMLSys '23, May 8, 2023, Rome, Italy

ACM ISBN 979-8-4007-0084-2/23/05…$15.00
https://doi.org/10.1145/3578356.3592581

design and implementation of ADIML. Next, we present use cases in Section 4. Related work is discussed in Section 5, followed by concluding remarks and future work in Section 6.

2 Motivation

We focus on the data challenges associated with improving ML models. In order to understand specific challenges and pain points, we interviewed more than a dozen of ML practitioners who work in different projects of various ML domains. The projects range from early-stage research to large-scale production features. The ML applications include image understanding, machine translation, voice isolation, and text understanding. The raw data used by those applications range from tens of thousands to millions of images, audio snippets, videos, translations, and others. The teams we interviewed mostly consist of specialists with deep domain expertise on different fields, such as linguistics, data engineering, and ML research and engineering.

We reflect the interview feedback on the recent movement of data-centric AI [4] and the data roadmap proposed in [8], including *data design for AI*, *data sculpting for AI*, and *data strategies for model testing*. As a result, we identified and prioritized the following obstacles that slow down the adoption of data-centric methods based on the degree of impediments.

1. Existing big data technologies require distributed system and parallel data processing expertise. ML teams either depend on dedicated teams to manage resources and conduct the data tasks, or have to acquire additional data and system expertise and absorb those overheads. This barrier prohibits ML engineers from **rapid ad-hoc data experiments**.
2. Throughout the ML lifecycle, development environments range from local machines, to data centers, to devices. In addition, ML data appear in a wide variety of data formats and data storage systems. A data solution that abstracts away the physical compute and storage fabric helps ML engineers **focus on the quality of data and models**.
3. ML model development is a scientific experiment. Every step in the ML lifecycle is a part of the experimental setup. As a result, the experiment quickly expands into a complex graph of different experiment configurations. Such a complex experimental process requires **collaboration and reproducibility**. Reproducibility enables efficient collaboration. By easily reproducing any failure analysis, it enables team members and partner teams to verify, to refine, and to improve the ML models.

3 Design and Implementation

We describe the high-level design goals and implementation through simple examples in this section and continue discussions on real-world use cases in Section 4.

3.1 Overview

ADIML consists of three components – *Data Diagnostics*, *Data Validation*, and *Interactive Session Service*. These components cover all major areas of data design, data sculpting, and model evaluation in ML model development lifecycle.

The Data Diagnostics component offers a software development kit (SDK) to define custom data visualization that supports interactive roll-up and drill-down, slicing-and-dicing by configured dimensions, and dynamic filters through a graphical user interface (GUI). The ability to roll up to statistical summaries and to drill down into individual raw assets in real-time over large scale of data enables engineers to obtain actionable data insights easily. The Data Diagnostics component provides its applications with *data access transparency* and *scaling transparency* by leveraging open source data processing frameworks like Dask [11] or Spark. This is achieved by establishing an abstraction for the necessary computation of aggregate statistics as well as filtering. We provide an in-memory implementation as well as distributed implementations so that aggregations and filters can be executed on a cluster of machines.

The Data Validation component provides an SDK for defining custom data validation logic. Data Validation is similar to unit-tests in software development, but on data. It offers a programmatic way to guarantee desired quality of the data before training. It also offers a programmatic way to cross validate training data with evaluation data during failure analysis.

Finally, the Interactive Session Service provides versioned, configurable, and persistent interactive sessions for the ML teams to share, to collaborate, and to reproduce results. Currently, the service provides Jupyter Notebook sessions as the development environment. Each session allows personalized configurations for code assets and software bundles from a software version control system, such as `git`. Data Diagnostics, and Validation SDKs can seamlessly transition from interactive session to unattended batch execution.

3.2 Data Diagnostics

To support data understanding, data exploration, and data improvement via an interactive GUI, the Data Diagnostics component provides the `Explorer` concept to define visualization elements in `View`, such as single column or multidimensional histograms, and custom raw data previews. One can also add custom `Filter` conditions to slice and dice data into desired subsets.

Listing 1 shows an example of creating an interactive statistical analysis for the data design and sculpting steps in Python. The interactive GUI supporting real-time data summary and filtering is shown in Figure 1. The `Explorer` expects a Pandas DataFrame or equivalent distributed data structure

as input. The user-defined function (UDF) `load_dataset` abstracted out the nuisances of loading data from bespoke data systems.

Listing 1. The sample code creates interactive statistical analysis.

```
def show_sample(row, **kwargs):.. # images with bounding box

animal_dataset = load_dataset(ANIMAL_DATA)
Explorer(filters = ["animal"],
        views = ["animal", "weight",
               ("animal", "background"),
               SampleView(details = [
                   CustomDetail(show_sample),
                   detail.Row(["animal", "background",
                        "weight", "age"]),
])]).show(animal_dataset)
```

The interactive GUI makes it easy for human experts to identify patterns, trends, or anomalies in large datasets. In addition to the initial dataset curation, the interactive visualization will also allow users to drill into error regions visually or audibly during model failure analysis. The visualization is integrated with Jupyter Notebooks for real-time programmability so that the applications can be deployed as a web page for no-code data exploration, or as a Notebook session for rapid prototyping. The Data Diagnostics component is extensible such that users can bring their own custom, task-specific visualization libraries, such as those introduced in [3].

Table 1 in Appendix A shows the programming API of the Data Diagnostics component. It consists of basic elements such as `Filter`, `View`, and `Detail`, which can be composed, customized, and re-used in `Explorer` specifications.

3.3 Data Validation

Similar to *check constraints* in database systems, a data validation can simply be a constraint on the column domain or a check for missing values. Data validation can also perform more comprehensive checks, such as setting expectations on columns' means, or setting an expectation on the maximum variance of a distribution. Expectations are also extensible to take user-defined functions, *e.g.*, using a pre-trained model to validate the object category distribution in a new dataset.

Table 2 in Appendix A shows the programming API of the Data Validation component. It consists of `filters` and `expectations`, which can be composed and customized in a `Validator` specification. The sample code in Listing 2 checks the dummy animal dataset that 1) animals must be one of the four types specified in the list, 2) the bounding box area must be bigger than 16000 pixels, 3) the *mean* age of animals in each group must be less than ten, and 4) the weight of each animal must be within 0.6 unit of its own group's *mean* weight. The execution of the `expectations` are optimized with the common sub-expression optimization, and the evaluation will scale out transparently based on the data volume. The execution of the `Validator` returns a Boolean value for each `expectation`, as well as an interactive GUI for drilling into the failed `expectations`. Figure 2a shows two out of four `expectations` failed. Engineers can drill into one of the failures to see which animal category failed the `expectation` in Figure 2b. The ease-of-use data validation abstraction enables every engineer to easily conduct data quality checks before training happens, instead of spending time on tedious model/data troubleshooting afterwards.

Listing 2. The sample code defines four expectations on the animal dataset, where `bbox_area`, `mean`, and `max_diff_from_mean` are UDFs.

```
results = Validator(
    expectations = [
        Cell("animal") in ["elephant", "tiger", "lion",
             "giraffe"],
        Cell("box", bbox_area) < 16000,
        Group("animal", Column("age", mean) < 10),
        Group("animal", Column("weight", max_diff_from_mean)
             < 0.6)
    ]
).run(animal_dataset)
```

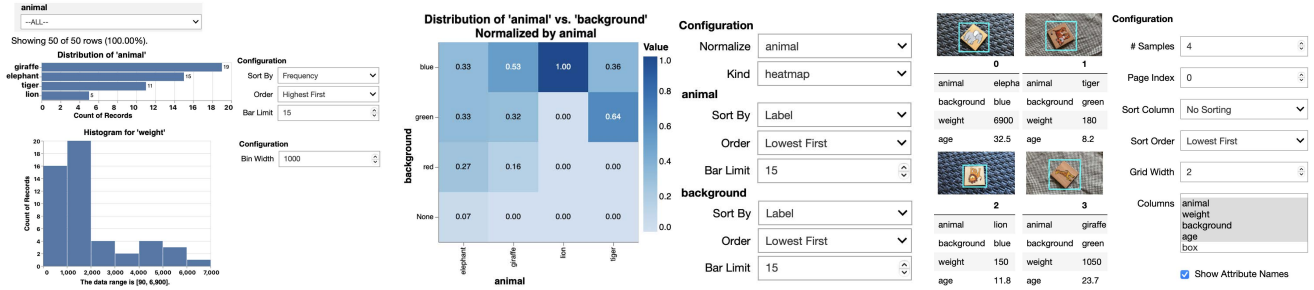
3.4 Interactive Session Service

Jupyter Notebook is a vital tool for data scientists and ML engineers to conduct rapid prototyping, troubleshooting, and to share results. However, collaboration requires manually passing the Notebooks around in various communication channels, making it difficult to keep track of who made what changes or which is the last known good version. Worse even, if the collaboration requires other software bundles, ensuring the integrity between code and configuration poses significant overheads.

To tackle these common pain points, we designed and deployed the Interactive Session Service, which is built on top of a general Jupyter Notebook service. The Interactive Session Service integrates with an in-house ML lifecycle management service, authentication and authorization services, a cloud storage service for session configurations, and a project-centric catalog for easy sharing and collaboration. The session configurations are strongly versioned and contain references to `git` repositories so that code version control is handled by `git`. Users have the capability to pin a session to a particular `git` hash for reproducibility.

Once a session is configured, with a clickable URL users can start an interactive Jupyter Notebook session with prepopulated software bundles, user code, and other assets based on the configuration. The URL can now be shared to enable collaboration easily and to ensure the consistent progress. Common example use cases include data validation, data exploration, or prototyping/debugging data pipelines.

To promote insight sharing, the service also offers the possibility to store and share artifacts. For example, the results from data validation and diagnostics examples from the

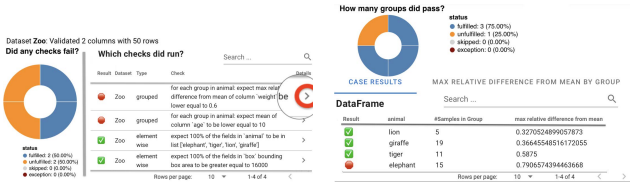


(a) An interactive filter and statistical summaries of animals and weights.

(b) A 2-D distribution of animal and background color.

(c) A sample image preview.

Figure 1. Exploring the animal dataset via interactive summaries and previews.



(a) Clicking the arrow circled in red to drill into failures, as shown in (b).

(b) Details showing which group(s) failed the expectation.

Figure 2. Interactive Data validation summary report.

previous section can be exported as static and self-contained HTML files, which include all graphical and textual representations of the results. The result files can either be archived for later reference, or embedded in rich READMEs for dataset documentation. For reproducibility of the results, the Interactive Session Service can turn these stored artifacts back into interactive live sessions. This not only allows team members to retrace the steps that lead to the results, but also to be able to continue or change the analysis.

4 Case Studies

To validate the initial product and the future directions, we have conducted case studies on various ML domains with positive feedback from our users.

Case 1: Robust and Reproducible Dataset Documentation

In this case study, we worked with a data engineering team whose charter is to curate datasets for downstream ML teams to conduct model training. Their pipelines start with data acquisition, annotation, QA, and end with publishing the datasets. The data range from images, to videos, to text. In order to support as many ML teams as possible, the team employs a high level of automation and standardization in their data preparation workflows. Initially, their existing pipelines met all the individual data requests. However, as the number of ML projects grew, the team faced additional scale challenges due to increased loads of specialized adjustments on the dataset composition and data request coordination with overlapping requirements.

Using ADIML, the team creates interactive dataset documentations, similar to data sheet proposed in [5], to allow their users to self-service data creation and sculpting. The rich interactive documentation enables data consumers from different ML teams to explore and filter existing datasets before initiating a new data curation request. The benefits are two-fold. First, exploring existing datasets and reusing them eliminates the duplicate efforts of starting a costly and time-consuming user-study and data acquisition. Second, since the data acquisition and preparation represents a significant amount of time spent on an ML project, any reduction in latency during this phase presents material gains in productivity.

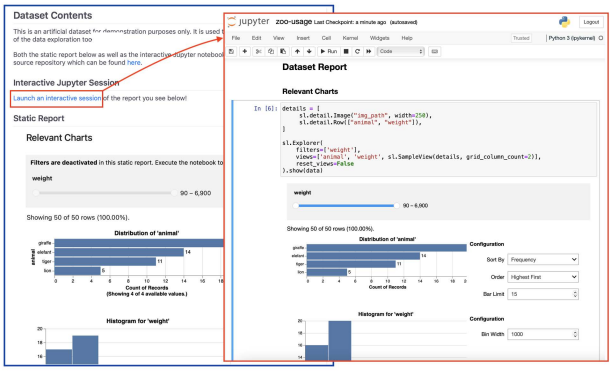


Figure 3. A static dataset summary can be easily turned into a live Notebook session for interactive data exploration.

The interactive dataset documentation is a reusable Notebook template that consists of necessary software packages and a set of tailored visual components using ADIML, as shown in Figure 3. The visual components may include statistical summaries and custom individual raw asset previews. The dataset owners can use the template to generate a static dataset report and publish the report with the dataset. The dataset consumers can use the template to initiate an interactive Notebook session to further understand the dataset composition or to dive deep into individual samples by slicing and dicing various feature dimensions. In the interactive session, the consumers can program additional `Explorer` or

4

`Validator`, as discussed in Section 3, to meet their custom data needs.

The data engineering team, in this case study, published a dataset Notebook template for each dataset to allow ML teams to self-service dataset composition and sculpting. Static reports derived from the Notebooks serve as easily accessible dataset summaries which are updated automatically with the creation of each new dataset version. ML teams can easily explore existing datasets by invoking the template into an interactive data exploration session to assess the suitability of a dataset. The Notebook also serves as a living documentation on how to consume the data, how to visualize individual raw assets, and how to join different datasets.

Case 2: Large Corpora Curation with Linguists in the Loop

Curating a high-quality corpus is a time-consuming effort, that often requires linguists and engineers to work in lockstep. In this case study, a huge volume of variety of language content was extracted from the web. However, it also contains *noise* in the data. After inspecting sample content, the linguists conveyed the idea of noise filtering to data engineers. The filtered data was then presented to the linguists for validation. Such iteration repeats as many times as necessary until a satisfactory corpus is created. In most scenarios, the linguists and the engineers will go back and repeat the entire process to refine the corpus if the trained ML models on the original corpus do not produce adequate quality.

The iterations between different domain experts can be tedious and time-consuming. A small communication hiccup or a minor change request in the filter parameters will result in another iteration. Between iterations, the ML project is in a busy-waiting state resulting in loss of productivity. The problem of productivity losses exacerbates as the iteration latency increases due to the increase of data volume.

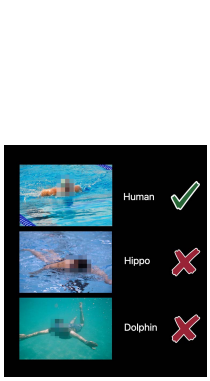
By employing ADIML, the engineers implement an interactive GUI for data exploration and filtering. It allows the linguists to fine-tune the filters by themselves without having to rely on engineers to intervene. It also enables engineers to quickly implement new filters, which the linguists may come up with, regardless of the scale of the data volume. With a little training, the linguists can even implement the visual filtering and exploration by themselves.

In this case study, the initial corpus contains 100+ million paragraphs, but some text contains large portion of numbers that are not very useful. Using a user-defined `digit_ratio` filter allows the linguists to easily remove text that exceeds the desired digit ratio. Visual samples of the filtered text are displayed in real-time, as shown in Figure 4. The statistical summaries over the entire filtered text are also available within seconds. With the reduced overhead within each iteration and the reduced number of iterations, the team can scale up the corpora curation for more language locales than previously possible within a given timeframe.

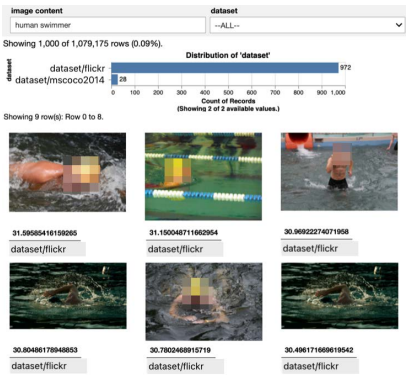
Figure 4. Using the `digits_ratio` slider to filter text.

Case 3: Actionable Failure Analysis of an Image Classifier

In this application, we demonstrate how ADIML helps with failure analysis of an image classification project. During the project, a candidate model yields a higher confusion ratio among “human”, “dolphin”, and “hippopotamus”. While there were no immediate correlations between incorrect predictions and available metadata, a visual inspection of the samples quickly revealed clues. Using sample previews in ADIML, as shown in Figure 5a, engineers quickly recognized a failure pattern that many of the incorrect predictions occurred on scenery with water. To test the hypothesis, the team used an image-text encoder model to search for additional images from multiple data sources with and without water scenery in the same Notebook session, as shown in Figure 5b. After the re-trained model exhibits improvements in both accuracy and recall, the team initiated a complementary data collection and annotation, adjusted the training data, re-trained the model, and re-ran the model analysis to verify that the misclassification problem is mitigated.



(a) Sample preview of failed images.



(b) Using an image-text encoder model to explore images by natural language phrases.

Figure 5. Actionable Failure Analysis.

The same process is repeated to tackle the next highest confusion errors. In many long-tail problems in ML, the ability to quickly address the most dominant errors and then move on to address the next ones is critical to achieve the desired model quality within a constrained timeframe. ADIML allows the team to iterate fast and to validate the hypotheses

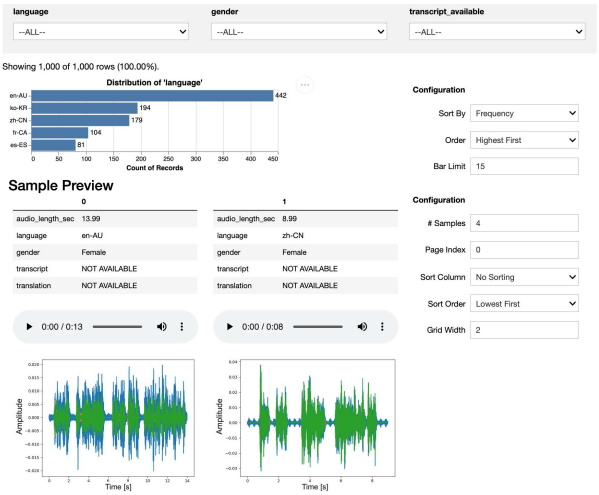


Figure 6. The customizable preview for multi-modal speech translation data: metadata, audio playback, and audio spectrograms.

before taking a potentially costly full-scale data collection and annotation.

Case 4: Rapid Iteration on Multi-Modal Datasets from Heterogeneous Data Sources

In this case study, we investigated a multi-modal machine translation ML project in its early stage. One of the challenges was due to the high variety of data sources, large volume of data, and inconsistent data qualities. In fact, these are common challenges that many ML teams face. The data modalities spanned from speech audio snippets, transcriptions, translations, to various combinations of these modalities.

Using the custom sample previews the user could listen to speech snippets together with visual inspection of the audio spectrograms, transcripts, translations, and derived metadata, such as predicted language or segment length, as shown in Figure 6.

The capability to inspect the data visually and audibly allows quick assessment of the data quality at this early stage of the ML project lifecycle. The validation library can be used to specify basic desired properties of the training dataset, such as "*transcripts should not be empty*", "*audio files should not be corrupted*", or "*original and translated phrase should be of similar length*". The set of validation criteria was then compiled into a unit-test suite for the training data. This unit-test suite becomes the automatic quality assurance for the training dataset as the dataset evolves.

5 Related Work

The importance of data quality for ML has attracted recent attention from both industry and academia [1, 2, 6, 12, 13]. Prior work focus on the validation of the training datasets, either using traditional algorithms or employing ML to predict the data quality metrics, or to detect anomalies in the data. They serve as inspiration and a reference of our work. ADIML enables interactive data diagnostics, both visually and audibly, to assist data activities throughout the ML lifecycle.

Data visualization is an active discipline in computer science by itself. It is not our goal to create novel data visualizations, but instead ADIML combines data diagnostic algorithms with user-defined interactive visualization components. Prior works, such as [3, 7], presented how data visualization helps ML engineers identify data anomalies effectively, can be integrated with ADIML to enrich its visualization experiences. Similarly, other visualization libraries, such as `matplotlib`, `plotly`, `altair`, or LUX, are complementary to ADIML.

Other data visualization BI tools, such as Tableau and Superset, are similar to ADIML in the way that they can create interactive data visualizations. However, they require clean, rectangular data as inputs. Scalability, pricing, and vendor lock-ins might also be concerns when choosing a data visualization solution. ADIML is lightweight Python package and depends on a minimum set of OSS packages. ADIML's data engine is horizontally scalable, based on open source project Dask [11], and it allows data processing of both rectangular data and unstructured blob data in a consistent programming experience. The data processing of ADIML can natively span heterogeneous computes (server machines or mobile devices) based on the modeling fidelity requirements. Lastly, ADIML can easily integrate with pre-trained ML models as user-defined functions for data diagnostics. The extensibility to take user-defined functions as the first-class concept in ADIML will enable ad-hoc use cases to satisfy the experimental nature of applied ML.

6 Concluding Remarks

We presented ADIML, a toolset to democratize data technology throughout the ML lifecycle and to enable the data-centric ML approach in a simple and intuitive way. The design of ADIML is based on the set of challenges and pain points we collected and validated from a wide range of ML teams. The case studies showing how easily ADIML can enable ML teams to focus and to improve data quality at scale are testimonies of its values.

Currently, ADIML provides the programming framework with limited set of system-defined intrinsic functions to cover common use cases. We would like to accelerate its adoption by creating a rich content of UDFs for data visualization, data validation, failure analysis, *etc.*, as reusable data recipes. In particular, we will integrate ML models as UDFs to assist data curation as well as failure diagnosis in the near future.

References

[1] Niels Bantilan. 2020. pandera: Statistical Data Validation of Pandas Dataframes. *Python in Science Conference (SCIPY)* (2020).

[2] E. Breck, M. Zinkevich, N. Polyzotis, S. Whang, and S. Roy. 2019. Data validation for machine learning. *Proceedings of the 2nd SysML*

Component	Arguments	Description
`Explorer`	`(filters, views)`	The `Explorer` supports explorative statistical analysis by combining `filters` and `views`. The `views` show summary statistics of the filtered data. Both `filters` and `views` can be of type `Filter` or `View` (as described below) or plain string type denoting column names. For the latter, ADIML derives suitable `Filter` and `View` objects based on the column.
`Filter`	`(column, **kwargs)`	`Filters` are column predicates and return a filtered dataset. Available built-in filters include: `CategoricalFilter`, `NumericalFilter`, `RegexFilter`. and `CustomFilter`. The `CustomFilter` allows filter operations based on a user-defined Boolean function.
`View`	`(column, **kwargs)`	`Views` define data visualization schemes. Available views are `CategoricalChart`, `NumericalChart`, `Chart2D`, `TableView`, `SampleView`, and `CustomView`. The `CustomView()` allows custom data summaries and integrates with existing plotting functionality. The `SampleView` accepts a list of customized `Details` (defined below). It also supports sorting and pagination.
`Detail`	`(column, **kwargs)`	`Details` visualize information per row. Available details are `Row`, `Image`, `Audio`, `Video`, and `CustomDetail`. The `CustomDetail()` allows custom sample previews, *e.g.*, user-defined functions which create audio spectrograms or images with bounding boxes.

Table 1. Programming API for Data Diagnostics.

Component	Arguments	Description
`Validator`	`(filters, expectations)`	The `Validator` supports data validation by combining `Filters` and `Expectations`. `Filters`, as defined in Table 1, are Boolean functions for selecting the target subset of data for validation. An `Expectation` is a supposition that a predicate on a given data *context* is true. A `Context` can be either `Cell`, `Column`, `Row`, or `Group`. Each expectation checks whether the specified data characteristics are met on the filtered subset of data.
`Cell`	`(column, transformation, predicate)`	`Cell` performs checks defined by the `predicate` on a single cell. If `transformation` is specified, it will be applied to the cell before the evaluation of the predicate.
`Column`	`(column, transformation, predicate)`	`Column` performs checks on the vector-value of the specified column. Similarly, `transformation` is optional and is evaluated before the predicate.
`Row`	`(transformation, predicate)`	`Row` performs checks on a row basis.
`Group`	`(column, context)`	`Group` performs checks defined in the `context` after grouping the data on the defined column. The `context` is used to express the condition that is evaluated on each group.

Table 2. Programming API for Data Validation.

Conference (2019).

[3] Alex Bäuerle, Ángel Alexander Cabrera, Fred Hohman, Megan Maher, David Koski, Xavier Suau, Titus Barik, and Dominik Moritz. 2022. Symphony: Composing Interactive Interfaces for Machine Learning. *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems* (2022).

[4] DeepLearningAI. 2021; accessed April, 2021. A Chat with Andrew on MLOps: From Model-centric to Data-centric AI. https://www.youtube.com/watch?v=06-AZXmwHjo.

[5] Timnit Gebru, Jamie Morgenstern, Briana Vecchione, Jennifer Wortman Vaughan, Hanna M. Wallach, Hal Daumé III, and Kate Crawford. 2021. Datasheets for datasets. *Communications of ACM* 64, 12 (2021).

[6] Hannes Hapke and Catherine Nelson. 2020. *Building Machine Learning Pipelines, Chapter 4: Data Validation.* O'Reilly Media, Inc.

[7] Fred Hohman, Kanit Wongsuphasawat, Mary Beth Kery, and Kayur Patel. 2020. Understanding and Visualizing Data Iteration in Machine Learning. *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems* (2020).

[8] Weixin Liang, Girmaw Abebe Tadesse, Daniel Ho, L. Fei-Fei, Matei Zaharia, Ce Zhang, and James Zou. 2022. Advances, challenges and opportunities in creating data for trustworthy AI. *Nature Machine Intelligence* 4, 8 (2022), 669–677.

[9] Lizhi Liao, Heng Li, Weiyi Shang, and Lei Ma. 2022. An Empirical Study of the Impact of Hyperparameter Tuning and Model Optimization on the Performance Properties of Deep Neural Networks. *ACM Transactions on Software Engineering and Methodology* 31, 3 (2022).

[10] Don Monroe. 2022. Accelerating AI. *Communications of ACM* 65, 3 (2022).

[11] M. Rocklin. 2015. Dask: Parallel computation with blocked algorithms and task scheduling. *Proceedings of the 14th python in science conference* (2015).

[12] Christian Ruiz. 2018. Improving Data Validation using Machine Learning. *Conference of European Statistics, Workshop on Statistical Data Editing* (2018).

[13] S. Schelter, D. Lange, P. Schmidt, M. Celikel, F. Viessmann, and A. Grafberger. 2018. "Automating Large-Scale Data Quality Verification". *Proceedings of the VLDB endowment* 11, 12 (2018).

A Programming API

Tables 1 lists the high-level concepts provided by the Data Diagnostics component. Applications start by defining an `Explorer`, which contains `Views` and optional `Filters`. `Filters` are Boolean functions for selecting the desirable subset of data. `Views` are defined by `Details`, which implement the appropriate visualization. The resulting `Explorer` object is schema-bound, not data-bound. To invoke an `Explorer`, one must apply it to an input dataset with compatible schema. As a result, an `Explorer` is sharable and reusable.

Table 2 lists the data validation API. Applications start by defining a `Validator`, which consists of optional filters and a list of expectations. `Filters` are used to select the subset of data for validation. `Expectations` are suppositions that the selected data meet certain criteria. Expectations can be expressed on cells, vectors, rows, or groups. Data transformations can also be used to define expectations on computed data. `Validators` are also schema-bound, sharable, and reusable.

Both APIs require tabular input. However, they are most commonly used for unstructured data. In those cases the table typically consists of metadata about the unstructured data that allows efficient filtering or checking. This metadata is typically inferred from the underlying raw data or acquired via human annotation. The Data Diagnostics component supports this setup explicitly by visualizing raw data that is given in form of a reference to the underlying data blob. This also reduces the amount of necessary data transfer to those elements that have been selected via filters.