# Towards machine learning-aware data validation

Sebastian Strasser[1]

[1]University of Regensburg, Bajuwarenstraße 4, 93053 Regensburg

**Abstract**

An important task when operating machine learning applications is model monitoring. Teams operating machine learning pipelines monitor the model performance based on common machine learning metrics like accuracy. However, in many real-world applications, monitoring model performance is difficult as ground truth is required to evaluate the performance. One possible way to draw conclusions about the current state of the pipeline is to observe drifting data, i.e., serving data deviating from the training data. However, approaches which give alerts on changing data are often too sensitive, leading to many false alarms. We propose an approach which provides more actionable data validation for machine learning monitoring. It is based on building so-called data assertions from initial training data. These assertions are then used as constraints to detect unexpected changes and data errors.

**Keywords**

data monitoring, machine learning, data validation, data assertions

## 1. Introduction

In the last decades, the field of machine learning has gone through tremendous progress which led to a wide adoption in numerous applications in academia and industry. Significant effort is spent on developing and optimizing algorithms. Another important factor in machine learning applications is the utilized data. The research field of *data-centric AI* – which can be seen as complementary to *model-centric AI* – thus addresses data aspects in machine learning applications [1]. One important factor is data maintenance for machine learning pipelines, including both training and serving data. Figure 1 shows a high-level representation of a machine learning pipeline: Training data is preprocessed and used for training. The output of this process is a model. In the serving environment, predictions are made for unseen data after a data preprocessing. When operating such machine learning pipelines, a major challenge is training-serving-skew [2], i.e., a deviation of the training to the serving environment. In particular, serving data which differs significantly from training data can cause numerous problems. When considering learning on tabular data, *structural changes* can cause errors in the serving pipeline, hence heavily influencing the model outcome or even breaking pipelines. But also differences in data characteristics can have significant impact on the downstream model. One frequently researched challenge in this context is *concept drift*. This term refers to a change of relationship between input and output data over time. Concept drift can have massive impact on model performance. Detecting such changes is therefore an important task which we plan to address in our approach.

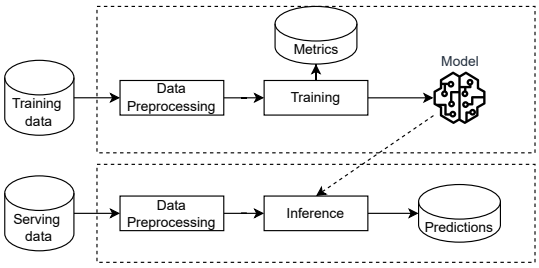


**Figure 1:** Schematic illustration of a machine learning pipeline (straight arrows indicate data flow, dashed arrows indicate a deployment of artifacts)

A comprehensive data monitoring is needed to detect aforementioned deviations in serving data. This process can be seen as a special type of *data validation* which is not only an important topic in machine learning, but in all applications where data is processed. Data science teams can choose from a vast amount of different tools for this task, e.g., TFX Data Validation [3], Deequ [4], or great expectations[1]. These tools can be effectively used to validate incoming data according to user-defined constraints or a baseline dataset. However, as models and processed data in production machine learning applications are often updated continuously [5], a comparison of serving data with a static baseline is not sufficient. There is a need for data validation tools for machine learning applications where this dynamic nature is considered.

Another limitation of many data validation tools used for machine learning is that their output does not lead to a quick and easy diagnosis of the underlying problem. Thus, Polyzotis and Zaharia [5] identifies outputs to be

*35th GI-Workshop on Foundations of Databases (Grundlagen von Datenbanken), May 22-24, 2024, Herdecke, Germany.*

sebastian.strasser@ur.de (S. Strasser)

0009-0001-8848-1368 (S. Strasser)



[1]https://greatexpectations.io/

*actionable* as one core requirement of monitoring tools for data-centric AI. *False alarms* can be a problem, too. When a systems outputs too many alerts, users tend to ignore or silence them entirely. One way to implement a more helpful diagnosis and to prevent alert fatigue is to alert only on changes which are probably going to have an impact on the downstream model. Here, the challenge lies in finding types of deviations which are likely to impact the model outcome.

Thus, we propose an approach for a system which monitors data throughout the machine learning lifecycle. It aims at validating new incoming data which is used as serving and/or training data. We focus especially on *actionable* outputs. This requires a deep analysis of the data and the impact of specific data characteristics on the pipeline. As successful machine learning deployments are operated for long time periods and are updated continuously, information used for data validation is also updated continuously. Metadata necessary for building useful constraints and actionable alerts is collected during the training phase.

This paper first reviews literature and tools which are aimed at monitoring and validating data for machine learning pipelines. Subsequently, we present our approach in more detail. We also exemplify identified research challenges and ideas on how to solve these challenges.

## 2. Related work

Multiple fields of research are relevant for our envisioned machine learning monitoring system focusing on data aspects. Firstly, we take a look at tools for metadata management in machine learning experiments. Secondly, approaches for validating data in general are discussed. Afterwards, existing ML monitoring tools are presented.

### 2.1. Metadata management

In our approach, mining metadata from the training process is an important first step. Numerous tools enabling metadata tracking from training pipelines exist. The goal of such tools is to guarantee the reproducibility of machine learning experiments. They also support comparison between different machine learning pipelines in regards to model performance or other metrics.

Schelter et al. [6] suggest a declarative approach to metadata tracking for machine learning pipelines. This refers to a decoupling of the actual artifacts produced in the machine learning process (i.e., code, models, datasets, etc.) with metadata describing these artifacts. Metadata is extracted from internal data structures of machine learning frameworks, e.g., from Spark DataFrames or MXNet computation graphs. This allows a fine-grained tracking and thus good replicability.

*MLFlow* [7] is a widely used tool which provides a tracking API to log metrics and artifacts of machine learning experiments. API calls are inserted into training code by the users which initiate the logging of metadata to a file or database. *MLFlow* then provides an API and UI to display metadata collected during an experiment, enabling comparison between different experiment runs. There are integrations for a lot of machine learning frameworks, thus supporting metadata tracking for numerous use cases.

*TFX ML Metadata*[2] is another library which retrieves metadata from data science workflows. It also allows user to track metadata about artifacts and outputs which are produced during an experiment. This metadata is then stored into a *metadata store* which data scientists can use for debugging models in production. For instance, they can trace the dataset a model is trained with or compare results of two experiments.

As metadata collection from machine learning scripts is an essential step in our approach, we plan on using concepts from these tools. Similar as it is proprosed by *TFX ML Metadata*, we also use the retrieved metadata for debugging in production. However, our focus is to collect and store metadata about the processed data rather than about the model.

### 2.2. Data validation

The main goal of our approach is to ensure that incoming serving data does not break the prediction pipeline or induces other problems like concept drift. Thus, we effectively *validate* serving data, i.e., check if it fits predefined criteria. Obviously, data validation is required in countless applications over multiple fields. Therefore, lots of development and research went into designing systems which validate data.

Deequ [4] is an example for such a system. It enables the validation of large-scale data in respect to the data quality. Users can set constraints or choose from suggestions generated by the system. The main focus of this system is to ensure a performant processing even for very large datasets. Redyuk et al. [8] suggest a system where data quality is monitored by computing descriptive statistics and detecting deviations with novelty detection methods. This is a contrast to other approaches where constraints are set manually or semi-automatically by the user.

There are also data validation systems designed specifically for machine learning applications. *TFX Data Validation* [3] is a part of the platform *TFX* implemented by Google. It enables the validation of both training and

[2]https://www.tensorflow.org/tfx/guide/mlmd

serving data. The authors make a differentation between *single-batch* and *inter-batch validation*. Single-batch validation is supposed to detect anomalies in a single batch of data while inter-batch validation is targeted at finding significant changes between training and serving data or batches of training data. *mlinspect* [9] is a tool which enables users to inspect training pipelines. It mainly focuses on debugging *data distribution changes* induced by pipeline steps. This can be used to detect *technical bias*, i.e., bias which is introduced by data preprocessing or other automated tasks.

Static data validation, as provided by most presented applications, is not sufficient for the problem of continuously monitoring input data for machine learning applications. Also, to the best of our knowledge, there are no systems which validate data based on the impact it is expected to have on a downstream model which is one of the core ideas of our approach.

### 2.3. Monitoring machine learning applications

Kreuzberger et al. [10] identified *continuous monitoring* as one of the main principles of *MLOps* which is a paradigm describing the effective development and operation of production machine learning applications. The monitoring component is needed to detect errors or changes influencing model quality. Various artifacts like data, model outputs, serving infrastructure, etc. are observed. Machine learning teams use various tools for this task. A popular choice are general-purpose monitoring systems like *Prometheus*[3] or the ELK stack[4].

There are also tools specifically designed for the monitoring of machine learning applications. *EvidentlyAI*[5] provides multiple modules to monitor data quality, data drift, and model performance. *Test suites* perform data and model quality checks based on conditions that are either manually set or generated from a reference dataset. *Reports* provide general metrics and interactive visualizations for analysis and debugging purposes. A continuous monitoring can be achieved by storing snapshots of test suites and reports and displaying it in a dashboard. Similar tools exist for various cloud services which provide tooling for machine learning, e.g., *Amazon SageMaker Model Monitor*[6]. In constrast to our approach, these tools incorporate ground truth into the model performance evaluation. We plan to evaluate models under the assumption that ground truth is not available.

A concept which can used for monitoring models is the abstraction of *model assertions* [11] which adapt software assertions to machine learning models. They allow domain experts to specify constraints over model input and output. This enables the detection of wrong model outputs in cases where confidence is high. Our approach also includes assertions, but focuses on the validation of data, not model optimization.

Shankar and Parameswaran [12] present a vision of a so-called *observability system* for machine learning pipelines. One primary goal is to detect and diagnose bugs in machine learning pipelines. The authors also emphasize the importance of measuring model performance with incomplete information, i.e., when no ground truth is available to evaluate model outputs. A prototype for such an observability system was also presented[7]. Our proposed system also tackles challenges that appear in machine learning applications where no ground truth is available. However, while the authors of [12] presented concepts of using partial or delayed labels for performance measurement, we keep the labels out of the equation and focus on detecting data with unexpected characteristics. Thus, our approach can be seen as complementary to the envisioned system presented in [12].

## 3. Concept

An architecture of our proposed system is depicted in Figure 2. It consists of two main components: (i) metadata collection and (ii) data assertion generation. The first component collects metadata from the initial training pipeline. This includes machine learning metrics and the training data. We use this metadata as a *baseline*. The next step is to infer expected data characteristics from the metadata. We refer to these expectations as *data assertions*. By collecting machine learning metrics additionally to the data, it is possible to measure how specific data characteristics affect the outcome of the model. By incorporating the effect on model outcome, more actionable alerts are possible. Teams operating machine learning pipelines are interested in data errors which are likely to impact model results. In the following, we present a general outline for those two components. We also exemplify research challenges and ideas on how to solve these.

### 3.1. Collection and storage of metadata in machine learning pipelines

Firstly, we identify a set of metadata to collect from the machine learning process. We differentiate between two types of metadata: (i) experimentation metadata and (ii) metadata about the training data. To track metadata about the experimentation, we intend to use a tracking tool like *mlflow*. Here, we are mostly interested in how

[3] https://www.prometheus.io/
[4] https://www.elastic.co/elastic-stack
[5] https://www.evidentlyai.com/
[6] https://docs.aws.amazon.com/sagemaker/latest/dg/model-monitor.html
[7] https://github.com/loglabs/mltrace

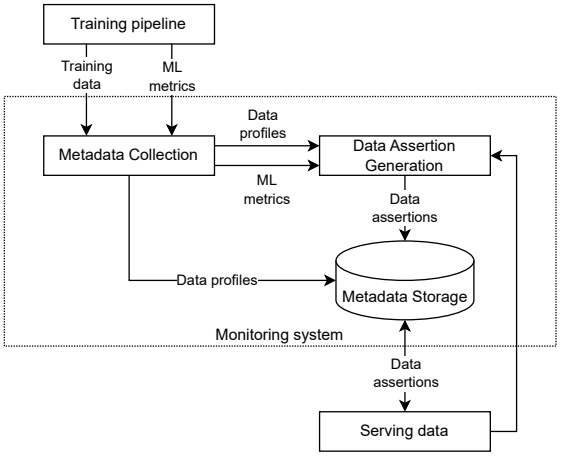


**Figure 2:** High-level overview of the proposed monitoring system

well a model performed on which datasets. Thus, we track *ML metrics* in combination with pointers to the training datasets.

More importantly, we collect comprehensive metadata about the actual data a model is trained with. For this, we create a *data profile* for each training dataset. For now, we focus on profiling tabular data. When considering tabular data, one could utilize comprehensive work on the profiling of relational data [13]. Various tools were developed specifically to profile data used in machine learning applications [14, 15]. However, an extensive profiling is not always feasible for the application of creating data profiles for machine learning datasets, as it would be too expensive for many use cases. Thus, a trade-off between getting insightful information about the data and performance has to be made. A first design of a data profile which considers these requirements is shown in Figure 3. The data profile contains general information and descriptive statistics describing the dataset. We track statistics about numeric and categorical features, as well as relationships between columns. The design of this data profile is not meant to be final, but rather serves as a suggestion which can be adjusted based on the use case.

There are also different possibilities to collect data from training scripts and other machine learning code. One possibility is to provide an API which enables the user to log data from variables or files, similar to implementations in mlflow [7] or other experiment tracking tools. Other approaches where data is captured directly from Python scripts [9, 16] could also be used.

An additional research challenge we identified is to figure out an efficient storage method for the metadata. This is especially important for machine learning applications which are operated over a long period of time, as the training data is updated every time models are retrained. An idea here is to save the data profiles in data structures that allow incremental updates. A similar approach is implemented in Deequ [4] where data quality metrics are incrementally updated for large-scale datasets.

## 3.2. Data assertions

In model serving, we differentiate between two cases: online and batch inference. Online inference means that the input is immediately passed to the model which then outputs a prediction. Thus, the serving data can be seen as unbounded. In batch inference, on the other hand, input data is accumulated into batches. Those batches are then passed to a model and inference is performed on all the collected data. This differentiation is important as these types can require highly different processing techniques.

We tackle challenges regarding the training-serving-skew which is an issue in production ML. Thus, detecting

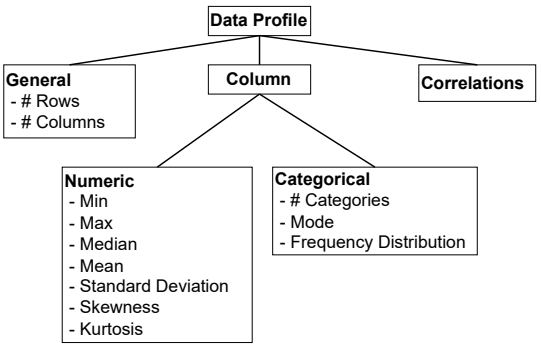


**Figure 3:** Components of a data profile

such differences between training and serving data is crucial to avoid performance decreases. In batch settings, one way to address this problem could be to compare data from the training phase with the serving dataset. Assuming data profiles as described in Section 3.1 are available, this comparison can be done by finding differences between the *baseline* data profile (i.e., data profile of the training data) and the serving data profile.

For unbounded data, the approach of building data profiles for input data is not feasible. As data profiles are meant to be summaries of datasets, they do not serve as a good means to describe unbounded data. Therefore, for this case, our approach to detect changes is to derive so-called *data assertions* from the data profiles that were collected in the training process. These data assertions are thought to ensure that incoming serving data does not lead to errors in the inference pipeline. Thus, they are meant to identify data which is significantly different from training data. If an assertion is violated, the monitoring system gives a warning.

Consider an application where the goal is to predict whether there will be a storm on the next day based on weather data from the current day. One column in this example would be the temperature. A data assertion for this column could look like depicted in Listing 1. In this example, the temperature is guaranteed to be a float and to have a value between -5.2 and 36.7.

```
{
  "column_assertion": {
    "name": "temperature",
    "dtype": float,
    "lower_bound": -5.2,
    "upper_bound": 36.7
  }
}
```

Listing 1: Example for a data assertion (planned)

In the context of data assertions, we identified two research challenges. The first one is the design of data assertions. Listing 1 only shows a rough idea for the design of a specific example. Data assertions can be thought of as a kind of *constraint*. Our goal is to define these constraints in a way that data violating them is likely to cause problems in the pipeline, ranging from pipeline breaking errors to deviations in data distribution which often lead to worse model outputs. For this, we first have to define which errors can occur in the pipeline. Then we can classify which errors are affecting the pipeline and which are not.

We start with "simple" assertions, i.e., data assertions based on simple data statistics like the min and the max. An example for such a data assertion was introduced


**Figure 4:** Process for generating data assertions

prior in Listing 1. These data assertions are similiar to constraints like implemented in Deequ [4] or other data validation tools. We then create a taxonomy of those data assertions, i.e., find suitable *assertion types*. For instance, on a high level, assertion types could be divided into *structural* and *semantical*. Also, assertions can be on feature- or table-level. Table-level assertions include constraints which model relationships between features. In the next step, we plan to cluster those constraints according to what *error types* are produced when data violates them. Here, we differentiate between changes breaking the preprocessing pipeline and changes influencing model outcome (also over time). Therefore, we also incorporate the model into the evaluation. This way we can also measure if specific data assertion violations tend to have similar impact on the downstream model. A research question in this context is to find approaches on how to evaluate effects of assertion violations on the model. An idea is to use metrics which measure the *feature importance* [17].

The outcome of this process are rules which model the relationship of data assertion violation to impact on the pipeline. These can be used to build a classification model. The independent variables are attributes describing the data assertion and the dependent variable is the impact on the pipeline. This model in turn can be used to improve the assertions by prioritizing assertions with a higher predicted impact and neglecting constraints with low estimated impact.

An important part of the concept of data assertions is that users do not have to define them all by hand. Rather, we plan to use a semi-automatic approach which is illustrated in Figure 4. The monitoring system suggests assertions based on the data profiles of the training process to the data scientist. Then the data scientist can accept or reject the assertion. They can also edit the assertion. For the example in which a constraint is generated for the column *temperature*, they could change the upper bound to 40 – if they know such temperatures are realistic in the observed area. This user feedback can also be incorporated into the data assertion generation process, enabling better constraints which produce less false alarms.

In a last step, the data assertions created before have to be evaluated. Several datasets used for benchmarking in machine learning research can be used. However, we will also allow a parametrization of various attributes of the data, e.g. schema or column contents. Variance in

data can therefore be controlled and data with different properties can be tested. In the evaluation, the input data is firstly split into "training" data with which the data assertion generation is executed and "test data" with which the accuracy of these data assertions is then evaluated. The evaluation is separated into two steps: (i) verification of the semantical correctness, i.e., does the generated data assertion hold for the test data, and (ii) comparison of the predicted impact with the actual impact on the model.

## 4. Conclusion

Validating data for machine learning applications is a task with many challenges yet to solve. We focus on the validation of data which is updated continuously and the minimization of false positives. Therefore, this paper proposes a data monitoring system which incorporates a comprehensive collection of metadata in machine learning pipelines and a creation of data constraints we call *data assertions*. The assertion building process is aware of the downstream model and measures the influence of data variance on model outcome. In our next steps, we implement the metadata collection component to collect comprehensive metadata not only on data, but also on the model. This metadata serves as input for the data assertion generation process. We presented research challenges we identified both in metadata collection and data assertion generation. We also made considerations on how to evaluate the generated data assertions.

## References


[1] D. Zha, et al., Data-centric artificial intelligence: A survey, 2023. arXiv:2303.10158.

[2] N. Polyzotis, et al., Data management challenges in production machine learning, in: Proceedings of the 2017 ACM International Conference on Management of Data, Association for Computing Machinery, 2017.

[3] N. Polyzotis, et al., Data validation for machine learning, in: Proceedings of Machine Learning and Systems, volume 1, 2019.

[4] S. Schelter, et al., Automating large-scale data quality verification, in: Proc. VLDB Endow., volume 11, VLDB Endowment, 2018.

[5] N. Polyzotis, M. Zaharia, What can data-centric ai learn from data and ml engineering?, 2021. arXiv:2112.06439.

[6] S. Schelter, et al., Declarative metadata management: A missing piece in end-to-end machine learning, in: Proceedings of SysML, 2018.

[7] M. A. Zaharia, et al., Accelerating the machine learning lifecycle with mlflow, IEEE Data Eng. Bull. 41 (2018).

[8] S. Redyuk, et al., Automating data quality validation for dynamic data ingestion, in: International Conference on Extending Database Technology, 2021.

[9] S. Grafberger, et al., Data distribution debugging in machine learning pipelines, The VLDB Journal 31 (2022).

[10] D. Kreuzberger, N. Kühl, S. Hirschl, Machine learning operations (mlops): Overview, definition, and architecture, IEEE Access 11 (2022).

[11] D. Kang, et al., Model assertions for monitoring and improving ml models, in: Proceedings of Machine Learning and Systems, volume 2, 2020.

[12] S. Shankar, A. G. Parameswaran, Towards observability for production machine learning pipelines, Proc. VLDB Endow. 15 (2022).

[13] Z. Abedjan, L. Golab, F. Naumann, Profiling relational data: a survey, The VLDB Journal 24 (2015).

[14] W. Epperson, et al., Dead or alive: Continuous data profiling for interactive data science, IEEE Transactions on Visualization and Computer Graphics 30 (2024).

[15] F. Clemente, et al., ydata-profiling: Accelerating data-centric ai with high-quality data, Neurocomputing 554 (2023).

[16] L. Murta, et al., noworkflow: Capturing and analyzing provenance of scripts, in: Provenance and Annotation of Data and Processes, Springer International Publishing, Cham, 2015.

[17] M. Saarela, S. Jauhiainen, Comparison of feature importance measures as explanations for classification models, SN Applied Sciences 3 (2021).