SHORT-PAPER

TensorFlow Data Validation: Data Analysis and Validation in Continuous ML Pipelines

EMILY CAVENESS, Google LLC, Mountain View, CA, United States

PAUL SUGANTHAN G. C., Google LLC, Mountain View, CA, United States

ZHUO PENG, Google LLC, Mountain View, CA, United States

NEOKLIS POLYZOTIS, Google LLC, Mountain View, CA, United States

SUDIP ROY, Google LLC, Mountain View, CA, United States

MARTIN A ZINKEVICH, Google LLC, Mountain View, CA, United States

PDF Download
3318464.3384707.pdf
26 January 2026
Total Citations: 34
Total Downloads: 2837

Published: 11 June 2020

Citation in BibTeX format

SIGMOD/PODS '20: International Conference on Management of Data
June 14 - 19, 2020
OR, Portland, USA

Conference Sponsors:
SIGMOD

TensorFlow Data Validation: Data Analysis and Validation in Continuous ML Pipelines

Emily Caveness, Paul Suganthan G. C., Zhuo Peng, Neoklis Polyzotis, Sudip Roy, Martin Zinkevich
Google Inc.

ABSTRACT

Machine Learning (ML) research has primarily focused on improving the accuracy and efficiency of the training algorithms while paying much less attention to the equally important problem of understanding, validating, and monitoring the data fed to ML. Irrespective of the ML algorithms used, data errors can adversely affect the quality of the generated model. This indicates that we need to adopt a data-centric approach to ML that treats data as a first-class citizen, on par with algorithms and infrastructure which are the typical building blocks of ML pipelines.

In this demonstration we showcase TensorFlow Data Validation (TFDV), a scalable data analysis and validation system for ML that we have developed at Google and recently open-sourced. This system is deployed in production as an integral part of TFX [5] – an end-to-end machine learning platform at Google. It is used by hundreds of product teams at Google and has received significant attention from the open-source community as well.

CCS CONCEPTS

• **Information systems → Data management systems**.

KEYWORDS

data management, machine learning

ACM Reference Format:
Emily Caveness, Paul Suganthan G. C., Zhuo Peng, Neoklis Polyzotis, Sudip Roy, Martin Zinkevich. 2020. TensorFlow Data Validation: Data Analysis and Validation in Continuous ML Pipelines. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data (SIGMOD'20), June 14–19, 2020, Portland, OR, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3318464.3384707

1 INTRODUCTION

Machine Learning (ML) crucially depends on the quality of the input data in order to produce a good model. In many cases, even simple errors in the data can affect significantly the resulting model. For example, consider a model that depends on an input data feature "Country" where the USA is represented with the string "US". However, if in the next batch of training data the value becomes "USA", then without any validation or pre-processing, the model will simply think that this is a new country.

Furthermore, it is often the case that the predictions from the generated models are logged and used to generate more data for training. Such feedback loops can amplify even "small" data errors and lead to gradual regression of model performance over a period of time. Hence, it is critical to catch data errors early, before they propagate through the complex loops and taint more of the pipeline's state.

Finally, error-free data is also critical for model understanding, since any attempt to debug and understand the output of the model or any quality metrics (e.g., AUC, precision, ...) is futile if the evaluation data has errors. All these observations indicate that we need to treat data as a first-class citizen in ML pipelines, on par with algorithms and infrastructure, with corresponding tooling to analyze, validate and monitor the data throughout the various stages of the pipeline.

Data validation is neither a new problem nor unique to ML, and so we can leverage techniques and principles from the field of data management. However, we argue that the problem acquires unique characteristics in the context of ML and hence we need to rethink existing solutions. First, we need a way to express ML-related constraints and expectations on the quality of the data. Second, the data validation system must generate reliable alerts with high precision, and provide enough context for the human to quickly identify the root cause of the problem. This is due to the fact that in most cases data errors cannot be fixed automatically – they require some human intervention, either in the ML pipeline (e.g., rolling back the trainer to a checkpoint unaffected by the suspect data) or in the data-generation code (e.g., fixing the bugs that cause the errors). Third, the system needs to scale to production pipelines which typically process billions to trillions of examples. Finally, the system needs to account for the fact that data is stored and managed externally from the ML pipeline, and often in a variety of storage systems, and hence a-priori knowledge about the data and its semantics is limited.

To address the above challenges in the context of Google's production ML pipelines, we developed TensorFlow Data Validation (TFDV), a scalable data analysis and validation system for ML. Our system is deployed in production as an integral part of TFX [5], an end-to-end ML platform, and is used by hundreds of product teams at Google to monitor and validate trillions of training and serving examples per day, amounting to several petabytes of data per day. We recently open sourced TFDV and the system has received significant attention from the open-source community as well: 2.6M downloads since the first release in October 2018, plus it has influenced the development of other open-source data validation systems such as Apache Spark Data Validation [1]. Furthermore, TFDV has been adopted by other large organizations using ML, e.g., see Spotify's keynote at TensorFlow World 2019 about using TFDV [2]. TFDV is described in depth in [6]. So in this paper we will describe it only briefly, focusing instead on the demonstration scenarios.

2 RELATED WORK

Few recent works ([10], [9]) have considered the importance of data validation for ML applications. For example, Amazon (Schelter et. al. [10]) proposed a system for automating data quality verification task that provides a declarative API to specify common quality constraints and custom validation code. While Amazon's system allows users to express arbitrary constraints, we opted to have a restrictive schema definition language that captures the data constraints for most of our users in order to focus on reliable high precision alerts. Further our work emphasizes on the co-evolution of the schema with data and the model with the user in the loop. As far as we can tell, TFDV is the first open-source data analysis and validation system for ML.

Numerous works from data cleaning literature [7] focus on automatically detecting and fixing errors in the data. However, these works have limited applicability in the production-ML scenarios that we target, for several reasons: (a) the data comes from external systems where in-place updates are not feasible, (b) any cleaning has to be applied consistently both on the training and the serving path, where the latter has stringent latency requirements, (c) data errors typically correspond to code bugs and the preferred fix is to eliminate the bug rather than "patch" the data.

[1]https://databricks.com/session/apache-spark-data-validation
[2]https://www.youtube.com/watch?v=zxd3Q2gdArY&t=748s

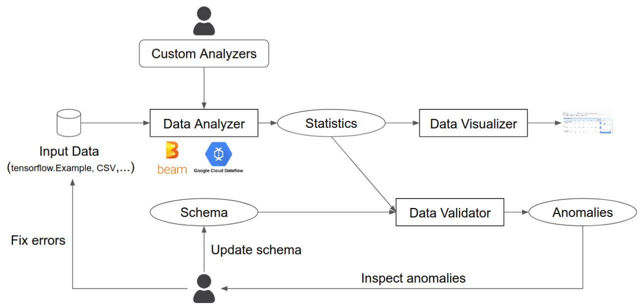


Figure 1: TensorFlow Data Validation Architecture.

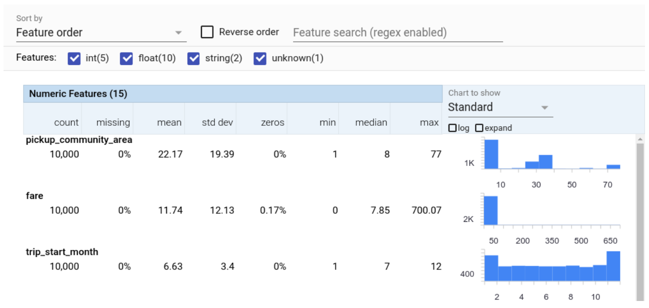
3 SYSTEM OVERVIEW

Figure 1 shows the TFDV architecture which consists of a Data Analyzer component which computes statistics in a scalable fashion over large amounts of data, a Data Validator which finds anomalies in the data, and a Data Visualizer which provides visualizations of the statistics, schema, and the anomalies.

Data Analyzer. Data Analyzer takes a collection of statistics generators and computes data statistics needed for validation. TFDV uses Apache Beam [1] to define and process its data pipelines. The statistics generators are implemented as Beam transforms. Users can provide custom statistics generators which are executed together with the default generators. The generator API takes Apache Arrow Tables as input, as it is powerful enough to encode popular logical training data formats: flat (tensorflow.Example, CSV), sequence (tensorflow.SequenceExample) or structured data (e.g. Protocol Buffers or Apache Avro). TFDV provides decoders for popular data formats like tensorflow.Example, and CSV. Users can write custom decoders (that convert their input to Arrow Tables) to handle arbitrary data formats. Realizing the data pipelines using Beam allows TFDV to transparently run the pipeline in different environments such as a single machine, Flink/Spark cluster, and Google Cloud Dataflow.

The statistics computed by TFDV include individual feature statistics depending on the type of the feature (e.g., statistics such as min, max, mean, median, histogram etc. for numeric features, statistics such as number of unique values, top-k values, average length etc. for categorical features.) and cross-feature statistics (e.g., correlation between features and mutual information of a feature with the label etc.). We represent the statistics as a protocol buffer message (See [4] for the complete list of statistics computed by TFDV.).

Data Validator. The Data Validator checks the properties of the data as specified through a schema. Typically we expect the data characteristics to remain stable across different splits of data (e.g., training data and evaluation data) or batches of data that are close in time. Hence, we consider any deviation



Numeric Features (15)

	count	missing	mean	std dev	zeros	min	median	max
pickup_community_area	10,000	0%	22.17	19.39	0%	1	8	77
fare	10,000	0%	11.74	12.13	0.17%	0	7.85	700.07
trip_start_month	10,000	0%	6.63	3.4	0%	1	7	12

Figure 2: Understanding training data.

within a batch (or split) from the expected data characteristics as an *anomaly*.

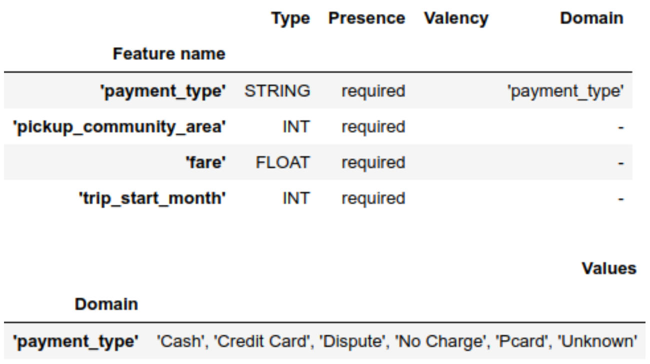
TFDV adapts the "battle-tested" principles from data management systems to the context of ML. Specifically, in order to codify these expected data characteristics, TFDV generalizes the traditional notion of a *schema* from database systems. The schema follows a logical data model where each training or serving example is a collection of features, with each feature having several constraints attached to it. This flat model has an obvious mapping to the flat data formats such as tensorflow.Example or CSV. The constraints associated with each feature cover some basic properties (e.g., type, domain, valency) but also constraints that are relevant to ML (See [6] for a more detailed discussion of our schema formalism.). Using a schema also allows us to verify any assumptions of training/serving code (e.g., the schema can be used to generate fuzzy examples and verify if the training/serving code crashes on those examples) and thereby catch potential model crashes early on. We represent the schema as a protocol buffer message.

TFDV supports two types of validation: (1) validating a single batch of data against the schema, and (2) validating two batches of data (e.g., are there any significant changes between training and serving data, or between successive batches of the training data?). Any disagreement found during validation is flagged as an anomaly for human inspection and further investigation. See [3] for the complete list of 52 anomalies identified by TFDV and the conditions on which each anomaly is raised.

Data Visualizer. TFDV provides visualizations for the statistics, schema and the anomalies. It provides a simple table-based view for the schema and the anomalies. It uses the Facets library [2] to visualize the statistics. Specifically, TFDV supports (1) visualizing the statistics of a batch of data, and (2) comparing statistics between batches of data.

4 DEMONSTRATION OVERVIEW

We now describe the proposed demonstration. We use real-world data sets to show the following features of TFDV: (1) how users can understand the data set by computing descriptive statistics and visualizing the result, (2) how users can infer a schema for the data which captures the properties of the data, (3) how users can validate new data against the schema and inspect anomalies in the data, (4) how users can perform skew/drift detection using TFDV, and (5) how users can easily extend TFDV with code to compute custom statistics which can power applications like feature engineering. Throughout the demonstration, we will assume that the data set is divided into three splits as in a typical ML workflow: training, evaluation and serving data.



Feature name	Type	Presence	Valency	Domain
'payment_type'	STRING	required		'payment_type'
'pickup_community_area'	INT	required		-
'fare'	FLOAT	required		-
'trip_start_month'	INT	required		-

Domain	Values
'payment_type'	'Cash', 'Credit Card', 'Dispute', 'No Charge', 'Pcard', 'Unknown'

Figure 3: Schema.

4.1 Understanding the training data

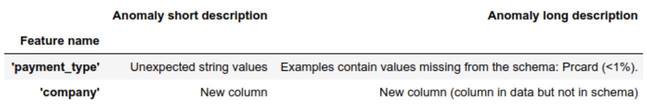
The user will begin by understanding the training data using TFDV by computing descriptive statistics over the data and visualizing the statistics within the notebook. TFDV uses Facets library to create the visualization which displays the charts showing the distribution of the individual features and various statistics. The user will look for potential errors in the data (e.g., a numeric feature having high number of missing or zero values). Figure 2 shows the visualization.

Next, the user will drill down into the data to understand the specific slices of the data. For example, the user may want to look at how the distribution of a feature (i.e., say tips) varies across areas. In order to do this, the user will use TFDV to compute per-slice statistics (where each slice corresponds to an area). Then the user can visualize statistics per slice or compare statistics between two slices (see Figure 5).

In addition, we will demonstrate the ability of TFDV's statistics computation pipeline to execute transparently in different execution environments such as a single machine and Google Cloud Dataflow.

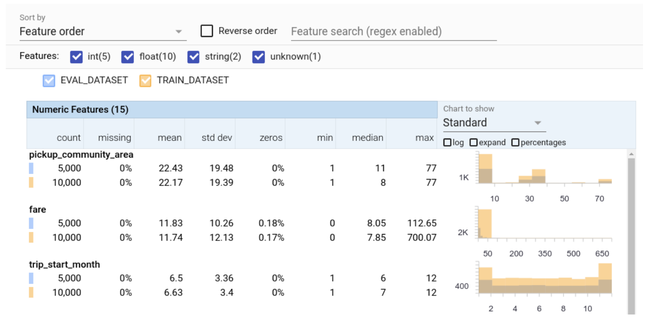
4.2 Inferring a schema

As described in Section 3, TFDV uses a schema to capture the stable data characteristics. TFDV employs a set of reasonable



Feature name	Anomaly short description	Anomaly long description
'payment_type'	Unexpected string values	Examples contain values missing from the schema: Prcard (<1%).
'company'	New column	New column (column in data but not in schema)

Figure 4: Anomalies.



Sort by: Feature order ☐ Reverse order Feature search (regex enabled)

Features: ✓ int(5) ✓ float(10) ✓ string(2) ✓ unknown(1)

✓ EVAL_DATASET ✓ TRAIN_DATASET

Numeric Features (15)

	count	missing	mean	std dev	zeros	min	median	max
pickup_community_area								
🟦	5,000	0%	22.43	19.48	0%	1	11	77
🟧	10,000	0%	22.17	19.39	0%	1	8	77
fare								
🟦	5,000	0%	11.83	10.26	0.18%	0	8.05	112.65
🟧	10,000	0%	11.74	12.13	0.17%	0	7.85	700.07
trip_start_month								
🟦	5,000	0%	6.5	3.36	0%	1	6	12
🟧	10,000	0%	6.63	3.4	0%	1	7	12

Figure 5: Comparing training and evaluation data.

heuristics to auto-generate the initial schema. In this step, the user will use TFDV to infer an initial schema based on the training data. Then the user will inspect the inferred schema and modify any properties if needed. Figure 3 shows a sample schema. The schema is represented as a protocol buffer and can be modified through its Python API.

4.3 Validating the evaluation data

So far the user has only looked at the training data. Next, the user will validate if the evaluation data is consistent with the training data. This involves four steps: (1) computing statistics over the evaluation data, (2) comparing evaluation data statistics with training data statistics (see Figure 5 which overlays the statistics of the training data and the evaluation data.), (3) validating the evaluation data against the expectations set in the schema, and (4) examining the anomalies and taking action to either fix errors in the evaluation data or update the schema if the detected anomaly is a natural evolution of the data (e.g. a new valid string value in a categorical feature). The anomalies can be examined in the notebook as shown in Figure 4 which displays a list of anomalies for each feature. For example, the first anomaly indicates that the evaluation data has an out-of-domain value (i.e., not specified in the schema) for the feature *payment_type*. If this is expected, the user can update the schema. TFDV also provides a way to automatically fix the schema based on the statistics. We will allow the user to modify the various schema properties and explore the different anomalies raised by TFDV.

4.4 Performing skew detection

Training/Serving skew refers to a difference in the feature values or distributions between the data used to train a model and the data observed by the serving system. Here, the user will check if there are significant changes between training and serving data. Specifically, the user will first compute statistics over the serving data. Then the user will visualize side-by-side the statistics of training and serving data to look for potential errors or drifts. Next the user will perform skew detection between the training and serving data to identify features whose distribution over training data is different from that seen over serving data.

TFDV measures skew using L_∞ distance ($d_\infty = max_{i \in S} |p_i - q_i|$) which compares two distributions p and q with probabilities p_i and q_i respectively for each value i (from some universe). Skew detection can be configured by specifying a distance threshold in the schema (i.e., an anomaly is raised if the distribution distance of a feature value between training and serving data is above the threshold.). See [6] to find about how we estimate the distance in a statistically sound manner based on observed frequencies of the feature values.

4.5 Extending TFDV

In this part of the demo, we will show that TFDV can be extended easily to incorporate additional functionalities. Specifically, we will show that users will be able to easily compute custom statistics in TFDV that can detect semantic types (e.g., NLP, image, date/time etc.) of the features in the data, which is crucial for various data science tasks such as data cleaning, automated feature engineering, and schema matching [8]. Concretely, the user will run TFDV with additional statistics generators that compute statistics to validate if a feature belongs to a natural language, image or time domain. Then the user will infer a schema based on the computed statistics to infer semantic types for features.

REFERENCES

[1] [n.d.]. Apache Beam. https://beam.apache.org/
[2] [n.d.]. Facets. https://pair-code.github.io/facets/
[3] [n.d.]. TensorFlow Data Validation. https://github.com/tensorflow/data-validation
[4] [n.d.]. TensorFlow Metadata. https://github.com/tensorflow/metadata
[5] Denis Baylor, Eric Breck, Heng-Tze Cheng, et al. 2017. TFX: A TensorFlow-Based Production-Scale Machine Learning Platform. In *ACM SIGKDD*.
[6] Eric Breck, Marty Zinkevich, Neoklis Polyzotis, Steven Whang, and Sudip Roy. 2019. Data Validation for Machine Learning. In *SysML*.
[7] Xu Chu, Ihab F. Ilyas, Sanjay Krishnan, and Jiannan Wang. 2016. Data Cleaning: Overview and Emerging Challenges. In *SIGMOD*.
[8] Madelon Hulsebos, Kevin Hu, Michiel Bakker, et al. 2019. Sherlock: A Deep Learning Approach to Semantic Data Type Detection. In *ACM SIGKDD*. 1500–1508.
[9] Stephan Rabanser, Stephan Günnemann, and Zachary Lipton. 2019. Failing Loudly: An Empirical Study of Methods for Detecting Dataset Shift. In *Advances in Neural Information Processing Systems 32*.
[10] Sebastian Schelter, Dustin Lange, Philipp Schmidt, et al. 2018. Automating Large-Scale Data Quality Verification. *VLDB* 11, 12 (2018), 1781–1794.