# DATA VALIDATION FOR MACHINE LEARNING

Pegah Rahimian

# Why data validation in ML?

- **Motivation:** Errors in the input data can nullify any benefits on speed and accuracy for training and inference.
- **Goal:** Present a data validation system that is designed to detect anomalies specifically in data fed into machine learning pipelines.
- **Contribution:** Early detection of errors, model-quality wins from using better data, savings in engineering hours to debug problems, and a shift towards data-centric workflows in model development.

# Challenges

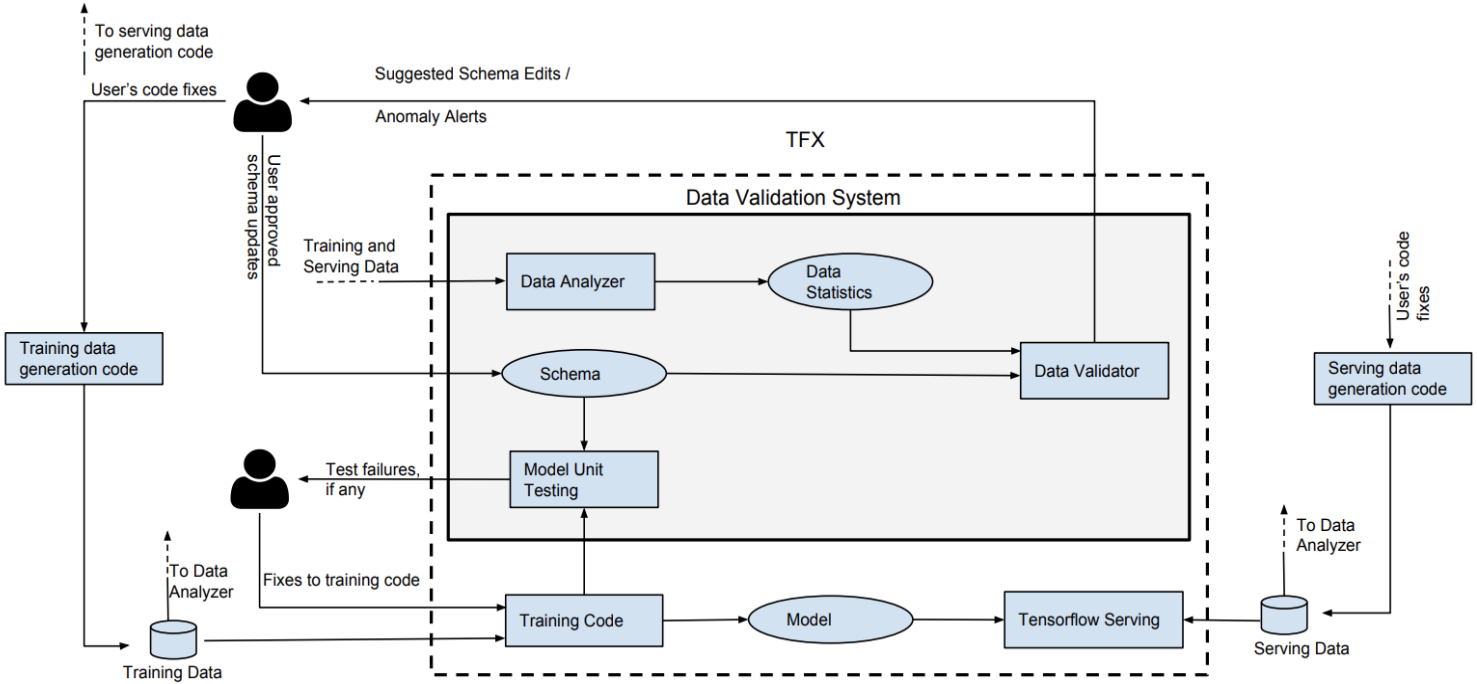
- In the ML systems, the root of error can be due to the following factors

1. the training data,
2. or training code

Sometimes, the error is valid for the taraining code, thus not suitable for training data and modelling. Thats the point where automatic debugging will fail.

# System validation overview

- A pipeline that ingests training data, passes it to data validation, then pipes it to a training algorithm that generates a model, and finally pushes the latter to a serving infrastructure for inference.



- each batch of input data will trigger a new run of the data-validation logic and hence potentially a new set of data anomalies.

- by validating over the entire batch we ensure that anomalies that are infrequent or manifest in small but important slices of data are not silently ignored.

# Data validation components

**Data analyzer**

- computes predefined set of data statistics sufficient for data validation

**Data validator**

- checks for properties of data as specified through a schema

**Model unit tester**

- checks for errors in the training code using synthetic data generated through the schema

# Data validation types

### 1. Single-batch validation

- answers the question: are there any anomalies in a single batch of data? The goal here is to alert the on-call about the error and kick-start the investigation

### 2. Inter-batch validation

- answers the question: are there any significant changes between the training and serving data, or between successive batches of the training data?

### 3. Model testing

- answers the question: are there any assumptions in the training code that are not reflected in the data.
- (e.g. are we taking the logarithm of a feature that turns out to be a negative number or a string?).

# 1. Single batch validation

- The first question we answer is: are there data errors within each new batch that is ingested by the pipeline?
- We expect the data characteristics to remain stable within each batch, as the latter corresponds to a single run of the data-generation code.
- given expert domain knowledge, as an anomaly.

# Protocol buffer message

```
message Schema {
  // Features described in this schema.
  repeated Feature feature;

  // String domains referenced in the features.
  repeated StringDomain string_domain;
}

message Feature {
  // The name of the feature.
  string name;

  // Type of the feature's values
  FeatureType type;

  // Constraints on the number of examples that have this
  // feature populated.
  FeaturePresence presence;

  // Minimum and maximum number of values.
  ValueCount value_count;

  // Domain for the values of the feature.
  oneof domain_info {
    // Reference to a domain defined at the schema
    // level.
    string domain;

    // Inline definitions of domains.
    IntDomain int_domain;
    FloatDomain float_domain;
    StringDomain string_domain;
    BoolDomain bool_domain;
  }

  LifecycleStage lifecycle_stage;
}
```

- The schema attempts to capture some of the semantics that are lost when the data are transformed to the format accepted by training algorithms, which is typically key-value lists. To illustrate this, consider a training example with a key-value (“age”, 150). If this feature corresponds to the age of a person in years then clearly there is an error. If, however, the feature corresponds to the age of a document in days then the value can be perfectly valid. The schema can codify our expectations for “age” and provide a reliable mechanism to check for errors

# Schema validation

- The Data Validator component of the system validates each batch of data by comparing it against the schema. Any disagreement is flagged as an anomaly and triggers an alert to the on-call for further investigation

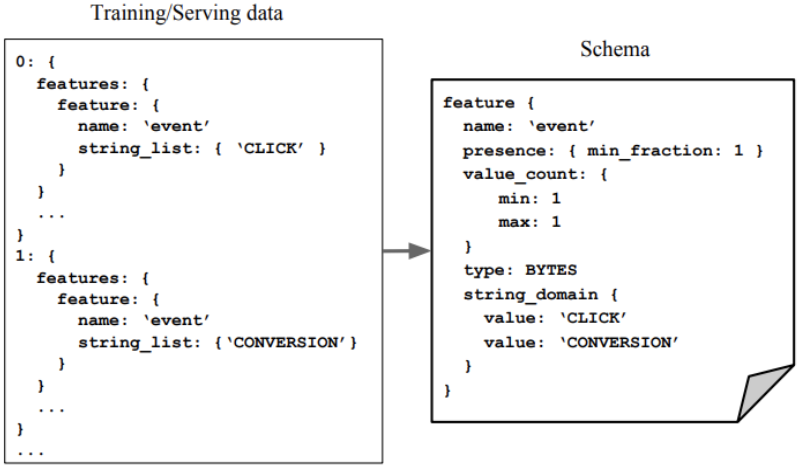



Figure 3: An example schema and corresponding data in the `tf.train.Example` (tfe) format.

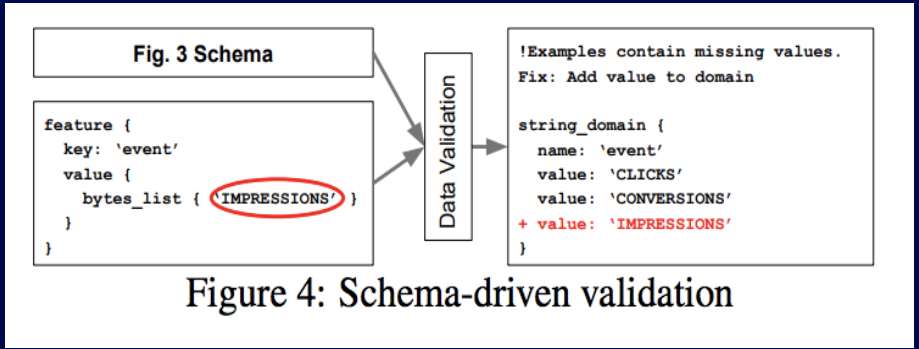



Figure 4: Schema-driven validation

# MODEL UNIT TESTING

- Up to this point, the model focused on detecting mismatches between the expected and the actual state of the data.
- The intent has been to uncover software errors in generating either training or serving data
- Danger!!!!!:  the training code can make several other types of assumptions that are not reflected in the schema,

# Model unit testing

- As we explain below, some computations may make assumptions that do not agree with the data and cause serious errors that propagate through the ML infrastructure.

- suppose that the training code applies a logarithm transform on a numeric feature, thus making the implicit assumption that the feature's value is always positive.
- Now, if the schema does not encode this assumption (e.g., the feature's domain includes non-positive values) and also that the specific instance of training data happens to carry positive values for this feature. As a result, two things happen:

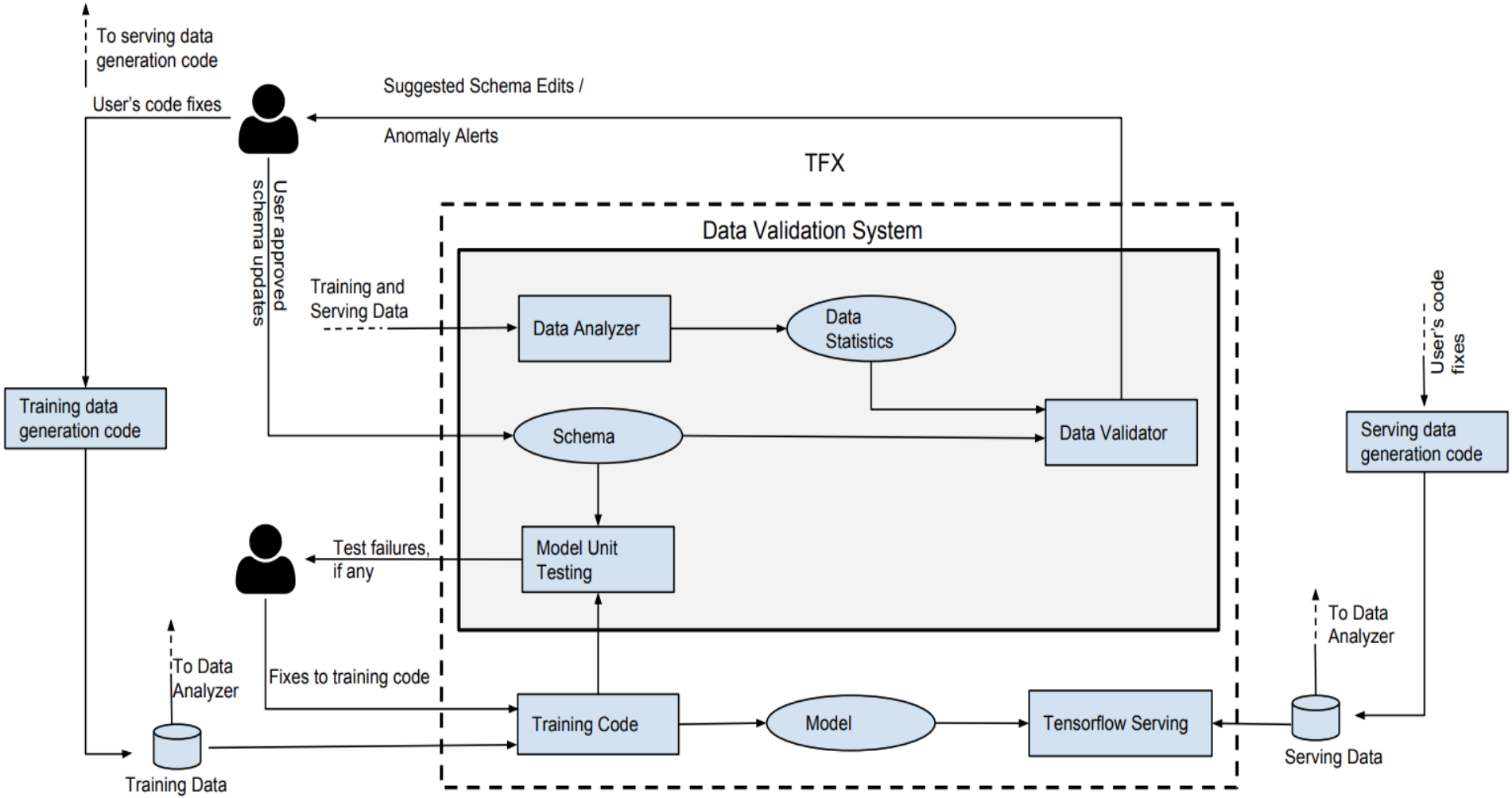
1. data validation does not detect any errors,
2. the generated model includes the same transform and hence the same assumption

# How to solve it? : Fuzz testing strategy

- Returning to our earlier example with the logarithm transform, a synthetic input with non-positive feature values will cause an error and hence uncover the mismatch between the schema and the training code.

1. At that point, the user can fix the training code, e.g., apply the transform on **max(value, 1),**
2. or extend the schema to mark the feature as positive (so that data validation catches this error).

- Doing both provides for additional robustness, as it enables alerts if the data violates the stated assumptions and protects the generated model from crashes if the data is wrong.


To serving data generation code
User's code fixes
Suggested Schema Edits /
Anomaly Alerts
TFX
User approved schema updates
Data Validation System
Training and Serving Data
Data Analyzer
Data Statistics
Training data generation code
Schema
Data Validator
User's code fixes
Serving data generation code
Test failures, if any
Model Unit Testing
To Data Analyzer
Fixes to training code
To Data Analyzer
Training Code
Model
Tensorflow Serving
Training Data
Serving Data

# Thank You