Data validation and management

SOEN 691: Engineering AI-based Software Systems

Emad Shihab, Diego Elias Costa
Concordia University

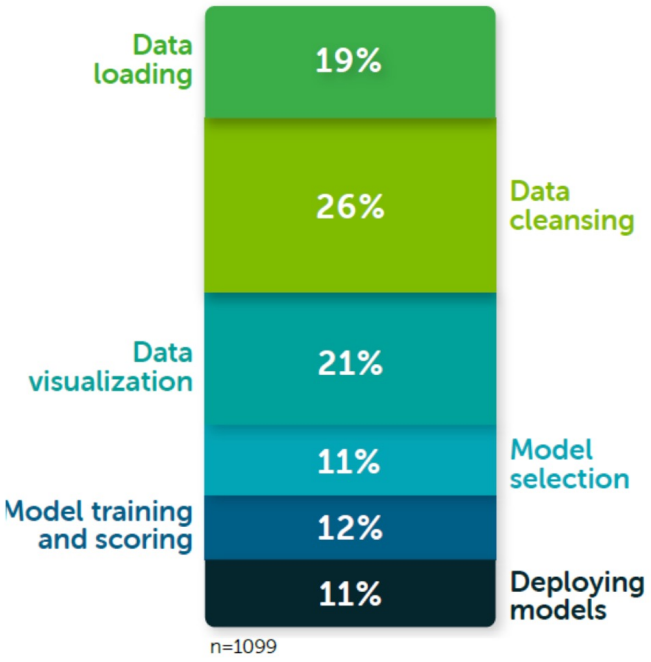
G I G O

Garbage In Garbage Out

Data quality is paramount for ML quality

- A machine learning model is only as good as the data it is fed

Data scientists spent considerably more time **ensuring data quality** then working with the models



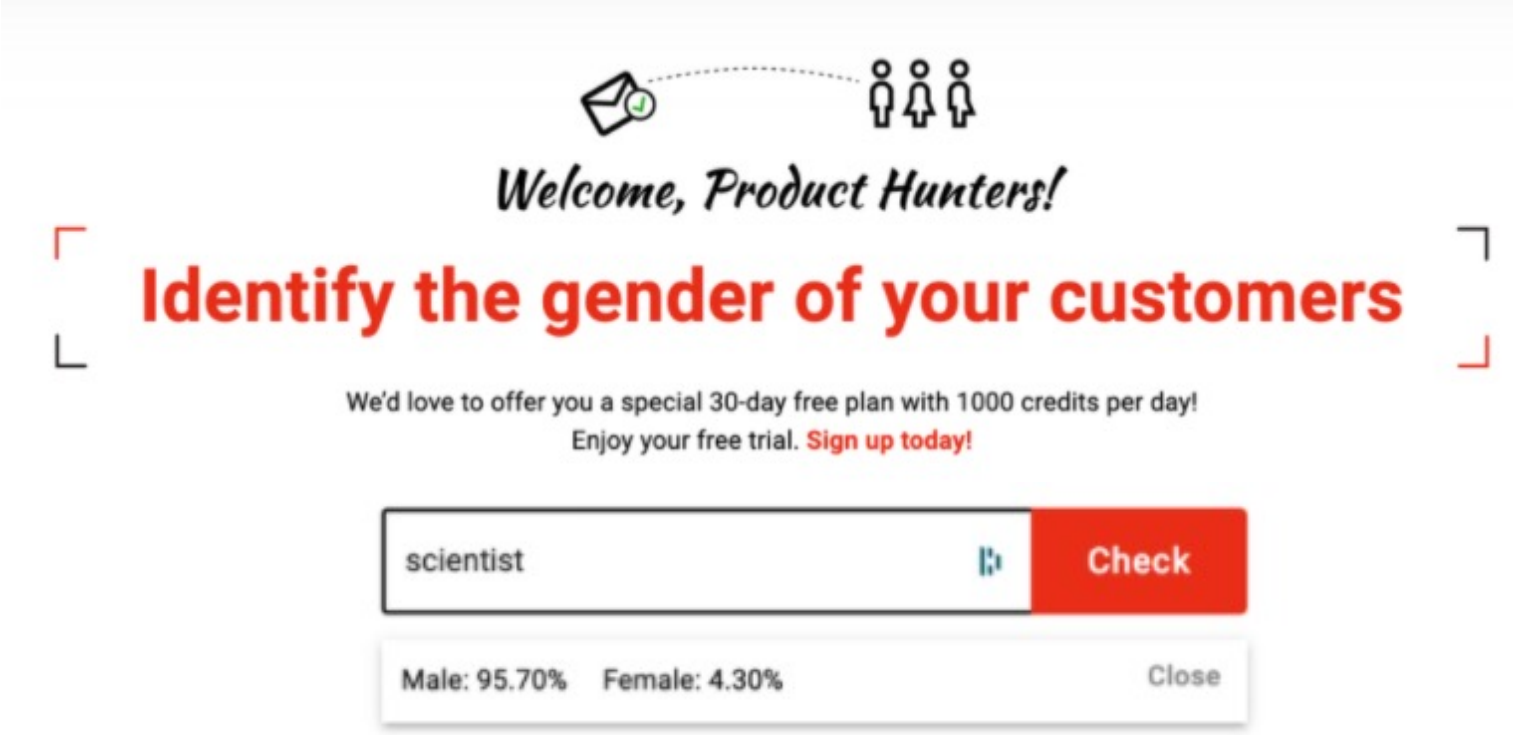
How data scientists spend their time (Image courtesy Anaconda "2020 State of Data Science: Moving From Hype Toward Maturity.")

Bad Data Behavior is Silent

- Typical software errors are identified via erractic behavior
  - Stack traces, error logs, etc.
- Bad Data best case scenario:
  - Errors in the processing pipeline
- Bad Data typical scenario:
  - ???

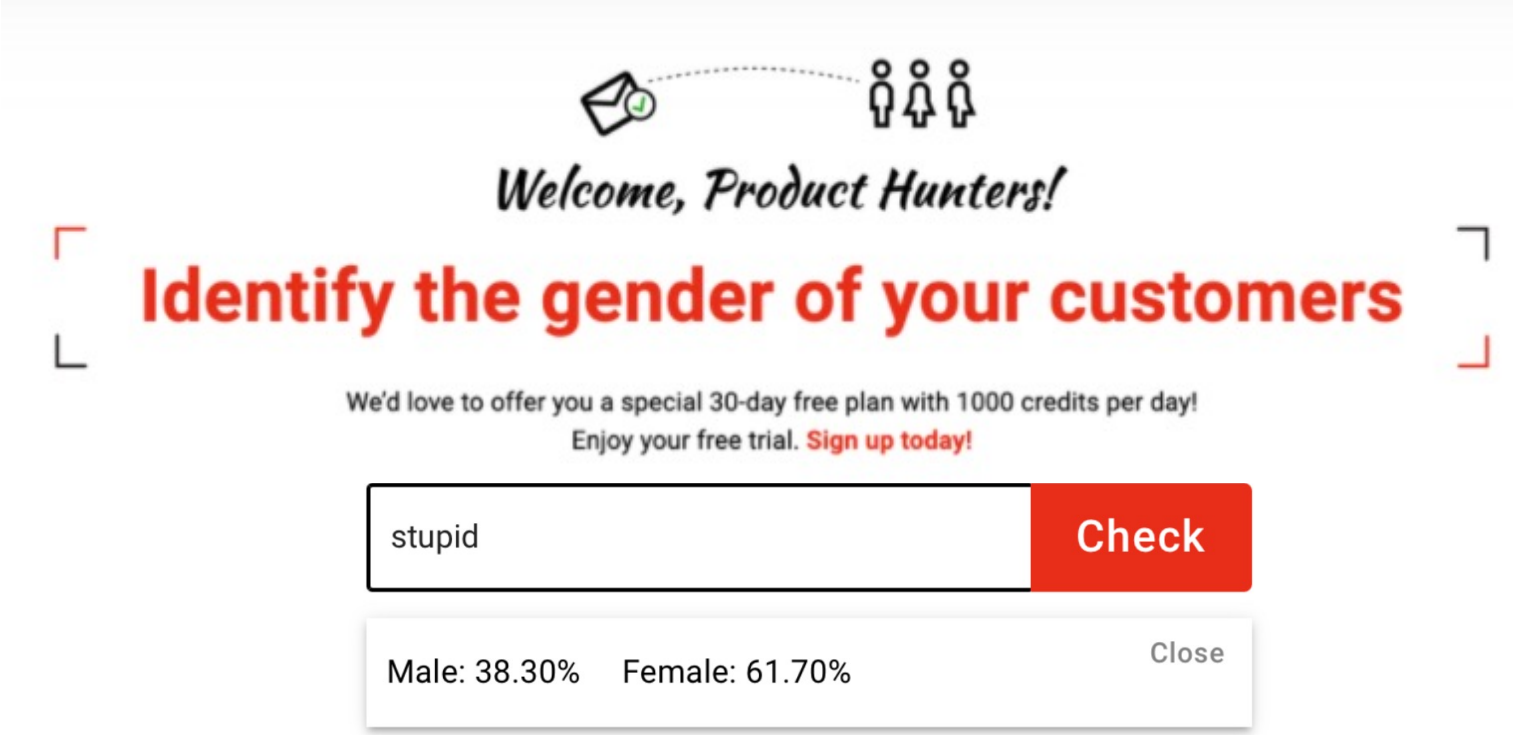
Bad data can be worse than no data

- The case of Genderify



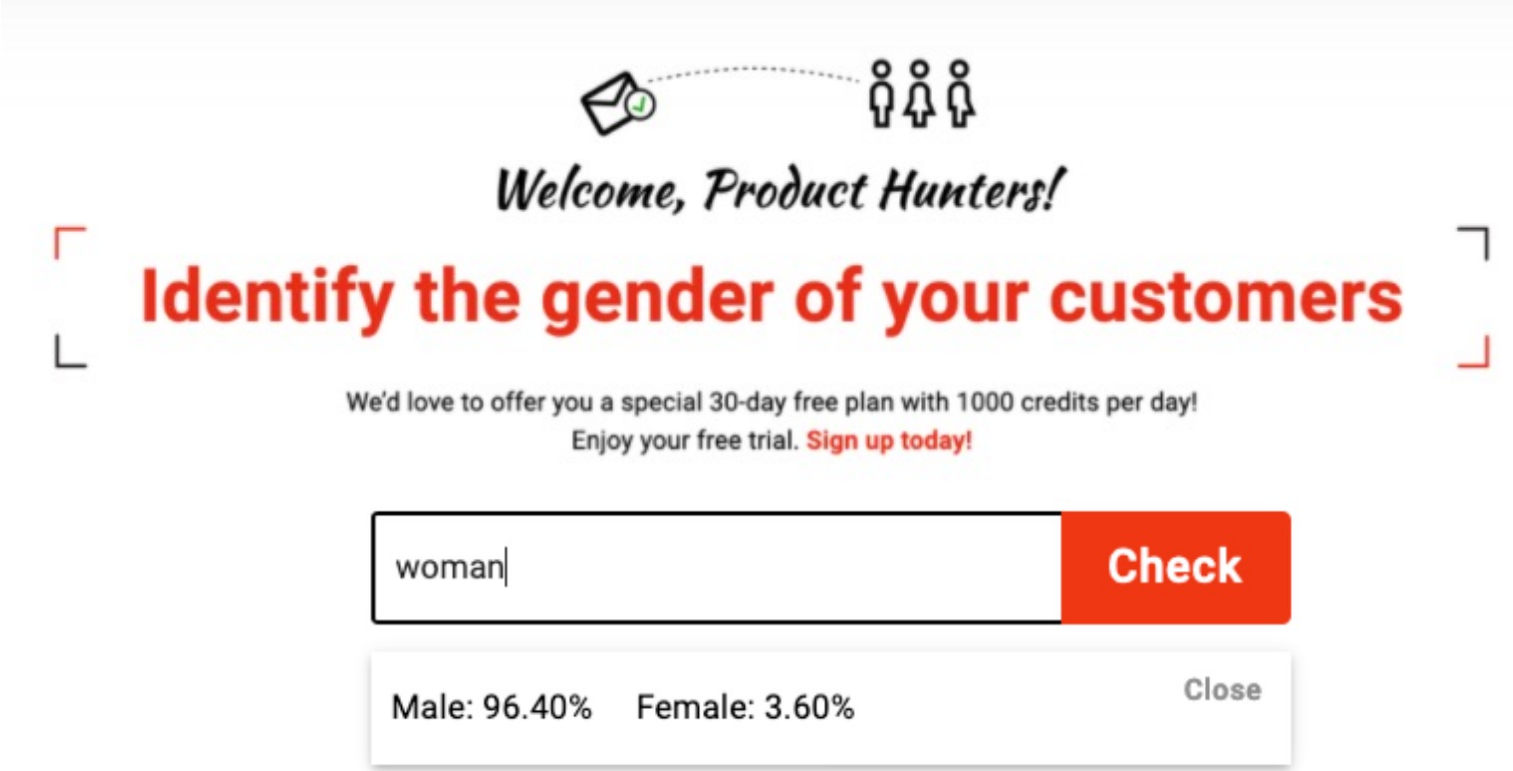
Bad data can be worse than no data

- The case of Genderify



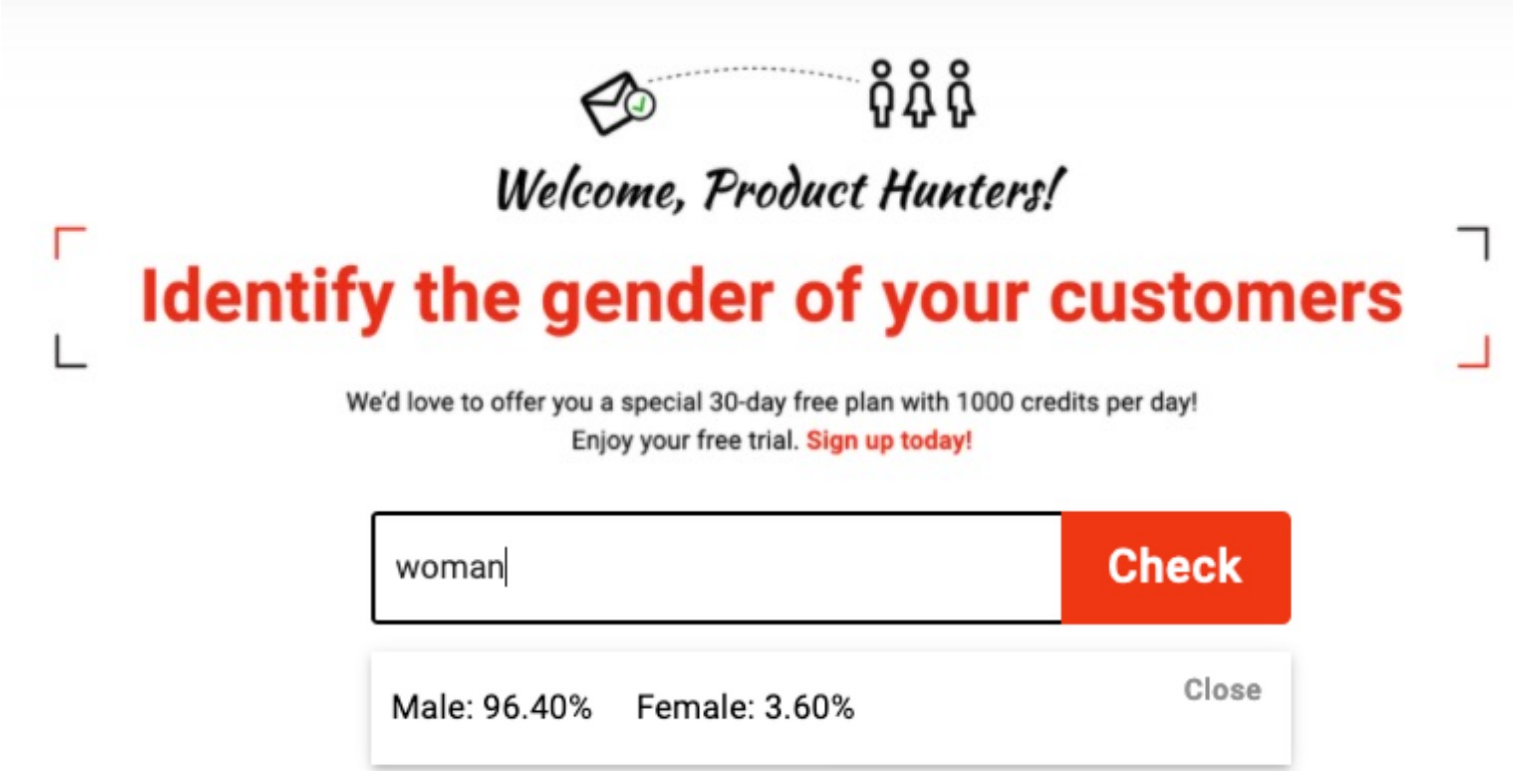
Bad data can be worse than no data

- The case of Genderify



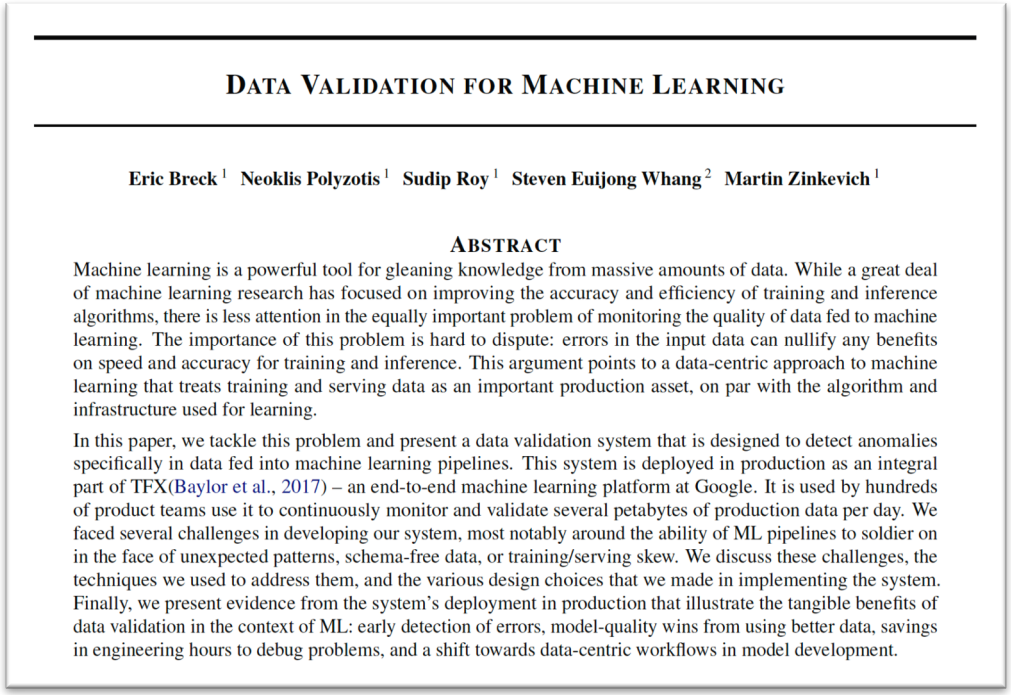
Bad data can be worse than no data

- The case of Genderify



Data Validation for Machine Learning

Presents a
data validation system
to detect anomalies
in ML pipelines



DATA VALIDATION FOR MACHINE LEARNING

Eric Breck [1] Neoklis Polyzotis [1] Sudip Roy [1] Steven Euijong Whang [2] Martin Zinkevich [1]

ABSTRACT

Machine learning is a powerful tool for gleaning knowledge from massive amounts of data. While a great deal of machine learning research has focused on improving the accuracy and efficiency of training and inference algorithms, there is less attention in the equally important problem of monitoring the quality of data fed to machine learning. The importance of this problem is hard to dispute: errors in the input data can nullify any benefits on speed and accuracy for training and inference. This argument points to a data-centric approach to machine learning that treats training and serving data as an important production asset, on par with the algorithm and infrastructure used for learning.

In this paper, we tackle this problem and present a data validation system that is designed to detect anomalies specifically in data fed into machine learning pipelines. This system is deployed in production as an integral part of TFX(Baylor et al., 2017) – an end-to-end machine learning platform at Google. It is used by hundreds of product teams use it to continuously monitor and validate several petabytes of production data per day. We faced several challenges in developing our system, most notably around the ability of ML pipelines to soldier on in the face of unexpected patterns, schema-free data, or training/serving skew. We discuss these challenges, the techniques we used to address them, and the various design choices that we made in implementing the system. Finally, we present evidence from the system's deployment in production that illustrate the tangible benefits of data validation in the context of ML: 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.

Motivation

- Machine Learning is a powerful tool to gleaning knowledge from massive amounts of data.

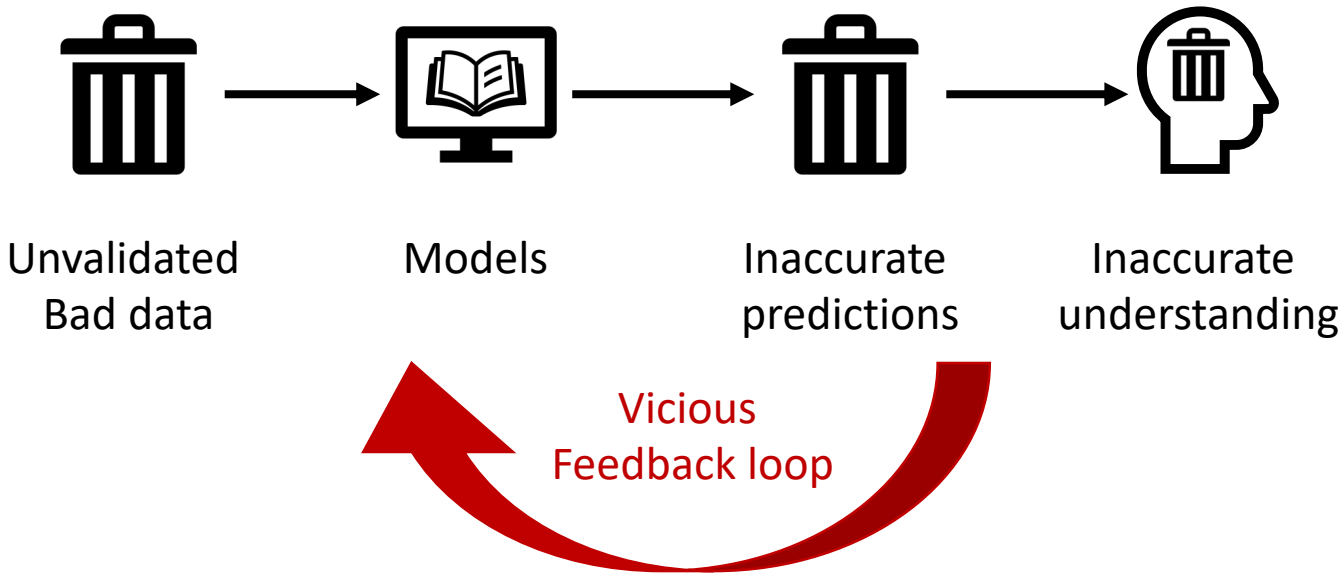
- Researchers have been focusing on
  - Efficiency of training
  - Better models

- There is a need to propose better Approaches to monitoring the quality of data

Scope of the paper

- Focus on the problem of validating the input data of ML pipelines

- Why?



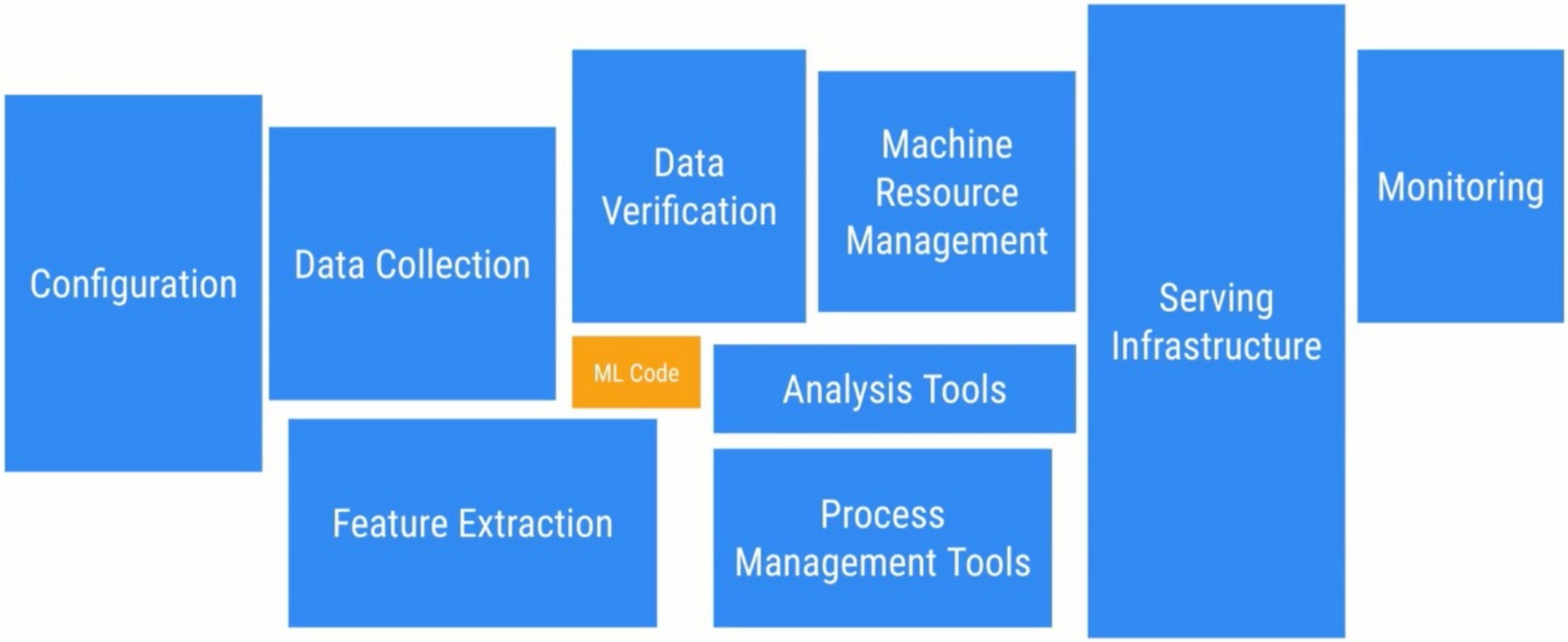
Scope of the paper

- Focus on the problem of validating the input data of ML pipelines

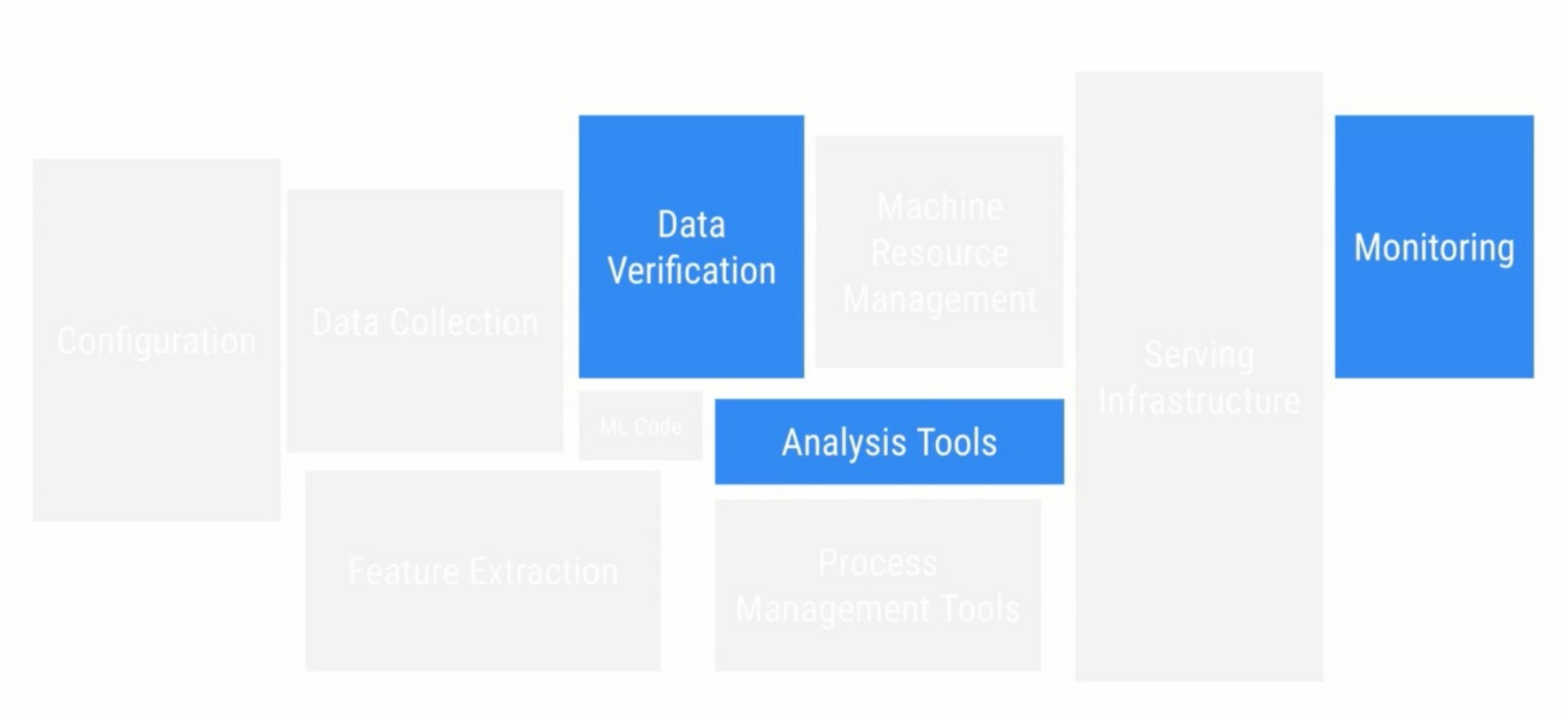
- Why?
  - Small errors in data, if unchecked, can be **amplified!**

- What to do?
  - Attempt to catch data errors early in the ML process

Scope within Machine Learning Systems



Scope within Machine Learning Systems



Contributions

- Authors propose a data validation system
  - Battle tested in industrial environment
  - Deployed as part of TFX

- Libraries are publicly available
  - https://github.com/tensorflow/data-validation

Study Design

- System's paper
- Industrial report
- Focus on providing the approach's overview
- Empirical evaluation
  - Production deployment
  - Case studies

ML Terminology

- Training data
- Serving data
- Model inference

- Training data generation code
- Serving data generation code

System's Overview

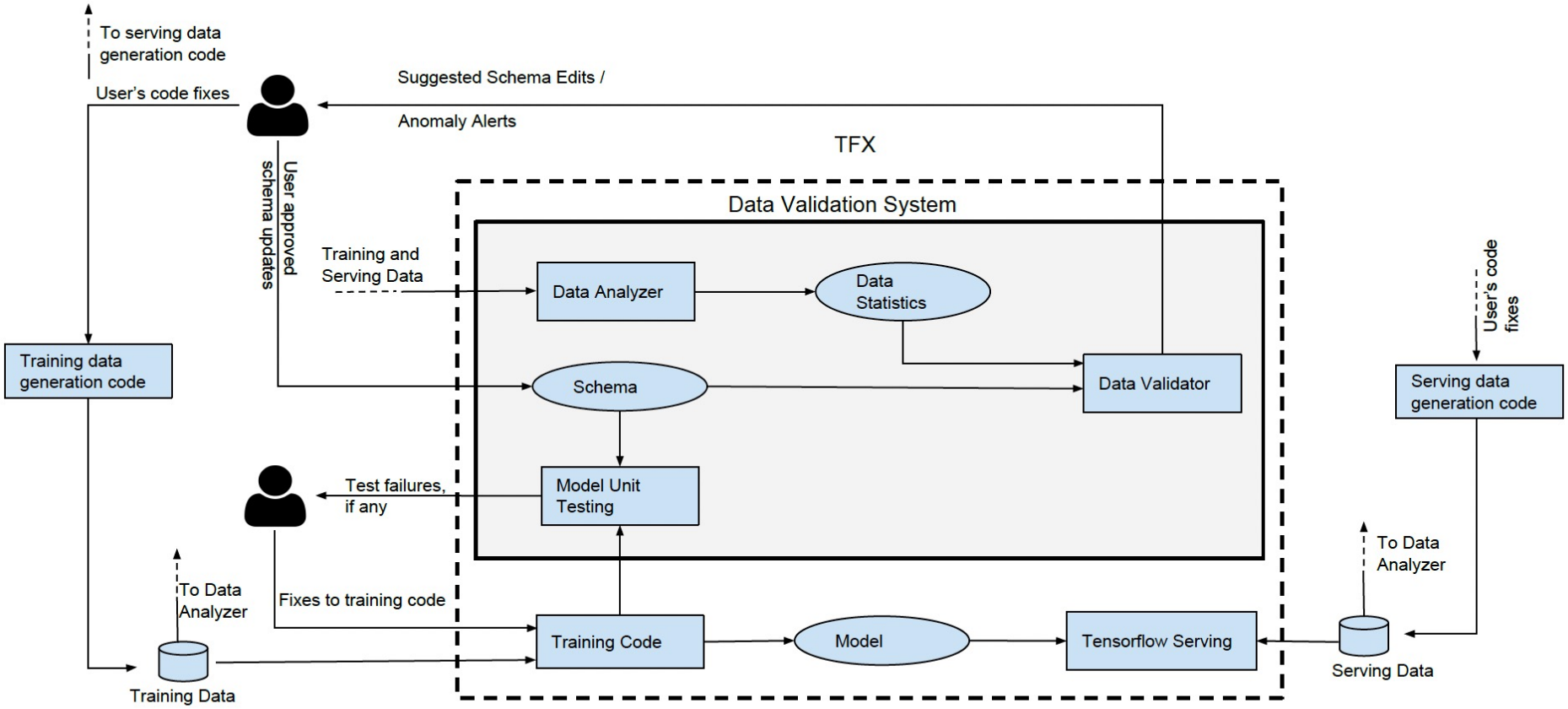


Figure 1: An overview of our data validation system and its integration with TFX.

System's Overview

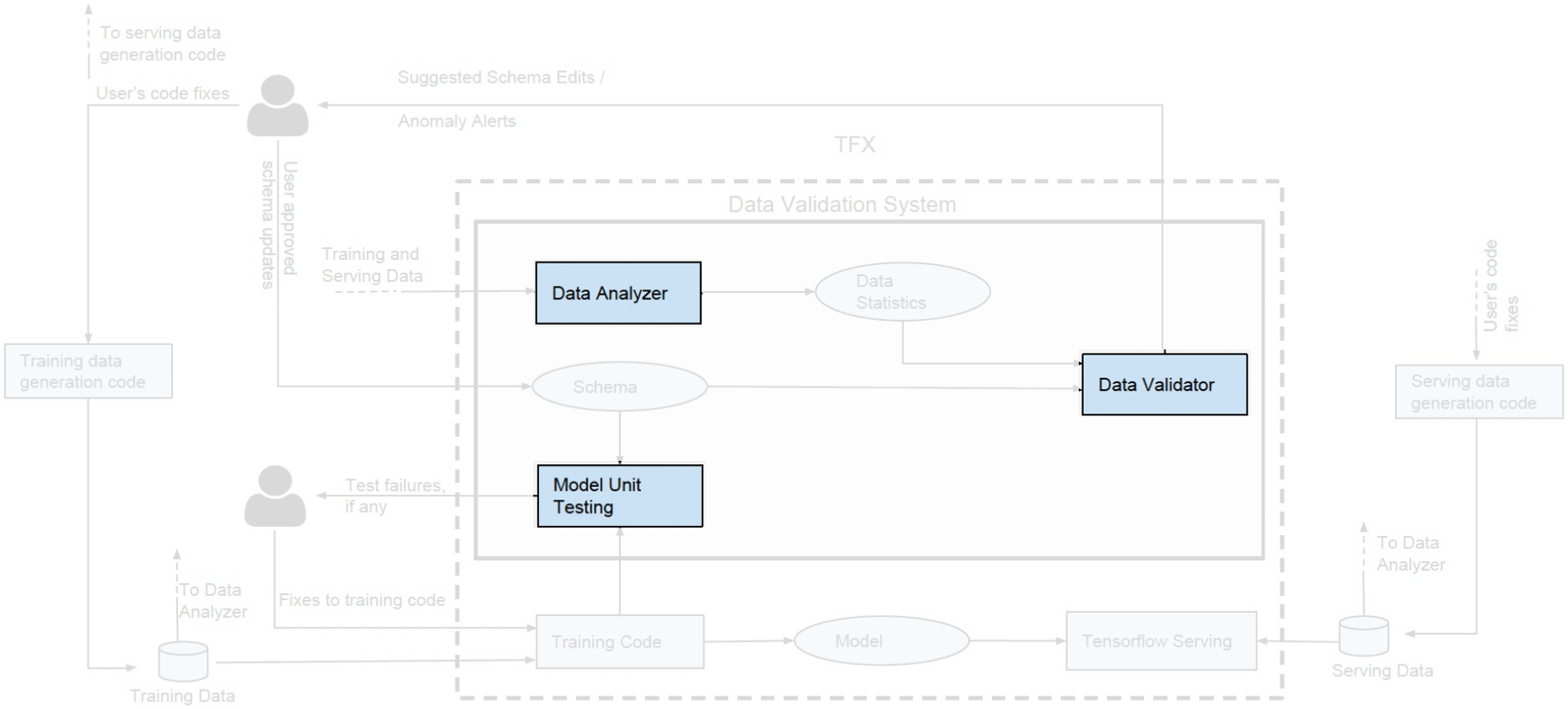
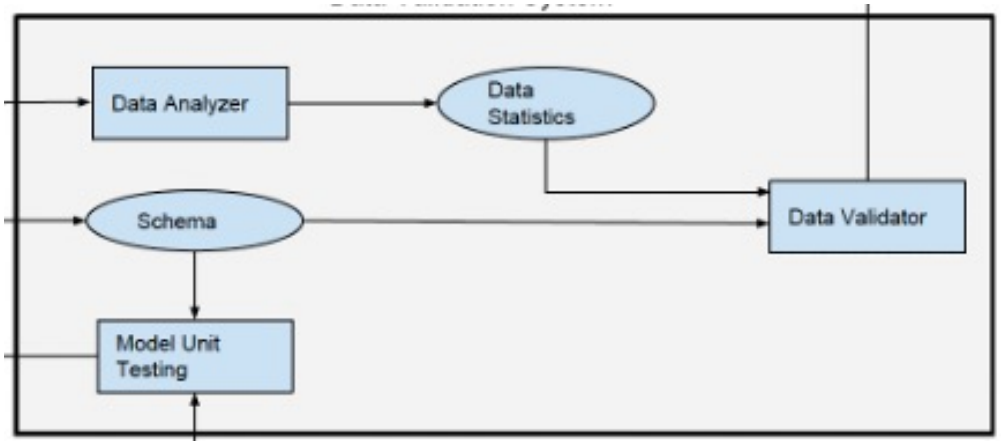


Figure 1: An overview of our data validation system and its integration with TFX.

Components



- Data Analyzer
  - Computes data statistics for data validation
- Data Validator
  - Checks for properties of data
- Model Unit Tester
  - Checks for errors in the training code using synthetic data

Types of Data Validation

- Single-batch validation
  - Are there anomalies in a single batch of data?
- Inter-batch validation
  - Are there changes between training x serving data?
  - Are there changes between successive batches of data?
- Model testing
  - Are there any assumptions in the training code not reflected in the data?

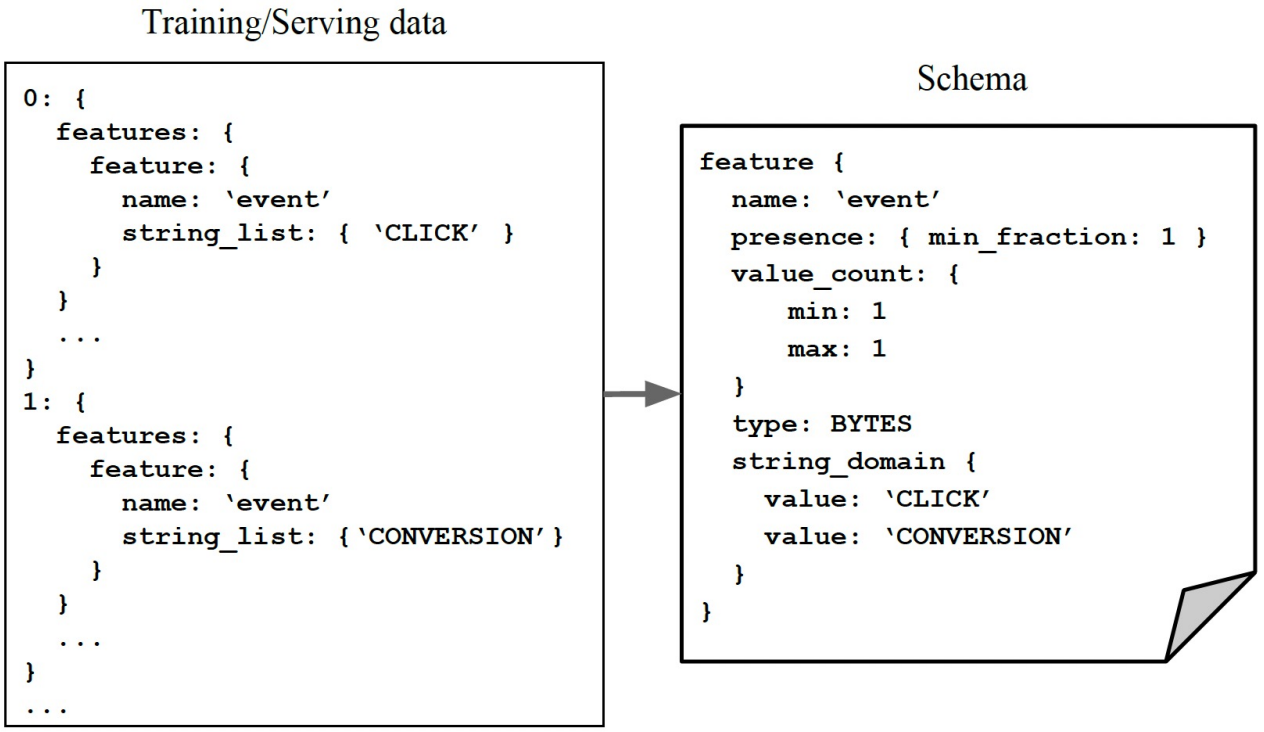
Single-batch validation

- Assumptions or expectations:
  - Data characteristics should be stable within each batch
  - Some characteristics remain stable across batches that are closer in time (no drastic changes)

- To validate these expected characteristics authors employ the traditional notion of **schema**

Schema

- Encode the data constraints and semantics



Schema (cont'd)

- When an anomaly is detected, the report includes remediation suggestions.
  - Fix the data or fix the schema

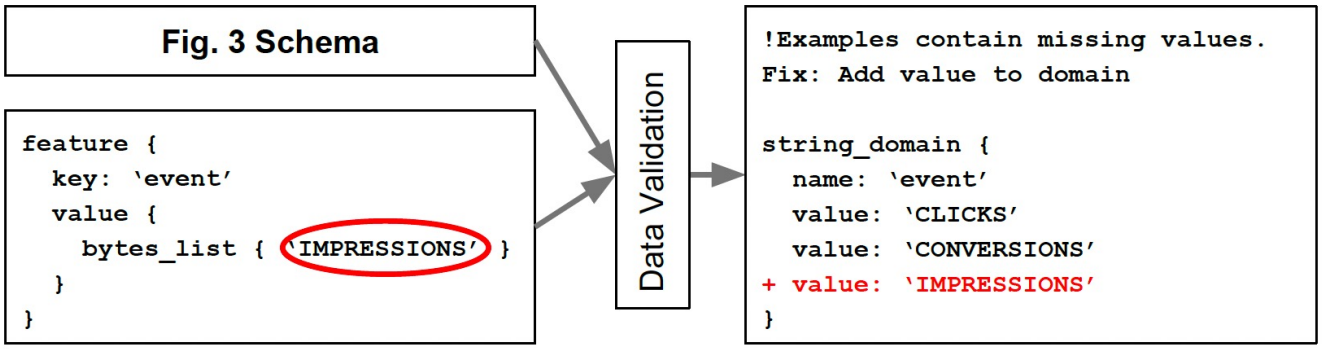


Figure 4: Schema-driven validation

Schema Life-Cycle

- Pipeline owners are responsible for the schema
  - Schema will be constantly updated and validated
- Do users need to create a schema for all properties?
  - What happens in datasets with hundreds of features?
- Data validator generates an initial schema for all features based on the initial batch
  - Based on “reasonable heuristics”

Inter-batch validation

- **Training serving skew:** training data vs serving data
- Feature skew: feature assumes different values when training versus at serving time
- Distribution skew: distribution of feature values is different from what is seen in serving time
- Scoring/Serving skew: only a subset of the scored examples are served

Quantifying Distribution Distance

- There will always be differences between training and serving data
  - Are they different enough to warrant an alert?

Experiment:

1M from N(0, 1)

+10k from N(0, 2)

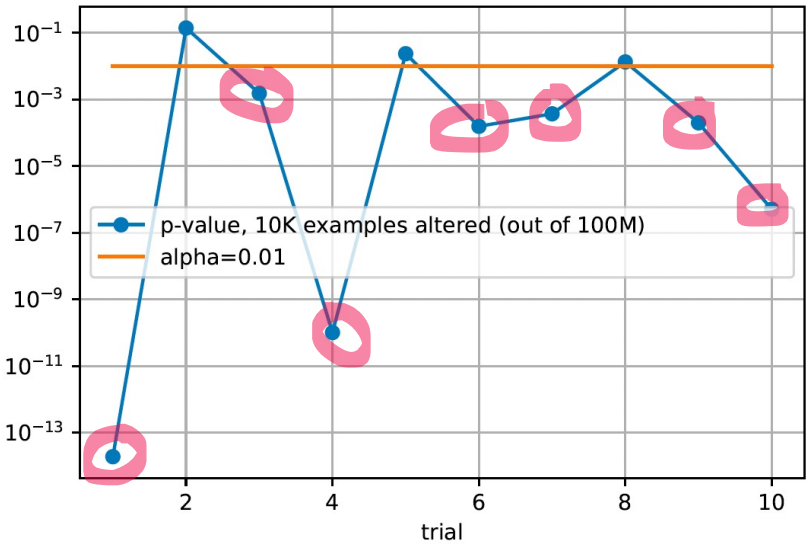


Figure 5: Sensitivity of chi-square test.

Too sensitive!

It would flag 7 out of 10 cases.

Quantifying Distribution Distance

- There will always be differences between training and serving data
  - Are they different enough to warrant an alert?

$$d_\infty(p, q) = \max_{i \in S} |p_i - q_i|$$

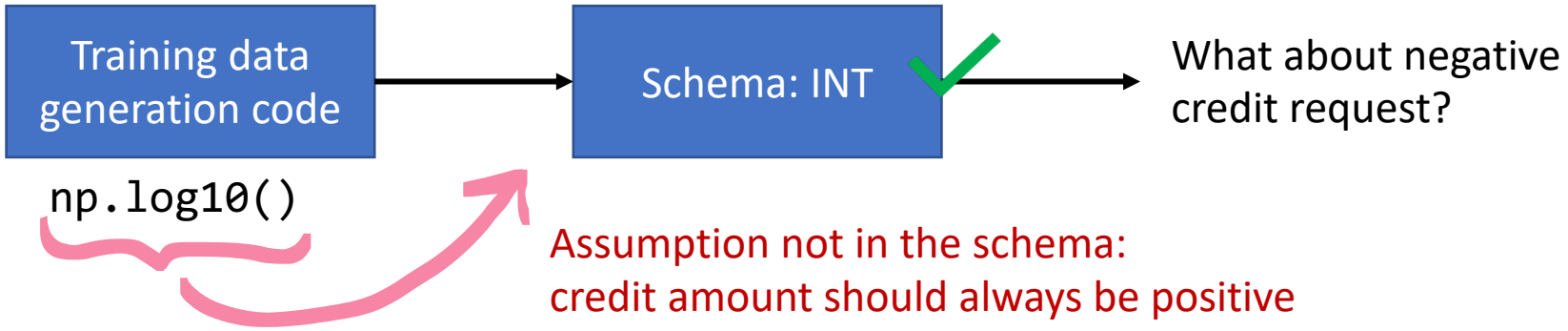
Let us take number of jobs in our German credit:

$$S = [0, \quad 1, \quad 2, \quad 3]$$
$$p = [0.2, \ 0.5, \ 0.2, \ 0.1]$$
$$\mathrm{q} = [0.1, \ 0.5, \ 0.1, \ 0.3]$$
$$\mathrm{p} - q = [0.1, \ 0.0, \ 0.1, \ 0.2]$$

Max of 20% of skew in one of the value probabilities

Model Unit Testing

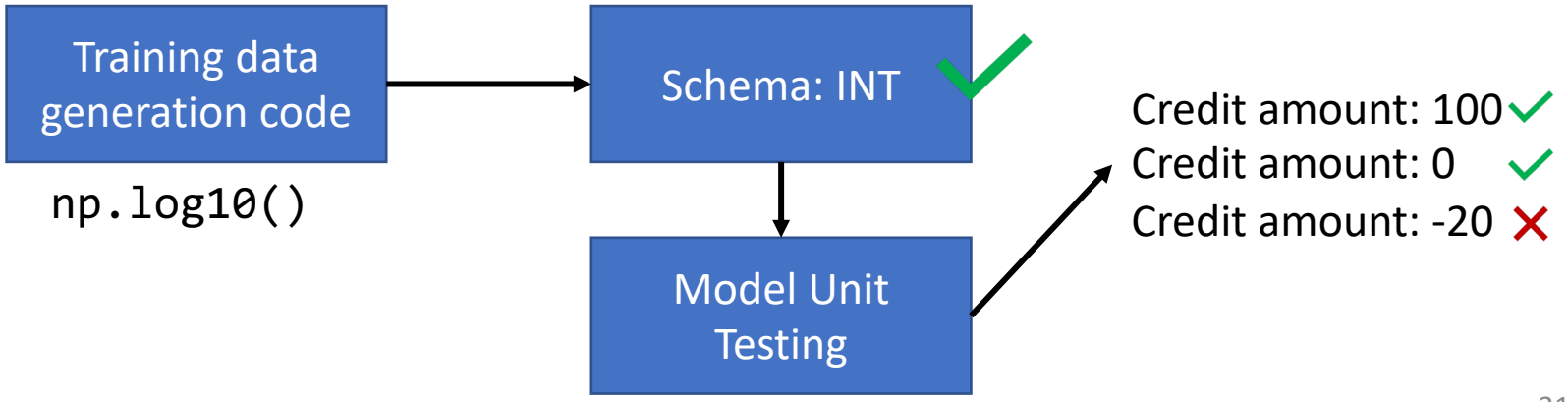
- Uses the schema to generate tests using Fuzz testing
  - Randomly generate inputs that adhere to the schema
- Example from the German credit report
  - Credit amount



Model Unit Testing (cont'd)

- Uses the schema to generate tests using Fuzz testing
  - Randomly generate inputs that adhere to the schema

- Example from the German credit report
  - Credit amount



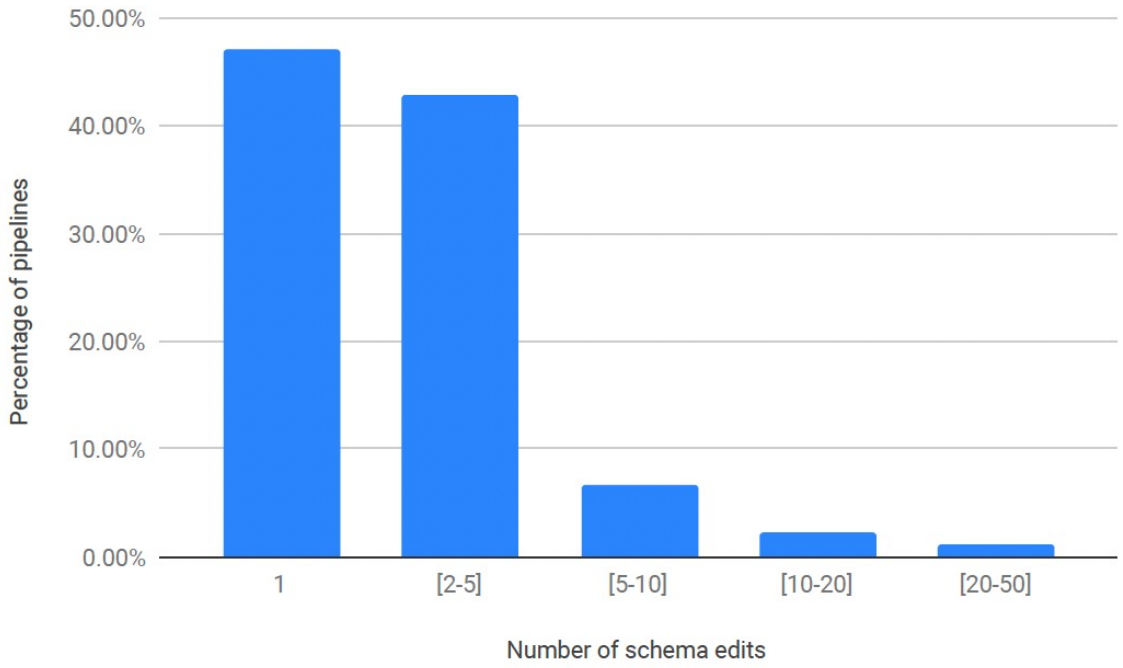
Empirical Validation

- The data validation system has been deployed in production
  - Part of the ML platform
  - Several petabytes of training and serving data per day

- How often users change their schemas?
- What are the most common anomalies detected?

How often users change their schemas?

- Based on 700 ML pipelines



Most common anomalies detected/fixed

Anomaly Category	Used	Fired	Fixed given Fired
New feature column (in data but not in schema)	100%	10%	65%
Out of domain values for categorical features	45%	6%	66%
Missing feature column (in schema but not in data)	97%	6%	53%
The fraction of examples containing a feature is too small	97%	3%	82%
Too small feature value vector for example	98%	2%	56%
Too large feature value vector for example	98%	<1%	28%
Data completely missing	100%	3%	65%
Incorrect data type for feature values	98%	<1%	100%
Non-boolean value for boolean feature type	14%	<1%	100%
Out of domain values for numeric features	67%	1%	77%

Fix if the anomaly was not triggered in the next 2 days.

Model Unit testing

- More than 70% of pipelines 1+ model unit defined
  - 80 thousand tests run per month
- 6% of the failed runs
  - Training code had incorrect assumptions
  - OR
  - The schema was underspecified

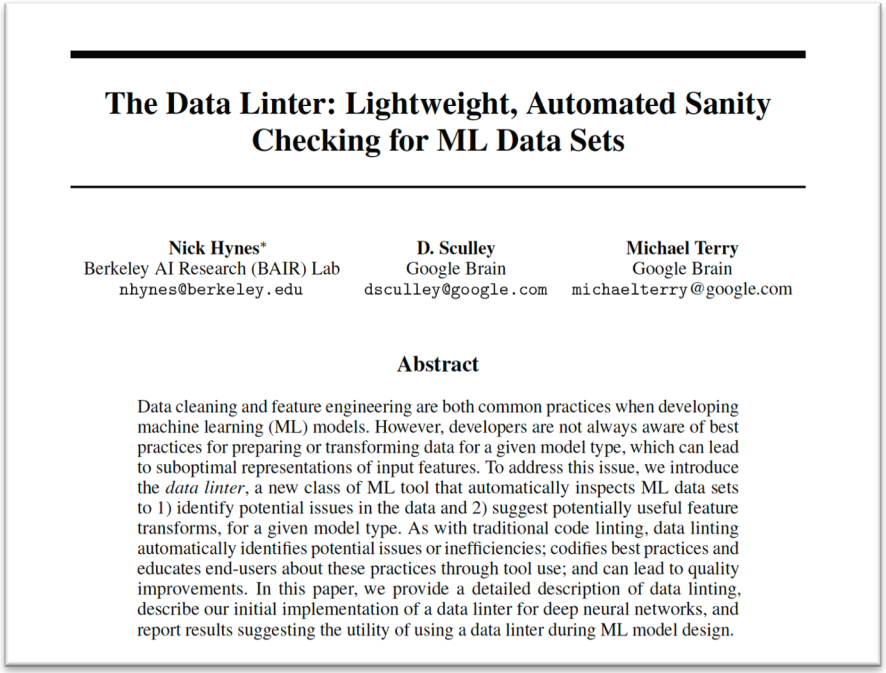
Case Studies

- Missing features in Google Play recommended pipeline
  - Identified features missing in serving but present in training data
  - Fix: 2% app install rate
- Data debugging
  - Features were silently dropped from the serving generating code
  - Fix: better performance parity (training and serving)

Open Discussion

Data Linter

Proposes a tool to inspect datasets to identify issues and improve feature engineering.



The Data Linter: Lightweight, Automated Sanity Checking for ML Data Sets

Nick Hynes*
Berkeley AI Research (BAIR) Lab
nhynes@berkeley.edu

D. Sculley
Google Brain
dsculley@google.com

Michael Terry
Google Brain
michaelterry@google.com

Abstract

Data cleaning and feature engineering are both common practices when developing machine learning (ML) models. However, developers are not always aware of best practices for preparing or transforming data for a given model type, which can lead to suboptimal representations of input features. To address this issue, we introduce the *data linter*, a new class of ML tool that automatically inspects ML data sets to 1) identify potential issues in the data and 2) suggest potentially useful feature transforms, for a given model type. As with traditional code linting, data linting automatically identifies potential issues or inefficiencies; codifies best practices and educates end-users about these practices through tool use; and can lead to quality improvements. In this paper, we provide a detailed description of data linting, describe our initial implementation of a data linter for deep neural networks, and report results suggesting the utility of using a data linter during ML model design.

Introduction

- **Lint** is a potential defect in the expression of a program

- There is a multitude of linters in traditional software
  - Spotting syntax errors, potential errors, bad practices, patterns of performance issues, …

Motivation

- Data issues can be identified with lightweight automation:
  - Numeric features on widely different scales
  - Values for special meaning (e.g., -9999 for missing data)
  - Malformed values
  - …
- The authors propose a Data Linter that implements all these checks

Contribution

- Data Linter
  - Analyzes user's training data
    - Summary statistics
    - Individual examples
    - Feature names
  - Suggest feature transformations for better performance
  - Available at https://github.com/brain-research/data-linter
- High-level architecture and implementation
- Empirical evaluation

High-level Architecture

- LintDetectors:
  - User-extensible
  - Logic to detect the problem

- DataLinter is the engine that applies LintDetectors
- LintExplorer
  - Presenting the results to user-friendly manner
  - Summarizes the data for LintDetectors

- Implemented as a Python library

Data Lints

- Miscoding Lints
  - Transform features to improve model fitness
  - Less robust models (linear models) may suffer from these problems

- Examples
  - Number as string
  - Enum as real
  - Tokenizable string
  - Circular domain (e.g., day of the week)
  - Date time as string
  - …

Data Lints (cont'd)

- Lints for Outlier and Scaling, for example…

- Unnormalized features
  - Wide range of values, consider normalizing them
- Tailed distribution
  - Presence of extreme values
- Uncommon list length
  - List feature has values with unusual length
- Uncommon sign detector
  - Negative values, etc.

Data Lints (cont'd)

- Packaging error data linters
  - Problems with the organization of the data

- Examples
  - Duplicate values
  - Empty examples

End-user Evaluation

- End-User evaluation
  - Eight software engineers used Data Linter
  - From novice to experienced engineers

- Best results
  - Linter suggestion (normalizing inputs) led to improvement in precision from 0.48 to 0.59.
- Reasonable performance
- Common feedback from users to silence some linters

Kaggle Dataset Evaluation

- Evaluation on 600 publicly available datasets
- Used the library pandas' automatic type-inference

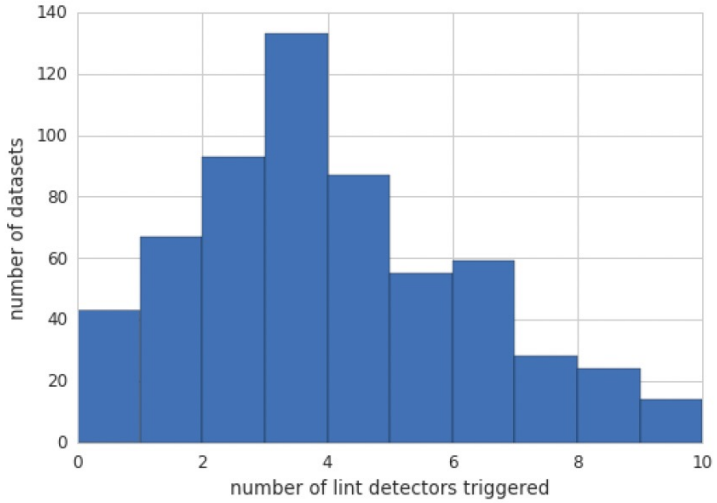


Figure 1: **Histogram of Number of Data Lints Per Data Set, Across 600 Kaggle Data Sets.** Interestingly, only about 7% of the data sets we examined as part of this study had zero data lints and were thus "lint free." The other 93% had at least one data lint, suggesting the utility of automated checking.

Kaggle Dataset Evaluation

- Evaluation on 600 publicly available datasets
- Used the library pandas' automatic type-inference

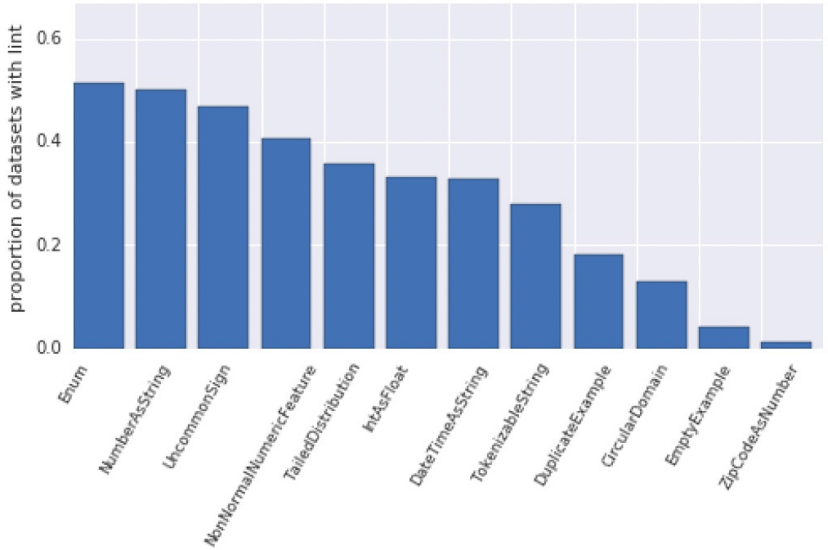


Figure 2: **Frequency of Data Lints Across 600 Kaggle Data Sets.** Some data issues are relatively common, including encoding numeric values as strings and issues around encoding enumerated values. But even rarer issues such as empty examples or potentially problematic encoding of postal codes do occur in the real-world data sets.

Validate the results

- Randomly sampled and manually examinate 35 data sets
  - Identified no false negative
- Validity of this analysis is limited by the author's knowledge
  - Threats to Validity?

Open Discussion

Course Project Presentation

- Next week, February 21st, 2022

- Presentations of 15 minutes + 5 minutes Q&A

References

- [Video] Data Validation for Machine Learning