Data Analysis and Validation for ML

Neoklis (Alkis) Polyzotis, Google Research

Collaborators: Eric Breck, Sudip Roy, Steven Whang, Martin Zinkevich
Outline

- ML in production is hard, and a big part of “hardness” is related to data
- We need mechanisms to analyze, validate, track, ... data in ML pipelines, in an end-to-end fashion
- Data analysis and validation (and their benefits) in the TensorFlow Extended platform
“Doing ML in production is hard.”

-Everyone who has ever tried
Starting point: Data and a question

I have data!
I have a question!
Let’s use ML!
Starting point: Data and a question

I have data!
I have a question!
Let's use ML!

- Sources: DBs, KV stores, Logs, …
- Formats: JSON, relational, unstructured, …
- Raw or curated
- We can assert few invariants [DHG+ SE4ML]
Data-access paths in training/serving

- Unit: all sessions in one day
- Large size
- High throughput

- Unit: current session
- Small size
- Low latency
Getting to a good model

- What features can be derived from the data?
- Is the model good enough?
- Should data be encoded differently?
- Should there be more data? More features?
Several experiments later...
Ready to launch!

Training Data → Train → Model → Evaluate → Model → Serve → Serving Data

Prepare → Training Input Data → Prepare

Serve → Serving Input Data → Prepare
So far, the user has dealt with this...
...but they actually have to worry about much more.
An example of data failure

- No new features or data, same training and serving logic

Refactor backend that generates a feature
An example of data failure

- No new features or data, same training and serving logic
An example of data failure

- No new features or data, same training and serving logic

Refactor backend that generates a feature

Prod rollout

Incompatible binaries result in errors ⇒ feature = -1
An example of data failure

- No new features or data, same training and serving logic
- Model performance goes south
- Issues propagate through the system (bad serving data ⇒ bad training data ⇒ bad models)
- Re-training can be expensive ⇒ Catching errors early is important
Life of an ML pipeline: Validating data

- Which data properties affect significantly the quality of the model?
- Any dependencies to other data/infrastructure?
Tracking training/serving skew

- What are possible deviations between training and serving data?
- Are they important?
Alerting on data errors

- How to formulate alerts so that they are understandable and actionable?
- What is the sensitivity for alerts?
Fixing data

- Will fixing the data improve the model?
- Which part of the data is problematic?
- What is the fix?
- How to backfill the data with the fix?
Everything in place

Now we can launch!
Several weeks (and production fires) later...
Life of an ML pipeline: The cycle starts over

Training Data → Train → Model → Serve → Serving Data

Validate → Fix → Evaluate

Prepare → Training Input Data → Serve Input Data → Prepare

Fix

I want to add data, features, models...
1st Dimension: High-level data activities

- Fixing
- Validation
- Understanding
- Preparation
2nd Dimension: Users

ML Expert

Broad knowledge of ML. Knows how to create models and how to use statistics. Advises on dozens of pipelines.

SWE

Understands the problem domain. Most ML experience is with this product. Coding is world class.

SRE

Problem fixer. On-call for possibly hundreds of pipelines. Can’t afford to know the details. Dealing with many issues simultaneously.
2nd Dimension: Users

- Rollback the pipeline to a working state
- Fix the quantization of price
- Implement and babysit a backfill
3rd Dimension: Time in the pipeline’s lifecycle

Validation

Fixing

Understanding

Preparation

Experiment
Launch
Refinement
Maintenance
...
“It’s the data, stupid!”

Every ML pipeline starts with the data

It doesn’t matter if you can train/serve fast if the data is problematic

Understanding the model requires a good understanding of the input data

⇒ Treat ML data as assets on par with source code and infrastructure

⇒ Develop processes for testing, monitoring, cataloguing, tracking, ..., ML data
Data Analysis and Validation in the TensorFlow Extended platform

https://youtu.be/fPTwLVCq00U

TFX: A TensorFlow-Based Production-Scale Machine Learning Platform
Denis Baylor, Eric Breck, Heng-Tze Cheng, Noah Fiedel, Chuan Yu Foo, Zakaria Haque, Salem Haykal, Mustafa Ispir, Vihan Jain, Levent Koc, Chiu Yuen Koo, Lukasz Lew, Clemens Mewald, Akshay Naresh Modi, Nokliss Polyzotis, Sukriti Ramesh, Sudip Roy, Steven Euijong Whang, Martin Wicke, Jarek Wilkiewicz, Xin Zhang, Martin Zinkevich
Google Inc.

ABSTRACT
Creating and maintaining a platform for reliably producing and deploying machine learning models requires careful orchestration of many components—a learner for generating models based on training data, modules for analyzing and validating both data as well as models, and finally infrastructure for serving models in production. This becomes particularly challenging when data changes over time and fresh models need to be produced continuously. Unfortunately, such orchestration is often done ad hoc using glue code and custom scripts developed by individual teams for specific use cases, leading to duplicated effort and fragile systems with high technical debt.

We present TensorFlow Extended (TFX), a TensorFlow-based general-purpose machine learning platform implemented at Google. By integrating the aforementioned components into one platform, we were able to standardize the components, simplify the platform configuration, and reduce the time to production from the order of months to weeks, while adopting machine learning as a tool to gain knowledge from data across a broad spectrum of use cases and products, ranging from recommender systems [6, 7], to clickthrough rate prediction for advertising [13, 15], and even the protection of endangered species [5].

The conceptual workflow of applying machine learning to a specific use case is simple: at the training phase, a learner takes a dataset as input and emits a learned model; at the inference phase, the model takes features as input and emits predictions. However, the actual workflow becomes more complex when machine learning needs to be deployed in production. In this case, additional components are required that, together with the learner and model, comprise a machine learning platform. The components provide automation to deal with a diverse range of failures that can happen in production and to ensure that model training and serving happen reliably. Building this type of automation is non-trivial, and it becomes even more challenging when we consider the following complications:
Focus of this paper
Data Ingestion
Data Analysis
Data Validation
Data Transformation
Trainer
Model Evaluation and Validation
Serving
Logging

Integrated Frontend for Job Management, Monitoring, Debugging, Data/Model/Evaluation Visualization
Shared Configuration Framework and Job Orchestration
Shared Utilities for Garbage Collection, Data Access Controls
Pipeline Storage

Figure 1: High-level component overview of a machine learning platform.
Goals

Provide turn-key functionality for a variety of use cases

Codify and enforce end-to-end best practices for ML data
**Problem:** Diverse data storage systems with different formats
**Problem:** Diverse data storage systems with different formats

**Solution:** Data ingestion normalizes data to a standard representation

When needed, enforces consistent data handling b/w training and serving
Problem: Gaining understanding of TB of data with O(1000s) of features is non-trivial

Solution: Scalable data analysis and visualization tools

**Problem**: Finding errors in TB of data with O(1000s) of features is challenging

- ML data formats have limited semantics
- Not all anomalies are important
- Data errors must be explainable
  
  E.g., “Data distribution changed” vs “Default value for feature `lang` is too frequent”

[Data management challenges in Production Machine Learning](https://www.sigmod.org/2017/tutorials/4.html) tutorial in SIGMOD’17
The Data Schema: Documenting what is “expected”

**Schema Example**

```json
feature {
    name: 'event'
    presence: REQUIRED
    valency: SINGLE
    type: BYTES
    domain {
        value: 'CLICK'
        value: 'CONVERSION'
    }
}
```
The Data Schema: Documenting what is “expected”

**Schema Example**

```json
feature {
  name: 'event'
  presence: REQUIRED
  valency: SINGLE
  type: BYTES
  domain {
    value: 'CLICK'
    value: 'CONVERSION'
  }
}
```

`event` is a required feature that takes exactly one bytes value in {“CLICK”, “CONVERSION”}. 
The Data Schema: Documenting what is “expected”

**Schema Example**

```yaml
feature {
  name: 'event'
  presence: REQUIRED
  valency: SINGLE
  type: BYTES
  domain {
    value: 'CLICK'
    value: 'CONVERSION'
  }
}
```

**event** is a required feature that takes exactly one bytes value in {"CLICK", "CONVERSION"}.

Also in the schema:
- Context (training vs serving) where feature appears
- Constraints on value distribution
- + many more ML-related constraints

Schema life cycle:
- TFX infers initial schema by analyzing the data
- TFX proposes changes as the data evolves
- User curates proposed changes
Schema Validation: Any errors in a single dataset?

**Schema**

```yaml
feature {
  name: 'event'
  presence: REQUIRED
  valency: SINGLE
  type: BYTES
  domain {
    value: 'CLICK'
    value: 'CONVERSION'
  }
}
feature {
  name: 'num_impressions'
  type: INT
}
```

**Training Example**

```yaml
feature {
  name: 'event'
  value: 'IMPRESSION'
}
feature {
  name: 'num_impressions'
  value: 0.64
}
```
Schema Validation: Any errors in a single dataset?

**Schema**

```json
feature {
    name: 'event'
    presence: REQUIRED
    valency: SINGLE
    type: BYTES
    domain {
        value: 'CLICK'
        value: 'CONVERSION'
    }
}
feature {
    name: 'num_impressions'
    type: INT
}
```

**Training Example**

```json
feature {
    name: 'event'
    value: 'IMPRESSION'
}
feature {
    name: 'num_impressions'
    value: 0.64
}
```

'TFData Validation: 'event': unexpected value
Fix: update domain

```json
feature {
    name: 'event'
    presence: REQUIRED
    valency: SINGLE
    type: BYTES
    domain {
        value: 'CLICK'
        value: 'CONVERSION'
        + value: 'IMPRESSSION'
    }
}
```
Schema Validation: Any errors in a single dataset?

**Schema**

```plaintext
feature {
  name: 'event'
  presence: REQUIRED
  valency: SINGLE
  type: BYTES
  domain {
    value: 'CLICK'
    value: 'CONVERSION'
  }
}
feature {
  name: 'num_impressions'
  type: INT
}
```

**Training Example**

```plaintext
feature {
  name: 'event'
  value: 'IMPRESSION'
}
feature {
  name: 'num_impressions'
  value: 0.64
}
```

**TFX Data Validation**

- `'event': unexpected value`
  Fix: update domain
  ```plaintext
  feature {
    name: 'event'
    presence: REQUIRED
    valency: SINGLE
    type: BYTES
    domain {
      value: 'CLICK'
      value: 'CONVERSION'
      +   value: 'IMPRESSION'
    }
  }
  }
  ```

- `'num_impressions': wrong type`
  Fix: deprecate feature
  ```plaintext
  feature {
    name: 'num_impressions'
    type: INT
    + deprecated: true
  }
  ```

---

Data Ingestion  
Data Validation  
Model-driven Validation  
Skew Detection
Model-driven Validation: Any data assumptions in the training logic?

Schema

```protobuf
feature {
  name: 'event'
  presence: REQUIRED
  valency: SINGLE
  type: BYTES
  domain {
    value: 'CLICK'
    value: 'CONVERSION'
  }
}
feature {
  name: 'num_impressions'
  type: INT
}
```

TF Training

```
10  ...
11  i = tf.log(num_impressions)
12  ...
```
Model-driven Validation: Any data assumptions in the training logic?

```python
10  ...
11  i = tf.log(num_impressions)
12  ...
```

Line 11: invalid argument for tf.log
Is training data in day N “similar” to day N-1?

Dataset “similarity” checks:

- Do the datasets conform to the same schema?
- Are the distributions “similar”?
- Are features exactly the same for the same examples?

Skew problems common in production and usually easy to fix once detected

⇒ Greatest bang for buck for data validation
Deployment stats

- Data analysis and validation used by hundreds of product teams through TFX
- O(PB) of training/serving data per day
- Certain teams set up TFX solely for data analysis/validation
- Several documented ML wins by catching data anomalies early
Why this matters: Google Play
Personalized Recommendation
"What we serve is what we train. ...Or is it?"

+2%

App install rate by fixing training-serving feature skew.
References and links

- “TFX: A TensorFlow-Based Production-Scale Machine Learning Platform”, KDD’17
- “Data Management Challenges in Production Machine Learning”, SIGMOD’17
- “Data Validation for ML”, soon on Arxiv
Parting thoughts

- ML in production is hard, and a big part of “hardness” is related to data
- We need mechanisms to analyze, validate, track, ... data in ML pipelines, in an end-to-end fashion
- Data analysis and validation in TensorFlow Extended are critical for the robustness of production ML pipelines
  - In addition: Many interesting research problems in the intersection of DBs & ML