Data Infrastructure for Machine Learning

Eric Breck, Neoklis Polyzotis, Sudip Roy, Steven Euijong Whang*, Martin Zinkevich
Google Research
{ebreck,npolyzotis,sudipr,swhang,martinz}@google.com

ABSTRACT

Data quality is critical for effective machine learning, and this makes data a first-class citizen in the context of machine learning, on par with algorithms, software, and infrastructure. As a result, machine-learning platforms need to support data analysis and validation in a principled manner, throughout the lifecycle of the machine learning process. This paper reviews the data infrastructure we built at Google to address these challenges in the context of large-scale production machine learning pipelines.

1 INTRODUCTION

At a high level, a machine learning pipeline uses training data to generate a model, which can then be used to perform inference with serving data (see Figure 1). Most work in academia and industry has focused on improving the efficiency and quality of training and inference. However, we contend that it is equally important to take a data point-of-view and examine critically the management of the (training and serving) data at the end points: simply put, the quality and speed of training and inference are immaterial if the input data is wrong.

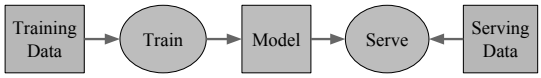

Figure 1: Machine learning pipeline from 10,000 feet

There are several challenges to address in managing training and serving data. Data is typically large, it may arrive continuously, and in the latter case it may also arrive in (incomplete) chunks. Moreover, data can contain errors that need to be caught early, before they propagate downstream and taint models. In addition, data typically comes with few semantics attached to it, which makes error detection a hard problem.

*Corresponding author

SysML, February 2018, Stanford, California USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

In this paper we describe the infrastructure that we built at Google to analyze and validate the (training and serving) data that drive production machine learning pipelines. Our system allows machine learning users to continuously check for errors in each instance of training or serving data, detect drifts between instances, and test how data errors can affect the correctness of models. At the core is a data *schema* that describes the user's expectations for correct data. This concept is borrowed from database management systems but its maintenance and application acquires new flavors in the context of machine learning.

Our system has been deployed at Google as part of the TFX platform [2] and currently analyzes and validates more than one petabyte of data per day. Our system has caught several data errors in production pipelines with two tangible benefits for machine learning users: savings in engineering hours to detect, debug, and fix the errors, and model-quality wins from using better data. While data schema and validation were introduced in the TFX paper [2], we provide more details on skew detection and introduce model unit testing.

2 DATA-DRIVEN SCHEMA

A schema represents a logical model of data for machine learning that contains constraints and captures semantics that are necessary for machine learning data validation and model testing.

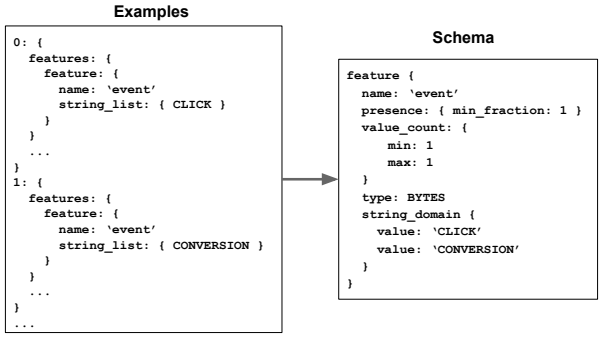

Figure 2: A data-driven schema

Figure 2 shows an example schema (represented in protocol-buffer format [1]). The data in this case is expected to contain an *event* feature which appears in 100% of the examples (the `presence` specification), takes exactly one value (the `value_count` specification), and this can be either 'CLICK' or 'CONVERSION' (the `domain` specification). Although not shown in the figure, the schema can also encode domains for numeric features and contain additional metadata including semantic information (e.g., string values for boolean True or False), whether a feature's integer values are considered as

identifiers, or constraints on the distribution of the data, to name a few.

The schema is designed to be both human- and machine-readable/writable: human-readable/writable because the machine learning user is ultimately the owner of the schema and is expected to curate it over time; machine-readable/writable because schema is used to perform data validation and model testing.

A schema can become large (many machine learning pipelines have 1000s of features) and so our system takes two steps to help the machine learning user curate the schema. First, our system can generate an initial version of the schema based on analysis of existing data. This initial version attempts to capture salient properties of the data without overfitting to the particular data instance. The latter is important, since an overfitted schema will lead to false-positives in data validation and hence noisy alerts for the machine learning user. Second, our system can recommend changes to the schema as new data is examined.

3 DATA VALIDATION AND SKEW DETECTION

Our system validates an instance of the data by comparing it with the schema (Figure 3). Any discrepancy results in an alert to the machine learning user. An important point is that these alerts must be interpretable and actionable. For example, saying that the label does not satisfy the schema is less actionable than saying that a label is not in 10% of examples. Our system can also suggest changes to the schema for errors that can reflect a natural evolution of the data (e.g., the appearance of new domain values). The goal is to help the machine learning user curate the schema as the data evolves over time.

Our system also detects skew between instances (e.g. drifts over time, or between training and serving) comparing them to each other. Of particular importance is training-serving skew, which can occur when different code paths are followed in the generation of training and serving data. This type of skew is common in production pipelines and has adverse effects on the quality of inferences. Our system detects skew both at the level of individual examples (if they have identifiers) and in aggregate (based on statistical goodness-of-fit metrics). Again, our system focuses on errors that are easy to localize and thus debug. For instance, we consider "the most frequent value changed" to be a more informative indication of drift than "the KL-divergence is too high". This intuition has led us to choose more interpretable, but perhaps less general, statistical measures of drift. For example, we use bounds on L-∞ rather than generic measures such as the chi-squared test or KL divergence. (The chi-squared test also has the problem of giving many false positives for large data where a small amount drift is expected.)

4 MODEL UNIT TESTING

In addition to validating data, the schema can also be used to generate synthetic data for model unit testing. Data fuzz

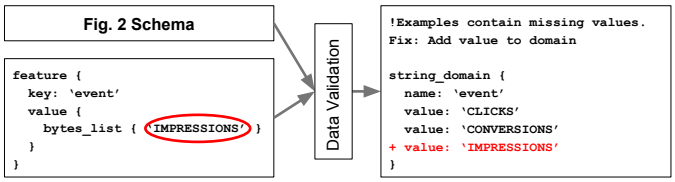

Figure 3: Schema-driven validation

testing is a common practice to evaluate services on a variety of inputs and is recently being used in machine learning model testing as well [4]. However, purely randomly generated data might lack required features or otherwise cause training code to crash in a way that would not occur with real data. In contrast, the schema enables us to randomly generate data in a principled fashion, such that crashes would indicate a problem needing to be addressed.

For instance, our unit test framework uses the schema in Figure 2 and ensures that each randomly generated data example has the *event* feature, each of which would have one value, uniformly chosen at random to be either 'CLICK' or 'CONVERSION'. Similarly, integral features would be random integers from the range specified in the schema, to name another case. The framework also includes specific generators for image examples and is easily extensible by users, e.g. to include additional generation constraints not expressible in the schema. It also includes the option to use a specific saved snapshot of data, but this is usually less robust than generating from data, and few teams use it.

The generated data is then used to train and evaluate a machine learning model for a small number of steps. The goal is not to test the model's ability to learn, but to test the code's ability to run, process data, and call machine learning APIs. Model unit testing is one part of testing an end-to-end machine learning system [3].

Using this type of testing, our system can uncover discrepancies between the schema (and hence, the machine learning user's expectation of the data) and the assumptions made in modeling code. For instance, suppose that the modeling code applies a logarithm transformation on an integer feature, but the schema does not specify the constraint that the feature is positive. During testing, the modeling code will be exercised with synthetic examples where the feature has non-positive values, thus leading to an error. This error can direct the machine learning user to update either the modeling code or the schema, so as to align data expectations between validation and training.

REFERENCES

[1] 2017. Protocol Buffers. https://developers.google.com/protocol-buffers/. (2017).

[2] 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, Neoklis Polyzotis, Sukriti Ramesh, Sudip Roy, Steven Euijong Whang, Martin Wicke, Jarek Wilkiewicz, Xin Zhang, and Martin Zinkevich. 2017. TFX: A TensorFlow-Based Production-Scale Machine Learning Platform. In *SIGKDD*. 1387–1395. https://doi.org/10.1145/3097983.3098021

[3] Eric Breck, Shanqing Cai, Eric Nielsen, Michael Salib, and D. Sculley. 2017. The ML Test Score: A Rubric for ML Production Readiness and Technical Debt Reduction. In *Proceedings of IEEE Big Data 2017*.

[4] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. 2017. DeepXplore: Automated Whitebox Testing of Deep Learning Systems. In *Proceedings of the 26th Symposium on Operating Systems Principles (SOSP '17)*. 1–18.