Towards Automated ML Model Monitoring: Measure, Improve and Quantify Data Quality

Tammo Rukat, Dustin Lange
[tammruka,langed]@amazon.com
Amazon Research

Sebastian Schelter
sebastian.schelter@nyu.edu
New York University

Felix Biessmann
fbiessmann@beuth-hochschule.de
Beuth University Berlin

1 INTRODUCTION

Machine learning (ML) has become a central component in modern software applications, giving rise to many new challenges [8, 15, 20]. Tremendous progress has been made in this context with respect to model serving [1, 6, 10], experiment tracking [14, 16, 22, 23], model diagnosis [4, 5, 11, 21] and data validation [4, 18].

In this paper, we focus on the arising challenge of automating the operation of deployed ML applications, especially with respect to monitoring the quality of their input data. Existing approaches [1, 18, 22] for this problem have not yet reached broad adoption. One reason for that is that they often require a large amount of domain knowledge, e.g., to define “data unit tests” and corresponding similarity metrics and thresholds for detecting data shifts. Additionally, it is very challenging to test data at early stages of a pipeline (e.g., during integration) without explicit knowledge of how the data will be processed by downstream applications. In other cases, the engineers in charge of operating a deployed ML model may not have access to the model internals, for example if they leverage a popular cloud ML service such as Google AutoML1 for training and inference. Integrating and automating data quality monitoring into ML applications is also difficult due to the lack of agreed upon abstractions for defining and deploying such applications.

We list three approaches to tackle data quality in ML applications from recent work: (i) Measuring data quality with “data unit tests” using the Deequ [18] library; (ii) Improving data quality with missing value imputation using the DataWig [3] library; and (iii) Quantifying the impact of data quality issues on the predictive performance of a deployed ML model [19]. Finally, we outline challenges and potential directions for combining these approaches and for automating their configuration in real world deployment settings.

2 INDIVIDUAL APPROACHES

Successful ML applications require careful data selection, data preprocessing and model building. Given that the responsibilities for such applications are usually distributed across people with different competencies [12], engineers in charge of operating ML applications should be supported with automatic data quality monitoring infrastructure, which (a) does not require domain expertise, and (b) does not require access to model internals.

**Measuring data quality.** When confronted with poor predictive performance of a model, it is usually a good idea to first inspect its input data. We have previously proposed Deequ2 [17, 18] for the definition and scalable execution of data quality tests. The library allows users to define quality tests for structured data in a declarative manner, and execute them on large datasets with Apache Spark. Parts of the model monitoring functionality of the cloud ML platform Amazon Sagemaker have been built upon Deequ3.

Despite the growing adoption of Deequ, a number of challenges remain for further automation. In particular, it is difficult to automatically decide which statistics to inspect for which columns, and how to set thresholds for alarming based on these statistics. We are currently experimenting with an automation approach for use cases where a new batch of

---

1. https://cloud.google.com/automl/
2. https://github.com/awslabs/deequ
data needs to be tested and ingested at regular intervals (e.g., daily ingestion of log files). In this approach, we maintain time series of column statistics from previously observed data batches. To decide whether a new batch of data should be accepted, we compare its statistics to a forecast-based estimate of the expected statistics based on the observed time series.

**Improving data quality.** Once data quality issues are detected, they need to be addressed appropriately. We may for instance want to impute missing values. Libraries like DataWig\(^4\) [2, 3] support missing value imputation for structured data in a fully automated manner. DataWig automatically featureizes all input columns, except the column with to-be-imputed values and trains a classifier on a concatenation of these features to predict the missing value. It automatically tunes hyperparameters in a given time budget, performs domain adaption by detecting and correcting shifts between rows that are used for training and rows that are subject to imputation, and calibrates likelihoods for imputed values allowing to control which imputed values are useful for downstream applications. These properties render it attractive for data engineers who do not have the necessary domain expertise and/or time to investigate every dataset in detail [7]. Exploiting dependencies between features is not only helpful for imputation, but also for monitoring quality on a single record level. Consider the example highlighted in [12], where the values for the column country change from upper case (US) to lower case (us), causing problems in downstream feature extractors. At serving time, a well-calibrated imputation model that treats these values as missing can identify the lower case version, (us), as unlikely under the previously learnt model and raise warnings automatically. A practical limitation of this approach, in particular with regard to automation, is the difficulty to deal with rare values that are common in heavy-tailed real-world datasets and with previously unseen values in general.

**Quantifying the impact of data quality issues on the predictive performance of an ML model.** A central question for data quality monitoring is the impact that any observed change in the distribution of the input data has on the predictive performance of the deployed ML model [12]. The approaches described above are model independent and, thus, have no means of quantifying this downstream effect. Imagine a scenario in which a model simply ignores a given input feature (e.g., due to L1 regularisation). A shift in this feature may be detected by systems monitoring the input data and might lead to unnecessary alarms. Recent approaches to model monitoring leverage statistics of the model predictions [9, 13] in a model agnostic way.

We have built upon this work to predict the performance of generic black box classifiers\(^5\) [19]. We achieve this by generating perturbed copies of the original data where each copy resembles typical errors and data quality problems. Ideally, typical types of perturbations are provided by a domain expert. The original classifier applied to the perturbed datasets yields pairs of the form (quantized output distribution, model performance). These are fed to a regression model that learns to predict the model performance, the “pipeline performance predictor” (PPP). An alarm is raised at serving time if the predicted performance falls below a predefined performance threshold.

The only step requiring domain knowledge within this procedure is the specification of potential data errors. Our experiments, however, indicate that it is often sufficient to train the regression model with pre-specified common error types [4]. One shortcoming of this approach lies in its unawareness of the particular data points or properties that cause a predicted performance drop. Future work should put emphasis on a better understanding of the effect that errors in the data have on a model’s output distribution.

3 VISION FOR INTEGRATION

While the three above described techniques each have their independent use case, fully automated data quality monitoring may greatly benefit from a tight integration. Consider, for instance, the following scenario: Deequ detects missing values or likely errors in a particular column of a dataset. Based on a DataWig model, values in this column are imputed or replaced. Eventually, PPP verifies that the errors would indeed have caused a large drop in model performance and that this drop is significantly mitigated with DataWig’s corrections.

The effort of creating an appropriate configuration would merely be comprised of (i) defining column constraints in Deequ (which can be semi-automated), (ii) specifying typical errors (which can be dispensed with for common scenarios) and required model performance thresholds for PPP, (iii) setting a threshold confidence for missing-value imputation in DataWig. Then the entire process operates without human intervention: The ML pipeline is patched automatically and warnings/errors are raised when degrading performance cannot be repaired.

**REFERENCES**


