Deequ - Data Quality Validation for Machine Learning Pipelines

Sebastian Schelter, Stefan Grafberger, Philipp Schmidt, Tammo Rukat
Mario Kiessling, Andrey Taptunov, Felix Biessmann, Dustin Lange
Amazon Research
{sseb,stefgraf,phschmid,tammruka,kiesslin,taptunov,biessman,langed}@amazon.com

Abstract

Modern machine learning (ML) systems are comprised of complex ML pipelines which typically have many implicit assumptions about the data they consume (e.g., about the scales of variables, the presence of missing values or the dictionary of categorical values). Violations of these assumptions can result in crashes or wrong predictions. We therefore present *Deequ*, a library that allows users to explicitly specify their assumptions about the data in a declarative way. *Deequ* enables the efficient automatic validation of these assumptions on large datasets. It is an open source library based on Apache Spark and meets the requirements of production use cases at Amazon.

Data is at the center of modern enterprises and institutions. Online retailers, for example, rely on data to support customers making buying decisions, to forecast demand [3] or to schedule deliveries. Missing or incorrect information seriously compromises any decision process. A common trend across different industries is to automate business processes with machine learning (ML) techniques on large datasets [7, 12]. Data-specific problems with ML pipelines commonly occur because of two reasons, erroneous data and missing data. Erroneous data, e.g., out-of-dictionary values for categorical variables or accidental changes in the scale of computed features can cause ML pipelines to unexpectedly change their predictions which is challenging to detect [8]. Furthermore, some model classes cannot handle missing data, and therefore missing entries are commonly replaced by default values. These default values need to be carefully chosen however as to not unexpectedly change the models' predictions [11]. Another hard-to-detect source of problems is a distribution shift between training data and data to predict on. Machine learning specific sanity checks [13, 4] and explicit data validation components [3, 2, 4, 5, 9] are actively being researched for exactly these reasons.

Data validation for such scenarios is becoming increasingly challenging, as data lives in many different places (databases, key-value stores, distributed filesystems) and comes in many different formats. Many such data sources do not support integrity contraints and data quality checks, and often there is not even an accompanying schema available, as the data is consumed in a 'schema-on-read' manner, where a particular application takes care of the interpretation. We recently proposed *Deequ*[1], an open-source library for automating the verification of data quality at scale [10] with Apache Spark. *Deequ* provides a declarative API, which combines common quality constraints with user-defined validation code, and thereby enables *unit tests for data*. It allows users to explicitly and declaratively state their expectations about the data which they consume or produce. Furthermore, *Deequ* allows users to automate the data validation process by integrating the tests into ML pipelines. In case of violations, data can be quarantined and data engineers can be automatically notified. *Deequ* is designed to scale to datasets with billions of rows, given that the constraints to evaluate are chosen carefully.

[1] https://github.com/awslabs/deequ

32nd Conference on Neural Information Processing Systems (NIPS 2018), Montréal, Canada.

Example. We exemplify the unit tests for data provided by *Deequ* in a fictitious use case of predicting clicks on movie trailers from observed impressions. We assume that we have already trained a model and now wish to predict clicks for new unseen data. Listing 1 shows a test for the implicit assumptions we might want to enforce on the unseen data. We assume that each sample contains a boolean value in the target column `has_clicked`. The column `trailer_duration` contains the length of the trailer in seconds. We check that the data is of type integer and that each trailer has a duration between 30 and 300 seconds. This allows us to catch errorenous samples that would be generated if someone accidentally changed the scale of this column from seconds to milliseconds or accidentally fills up missing values with 0 (the default value for integers in many languages). Next, we issue constraints on the categorical `genre` column. We demand that there are no missing values and that we only see genres that we have already observed in the training data (by constraining the range to the previously computed set `genresSeenInTrainingData`). Additionally, we compare the distribution of genres in the data to predict on to the corresponding distribution in the training data via the `histogramSatisfies` constraint. The user-defined function `notDiverged` compares the categorical distributions in the `genre` column of the training set and test set (e.g., with a G-test [6]) and returns a boolean value afterwards. Finally, we demand that every sample has an associated `movie_id` and that the overall number of distinct movies in the unseen data is not larger than the number of movies we would expect in the system.

```
// Computed in advance
val numAvailableMovies = ...
val genresSeenInTrainingData = ...
val distInTrainset = ...

// assumptions about data to predict on
val validationResultForTestData = VerificationSuite()
 .onData(testSet)
 .addCheck()
  .isComplete("has_clicked", "genre", "movie_id")
  .hasDataType("has_clicked", Boolean)
  .hasDataType("trailer_duration", Integral)
  .isContainedIn("trailer_duration", 30 to 300)
  .isContainedIn("genre", genresSeenInTrainingData)
  .histogramSatisfies("genre", { distInTestset =>
    notDiverged(distInTrainset, distInTestset) })
  .hasCountDistinct("movie_id", _ <= numAvailableMovies)
 .run()

if (validationResultForTestData.status != Success) {
  // abort pipeline, notify data engineers
}
```

Listing 1: Test data validation in a fictitious click prediction use case for movie trailers.

Execution. *Deequ* identifies the data statistics required for evaluating the constraints and generates queries in SparkSQL [1] with custom designed aggregation functions in order to compute the statistics. For performance reasons, it applies multi-query optimization to enable scan-sharing for the aggregation queries in order to minimize the number of required passes over the input data. Once the data statistics are computed, *Deequ* invokes the user-defined validation functions contained in the test code (e.g., `_ <= numAvailableMovies`) and returns the evaluation results to the user. Additionally, it supports efficient constraint evaluation on changing data (e.g., incrementally growing logs) with a stateful abstraction (which we do not show here due to lack of space).

Advanced Features and Future Work. *Deequ* contains several advanced features not covered in the example. In cases where it is difficult to supply exact thresholds for certain data statistics (e.g., due to seasonality), we allow users to plug in anomaly detection algorithms that can supply sensible thresholds given past data quality metrics. Furthermore, we provide tools to profile large datasets and automatically suggest constraints based on the resulting profiles. In future work, we intend to investigate methods for the automatic generation of tests for specific ML model classes and data to be used in ML pipelines.

References

[1] M. Armbrust, R. S. Xin, C. Lian, Y. Huai, D. Liu, J. K. Bradley, X. Meng, T. Kaftan, M. J. Franklin, A. Ghodsi, et al. Spark sql: Relational data processing in spark. pages 1383–1394, 2015.

[2] D. Baylor, E. Breck, H.-T. Cheng, N. Fiedel, C. Y. Foo, Z. Haque, S. Haykal, M. Ispir, V. Jain, L. Koc, et al. TFX: A TensorFlow-Based Production-Scale Machine Learning Platform. pages 1387–1395, 2017.

[3] J.-H. Böse, V. Flunkert, J. Gasthaus, T. Januschowski, D. Lange, D. Salinas, S. Schelter, M. Seeger, and Y. Wang. Probabilistic demand forecasting at scale. *PVLDB*, 10(12):1694–1705, 2017.

[4] E. Breck, S. Cai, E. Nielsen, M. Salib, and D. Sculley. The ml test score: A rubric for ml production readiness and technical debt reduction. *IEEE Big Data*, 2017.

[5] E. Breck, N. Polyzotis, S. Roy, S. E. Whang, and M. Zinkevich. Data Infrastructure for Machine Learning. 2018.

[6] T. Dunning. Accurate methods for the statistics of surprise and coincidence. *Computational linguistics*, 19(1):61–74, 1993.

[7] A. Y. Halevy, P. Norvig, and F. Pereira. The unreasonable effectiveness of data. *IEEE Intelligent Systems*, 24(2):8–12, 2009.

[8] N. Polyzotis, S. Roy, S. E. Whang, and M. Zinkevich. Data management challenges in production machine learning. pages 1723–1726, 2017.

[9] S. Schelter, J.-H. Böse, J. Kirschnick, T. Klein, and S. Seufert. Automatically tracking metadata and provenance of machine learning experiments. 2017.

[10] S. Schelter, D. Lange, P. Schmidt, M. Celikel, F. Biessmann, and A. Grafberger. Automating large-scale data quality verification. *PVLDB*, 11(12), 2018.

[11] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, M. Young, J.-F. Crespo, and D. Dennison. Hidden technical debt in machine learning systems. pages 2503–2511, 2015.

[12] C. Sun, A. Shrivastava, S. Singh, and A. Gupta. Revisiting unreasonable effectiveness of data in deep learning era. pages 843–852, 2017.

[13] M. Terry, D. Sculley, and N. Hynes. The Data Linter: Lightweight, Automated Sanity Checking for ML Data Sets. 2017.