DuckDQ
Data Quality Validation for Machine Learning Pipelines

• Introduction
• 3 key design choices behind DuckDQ
• Experiments
• Conclusion

02.07.2021, DSDSD, Till Döhmen, Fraunhofer FIT
Data Quality Validation in Python

- Deequ (Spark) [1]
- TFX Data Validation (TensorFlow) [2]
- Great Expectations (pandas + SQL) [3]
- Hooqu (pandas) [4]

```python
import com.amazon.deequ.VerificationSuite
import com.amazon.deequ.checks.{check, CheckLevel, CheckStatus}

val verificationResult = VerificationSuite()
  .onDataFrame(data) // spark data frame
  .addCheck(
    Check(new CheckLevel.Fuzz, "unit testing my data")
      .hasPattern("data_time", "\d{4}\-\d{2}\-\d{2} \d{2}:\d{2}:\d{2}")) // check data format
      .isPositive("trip_distance") // positive trip distance
      .hasApproxQuantile("id", 0.5, _ <= expected_dist()) // check median trip distance via external function
  .run()
```

Deequ Verification API: Code Example

Data Quality Validation in Python

Data Quality Validation for ML in Python:
- either designed for very large datasets,
- or tightly coupled to ML platforms,
- or lightweight but not integrated with the ML lifecycle and (potentially) not optimized for production use.

DuckDQ offers...
- “Data Assertions” that can be serialized together with ML models and shipped into production seamlessly.
- Efficient integration with Pandas and scikit-learn pipelines.
- Opportunities for failure analysis and drift detection even when the original data is not around anymore.
- A flexible, SQL-based computational backend which is also suitable for efficient validation of data in other SQL databases.
What are Data Assertions?

DuckDQ Assertion API: Code Example

```python
from duckdq.sklearn import Assertion, DQPipeline, CheckLevel

inp_assert = Assertion(CheckLevel.EXCEPTION)
    # check date format and positive trip distance
    .has_pattern("date_time", r"(\d{4}-\d{2}-\d{2}\ 1\d{2}:\d{2}:\d{2})")
    .is_positive("trip_distance")
    # check median trip distance via external function
    .has_quantile("trip distance", 0.5, lambda x: x <= expected_dist)

outp_assert = Assertion(CheckLevel.WARN)
    # check ratio of positives is less than 10%
    .has_histogram_values("y", lambda x: x["1"].ratio < 0.1)

pipeline = DQPipeline(
    ['feature_encoding', Pipeline(...)],
    ['clf', RandomForestClassifier()],
    input_assertion=inp_assert,
    output_assertion=outp_assert
)

model = pipeline.fit(X_train, y_train)
```
Under the Hood...

- Stateful computation of metrics
- Scan sharing optimization
- Integration with DuckDB
Overview

DuckDQ Verification Suite

- Engine
- Operators
- State
- Metric/State Repository
- Evaluation
- Data
- Assertions

Mapping

Result
Stateful Computation of Metrics

- States are **intermediate representations of metrics**, which **can be merged across multiple batches** of data.

**Examples:**

<table>
<thead>
<tr>
<th>Minimum</th>
<th>Mean</th>
<th>Standard Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>States: min</td>
<td>States: total, count</td>
<td>States: mean, n, dsq</td>
</tr>
<tr>
<td>Merge: minC = min(minA,minB)</td>
<td>Merge: totalC = totalA+totalB, countC = countA+countB</td>
<td>Merge: nC = nA + nB</td>
</tr>
<tr>
<td>Metric: minC</td>
<td>Metric: mean = totalC/countC</td>
<td>avgC = (nA * meanA + nB * meanB) / nC</td>
</tr>
</tbody>
</table>

![math_expression]

- What we call „stateful computations“ are basically streaming/online algorithms
- DataSketches\(^1\) such as KLL (approx. quantiles) and HyperLogLog (approx. distinctness) fit in well with this paradigm.

\(^1\)https://datasketches.apache.org/
Stateful Metrics in the ML Pipelines Context

- States are recorded in every prediction run
- All metrics can be merged across multiple prediction batches
- Enables, e.g., drift detection without processing the data twice / needing to keep the original data around at all.
Scan Sharing (SQL Engine)

- DuckDQ’s default engine is SQL-based
- Most operators require only a SELECT-statements (blue)
- Some require an additional GROUP BY-statement (black)

Operators/Assertions
- Minimum
- Maximum
- Mean
- Sum
- StandardDeviation
- Completeness
- Uniqueness
- Distinctness
- HistogramProperty
- ApproxQuantile
- PatternMatch
- MaxLength
- MinLength
- …
Scan Sharing (SQL Engine)

• Instead of executing each operator individually, we group all possible operators into a single query, which requires only one full table scan at max (scan sharing).

  SELECT count(..), sum(..) FROM dataframe
  SELECT count(*), count(..) FROM dataframe
  SELECT min(..) FROM dataframe

  SELECT count(..), sum(..), count(*), count(..), min(..) FROM dataframe

• For GROUP BY-Operators, scan sharing is done as well, but scans can only be shared among operators which require the same groupings.
Integration with DuckDB

„But in the context of Python-based ML Pipelines, we are dealing with DataFrames, so what’s the use of an SQL engine??“

• That’s where DuckDB comes in...
• DuckDB is an efficient, in-process analytical RDBMS with a neat Python integration.
• Pandas DataFrames can be queried by the DuckDB engine similar to physical tables.

```
import duckdq
import pandas as pd
hotel_search_logs = pd.read_csv("hotel_search_logs.csv")
count = duckdb.query("SELECT count(*) FROM hotel_search_logs").fetchone()
print(count)
```

Query a Pandas DataFrame with DuckDB¹

Integration with DuckDB

- We can hence apply our SQL-based optimization to Pandas DataFrames.

As a side-effect, we can practically use the same SQL engine for general purpose data quality validation on other SQL-based databases.
Evaluation

Tests for *completeness* of 5 cols (4 numeric, one string) and *uniqueness* of 5 cols (4 numeric, one string)


Loading the same data into MySQL and check the same quality constraints...
Evaluation

• Is the difference relevant?
• Let’s put the validation time is relation to the prediction time across different data set sizes.

![Graph with varying number of rows](image)

![Graph with varying number of features](image)
Conclusion

• Current solutions for DQA in the Python data science ecosystem pose a potential runtime bottleneck in ML pipelines.

• DuckDQ:
  • is a lightweight and efficient solution for keeping ML pipelines error-free which does not require any heavy additional infrastructure.
  • It is particularly well suited for medium-sized data sets.
  • Offers opportunities for ML-monitoring (e.g. drift detection).

• Lowering the entry barrier for the adoption of MLOps best practices.
More about DuckDQ...

• Source Code: [https://github.com/tdoehmen/duckdq](https://github.com/tdoehmen/duckdq)