Blue Elephants Inspecting Pandas: Inspection and Execution of Machine Learning Pipelines in SQL

Maximilian E. Schüle[1], Luca Scalerandi[2], Alfons Kemper[2] and Thomas Neumann[2]

[1]*University of Bamberg*
[2]*TUM*

Abstract
Data preprocessing, the step of transforming data into a suitable format for training a model, rarely happens within database systems but rather in external Python libraries and thus requires extraction from the database systems first. However, database systems are tuned for efficient data access and offer aggregate functions to calculate the distribution frequencies necessary to detect the under- or overrepresentation of a certain value within the data (bias).

We argue that database systems with SQL are capable of executing machine learning pipelines as well as discovering technical biases—introduced by data preprocessing—efficiently. Therefore, we present a set of SQL queries to cover data preprocessing and data inspection: During preprocessing, we annotate the tuples with an identifier to compute the distribution frequency of columns. To inspect distribution changes, we join the preprocessed dataset with the original one on the tuple identifier and use aggregate functions to count the number of occurrences per sensitive column. This allows us to detect operations which filter out tuples and thus introduce a technical bias even for columns preprocessing has removed. To automatically generate such queries, our implementation extends the mlinspect project to transpile existing data preprocessing pipelines written in Python to SQL queries, while maintaining detailed inspection results using views or common table expressions (CTEs). The evaluation proves that a modern beyond main-memory database system, i.e. Umbra, accelerates the runtime for preprocessing and inspection. Even PostgreSQL as a disk-based database system shows similar performance for inspection to Umbra when materialising views.

Keywords
Inspection, SQL, Database Systems

1. Extended Abstract

To allow inspection directly within SQL, we extend the mlinspect framework by tuple tracking [1]. To implement tuple tracking, an identifier for the tuples is needed. Most database systems allow access to the physical table location, which could be used as an identifier. For PostgreSQL, the `ctid` is such an identifier, although it is not suitable as a long-term identifier[1]. This is based on the possibility of garbage collection routines changing the physical location.

LWDA'24: Lernen, Wissen, Daten, Analysen. August 23–25, 2024, Würzburg, Germany
maximilian.schuele@uni-bamberg.de (M. E. Schüle); luca.scalerandi@tum.de (L. Scalerandi); kemper@in.tum.de (A. Kemper); neumann@in.tum.de (T. Neumann)
www.uni-bamberg.de/dt (M. E. Schüle)
0000-0003-1546-269X (M. E. Schüle)

[1]https://www.postgresql.org/docs/12/ddl-system-columns.html

```
 1 CREATE TABLE data (a int, s int); -- sensitive column: ”s”
 2 INSERT INTO data (values (1,1),(1,2));
 3 WITH orig AS ( -- the original data with exposed ctid
 4   SELECT ctid, a, s FROM data),
 5 curr AS ( -- current representation after preprocessing
 6   SELECT ctid, s FROM orig WHERE s > 1),
 7 orig_count AS ( -- original count per value of column ”s”
 8   SELECT s, count(*) AS cnt FROM orig GROUP BY s),
 9 curr_count AS ( -- current count per value of column ”s”
10   SELECT s, count(*) AS cnt FROM curr GROUP BY s),
11 orig_ratio AS ( -- original ratio per value of column ”s”
12   SELECT s, (cnt*1.0 / (select count(*) FROM orig)) AS ratio
13   FROM orig_count),
14 curr_ratio AS ( -- current ratio per value of column ”s”
15   SELECT s, (cnt*1.0/(select sum(cnt) FROM curr_count)) AS ratio
16   FROM curr_count)
17 -- join on the sensitive column to calculate the ratio change
18 SELECT o.s, o.ratio - COALESCE(c.ratio,0) AS bias_change
19 FROM curr_ratio c RIGHT OUTER JOIN orig_ratio o ON o.s = c.s;
```

Listing 1: Ratio measurement (column present).

As we extract the `ctid`, initially only once within a common table expression (Listing 1 lines 3-6), we ensure that we use consistent identifiers during the checks.

To reproduce mlinspect's bias inspections in SQL, we need to compute and compare the ratios for each sensitive column (*HistogramForColumns*). In order to compare the ratios (Listing 1 lines 11-19), it is necessary to group by each sensitive column to count the initial and current number of occurrences per value. One common table expression retrieves the original number (Listing 1 lines 7f). In the simple case, the result table already contains the sensitive column. In this simple case, an aggregate function counts the number of occurrences per column directly (Listing 1 lines 9f).

References

[1] M. E. Schüle, L. Scalerandi, A. Kemper, T. Neumann, Blue elephants inspecting pandas: Inspection and execution of machine learning pipelines in SQL, in: J. Stoyanovich, J. Teubner, N. Mamoulis, E. Pitoura, J. Mühlig, K. Hose, S. S. Bhowmick, M. Lissandrini (Eds.), Proceedings 26th International Conference on Extending Database Technology, EDBT 2023, Ioannina, Greece, March 28-31, 2023, OpenProceedings.org, 2023, pp. 40–52. URL: https://doi.org/10.48786/edbt.2023.04. doi:10.48786/EDBT.2023.04.