# Efficient Computation of ML Explanations Using Provenance


Nkechi Jennifer Akinwale, Seokki Lee

University of Cincinnati


LinkedIn

Personal Webpage

## A. Motivation

- **High computational costs for computing explanations for ML models –** *Shapley value-based explanation involves evaluating all of the model's weights, which can grow exponentially with the number of features.*
- **Expensive recursion overhead from evaluating ML model encoded in Datalog** - *ML models can be expressed as recursive Datalog queries involving nested, mutual, and non-linear recursion.*

## B. Objective

- Introduce a provenance-based technique that efficiently computes all of the model's weights necessary to compute explanations.
- Capture provenance within a reasonable computational cost, in a single query evaluation queue.

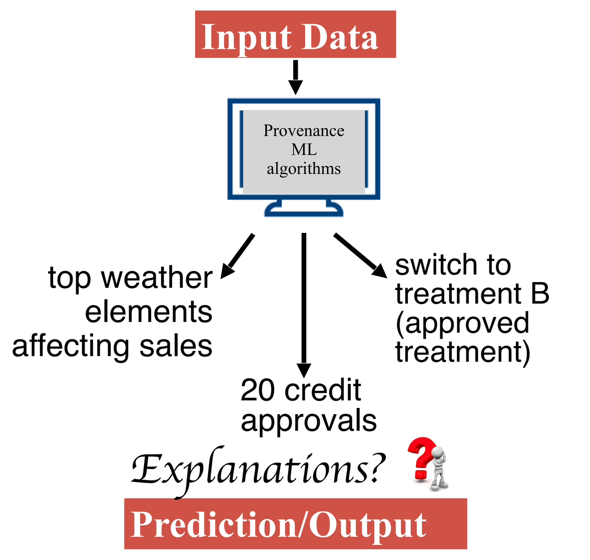



**Figure 1:** Image depicting the need for explanations

## C. Method

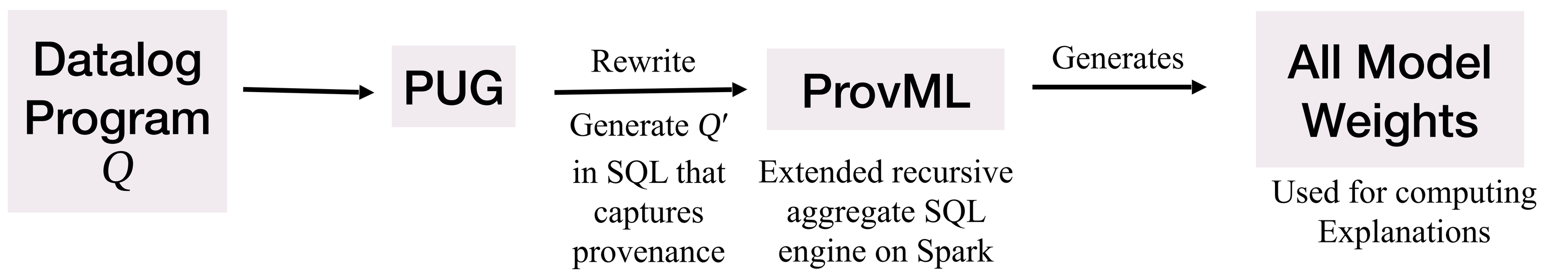
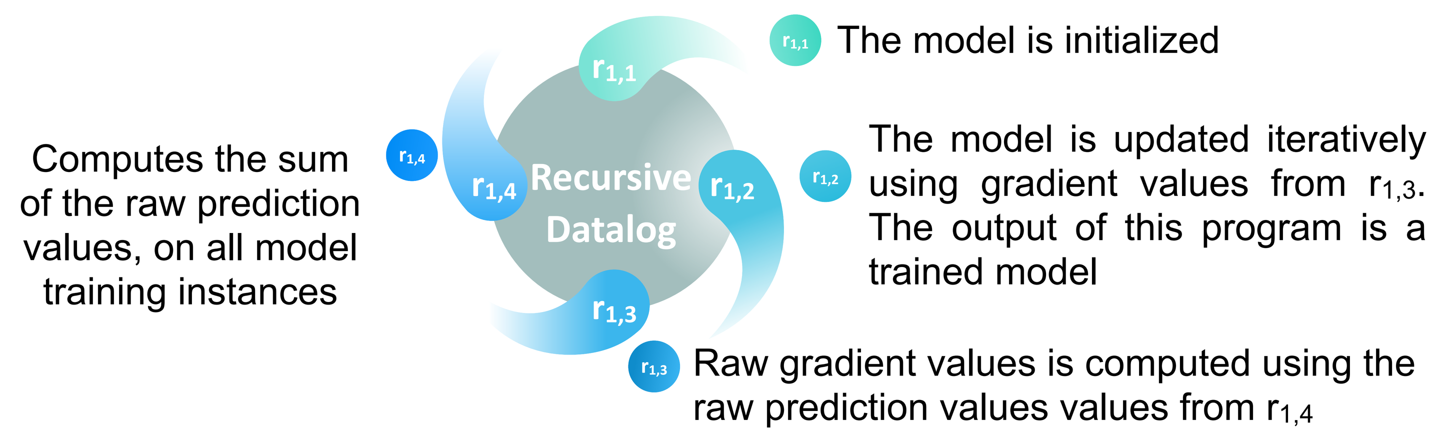



**Figure 2:** Proposed provenance framework

- **Input**: A Datalog program $Q$ that encodes an ML algorithm and a training dataset $Train_v$ [3]
- **Output**: All the model's weights
  - Extend PUG [2], a framework that captures provenance for Datalog, to instrument $Q$ into $Q'$ in SQL.
  - Extend RaSQL [1], a recursive-aggregate query engine, for computing the model's weights

$$
\begin{aligned}
Q:\quad & r_{1,1} : Model(0, C, 0.01) :- Train^v(I, C, V, Y) \\
& r_{1,2} : Model(J1, C, NP) :- Model(J, C, P), Gradient(J, C, G), \\
& \qquad NP = P - lr \cdot G/n, J1 = J + 1 \\
& r_{1,3} : Gradient(J, C, sum\langle I, G0\rangle) :- Train^v(I, C, V, Y), \\
& \qquad Predict(J, I, YP), G0 = 2 \cdot (YP - Y) \cdot V \\
& r_{1,4} : Predict(J, I, sum\langle C, Y0\rangle) :- Train^v(I, C, V, Y), Model(J, C, P), Y0 = V \cdot P
\end{aligned}
$$

**Figure 3:** An example Datalog program r1,* of BGD for LR




**$Train^v$ (input)**

| I | C | V | Y |
|---|---|---|---|
| 0 | temp | 0.3442 | 985 |
| 1 | temp | 0.3635 | 801 |
| 2 | temp | 0.1964 | 1349 |
| 0 | hum | 0.8058 | 985 |
| 1 | hum | 0.6961 | 801 |
| 2 | hum | 0.4373 | 1349 |

**Predict**

| J | I | YP |
|---|---|---|
| 0 | 0 | 0.0115 |
| 0 | 1 | 0.0106 |
| 0 | 2 | 0.0063 |

**Gradient**

| J | C | G |
|---|---|---|
| 1 | temp | -1787.99 |
| 1 | hum | -3881.79 |

**Model**

| J | C | NP |
|---|---|---|
| 0 | temp | 0.01 |
| 0 | hum | 0.01 |
| 1 | temp | 59.6103 |
| 1 | hum | 129.4037 |

**Table 1:** Input data and query result of $Q$ over example inout data - bike sharing dataset

- ProvML ensures the intermediate result size from model training is small by keeping the result from the previous iteration only, which is sufficient to compute the model's weights for the current iteration.
- Reduced recursion overhead – our method eliminates mutual recursion *occurring in Figure 3 where $r_{1,2}$ invokes $r_{1,3}$ which invokes $r_{1,4}$, and $r_{1,4}$, in turn updates the model.*

## D. ProvML

- **ProvML** is an extension of Recursive-aggregate SQL (RaSQL) built on top of Apache Spark. RaSQL uses a distributed semi-naïve evaluation for efficient computation. We leverage
  - efficient fixpoint evaluation,
  - its support for aggregation in recursion and
  - the computational benefits of query execution in a distributed data processing environment.

$Q'$:

```
model
AS (
    SELECT * FROM initial_model
    UNION ALL
    SELECT
        c, cf,
        CAST((p - (0.1 * gradient)) /
            (SELECT COUNT(*) FROM trainv) AS double precision) AS p,
        J + 1 AS J
    FROM
        (
            WITH trainvp AS (
                SELECT m.j, t.i, t.c, t.cf, t.v, t.y, m.p
                FROM trainvcf t JOIN model m USING (c,cf)),
            predictProv AS (
                SELECT t.j, t.i, t.c, t.cf, t.v, t.p, t.y,
                SUM(t.v * t.p) OVER(PARTITION BY t.i, t.cf) AS YP FROM trainvp t),
            gradientProv AS (
                SELECT *, 2 * (YP - y) * v AS G0 FROM predictProv P),
            gradient AS (
                SELECT *, SUM(G0) OVER(PARTITION BY j, c, cf) AS gradient
                FROM gradientProv)
            SELECT * FROM gradient
        ) g
        WHERE j < 2  -- iterations
)
-- Final output after all iterations
SELECT DISTINCT j, c, cf, p
FROM model
ORDER BY j, c, cf;
```

**Figure 4:** Snippet of rewritten SQL

- predictProv is generated based on $r_{1,4}$ by adding all the existential variables in the body to the head and express sum<C,Y0> in a window function.
- gradient is generated based on $r_{1,3}$ by replacing the body atoms with predictProv, including all the corresponding variables in the body to the head atom, and converting sum<I,G0> to a window function.

**$Train^v$ (input)**

| I | C | V | Y |
|---|---|---|---|
| 0 | temp | 0.3442 | 985 |
| 1 | temp | 0.3635 | 801 |
| 2 | temp | 0.1964 | 1349 |
| 0 | hum | 0.8058 | 985 |
| 1 | hum | 0.6961 | 801 |
| 2 | hum | 0.4373 | 1349 |

**predictProv**

| J | I | C | CF | V | P | Y | YP |
|---|---|---|---|---|---|---|---|
| 0 | 0 | hum | hum | 0.8058 | 0.01 | 985 | 0.0081 |
| 0 | 1 | hum | hum | 0.6961 | 0.01 | 801 | 0.0070 |
| 0 | 1 | hum | humtemp | 0.6961 | 0.01 | 801 | 0.0106 |
| 0 | 0 | hum | humtemp | 0.8058 | 0.01 | 985 | 0.0115 |
| 0 | 0 | temp | humtemp | 0.3442 | 0.01 | 985 | 0.0115 |
| 0 | 1 | temp | humtemp | 0.3635 | 0.01 | 801 | 0.0106 |
| 0 | 0 | temp | temp | 0.3442 | 0.01 | 985 | 0.0034 |
| 0 | 1 | temp | temp | 0.3635 | 0.01 | 801 | 0.0036 |

**gradientProv**

| J | I | C | CF | V | P | Y | YP | G0 |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | hum | hum | 0.8058 | 0.01 | 985 | 0.0081 | -1587.4130 |
| 0 | 1 | hum | hum | 0.6961 | 0.01 | 801 | 0.0070 | -1115.1425 |
| 0 | 1 | hum | humtemp | 0.6961 | 0.01 | 801 | 0.0106 | -1115.1374 |
| 0 | 0 | hum | humtemp | 0.8058 | 0.01 | 985 | 0.0115 | -1587.4075 |
| 0 | 0 | temp | humtemp | 0.3442 | 0.01 | 985 | 0.0115 | -678.0661 |
| 0 | 1 | temp | humtemp | 0.3635 | 0.01 | 801 | 0.0106 | -582.3193 |
| 0 | 0 | temp | temp | 0.3442 | 0.01 | 985 | 0.0034 | -678.0716 |
| 0 | 1 | temp | temp | 0.3635 | 0.01 | 801 | 0.0036 | -582.3244 |

**$Q'$ (output)**

| J | C | CF | P |
|---|---|---|---|
| 1 | hum | hum | 64.7081 |
| 1 | hum | humtemp | 64.7079 |
| 1 | temp | humtemp | 29.8395 |
| 1 | temp | temp | 29.8397 |

**Table 2:** Sample $Q'$ output - result of predictProv and gradientProv with example data

## E. PUG and ProvML Ongoing Key Enhancements

- Developed the algorithm that rewrites the input Datalog program for an ML model to a query in SQL that returns all the model weights by supporting window functions & subqueries for efficiently capturing provenance for recursive-aggregate queries. Currently implementing it in PUG.
- Developing ProvML that supports executing recursive-aggregate SQL that has window functions and subqueries necessary for maintaining the provenance of *Predict* and *Gradient*.

## F. References


1. JiaqiGu,YugoHWatanabe,WilliamAMazza,AlexanderShkapsky,MohanYang, Ling Ding, and Carlo Zaniolo. 2019. Rasql: Greater power and performance for big data analytics with recursive-aggregate-sql on spark. In SIGMOD.
2. Seokki Lee, Bertram Ludäscher, and Boris Glavic. 2018. PUG: a framework and practical implementation for why and why-not provenance. VLDB J. (2018).
3. Jin Wang, Jiacheng Wu, Mingda Li, Jiaqi Gu, Ariyam Das, and Carlo Zaniolo. 2021. Formal semantics and high performance in declarative machine learning using Datalog. The VLDB Journal (2021).