Chioma A. Onyekpere

FEATURE ENGINEERING WITH SQL

Preparing ML Data in PostgreSQL

OUTLINE

01

INTRODUCTION

Why SQL for Feature Engineering?

02

KEY POSTGRESQL FEATURE

SQL Techniques for Preprocessing

03

PRACTICAL EXAMPLES

04

PERFORMANCE & BEST PRACTICES

Introduction to Feature Engineering

Transform raw data into meaningful features to improve ML model performance.

Why PostgreSQL?

- **Scale**: Handles large datasets efficiently
- **Declarative speed**: Writing SELECT ... GROUP BY ... is faster to prototype than looping in code.
- **Production-ready**: Integration with production databases

Key/Powerful PostgreSQL Features

- Window Functions (ROW_NUMBER, LAG, LEAD) for rolling and comparative features.
- Common Table Expressions (CTEs) for breaking complex logic into readable steps.
- JSON/JSONB support to parse and extract nested data.
- Array & string functions to manipulate lists and text within SQL.

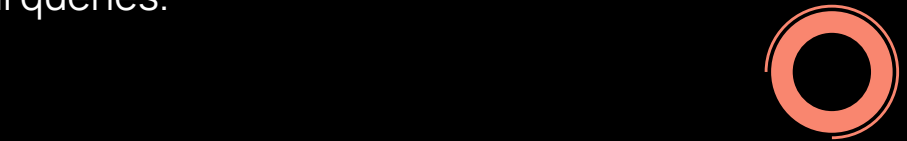
Techniques for Preprocessing

- **Rolling/Lag features**: compute previous or next values in a series.
- **Aggregations & grouping:** summarize data at various granularities (counts, averages).
- **Missing-value handling**: coalesce, conditional imputation, or filtering out nulls.
- **One-hot encoding**: use CASE WHEN … THEN 1 ELSE 0 END to turn categories into numeric flags.

Practical Example: Window Functions

```
SELECT
  month,
  revenue,
  LAG(revenue) OVER (ORDER BY month) AS previous_month_revenue,
  LEAD(revenue) OVER (ORDER BY month) AS next_month_revenue
FROM sales;
```

This query allows you to analyze month-to-month trends, making PostgreSQL a powerful tool for time-series and analytical queries.



Practical Example: JSON & Arrays

```
SELECT
  data->>'user_preferences' AS prefs_json,
  jsonb_array_elements_text(prefs_json) AS preference,
  COUNT(*) FILTER (WHERE preference = 'active') AS active_pref_count
FROM user_events
GROUP BY prefs_json;
```

This query pulls a JSON array of preferences, unnests it into rows, and counts how many are labeled “active.”

Best Practices

Key tips to keep SQL transformations efficient:

- Index key columns used in WHERE, JOIN, and PARTITION BY.
- Use materialized views for rarely rarely changing aggregates.
- Batch vs. on-the-fly: decide whether to compute features in bulk during off-hours or on demand at query time.

CONCLUSION

Feature engineering in SQL unlocks efficient, production-ready ML pipelines.

- PostgreSQL's advanced SQL features streamline preprocessing.
- A good balance would be to combine manual SQL control with orchestration tools like Airflow

RESOURCES

- https://www.postgresql.org/docs/current/functions-window.html
- https://www.postgresql.org/docs/current/queries-with.html
- https://www.postgresql.org/docs/current/functions-json.html
- https://www.postgresql.org/docs/current/functions-array.html
- https://www.postgresql.org/docs/current/functions-conditional.html
- https://www.postgresql.org/docs/current/rules-materializedviews.html