Use Apache Beam to build Machine Learning Feature System at Affirm
Use Apache Beam To Build Machine Learning Feature System At Affirm

- Hao Xu
ABOUT ME

# TABLE OF CONTENTS

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>01</strong></td>
<td><strong>02</strong></td>
</tr>
<tr>
<td><strong>BACKGROUND</strong></td>
<td><strong>PAIN POINTS</strong></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>03</strong></td>
<td><strong>04</strong></td>
</tr>
<tr>
<td><strong>SOLUTION</strong></td>
<td><strong>OUTCOME</strong></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Background

- BNPL
- Machine learning feature store
- Streaming and Batch Compute Platform
The Story of BNPL

Your 3 payments of $50.00

Total of payments $150.00
$50.00 is due next month

Set up automatic payments (optional)
You’ll pay $50.00 on each due date.

Complete your order

Pick a payment plan

$233.00/monthly
APR 0.00%
Interest $0.00
Total $233.00

$120.00/monthly
APR 15.01%
Interest $22.06
Total $142.06

$62.00/monthly
APR 15.01%
Interest $42.00
Total $62.00
If a user failed the third payment, is it likely that they will also fail the fourth one?

Has the user failed to make a loan payment, and if so, have we identified the issue? Should we approve another loan for them?
The payment data was processed in batches, resulting in a delay of a couple of days. Utilizing stream data can help prevent such delays in the future.
Feature Store

Figure 1. A feature store is the interface between feature engineering and model development.
Pain Points
Pain Points

Development Velocity
Slow backfilling of stream features. Excessive code required to define a feature.

Variety
Inability to join two streams from Kinesis together, which is typically required for stateful processing.

Visibility
Lack of registry to quickly lookup data sources, features and metadata.
Solution
MLFS Architecture

Diagram showing the architecture with components such as Flink, Apache Beam, Spark, MySQL, and Airflow.
Complex of Backfilling

Backfilling is the process to backfill a feature data to the historical point in time.
Unified Transformation Interface

class UnifiedTransformer(Transformer[beam.PCollection, beam.PCollection]):

    @property
def window(self) -> beam.WindowInto:
        return self._window

    @property
def event_transform(self) -> beam.PTransform:
        return self._event_transform

    @property
def aggregator(self) -> beam.PTransform:
        return self._aggregator

    def run(self, inputs: beam.PCollection) -> beam.PCollection:
        if self.feast_context.runner == Runner.flink:
            if self.window:
                inputs = inputs | self.window
            return |
                inputs
                | self.event_transform.with_output_types(Tuple)
                | self.aggregator.with_output_types(Tuple)
        elif self.feast_context.runner == Runner.spark:
            return |
                inputs
                | self.event_transform.with_output_types(Tuple)
                | self.aggregator.with_output_types(Tuple)
        else:
            raise ValueError("Unsupported runner: {}.",\n                format(self.feast_context.runner))
Unified Transformation Interface

@stream_feature_view(
    entities=[entity_registry['user_ari']],
    ttl=timedelta(days=0),
    schema=[
        Field(name="user_ari", dtype=String),
        Field(name="timestamp", dtype=UnixTimestamp),
        Field(name="latest_payment_fail", dtype=UnixTimestamp),
        Field(name="latest_payment_fail_ach_nsf", dtype=UnixTimestamp),
    ],
    online=True,
    source=user_payment_fails_stream_source,
    timestamp_field="timestamp",
    tags={},
    mode="flink",
)

def user_last_payment_fail(feast_context: FeastContext, inputs: PCollection) -> PCollection:
    transformer = UnifiedTransformer(
        feast_context=feast_context,
        aggregator=LatestFeatureAggregator(feast_context, 'timestamp'),
        event_transform=extract_payment_fail_data,
    )
    return transformer.run(inputs)
Outcome
Performance boost

**Backfilling time**
Backfilling time improved by 80%

**Code lines**
Reduced 100+ lines to 20+ lines

**Registry**
200+ data sources

The time spent to backfill features for feature
*time_since_user_checkout* and
*tiem_since_user_last_payment_failure*
Future improvement

1. OOTB transformation interface
2. Transformation framework
3. Improvement on Beam Spark Runner