Zipline – Airbnb’s ML Data Management Framework

Varant Zanoyan / Strata
Varant Zanoyan

Team: Machine Learning Infrastructure
Role: Software Engineer
Agenda

1) The Machine Learning Workflow
2) Motivation for Zipline (the problem)
3) Zipline implementation (the solution)
4) Deep dive (technical)
Team Mission

Equip Airbnb with shared technology to build *production-ready* ML applications with no *incidental complexity*.
The Machine Learning Workflow
Zipline in the ML Iteration Workflow
Zipline in the ML Workflow

- Idea
- Data
- Training Set
- Training & Evaluation
- Discover Data

Zipline
Zipline in the ML Production Workflow

- Training Set Updates
- Model Retraining
- Push model to prediction service
- Production Features
- Model monitoring
- Data quality monitoring
Motivation
“I spend 60% of my time generating training data”
We already have a data warehouse

Why do we even need Zipline?

- We have data
- Defining new pipelines is easy enough (business analysts do it all the time)
- We already built a lot of tooling for all that
- Why build something new?
Motivating Example – Likelihood to book

- Predict likelihood to book when a user views an experience
- Example feature: sum of prior bookings in past 7 days
Timelines in ML Workflow

Features

Product

Problem

Pred

Label

Time

Training data set

- F1: 0 5 7 4
- F2: 3 2 4
- F3: 0 8 4

7 4 L L
3 2 8 8
Limitations of a standard warehouse for Machine Learning: Bound to daily accuracy

<table>
<thead>
<tr>
<th>User</th>
<th>date</th>
<th>Sum of bookings</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>2018-01-01</td>
<td>1</td>
</tr>
<tr>
<td>123</td>
<td>2018-01-02</td>
<td>3</td>
</tr>
</tbody>
</table>

Data warehouse (human consumption)

<table>
<thead>
<tr>
<th>User</th>
<th>time</th>
<th>Sum of bookings</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>2018-01-01 11:15:24.142</td>
<td>Is it 0 or 1?</td>
</tr>
<tr>
<td>123</td>
<td>2018-01-02 18:15:24.142</td>
<td>Is it 2 or 3?</td>
</tr>
</tbody>
</table>
Limitations #1 of a standard warehouse for Machine Learning: Bound to daily accuracy

Data warehouse (human consumption)

<table>
<thead>
<tr>
<th>User</th>
<th>date</th>
<th>Sum of bookings</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>2018-01-01</td>
<td>1</td>
</tr>
<tr>
<td>123</td>
<td>2018-01-02</td>
<td>3</td>
</tr>
</tbody>
</table>

ML use case (machine consumption)

<table>
<thead>
<tr>
<th>User</th>
<th>time</th>
<th>Sum of bookings</th>
<th>Sum of bookings in past 12hrs</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>2018-01-01 11:15:24.14 2</td>
<td>Is it 0 or 1?</td>
<td>???</td>
</tr>
<tr>
<td>123</td>
<td>2018-01-02 18:15:24.14 2</td>
<td>Is it 2 or 3?</td>
<td>???</td>
</tr>
</tbody>
</table>
Why does this matter so much?

Just use the end of day value?
Why does this matter so much?

Just use the end of day value?

- Label leakage: “My model performs well on the test data, but not in production. I don’t know how to debug.”

Just use the start of day value?
Why does this matter so much?

Just use the end of day value?

- Label leakage: “My model performs well on the test data, but not in production. I don’t know how to debug.”

Just use the start of day value?

- You deprive your model of recent data
- Ex feature: number of searches in the past 24 hours.
If you missed that...

Point in time correctness is **important** and **hard**
We already have a production database
Why do we even need Zipline?

Production DB serves Airbnb.com just fine, surely it can handle “online” scoring traffic too?

1. Number of searches in the past 30 days? *Not in prod DB.*
2. Sum of bookings in past year. *airbnb.com goes down.*
We already have a production database

Why do we even need Zipline?

- We need the *exact same data* when *training* and *scoring*
“My model performs well on the test data, but not in production. I don’t know how to debug.”
If you missed that again...

Point in time correctness
Consistent data across training/scoring
+ Data quality and monitoring
+ Sharing and discovery
The Solution
Time Travel
Zipline puts a time machine on your data warehouse

Your data warehouse

Your data warehouse with Zipline
Training/Predicting Consistency Guaranteed

Zipline travels through time *and space*

Diagram:
- Production Data Stores → Zipline
- Data Warehouse → Zipline
- Zipline → Features → Model Scoring
- Zipline → Features → Model Training
Other requirements

- Monitoring
- Sharing
- Integrate smoothly with the bigger picture ML workflow (see bighead)
User Interface
Feature Sharing and Discovery

1. Searchable
2. Easily understandable
3. Find outliers
4. Identify transformations
5. Shopping cart
Features Definition (Time Travel)

1. Count the bookings
2. Average their values
3. 7d, 14d, 30d, 180d, 1y exact windows
Features Definition (Time Travel)

You define features in a way that allows point in time correct computations.

```json
source: {
  type: hive
  query: ""
  SELECT
    , guest as user
    , ts
    , value as value
    1 as count
  FROM bookings
  WHERE
    ds BETWEEN '{{ start_date }}' AND '{{ end_date }}'
"
  dependencies: [bookings]
}
features: {
  count: {
    doc: "Total bookings."
    column: count
    operation: sum
    windows: ["7d", "30d", "180d", "1y"]
  },
  avg_price: {
    doc: "Average booking value."
    column: price
    operation: avg
    windows: ["7d", "30d", "180d", "1y"]
  }
}
```
Features Definition (Time Travel)

- Now Zipline knows **how** to time travel that feature... What happens next?
- **Nothing!** Until someone asks for a point in time computation (what is the value for this user at this time).
- ZiplineSource API
User provides this

<table>
<thead>
<tr>
<th>user</th>
<th>listing</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>345</td>
<td>789</td>
<td>2018-01-02 01:45:55.891</td>
</tr>
</tbody>
</table>
Features Definition (Time Travel)

<table>
<thead>
<tr>
<th>user</th>
<th>listing</th>
<th>time</th>
<th>bookings_sum_7d</th>
<th>bookings_sum_14d</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>567</td>
<td>2018-01-01</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>234</td>
<td>678</td>
<td>2018-01-01</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>345</td>
<td>789</td>
<td>2018-01-02</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

User provides this

Zipline fills in this
Features Definition (iteration)

- Schema change? Don’t worry about it
- Bugfixed a feature? There’s an API for that

Incremental data only on old features

New feature gets efficient backfill
If you missed that again...

training set = f(features, primary keys, timestamps)
Zipline in the ML Iteration Workflow

What does this connection look like?

We’re here
Zipline in the ML Iteration Workflow

1. You build your Bighead model with a ZiplineSource
2. Configure it for daily training/scoring

Daily request to `get_{training/scoring}_dataframe()`
Zipline in the production workflow

- Bighead knows about your ZiplineSource
- ZiplineSource knows about your features
Zipline in the production workflow

- Scoring requests only require primary key vectors (not feature vectors)

{model=boking_likelihood, user=123, listing=345} 

{user=123, listing=345, ZiplineSource=bl_source}

{prob=0.8} 

{booking_count_7d=1, search_count_30d=3, ...}
If you missed that again...

Features are the same in all environments
Further Technical Details
Train/Predict data consistency
Lambda Architecture

Feature definition

Batch
*Daily

Streaming
*Continuous

KV Store

Zipline Client

Bighead

Production Traffic

Spark

Apache Flink
Time travel
How to do it efficiently

- Making this fast pays off
- Can get very expensive (many timestamps x many events)
- Skew is the enemy
- Caching partial aggregates can help
- Exact windows make it tricky
Time traveling on production DBs

Processing binlogs

- Daily dumps of production tables
- Lack of intra-day accuracy
- Zipline can ingest transaction logs
- Mutable events are tricky
Summary: Zipline is...

- Time travel
- Consistency
- Data quality and monitoring
- Searchable and sharable
- Integrated with end-to-end workflow
Drumroll...
Open Sourcing Q1 2019

Reach out to andrew.hoh@airbnb.com for more info
Questions
Appendix
ZiplineSource API

ZiplineSource is a python API with two primary user facing functions
1. Get **training** dataframe (arguments for sampling, time ranges, etc.)
2. Get **scoring** dataframe (arguments for sampling, time ranges, etc.)