Managing Production Data Prep Pipelines

Dr. Venkata Pingali
pingali@scribbledata.io
Scribble Data
Anecdotally only **2%** of models* are **productionized**!

* Most of these are in Python
Outline

- ML Infrastructure Overview
  - Why data prep is important
- Pipeline Structure and Challenges
- Where Does Time & Effort Go
- Required Capabilities
ML Infrastructure Overview
Production ML - Emerging Generic Architecture

GoJEK @ Spark AI Summit, April 2019

https://databricks.com/session/scaling-ride-hailing-with-machine-learning-on-mlflow
“Only a small fraction of real-world ML systems is composed of the ML code, as shown by the small black box in the middle. The required surrounding infrastructure is vast and complex.”

Paper from Google - NeurIPS 2015

“Only a small fraction of real-world ML systems is composed of the ML code, as shown by the small black box in the middle. The required surrounding infrastructure is vast and complex.”

Paper from Google - NeurIPS 2015

Implement, Operate, Audit, Access, Monitor Model Input and Output

Data Prep for Models - Nature

- Also called Feature Engineering
- Features are variables generated from data
  - Continuous process (Batch + Near Realtime + Realtime)
- Large in number (‘00s to ‘000s) & evolving
- Frequently executed

<table>
<thead>
<tr>
<th>Customer</th>
<th>SKU</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>17826162</td>
<td>0293192</td>
<td>Thai Dragon Fruit</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Customer</th>
<th>Premium</th>
<th>Imported</th>
</tr>
</thead>
<tbody>
<tr>
<td>17826162</td>
<td>15% of txns</td>
<td>5% of spend</td>
</tr>
</tbody>
</table>

Retail Customer (X GB) Features (~X/1000)
## Data Prep Pipeline Consumers

<table>
<thead>
<tr>
<th>Nature</th>
<th>Example</th>
<th>Timing</th>
<th>Users</th>
</tr>
</thead>
<tbody>
<tr>
<td>Models</td>
<td>Prediction</td>
<td>Batch or Realtime</td>
<td>Data savvy (python etc) Understand scale &amp; contracts</td>
</tr>
<tr>
<td>Automation</td>
<td>Reordering</td>
<td>Typically batch</td>
<td>Application developers (Java) Less flexibility &amp; More Contracts</td>
</tr>
<tr>
<td>Analysis</td>
<td>Segmentation</td>
<td>Adhoc</td>
<td>SQL-based tooling Explainability critical Availability and access focus Completeness nice to have</td>
</tr>
</tbody>
</table>
Nature of the Problem
Data Prep Pipelines - Structure

- Multiple, intersecting DAGs
- Can be broad and deep
- Continuous change
- Compute intensive
- Long execution times
- High volume of data

Sample: 50GB/day, 2M customers, 200 features, 10 pipelines, 4 hours execution time
Network+Time Makes Everything Hard

- Fluid systems that evolve over time
- Change propagates thru network
- Validation is always incomplete
- Hidden dependencies
- Impact can’t be undone

Sample: 6TB recompute and unknown $$ cost when profile is wrong!
Where does time go?
Where does time go? Stitching Systems

- Data sources, semantics, transformations span large space
- Rapid change in business need
- Natural fit for Python
- Flexibility and speed critical
  - Quick movement from test to prod
- Limited organizational resources
  - Robustness and productivity is critical

Python’s advantage: Interfacing (sqlalchemy, REST etc), computation (pandas etc), application framework (django etc)
Where does time go? xData

- Explanation for each feature & value
  - Three different languages/tests/contracts
  - Business/application consumers want to know
- Everyday, for every output
  - Many combinations - versions x runs x dependencies
- Reproducibility is a requirement
  - No explanation is credible without one

xData will enter conversations soon
Where does time go? Changes

- Changes - Involuntary (bugs) and Voluntary (functionality)
  - Different classes of features with different behaviors
- Thousands of lines of dense code
  - Corner cases + large volumes of data
- Correctness issues can be very expensive
  - Embarrassment and $$$
  - Laborious investigation, fixing code and data

Expect Python data management layer!
Where does time go? Resource Management

- Pandas is memory intensive: 5x rule
  - Continuous optimization and careful coding
  - Explicit memory management
- Implementing tradeoffs
  - Dev speed (D), Ops cost (O), Scalability (S)
- Need more high perf data structures (lists, dicts)
- Hidden Gem - Itertoolz
Required Capabilities
Implement: Provide Structure to Development

- Modular class structure
  - Flexible configuration
  - Pre/post exec validation
  - Pytest integration
  - Automatic documentation
  - Data quality checking
- Productivity enhancers
  - Feature specification DSL
  - Query/other templates
Operate: Flexible & Controlled Execution Management

- Parameterization
  - Easily extensible
  - Dynamic defaults
- Notifications w/ callouts
- Automated deployment
  - Coordinated across modules
  - Impact analysis of changes
- Service integration
  - Prefect, Netdata, Supervisor
Audit: Use and Manage Metadata Extensively

- Knowing what changes your data goes through
  - End-to-end auditability
    - All data and all runs
    - Metadata standardization
  - Discovery and reuse
    - Pipelines, modules
    - Lineage search
  - Early warning systems
    - Input/output quality checks
    - Note critical decisions
Access: Stable, Safe, Continuous Consumption

- Marketplace for data discovery
- Data contracts
- Isolation: Multi-tenant namespaces
  - File system, tables, S3
- Time: Versioned namespaces
  - Storage locations
  - Metadata
- Linked data and code
Takeaways

- Data prep required for all ML
  - Costly, cumbersome, error prone
  - Structure of the problem makes it hard
- Provide support in all stages of lifecycle
  - Implement, operate, audit, and consume
- xData will grow
  - Model correctness q’s are often data correctness q’s
THANK YOU FOR YOUR TIME