Train, Predict, Serve: How to go into production your Machine Learning model

Aki Ariga | Field Data Scientist
Aki Ariga

- Field Data Scientist at Cloudera
- Previously research engineer at Toshiba, Rails developer at Cookpad
- Contributor of sparklyr and creator of tabula-py
- Co-author of Machine Learning at work (in Japanese)
- Twitter: @chezou
- Linkedin: https://www.linkedin.com/in/aki-ariga/
- GitHub: https://github.com/chezou/
- Slideshare: https://www.slideshare.net/chezou
What does machine learning?
What machine learning does?

- **Classification**
  - Classify data into categories. *e.g.*) Spam detection, Image classification

- **Regression**
  - Predict continuous value. *e.g.*) Power consumption prediction

- **Clustering**
  - Grouping similar data into same group *e.g.*) Exploratory analysis

- **Anomaly detection**
  - Detect anomalous data. *e.g.*) Fraud detection

- **Recommendation**
  - Suggest contents which are interested in. *e.g.*) Amazon, Netflix

- **Reinforcement learning**
  - Learn strategy to maximize reward. *e.g.*) Alpha-Go, Self-driving car

- etc..
Example: Spam detection

Classify email SPAM or not
Training phase of Machine Learning pipeline (supervised learning)

Documents

Feature Extraction

e.g.) Extract:
  from Image: Array of RGB
  from Text: word frequency

Feature vector

[0, 1, 0, 2.5, 0, -1, ...]
[1, 0.5, 0.1, -2, 3, 2, ...]
[1, 0, 1.0, 1.1, 0, 0, ...]

Model Training

Algorithm & parameters

Logistic Regression, SVM, Random Forest, NN...

w1=1, w2=-1, w3=0 ...

Predictive model
Sample data science/machine learning workflow
From data to exploration to action

Data Engineering
- Acquisition
- Processing
- Governance

Data Science (Exploratory)
- Data Wrangling
- Visualization and Analysis
- Model Training & Testing

Production (Operational)
- Reports, Dashboards
- Production Data Pipelines
- Online Scoring
- Batch Scoring
- Serving

Data → Models → Predictions → Business value
What is “production” of a ML system?
Production of ML systems

- Reports
- Dashboards
- Scoring
Patterns of ML scoring systems
Patterns of ML scoring systems

1. Train by batch, **predict on the fly**, serve via **REST API**
2. Train by batch, **predict by batch**, serve through the **shared DB**
3. Train, predict, serve **by streaming**
4. Train by batch, predict **on mobile app**
Patterns of ML scoring systems

1. Train by batch, **predict on the fly**, serve via **REST API**
2. Train by batch, **predict by batch**, serve through the **shared DB**
3. Train, predict, serve **by streaming**
4. Train by batch, predict **on mobile app**
Pattern 1: Train by batch, predict on the fly, serve via REST API
What is the deployment target?

- **Web Application**
  - DB
  - Activity log/Contents data
  - Prediction result
  - User ID/Item ID

- **Production**
  - REST API
  - Validated Model
  - Training result
  - Execute training
  - Feature
  - Extract feature

- **Development**
  - Validated Model
  - Training result
  - Feature
  - Extract feature

- **Deploy**
  - Model

**Deployment Targets:**
- **Production**
- **Development**

**Feature Components:**
- User ID/Item ID
- Activity log/Contents data
- Prediction result
- REST API
- Training result
- Execute training
- Extract feature
Pattern 1-a: Training phase

Train using activity logs in the DB and store model into the storage.
Pattern 1-b,c: Prediction & serving phase

Predict from logs/contents triggered by REST API.
Example architecture 1: PMML + OpenScoring

**Predicting & serving layer**
- Request to predict
- Prediction results
- Extract feature & Predict

**Model building layer**
- CDSW
- Trained Model
- Updated model
- Extract feature & Train/update model
- Activity log
- HDFS
- Export model as PMML

*PMML* (Predictive Model Markup Language)

*Openscoring.io*
Example architecture 2: Docker based API Server

Predicting & serving layer
- Extract feature & Predict
- Load model
- Request to predict
- Prediction results

Model building layer
- Trained Model
- Updated model
- Pack the runtime env with Docker
- Extract feature & Train/update model
- Save model on object storage
- Activity log
- HDFS

Pack the runtime env with Docker
- CDSW

Cloudera
Demo:
Docker based prediction API
https://github.com/chezou/cdsw-serve-docker
Pattern 1: Train by batch, predict on the fly, serve via REST API

- **Pros**
  - Able to choose a **different system configuration** for the front-end web application and the ML system.
  - Choose favorite languages for both systems
  - Use different machine spec for both systems
  - Deploy independently
  - Able to serve with **low latency**
  - **Rapid prototyping** with flexible ML libraries
  - Easy to **A/B testing**
  - Prevent reimplementation of ML algorithm

- **Cons**
  - **Complex to implement** scalable API server
  - Unable to use slow algorithm
Patterns of ML scoring systems

1. Train by batch, **predict on the fly**, serve via **REST API**
2. Train by batch, **predict by batch**, serve through the **shared DB**
3. Train, predict, serve **by streaming**
4. Train by batch, predict **on mobile app**
Pattern 2: Train by batch, predict by batch, serve through the shared DB
Pattern 2-a: Training phase

Training using activity logs in the DB and store model into the storage.
Predict from contents data in the DB and store prediction results.

Pattern 2-b: Prediction phase

- **Web Application**
- **Batch System**
  - Training Batch
  - Training result
  - Execute training
  - Feature
  - Extract feature
- **DB**
  - Activity log/Contents data
  - Prediction result
  - Prediction Batch
  - Serve prediction
- **Trained Model**
Pattern 2-c: Serving phase

Serve prediction results stored in the shared DB.
Example architecture: Serving by HBase/Kudu

Serving layer

Kudu/HBase

Prediction results

Activity log

Model building & predicting layer

Extract feature & Predict

Load trained model

Updated model

HDFS

Trained Model

Historical data

Extract feature & Train/update model

Activity log

CDSW

 historical data
Pattern 2: Train by batch, predict by batch, serve through the shared DB

- **Pros**
  - Able to choose a different system configuration for the front-end web application and the batch system.
    - Chose favorite languages for both systems
    - Use different machine spec for both systems
    - Deploy independently
  - Able to use **slow and complex algorithm**
  - **Easy to manage**: versioning model/prediction results
  - Rapid prototyping with flexible ML libraries
  - Prevent reimplementation of ML algorithm

- **Cons**
  - Unable to predict triggered with certain event (e.g. PV, purchase)
  - Unable to prevent **time lag from prediction to serving**
Patterns of ML scoring systems

1. Train by batch, **predict on the fly**, serve via **REST API**
2. Train by batch, **predict by batch**, serve through the **shared DB**
3. Train, predict, serve **by streaming**
4. Train by batch, predict **on mobile app**
Pattern 3: Train, predict, serve by streaming

- Querying for prediction
- Showing or sending alerts
- This component may work with message queue like Kafka

Web Application

Message queue (e.g. Kafka)

Log data

Recent log data

Prediction results

Prediction results

Extract feature

Feature

Model updates

Train & Predict

Trained Model

Stream-based ML System (e.g. Spark Streaming)
Example architecture: Lambda architecture with Oryx
Pattern 3: Train, predict, serve by streaming

- Pros
  - Predict and serve with low latency
  - Update model interactively
  - Able to predict triggered with certain event (e.g. purchase, PV)
  - **Strong capability for streamed data**, especially for anomaly detection and recommendation

- Cons
  - **Complex to manage** model and system
    - Hard to versioning models
Patterns of ML scoring systems

1. Train by batch, predict on the fly, serve via REST API
2. Train by batch, predict by batch, serve through the shared DB
3. Train, predict, serve by streaming
4. Train by batch, predict on mobile app
Pattern 4: Train by batch, predict on a mobile app

Mobile Application

- Request for prediction
- Extract feature
- Activity logs/Contents data
- Trained Model
- Prediction result

Batch System

- Execute training
- Training result
- Feature
- Extract feature
- Activity log/Contents data
- DB
- Convert model
Example architecture: Serving on a mobile app

Predicting & serving layer

- Storage in a smart phone
- Extract feature & Predict
- Request to predict
- Prediction results

Load model

Model building layer

- Trained Model
- Updated model
- Extract feature & Train/update model
- Activity log
- HDFS

Convert model to TFLite/CoreML
# 4 patterns Comparison

<table>
<thead>
<tr>
<th></th>
<th>Pattern 1 (REST API)</th>
<th>Pattern 2 (Shared DB)</th>
<th>Pattern 3 (Streaming)</th>
<th>Pattern 4 (Mobile app)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Training</strong></td>
<td>by batch</td>
<td>by batch</td>
<td>by streaming</td>
<td>by batch</td>
</tr>
<tr>
<td><strong>Prediction</strong></td>
<td>on the fly</td>
<td>by batch</td>
<td>by streaming</td>
<td>on the fly</td>
</tr>
<tr>
<td><strong>Prediction result delivery</strong></td>
<td>via REST API</td>
<td>through the shared DB</td>
<td>by streaming via MQ</td>
<td>via in-process API on mobile</td>
</tr>
<tr>
<td><strong>Latency for prediction from getting new data</strong></td>
<td>So so</td>
<td>So so ~ Long</td>
<td>Very low</td>
<td>Low</td>
</tr>
<tr>
<td><strong>Required time to predict</strong></td>
<td>Short</td>
<td>Long</td>
<td>Short</td>
<td>Short</td>
</tr>
<tr>
<td><strong>Tight/loose coupling with app</strong></td>
<td>Loose</td>
<td>Loose</td>
<td>Loose</td>
<td>Tight</td>
</tr>
<tr>
<td><strong>Dependency of languages</strong></td>
<td>Independent</td>
<td>Independent</td>
<td>Independent</td>
<td>Depends on frameworks</td>
</tr>
<tr>
<td><strong>System management difficulty</strong></td>
<td>So so</td>
<td>Easy</td>
<td>Very Hard</td>
<td>So so</td>
</tr>
</tbody>
</table>
Good to read/watch for related topics

- Lucy Park, “My model has higher BLEU, can I ship it? The Joel Test for machine learning systems”, ACML-MLAIP, 2017
Monitor model performance with recurring model update with CI

Model versioning & monitoring

- Monitor model performance with recurring model update with CI
Thank you

Aki Ariga
aki@cloudera.com
Appendix
How does CDSW help for each deployment pattern?

- Reports
  - Send Email via CDSW Job
- Dashboards
  - NA (Web app can be run while container is running)
- Scoring
  - Pattern 1
    - Export model with PMML serve OpenScoring/Use MLeap etc...
    - Create API server with Web framework and pack the environment with Docker
    - Need to manage API server by users
  - Pattern 2
    - Store prediction results into HBase/Kudu/RDB from CDSW
  - Pattern 3
    - You can develop and deploy jobs working on the same cluster
  - Pattern 4
    - Convert to TFLite/CoreML model and bring the model on your app