Productionizing Spark ML Pipelines with the Portable Format for Analytics

—

Nick Pentreath
Principal Engineer, IBM

@MLnick

About

@MLnick on Twitter & Github

Principal Engineer, IBM

Spark Technology Center & Cognitive Open Technologies

Machine Learning & AI

Apache Spark committer & PMC

Author of *Machine Learning with Spark*

Various conferences & meetups

Agenda

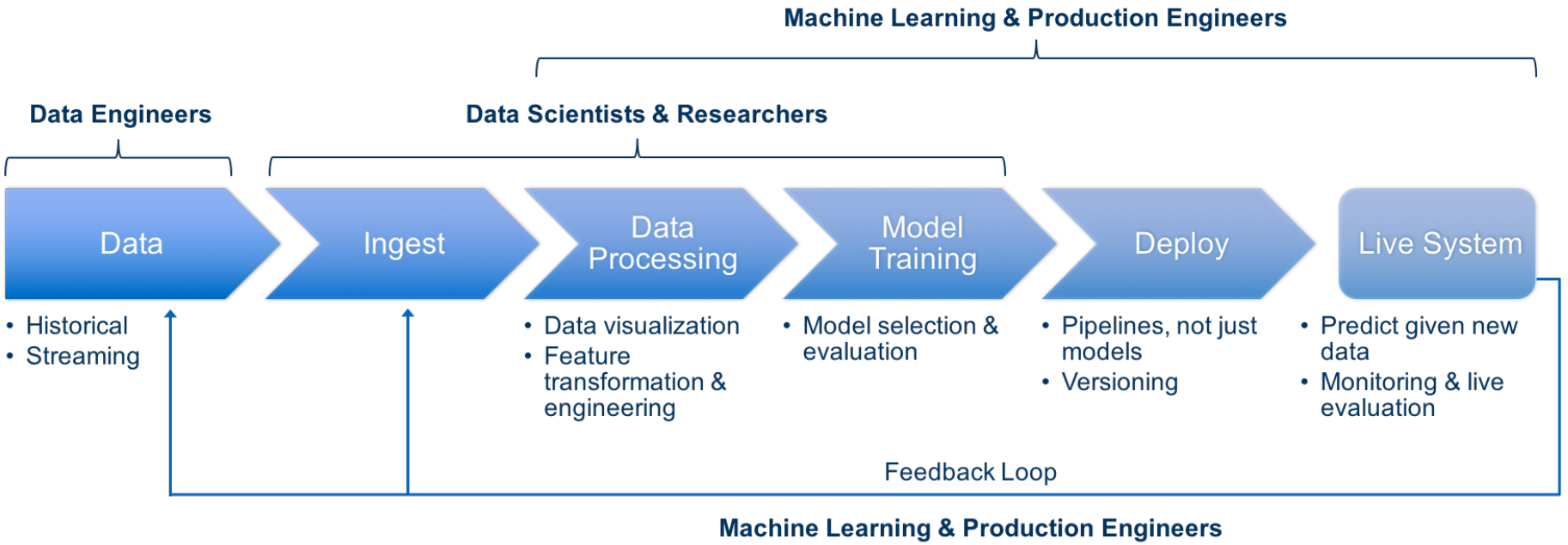
The Machine Learning Workflow

Challenges of ML Deployment

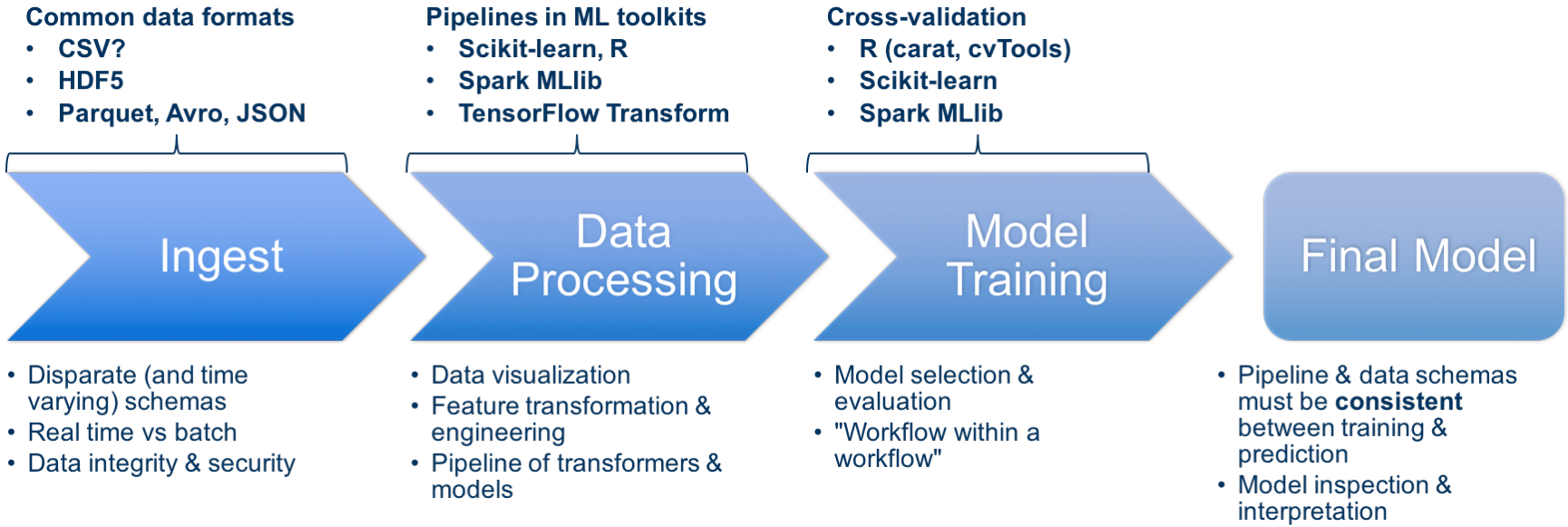
Open Standards for ML Deployment

Summary and Future Directions

ML workflow spans teams …



… and tools


Common data formats
• CSV?
• HDF5
• Parquet, Avro, JSON
Pipelines in ML toolkits
• Scikit-learn, R
• Spark MLlib
• TensorFlow Transform
Cross-validation
• R (carat, cvTools)
• Scikit-learn
• Spark MLlib
Ingest
Data Processing
Model Training
Final Model
• Disparate (and time varying) schemas
• Real time vs batch
• Data integrity & security
• Data visualization
• Feature transformation & engineering
• Pipeline of transformers & models
• Model selection & evaluation
• "Workflow within a workflow"
• Pipeline & data schemas must be consistent between training & prediction
• Model inspection & interpretation

Challenges

- Need to manage and bridge many different:
  - Languages
  - Frameworks
  - Dependencies
  - Versions
- Performance characteristics can be highly variable across these dimensions
- Lack of standardization leads to custom solutions
- Where standards exist, limitations lead to custom extensions, eliminating the benefits

- Note:
  - “Deployment” in this context is different from “deployment” in the purely devops sense
  - e.g. containers are useful but incomplete solutions

Challenges specific to Spark

- Tight coupling to Spark runtime
  - Introduces complex dependencies
  - Managing version & compatibility issues
- Scoring models in Spark is **slow**
  - Overhead of DataFrames, especially query planning
  - Overhead of task scheduling, even locally
  - Optimized for batch scoring (includes streaming “micro-batch” settings)
- Spark is **not suitable** for real-time scoring (< few 100ms latency)

- Currently, in order to use trained models (pipelines) outside of Spark, users must:
  - Write *custom* readers for Spark’s native format; or
  - Create their own *custom* format; or
  - Export to a standard format (not currently supported within Spark, hence requiring a *custom* solution)
- To score models outside of Spark, users must also write their own *custom* translation between Spark ML components and an existing (or custom) ML library

Everything is custom!

Portable Format for Analytics

- PFA is being championed by the Data Mining Group (IBM is a founding member)
- DMG previously created PMML (Predictive Model Markup Language), arguably the only viable open standard currently
  - PMML has many limitations
  - PFA was created specifically to address these shortcomings

- PFA consists of:
  - JSON serialization format
  - AVRO schemas for data types
  - Encodes functions (*actions*) that are applied to inputs to create outputs with a set of built-in functions and language constructs (e.g. control-flow, conditionals)
  - Essentially a *mini functional math language + schema specification*
- Type and function system means PFA can be fully & statically verified on load and run by any compliant execution engine
- => true portability across languages, frameworks, run times and versions

Portable Format for Analytics

- Example – multi-class logistic regression
- Specify input and output types using Avro schemas

```
{
  "name":"logistic-regression-model",
  "input":{
    "type":{
      "type":"array",
      "items":"double"
    }
  },
  "output":{
    "type":"double"
  },
```

- Specify the *action* to perform (typically on input)

```
"action":[
  {
    "a.argmax": [
      {
        "m.link.softmax": [
          {
            "model.reg.linear": [
              "input",
              {
                "cell": "model"
              }
            ]
          }
        ]
      }
    ]
  }
],
```

Aardpfark

- PFA export for Spark ML pipelines
  - `aardpfark-core` – Scala DSL for creating PFA documents
    - `avro4s` to generate schemas from case classes; `json4s` to serialize PFA document to JSON
  - `aardpfark-sparkml` – uses DSL to export Spark ML components and pipelines to PFA
- Coverage
  - Almost all predictors (ML models)
  - Most feature transformers
  - Pipeline support
- Missing features
  - Generic vector support (mixed dense/sparse)

```
val input = StringExpr("input")
val cell = Cell[LinearModelData](
  DenseLinearModelData(const, coeff)
)
val modelCell = NamedCell("model", cell)
val action = a.argmax(
  m.link.softmax(
    model.reg.linear(input, modelCell.ref)
  )
)

val pfa: PFADocument = PFABuilder()
  .withName("logistic-regression-model")
  .withInput[Seq[Double]]
  .withOutput[Double]
  .withCell(modelCell)
  .withAction(action)
  .pfa
```


Similar projects

- MLeap
  - Created by Combust.ML, a startup focused on ML model serving
  - Model interchange format in JSON / Protobuf
  - Components implemented in Scala code
  - Initially focused on Spark ML. Offers almost complete support for Spark ML components
  - Recently added some sklearn; working on TensorFlow
  - “Open” format, but not a “standard”
  - No concept of well-defined *operators / functions*
  - Effectively forces a tight coupling between versions of model *producer / consumer*

- Open Neural Network Exchange (ONNX)
  - Championed by Facebook & Microsoft
  - Protobuf serialization format
  - Describes computation graph (including operators)
    - In this way it is similar to PFA in the sense that the serialized graph is “self-describing”
  - More focused on Deep Learning / tensor operations
  - No or poor support for more “traditional” ML or language constructs (currently)
    - Tree-based models & ensembles
    - String / categorical processing
    - Control flow
    - Intermediate variables

Summary

- PFA provides an *open standard* for serialization and deployment of analytic workflows
  - Portability across languages, frameworks, runtimes and versions
  - Execution environment is independent of the producer (R, scikit-learn, Spark ML, weka, etc)
- Solves a significant pain point for the Spark ML ecosystem
- Also benefits the wider ML ecosystem (e.g. many currently use PMML for exporting models from R, scikit-learn, XGBoost, LightGBM, etc)
- However there are risks
  - PFA is still young and needs to gain adoption
  - Performance in production, at scale, is relatively untested
  - What about Deep Learning / comparison to ONNX?
  - Limitations of PFA
  - A standard can move slowly in terms of new features, fixes and enhancements

Future directions

- Open source release of Aardpfark
  - Initially focused on Spark ML pipelines
  - Later add support for scikit-learn pipelines, XGBoost, LightGBM, etc
  - (Support for many R models exist already in the Hadrian project)
- Performance testing in progress vs Spark & MLeap
- More automated translation (Scala -> PFA, ASTs etc)
- Propose improvements to PFA
  - Generic vector (tensor) support
  - Less cumbersome schema definitions
  - Performance improvements to scoring engine
- PFA for Deep Learning?
  - Comparing to ONNX and other emerging standards
  - Requires all the various DL-specific operators
  - Requires tensor schema and better tensor support built-in to the PFA spec
  - Should have GPU support

Thank you!

Nick Pentreath
Principal Engineer
—
nickp@za.ibm.com
@MLnick
ibm.com

Links & References

Portable Format for Analytics

PMML

Spark MLlib – Saving and Loading Pipelines

Hadrian – Reference Implementation of PFA Engines for JVM, Python, R

MLeap

Open Neural Network Exchange