Productionizing Machine Learning Pipelines with PFA

Nick Pentreath
Principal Engineer

@MLnick

IBM **Developer**

About

@MLnick on Twitter & Github

Principal Engineer, IBM

CODAIT - Center for Open-Source Data & AI Technologies

Machine Learning & AI

Apache Spark committer & PMC

Author of *Machine Learning with Spark*

Various conferences & meetups

Center for Open Source Data and AI Technologies

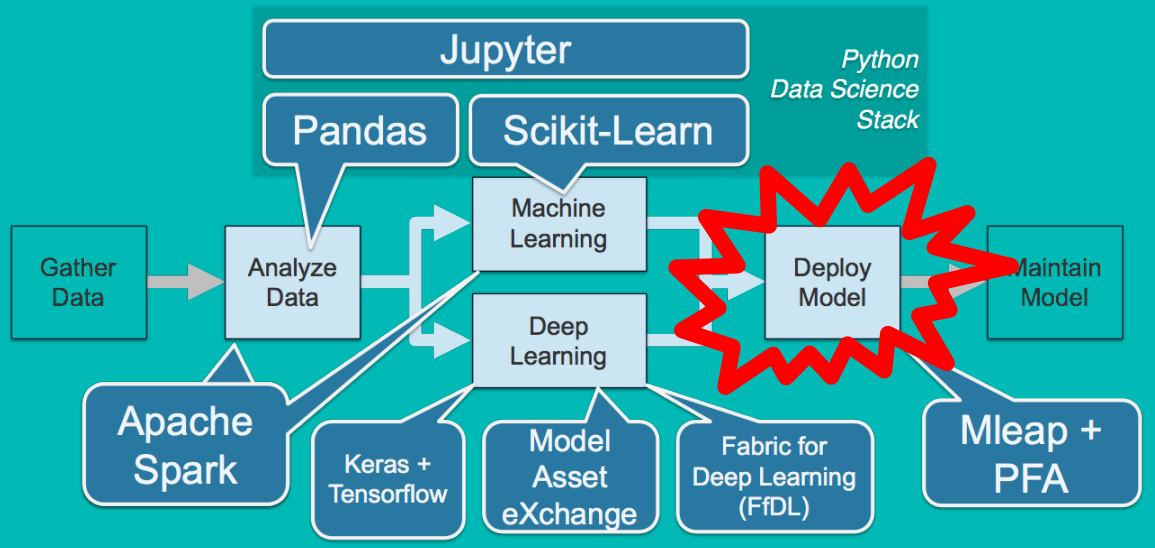
CODAIT

codait.org

CODAIT aims to make AI solutions dramatically easier to create, deploy, and manage in the enterprise

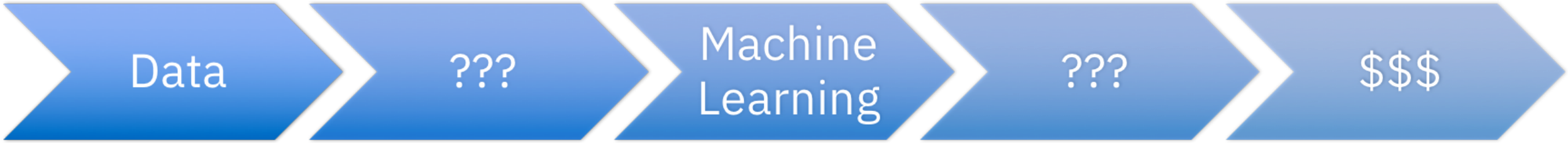
Relaunch of the Spark Technology Center (STC) to reflect expanded mission

Improving Enterprise AI Lifecycle in Open Source

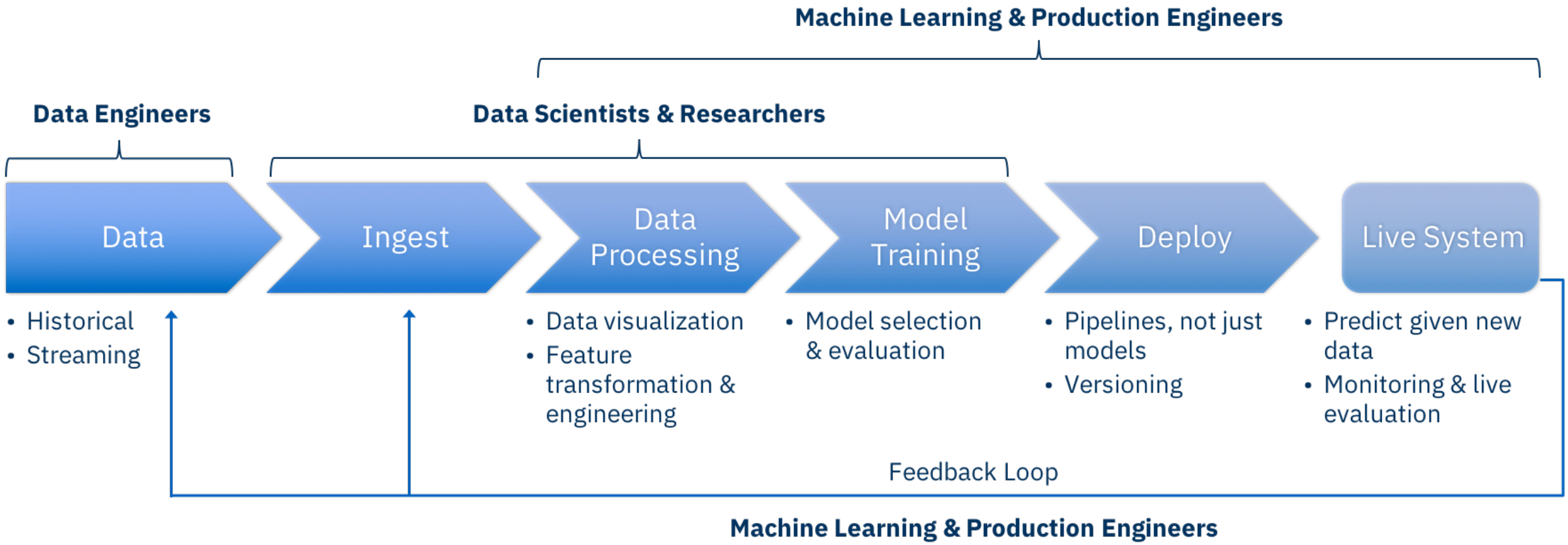


The Machine Learning Workflow

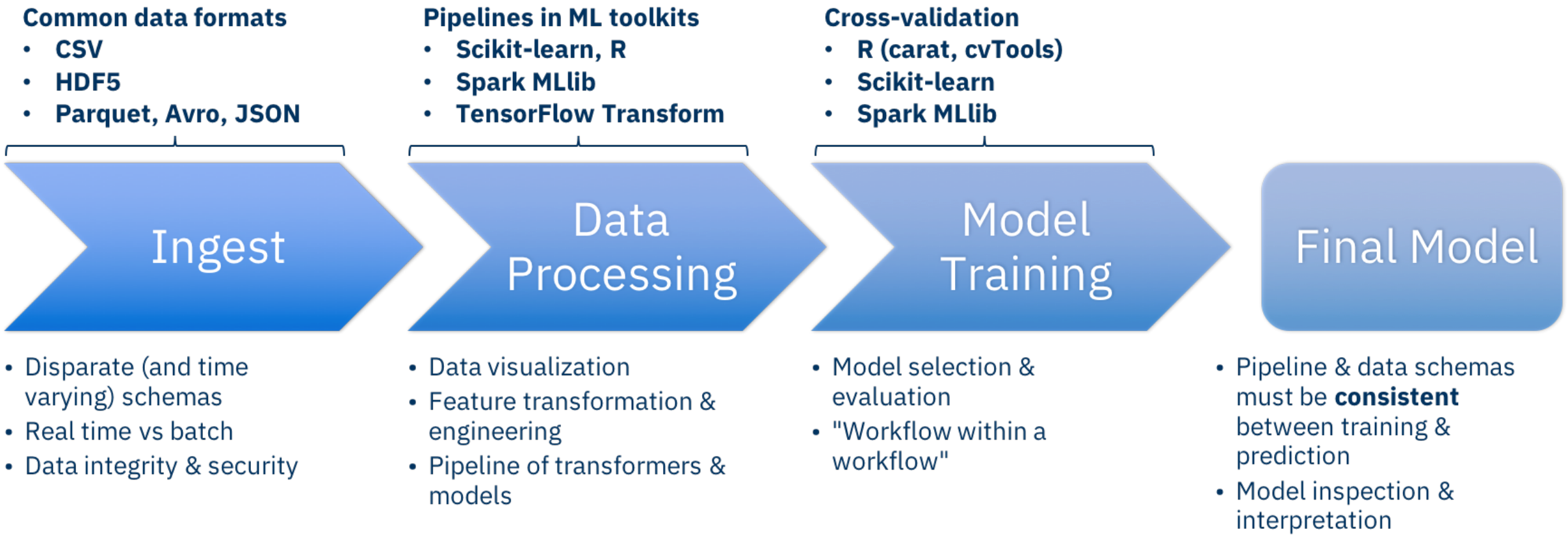
Perception



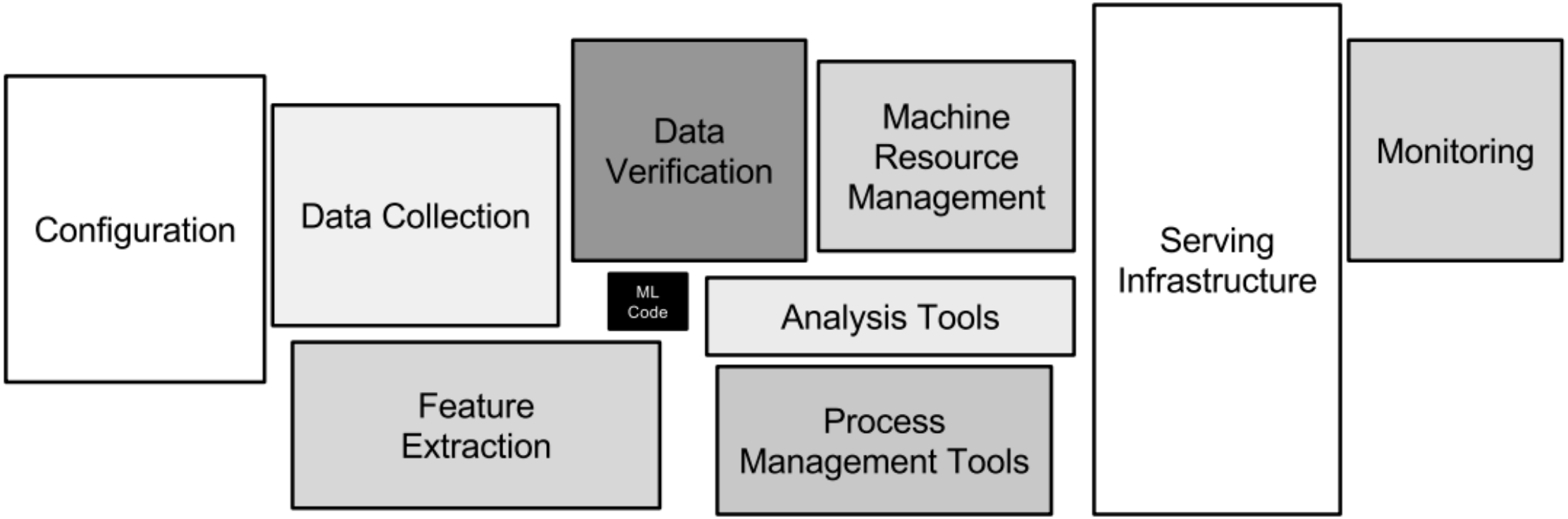
In reality the workflow spans teams ...



… 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

... and is a small (but critical!) piece of the puzzle



*Source: Hidden Technical Debt in Machine Learning Systems

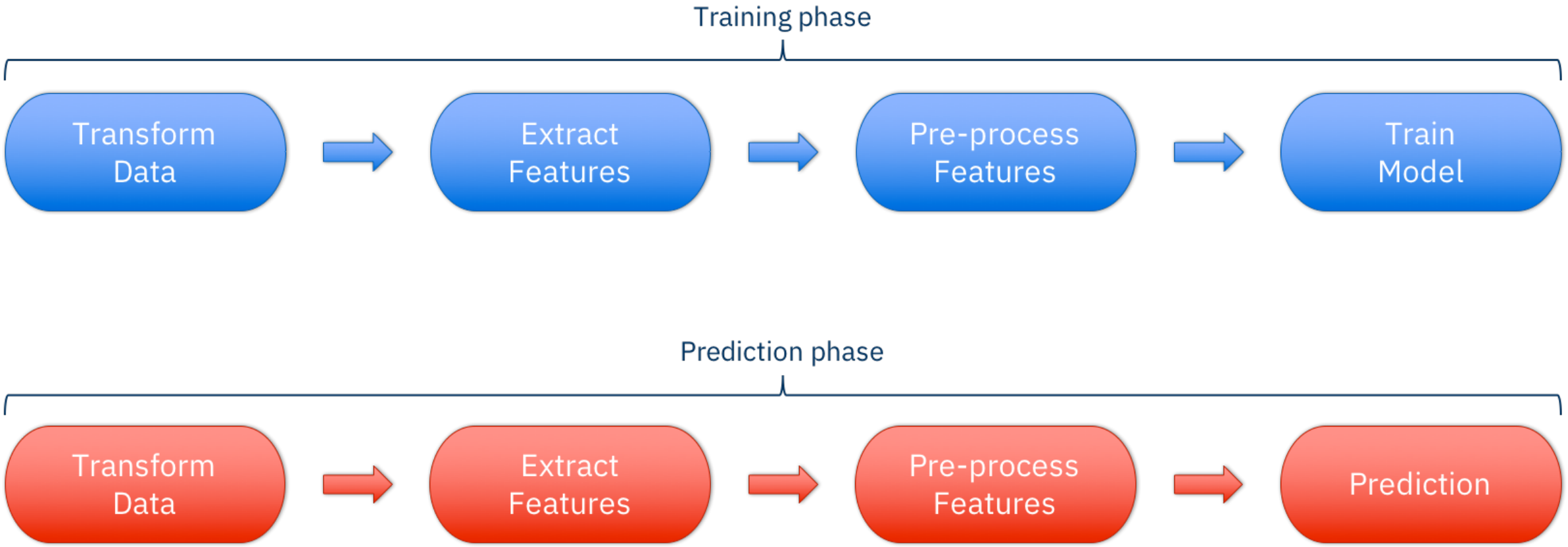
Machine Learning Deployment

What, Where, How?

- What are you deploying?
  - What is a “model”?
- Where are you deploying?
  - Target environment
  - Batch, streaming, real-time?
- How are you deploying?
  - “devops” deployment mechanism
  - Serving framework

We will talk mostly about the what

What is a “model”?



Pipelines, not Models

- Deploying just the model part of the workflow is not enough
- Entire pipeline must be deployed
  - Data transform
  - Feature extraction & pre-processing
  - ML model itself
  - Prediction transformation
- Technically even ETL is part of the pipeline!

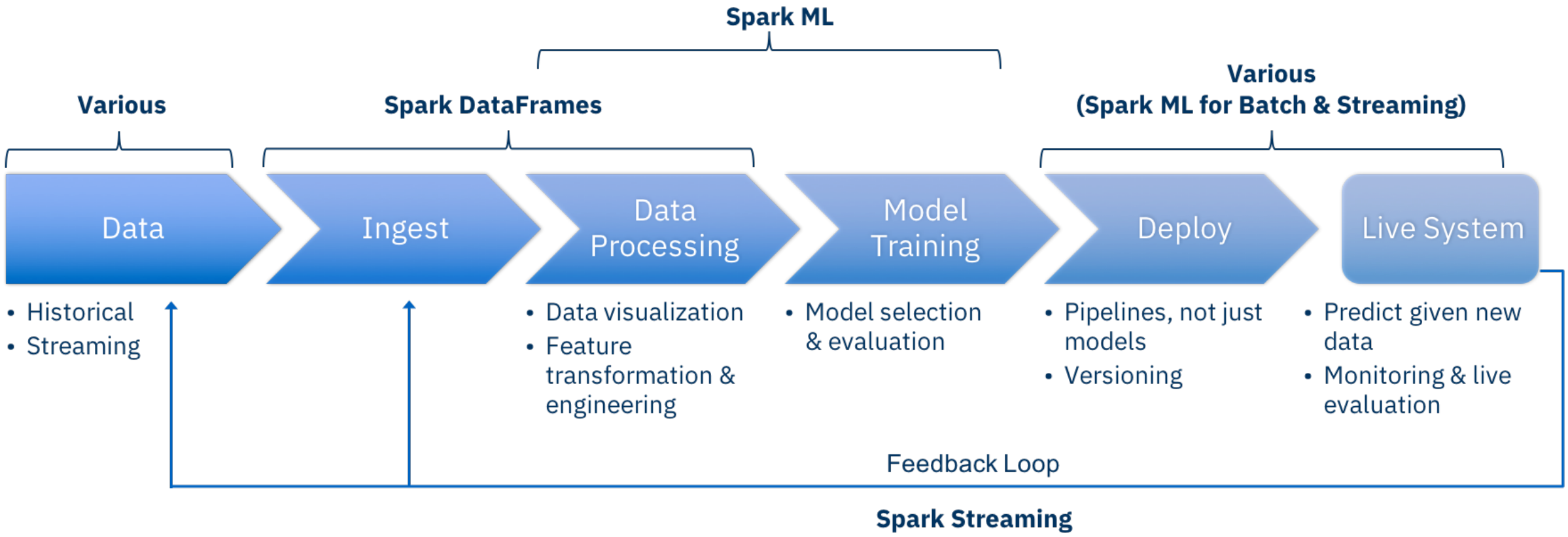
- Pipelines in frameworks
  - scikit-learn
  - Spark ML pipelines
  - TensorFlow Transform
  - pipeliner (R)

Challenges

- Need to manage and bridge many different:
  - Languages - Python, R, Notebooks, Scala / Java / C
  - Frameworks – too many to count!
  - Dependencies
  - Versions
- Performance characteristics can be highly variable across these dimensions
- Friction between teams
  - Data scientists & researchers – latest & greatest
  - Production – stability, control, minimize changes, performance
  - Business – metrics, business impact, product must always work!

- Proliferation of formats
  - Open source, open standard: PMML, PFA, ONNX
  - Open-source, non-standard: MLeap, Spark, TensorFlow, Keras, PyTorch, Caffe, …
  - Proprietary formats: lock-in, not portable
- Lack of standardization leads to custom solutions
- Where standards exist, limitations lead to custom extensions, eliminating the benefits

Spark solves many problems ...



… but introduces additional challenges

- 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 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 (limited support => custom solution)
- Scoring outside of Spark also requires custom translation layer between Spark and another ML library

Everything is custom!

Containers for ML Deployment

Containers for ML Deployment

- Container-based deployment has significant benefits
  - Repeatability
  - Ease of configuration
  - Separation of concerns – focus on what, not how
  - Allow data scientists & researchers to use their language / framework of choice
  - Container frameworks take care of (certain) monitoring, fault tolerance, HA, etc.

- But …
  - What goes in the container is most important
  - Performance can be highly variable across language, framework, version
  - Requires devops knowledge, CI / deployment pipelines, good practices
  - Does not solve the issue of standardization
    - Formats
    - APIs exposed
  - A serving framework is still required on top

The Portable Format for Analytics

Overview

- 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
- => portability across languages, frameworks, run times and versions

A Simple Example

- 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"
              }
            ]
          }
        ]
      }
    ]
  }
],
```

Managing State

- Data storage specified by *cells*
  - A cell is a named value acting as a global variable
  - Typically used to store state (such as model coefficients, vocabulary mappings, etc)
  - Types specified with Avro schemas
  - Cell values are mutable within an action, but immutable between action executions of a given PFA document
- Persistent storage specified by *pools*
  - Closer in concept to a database
  - Pools values are mutable across action executions

Other Features

- Special forms
  - Control structure – conditionals & loops
  - Creating and manipulating local variables
  - User-defined functions including lambdas
  - Casts
  - Null checks
  - (Very) basic try-catch, user-defined errors and logs
- Comprehensive built-in function library
  - Math, strings, arrays, maps, stats, linear algebra
  - Built-in support for some common models - decision tree, clustering, linear models

Current Status

- Reference implementations
  - Hadrian project by Open Data Group
  - Covers PFA export / DSL in Python, R
  - Covers scoring for PFA in JVM, Python, R
- What does PFA do well?
  - Type system
  - Flexibility & composability – functional approach
  - User-defined functions
  - Control flow
  - Strong support for traditional ML operations
- Major missing features / limitations
  - No built-in support for mixed dense/sparse vectors
  - No built-in support for generic tensors
  - No built-in functions for typical Deep Learning models (e.g. CNN, RNN)
  - No support / awareness of GPU
- Open questions
  - Industry usage and adoption
  - Performance and scalability

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

```
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
```

Aardpfark - Challenges

- Spark ML Model has no schema knowledge
  - E.g. `Binarizer` can operate on numeric or vector columns
  - Need to use Avro union types for standalone PFA components and handle all cases in the action logic

```
`type`()
.unionOf().array().items().doubleType().and()
.doubleType().endUnion()

Cast(inputExpr, Seq(asDouble, asArray))
```

- Combining components into a pipeline
  - Trying to match Spark's DataFrame-based input/output behavior (typically appending columns)
  - Each component is wrapped as a user-defined function in the PFA document
  - Current approach mimics passing a Row (i.e. Avro record) from function to function, adding fields

- Missing features in PFA
  - Generic vector support (mixed dense/sparse)

Aardpfark is open-source!

- Coverage
  - Almost all predictors (ML models)
  - Many feature transformers
  - Pipeline support (still needs work)
  - Equivalence tests Spark <-> PFA
  - Tests for core Scala DSL

- Need your help!
  - Finish implementing components
  - Improve pipeline support
  - Complete Scala DSL for PFA
  - Python support
  - Tests, docs, testing it out!

https://github.com/CODAIT/aardpfark

https://github.com/salesforce/TransmogrifAI/tree/master/local

Related Open Standards

PMML

- Data Mining Group (DMG)
- Model interchange format in XML with operators
- Widely used and supported; open standard
- Spark support lacking natively but 3rd party projects available: jpmml-sparkml
  - Comprehensive support for Spark ML components (perhaps surprisingly!)
  - Watch SPARK-11237
- Other exporters include scikit-learn, R, XGBoost and LightGBM

- Shortcomings
  - Cannot represent arbitrary programs / analytic applications
  - Flexibility comes from custom plugins => lose benefits of standardization

MLeap

- Created by Combust.ML, a startup focused on ML model serving
- Model interchange format in JSON / Protobuf
- Components implemented in Scala code
- Good performance
- Initially focused on Spark ML. Offers almost complete support for Spark ML components
- Recently added some sklearn; working on TensorFlow

- Shortcomings
  - “Open” format, but not a “standard”
  - No concept of well-defined operators / functions
  - Effectively forces a tight coupling between versions of model producer / consumer
  - Must implement custom components in Scala
  - Impossible to statically verify serialized model

Open Neural Network Exchange (ONNX)

- Championed by Facebook & Microsoft
- Protobuf serialization format
- Describes computation graph (including operators)
  - In this way the serialized graph is “self-describing” similarly to PFA
- More focused on Deep Learning / tensor operations
- Will be baked into PyTorch 1.0.0 / Caffe2 as the serialization & interchange format

- Shortcomings
  - No or poor support for more “traditional” ML or language constructs (currently)
    - Tree-based models & ensembles
    - String / categorical processing
    - Control flow
    - Intermediate variables

Neural Network Exchange Format (NNEF)

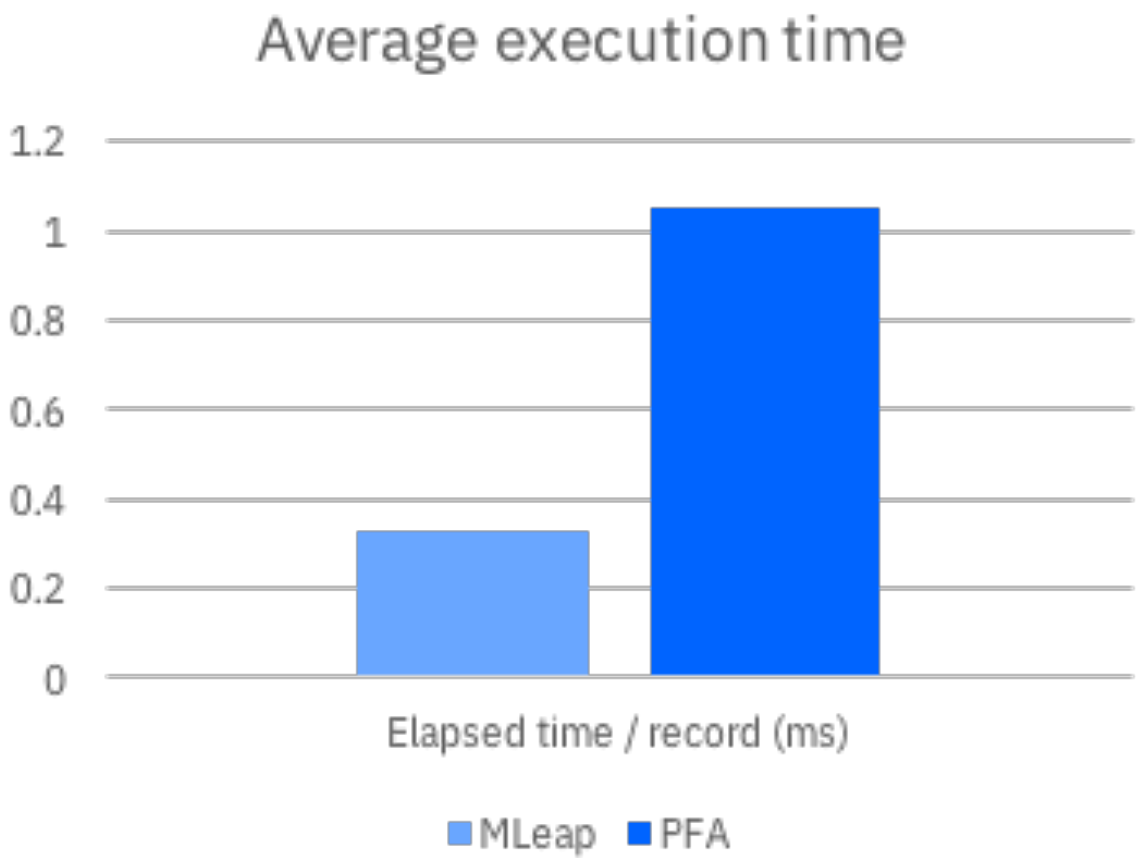
- Championed by Khronos Group
- Dual format
  - Network structure file: describes computation graph & operators
  - Network data file: binary format for parameters
- More focused on Deep Learning / tensor operations
- Supports compound operators (user-defined functions)

- Shortcomings
  - No or poor support for more “traditional” ML or language constructs (currently)
    - Tree-based models & ensembles
    - String / categorical processing
    - Control flow
    - Intermediate variables

Performance

Scoring Performance Comparison

- Comparing scoring performance of PFA with Spark and MLeap
- PFA uses Hadrian reference implementation for JVM
- Test dataset of ~80,000 records
  - String indexing of 47 categorical columns
  - Vector assembling the 47 categorical indices together with 27 numerical columns
  - Linear regression predictor
- *Note: Spark time is 1.9s / record (1901ms) - not shown on the chart*



Wrapping Up

Summary

- PFA provides an open standard for serialization and deployment of analytic workflows
  - True portability across languages, frameworks, runtimes and versions
  - Execution environment is independent of the producer
- Solves a significant pain point for the deployment of ML pipelines and 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
  - Tests indicate PFA reference engines need some work on robustness and performance
  - What about Deep Learning (e.g. ONNX)?
  - Limitations of PFA
  - A standard can move slowly in terms of new features, fixes and enhancements

Future directions

- Extend our work in Aardpfark
  - Initial focus on Spark ML
  - Later add support for scikit-learn pipelines, XGBoost, LightGBM, etc
- Performance testing & improvements
- 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
  - Better suited for the more general pre-processing steps of DL pipelines
  - Requires supporting DL-specific operators
  - Requires standardized tensor schema and support for tensors in PFA function library
  - GPU support

Thank you!

http://ibm.biz/model-exchange

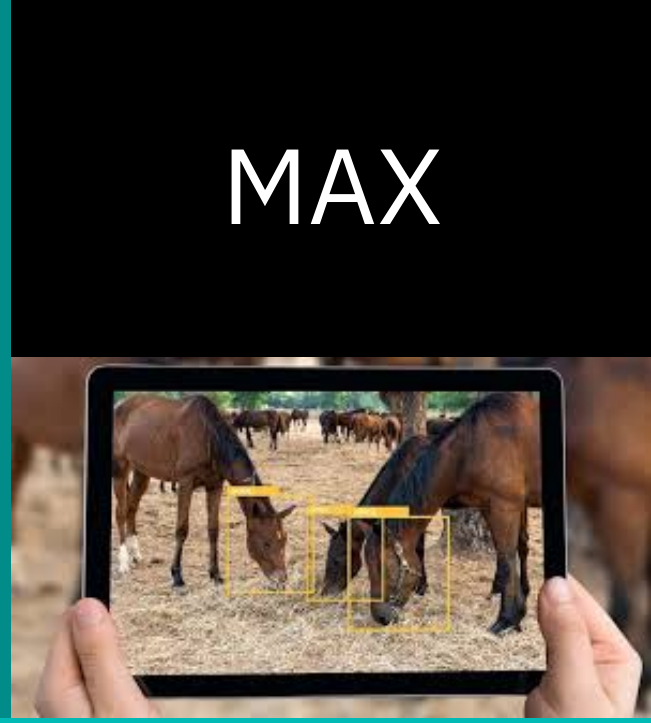
http://ibm.biz/max-developers

codait.org

twitter.com/MLnick

github.com/MLnick

developer.ibm.com



Sign up for IBM Cloud and try Watson Studio!

https://ibm.biz/BdYVQS

Links & References

Portable Format for Analytics

PMML

Spark MLlib – Saving and Loading Pipelines

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

jpmml-sparkml

MLeap

Open Neural Network Exchange