Deploying Machine Learning Models in Practice

Nick Pentreath
Principal Engineer

@MLnick

IBM CODE

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

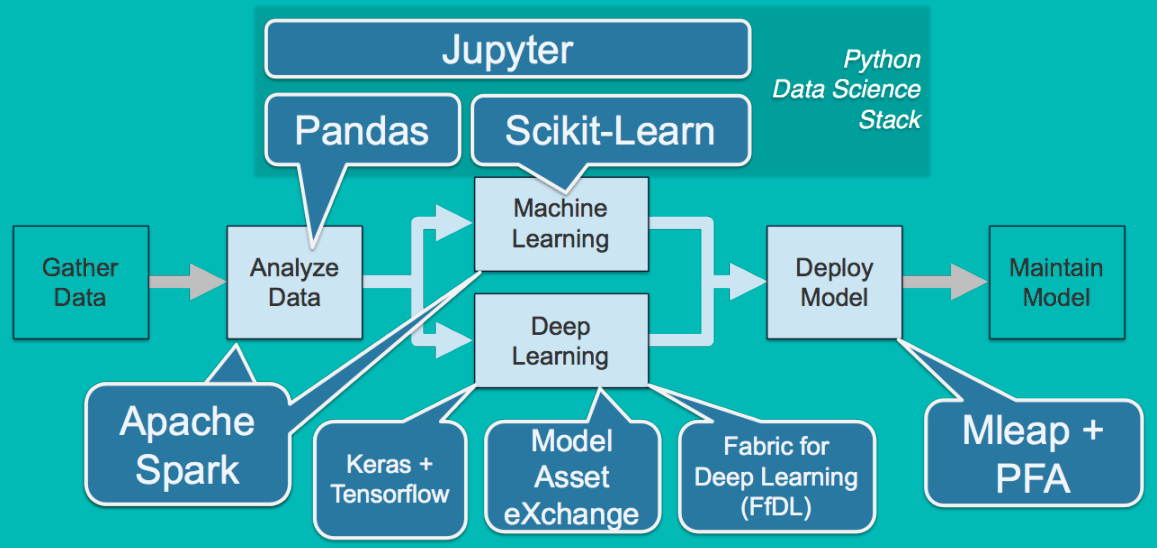
CODAIT

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



Agenda

The Machine Learning Workflow

Machine Learning Deployment

Containers

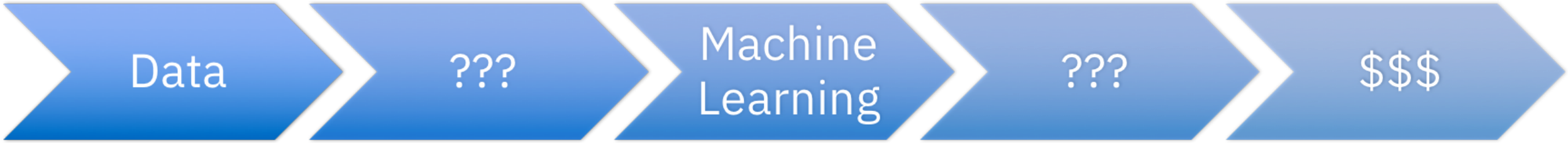
Open Standards for Model Deployment

Monitoring & Feedback

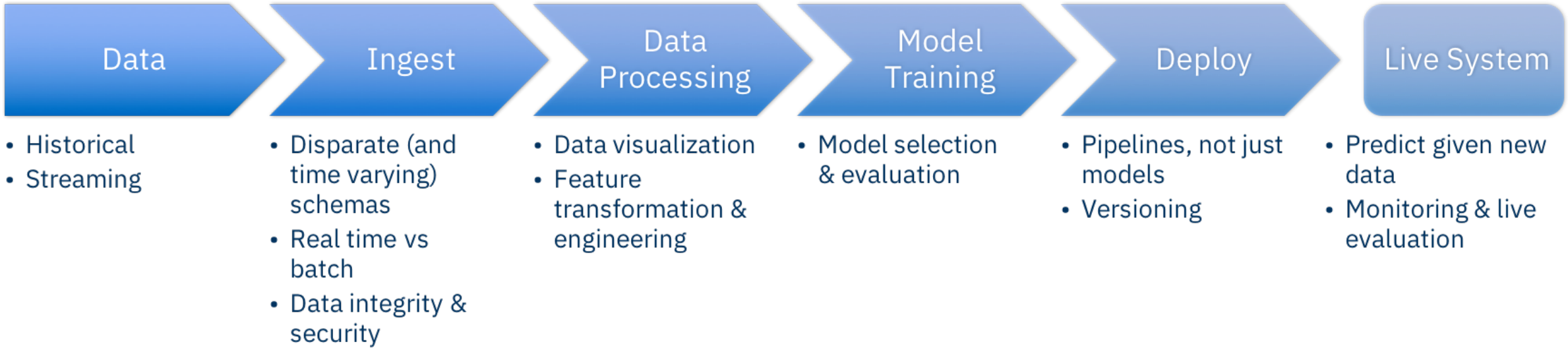
Wrap Up

The Machine Learning Workflow

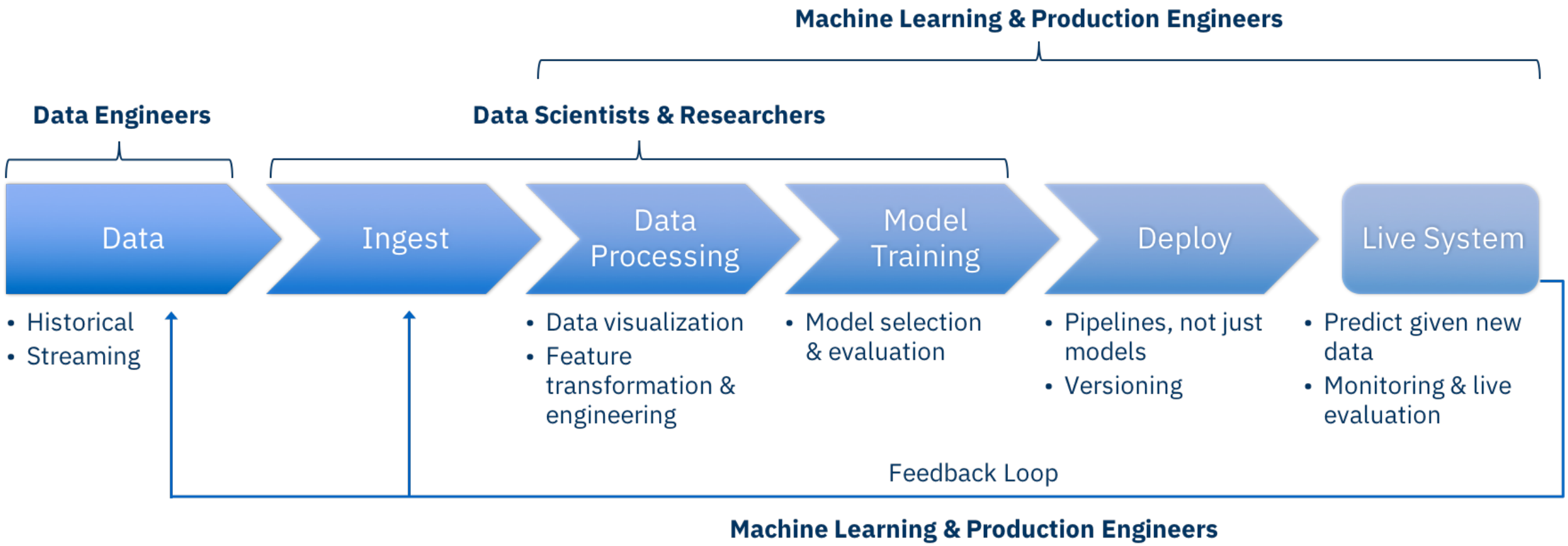
Perception



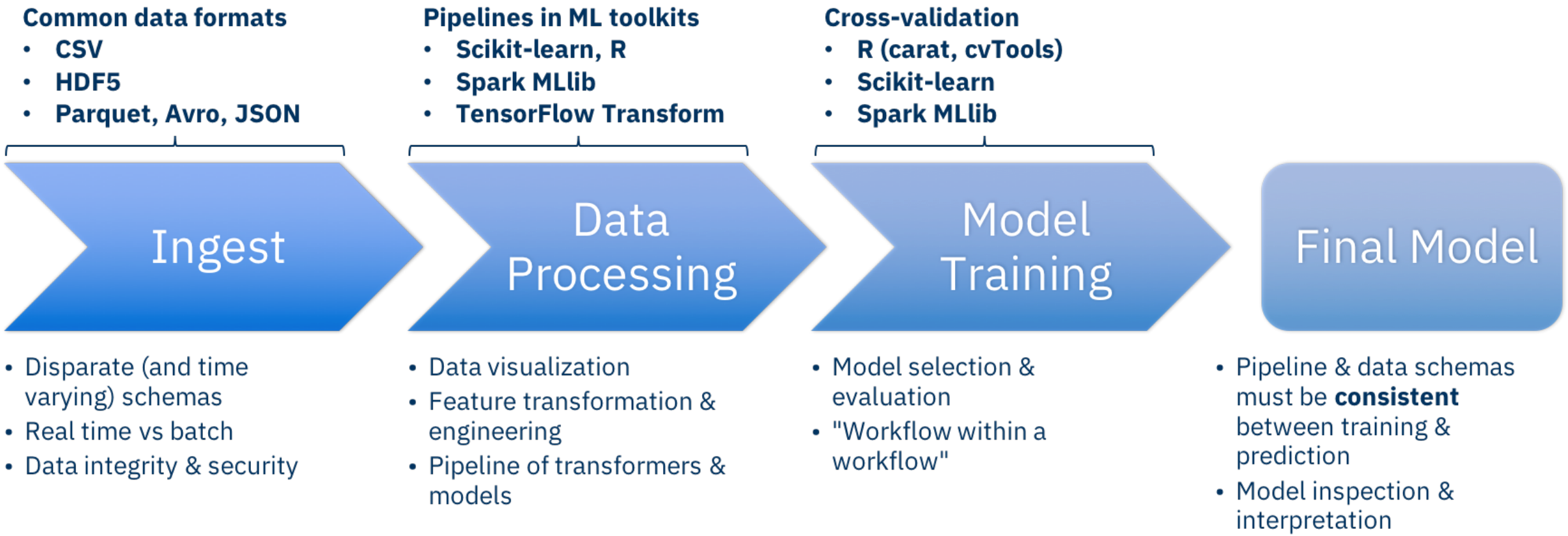
Reality



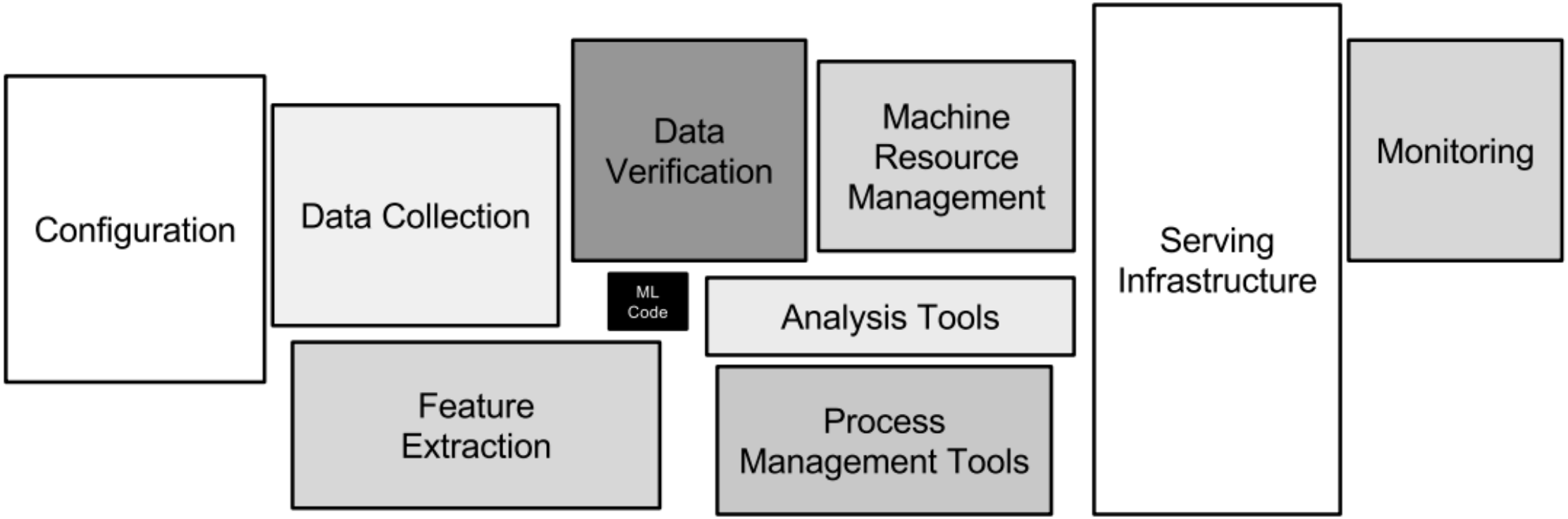
The workflow is a loop that spans teams ...



… and tools …



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



*Source: Hidden Technical Debt in Machine Learning Systems

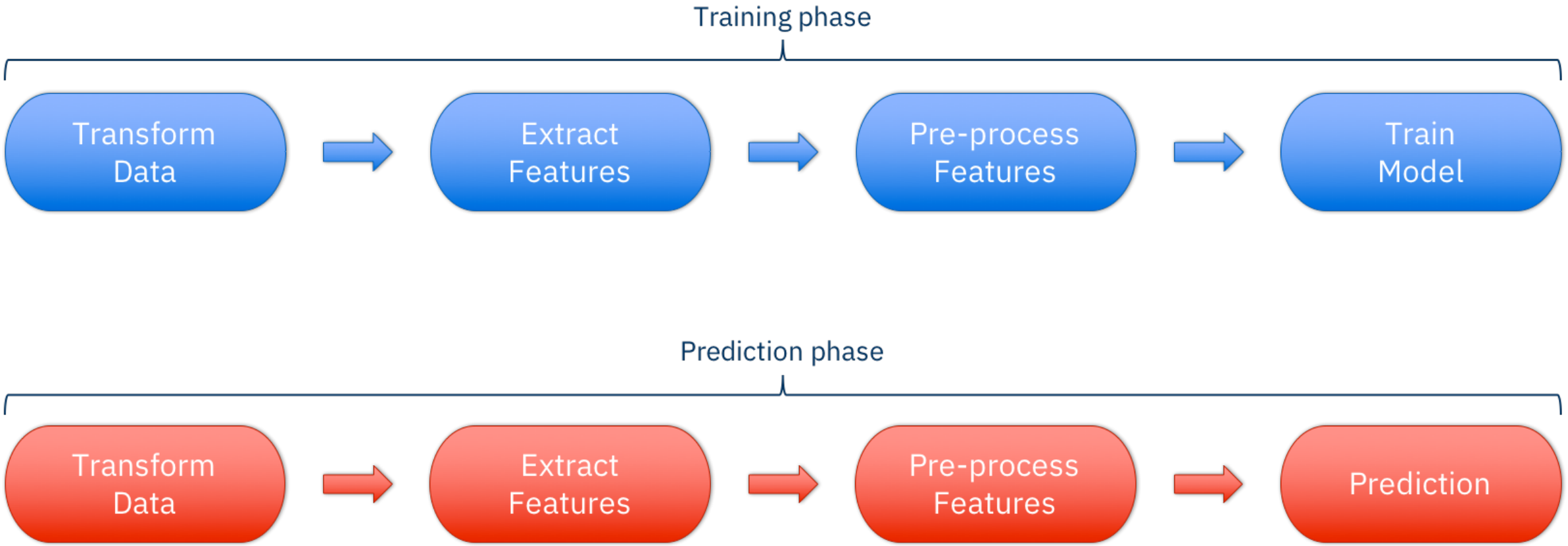
Machine Learning Deployment

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, a little about the how, not much about the where …

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

- Versioning & lineage
- Model size & complexity
  - Recommenders, deep learning: 100s MB – many GBs
  - More complex models: GPUs, batching
- Interpretability
- Multi-model serving
  - Ensemble models
  - Rolling out new versions
  - A/B testing & bandit-based approaches
- Monitoring & Feedback
- Fault tolerance, high availability, fallbacks
- Regulatory compliance

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


Containers

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.

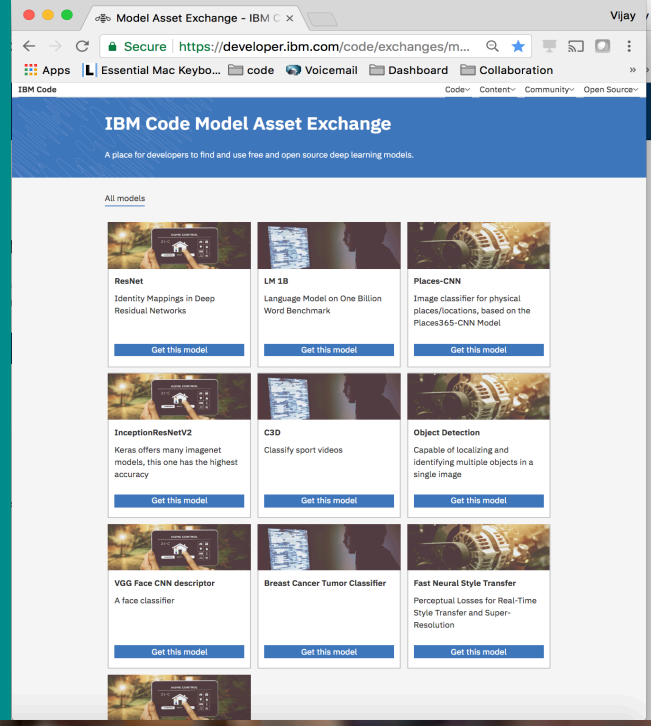
- But …
  - What goes in the container is most important
  - Recall performance can be highly variable across language, framework, version
  - Requires devops knowledge, CI / deployment pipelines, good practices
  - Does not solve the issue of standardization
    - Formats
    - APIs exposed
  - A serving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

IBM Code

Model Asset eXchange

MAX is a one stop exchange to find ML/ DL models created using popular machine learning engines and provides a standardized approach to consume these models for training and inferencing.

MAX



IBM CODE

MAX Serving Framework

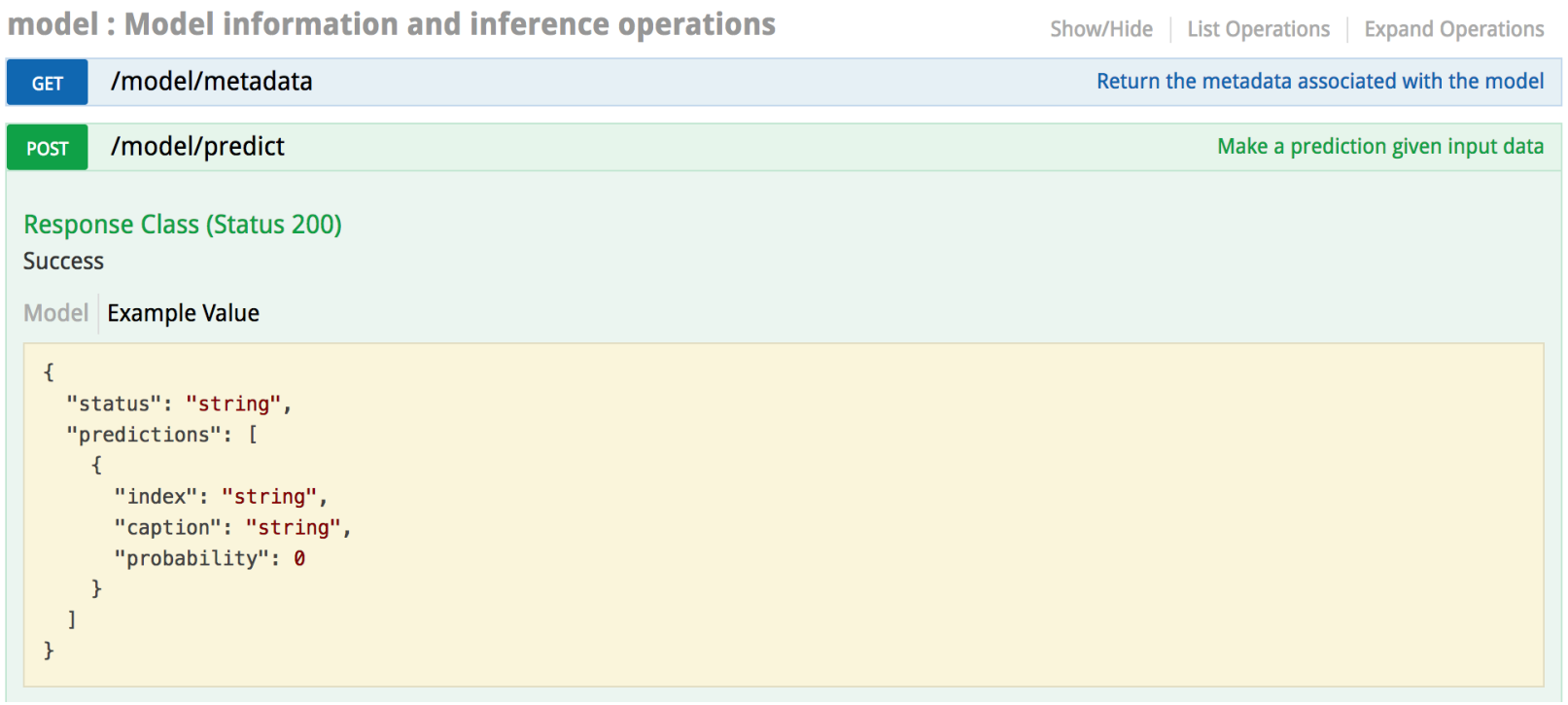
- Docker container

```
$ docker run -it -p 5000:5000 max-im2txt
```

- Lightweight REST API

```
$ curl -F "image=@assets/surfing.jpg" -X POST http://127.0.0.1:5000/model/predict
```

- Swagger API documentation



- Generic model wrapper + pre- / post-processing

Container-based Serving Frameworks

- Seldon Core
  - Runs on Kubernetes
  - Created and supported by ML startup Seldon
  - Seems to be gaining momentum
  - Kubeflow integration
- Clipper
  - UC Berkeley RISElab project
- Both specify some level of common API
- Both support (or aim to) meta-algorithms (ensembles, A/B testing)

Open Standards for Model Deployment

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
- 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

TensorFlow Serving

- Works well for serving TF models
- Supports GPU since it is executing TF Graphs
- Newer SavedModel format exposes standardized APIs
  - Classification
  - Regression
  - Prediction (generic)
- Supports arbitrary Serveables (but C++)

- Shortcomings
  - “Open” format, but not a “standard”
  - In reality difficult to use if not TF Graphs
  - Only supports pipelines to the extent that they are baked into the TF Graph
  - Post-prediction transform not directly supported

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”
- 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

Portable Format for Analytics (PFA)

- 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

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
  - Equivalence tests Spark <-> 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
- 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)

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

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

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 (R, scikit-learn, Spark ML, weka, etc)
- Solves a significant pain point for the deployment of ML pipelines and benefits the wider 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 / comparison to ONNX?
  - Limitations of PFA
  - A standard can move slowly in terms of new features, fixes and enhancements

Future

- 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)
- Further performance testing in progress vs Spark & MLeap; also test vs PMML
- 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
  - Better suited for the more general pre-processing steps of DL pipelines
  - Requires all the various DL-specific operators
  - Requires tensor schema and better tensor support built-in to the PFA spec
  - Should have GPU support

Monitoring & Feedback

Monitoring

- Needs to combine traditional software system monitoring ...
  - Latency, throughput, resource usage, etc
- ... with model performance metrics
  - Traditional ML evaluation measures (accuracy, prediction error)
- ... with business metrics: impact of predictions on business outcomes
  - Additional revenue - e.g. uplift from recommender
  - Cost savings – e.g. value of fraud prevented
  - Metrics implicitly influencing these – e.g. user engagement
- All are important
  - If your model is offline, uplift (or cost savings) are zero!
  - Model performance degrades over time
  - Model performance vs business needs changes over time

Feedback

- A ML system needs to automatically learn from & adapt to feedback from the world around it
- Continual, real-time training
- Reinforcement learning
- Explicit feedback loops – models create or directly influence their own training data
  - Examples include search ranking, recommender systems, bots, AI assistants – influence user behavior through recommendations or suggestions
- Implicit feedback loops – predictions influence behavior in longer-term or indirect ways
- Humans in the loop

Wrapping Up

Deploy pipelines, not just models

Containers are good, but not sufficient

Embrace open standards and formats

Deployment is the beginning, not the end!

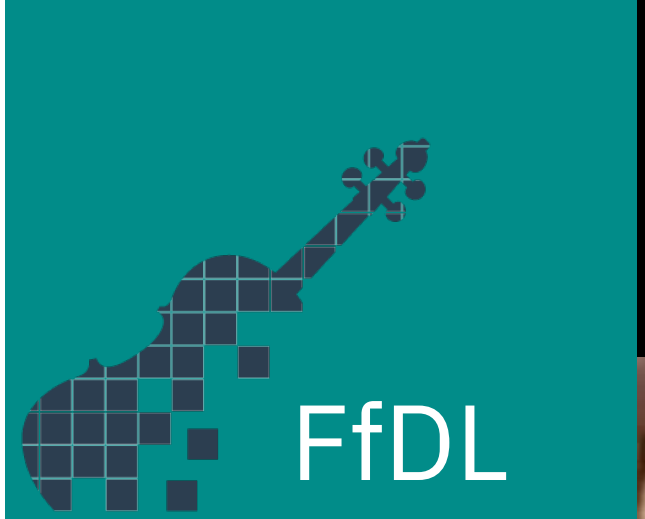
Obrigado!

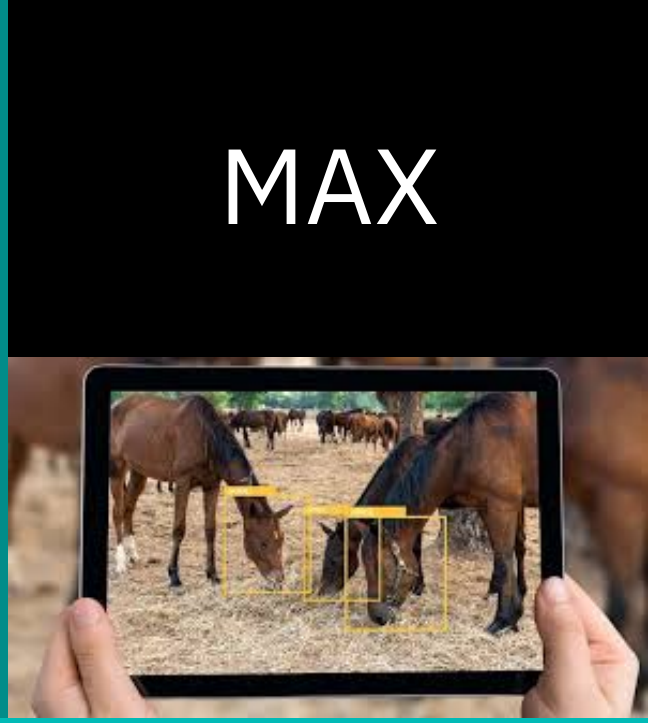
codait.org

twitter.com/MLnick

github.com/MLnick

developer.ibm.com/code





Sign up for IBM Cloud and try Watson Studio!

https://datascience.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

jpmml-sparkml

MLeap

Open Neural Network Exchange

Seldon Core

Clipper