

Open Standards for Machine Learning Deployment

IBM Developer

Nick Pentreath
Principal Engineer

@MLnick

Animesh Singh
STSM and Program Director

@animeshsingh

Hou Gang Liu
Advisory Software Developer

 hougangliu

Center for Open Source Data and AI Technologies

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

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

We contribute to **foundational open source software** across the enterprise AI lifecycle.

40 open-source developers & advocates!



CODAIT

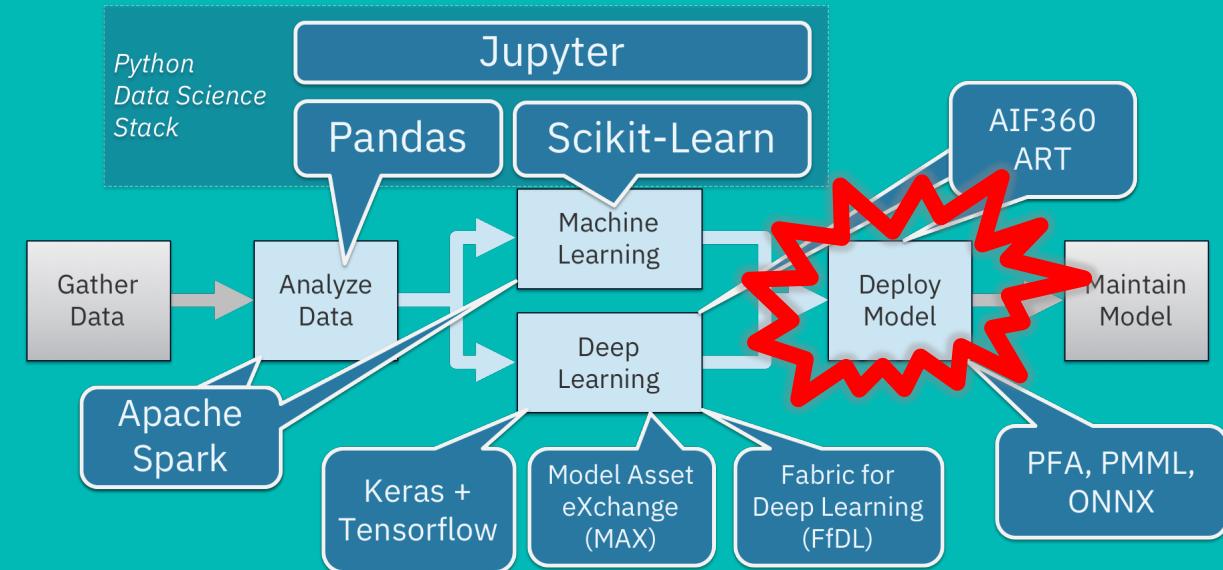
codait.org



HQ in San Francisco, California

Team located across US, some in EMEA

Improving Enterprise AI Lifecycle in Open Source



Agenda

- Background
- Open Standards for Machine Learning Deployment
- Summary

What is Machine Learning?



What is Machine Learning?

Learn from **data** to make predictions



What is Machine Learning?

Learn from **historical** data to make predictions about the **future**



Applied Machine Learning

Learn from historical data to make predictions about the future, in order to make **decisions**



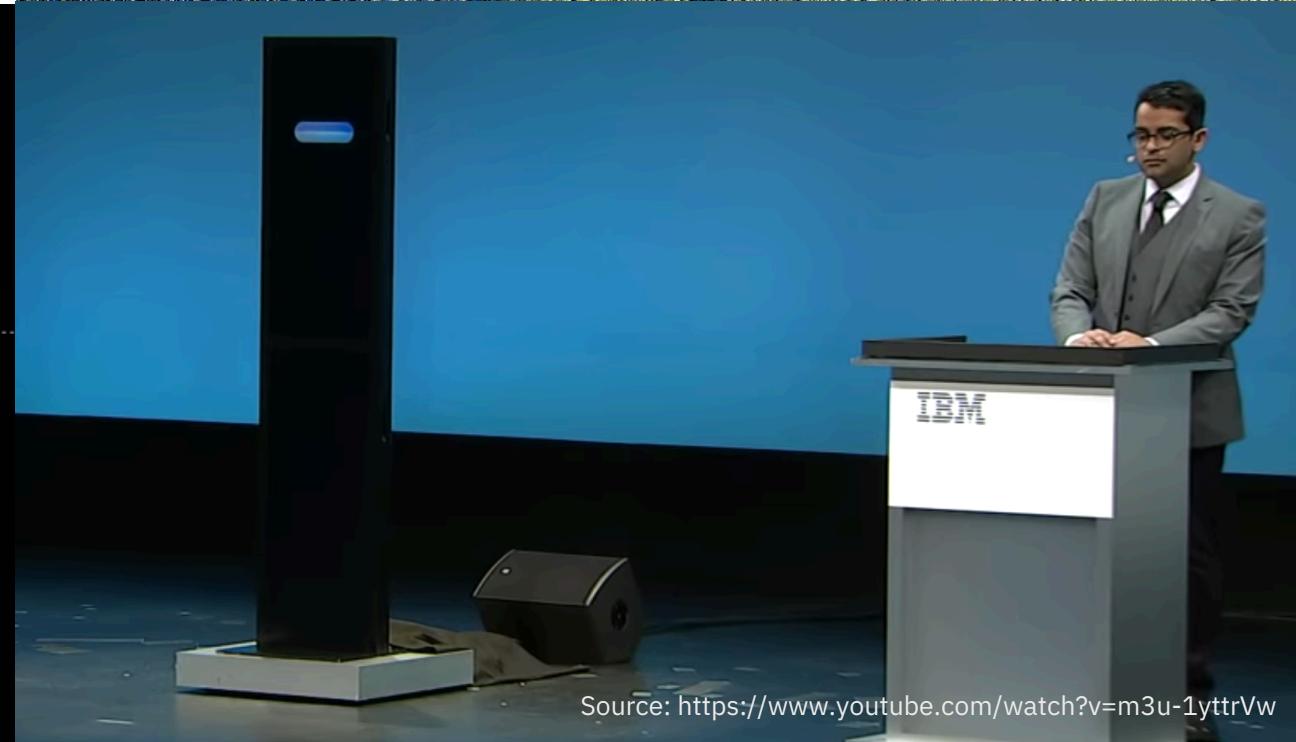
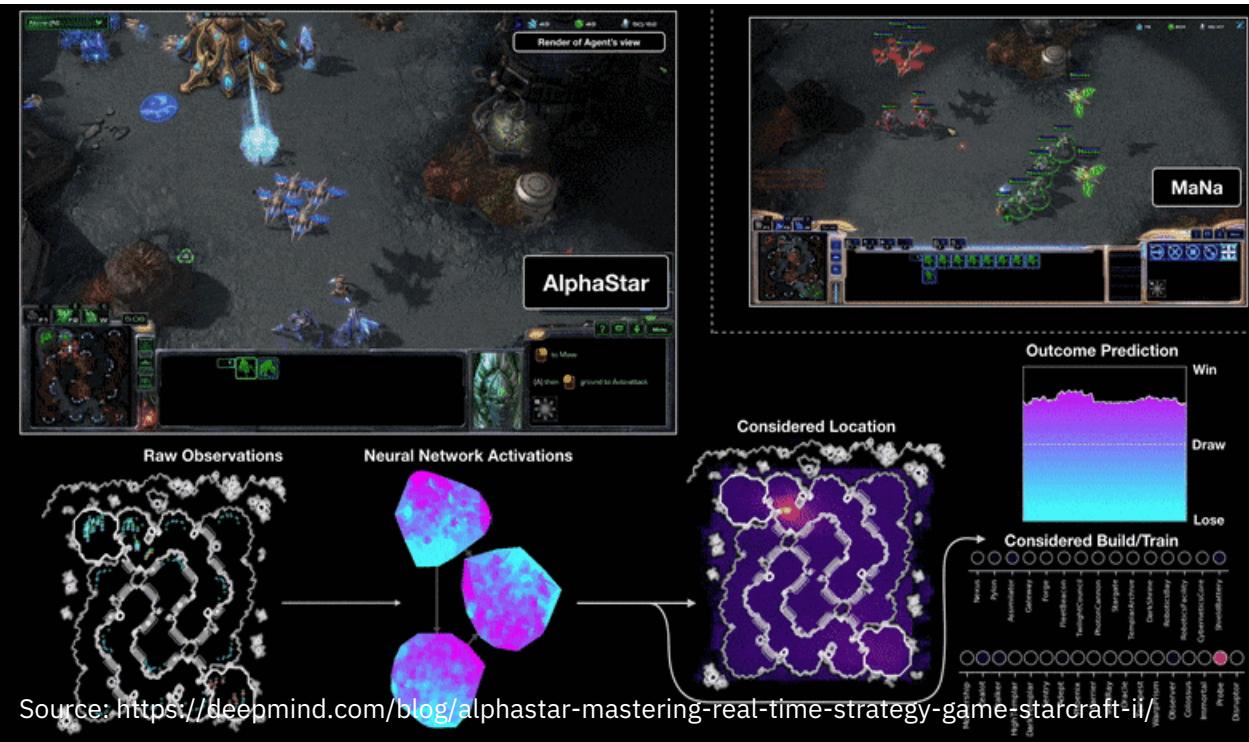
Intelligent Systems

Automated decision-making

Continual learning (new data & feedback)

Adapting to environment

Memory & generalization

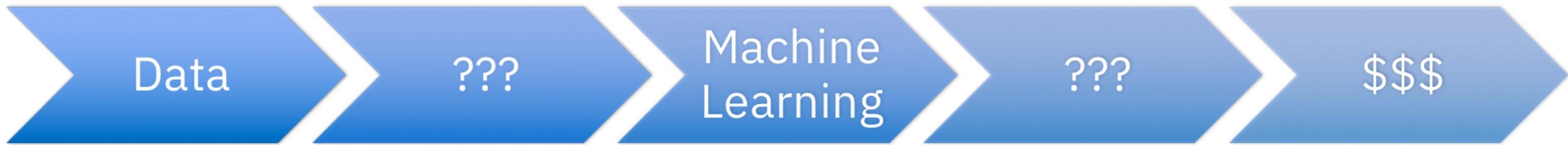


Source: <https://www.youtube.com/watch?v=m3u-1ytrVw>

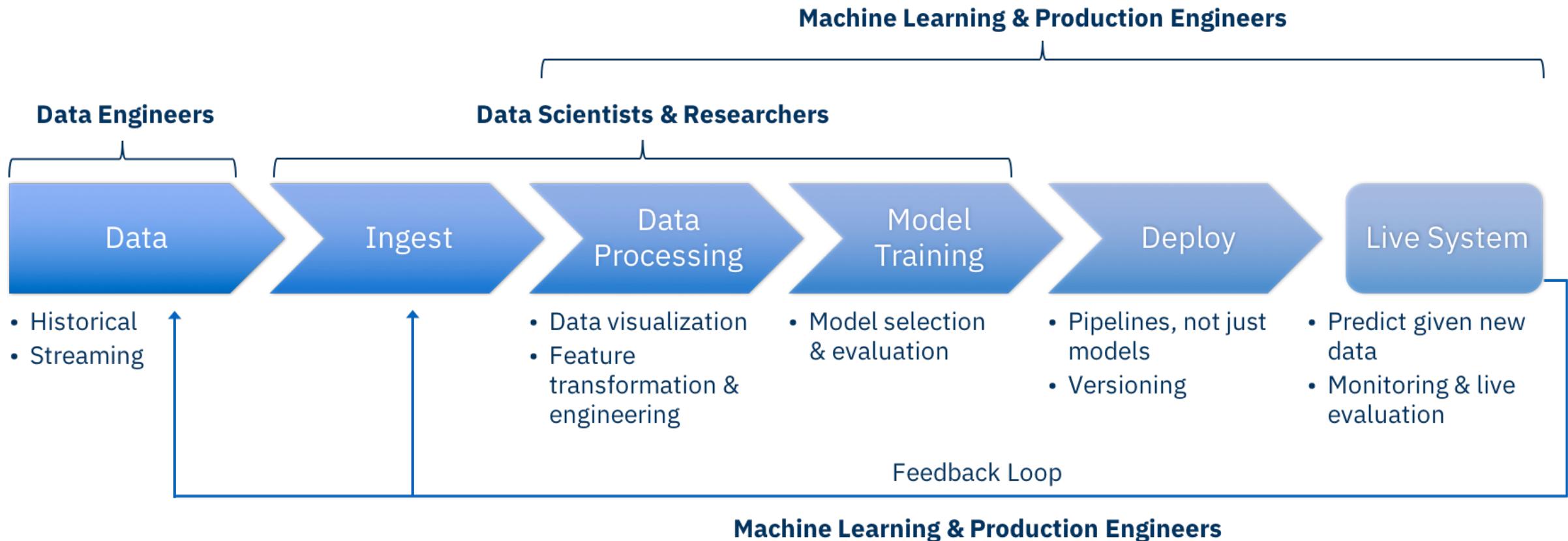
The Machine Learning Workflow



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

- Disparate (and time varying) schemas
- Real time vs batch
- Data integrity & security

Data Processing

- Data visualization
- Feature transformation & engineering
- Pipeline of transformers & models

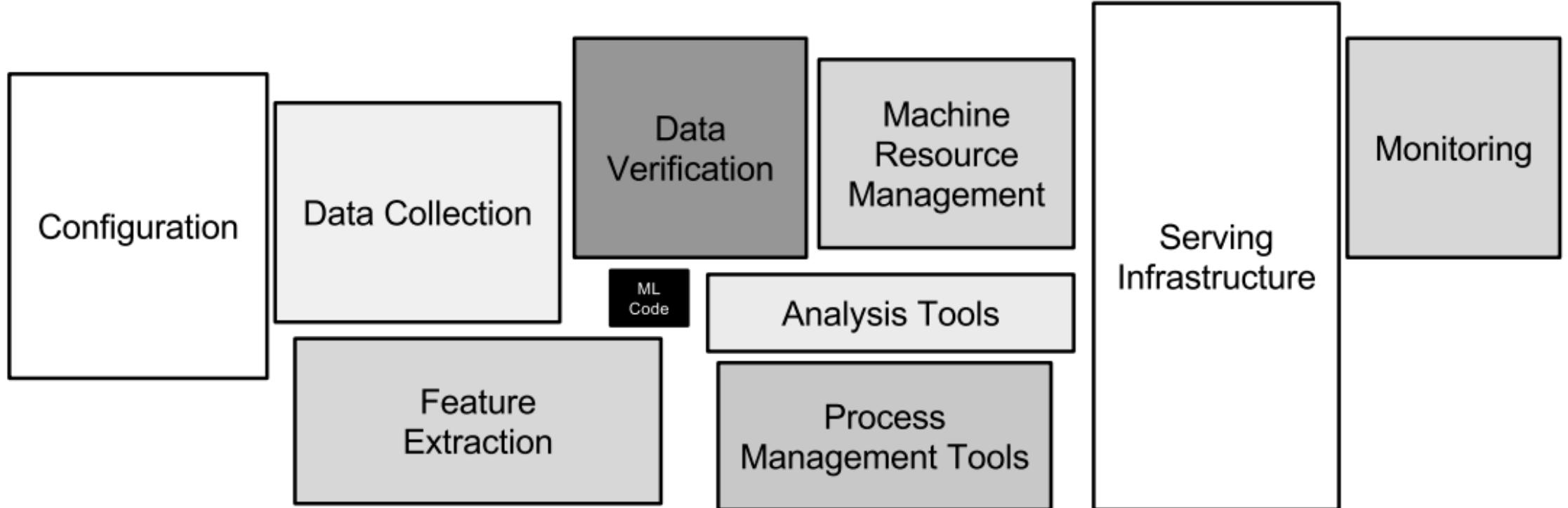
Model Training

- Model selection & evaluation
- "Workflow within a workflow"

Final Model

- Pipeline & data schemas must be **consistent** between training & prediction
- Model inspection & interpretation

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





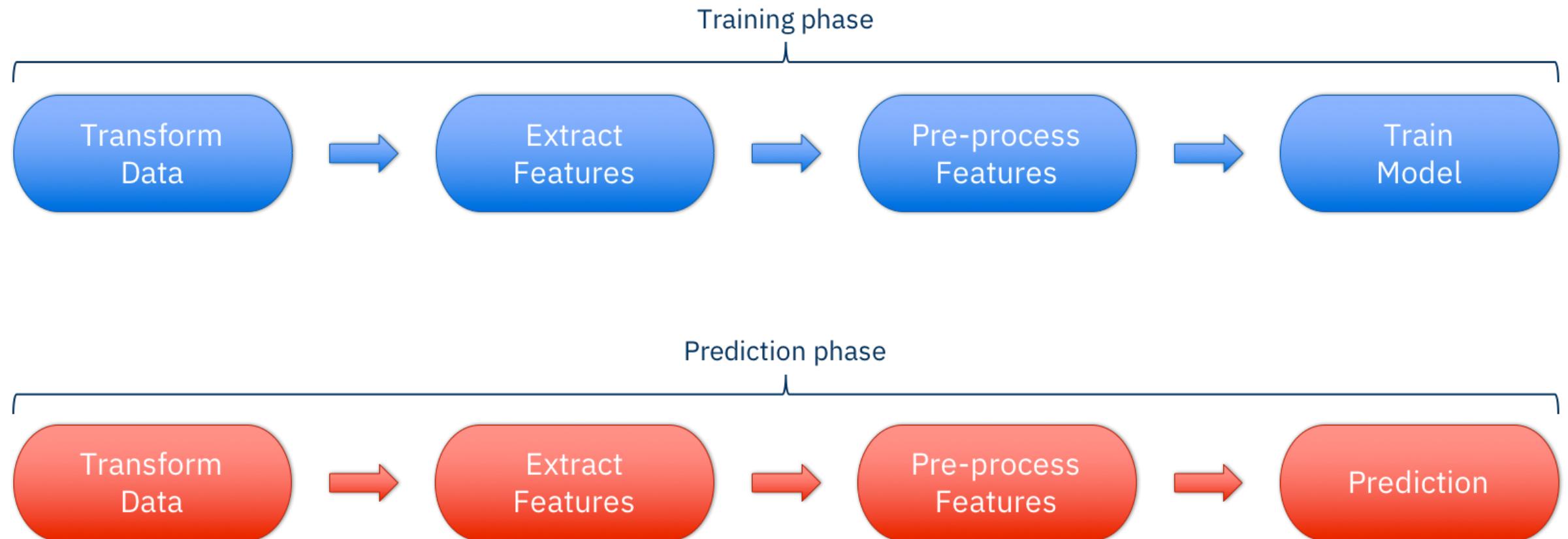
Machine Learning
Deployment

What, Where, How?

- **What** are you deploying?
 - What is a “model”?
- **Where** are you deploying?
 - Target environment (cloud, browser, edge)
 - 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!

- Formats
 - Each framework does things differently
 - Proprietary formats: lock-in, not portable
- Lack of standardization leads to custom solutions and extensions





Containers for ML
Deployment

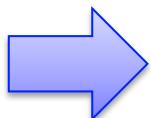
Containers are The Solution... right?

- 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 still the most important factor
 - 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



Open Standards for
Model Serialization

Why a standard?



Standard
Format



Single stack

Execution

Optimization

Tooling
(Viz, analysis, ...)

Predictive Model Markup Language

- PMML was created by Data Mining Group (DMG) - IBM is a founding member
- Model interchange format in XML
- First release in 1997
- Svetlana Levitan from CODAIT is release manager for PMML (and PFA)
- Widely used and supported – [over 30 vendors and organizations](#)
- Spark support lacking natively but 3rd party projects available: [jpmmml-sparkml](#)
- Other exporters include [scikit-learn](#), [R](#), [XGBoost](#) and [LightGBM](#)

[**http://dmg.org/pmml**](http://dmg.org/pmml)

PMML – Format overview

```
▼<PMML xmlns="http://www.dmg.org/PMML-4_1" version="4.1">
  ▼<Header copyright="KNIME">
    <Application name="KNIME" version="2.8.0"/>
  </Header>
  ▼<DataDictionary numberOfFields="10">
    ▼<DataField dataType="integer" name="Age" optype="continuous">
      <Interval closure="closedClosed" leftMargin="17.0" rightMargin="90.0"/>
    </DataField>
    ▶<DataField dataType="string" name="Employment" optype="categorical">...</DataField>
    ▶<DataField dataType="string" name="Education" optype="categorical">...</DataField>
    ▶<DataField dataType="string" name="Marital" optype="categorical">...</DataField>
    ▶<DataField dataType="string" name="Occupation" optype="categorical">...</DataField>
    ▶<DataField dataType="double" name="Income" optype="continuous">...</DataField>
    ▼<DataField dataType="string" name="Gender" optype="categorical">
      <Value value="Female"/>
      <Value value="Male"/>
    </DataField>
    ▶<DataField dataType="double" name="Deductions" optype="continuous">...</DataField>
    ▶<DataField dataType="integer" name="Hours" optype="continuous">...</DataField>
    ▶<DataField dataType="string" name="TARGET_Adjusted" optype="categorical">...</DataField>
  </DataDictionary>
  ▼<GeneralRegressionModel modelType="multinomialLogistic" functionName="classification" algorithmName="LogisticRegression" Logistic Regression">
    ▶<MiningSchema>...</MiningSchema>
    ▶<ParameterList>...</ParameterList>
    ▶<FactorList>...</FactorList>
    ▶<CovariateList>...</CovariateList>
    ▶<PPMatrix>...</PPMatrix>
    ▶<ParamMatrix>...</ParamMatrix>
  </GeneralRegressionModel>
</PMML>
```

PMML - Coverage

- Association Rules Model
- Clustering Model
- General Regression
- Naïve Bayes
- Nearest Neighbor Model
- Neural Network
- Regression
- Tree Model
- Mining Model: ensemble / composition
- Baseline Model
- Bayesian Network
- Gaussian Process
- Ruleset
- Scorecard
- Sequence Model
- Support Vector Machine
- Time Series
- Various feature transformation functions
(norm, discretize, UDFs)

PMML - Shortcomings

- Shortcomings
 - Cannot represent arbitrary programs / analytic applications
 - Flexibility comes from custom plugins => lose benefits of standardization
 - Potential questions around licensing for open-source scoring engine – [jpml-evaluator](#) (dual AGPL 3.0 / commercial license)

Portable Format for Analytics

- PFA was also created by DMG
- PMML has some important limitations around flexibility and extensibility
- PFA is an attempt 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

<http://dmg.org/pfa>

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

```
"cells":{  
    "vocab-mapping":{  
        "init":{  
            ...  
        },  
        "type":{  
            "type":"record",  
            "name":"Vocab",  
            "fields": [  
                {  
                    "name":"vocab",  
                    "type":{  
                        "type":"map",  
                        "values":"int"  
                    }  
                }  
            ]  
        }  
    }  
}
```

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

Function library +

- core
- m
- m.special
- m.link
- m.kernel
- la
- metric
- rand
- s
- re
- parse
- cast
- a
- map
- bytes
- fixed
- enum
- time
- impute
- interp
- prob.dist
- stat.test
- stat.sample
- stat.change
- model.reg
- model.tree
- model.cluster
- model.neighbor
- model.naive
- model.neural
- model.svm

{"s.len": [s]}

s string
(returns) int

Description: Return the length of string s.

{"s.substr": [s, start, end]}

s string
start int
end int
(returns) string

Description: Return the substring of s from start (inclusive) until end (exclusive).

Details:

- Negative indexes count from the right (-1 is just before the last item), indexes beyond the legal range are truncated, and end \leq start specifies a zero-length subsequence just before the start character. All of these rules follow Python's slice behavior.

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 (3D+)
 - 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 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
- Help welcome!
 - 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>

PFA - 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
 - PFA / PMML / ONNX
- 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

Open Neural Network Exchange (ONNX)

- Championed by Facebook & Microsoft
- Protobuf for serialization format and type specification
- Describes computation graph (including operators)
 - In this way the serialized graph is “self-describing” similarly to PFA
- More focused on Deep Learning / tensor operations
- Baked into PyTorch 1.0.0 / Caffe2 as the serialization & interchange format
- ONNX-ML
 - Provides support for (parts of) “traditional” machine learning
 - Additional types – lists and maps
 - Operators
 - Vectorizers (numeric & string data)
 - One hot encoding, label encoding
 - Scalers (normalization, scaling)
 - Models (linear, SVM, TreeEnsemble)
 - ...

ONNX Recent Developments

- Open source scoring engine for ONNX released by Microsoft
 - Supports ONNX-ML spec

<https://github.com/Microsoft/onnxruntime>

- Strong recent activity in converters – e.g.
[scikit-learn](#), [SparkML](#)
- Open governance model

<https://github.com/onnx/onnx/tree/master/community>

ONNX Supported Tools

Frameworks



PyTorch



PaddlePaddle



MATLAB®



Neural Network Libraries

Converters



LibSVM



Runtimes



BITMAIN



Windows



CEVA



Compilers



Visualizers



ONNX - future directions

- ONNX Training Working Group
 - Enable ONNX to support model training
- Performance testing & improvements
 - PFA / PMML / ONNX
- ONNX functions and control flow still developing (experimental)
- Extensions to ONNX-ML ecosystem
 - Text pre-processing, e.g. Tokenization / better NLP support
 - Still limited image pre-processing / transformers
 - (*Crop* added)
- Further development on converters – TensorFlow, PyTorch, ...



Conclusion

Summary

PMML



- Established standard, backed by DMG members
- Widely supported
- Good support for standard machine learning models and feature processing
- Custom extensions



- Custom extensions
- Less flexibility
- State of open source scoring engine
- Poor support for deep learning, GPUs

Summary



- Backing by DMG
- Growing adoption
- Greatest flexibility and extensibility
- Can handle (almost) any model or feature processing workflow



- Complex and difficult to learn
- Still relatively new
- Open questions around performance and scalability
- Poor support for deep learning, GPUs

Summary



- Backing by large industry players
- Growing rapidly with lots of momentum
- Focused on deep learning operators
- ONNX-ML provides some support for "traditional" ML and feature processing



- Still relatively new
- Difficult to keep up with breadth and depth of framework evolution
- Relatively poor support for feature processing and other data types (strings, datetime, etc)

Summary

- **Open standards** 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 in a truly open manner
- However there are risks
 - PFA, ONNX still relatively young and needs to gain adoption
 - Performance in production, at scale, is relatively untested
 - Limitations of the various standards
 - Can one standard encompass all requirements & use cases?
 - If not, how to optimally use and combine each?

Get involved - it's open source, open governance!

Thank you!

<http://ibm.biz/model-exchange>

<http://ibm.biz/max-developers>

 codait.org

 github.com/hougangliu

 developer.ibm.com



Sign up for IBM Cloud and try Watson Studio!

<https://ibm.biz/Bd2NjG>

