Building Machine Learning inference pipelines at scale

Julien Simon
Global Evangelist, AI & Machine Learning
@julsimon

Problem statement

- Real-life Machine Learning applications require more than a single model.
- Data may need pre-processing: normalization, feature engineering, dimensionality reduction, etc.
- Predictions may need post-processing: filtering, sorting, combining, etc.

Our goal:
build scalable ML pipelines with open source (Spark, Scikit-learn, XGBoost) and managed services (Amazon EMR, AWS Glue, Amazon SageMaker)

Building pipelines with Spark

Apache Spark

https://spark.apache.org/

- Open-source, distributed processing system
- In-memory caching and optimized execution for fast performance (typically 100x faster than Hadoop)
- Batch processing, streaming analytics, machine learning, graph databases and ad hoc queries
- API for Java, Scala, Python, R, and SQL
- Available in Amazon EMR and AWS Glue

MLlib – Machine learning library

https://spark.apache.org/docs/latest/ml-guide.html

- Algorithms: classification, regression, clustering, collaborative filtering.
- Featurization: feature extraction, transformation, dimensionality reduction.
- Tools for constructing, evaluating and tuning pipelines
  - *Transformer* – a transform function that maps a *DataFrame* into a new one
    - Adding a column, changing the rows of a specific column, etc.
    - Predicting the label based on the feature vector
  - *Estimator* – an algorithm that trains on data
    - Consists of a *fit*() function that maps a *DataFrame* into a *Model*

Example: binary classification for text samples

https://github.com/apache/spark/blob/master/examples/src/main/scala/org/apache/spark/examples/ml/PipelineExample.scala

```
// Prepare training documents from a list of (id, text, label) tuples.
val training = <LOAD_TRAINING_DATA>

// Configure an ML pipeline with three stages: tokenizer, hashingTF, and lr.
val tokenizer = new Tokenizer().setInputCol("text”).setOutputCol("words")
val hashingTF = new HashingTF()
   .setNumFeatures(1000)
   .setInputCol(tokenizer.getOutputCol)
   .setOutputCol("features")
val lr = new LogisticRegression().setMaxIter(10).setRegParam(0.001)

val pipeline = new Pipeline().setStages(Array(tokenizer, hashingTF, lr))

// Fit the pipeline to training documents.
val model = pipeline.fit(training)

// Prepare test documents, which are unlabeled (id, text) tuples.
val test = <LOAD_TEST_DATA>

// Make predictions on test documents.
model.transform(test)
```

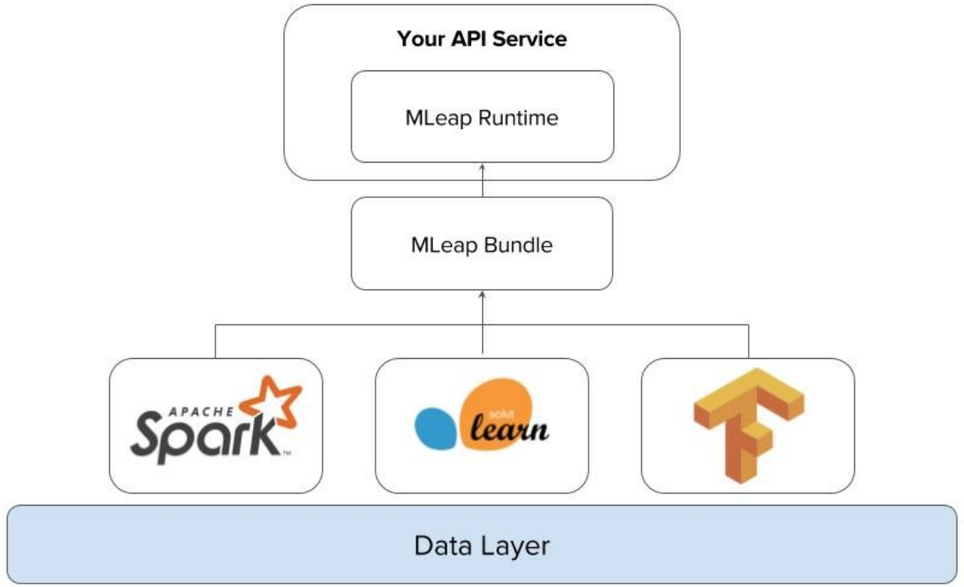
This is a naïve example. What about real-life challenges?

1. Exporting models to other applications
2. Decoupling pipeline infrastructure
3. Using any ML algorithm in any language
4. Getting low latency predictions

… while avoiding complex and time-consuming infrastructure work

#1 – Exporting models

- Save the model and load it in another Spark application
- Export the model to PMML, and use it with Java, R, etc.
- Export the model to MLeap
  - http://mleap-docs.combust.ml/
  - Lightweight runtime independent from Spark
  - Interoperability between SparkML, TensorFlow and scikit-learn



#2 – Decoupling pipeline infrastructure

- Different steps require different hardware configurations
  - Say, R5 for ETL, P3 for training and C5 for prediction?
  - If you need GPUs for training, does it make sense to run ETL on GPUs?
  - Do you want to build and manage a specific cluster for each step?

- Size and scale each step independently
  - Avoid oversizing your Spark cluster because one step requires it
  - Avoid time-consuming resizing operations on Amazon EMR
  - We often run ETL once, and train many models in parallel

#3 – Using any ML algorithm

- MLlib is great, but you may need something else
- Other ML algorithms
- Deep Learning: TensorFlow, Apache MXNet, PyTorch, etc.
- Your own custom code in any language

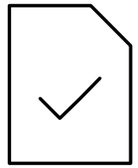
#4 – Getting low latency predictions

- Run ML predictions without using Spark.
  - Save the overhead of the Spark framework
  - Save loading your data in a *DataFrame*
  - Deploy MLeap models

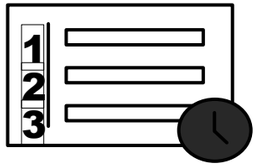
- Improve latency for small-batch predictions.
  - It can be difficult to achieve low-latency predictions with Spark
  - Use the optimal instance type for prediction

Combining Spark and Amazon SageMaker

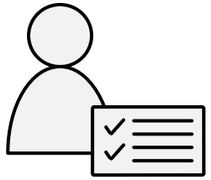
Amazon SageMaker:
Build, Train, and Deploy ML Models at Scale



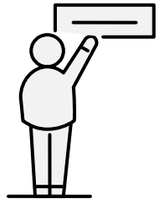
Collect and prepare training data



Choose and optimize your ML algorithm

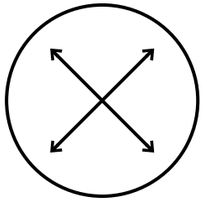


Set up and manage environments for training



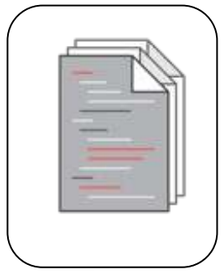
Train and Tune ML Models

Deploy models in production



Scale and manage the production environment

Model options



Training code

Factorization Machines
Linear Learner
Principal Component Analysis
K-Means Clustering
XGBoost
And more

Built-in Algorithms (17)

No ML coding required
Distributed training
Pipe mode

mxnet Chainer TensorFlow PyTorch scikit learn

Built-in Frameworks

Bring your own code: script mode
Open source containers
Distributed training
Pipe mode

Bring Your Own Container

Full control, run anything!
R, C++, etc.

No infrastructure work required

Amazon SageMaker SDKs

- SageMaker API
  - AWS SDKs (boto3, etc.), CLI
  - Nice for low-level management and automation
- Python SDK (aka 'SageMaker SDK')
  - Algorithm selection, training, deploying, automatic model tuning, etc.
  - Nice for notebooks and experimentation
- PySpark/Scala SDK for Apache Spark 2.1.1 and 2.2
  - Pre-installed on Amazon EMR 5.11 and later
  - Train, deploy and predict with SageMaker directly from your Spark application
  - Supports standalone models and MLlib pipelines
  - *DataFrames* in, *DataFrames* out: no data conversion needed

Demo #1

Train a Spark MLlib model on Amazon EMR
Export it to MLeap format
Deploy it on Amazon SageMaker

https://github.com/awslabs/amazon-sagemaker-examples/tree/master/sagemaker-python-sdk/sparkml_serving_emr_mleap_abalone

Inference Pipelines with Amazon SageMaker

Inference Pipelines

- Sequence of 2-5 containers processing inference requests

- Train a model for each step, deploy pipeline as a single unit
  - Real-time prediction endpoint (HTTPS)
  - Batch transform

- Use any model
  - Built-in algorithms,
  - Built-in frameworks (including MLeap)
  - Custom containers

Demo #2

Train a preprocessing model (Scikit-learn) model on Amazon SageMaker
Train a prediction model (XGBoost) on Amazon SageMaker
Deploy an Inference Pipeline with both models on Amazon SageMaker

https://github.com/awslabs/amazon-sagemaker-examples/tree/master/sagemaker-python-sdk/scikit_learn_inference_pipeline

Getting started

https://ml.aws

https://aws.amazon.com/sagemaker

https://github.com/awslabs/amazon-sagemaker-examples
https://github.com/aws/sagemaker-spark
https://github.com/aws/sagemaker-sparkml-serving-container

https://medium.com/@julsimon

Thank you!

Julien Simon
Global Evangelist, AI & Machine Learning
@julsimon