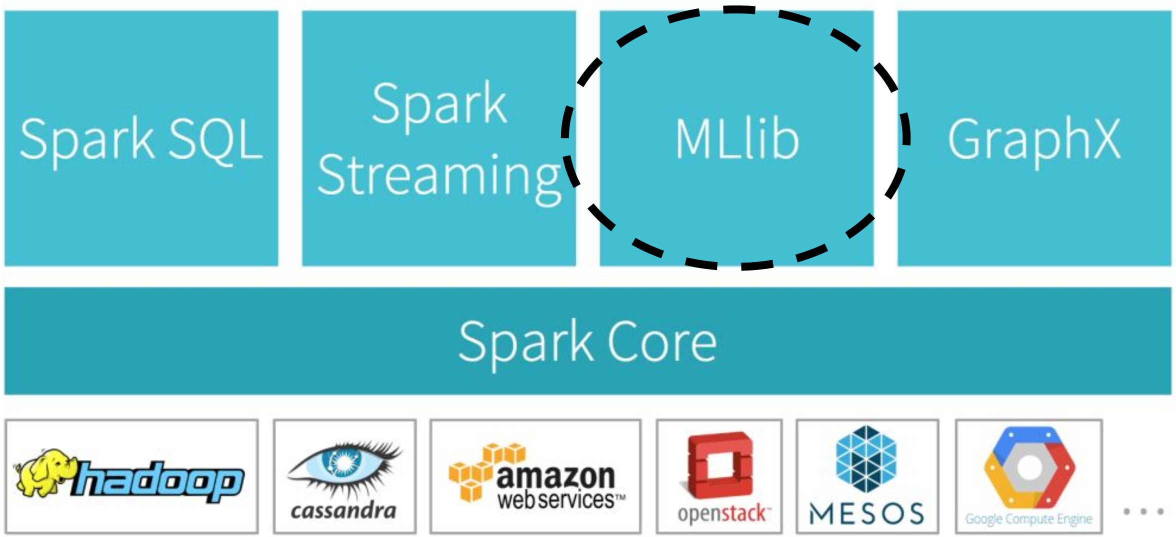
MLlib

Jorge Acosta Hernández
jorge.acosta@upm.es

With some slides from Jesús Montes

Dec. 2024

Machine Learning in the Spark Stack



Machine Learning Library (MLlib)

MLlib is Spark’s machine learning (ML) library. **Its goal is to make practical machine learning scalable and easy**. It provides tools such as:

- **Featurization**: feature extraction, transformation and selection.
- **ML Algorithms**: common learning algorithms such as classification, regression, clustering and more complex as recommendation systems.
- **Pipelines**: tools for constructing, evaluating, and tuning ML Pipelines.
- **Persistence**: saving and load algorithms, models, and Pipelines.
- **Utilities**: linear algebra, statistics, data handling, etc.

MLlib API

- All tools in MLlib can be accessed through the MLlib API
  - Available in Scala, Java and Python.
  - Limited functionality in R, but growing.
- **The MLlib API is DataFrame-based**
  - All algorithms, transformations and other operations take DataFrames as input and (usually) produce DataFrames as output.
- From earlier versions of Spark, it still exists an RDD-based MLlib API
  - This API is currently in “maintenance mode”, which means that bugs are still being fixed, but no new functionalities are added.
  - The main focus is now the DataFrame-based API.
  - **The DataFrame-based API is recommended.**

MLlib API components

The MLlib API is based on five main components:

- **DataFrame**: Input Data Structure.
- **Transformer**: A Transformer is an algorithm which can transform one DataFrame into another DataFrame.
- **Estimator**: An Estimator is an algorithm which can be fit on a DataFrame to produce a Transformer.
- **Pipeline**: A Pipeline chains multiple Transformers and Estimators together to specify an ML workflow.
- **Parameter**: All Transformers and Estimators share a common API for specifying parameters.

Transformer

- A Transformer is an abstraction that includes feature transformers and learned models.
- A Transformer implements a method `transform()`, which converts one DataFrame into another, generally by appending one or more columns.
- For example:
  - A feature transformer might take a DataFrame, read a column (e.g., text), map it into a new column (e.g., feature vectors), and output a new DataFrame with the mapped column appended.
  - A learning model might take a DataFrame, read the column containing feature vectors, predict the label for each feature vector, and output a new DataFrame with predicted labels appended as a column.

Examples of Transformers

```
df = spark.createDataFrame(data, columns)

# Create a VectorAssembler
assembler = VectorAssembler(inputCols=["x1", "x2"],
outputCol="features")

# Transform the DataFrame
output = assembler.transform(df)

# Show the resulting DataFrame
output.show(truncate=False)
####################################################
# Create a Normalizer
normalizer = Normalizer(inputCol="features",
outputCol="normFeatures", p=1.0)

# Transform the DataFrame using the Normalizer
l1NormData = normalizer.transform(output)

# Show the resulting DataFrame
l1NormData.show(truncate=False)
```

The first example (top) creates a vector of features from each row of the DataFrame. This vector can be later used for training a regression model.

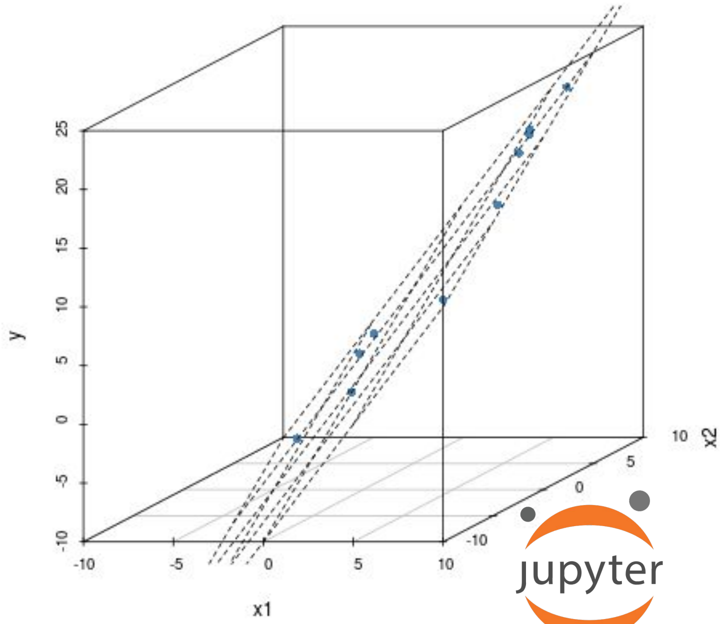
The second example (bottom) normalizes the vector of features.

Estimator

- An Estimator abstracts the concept of a learning algorithm or any algorithm that fits or trains on data.
- An Estimator implements a method `fit()`, which accepts a DataFrame and produces a Model, which is a Transformer.
- For example:
  - A learning algorithm such as `LogisticRegression` is an Estimator, and calling `fit()` trains a `LogisticRegressionModel`, which is a `Model` and hence a Transformer.

Estimator example 1: LinearRegression

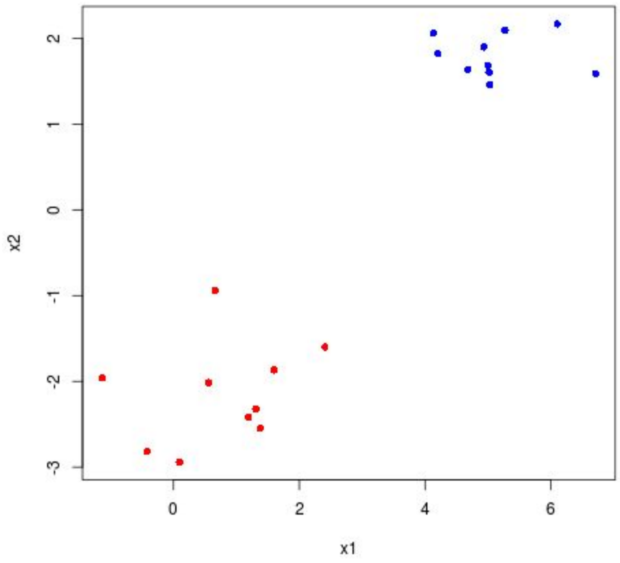
```
from pyspark.ml.regression import LinearRegression

columns = ["x1", "x2", "y"]

df = spark.createDataFrame(data, columns)
# Create a VectorAssembler
assembler = VectorAssembler(inputCols=["x1", "x2"],
outputCol="features")

# Transform the DataFrame
df = assembler.transform(df)

# Create a LinearRegression model
lr = LinearRegression(featuresCol="features", labelCol="y",
maxIter=10, elasticNetParam=0.8)

# Fit the model to the data
lrModel = lr.fit(df)
```



Estimator example 2: K-Means

```python
# Sample data
data = [(1.60193653, -1.8679101),
        (-1.1328963, -1.9607465),
        (2.40675869, -1.5994823),
        (0.09330145, -2.9446696),
        (1.3795901, -2.5489864),
        (-0.42065496, -2.8165693),
        (0.55753398, -2.0145494),
        (1.3066549, -2.3208153),
        (0.66224722, -0.9406476),
        (1.19072851, -2.4178092),
        (4.67961769, 1.6375689),
        (5.03015133, 1.4575724),
        (6.1003413, 2.1673923),
        (4.20259176, 1.8237144),
        (4.93339445, 1.8983999),
        (6.70975052, 1.5899655),
        (5.01461979, 1.6051478),
        (5.00005277, 1.6855351),
        (4.12926186, 2.0582398)]

columns = ["x1", "x2"]
```

```python
df2 = spark.createDataFrame(data, columns)

# Create a VectorAssembler
assembler =
VectorAssembler(inputCols=["x1", "x2"],
outputCol="features")
dataset = assembler.transform(df2)

# Create K-Means model
kmeans = KMeans(featuresCol="features",
k=2)
model = kmeans.fit(dataset)

# Print cluster centers
print("Cluster Centers: ")
for center in model.clusterCenters():
    print(center)
```

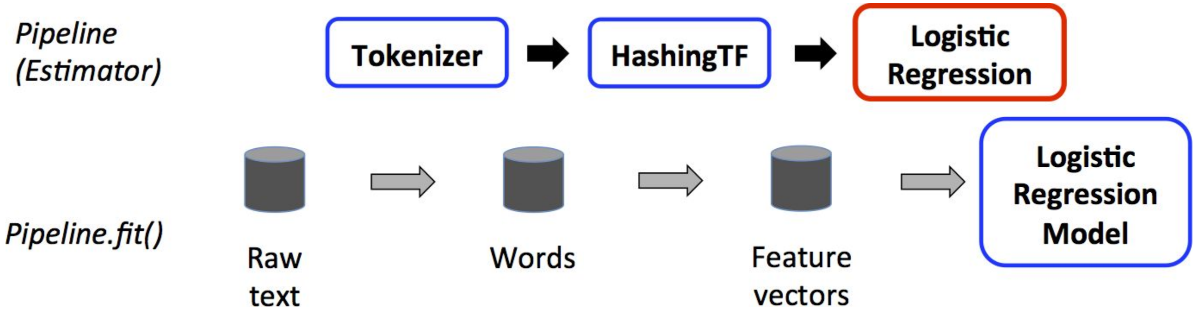


Pipeline

- In machine learning, it is common to run a sequence of algorithms to process and learn from data.
- Example: A simple text document processing workflow might include several stages:
    - Split each document's text into words.
    - Convert each document's words into a numerical feature vector.
    - Learn a prediction model using the feature vectors and labels.
- MLlib represents such a workflow as a Pipeline, which consists of a sequence of PipelineStages (Transformers and Estimators) to be run in a specific order.

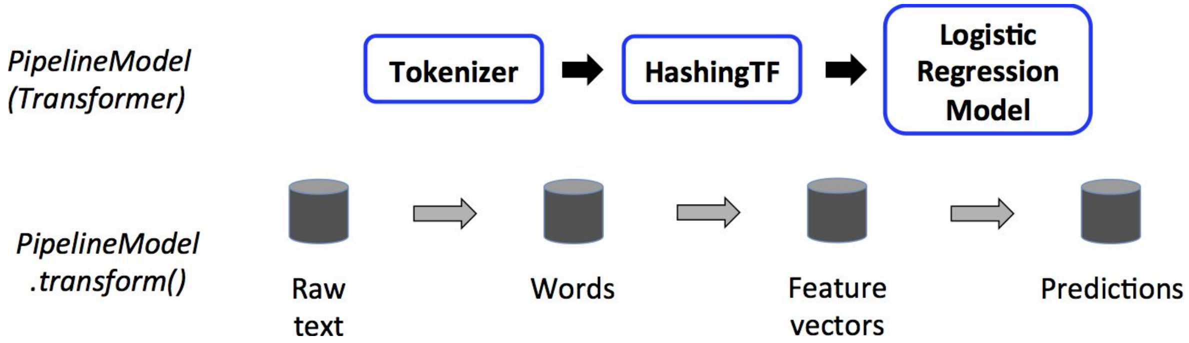
Pipeline

- A Pipeline is specified as a sequence of stages (Transformer or Estimator).
- Stages are run in order.
- The input DataFrame is transformed as it passes through each stage.
  - For Transformer stages, the `transform()` method is called on the DataFrame.
  - For Estimator stages, the `fit()` method is called to produce a Transformer, and that Transformer's `transform()` method is called on the DataFrame.



Pipeline

- A Pipeline is an Estimator.
- After a Pipeline's `fit()` method runs, it produces a PipelineModel, which is a Transformer.
- This PipelineModel is used at test time.



Pipeline Example

70% for training, 30% for test

```
df2 = spark.createDataFrame(data, columns)
# Split the data into training and testing sets
split = df2.randomSplit([0.7, 0.3])
training = split[0]
test = split[1]
```

Split the data in training and test

```
# Create a VectorAssembler
assembler = VectorAssembler(inputCols=["x1", "x2"],
outputCol="features")

# Create a K-Means model
kmeans = KMeans(featuresCol="features", k=2)
```

Create the pipeline stages

```
# Create a Pipeline
pipeline = Pipeline(stages=[assembler, kmeans])
```

Create the pipeline

```
# Fit the Pipeline on the training data
model = pipeline.fit(training)

# Transform the test data and show the results
transformed_data = model.transform(test)
transformed_data.show(truncate=False)
```

Train and use the pipeline

MLlib documentation

- The list of Transformers and Estimators available in MLlib grows with each Spark release.
- Keep the Spark Programming guides always at hand. They are extremely valuable when learning new models and techniques.
- http://spark.apache.org/docs/latest/ml-guide.html