

Big Data

Spark ML

Instructor: Trong-Hop Do

April 24th 2021

S³Lab

Smart Software System Laboratory



“Big data is at the foundation of all the megatrends that are happening today, from social to mobile to cloud to gaming.”

– Chris Lynch, Vertica Systems

Spark Machine Learning



Spark Mlib Utilities

Linear Algebra

- API Guide

<http://spark.apache.org/docs/3.0.1/api/python/pyspark.ml.html#module-pyspark.ml.linalg>

- MLib RDD-Base Guide

<http://spark.apache.org/docs/latest/mllib-data-types.html>

Spark Mlib Utilities

Linear Algebra

▲ Not secure | spark.apache.org/docs/3.0.1/api/python/pyspark.ml.html#module-pyspark.ml.linalg

pyspark.ml.linalg module

MLlib utilities for linear algebra. For dense vectors, MLlib uses the NumPy *array* type, so you can simply pass NumPy arrays around. For sparse vectors, users can construct a **SparseVector** object from MLlib or pass SciPy *scipy.sparse* column vectors if SciPy is available in their environment.

`class pyspark.ml.linalg.Vector`

[\[source\]](#)

`toArray()`

[\[source\]](#)

Convert the vector into an numpy.ndarray

Returns: numpy.ndarray

`class pyspark.ml.linalg.DenseVector(ar)`

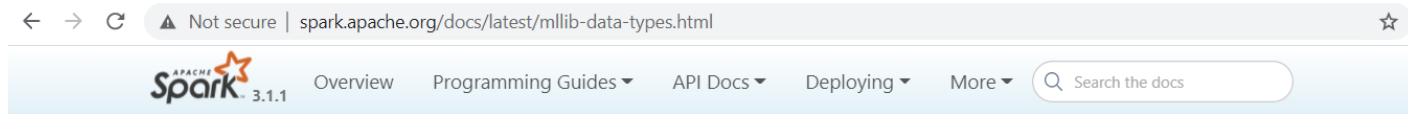
[\[source\]](#)

A dense vector represented by a value array. We use numpy array for storage and arithmetics will be delegated to the underlying numpy array.

```
>>> v = Vectors.dense([1.0, 2.0])
>>> u = Vectors.dense([3.0, 4.0])
>>> v + u
DenseVector([4.0, 6.0])
>>> 2 - v
DenseVector([1.0, 0.0])
>>> v / 2
DenseVector([0.5, 1.0])
>>> v * u
DenseVector([3.0, 8.0])
>>> u / v
DenseVector([3.0, 2.0])
>>> u % 2
```

Spark Mlib Utilities

Linear Algebra



Guide

- Basic statistics
- Data sources
- Pipelines
- Extracting, transforming and selecting features
- Classification and Regression
- Clustering
- Collaborative filtering
- Frequent Pattern Mining
- Model selection and tuning
- Advanced topics

MLlib: RDD-based API Guide

- **Data types**
- Basic statistics
- Classification and regression
- Collaborative filtering
- Clustering
- Dimensionality reduction

Data Types - RDD-based API

- Local vector
- Labeled point
- Local matrix
- Distributed matrix
 - RowMatrix
 - IndexedRowMatrix
 - CoordinateMatrix
 - BlockMatrix

MLlib supports local vectors and matrices stored on a single machine, as well as distributed matrices backed by one or more RDDs. Local vectors and local matrices are simple data models that serve as public interfaces. The underlying linear algebra operations are provided by [Breeze](#). A training example used in supervised learning is called a "labeled point" in MLlib.

Local vector

A local vector has integer-typed and 0-based indices and double-typed values, stored on a single machine. MLlib supports two types of local vectors: dense and sparse. A dense vector is backed by a double array representing its entry values, while a sparse vector is backed by two parallel arrays: indices and values. For example, a vector (1.0, 0.0, 3.0) can be represented in dense format as [1.0, 0.0, 3.0] or in sparse format as (3, [0, 2], [1.0, 3.0]), where 3 is the size of the vector.

Scala

Java

Python

Spark Mlib Utilities

Linear Algebra

Vectors

- Vectors (main class to use)
- DenseVector: MLib uses the NumPy array type
- SparseVector : You can pass SciPy scipy.sparse column vectors
- VectorUDT
- Vector

Matrices

- Matrices (_main class to use_)
- DenseMatrix: Column-major dense matrix
- SparseMatrix : Sparse matrix stored in CSC format
- MatrixUDT
- Matrix

Vectors and matrices are important data types and provide integration with NumPy and SciPy

Spark Mlib Utilities

Linear Algebra

```
[ ]: import findspark
      findspark.init()
      import pyspark
      from pyspark.sql import SparkSession
      spark = SparkSession.builder.getOrCreate()
```


Spark Mlib Utilities

Linear Algebra

Sparse Vector Example

```
[ ]: import numpy as np
import scipy.sparse as sps
from pyspark.ml.linalg import Vectors

# Use a NumPy array as a dense vector.
dv1 = np.array([1.0, 0.0, 3.0])

# Create a SparseVector.
sv1 = Vectors.sparse(3, [0, 2], [1.0, 3.0])

print(dv1)
print(sv1)
```

Dense Vector Example

```
[ ]: # Use a Python list as a dense vector.
dv2 = [1.0, 0.0, 3.0]

# Use a single-column SciPy csc_matrix as a sparse vector.
sv2 = sps.csc_matrix(
    (np.array([1.0, 3.0]), np.array([0, 2]), np.array([0, 2])), shape=(3, 1)
)

print(dv2)
print(sv2)
```

Spark Mlib Utilities

Linear Algebra

Dense Matrix

```
[ ]: from pyspark.ml.linalg import Matrix, Matrices

# Create a dense matrix ((1.0, 2.0), (3.0, 4.0), (5.0, 6.0))
dm2 = Matrices.dense(3, 2, [1, 3, 5, 2, 4, 6])
print(dm2)
```

Sparse Matrix

```
[ ]: # Create a sparse matrix ((9.0, 0.0), (0.0, 8.0), (0.0, 6.0))
sm = Matrices.sparse(3, 2, [0, 1, 3], [0, 2, 1], [9, 6, 8])
print(sm)
```

Spark Mlib Utilities

Data Sources

<http://spark.apache.org/docs/3.0.1/api/python/pyspark.ml.html#module-pyspark.ml.image>

<https://spark.apache.org/docs/latest/ml-datasource#image-data-source>

Spark Mlib Utilities

Data Sources

→ ↺ ⚠ Not secure | spark.apache.org/docs/3.0.1/api/python/pyspark.ml.html#module-pyspark.ml.image

pyspark.ml.image module

`pyspark.ml.image.ImageSchema`

An attribute of this module that contains the instance of `_ImageSchema`.

`class pyspark.ml.image._ImageSchema`

[\[source\]](#)

Internal class for `pyspark.ml.image.ImageSchema` attribute. Meant to be private and not to be instantiated. Use `pyspark.ml.image.ImageSchema` attribute to access the APIs of this class.

`property columnSchema`

Returns the schema for the image column.

Returns: a `StructType` for image column, `struct<origin:string, height:int, width:int, nChannels:int, mode:int, data:binary>`.

New in version 2.4.0.

`property imageFields`

Returns field names of image columns.

Returns: a list of field names.

New in version 2.3.0.

`property imageSchema`

Returns the image schema.

Returns: a `StructType` with a single column of images named "image" (nullable) and having the same type returned by `columnSchema()`.


Spark Mlib Utilities

Data Sources

← → ↺

spark.apache.org/docs/latest/ml-datasource#image-data-source

🔍 ☆

 3.1.1

Overview

Programming Guides ▾

API Docs ▾

Deploying ▾

More ▾

MLlib: Main Guide

- Basic statistics
- Data sources**
- Pipelines
- Extracting, transforming and selecting features
- Classification and Regression
- Clustering
- Collaborative filtering
- Frequent Pattern Mining
- Model selection and tuning
- Advanced topics

MLlib: RDD-based API Guide

- Data types
- Basic statistics
- Classification and regression
- Collaborative filtering
- Clustering
- Dimensionality reduction

Data sources

In this section, we introduce how to use data source in ML to load data. Besides some general data sources such as Parquet, CSV, JSON and JDBC, we also provide some specific data sources for ML.

Table of Contents

- [Image data source](#)
- [LIBSVM data source](#)

Image data source

This image data source is used to load image files from a directory, it can load compressed image (jpeg, png, etc.) into raw image representation via `ImageIO` in Java library. The loaded `DataFrame` has one `StructType` column: "image", containing image data stored as image schema. The schema of the image column is:

- origin: `StringType` (represents the file path of the image)
- height: `IntegerType` (height of the image)
- width: `IntegerType` (width of the image)
- nChannels: `IntegerType` (number of image channels)
- mode: `IntegerType` (OpenCV-compatible type)
- data: `BinaryType` (Image bytes in OpenCV-compatible order: row-wise BGR in most cases)

Scala

Java

Python

R

13

Spark Mlib Utilities

Data Sources

```
[3]: import findspark
      findspark.init()
      import pyspark
      from pyspark.sql import SparkSession
      spark = SparkSession.builder.getOrCreate()
```

Example of loading image data using the `kittens` dataset that ships with Spark

Refer to ML Main Guide for more info: <https://spark.apache.org/docs/latest/ml-datasource#image-data-source>

The schema of the image column is:

- origin: StringType (represents the file path of the image)
- height: IntegerType (height of the image)
- width: IntegerType (width of the image)
- nChannels: IntegerType (number of image channels)
- mode: IntegerType (OpenCV-compatible type)
- data: BinaryType (Image bytes in OpenCV-compatible order: row-wise BGR in most cases)

Spark Mlib Utilities

Data Sources

```
[22]: PATH = "D:/Spark/spark-3.0.1-bin-hadoop2.7/data/mllib/images/origin/kittens"
df = (
    spark.read.format("image")
    .option("dropInvalid", True)
    .load(PATH)
    .select("image.origin", "image.height", "image.width", "image.nChannels", "image.mode", "image.data")
)
df.toPandas()
```

```
[22]:
```

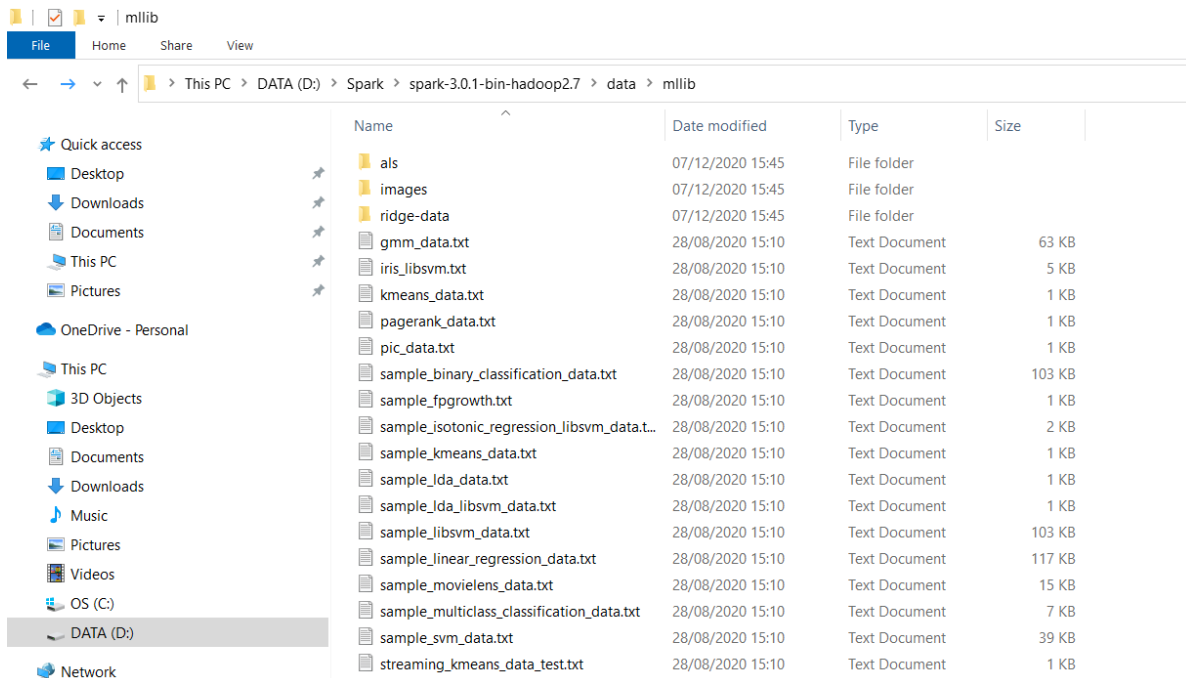
		origin	height	width	nChannels	mode	data
0	file:///usr/local/spark-2.4.3-bin-hadoop2.7/da...		311	300	3	16	[193, 193, 193, 194, 194, 194, 194, 194, ...
1	file:///usr/local/spark-2.4.3-bin-hadoop2.7/da...		313	199	3	16	[208, 229, 237, 202, 223, 231, 210, 231, 239, ...
2	file:///usr/local/spark-2.4.3-bin-hadoop2.7/da...		200	300	3	16	[88, 93, 96, 88, 93, 96, 88, 93, 96, 89, 94, 9...
3	file:///usr/local/spark-2.4.3-bin-hadoop2.7/da...		296	300	3	16	[203, 230, 244, 202, 229, 243, 201, 228, 242, ...

```
[13]: df.printSchema()

root
|-- image: struct (nullable = true)
|   |-- origin: string (nullable = true)
|   |-- height: integer (nullable = true)
|   |-- width: integer (nullable = true)
|   |-- nChannels: integer (nullable = true)
|   |-- mode: integer (nullable = true)
|   |-- data: binary (nullable = true)
```

Spark Mlib Utilities

Sample Data



The screenshot shows a Windows File Explorer window with the address bar displaying the path: > This PC > DATA (D:) > Spark > spark-3.0.1-bin-hadoop2.7 > data > mllib. The left sidebar shows the 'Quick access' pane with 'This PC' selected, and the 'DATA (D:)' drive highlighted in the 'This PC' section. The main pane displays a list of files and folders in the 'mllib' directory.

Name	Date modified	Type	Size
als	07/12/2020 15:45	File folder	
images	07/12/2020 15:45	File folder	
ridge-data	07/12/2020 15:45	File folder	
gmm_data.txt	28/08/2020 15:10	Text Document	63 KB
iris_libsvm.txt	28/08/2020 15:10	Text Document	5 KB
kmeans_data.txt	28/08/2020 15:10	Text Document	1 KB
pagerank_data.txt	28/08/2020 15:10	Text Document	1 KB
pic_data.txt	28/08/2020 15:10	Text Document	1 KB
sample_binary_classification_data.txt	28/08/2020 15:10	Text Document	103 KB
sample_fpgrowth.txt	28/08/2020 15:10	Text Document	1 KB
sample_isotonic_regression_libsvm_data.t...	28/08/2020 15:10	Text Document	2 KB
sample_kmeans_data.txt	28/08/2020 15:10	Text Document	1 KB
sample_lda_data.txt	28/08/2020 15:10	Text Document	1 KB
sample_lda_libsvm_data.txt	28/08/2020 15:10	Text Document	1 KB
sample_libsvm_data.txt	28/08/2020 15:10	Text Document	103 KB
sample_linear_regression_data.txt	28/08/2020 15:10	Text Document	117 KB
sample_movielens_data.txt	28/08/2020 15:10	Text Document	15 KB
sample_multiclass_classification_data.txt	28/08/2020 15:10	Text Document	7 KB
sample_svm_data.txt	28/08/2020 15:10	Text Document	39 KB
streaming_kmeans_data_test.txt	28/08/2020 15:10	Text Document	1 KB

Spark Mlib Utilities

Sample Data

Github: <https://github.com/apache/spark/tree/v3.0.1/data/mllib>

The screenshot shows the GitHub repository page for the Spark MLib sample data. The browser address bar shows the URL `github.com/apache/spark/tree/v3.0.1/data/mllib`. The repository path is `spark / data / mllib /`. The commit message is "WeichenXu123 and mengxr [SPARK-22666][ML][SQL] Spark datasource for image format" by user 9254492, dated Sep 6, 2018. The commit history is visible. The file list includes:

File	Description	Time
als	[SPARK-12247][ML][DOC] Documentation for spark.ml's ALS and collabora...	5 years ago
images	[SPARK-22666][ML][SQL] Spark datasource for image format	3 years ago
ridge-data	SPARK-2363. Clean MLib's sample data files	7 years ago
gmm_data.txt	[SPARK-5013] [MLlib] Added documentation and sample data file for Gau...	6 years ago
iris_libsvm.txt	[SPARK-14516][ML][FOLLOW-UP] Move ClusteringEvaluatorSuite test data ...	4 years ago
kmeans_data.txt	SPARK-2363. Clean MLib's sample data files	7 years ago
pagerank_data.txt	SPARK-2363. Clean MLib's sample data files	7 years ago
pic_data.txt	[SPARK-8758] [MLlib] Add Python user guide for PowerIterationClustering	6 years ago
sample_binary_classification_data.txt	[SPARK-1874][MLlib] Clean up MLib sample data	7 years ago
sample_fpgrowth.txt	[SPARK-5939][MLlib] make FPGrowth example app take parameters	6 years ago
sample_isotonic_regression_libsvm_data.txt	[SPARK-15608][ML][EXAMPLES][DOC] add examples and documents of mliso...	5 years ago
sample_kmeans_data.txt	[SPARK-14340][EXAMPLE][DOC] Update Examples and User Guide for ml.Bis...	5 years ago

Spark Mlib Utilities

Heppers

- <http://spark.apache.org/docs/3.0.1/api/python/pyspark.ml.html#module-pyspark.ml.util>

- | | |
|-------------------------|-----------------------|
| ● BaseReadWrite | ● JavaMLReadable |
| ● DefaultParamsReadable | ● JavaMLReader |
| ● DefaultParamsReader | ● JavaMLWritable |
| ● DefaultParamsWritable | ● JavaMLWriter |
| ● DefaultParamsWriter | ● JavaPredictionModel |
| ● GeneralJavaMLWritable | ● MLReadable |
| ● GeneralJavaMLWriter | ● MLReader |
| ● GeneralMLWriter | ● MLWritable |
| ● Identifiable | ● MLWriter |

Spark Mlib

Learning Resources

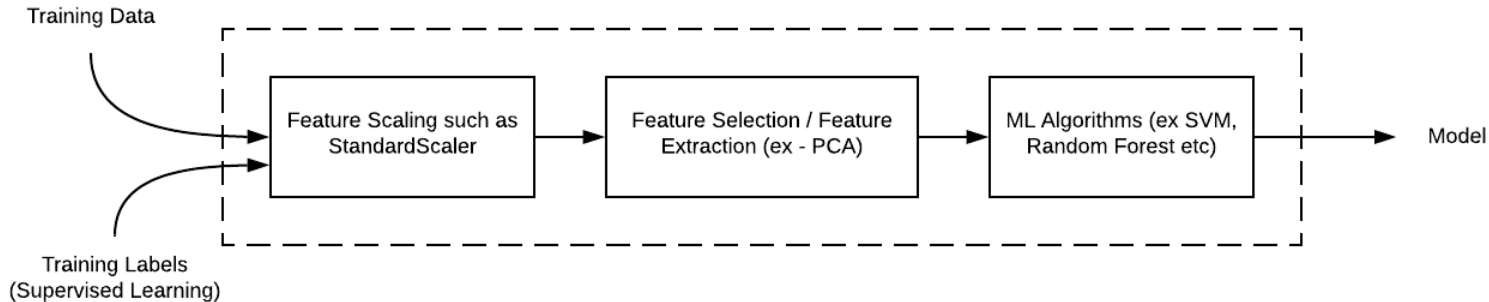
- Main guide
 - <http://spark.apache.org/docs/3.0.1/ml-guide.html>
- API guide
 - <http://spark.apache.org/docs/3.0.1/api/python/pyspark.ml.html>
- Github
 - <https://github.com/apache/spark/tree/v3.0.1/python/pyspark/ml>

Spark MLlib Pipelines

- Pipelines API
 - Provide a Uniform Set of High-level APIs Built on Top of DataFrames that Help Users Create and Tune Practical Machine Learning Pipelines
 - <https://spark.apache.org/docs/latest/ml-pipeline.html>

Spark MLlib Pipelines

- Algorithms Working Together: Assembled in a Pipeline



Machine Learning Pipeline

Spark MLlib Pipelines

5 Key Concepts

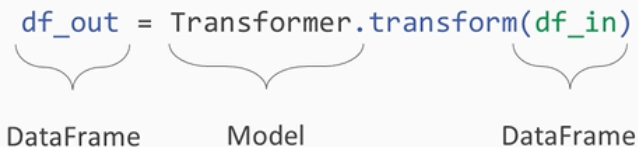
- DataFrame
 - Estimator
 - Transformer
 - Parameters
 - Pipeline
- } Predictor

Spark MLlib Pipelines

Transformer

- A Transformer is an abstraction that includes feature transformers and learned models
- Technically, a transformer implements a method `.transform()`, which converts one DataFrame into another, generally by appending one or more columns
- In code:

```
df_out = Transformer.transform(df_in)
```



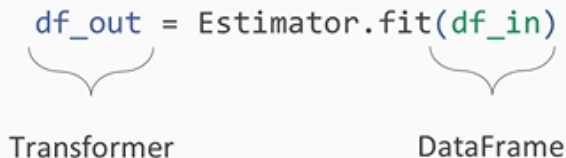
The diagram illustrates the types of the variables in the code snippet `df_out = Transformer.transform(df_in)`. Brackets are placed under each variable, and labels are placed below them: `df_out` is labeled "DataFrame", `Transformer` is labeled "Model", and `df_in` is labeled "DataFrame".

Spark MLlib Pipelines

Estimator

- An estimator abstracts the concept of a learning algorithm or any algorithm that fits or trains on data
- Technically, an Estimator implements a method `.fit()`, which accepts a `DataFrame` and produces a model which is a transformer
- In code:

```
df_out = Estimator.fit(df_in)
```



Transformer DataFrame

Spark MLlib Pipelines

Estimator vs Transformer

```
model_out = Estimator.fit(df_in)
```

Transformer

DataFrame

```
df_out = Transformer.transform(df_in)
```

DataFrame

Model

DataFrame

Spark MLlib Pipelines

Pipeline

- Stages:
 - Pipelines consist of a sequence of stages that run in order
 - Each stage is either a transformer or an estimator

.transform()

.fit()

```
class Pipeline
```

.fit(DataFrame)

Acts as an Estimator, consists of stages

Stages execute in order when .fit() is called

```
class PipelineModel
```

.transform(DataFrame)

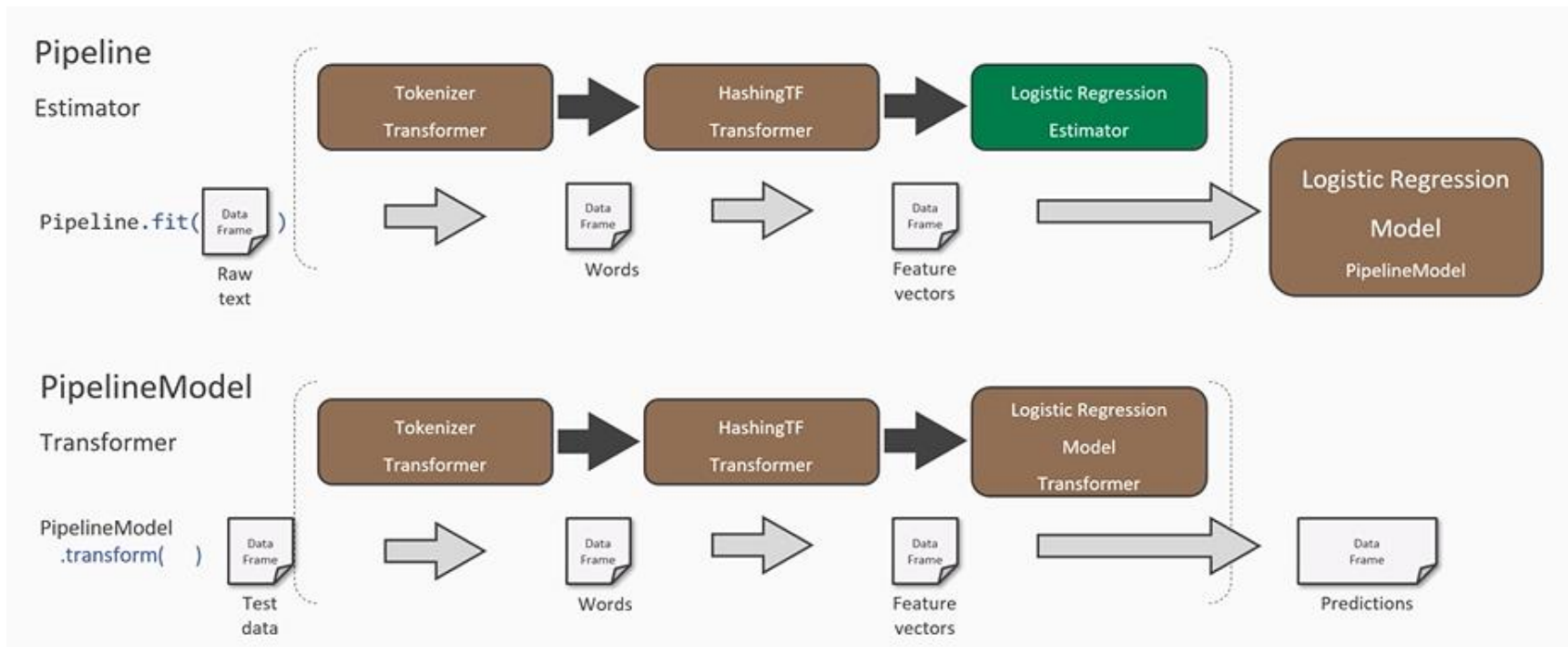
Represents a compiled pipeline with transformers and fitted models

A pipeline's fit() method, produces a PipelineModel

Acts as a transformer, consists of stages

Spark MLlib Pipelines

Example



Spark MLlib Pipelines

- Spark MLlib pipeline API:
 - MLlib standardizes APIs for its many machine learning algorithms to make it easier to combine multiple algorithms
 - The key concepts of pipelines are: Estimator, transformer, and parameter
 - Pipelines are a sequence of stages
 - Pipelines produce PipelineModels

Spark MLlib Pipelines

- Spark MLlib pipeline API:
 - A PipelineModel represents a compiled pipeline with transformers and fitted models
 - PipelineModels perform the same transformations before running data through the compiled model

Spark MLlib Pipelines

Classification and regression

- <https://spark.apache.org/docs/latest/ml-classification-regression.html>

The screenshot shows the Apache Spark 3.1.1 documentation page for MLlib Classification and Regression. The page has a navigation bar with links for Overview, Programming Guides, API Docs, Deploying, and More. A search bar is also present. The main content area is divided into two columns. The left column contains a 'MLlib: Main Guide' with links to Basic statistics, Data sources, Pipelines, Extracting, transforming and selecting features, Classification and Regression, Clustering, Collaborative filtering, Frequent Pattern Mining, Model selection and tuning, and Advanced topics. Below this is a 'MLlib: RDD-based API Guide' with links to Data types, Basic statistics, Classification and regression, Collaborative filtering, and Clustering. The right column contains a 'Table of Contents' with links to Classification (Logistic regression, Decision tree classifier, Random forest classifier, Gradient-boosted tree classifier, Multilayer perceptron classifier, Linear Support Vector Machine, One-vs-Rest classifier (a.k.a. One-vs-All), Naive Bayes, Factorization machines classifier) and Regression (Linear regression, Generalized linear regression, Decision tree regression, Random forest regression, Gradient-boosted tree regression, Survival regression, Isotonic regression).

← → ↺ spark.apache.org/docs/latest/ml-classification-regression.html

APACHE Spark 3.1.1 Overview Programming Guides API Docs Deploying More Search the docs

This page covers algorithms for Classification and Regression. It also includes sections discussing specific classes of algorithms, such as linear methods, trees, and ensembles.

Table of Contents

- Classification
 - Logistic regression
 - Binomial logistic regression
 - Multinomial logistic regression
 - Decision tree classifier
 - Random forest classifier
 - Gradient-boosted tree classifier
 - Multilayer perceptron classifier
 - Linear Support Vector Machine
 - One-vs-Rest classifier (a.k.a. One-vs-All)
 - Naive Bayes
 - Factorization machines classifier
- Regression
 - Linear regression
 - Generalized linear regression
 - Available families
 - Decision tree regression
 - Random forest regression
 - Gradient-boosted tree regression
 - Survival regression
 - Isotonic regression

Spark MLlib Pipelines

Classification and regression

```
[2]: import findspark
findspark.init()
import pyspark
from pyspark.sql import SparkSession
spark = SparkSession.builder.getOrCreate()

# Load and parse the data file, converting it to a DataFrame.
sample_libsvm_data = spark.read.format("libsvm").load("D:/Spark/spark-3.0.1-bin-hadoop2.7/data/mllib/sample_libsvm_data.txt")
```

Spark MLlib Pipelines

DecisionTreeClassifier

```
from pyspark.ml import Pipeline
from pyspark.ml.classification import DecisionTreeClassifier
from pyspark.ml.feature import StringIndexer, VectorIndexer
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
```

```
sample_libsvm_data.show(5)
```

```
+-----+-----+
|label|      features|
+-----+-----+
|  0.0|(692,[127,128,129...|
|  1.0|(692,[158,159,160...|
|  1.0|(692,[124,125,126...|
|  1.0|(692,[152,153,154...|
|  1.0|(692,[151,152,153...|
```

```
+-----+-----+
only showing top 5 rows
```


Spark MLlib Pipelines

DecisionTreeClassifier

```
: # Creating our stages:

# STAGE 1:
# Index labels, adding metadata to the label column.
# Fit on whole dataset to include all labels in index.
label_indexer = StringIndexer(inputCol="label", outputCol="indexedLabel").fit(
    sample_libsvm_data
)

# STAGE 2:
# Automatically identify categorical features, and index them.
# We specify maxCategories so features with > 4 distinct values are treated as continuous.
feature_indexer = VectorIndexer(
    inputCol="features", outputCol="indexedFeatures", maxCategories=4
).fit(sample_libsvm_data)

# STAGE 3:
# Train a DecisionTree model.
decision_tree_classifier_model = DecisionTreeClassifier(
    labelCol="indexedLabel", featuresCol="indexedFeatures"
)

print(type(decision_tree_classifier_model))

<class 'pyspark.ml.classification.DecisionTreeClassifier'>
```

Spark MLlib Pipelines

DecisionTreeClassifier

```
# Creating our Pipeline:  
  
# Chain indexers and tree in a Pipeline  
pipeline = Pipeline(  
    stages=[  
        label_indexer, # STAGE 1  
        feature_indexer, # STAGE 2  
        decision_tree_classifier_model, # STAGE 3  
    ]  
)
```

Spark MLlib Pipelines

DecisionTreeClassifier

```
# Split the data into training and test sets (30% held out for testing)
(training_data, test_data) = sample_libsvm_data.randomSplit([0.7, 0.3])

# Train model. This also runs the indexers.
model = pipeline.fit(training_data)

# Make predictions.
predictions = model.transform(test_data)

# Select example rows to display.
predictions.select("prediction", "indexedLabel", "features").show(5)

# Select (prediction, true label) and compute test error
evaluator = MulticlassClassificationEvaluator(
    labelCol="indexedLabel", predictionCol="prediction", metricName="accuracy"
)
accuracy = evaluator.evaluate(predictions)
print(f"Test Error = {1.0 - accuracy:.5f} ")
```

```
+-----+-----+-----+
|prediction|indexedLabel|      features|
+-----+-----+-----+
|      1.0|          1.0|(692,[95,96,97,12...|
|      1.0|          1.0|(692,[123,124,125...|
|      1.0|          1.0|(692,[124,125,126...|
|      0.0|          1.0|(692,[124,125,126...|
|      1.0|          1.0|(692,[126,127,128...|
+-----+-----+-----+
```

only showing top 5 rows

Test Error = 0.19355

Spark MLlib Pipelines

DecisionTreeClassifier

```
# You can see that the Pipeline and the PipelineModel have the same stages  
print(pipeline.getStages())  
print(model.stages)
```

```
[StringIndexerModel: uid=StringIndexer_bde466415729, handleInvalid=error, VectorIndexerModel: uid=VectorIndexer_5be5f8abe571, numFeatures=692,  
handleInvalid=error, DecisionTreeClassifier_2381fa7fa6a8]  
[StringIndexerModel: uid=StringIndexer_bde466415729, handleInvalid=error, VectorIndexerModel: uid=VectorIndexer_5be5f8abe571, numFeatures=692,  
handleInvalid=error, DecisionTreeClassificationModel: uid=DecisionTreeClassifier_2381fa7fa6a8, depth=2, numNodes=5, numClasses=2, numFeatures=6  
92]
```

Spark MLlib Pipelines

Random Forest Regression

```
from pyspark.ml import Pipeline
from pyspark.ml.regression import RandomForestRegressor
from pyspark.ml.feature import VectorIndexer
from pyspark.ml.evaluation import RegressionEvaluator

sample_libsvm_data.show(5)
```

```
+-----+-----+
|label|          features|
+-----+-----+
|  0.0|(692,[127,128,129...|
|  1.0|(692,[158,159,160...|
|  1.0|(692,[124,125,126...|
|  1.0|(692,[152,153,154...|
|  1.0|(692,[151,152,153...|
+-----+-----+
```

only showing top 5 rows

Spark MLlib Pipelines

Random Forest Regression

```
# Creating our stages:

# STAGE 1:
# Automatically identify categorical features, and index them.
feature_indexer = VectorIndexer(
    inputCol="features",
    outputCol="indexedFeatures",
    # Set maxCategories so features with > 4 distinct values are treated as continuous.
    maxCategories=4,
).fit(sample_libsvm_data)

# STAGE 2:
# Train a RandomForest model.
random_forest_model = RandomForestRegressor(featuresCol="indexedFeatures")
```

Spark MLlib Pipelines

Random Forest Regression

```
# Creating our Pipeline:  
# Chain indexer and forest in a Pipeline  
pipeline = Pipeline(stages=[feature_indexer, random_forest_model])
```

Spark MLlib Pipelines

Random Forest Regression

```
# Split the data into training and test sets (30% held out for testing)
(training_data, test_data) = sample_libsvm_data.randomSplit([0.7, 0.3])

# Train model. This also runs the indexer.
model = pipeline.fit(training_data)

# Make predictions.
predictions = model.transform(test_data)

# Select example rows to display.
predictions.select("prediction", "label", "features").show(5)

# Select (prediction, true label) and compute test error
evaluator = RegressionEvaluator(
    labelCol="label", predictionCol="prediction", metricName="rmse"
)
rmse = evaluator.evaluate(predictions)
print("Root Mean Squared Error (RMSE) on test data = %g" % rmse)
```

```
+-----+-----+-----+
|prediction|label|          features|
+-----+-----+-----+
|    0.05|  0.0|(692,[122,123,148...|
|    0.0|  0.0|(692,[123,124,125...|
|    0.05|  0.0|(692,[124,125,126...|
|    0.0|  0.0|(692,[125,126,127...|
|    0.0|  0.0|(692,[126,127,128...|
+-----+-----+-----+
```

only showing top 5 rows

Root Mean Squared Error (RMSE) on test data = 0.0310087

Spark MLlib Pipelines

Random Forest Regression

You can see that the Pipeline and the PipelineModel have the same stages

```
print(pipeline.getStages())  
print(model.stages)
```

```
[VectorIndexerModel: uid=VectorIndexer_98f042bd5bbd, numFeatures=692, handleInvalid=error, RandomForestRegressor_77ce3e646d07]
```

```
[VectorIndexerModel: uid=VectorIndexer_98f042bd5bbd, numFeatures=692, handleInvalid=error, RandomForestRegressionModel: uid=RandomForestRegressor_77ce3e646d07, numTrees=20, numFeatures=692]
```

The last stage in a PipelineModel is usually the most informative

```
print(model.stages[-1])
```

```
RandomForestRegressionModel: uid=RandomForestRegressor_77ce3e646d07, numTrees=20, numFeatures=692
```

Here you can see that pipeline and model are Pipeline and PipelineModel classes

```
print("pipeline:", type(pipeline))  
print("model:", type(model))
```

```
pipeline: <class 'pyspark.ml.pipeline.Pipeline'>  
model: <class 'pyspark.ml.pipeline.PipelineModel'>
```

Spark MLlib Pipelines

Classification and regression

- <https://spark.apache.org/docs/latest/ml-classification-regression.html>



Cảm ơn đã theo dõi

Chúng tôi hy vọng cùng nhau đi đến thành công.