# 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 Mllib Utilities

Linear Algebra

- API Guide

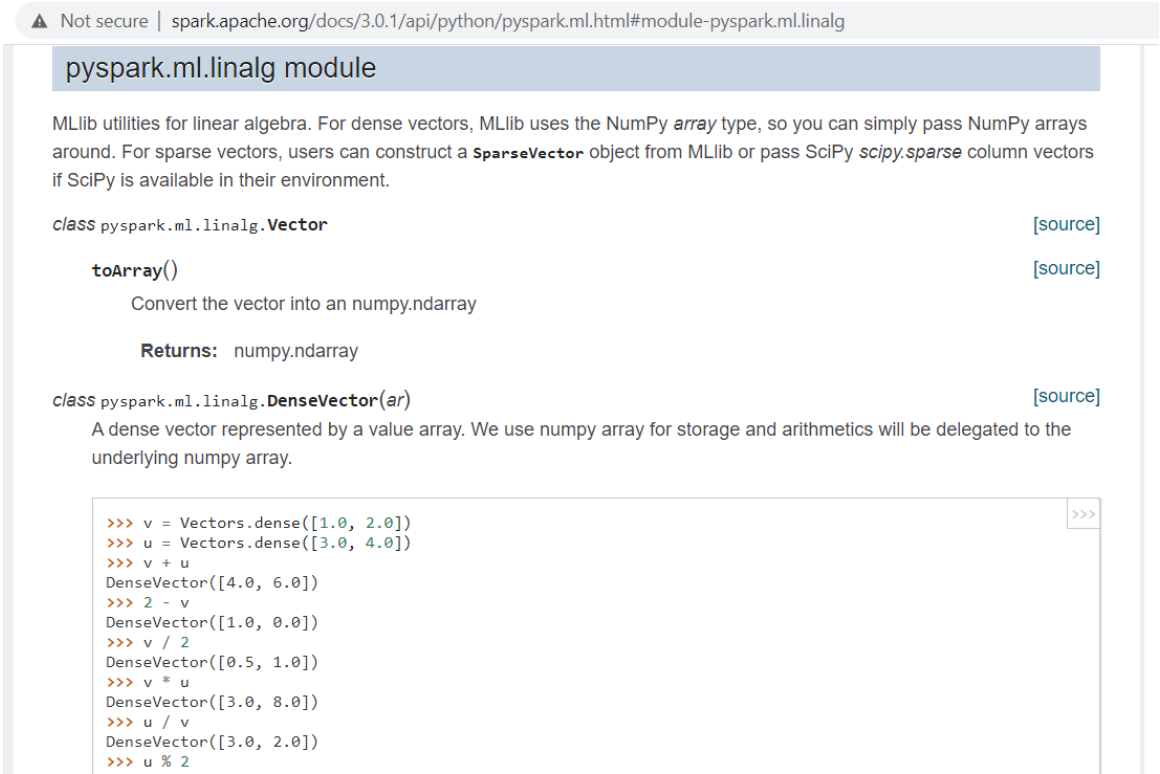
  http://spark.apache.org/docs/3.0.1/api/python/pyspark.ml.html#module-pyspark.ml.linalg

- MLlib RDD-Base Guide

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

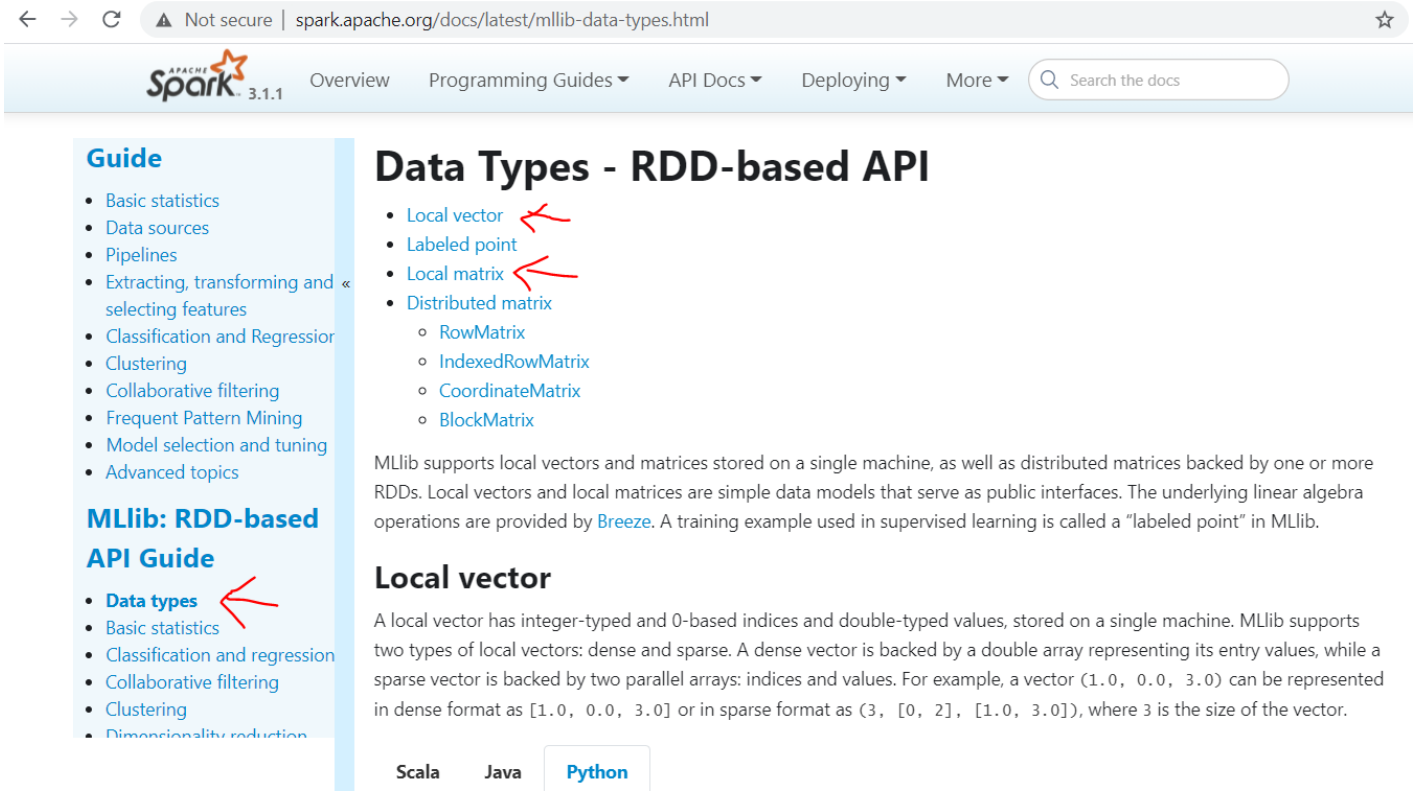
# Spark Mllib Utilities

## Linear Algebra



# Spark Mllib Utilities

## Linear Algebra



# Spark Mllib Utilities

## Linear Algebra

### Vectors

- Vectors (main class to use)
- DenseVector: MLlib 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 Mllib Utilities

## Linear Algebra

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

# Spark Mllib 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 Mllib 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 Mllib Utilities

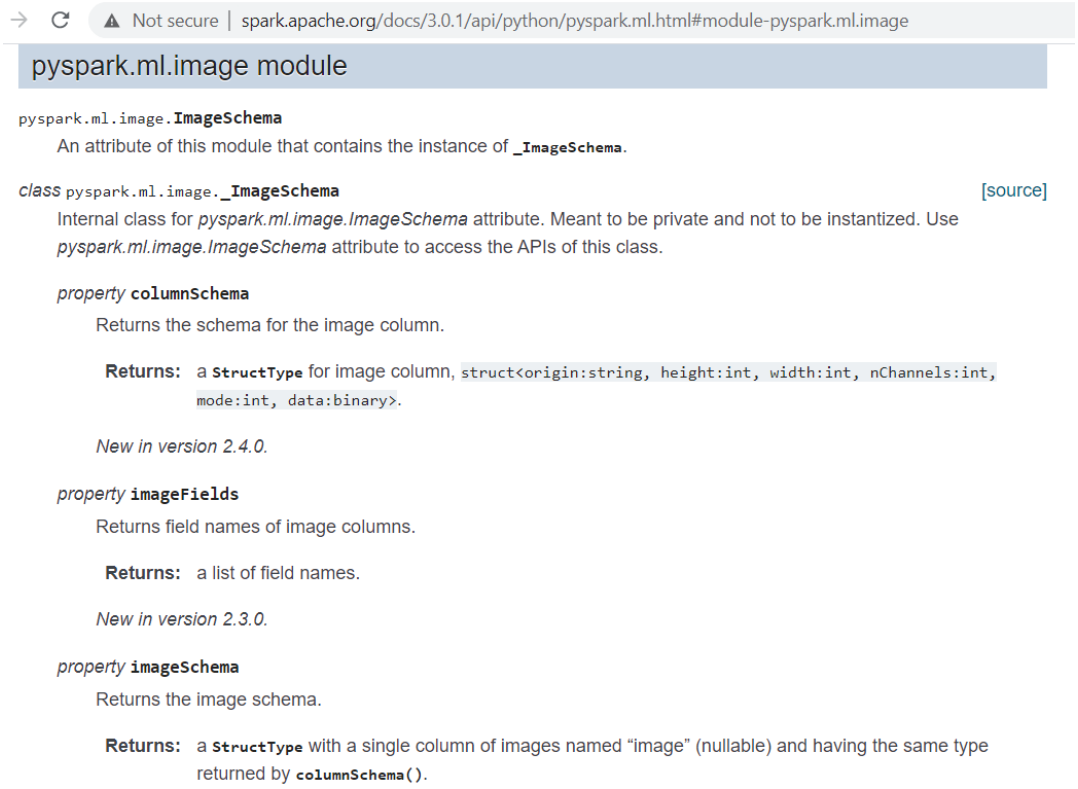
## Data Sources

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

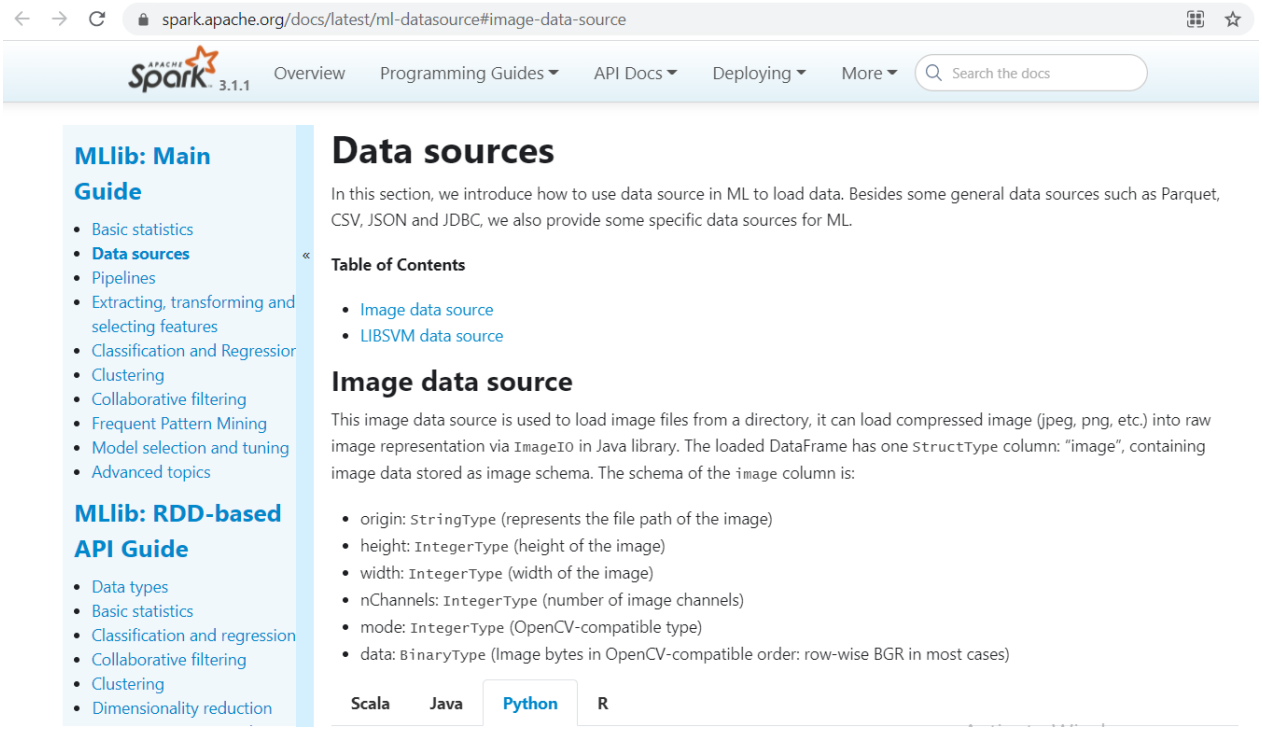
# Spark Mllib Utilities

## Data Sources



# Spark Mllib Utilities

## Data Sources



# Spark Mllib 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 Mllib 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]:

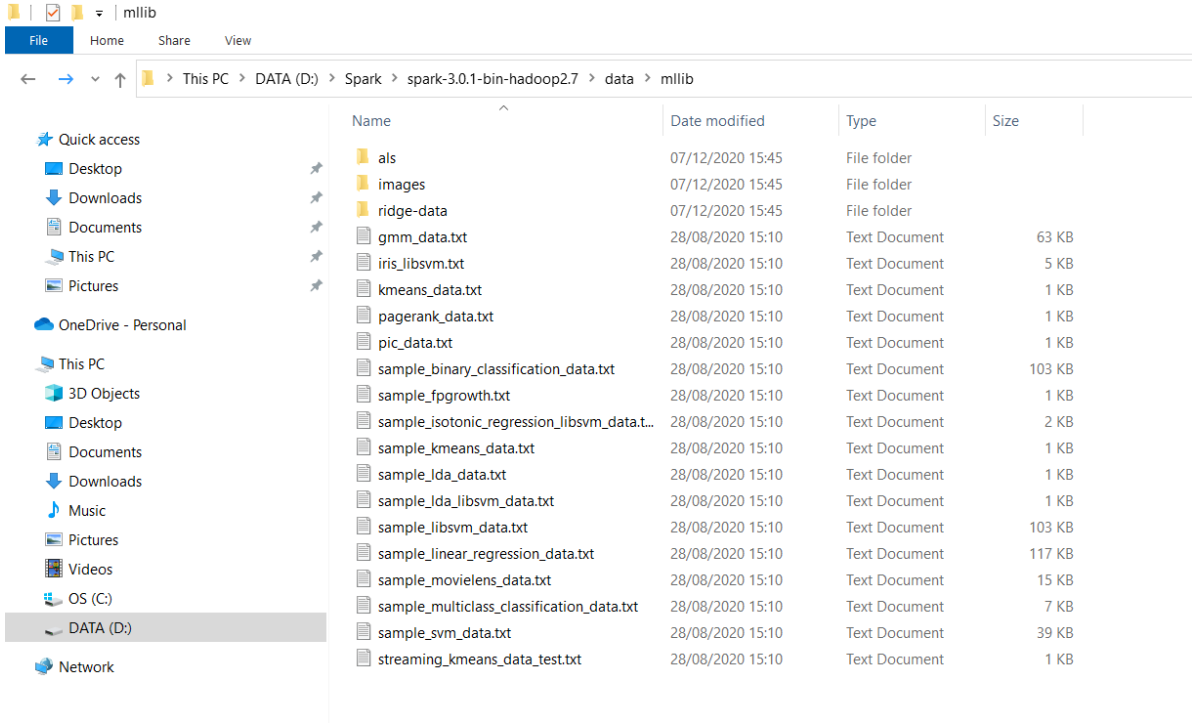
| | origin | height | width | nChannels | mode | data |
|---|---|---|---|---|---|---|
| **0** | file:///usr/local/spark-2.4.3-bin-hadoop2.7/da... | 311 | 300 | 3 | 16 | [193, 193, 193, 194, 194, 194, 194, 194, 194, ... |
| **1** | file:///usr/local/spark-2.4.3-bin-hadoop2.7/da... | 313 | 199 | 3 | 16 | [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 Mllib Utilities

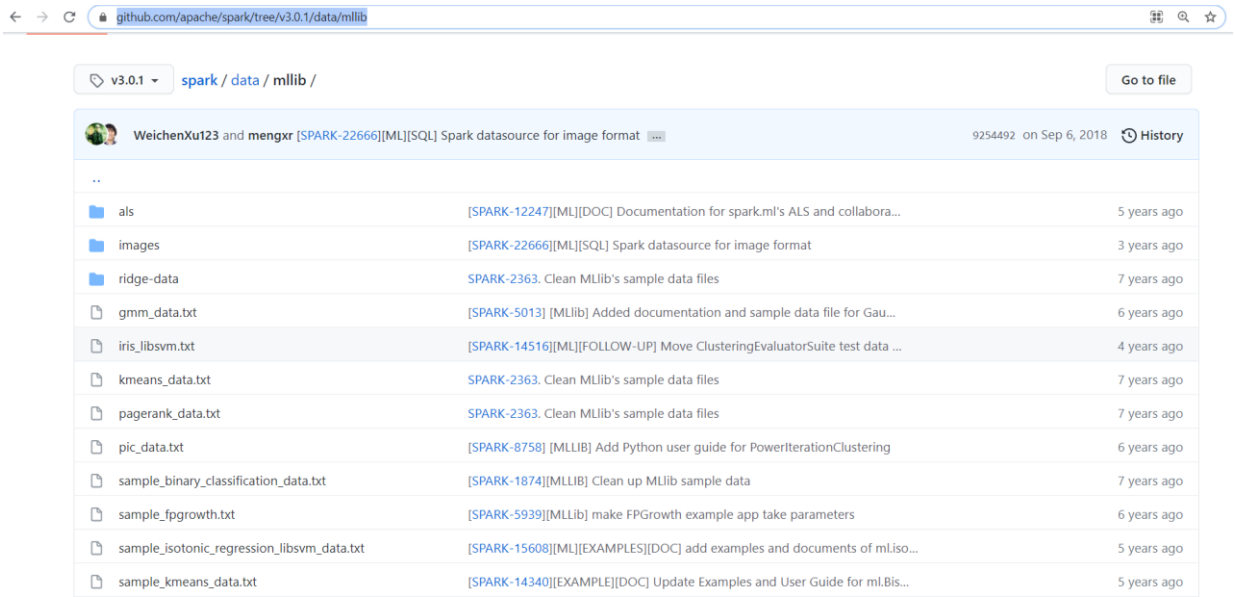
## Sample Data



# Spark Mllib Utilities

## Sample Data

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



# Spark Mllib Utilities

## Heppers

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

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

# Spark Mllib

## Learning Resources

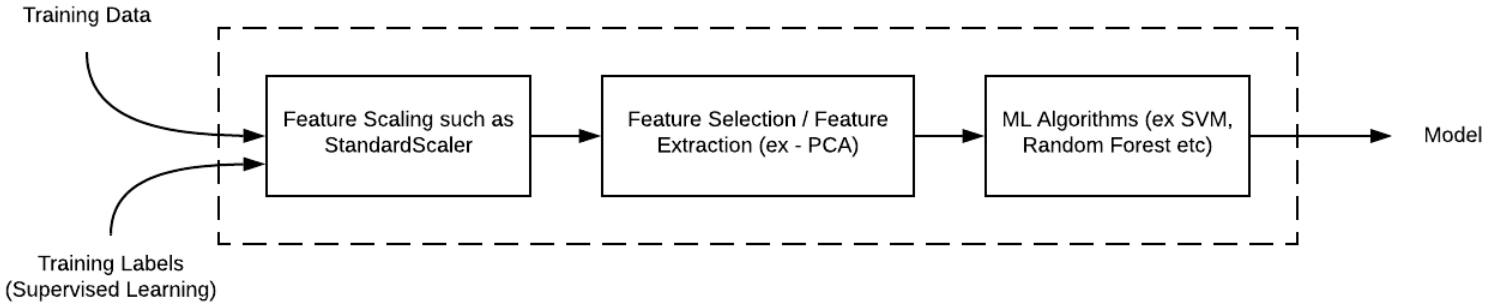
- Main guide
  - http://spark.apache.org/docs/3.0.1/ml-guide.html
- API guide
  - http://spark.apache.org/docs/3.0.1/api/python/pyspark.ml.html
- Github
  - https://github.com/apache/spark/tree/v3.0.1/python/pyspark/ml

# 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

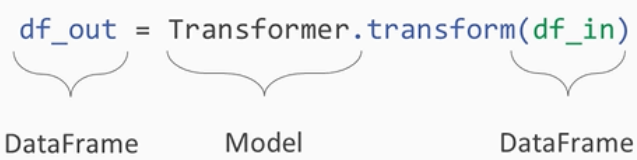
- DataFrame
- Estimator
- Transformer
- Parameters
- Pipeline

Estimator, Transformer } Predictor

# Spark MLlib Pipelines

## Transfomer

- 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:



# 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:

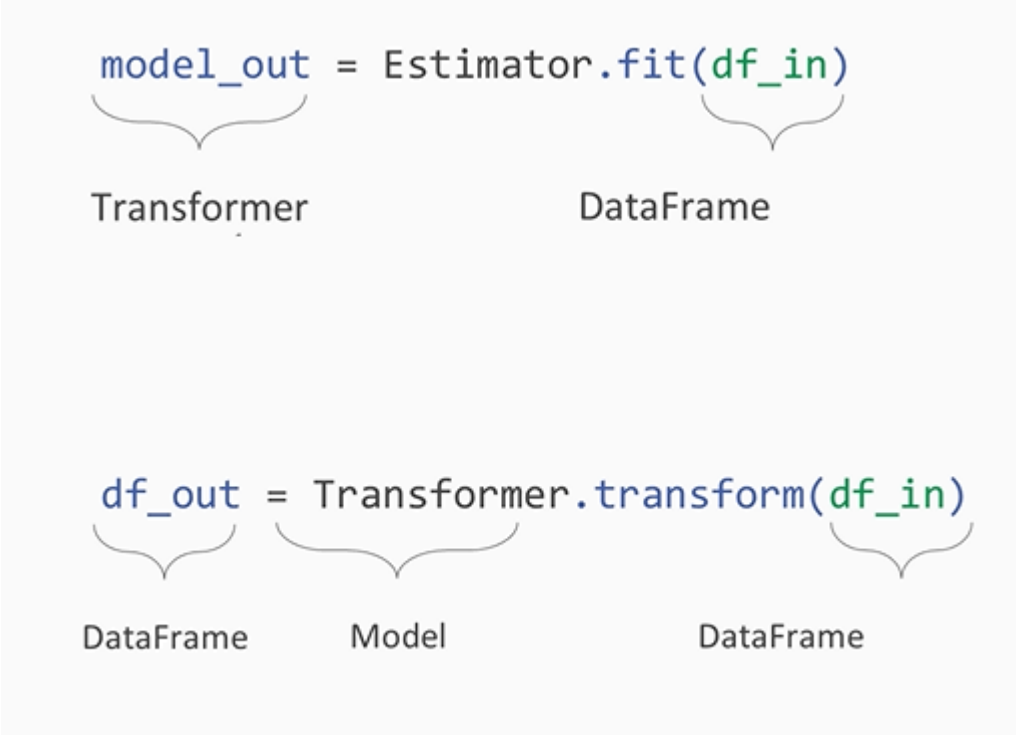
```
df_out = Estimator.fit(df_in)
```

Transformer

DataFrame

# Spark MLlib Pipelines

## Estimator vs Transformer



# 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)

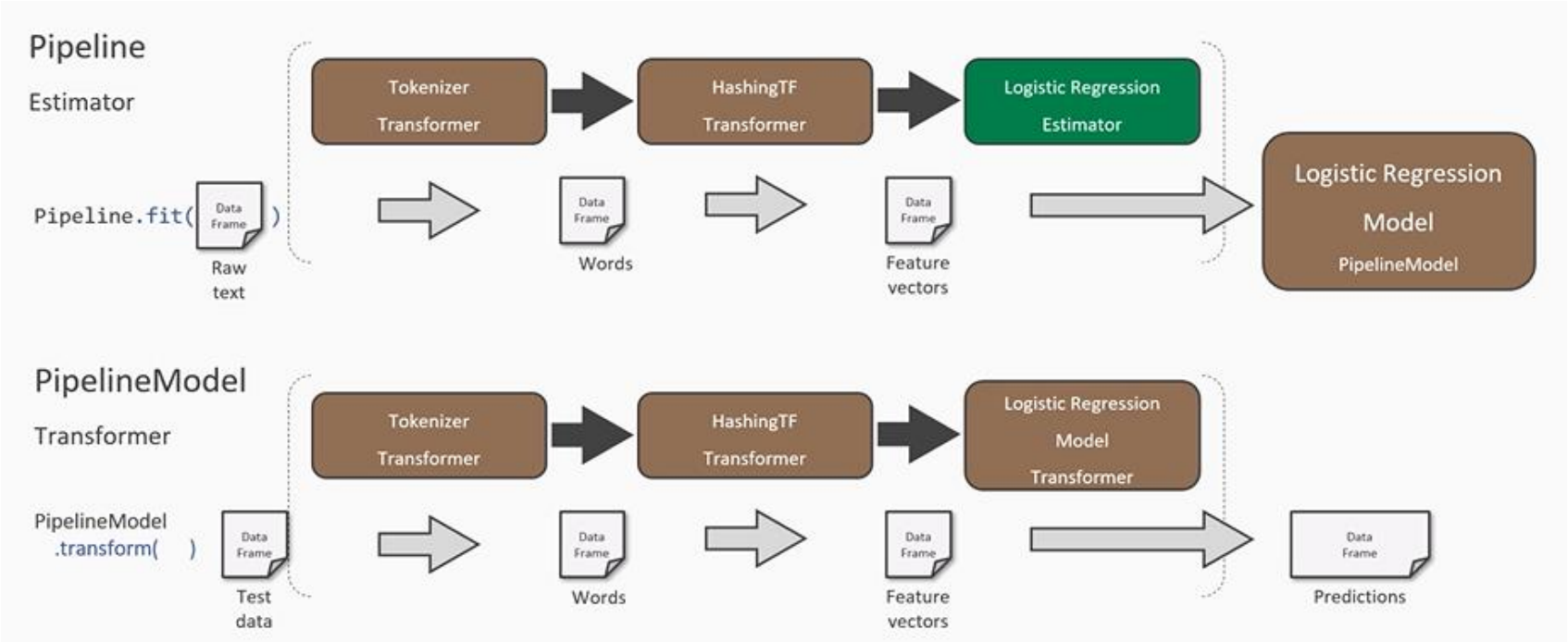
Represents a compiled pipeline with transformers and fitted models

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

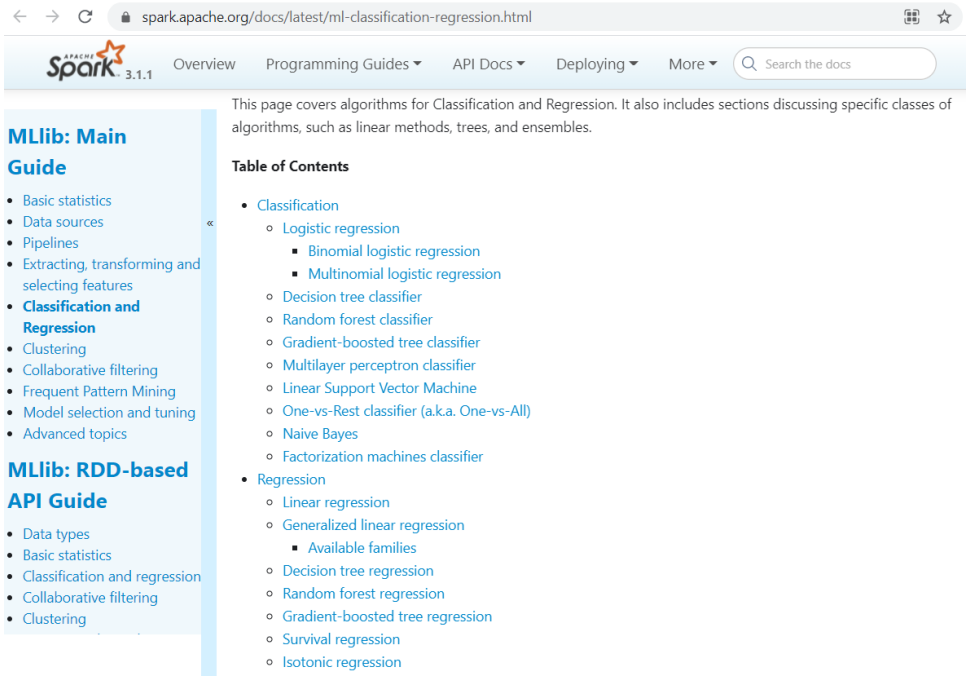
# Spark MLlib Pipelines

- 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



# 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 – Feature Transformers

StringIndexer encodes a string column of labels to a column of label indices

```
label_indexer = StringIndexer(inputCol="label", outputCol="indexedLabel").fit(sample_libsvm_data)

print(type(label_indexer))
label_indexer.transform(sample_libsvm_data).show()
```

| id | category |
|---|---|
| 0 | a |
| 1 | b |
| 2 | c |
| 3 | a |
| 4 | a |
| 5 | c |

| id | category | categoryIndex |
|---|---|---|
| 0 | a | 0.0 |
| 1 | b | 2.0 |
| 2 | c | 1.0 |
| 3 | a | 0.0 |
| 4 | a | 0.0 |
| 5 | c | 1.0 |

```
<class 'pyspark.ml.feature.StringIndexerModel'>
+-----+--------------------+------------+
|label|            features|indexedLabel|
+-----+--------------------+------------+
|  0.0|(692,[127,128,129...|         1.0|
|  1.0|(692,[158,159,160...|         0.0|
|  1.0|(692,[124,125,126...|         0.0|
|  1.0|(692,[152,153,154...|         0.0|
|  1.0|(692,[151,152,153...|         0.0|
+-----+--------------------+------------+
```

# Spark MLlib Pipelines

## DecisionTreeClassifier – Feature Transformers

VectorIndexer helps index categorical features in datasets of Vectors. It can both automatically decide which features are categorical and convert original values to category indices.

```
# 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)

feature_indexer.transform(sample_libsvm_data).show(5,truncate=50)
```

```
+-----+--------------------------------------------------+--------------------------------------------------+
|label|                                          features|                                   indexedFeatures|
+-----+--------------------------------------------------+--------------------------------------------------+
|  0.0|(692,[127,128,129,130,131,154,155,156,157,158,1...|(692,[127,128,129,130,131,154,155,156,157,158,1...|
|  1.0|(692,[158,159,160,161,185,186,187,188,189,213,2...|(692,[158,159,160,161,185,186,187,188,189,213,2...|
|  1.0|(692,[124,125,126,127,151,152,153,154,155,179,1...|(692,[124,125,126,127,151,152,153,154,155,179,1...|
|  1.0|(692,[152,153,154,180,181,182,183,208,209,210,2...|(692,[152,153,154,180,181,182,183,208,209,210,2...|
|  1.0|(692,[151,152,153,154,179,180,181,182,208,209,2...|(692,[151,152,153,154,179,180,181,182,208,209,2...|
+-----+--------------------------------------------------+--------------------------------------------------+
```

# 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.
decission_tree_classifier_model = DecisionTreeClassifier(
    labelCol="indexedLabel", featuresCol="indexedFeatures"
)

print(type(decission_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
        decission_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=692]
```

# Spark MLlib Pipelines

## DecisionTreeClassifier

```
decission_tree_classifier_model = DecisionTreeClassifier(labelCol="label", featuresCol="features")

pipeline = Pipeline( stages=[decission_tree_classifier_model, ])

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

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

predictions.show()
```

```
+----------+-----+--------------------+
|prediction|label|            features|
+----------+-----+--------------------+
|       1.0|  0.0|(692,[98,99,100,1...|
|       0.0|  0.0|(692,[100,101,102...|
|       0.0|  0.0|(692,[122,123,124...|
|       0.0|  0.0|(692,[123,124,125...|
|       0.0|  0.0|(692,[124,125,126...|
+----------+-----+--------------------+
```

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

```
[DecisionTreeClassifier_7387b0a8a12c]
[DecisionTreeClassificationModel: uid=DecisionTreeClassifier_7387b0a8a12c, depth=2, numNodes=5, numClasses=2, numFeatures=692]
```

# 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=RandomForestRegress
or_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

# Spark MLlib Persistence

- Saving and loading of algorithms, models, and pipelines


**R** uses a modified format

# Spark MLlib Persistence

Saving/Writing

```
Pipeline.write().save(path)
Pipeline.save(path)
```

class MLWriter

Loading/Reading

```
Pipeline.read().load(path)
Pipeline.load(path)
```

class MLReader

# Spark MLlib Parameters

- Param: A named parameter with self-contained documentation

  `.extractParams()`

- ParamMap: a set of (parameter, value) pairs

  `.extractParamMap()`

# Spark MLlib Parameters

- MLlib Estimators and Transformers use a uniform API for specifying parameters.
- A Param is a named parameter with self-contained documentation. A ParamMap is a set of (parameter, value) pairs.
- There are two main ways to pass parameters to an algorithm:
    - Set parameters for an instance. E.g., if lr is an instance of LogisticRegression, one could call lr.setMaxIter(10) to make lr.fit() use at most 10 iterations. This API resembles the API used in spark.mllib package.
    - Pass a ParamMap to .fit() or .transform(). Any parameters in the ParamMap will override parameters previously specified via setter methods.
- Parameters belong to specific instances of Estimators and Transformers. E.g. if we have two instances lr1 and lr2, then we can build a ParamMap with both maxIter parameters: ParamMap({lr1.maxIter: 10, lr2.maxIter:  20}). This is useful if there are two algorithms with the maxIter parameter in a Pipeline.

# Spark MLlib Parameters

```
from IPython.core.display import display, HTML
from pyspark.sql import SparkSession
from pyspark.ml.linalg import Vectors
from pyspark.ml.classification import LogisticRegression

spark = SparkSession.builder.getOrCreate()

# Prepare training data from a list of (label, features) tuples.
training = spark.createDataFrame(
    [
        (1.0, Vectors.dense([0.0, 1.1, 0.1])),
        (0.0, Vectors.dense([2.0, 1.0, -1.0])),
        (0.0, Vectors.dense([2.0, 1.3, 1.0])),
        (1.0, Vectors.dense([0.0, 1.2, -0.5])),
    ],
    ["label", "features"],
)

# Prepare test data
test = spark.createDataFrame(
    [
        (1.0, Vectors.dense([-1.0, 1.5, 1.3])),
        (0.0, Vectors.dense([3.0, 2.0, -0.1])),
        (1.0, Vectors.dense([0.0, 2.2, -1.5])),
    ],
    ["label", "features"],
)
```

# Spark MLlib Parameters

```
# Create a LogisticRegression instance. This instance is an Estimator.
# maxIter and regParam are parameters
lr = LogisticRegression(maxIter=10, regParam=0.01)

# Print out the parameters, documentation, and any default values.
print(f"LogisticRegression parameters:\n{lr.explainParams()}")
```

```
LogisticRegression parameters:
aggregationDepth: suggested depth for treeAggregate (>= 2). (default: 2)
elasticNetParam: the ElasticNet mixing parameter, in range [0, 1]. For alpha = 0, the penalty is an L2 penalty. For alpha = 1, it is an L1 penalty. (default: 0.0)
family: The name of family which is a description of the label distribution to be used in the model. Supported options: auto, binomial, multinomial (default: auto)
featuresCol: features column name. (default: features)
fitIntercept: whether to fit an intercept term. (default: True)
labelCol: label column name. (default: label)
lowerBoundsOnCoefficients: The lower bounds on coefficients if fitting under bound constrained optimization. The bound matrix must be compatible with the shape (1, n
umber of features) for binomial regression, or (number of classes, number of features) for multinomial regression. (undefined)
lowerBoundsOnIntercepts: The lower bounds on intercepts if fitting under bound constrained optimization. The bounds vector size must beequal with 1 for binomial regr
ession, or the number oflasses for multinomial regression. (undefined)
maxIter: max number of iterations (>= 0). (default: 100, current: 10)
predictionCol: prediction column name. (default: prediction)
probabilityCol: Column name for predicted class conditional probabilities. Note: Not all models output well-calibrated probability estimates! These probabilities sho
uld be treated as confidences, not precise probabilities. (default: probability)
rawPredictionCol: raw prediction (a.k.a. confidence) column name. (default: rawPrediction)
regParam: regularization parameter (>= 0). (default: 0.0, current: 0.01)
standardization: whether to standardize the training features before fitting the model. (default: True)
threshold: Threshold in binary classification prediction, in range [0, 1]. If threshold and thresholds are both set, they must match.e.g. if threshold is p, then thr
esholds must be equal to [1-p, p]. (default: 0.5)
thresholds: Thresholds in multi-class classification to adjust the probability of predicting each class. Array must have length equal to the number of classes, with
values > 0, excepting that at most one value may be 0. The class with largest value p/t is predicted, where p is the original probability of that class and t is the
class's threshold. (undefined)
```

# Spark MLlib Parameters

```
print_explainParams(lr)
```

### Parameters for: LogisticRegression_49aa5a3b7809

**aggregationDepth**

suggested depth for treeAggregate (>= 2). (default: 2)

**elasticNetParam**

the ElasticNet mixing parameter, in range [0, 1]. For alpha = 0, the penalty is an L2 penalty. For alpha = 1, it is an L1 penalty. (default: 0.0)

**family**

The name of family which is a description of the label distribution to be used in the model. Supported options: auto, binomial, multinomial (default: auto)

**featuresCol**

features column name. (default: features)

**fitIntercept**

whether to fit an intercept term. (default: True)

**labelCol**

label column name. (default: label)

**lowerBoundsOnCoefficients**

The lower bounds on coefficients if fitting under bound constrained optimization. The bound matrix must be compatible with the shape (1, number of features) for binomial regression, or (number of classes, number of features) for multinomial regression. (undefined)

**lowerBoundsOnIntercepts**

The lower bounds on intercepts if fitting under bound constrained optimization. The bounds vector size must beequal with 1 for binomial regression, or the number oflasses for multinomial regression. (undefined)

# Spark MLlib Parameters

```
# Learn a LogisticRegression model. This uses the parameters stored in lr.
model1 = lr.fit(training)

# Since model1 is a Model (i.e., a transformer produced by an Estimator),
# we can view the parameters it used during fit().
# This prints the parameter (name: value) pairs, where names are unique IDs for this
# LogisticRegression instance.
print("Model 1 was fit using parameters: ")
print(model1.extractParamMap())
```

```
Model 1 was fit using parameters:
{Param(parent='LogisticRegression_49aa5a3b7809', name='aggregationDepth', doc='suggested depth for treeAggregate (>= 2)'): 2, Param(parent='LogisticRegression_49aa5a
3b7809', name='elasticNetParam', doc='the ElasticNet mixing parameter, in range [0, 1]. For alpha = 0, the penalty is an L2 penalty. For alpha = 1, it is an L1 penal
ty'): 0.0, Param(parent='LogisticRegression_49aa5a3b7809', name='family', doc='The name of family which is a description of the label distribution to be used in the
model. Supported options: auto, binomial, multinomial.'): 'auto', Param(parent='LogisticRegression_49aa5a3b7809', name='featuresCol', doc='features column name'): 'f
eatures', Param(parent='LogisticRegression_49aa5a3b7809', name='fitIntercept', doc='whether to fit an intercept term'): True, Param(parent='LogisticRegression_49aa5a
3b7809', name='labelCol', doc='label column name'): 'label', Param(parent='LogisticRegression_49aa5a3b7809', name='maxIter', doc='maximum number of iterations (>=
0)'): 10, Param(parent='LogisticRegression_49aa5a3b7809', name='predictionCol', doc='prediction column name'): 'prediction', Param(parent='LogisticRegression_49aa5a3
b7809', name='probabilityCol', doc='Column name for predicted class conditional probabilities. Note: Not all models output well-calibrated probability estimates! The
se probabilities should be treated as confidences, not precise probabilities'): 'probability', Param(parent='LogisticRegression_49aa5a3b7809', name='rawPredictionCo
l', doc='raw prediction (a.k.a. confidence) column name'): 'rawPrediction', Param(parent='LogisticRegression_49aa5a3b7809', name='regParam', doc='regularization para
meter (>= 0)'): 0.01, Param(parent='LogisticRegression_49aa5a3b7809', name='standardization', doc='whether to standardize the training features before fitting the mo
del'): True, Param(parent='LogisticRegression_49aa5a3b7809', name='threshold', doc='threshold in binary classification prediction, in range [0, 1]'): 0.5, Param(pare
nt='LogisticRegression_49aa5a3b7809', name='tol', doc='the convergence tolerance for iterative algorithms (>= 0)'): 1e-06}
```

# Spark MLlib Parameters

```
print_explainParamMap(model1)
```

**Parameter Map**

LogisticRegressionModel: uid = LogisticRegression_49aa5a3b7809, numClasses = 2, numFeatures = 3

**aggregationDepth**

doc: *suggested depth for treeAggregate (>= 2)*

value: 2

**elasticNetParam**

doc: *the ElasticNet mixing parameter, in range [0, 1]. For alpha = 0, the penalty is an L2 penalty. For alpha = 1, it is an L1 penalty*

value: 0.0

**family**

doc: *The name of family which is a description of the label distribution to be used in the model. Supported options: auto, binomial, multinomial.*

value: auto

**featuresCol**

doc: *features column name*

value: features

**fitIntercept**

doc: *whether to fit an intercept term*

value: True

# Spark MLlib Parameters

```python
# We may alternatively specify parameters using a Python dictionary as a paramMap
paramMap = {lr.maxIter: 20}
paramMap[lr.maxIter] = 30  # Specify 1 Param, overwriting the original maxIter.
paramMap.update({lr.regParam: 0.1, lr.threshold: 0.55})  # Specify multiple Params.

# You can combine paramMaps, which are python dictionaries.
paramMap2 = {lr.probabilityCol: "myProbability"}  # Change output column name
paramMap2 = {lr.maxIter: 10}
paramMapCombined = paramMap.copy()
paramMapCombined.update(paramMap2)

# Now learn a new model using the paramMapCombined parameters.
# paramMapCombined overrides all parameters set earlier via lr.set* methods.
model2 = lr.fit(training, paramMapCombined)


print("Model 2 fit used these parameters: ")
print_explainParamMap(model2, False)
```

```
Model 2 fit used these parameters:
```

- **aggregationDepth:** 2
- **elasticNetParam:** 0.0
- **family:** auto
- **featuresCol:** features
- **fitIntercept:** True
- **labelCol:** label
- **maxIter:** 10
- **predictionCol:** prediction
- **probabilityCol:** probability
- **rawPredictionCol:** rawPrediction
- **regParam:** 0.01
- **standardization:** True
- **threshold:** 0.5
- **tol:** 1e-06

# Spark MLlib Parameters

```
# Make predictions on test data using the Transformer.transform() method.
# LogisticRegression.transform will only use the 'features' column.
# Note that model2.transform() outputs a "myProbability" column instead of the usual
# 'probability' column since we renamed the lr.probabilityCol parameter previously.
prediction = model2.transform(test)
result = prediction.select("features", "label", "myProbability", "prediction").collect()

for row in result:
    print(
        "features=%s, label=%s -> prob=%s, prediction=%s"
        % (row.features, row.label, row.myProbability, row.prediction)
    )
```

```
features=[-1.0,1.5,1.3], label=1.0 -> prob=[0.057073041710340174,0.9429269582896599], prediction=1.0
features=[3.0,2.0,-0.1], label=0.0 -> prob=[0.9238522311704104,0.07614776882958973], prediction=0.0
features=[0.0,2.2,-1.5], label=1.0 -> prob=[0.10972776114779419,0.8902722388522057], prediction=1.0
```

# Spark MLlib Featurization

- This module contains algorithms for working with features, roughly divided into these groups:
  - Feature extractors: Extracting features from “raw” data
  - Feature transformers: Scaling, converting, or modifying features
  - Feature selectors: Selecting a subset from a larger set of features
  - Locality Sensitive Hashing (LSH): This group of algorithms combines aspects of feature transformation with other algorithms

# Spark MLlib Featurization

## Feature Extractors

- HashingTF
- IDF & IDFModel
- Word2Vec & Word2VecModel
- CountVectorizer & CountVectorizerModel
- FeatureHasher

## Feature Transformers

- Tokenizer
- RegexTokenizer
- StopWordsRemover
- NGram
- Binarizer
- PCA
- PCAModel
- PolynomialExpansion
- DCT
- StringIndexer & StringIndexerModel
- IndexToString
- OneHotEncoder & OneHotEncoderModel
- OneHotEncoderEstimator
- VectorIndexer
- VectorIndexerModel
- Normalizer
- StandardScaler & StandardScalerModel
- MinMaxScaler & MinMaxScalerModel
- MaxAbsScaler & MaxAbsScalerModel
- Bucketizer
- ElementwiseProduct
- SQLTransformer
- VectorAssembler
- VectorSizeHint
- QuantileDiscretizer
- Imputer & ImputerModel

Not available in Python:

- Interaction

## Feature Selectors

- VectorSlicer
- RFormula & RFormulaModel
- ChiSqSelector & ChiSqSelectorModel

## Locality Sensitive Hashing (LSH)

- BucketedRandomProjectionLSH & BucketedRandomProjectionLSHModel
- MinHashLSH & MinHashLSHModel

# Spark MLlib Featurization

```
from pyspark.ml.feature import QuantileDiscretizer

data = [(0, 18.0), (1, 19.0), (2, 8.0), (3, 5.0), (4, 2.2)]
df = spark.createDataFrame(data, ["id", "hour"])

discretizer = QuantileDiscretizer(numBuckets=3, inputCol="hour", outputCol="result")

result = discretizer.fit(df).transform(df)
result.show()
```

```
+---+----+------+
| id|hour|result|
+---+----+------+
|  0|18.0|   2.0|
|  1|19.0|   2.0|
|  2| 8.0|   1.0|
|  3| 5.0|   1.0|
|  4| 2.2|   0.0|
+---+----+------+
```

# Spark MLlib Featurization

```
from pyspark.ml.feature import Word2Vec

# Input data: Each row is a bag of words from a sentence or document.
documentDF = spark.createDataFrame(
    [
        ("Hi I heard about Spark".split(" "),),
        ("I wish Java could use case classes".split(" "),),
        ("Logistic regression models are neat".split(" "),),
    ],
    ["text"],
)

# Learn a mapping from words to Vectors.
word2Vec = Word2Vec(vectorSize=3, minCount=0, inputCol="text", outputCol="result")
model = word2Vec.fit(documentDF)

result = model.transform(documentDF)
for row in result.collect():
    text, vector = row
    print(f"Text: {text} => \nVector: {vector}\n")
```

```
Text: ['Hi', 'I', 'heard', 'about', 'Spark'] =>
Vector: [0.01781592555344105,0.02203895077109337,0.02248857170343399]

Text: ['I', 'wish', 'Java', 'could', 'use', 'case', 'classes'] =>
Vector: [-0.00022016598709991998,-0.0291545108027224,-0.01409794549856867]

Text: ['Logistic', 'regression', 'models', 'are', 'neat'] =>
Vector: [-0.03180776406079531,0.0591656094416976,0.005205358564853668]
```

# Spark MLlib Featurization

```
from pyspark.ml.feature import FeatureHasher

dataset = spark.createDataFrame(
    [
        (2.2, True, "1", "foo"),
        (3.3, False, "2", "bar"),
        (4.4, False, "3", "baz"),
        (5.5, False, "4", "foo"),
    ],
    ["real", "bool", "stringNum", "string"],
)

hasher = FeatureHasher(
    inputCols=["real", "bool", "stringNum", "string"], outputCol="features"
)

featurized = hasher.transform(dataset)
featurized.show(truncate=False)
```

```
+----+-----+---------+------+-----------------------------------------------------------+
|real|bool |stringNum|string|features                                                   |
+----+-----+---------+------+-----------------------------------------------------------+
|2.2 |true |1        |foo   |(262144,[174475,247670,257907,262126],[2.2,1.0,1.0,1.0])|
|3.3 |false|2        |bar   |(262144,[70644,89673,173866,174475],[1.0,1.0,1.0,3.3])  |
|4.4 |false|3        |baz   |(262144,[22406,70644,174475,187923],[1.0,1.0,4.4,1.0])  |
|5.5 |false|4        |foo   |(262144,[70644,101499,174475,257907],[1.0,1.0,5.5,1.0]) |
+----+-----+---------+------+-----------------------------------------------------------+
```

# Spark MLlib Featurization

```
from pyspark.ml.feature import StringIndexer

df = spark.createDataFrame(
    [
        (0, "a"),
        (1, "b"),
        (2, "c"),
        (3, "a"),
        (4, "a"),
        (5, "c"),
        (6, "d"),
        (7, "a"),
        (8, "d"),
    ],
    ["id", "category"],
)

indexer = StringIndexer(inputCol="category", outputCol="categoryIndex")
indexed = indexer.fit(df).transform(df)
indexed.show()
```

```
+---+--------+-------------+
| id|category|categoryIndex|
+---+--------+-------------+
|  0|       a|          0.0|
|  1|       b|          3.0|
|  2|       c|          1.0|
|  3|       a|          0.0|
|  4|       a|          0.0|
|  5|       c|          1.0|
|  6|       d|          2.0|
|  7|       a|          0.0|
|  8|       d|          2.0|
+---+--------+-------------+
```

# Spark MLlib Featurization

```
from pyspark.ml.feature import StringIndexer

df = spark.createDataFrame(
    [
        (0, "a"),
        (1, "b"),
        (2, "c"),
        (3, "a"),
        (4, "a"),
        (5, "c"),
        (6, "d"),
        (7, "a"),
        (8, "d"),
    ],
    ["id", "category"],
)

indexer = StringIndexer(inputCol="category", outputCol="categoryIndex")
indexed = indexer.fit(df).transform(df)
indexed.show()
```

```
+---+--------+-------------+
| id|category|categoryIndex|
+---+--------+-------------+
|  0|       a|          0.0|
|  1|       b|          3.0|
|  2|       c|          1.0|
|  3|       a|          0.0|
|  4|       a|          0.0|
|  5|       c|          1.0|
|  6|       d|          2.0|
|  7|       a|          0.0|
|  8|       d|          2.0|
+---+--------+-------------+
```

# Spark MLlib Featurization

```
from pyspark.ml.feature import VectorIndexer

data = spark.read.format("libsvm").load(
    "/usr/local/spark-2.4.3-bin-hadoop2.7/data/mllib/sample_libsvm_data.txt"
)

indexer = VectorIndexer(inputCol="features", outputCol="indexed", maxCategories=10)
indexerModel = indexer.fit(data)

categoricalFeatures = indexerModel.categoryMaps
print(
    f"Chose {len(categoricalFeatures)} categorical "
    f"features: {[str(k) for k in categoricalFeatures.keys()]}"
)

# Create new column "indexed" with categorical values transformed to indices
indexedData = indexerModel.transform(data)
indexedData.show()
```

```
+-----+--------------------+--------------------+
|label|            features|             indexed|
+-----+--------------------+--------------------+
|  0.0|(692,[127,128,129...|(692,[127,128,129...|
|  1.0|(692,[158,159,160...|(692,[158,159,160...|
|  1.0|(692,[124,125,126...|(692,[124,125,126...|
|  1.0|(692,[152,153,154...|(692,[152,153,154...|
|  1.0|(692,[151,152,153...|(692,[151,152,153...|
|  0.0|(692,[129,130,131...|(692,[129,130,131...|
|  1.0|(692,[158,159,160...|(692,[158,159,160...|
|  1.0|(692,[99,100,101,...|(692,[99,100,101,...|
|  0.0|(692,[154,155,156...|(692,[154,155,156...|
|  0.0|(692,[127,128,129...|(692,[127,128,129...|
|  1.0|(692,[154,155,156...|(692,[154,155,156...|
|  0.0|(692,[153,154,155...|(692,[153,154,155...|
|  0.0|(692,[151,152,153...|(692,[151,152,153...|
|  1.0|(692,[129,130,131...|(692,[129,130,131...|
|  0.0|(692,[154,155,156...|(692,[154,155,156...|
|  1.0|(692,[150,151,152...|(692,[150,151,152...|
|  0.0|(692,[124,125,126...|(692,[124,125,126...|
|  0.0|(692,[152,153,154...|(692,[152,153,154...|
|  1.0|(692,[97,98,99,12...|(692,[97,98,99,12...|
|  1.0|(692,[124,125,126...|(692,[124,125,126...|
+-----+--------------------+--------------------+
```

# Spark MLlib Featurization

```
from pyspark.ml.feature import Bucketizer

splits = [-float("inf"), -0.5, 0.0, 0.5, float("inf")]

data = [
    (-999.9,),
    (-0.5,),
    (-0.3,),
    (-0.4,),
    (-0.2,),
    (0.0,),
    (0.1,),
    (0.2,),
    (0.3,),
    (999.9,),
]
dataFrame = spark.createDataFrame(data, ["features"])

bucketizer = Bucketizer(
    splits=splits, inputCol="features", outputCol="bucketedFeatures"
)

# Transform original data into its bucket index.
bucketedData = bucketizer.transform(dataFrame)

print(f"Bucketizer output with {len(bucketizer.getSplits())-1} buckets")
bucketedData.show()
```

```
Bucketizer output with 4 buckets
+--------+----------------+
|features|bucketedFeatures|
+--------+----------------+
|  -999.9|             0.0|
|    -0.5|             1.0|
|    -0.3|             1.0|
|    -0.4|             1.0|
|    -0.2|             1.0|
|     0.0|             2.0|
|     0.1|             2.0|
|     0.2|             2.0|
|     0.3|             2.0|
|   999.9|             3.0|
+--------+----------------+
```

# Spark MLlib Statistic

- A module used for correlation calculations, hypothesis testing, and column-wise summary statistics for vectors using summarizer
- Correlation:
  - `class pyspark.ml.stat.Correlation`: Supports Pearson's and Spearman's correlation to calculate correlation between two series of data

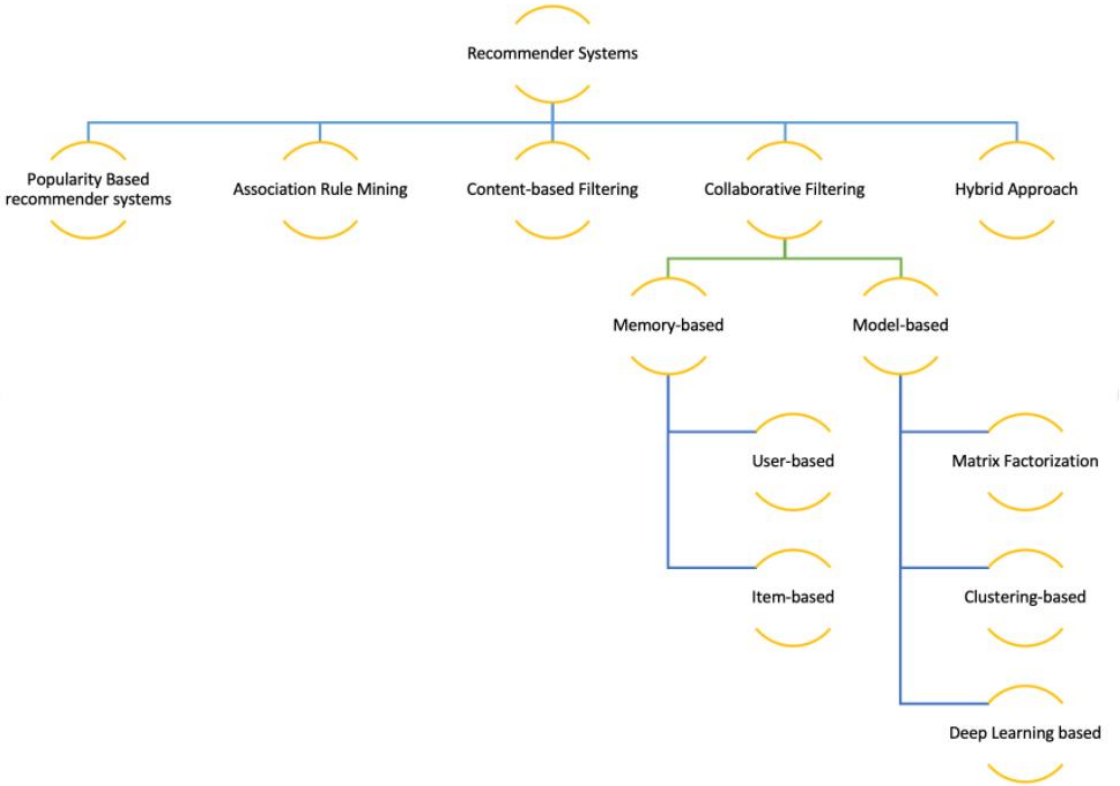
# Spark MLlib Statistic

- Hypothesis testing:
  - `class pyspark.ml.ChiSquareTest`: Conducts Pearson's independence test for every feature against the label
  - `class pyspark.ml.KolmogorovSmirnovTest`: Conduct the two-sided Kolmogorov-Smirnov (KS) test for data sampled from a continuous distribution
- Summarizer:
  - `class pyspark.ml.Summarizer` and `class pyspark.ml.SummaryBuilder`: Column-wise summary statistics for vectors

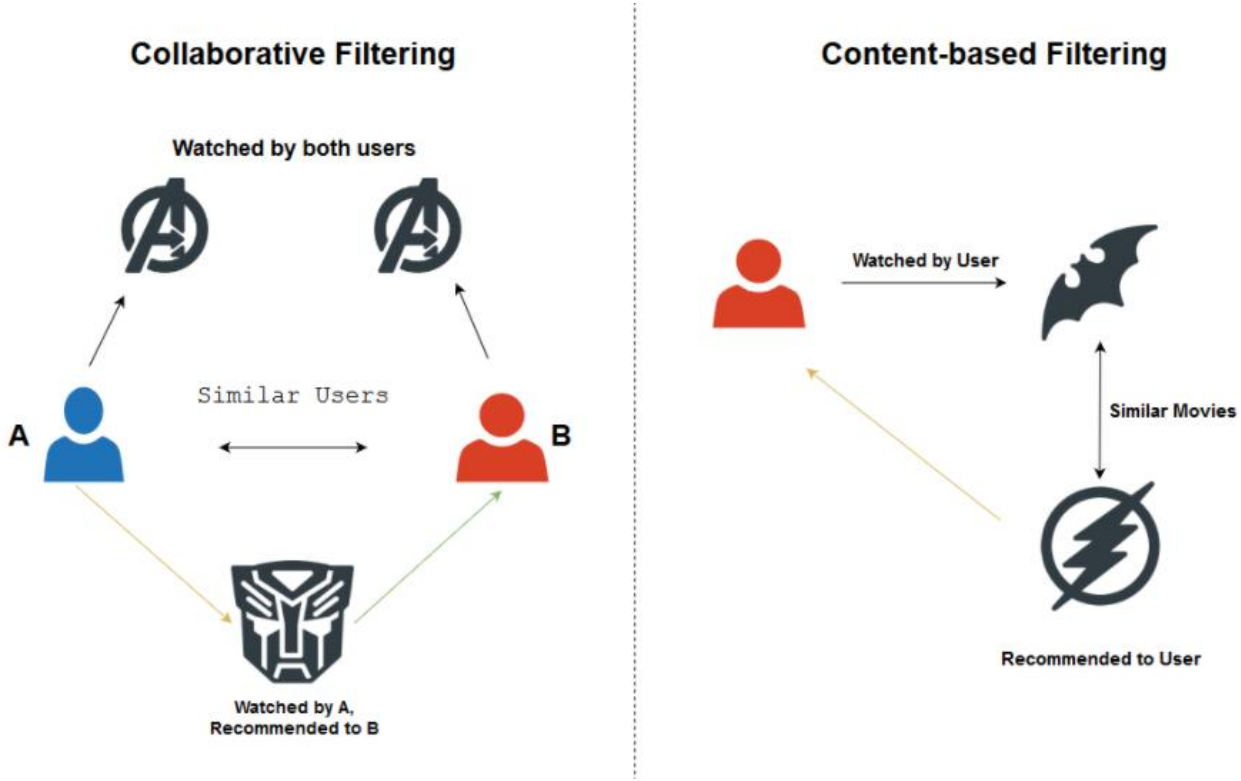
# Spark MLlib Statistic

- Summarizer:
  - `class pyspark.ml.Summarizer` and `class pyspark.ml.SummaryBuilder`: Column-wise summary statistics for vectors

# Spark MLlib Recommendation System



# Spark MLlib Recommendation System



# Spark MLlib Recommendation System

## Content-based recommendation system

- Content-based recommendations based on a user's likes/dislikes
- Advantage:
  - Highly personalized for the user as it learns the user's preferences and tastes

# Spark MLlib Recommendation System

## Content-based recommendation system

- Disadvantage:
  - Determining what a user likes or dislikes about an item is not always obvious
  - Low-quality recommendations might occur as it doesn't consider what the 'neighbors' think
  - Unless the user indicates their preference for a new category (genre of movie in our case), no new categories will ever get recommended to the user

# Spark MLlib Recommendation System

## Collaborative filtering

- We make filtered predictions around the interest of users by collecting preferences or information from many users
- Working under the assumption that user might like to save items popular among their neighbours

# Spark MLlib Recommendation System

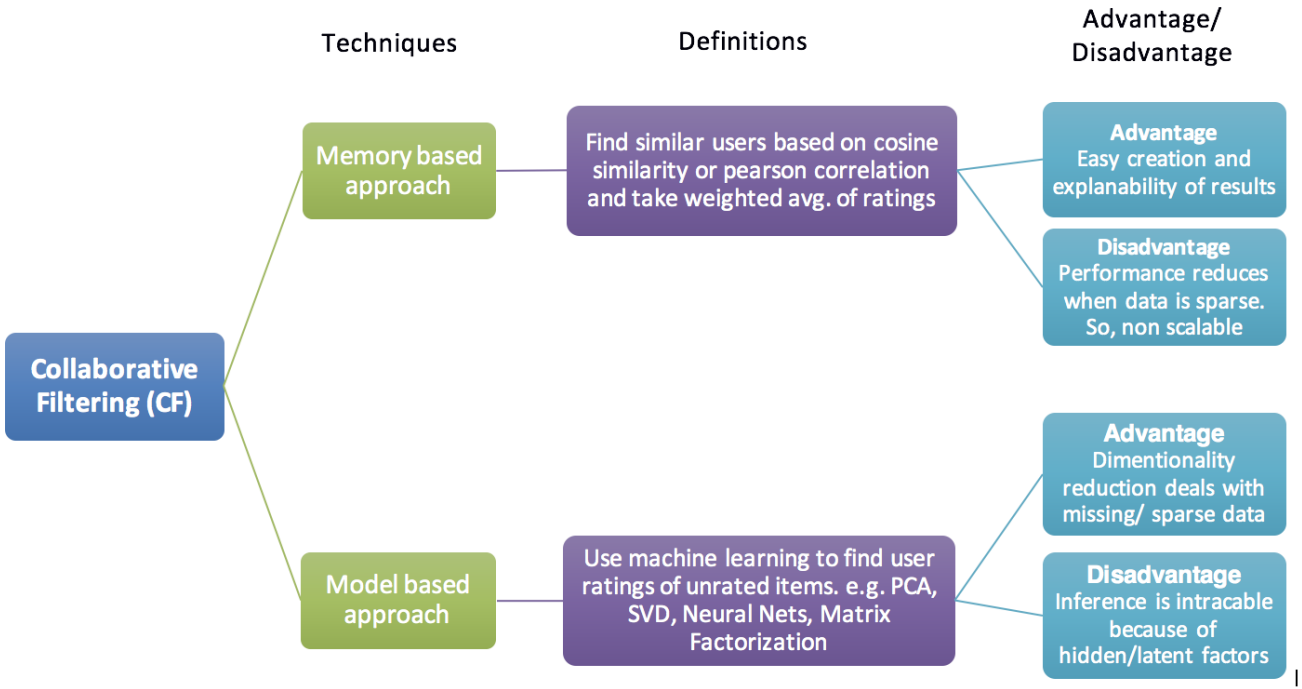
## Collaborative filtering

- Our use case:
  - Building a recommender system based on users rating movies
- The challenge:
  - We are dealing with an extremely sparse matrix; where more than 99% of entries are missing values

| User \ Movie | E | F | G | H |
|---|---|---|---|---|
| A | 1.0 | 4.0 | 3.5 | |
| B | 2.0 | | 5.0 | |
| C | | 3.5 | | 4.0 |
| D | | 2.0 | 4.5 | |

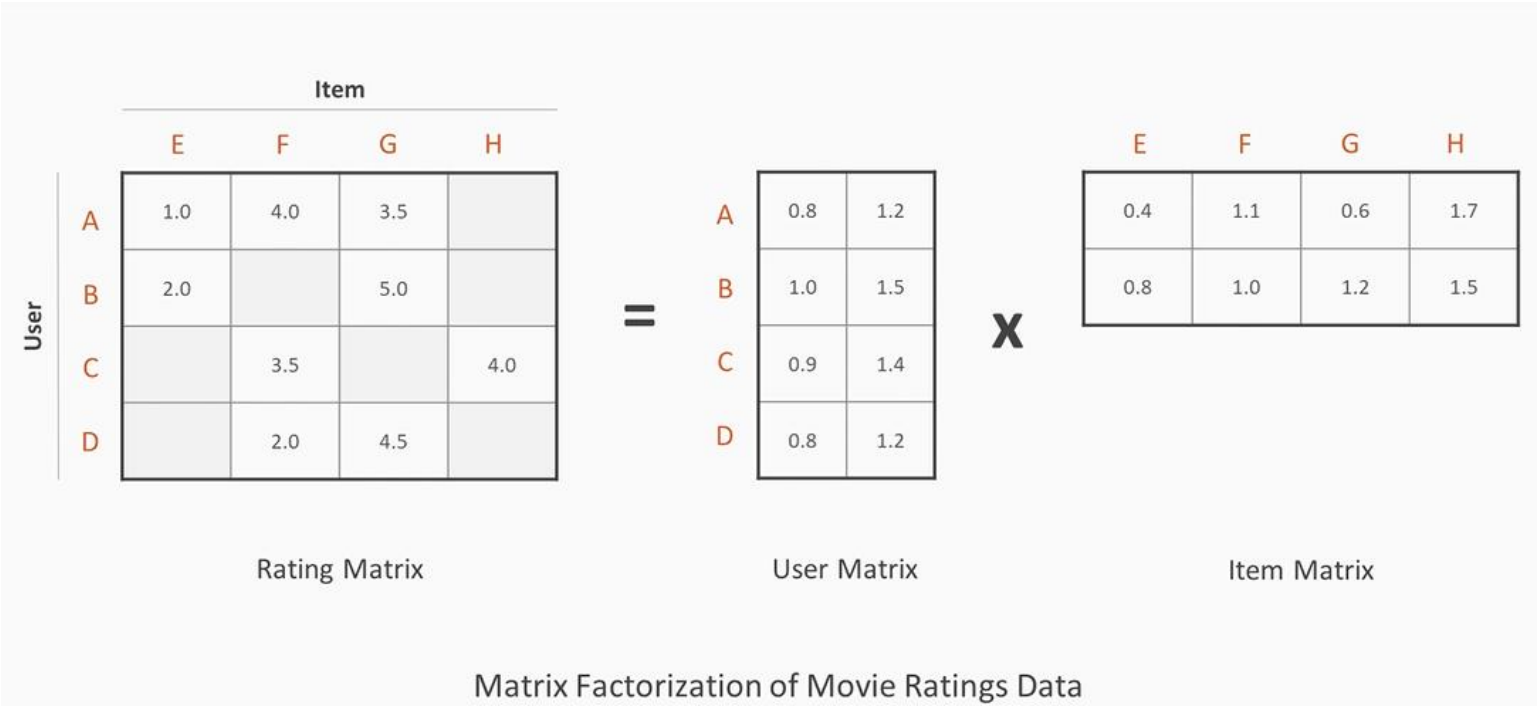
# Spark MLlib Recommendation System

## Collaborative filtering



# Spark MLlib Recommendation System

Collaborative filtering – model based



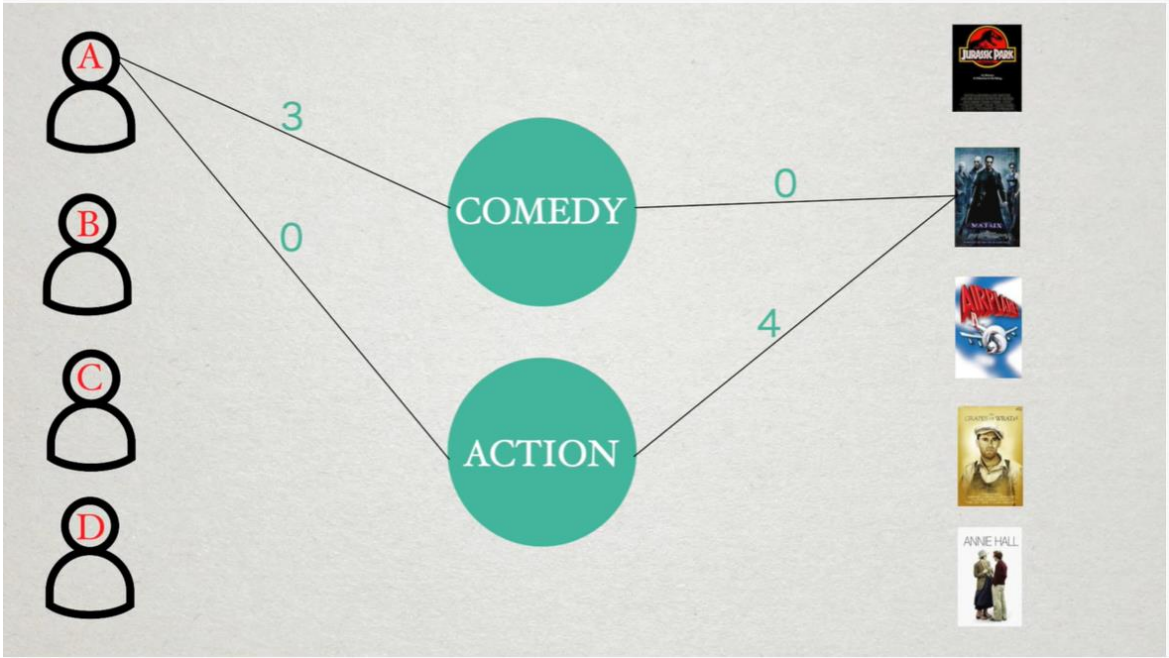

Matrix Factorization of Movie Ratings Data

# Spark MLlib Recommendation System

Collaborative filtering – model based

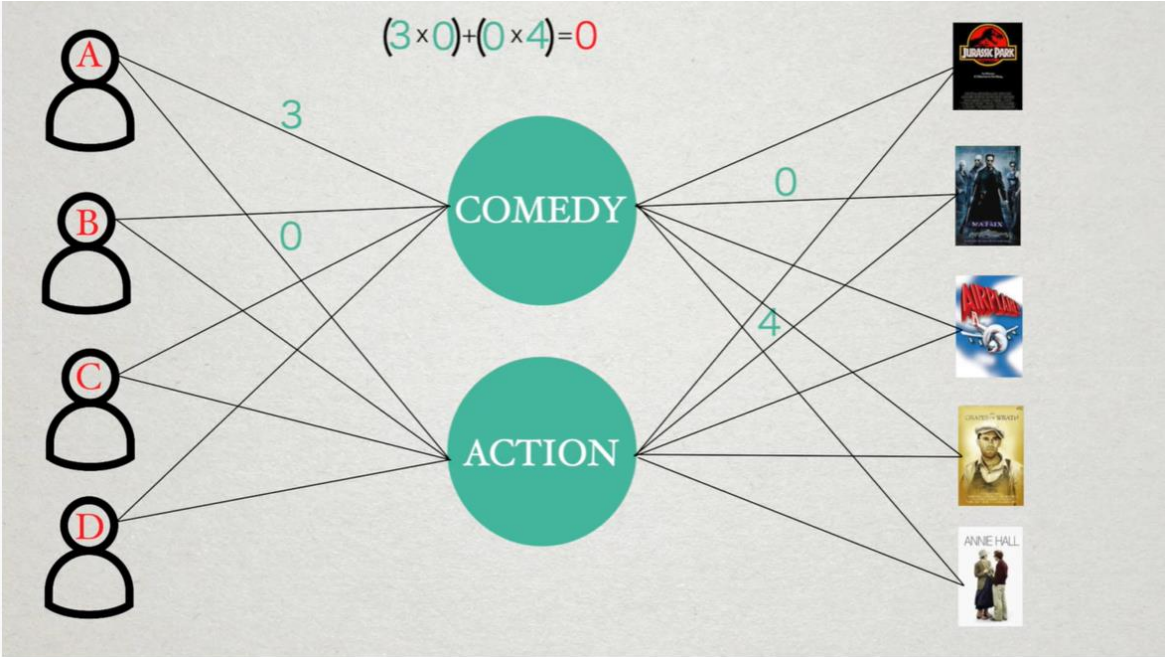
# Spark MLlib Recommendation System

Collaborative filtering – model based



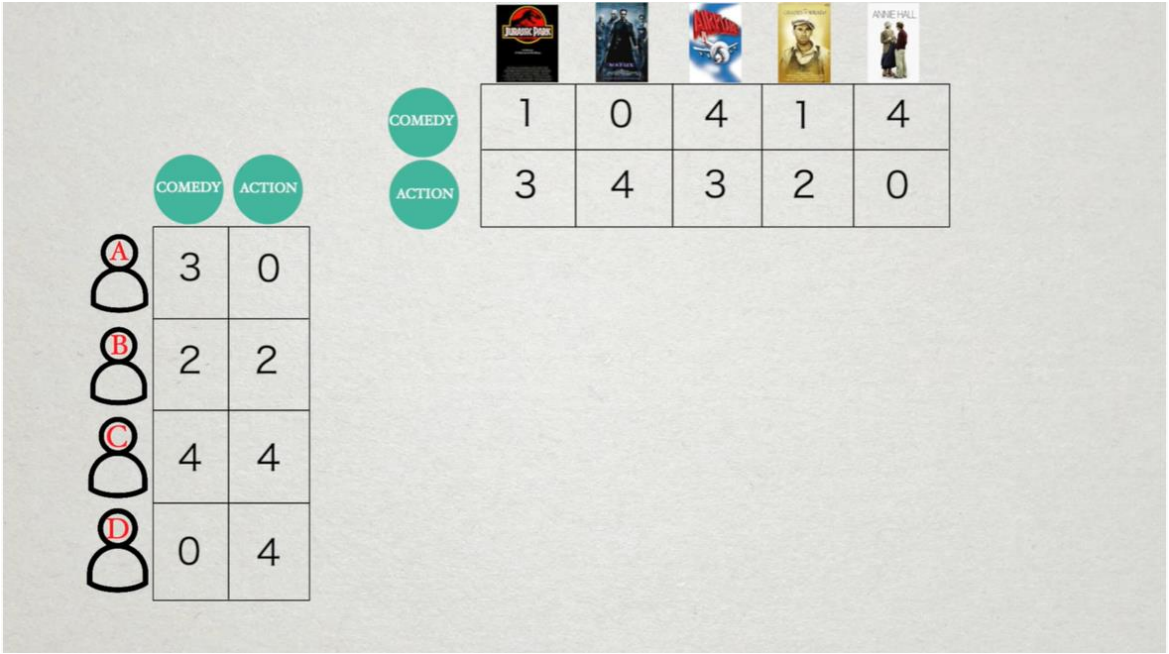
# Spark MLlib Recommendation System

Collaborative filtering – model based



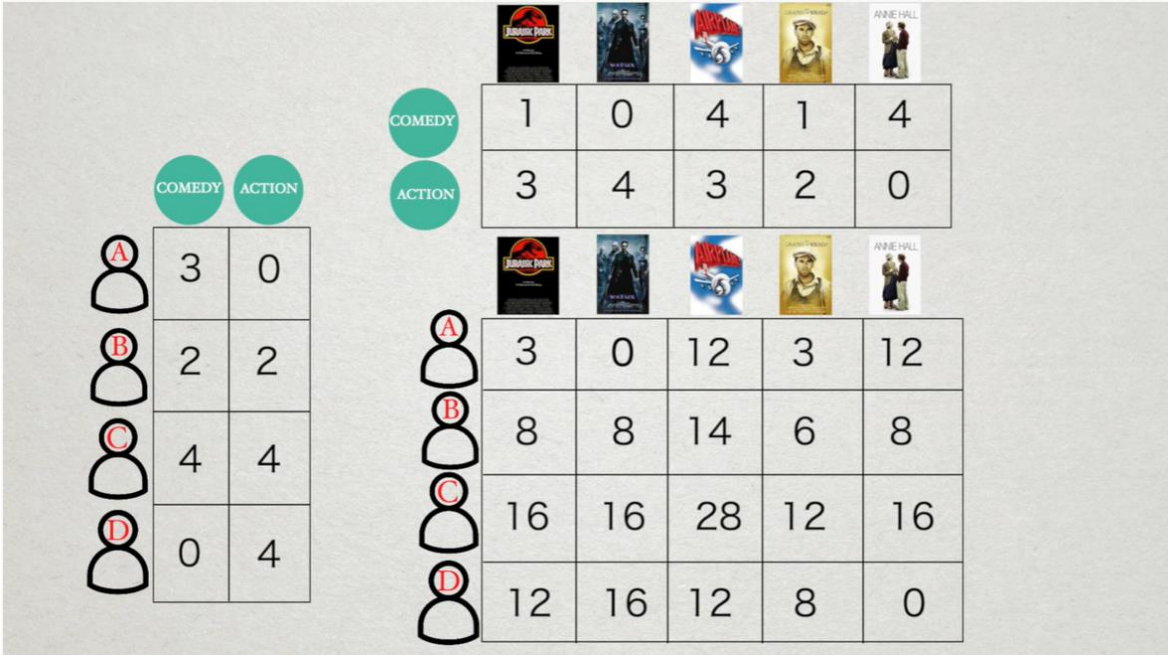
# Spark MLlib Recommendation System

Collaborative filtering – model based



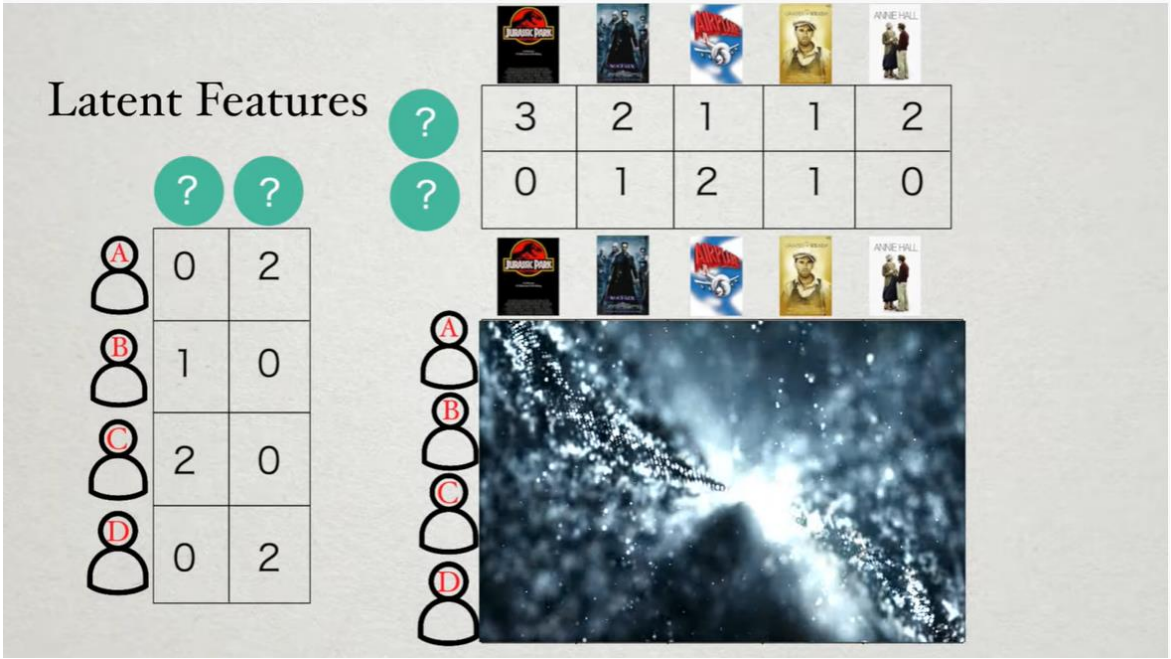
# Spark MLlib Recommendation System

Collaborative filtering – model based



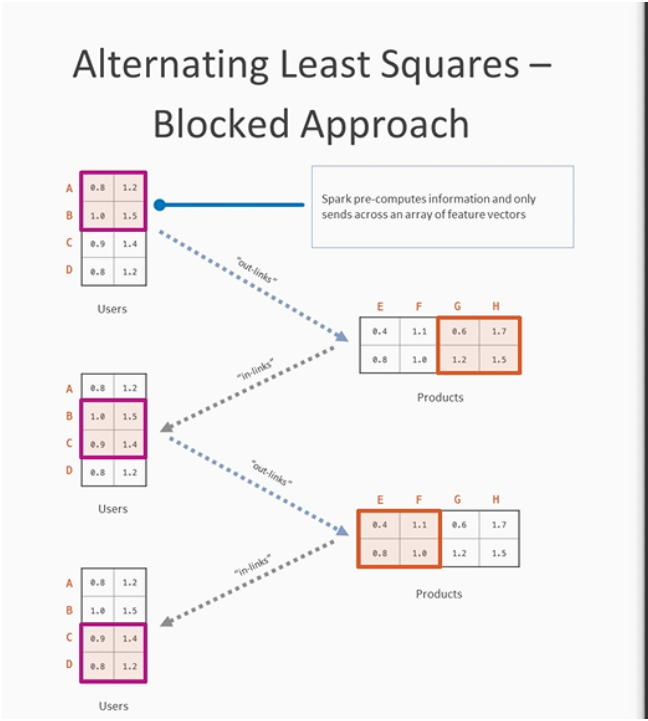
# Spark MLlib Recommendation System

Collaborative filtering – model based



# Spark MLlib Recommendation System

## Collaborative filtering – Altenating Least Squares




### Blocked Implementation

- By pre-computing and grouping factors into blocks:
  - It reduces JVM garbage collection overhead
  - It makes communication between cluster nodes as efficient as possible
  - It is optimized for scale

# Spark MLlib Recommendation System

## Collaborative filtering – pyspark.ml

```
import findspark
findspark.init()
from pyspark.sql import SparkSession
from pyspark.sql import functions as f

spark = SparkSession.builder.appName("movieRecommendationPySpark").getOrCreate()
```

```
ratings = (
    spark.read.csv(
        path="ratings_small.csv",
        sep=",",
        header=True,
        quote='"',
        schema="userId INT, movieId INT, rating DOUBLE, timestamp INT",
    )
    # .withColumn("timestamp", f.to_timestamp(f.from_unixtime("timestamp")))
    .select("userId", "movieId", "rating")
    .cache()
)
```

The ALS class has this signature:

```
class pyspark.ml.recommendation.ALS(
    rank=10,
    maxIter=10,
    regParam=0.1,
    numUserBlocks=10,
    numItemBlocks=10,
    implicitPrefs=False,
    alpha=1.0,
    userCol="user",
    itemCol="item",
    seed=None,
    ratingCol="rating",
    nonnegative=False,
    checkpointInterval=10,
    intermediateStorageLevel="MEMORY_AND_DISK",
    finalStorageLevel="MEMORY_AND_DISK",
    coldStartStrategy="nan",
)
```

# Spark MLlib Recommendation System

Collaborative filtering – pyspark.ml

```
ratings.show(10,False)
```

```
+------+-------+------+
|userId|movieId|rating|
+------+-------+------+
|1     |1      |4.0   |
|1     |3      |4.0   |
|1     |6      |4.0   |
|1     |47     |5.0   |
|1     |50     |5.0   |
|1     |70     |3.0   |
|1     |101    |5.0   |
|1     |110    |4.0   |
|1     |151    |5.0   |
|1     |157    |5.0   |
+------+-------+------+
```

```
ratings.summary().show()
```

```
+-------+------------------+-----------------+------------------+
|summary|            userId|          movieId|            rating|
+-------+------------------+-----------------+------------------+
|  count|            100836|           100836|            100836|
|   mean|326.12756356856676|19435.2957177992| 3.501556983616962|
| stddev| 182.6184914635004|35530.9871987003|1.0425292390606342|
|    min|                 1|                1|               0.5|
|    25%|               177|             1199|               3.0|
|    50%|               325|             2991|               3.5|
|    75%|               477|             8092|               4.0|
|    max|               610|           193609|               5.0|
+-------+------------------+-----------------+------------------+
```

# Spark MLlib Recommendation System

## Collaborative filtering – pyspark.ml

```
from pyspark.ml.recommendation import ALS
from pyspark.ml.evaluation import RegressionEvaluator
```

```
als = ALS(
    userCol="userId",
    itemCol="movieId",
    ratingCol="rating",
)

(training_data, validation_data) = ratings.randomSplit([8.0, 2.0])

evaluator = RegressionEvaluator(
    metricName="rmse", labelCol="rating", predictionCol="prediction"
)

model = als.fit(training_data)
predictions = model.transform(validation_data)
```

```
predictions.show(10, False)
```

```
+------+-------+------+----------+
|userId|movieId|rating|prediction|
+------+-------+------+----------+
|91    |471    |1.0   |3.0327234 |
|409   |471    |3.0   |4.679796  |
|218   |471    |4.0   |3.8422227 |
|387   |471    |3.0   |3.2297335 |
|312   |471    |4.0   |3.8340437 |
|414   |471    |5.0   |3.7297359 |
|541   |471    |3.0   |3.7804155 |
|260   |471    |4.5   |4.155776  |
|609   |833    |3.0   |1.6748803 |
|111   |1088   |3.0   |2.5603733 |
+------+-------+------+----------+
only showing top 10 rows
```

```
rmse = evaluator.evaluate(predictions.na.drop())
```

```
print(rmse)
```

```
0.8786288108243542
```

# Spark MLlib Recommendation System

## Collaborative filtering – pyspark.ml

```
userFactors = model.userFactors
itemFactors = model.itemFactors
userFactors.sort('id').show(5,False)
itemFactors.sort('id').show(5,False)
import numpy as np

user91Feature = model.userFactors.filter(f.col('id')==91).select(f.col('features')).rdd.flatMap(lambda x: x).collect()[0]
item471Feature= model.itemFactors.filter(f.col('id')==471).select(f.col('features')).rdd.flatMap(lambda x: x).collect()[0]

print(user91Feature)
print(item471Feature)
print('Predicted rating of user 91 for movie 471: ' + str(np.dot(user91Feature,item471Feature)))
```

```
+---+----------------------------------------------------------------------------------------------------------------------------------------------------------+
|id |features                                                                                                                                                  |
+---+----------------------------------------------------------------------------------------------------------------------------------------------------------+
|1  |[-1.1396486, 0.046824947, 0.08606943, -0.89530176, 0.381437, -0.2385697, 0.62992984, -1.1538332, 0.22461006, -0.9601014]                                 |
|2  |[-0.57027817, -0.54838854, 0.80637234, -0.97872186, 0.6183535, -0.042785455, 0.7865455, -0.4062991, -0.098041765, -0.47591898]                           |
|3  |[0.24463238, 0.9600048, 0.98998576, -0.9759907, -0.40930456, 0.09272141, 1.3343368, -0.30400163, -0.33486682, -0.39383462]                               |
|4  |[-0.16313235, -0.59958106, -0.5850466, -0.39807796, 0.89300704, -0.3094769, -0.053823743, -1.4678935, 0.88977605, -0.539779]                             |
|5  |[-0.5790872, -0.30325407, -0.04732976, -0.60516685, 0.08863028, 0.18239452, 0.10536681, -1.547137, 0.7643438, -0.72799474]                               |
+---+----------------------------------------------------------------------------------------------------------------------------------------------------------+
only showing top 5 rows

+---+----------------------------------------------------------------------------------------------------------------------------------------------------------+
|id |features                                                                                                                                                  |
+---+----------------------------------------------------------------------------------------------------------------------------------------------------------+
|1  |[-0.94216657, -0.7023109, 0.009909666, -0.79786617, 0.6519753, -0.20649733, 0.8735492, -1.1982708, 0.55822855, -0.54331917]                              |
|2  |[-0.9681279, -0.47008696, 0.17003863, -0.4119318, 0.38166243, 0.20443514, 1.0768436, -1.1046029, 0.116564624, -0.63136214]                               |
|3  |[-0.83121145, -0.31397244, 0.55451936, -0.7868329, 0.79168385, 0.24812174, 0.16610332, -1.0381806, -0.52568763, -0.93329155]                             |
|4  |[-0.600918, -0.61637694, 0.5325797, -0.23614328, 0.7943548, 0.12294494, 0.17635046, -0.6797515, -0.04617397, -0.41124898]                                |
|5  |[-0.5220082, -0.17868853, 0.27958348, -0.8080285, 0.7605523, 0.37655824, 0.3978313, -1.0887363, -0.10682703, -0.62507665]                                |
+---+----------------------------------------------------------------------------------------------------------------------------------------------------------+
only showing top 5 rows
```

```
predictions.show(10, False)
```

```
+------+-------+------+----------+
|userId|movieId|rating|prediction|
+------+-------+------+----------+
|91    |471    |1.0   |3.0327234 |
```

```
[-0.3994467258453369, -0.274524986743927, 0.34672558307647705, -0.8329096436500549, -0.2576223611831665, -0.07359780371189117, 1.2851195335388184, -0.4822205603122711, 0.11742987483739853, -0.8312146663665771]
[-0.618400514125824, -0.36738458275794983, -0.43478521704673767, -0.10723729431629181, 0.4398992359638214, 0.1055564135313034, 0.754115641117096, -0.8633681535720825, 0.513541579246521, -1.7102802991867065]
Predicted rating of user 91 for movie 471: 3.0327233343608766
```

# Spark MLlib Recommendation System

## Collaborative filtering – pyspark.ml

```
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder

parameter_grid = (
    ParamGridBuilder()
    .addGrid(als.rank, [5, 10])
    .addGrid(als.maxIter, [20])
    .addGrid(als.regParam, [0.05, 0.1])
    .build()
)
```

```
type(parameter_grid)
```

```
<class 'list'>
```

```
from pprint import pprint

pprint(parameter_grid)
```

```
[{Param(parent='ALS_82b6ebc09a93', name='maxIter', doc='max number of iterations (>= 0).'): 20,
  Param(parent='ALS_82b6ebc09a93', name='rank', doc='rank of the factorization'): 5,
  Param(parent='ALS_82b6ebc09a93', name='regParam', doc='regularization parameter (>= 0).'): 0.05},
 {Param(parent='ALS_82b6ebc09a93', name='maxIter', doc='max number of iterations (>= 0).'): 20,
  Param(parent='ALS_82b6ebc09a93', name='rank', doc='rank of the factorization'): 5,
  Param(parent='ALS_82b6ebc09a93', name='regParam', doc='regularization parameter (>= 0).'): 0.1},
 {Param(parent='ALS_82b6ebc09a93', name='maxIter', doc='max number of iterations (>= 0).'): 20,
  Param(parent='ALS_82b6ebc09a93', name='rank', doc='rank of the factorization'): 10,
  Param(parent='ALS_82b6ebc09a93', name='regParam', doc='regularization parameter (>= 0).'): 0.05},
 {Param(parent='ALS_82b6ebc09a93', name='maxIter', doc='max number of iterations (>= 0).'): 20,
  Param(parent='ALS_82b6ebc09a93', name='rank', doc='rank of the factorization'): 10,
  Param(parent='ALS_82b6ebc09a93', name='regParam', doc='regularization parameter (>= 0).'): 0.1}]
```

# Spark MLlib Recommendation System

## Collaborative filtering – pyspark.ml

```
rmse = evaluator.evaluate(predictions.na.drop())
print(rmse)
```

```
0.9151240171424524
```

```
model = crossval_model.bestModel
```

```
model.userFactors.show(5,False)
```

```
+---+-----------------------------------------------------------+
|id |features                                                   |
+---+-----------------------------------------------------------+
|10 |[1.2530712, -0.2791498, -1.7960297, 0.48610744, 0.71415794]  |
|20 |[-0.015548448, -1.44367, -1.3121722, 0.062264383, -0.9449332]|
|30 |[0.65103686, -0.57082814, -1.7171123, 0.38074148, -0.8411453]|
|40 |[1.2579815, -1.3042885, -0.27397826, 0.42265755, -1.0877987] |
|50 |[0.560652, -1.1323379, -0.21379222, -0.14114618, -0.7281288] |
+---+-----------------------------------------------------------+
only showing top 5 rows
```

# Spark MLlib Recommendation System

## Collaborative filtering – pyspark.ml

```
predictions.toPandas()
```

| | userId | movieId | rating | prediction |
|---|---|---|---|---|
| **0** | 91 | 471 | 1.0 | 2.535147 |
| **1** | 409 | 471 | 3.0 | 4.819777 |
| **2** | 218 | 471 | 4.0 | 3.222253 |
| **3** | 387 | 471 | 3.0 | 3.149618 |
| **4** | 312 | 471 | 4.0 | 4.164871 |
| **...** | ... | ... | ... | ... |
| **19500** | 204 | 79008 | 3.5 | 4.002772 |
| **19501** | 522 | 79008 | 4.5 | 4.013596 |
| **19502** | 177 | 84374 | 3.5 | 2.969700 |
| **19503** | 432 | 84374 | 4.0 | 3.434984 |
| **19504** | 249 | 84374 | 3.5 | 3.237358 |

19505 rows × 4 columns

# Spark MLlib Recommendation System

## Collaborative filtering – pyspark.ml

### Top 5 Movies for each user, for all users

```
userID = 91

rec_all_users = model.recommendForAllUsers(5).cache()
rec_all_users.show(5,False)
rec_all_users.printSchema()
```

```
+------+----------------------------------------------------------------------------------------------------+
|userId|recommendations                                                                                     |
+------+----------------------------------------------------------------------------------------------------+
|471   |[[158783, 5.590863], [6818, 5.5682793], [8477, 5.514393], [68945, 5.241924], [96004, 5.241924]]     |
|463   |[[86377, 5.835226], [299, 5.645237], [26865, 5.3744984], [187593, 5.302268], [96004, 5.238351]]     |
|496   |[[70946, 7.0032015], [26865, 6.577416], [6818, 6.3157024], [3819, 6.225935], [306, 6.0877237]]      |
|148   |[[4256, 5.8173766], [112804, 5.343978], [49932, 5.2747917], [74946, 5.143712], [4442, 5.1404366]]   |
|540   |[[299, 6.086707], [87234, 5.9714355], [158872, 5.9553595], [68945, 5.949146], [96004, 5.949146]]    |
+------+----------------------------------------------------------------------------------------------------+
only showing top 5 rows

root
 |-- userId: integer (nullable = false)
 |-- recommendations: array (nullable = true)
 |    |-- element: struct (containsNull = true)
 |    |    |-- movieId: integer (nullable = true)
 |    |    |-- rating: float (nullable = true)
```

# Spark MLlib Recommendation System

Collaborative filtering – pyspark.ml

```
recommendations_for_user91.show(5,False)
```

```
+-------+-----------------------------------+----+
|movieId|title                              |year|
+-------+-----------------------------------+----+
|86377  |Louis C.K.: Shameless              |2007|
|70946  |Troll 2                            |1990|
|26865  |Fist of Legend (Jing wu ying xiong)|1994|
|299    |Priest                             |1994|
|3819   |Tampopo                            |1985|
+-------+-----------------------------------+----+
```

# Run PySpark on Kaggle Notebook

- Make sure Internet is turned On on Kaggle Notebook
- Run !pip install pyspark

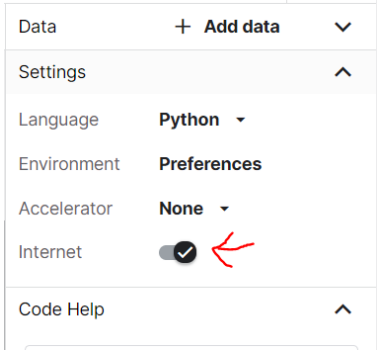
```
!pip install pyspark
import pyspark
from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("sparkOnKaggle").getOrCreate()
```

```
Collecting pyspark
  Downloading pyspark-3.1.2.tar.gz (212.4 MB)
     |████████████████████████████████| 212.4 MB 57 kB/s s eta 0:00:01   |█                               | 6.2 MB 10.2 MB/s eta 0:00:21     |██████
██████████████▏          | 186.4 MB 44.5 MB/s eta 0:00:01
Collecting py4j==0.10.9
  Downloading py4j-0.10.9-py2.py3-none-any.whl (198 kB)
     |████████████████████████████████| 198 kB 39.6 MB/s eta 0:00:01
Building wheels for collected packages: pyspark
  Building wheel for pyspark (setup.py) ... done
  Created wheel for pyspark: filename=pyspark-3.1.2-py2.py3-none-any.whl size=212880768 sha256=0b1eec842dd65803d8c4a025067a20de79057b3a1fc32a1ab74a78
5eab565b25
  Stored in directory: /root/.cache/pip/wheels/a5/0a/c1/9561f6fecb759579a7d863dcd846daaa95f598744e71b02c77
Successfully built pyspark
Installing collected packages: py4j, pyspark
Successfully installed py4j-0.10.9 pyspark-3.1.2
```

+ Code + Markdown

```
[8]: spark.version
```

```
[8]: '3.1.2'
```



# Q & A

## Cảm ơn đã theo dõi

Chúng tôi hy vọng cùng nhau đi đến thành công.