Apache Spark in Data Science

Real World Applications

Kirill Pavlov

Data Science Team, Asia Miles Limited

May 12, 2016

亞洲萬里通
ASIA MILES

Disclaimer

The opinions expressed in this presentation and on the following slides are solely those of the presenter and not necessarily those of Asia Miles Limited.

Quick Questionnaire

- How many people have attended previous Spark talks?

Quick Questionnaire

- How many people have attended previous Spark talks?
- How many people are currently working with Spark?

Quick Questionnaire

- How many people have attended previous Spark talks?
- How many people are currently working with Spark?
- How many people are familiar with Scala?

About the Presenter

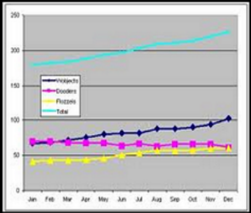
- MS degree from Moscow Institute of Physics and Technology with distinction.
- 8+ years of data science and machine learning experience.
- Worked in Yandex (Russian Google) on search and on-line contextual ads ranking algorithms.
- Developed and consulted start-ups in digital marketing, healthcare, real estate and home automation areas.
- Open-source contributor, full-stack engineer and data mining evangelist.
- Now data scientist in Asia Miles.

Data Scientist

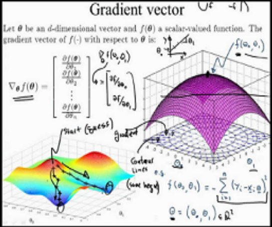
What my friends think I do

What my mom thinks I do

What society thinks I do



What my boss thinks I do

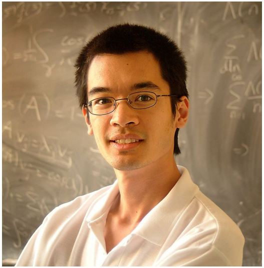


What I think I do

What I actually do

Data Scientist skills

Software Engineering



Data Mining

Presentation

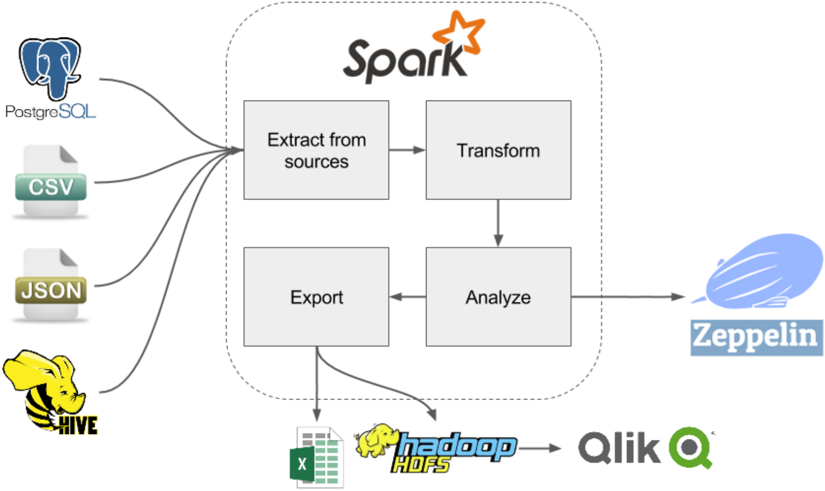
Table of content

Table of content

Environment

Spark Scala Zeppelin

Spark connection with other tools



Software Engineering

- External libraries could be included to spark-submit and Zeppelin from mvn repository.

Software Engineering

- External libraries could be included to spark-submit and Zeppelin from mvn repository.
- Internal libraries are continuously tested and packaged to JARs.

Software Engineering

- External libraries could be included to spark-submit and Zeppelin from mvn repository.
- Internal libraries are continuously tested and packaged to JARs.
- Frequent tasks are executed with schedulers, such as Airflow.

External JARs usage

Example of spark-submit with dependencies:

```
spark-submit --master yarn \
  --jars spark-csv_2.11.jar,dependency.jar \
  --class com.example.ComputeSomething \
  mypackage.jar
```

Zeppelin example:

```
%dep
z.load("/path/to/spark-csv_2.11.jar")
z.load("/path/to/dependency.jar")
```

- Data generation and visualization are two **independent** components.

Visualization

- Data generation and visualization are two **independent** components.
- Neither of Spark/Python/R produces easy customizable charts.

Visualization

- Data generation and visualization are two **independent** components.
- Neither of Spark/Python/R produces easy customizable charts.
- Solution: BI dashboards (QlikView, Tableau) and Excel. Sometimes JavaScript (d3js, MapBox) works better.

Visualization

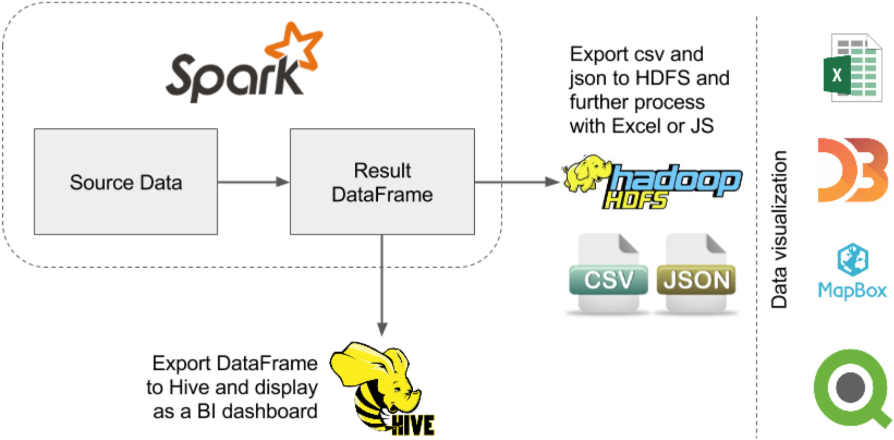


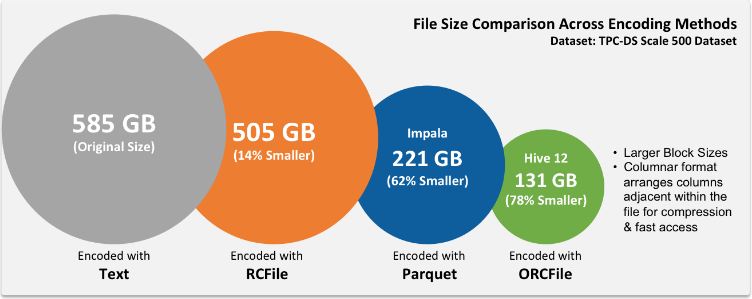
Table of content

Data manipulation options

- Spark.
- MapReduce/Hive.
- RDBMS: Postgres/MySQL/etc.
- Pandas/R DataFrames.

Persistent storage options

- Hive: out of the box fast access to the data.
- HDFS ORC: data could be partitioned.
- HDFS CSV: easy to export to local file system.



Code examples

User-defined functions: data

```
val colors = sc.parallelize(Array(
  ("FFFFFF"),
  ("000000"),
  ("123456")
)).toDF("color")

colors.show()

+------+
| color|
+------+
|FFFFFF|
|000000|
|123456|
+------+
```

User-defined functions: example

```scala
def hex2rgb(s: String): (Int, Int, Int) = {
    val hex = Integer.parseInt(s, 16)
    val r = (hex & 0xFF0000) >> 16
    val g = (hex & 0xFF00) >> 8
    val b = (hex & 0xFF)
    return (r, g, b)
}

val hex2rgbUDF = sqlContext.udf
  .register("hex2rgb", (s: String) => hex2rgb(s))

colors.withColumn("rgb", hex2rgbUDF($"color"))
  .show()
```

```
+------+-------------+
| color|          rgb|
+------+-------------+
|FFFFFF|[255,255,255]|
|000000|      [0,0,0]|
|123456|   [18,52,86]|
+------+-------------+
```

Window functions: motivation

- Operate on a frame[1] while still returning a single value for every input row. Many to one is aggregation, one to one is UDF.
- Calculating a moving average, cumulative sum, accessing previous/next values of a row.
- Ranking and calculating percentiles.

[1] Frame (Window) – group of rows associated with every input row.

Window functions: data

```
val products = sc.parallelize(Array(
  ("steak", "1990-01-01", "2000-01-01", 150),
  ("steak", "2000-01-02", "2010-01-01", 180),
  ("steak", "2010-01-02", "2020-01-01", 200),
  ("fish", "1990-01-01", "2020-01-01", 100)
)).toDF("name", "startDate", "endDate", "price")

products.show()
```

```
+-----+----------+----------+-----+
| name| startDate|   endDate|price|
+-----+----------+----------+-----+
|steak|1990-01-01|2000-01-01|  150|
|steak|2000-01-02|2010-01-01|  180|
|steak|2010-01-02|2020-01-01|  200|
| fish|1990-01-01|2020-01-01|  100|
+-----+----------+----------+-----+
```

Window functions: example

```
import org.apache.spark.sql.expressions.Window

val win1 = Window.partitionBy("name").orderBy("endDate")
val win2 = Window.partitionBy("name").orderBy("endDate")
  .rowsBetween(Long.MinValue, 0)

products
  .withColumn("monthsFromLastUpdate",
    months_between($"endDate", lag("endDate", 1).over(win1)))
  .withColumn("origPriceUplift", $"price" - first($"price").over(win2))
  .show()
```

```
+-----+----------+----------+-----+--------------------+---------------+
| name| startDate|   endDate|price|monthsFromLastUpdate|origPriceUplift|
+-----+----------+----------+-----+--------------------+---------------+
| fish|1990-01-01|2020-01-01|  100|                null|              0|
|steak|1990-01-01|2000-01-01|  150|                null|              0|
|steak|2000-01-02|2010-01-01|  180|               120.0|             30|
|steak|2010-01-02|2020-01-01|  200|               120.0|             50|
+-----+----------+----------+-----+--------------------+---------------+
```

Non-equi joins: data

```
val orders = sc.parallelize(Array(
  ("1995-01-01", "steak"),
  ("2000-01-01", "fish"),
  ("2005-01-01", "steak"),
  ("2010-01-01", "fish"),
  ("2015-01-01", "steak")
)).toDF("date", "product")

orders.show()
```

```
+----------+-------+
|      date|product|
+----------+-------+
|1995-01-01|  steak|
|2000-01-01|   fish|
|2005-01-01|  steak|
|2010-01-01|   fish|
|2015-01-01|  steak|
+----------+-------+
```

Non-equi joins: example

```
orders
  .join(products, $"product" === $"name"
    && $"date" >= $"startDate"
    && $"date" <= $"endDate")
  .show()
```

```
+----------+-------+-----+----------+----------+-----+
|      date|product| name| startDate|   endDate|price|
+----------+-------+-----+----------+----------+-----+
|2000-01-01|   fish| fish|1990-01-01|2020-01-01|  100|
|2010-01-01|   fish| fish|1990-01-01|2020-01-01|  100|
|1995-01-01|  steak|steak|1990-01-01|2000-01-01|  150|
|2005-01-01|  steak|steak|2000-01-02|2010-01-01|  180|
|2015-01-01|  steak|steak|2010-01-02|2020-01-01|  200|
+----------+-------+-----+----------+----------+-----+
```

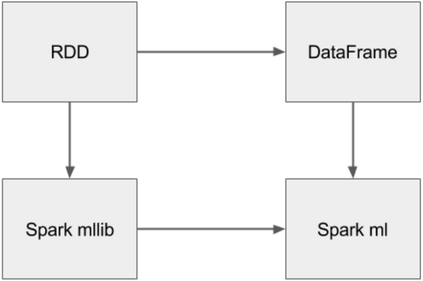
More examples: here.

Table of content

Use spark.ml instead of spark.mllib

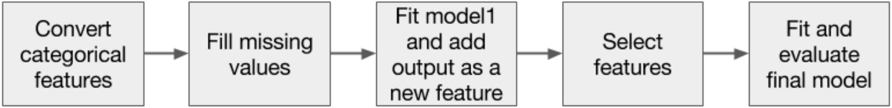
- spark.mllib contains the original API built on top of RDDs.
- spark.ml provides higher-level API built on top of DataFrames for constructing ML pipelines.

"Using spark.ml is recommended because with DataFrames the API is more versatile and flexible. . . "



Pipelines

- Main concept of spark.ml.
- Sequence of stages (estimators, transformers, models).
- It is easy to maintain a Pipeline.
- Used for feature engineering, cross-validation and model fitting.



Estimators and transformers

- Estimators use existing data to estimate parameters, for example categorical features. Learning algorithms are also estimators. Output of estimator is a transformer.
- Adds new colums to a dataframe, such as new (engineered) feature or model output (estimation, probability, etc.).

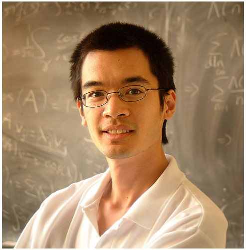
Estimators and transformers: example

```scala
// Define indexers and encoders
val fieldsToIndex = Array("gender", "language")
val indexers = fieldsToIndex.map(f => new StringIndexer()
  .setInputCol(f).setOutputCol(f + "_index"))

val fieldsToEncode = Array("gender", "language")
val oneHotEncoders = fieldsToEncode.map(f => new OneHotEncoder()
  .setInputCol(f + "_index").setOutputCol(f + "_flags"))

val featureAssembler = new VectorAssembler()
  .setInputCols(Array("gender_flags", "language_flags"))
  .setOutputCol("features")

// Combine stages into pipeline
val pipeline = new Pipeline()
  .setStages(indexers ++ oneHotEncoders :+ featureAssembler)
```

Estimators and transformers: example

```
val data = sc.parallelize(Array(
  ("M", "EN", 1.0),
  ("M", "ES", 0.0),
  ("F", "EN", 1.0),
  ("F", "ZH", 0.1)
)).toDF("gender", "language", "label")

pipeline.fit(data).transform(data)
  .drop("gender_flags").drop("language_flags")
  .show()

+------+--------+-----+------------+--------------+-------------+
|gender|language|label|gender_index|language_index|     features|
+------+--------+-----+------------+--------------+-------------+
|     M|      EN|  1.0|         0.0|           0.0|[1.0,1.0,0.0]|
|     M|      ES|  0.0|         0.0|           1.0|[1.0,0.0,1.0]|
|     F|      EN|  1.0|         1.0|           0.0|[0.0,1.0,0.0]|
|     F|      ZH|  0.1|         1.0|           2.0|    (3,[],[])|
+------+--------+-----+------------+--------------+-------------+
```

Feature engineering



- Feature creation is up to analyst. Spark is convenient from software engineering point of view, but not as practical as Python/R with in-memory dataframes.
- Once features are defined, spark creates sparse vectors based on them. At this point only spark algorithms could be used.

Cross-validation and grid search example

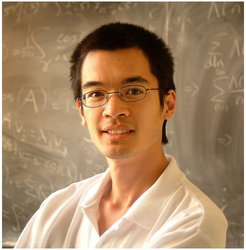
```
val Array(training, test) = data.randomSplit(Array(0.9, 0.1), seed = 1234)

val featureAssembler = new VectorAssembler()
  .setInputCols(
    Array("sepal_length", "sepal_width", "petal_length", "petal_width")
  ).setOutputCol("features")
val lr = new LogisticRegression().setMaxIter(10)
val fullPipeline = new Pipeline().setStages(Array(featureAssembler, lr))

val paramGrid = new ParamGridBuilder()
  .addGrid(lr.regParam, Array(0.1, 0.01))
  .build()

val cv = new CrossValidator()
  .setEstimator(fullPipeline)
  .setEvaluator(new BinaryClassificationEvaluator)
  .setEstimatorParamMaps(paramGrid)
  .setNumFolds(5)

val cvModel = cv.fit(training)
```

Cross-validation and grid search example

```
cvModel.transform(test)
    .select("features", "label", "prediction", "probability")
    .show()
```

```
+-----------------+-----+----------+--------------------+
|         features|label|prediction|         probability|
+-----------------+-----+----------+--------------------+
|[5.9,3.0,4.2,1.5]|  0.0|       0.0|[0.80461154054846...|
|[5.5,2.4,3.8,1.1]|  0.0|       0.0|[0.92788428007362...|
|[5.8,2.7,3.9,1.2]|  0.0|       0.0|[0.91612929982336...|
|[6.0,2.7,5.1,1.6]|  0.0|       1.0|[0.42013327746284...|
|[6.0,2.9,4.5,1.5]|  0.0|       0.0|[0.71645104521227...|
|[6.7,3.0,5.0,1.7]|  0.0|       1.0|[0.46088659238049...|
|[6.4,2.7,5.3,1.9]|  1.0|       1.0|[0.21021735245935...|
|[7.6,3.0,6.6,2.1]|  1.0|       1.0|[0.03313839499693...|
|[6.4,3.2,5.3,2.3]|  1.0|       1.0|[0.11969201314865...|
|[6.0,3.0,4.8,1.8]|  1.0|       1.0|[0.45195167667592...|
|[7.7,3.0,6.1,2.3]|  1.0|       1.0|[0.04122882687014...|
+-----------------+-----+----------+--------------------+
```

Available algorithms



- Spark is processing engine on top of **distributed** system. Not every algorithm is scalable, so spark.ml does not have them.
- Current spark.ml algorithms: logistic regression, decision tree, neural network.
- Interface is convenient, but speed and quality are not as good as xgboost.

Workflow

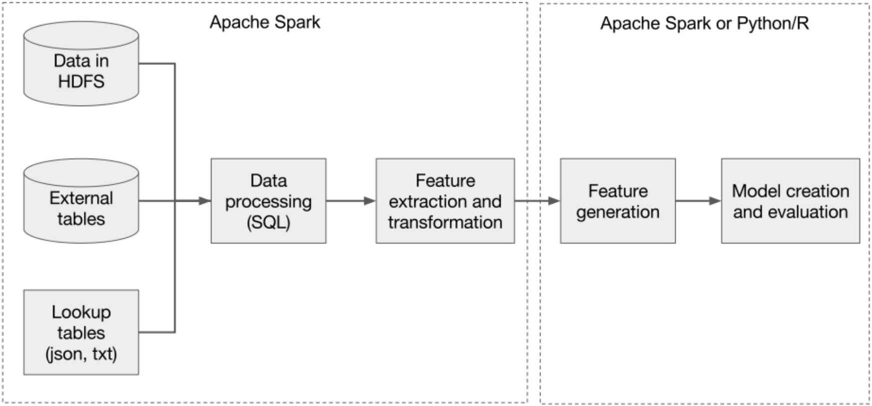

Apache Spark
Data in HDFS
External tables
Lookup tables (json, txt)
Data processing (SQL)
Feature extraction and transformation
Apache Spark or Python/R
Feature generation
Model creation and evaluation

Table of content

Conclusion

- Spark is a good tool, but not for every task.
- Data manipulation is easy and fast with Spark.
- It's machine learning library is well designed, but accuracy is not as good as for in-memory solutions (xgboost/deep learning).

Quiz

1. Does spark have visualization module?
   A) Yes B) No

Quiz

1. Does spark have visualization module?
   A) Yes B) No

Quiz

1. Does spark have visualization module?
   A) Yes B) No
2. Main concept of spark.ml is:
   A) Pipeline B) Estimator C) Transformer D) Model

Quiz

1. Does spark have visualization module?
   A) Yes B) No
2. Main concept of spark.ml is:
   A) Pipeline B) Estimator C) Transformer D) Model

Thank you!

Kirill Pavlov, Data Science Team, Asia Miles Limited
kirill_pavlov@asiamiles.com