Bài 5 - Model Pipeline - SparkSQL

15 Jul 2019 - phamdinhkhanh

Menu

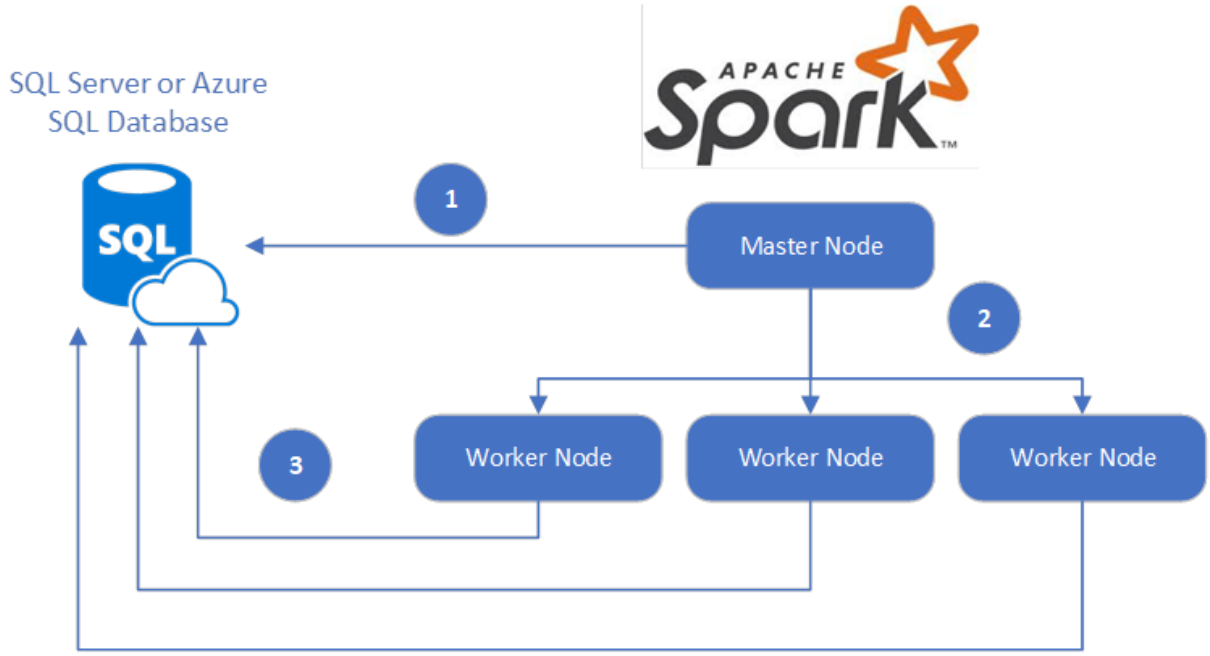
1. Giới thiệu về SparkSQL

SparkSQL là một module của Apache spark cho phép thực hiện các biến đổi với dữ liệu có cấu trúc và các tính toán dựa trên cụm xử lý. Spark cho phép tính toán data phân tán bằng việc chia dữ liệu thành nhiều phần nhỏ và thực hiện tính toán song song từng phần trên nhiều node server khác nhau. Chính vì thế tốc độ xử lý của spark rất nhanh và phù hợp với những dữ liệu kích thước lớn.

Một vấn đề thường xuyên gặp phải đối với các quá trình xử lý song song là dữ liệu sẽ dễ dàng bị gặp lỗi do yêu cầu sự đồng bộ từ các node xử lý. Nhưng Spark có khả năng dung hòa lỗi thông qua cơ chế khôi phục từ checkpoint và ghi log.

Mỗi một tính toán trên Spark sẽ được thực hiện trên một cluster được quản lý bởi một máy chủ (master). Máy chủ sẽ thực hiện 2 nhiệm vụ chính đó là: Phân chia vùng dữ liệu tính toán về các nodes được phụ trách bởi các máy con (wokers) để thực hiện xử lý dữ liệu. Sau khi các máy con hoàn thành nhiệm vụ, master sẽ tổng hợp lại các kết quả từ các máy con và trả về client kết quả cuối cùng. Cơ chế này có thể được mô tả thông qua sơ đồ bên dưới.

Top



Hình 1: Kiến trúc của spark cluster (nguồn: Microsoft Azure)

Bên cạnh đó spark SQL còn có những ưu điểm sau:

- Làm việc trên nhiều kiểu định dạng dữ liệu có cấu trúc khác nhau như: csv, json, txt, parquet, hdfs, rdds, hive table, avro, parquet, orc.
- Hỗ trợ các cổng kết nối chuẩn là JDBC và ODBC để đưa lên các BI tools.
- Cho phép tạo các streaming data biến đổi dữ liệu.
- Được implement trên nhiều ngôn ngữ khác nhau như scala, python, R, java.
- Có các submodule tiện ích như: pyspark.Dataframe cho phép tạo các DataFrame với kiến trúc tương tự DataFrame trong python. submodule pyspark.sql triển khai các câu lệnh của SQL trên spark thông qua Spark SQL engine. Đây đều là những định dạng dữ liệu và câu lệnh quen thuộc đối với data scientist nên dễ dàng học và sử dụng.
- submodule pyspark.ml hỗ trợ rất nhiều các model nên dễ dàng triển khai lớp các bài toán dự báo và phân loại của machine learning.
- Kết hợp các Transformer (biến đổi dữ liệu) và Estimator (model dự báo, phân loại) ta dễ dàng tạo ra một pipeline đi qua tuần tự các bước xử lý.

Những ưu điểm trên đã giúp cho Spark trở thành một trong những giải pháp được ưa chuộng trong xây dựng mô hình và xử lý dữ liệu lớn. Sau đây chúng ta cùng tìm hiểu về cơ chế hoạt động và cách thức biến đổi, xây dựng model trên spark.

2. Cơ chế hoạt động của spark

Bước đầu tiên để thiết lập spark là tạo ra 1 cụm xử lý (cluster) trên một máy chủ. Cụm xử lý này được kết nối tới rất nhiều nodes khác nhau. Máy chủ (*master*) sẽ làm nhiệm vụ: phân chia dữ liệu và quản lý tính toán trên các máy con. máy chủ sẽ kết nối đến các máy con (*slaves*) trong cụm bằng các session. Máy chủ sẽ gửi dữ liệu và yêu cầu tính toán để máy con thực thi. Sau khi có kết quả máy con có nhiệm vụ trả về máy chủ. Máy chủ tổng hợp tất cả các tính toán trên máy con để tính ra kết quả cuối cùng.

Trong bài này do mới làm quen với spark nên mình sẽ khởi tạo một cluster trên local. Thay vì kết nối tới những máy khác, các tính toán sẽ được thực hiện chỉ trên server local thông qua một giả lập cụm.

Top

2.1. Khởi tạo một connection:

Để khởi tạo một connection chúng ta đơn giản là tạo ra một instance của SparkContext class. Class này sẽ có một vài tham số yêu cầu chúng ta phải khai báo để xác định các thuộc tính của cụm mà chúng ta muốn connect tới.

Những tham số này có thể được cấu hình thông qua constructor SparkConf() (http://spark.apache.org/docs/2.1.0/api/python/pyspark.html).

Một vài tham số quan trọng:

- Master: url connect tới master.
- AppName: Tên ứng dụng.
- SparkHome: Đường dẫn nơi spark được cài đặt trên các nodes.

Bên dưới ta cùng khởi tạo một SparkContext là `sc` :

```
from pyspark import SparkContext
# Stop spark if it existed.
try:
    sc.stop()
except:
    print('sc have not yet created!')

sc = SparkContext(master = "local", appName = "First app")
# Check spark context
print(sc)
# Check spark context version
print(sc.version)
```

```
<SparkContext master=local appName=First app>
2.3.1
```

Lưu ý chúng ta chỉ có thể khởi tạo SparkContext 1 lần nên nếu chưa stop `sc` mà chạy lại lệnh trên sẽ bị lỗi. Do đó lệnh SparkContext.getOrCreate() được sử dụng để khởi tạo mới SparkContext nếu nó chưa xuất hiện và lấy lại SparkContext cũ nếu đã được khởi tạo và đang run.

```
sc = SparkContext.getOrCreate()
print(sc)
```

```
<SparkContext master=local appName=First app>
```

2.2. Khởi tạo một Session trong SparkContext:

Như vậy sau bước trên ta đã có môi trường kết nối tới cluster. Tuy nhiên để hoạt động được trong môi trường này thì chúng ta cần phải khởi tạo session thông qua hàm SparkSession.

Top

```
from pyspark.sql import SparkSession
my_spark = SparkSession.builder.getOrCreate()
# Print my_spark session
print(my_spark)
```

```
<pyspark.sql.session.SparkSession object at 0x000001D9D96E18D0>
```

Hàm getOrCreate() của session cũng tương tự như context để tránh trường hợp phát sinh lỗi khi session đã tồn tại và đang hoạt động nhưng được khởi tạo lại.

Thuộc tính `catalog` trong 1 Session sẽ liệt kê toàn bộ các dữ liệu có bên trong 1 cluster. Trong đó để xem toàn bộ các bảng đang có trong cluster chúng ta sử dụng hàm `.listTables()`.

```
# List all table exist in spark sesion
my_spark.catalog.listTables()
```

```
[]
```

Hiện tại trên cluster đang chưa có bảng nào. Để thêm một bảng vào cluster chúng ta có thể đọc từ my_spark.read.csv() Dữ liệu file csv các bạn có thể download từ link sau flights.csv (https://www.kaggle.com/miquar/explore-flights-csv-airports-csv-airlines-csv/data)

```
flights = my_spark.read.csv('flights.csv', header = True)
# show fligths top 5
flights.show(5)
```

```
 +----+-----+---+-----------+-------+-------------+-----------+-------
 |YEAR|MONTH|DAY|DAY_OF_WEEK|AIRLINE|FLIGHT_NUMBER|TAIL_NUMBER|ORIGIN_
 +----+-----+---+-----------+-------+-------------+-----------+-------
 |2015|    1|  1|          4|     AS|           98|     N407AS|
 |2015|    1|  1|          4|     AA|         2336|     N3KUAA|
 |2015|    1|  1|          4|     US|          840|     N171US|
 |2015|    1|  1|          4|     AA|          258|     N3HYAA|
 |2015|    1|  1|          4|     AS|          135|     N527AS|
 +----+-----+---+-----------+-------+-------------+-----------+-------
 only showing top 5 rows

 []
```

Để kiểm tra các schema có trong một table chúng ta sử dụng hàm `printSchema()`.

```
flights.printSchema()
```

Top

```
root
 |-- YEAR: string (nullable = true)
 |-- MONTH: string (nullable = true)
 |-- DAY: string (nullable = true)
 |-- DAY_OF_WEEK: string (nullable = true)
 |-- AIRLINE: string (nullable = true)
 |-- FLIGHT_NUMBER: string (nullable = true)
 |-- TAIL_NUMBER: string (nullable = true)
 |-- ORIGIN_AIRPORT: string (nullable = true)
 |-- DESTINATION_AIRPORT: string (nullable = true)
 |-- SCHEDULED_DEPARTURE: string (nullable = true)
 |-- DEPARTURE_TIME: string (nullable = true)
 |-- DEPARTURE_DELAY: string (nullable = true)
 |-- TAXI_OUT: string (nullable = true)
 |-- WHEELS_OFF: string (nullable = true)
 |-- SCHEDULED_TIME: string (nullable = true)
 |-- ELAPSED_TIME: string (nullable = true)
 |-- AIR_TIME: string (nullable = true)
 |-- DISTANCE: string (nullable = true)
 |-- WHEELS_ON: string (nullable = true)
 |-- TAXI_IN: string (nullable = true)
 |-- SCHEDULED_ARRIVAL: string (nullable = true)
 |-- ARRIVAL_TIME: string (nullable = true)
 |-- ARRIVAL_DELAY: string (nullable = true)
 |-- DIVERTED: string (nullable = true)
 |-- CANCELLED: string (nullable = true)
 |-- CANCELLATION_REASON: string (nullable = true)
 |-- AIR_SYSTEM_DELAY: string (nullable = true)
 |-- SECURITY_DELAY: string (nullable = true)
 |-- AIRLINE_DELAY: string (nullable = true)
 |-- LATE_AIRCRAFT_DELAY: string (nullable = true)
 |-- WEATHER_DELAY: string (nullable = true)
```

2.3. Thêm một spark DataFrame lưu trữ local vào một catalog

Tuy nhiên lúc này flights vẫn chỉ là một spark DataFrame chưa có trong catalog của của cluster. Sử dụng hàm `listTable()` liệt kê danh sách bảng ta thu được 1 list rỗng.

```
# list all table exist in spark session
print(my_spark.catalog.listTables())
```

```
[]
```

Lý do là bởi khi được đọc từ hàm `read.csv()` thì dữ liệu chỉ được lưu trữ ở Local dưới dạng một spark DataFrame. Để đưa dự liệu từ local lên cluster chúng ta cần save nó dưới dạng một `temporary table` thông một trong những lệnh bên dưới:

- **.createTempView():** là một phương thức của spark DataFrame trong đó tham số duy nhất được truyền vào là tên bảng mà bạn muốn lưu trữ dưới dạng temporary table. Bảng được tạo ra là tạm thời và chỉ có thể được truy cập từ session được sử dụng để tạo ra spark DataFrame.

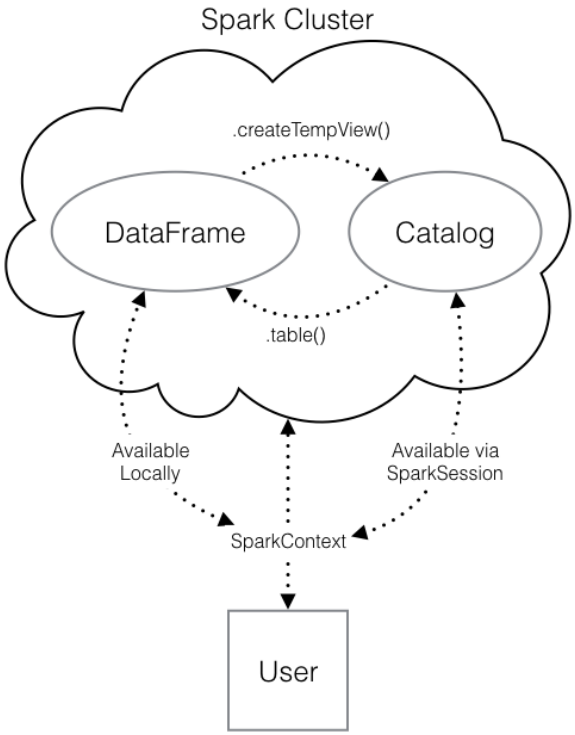
Top

- **.createOrReplaceTempView():** Có tác dụng hoàn toàn giống như `.createTempView()` nhưng nó sẽ update lại temporary table nếu nó đã tồn tại hoặc tạo mới nếu chưa tồn tại trước dây. Tránh trường hợp duplicate dữ liệu.

Ngoài ra chúng ta còn sử dụng:

- **.createDataFrame():** Tạo một spark DataFrame từ một pandas DataFrame.

Để hiểu rõ hơn về nguyên tắc khởi tạo bảng chúng ta có thể xem sơ đồ bên dưới.



Hình 2: Sơ đồ chuyển đổi giữa spark DataFrame và catalog. Tử sơ đồ trên ta có thể thấy Spark Cluster sẽ tương tác với user thông qua kết nối từ SparkContext. Có 2 dạng lưu trữ chính ở SparkContext là spark DataFrame và catalog. Trong đó spark DataFrame là định dạng dạng bảng được lưu trữ ở Local và catalog là các temporary table sống trong các SparkSession. Để dữ liệu có thể truy cập từ một session chúng ta cần chuyển nó từ định dạng spark DataFrame sang temporary table thông qua hàm `.createTempView()` hoặc `.createOrReplaceTempView()`.

Bên dưới ta sẽ khởi tạo một temporary table với tên là flights_temp cho bảng flights.

```
# Create a temporary table on catalog of local data frame flights as n
flights.createOrReplaceTempView('flights_temp')
# check list all table available on catalog
my_spark.catalog.listTables()
```

Top

```
[Table(name='flights_temp', database=None, description=None, tableType=
```

2.4. Các lệnh biến đổi dữ liệu của spark DataFrame

Thông thường sẽ gồm các lệnh chính như tạo trường, update trường, xóa trường. Các lệnh bên dưới đều là các thuộc tính của spark DataFrame.

- **.withColumn("newColumnName", formular):** Thêm một trường mới vào một bảng sẵn có. Gồm 2 tham số chính, tham số thứ nhất là tên trường mới, tham số thứ 2 là công thức cập nhật tên trường. Lưu ý rằng spark DataFrame là một dạng dữ liệu immutable (không thể modified được). Do đó ta không thể inplace update (như các hàm `fillna()` hoặc `replace()` của pandas dataframe) mà cần phải gán giá trị trả về vào chính tên bảng ban đầu để cập nhật trường mới.

Chẳng hạn bên dưới ta sẽ thêm 1 trường mới là HOUR_ARR được tính dựa trên AIR_TIME/60 (qui từ phút ra h) của bảng flights như sau:

```
flights = flights.withColumn('HOUR_ARR', flights.AIR_TIME/60)
flights.printSchema()
```

Top

```
root
 |-- YEAR: string (nullable = true)
 |-- MONTH: string (nullable = true)
 |-- DAY: string (nullable = true)
 |-- DAY_OF_WEEK: string (nullable = true)
 |-- AIRLINE: string (nullable = true)
 |-- FLIGHT_NUMBER: string (nullable = true)
 |-- TAIL_NUMBER: string (nullable = true)
 |-- ORIGIN_AIRPORT: string (nullable = true)
 |-- DESTINATION_AIRPORT: string (nullable = true)
 |-- SCHEDULED_DEPARTURE: string (nullable = true)
 |-- DEPARTURE_TIME: string (nullable = true)
 |-- DEPARTURE_DELAY: string (nullable = true)
 |-- TAXI_OUT: string (nullable = true)
 |-- WHEELS_OFF: string (nullable = true)
 |-- SCHEDULED_TIME: string (nullable = true)
 |-- ELAPSED_TIME: string (nullable = true)
 |-- AIR_TIME: string (nullable = true)
 |-- DISTANCE: string (nullable = true)
 |-- WHEELS_ON: string (nullable = true)
 |-- TAXI_IN: string (nullable = true)
 |-- SCHEDULED_ARRIVAL: string (nullable = true)
 |-- ARRIVAL_TIME: string (nullable = true)
 |-- ARRIVAL_DELAY: string (nullable = true)
 |-- DIVERTED: string (nullable = true)
 |-- CANCELLED: string (nullable = true)
 |-- CANCELLATION_REASON: string (nullable = true)
 |-- AIR_SYSTEM_DELAY: string (nullable = true)
 |-- SECURITY_DELAY: string (nullable = true)
 |-- AIRLINE_DELAY: string (nullable = true)
 |-- LATE_AIRCRAFT_DELAY: string (nullable = true)
 |-- WEATHER_DELAY: string (nullable = true)
 |-- HOUR_ARR: double (nullable = true)
```

- **.withColumnRenamed(“oldColumnName”, “newColumnName”):** Đổi tên của một column name trong pandas DataFrame.
- **.select(“column1”, “column2”, … , “columnt”, formular):** Lựa chọn danh sách các trường trong spark DataFrame thông qua các tên column được truyền vào đưới dạng string và tạo ra một trường mới thông qua formular. Lưu ý để đặt tên cho trường mới ứng với formular chúng ta sẽ cần sử dụng hàm `formula.alias("columnName")`.

Bên dưới chúng ta sẽ tạo ra trường `avg_speed` tính vận tốc trung bình của các máy bay bằng cách lấy khoảng cách (DISTANCE) chi cho thời gian bay (HOUR_ARR) group by theo mã máy bay (TAIL_NUMBER) bằng lệnh select.

```
avg_speed = flights.select("ORIGIN_AIRPORT", "DESTINATION_AIRPORT", "TA
avg_speed.printSchema()
avg_speed.show(5)
```

Top

```
+--------------+-------------------+-----------+-----------------+
|ORIGIN_AIRPORT|DESTINATION_AIRPORT|TAIL_NUMBER|        avg_speed|
+--------------+-------------------+-----------+-----------------+
|           ANC|                SEA|     N407AS|514.0828402366864|
|           LAX|                PBI|     N3KUAA|531.5589353612166|
|           SFO|                CLT|     N171US|517.8947368421052|
|           LAX|                MIA|     N3HYAA|544.6511627906978|
|           SEA|                ANC|     N527AS|436.5829145728643|
+--------------+-------------------+-----------+-----------------+
only showing top 5 rows
```

- **.selectExpr("column1", "column2", ... , "columnt", "formularExpr"):** Hoàn toàn tương tự như `.select()` nhưng tham số formular được thay thế bằng chuỗi string biểu diễn công thức như trong câu lệnh SQL.

```
avg_speed_exp = flights.selectExpr("ORIGIN_AIRPORT", "DESTINATION_AIRP
avg_speed_exp.printSchema()
avg_speed_exp.show(5)
```

```
+--------------+-------------------+-----------+-----------------+
|ORIGIN_AIRPORT|DESTINATION_AIRPORT|TAIL_NUMBER|        avg_speed|
+--------------+-------------------+-----------+-----------------+
|           ANC|                SEA|     N407AS|514.0828402366864|
|           LAX|                PBI|     N3KUAA|531.5589353612166|
|           SFO|                CLT|     N171US|517.8947368421052|
|           LAX|                MIA|     N3HYAA|544.6511627906978|
|           SEA|                ANC|     N527AS|436.5829145728643|
+--------------+-------------------+-----------+-----------------+
only showing top 5 rows
```

- **.filter(condition):** Lọc một bảng theo một điều kiện nào đó. Condition có thể làm một string expression biểu diễn công thức lọc hoặc một công thức giữa các trường trong spark DataFrame. Lưu ý Condition phải trả về một trường dạng Boolean type.

Chẳng hạn chúng ta muốn lọc những chuyến bay xuất phát từ sân bay `SEA` và điểm đến là `ANC` ta có thể sử dụng filter như sau:

```
filter_SEA_ANC = flights.filter("ORIGIN_AIRPORT == 'SEA'") \
                        .filter("DESTINATION_AIRPORT == 'ANC'")
filter_SEA_ANC.show(5)
```

```
+----+-----+---+-----------+-------+-------------+-----------+-------
|YEAR|MONTH|DAY|DAY_OF_WEEK|AIRLINE|FLIGHT_NUMBER|TAIL_NUMBER|ORIGIN_
+----+-----+---+-----------+-------+-------------+-----------+-------
|2015|    1|  1|          4|     AS|          135|     N527AS|
|2015|    1|  1|          4|     AS|           81|     N577AS|
|2015|    1|  1|          4|     AS|           83|     N532AS|
|2015|    1|  1|          4|     AS|          111|     N570AS|
|2015|    1|  1|          4|     AS|           85|     N764AS|
+----+-----+---+-----------+-------+-------------+-----------+-------
only showing top 5 rows
```

Top

- **.groupBy(“column1”, “column2”,…,”columnt”):** Tương tự như lệnh GROUP BY của SQL, lệnh này sẽ nhóm các biến theo các dimension được truyền vào groupBy. Theo sau lệnh `groupBy()` là một build-in function của spark DataFrame được sử dụng để tính toán theo một biến đo lường nào đó chẳng hạn như hàm `avg()`, `min()`, `max()`, `sum()`. Tham số được truyền vào các hàm này chính là tên biến đo lường.

Bên dưới chúng ta sẽ tính thời gian bay trung bình theo điểm xuất phát (ORIGIN_AIRPORT).

```
avg_time_org_airport = flights.groupBy("ORIGIN_AIRPORT").avg("HOUR_ARR
avg_time_org_airport.show(5)
```

```
+--------------+------------------+
|ORIGIN_AIRPORT|     avg(HOUR_ARR)|
+--------------+------------------+
|           BGM| 1.096525096525096|
|           PSE|3.0352529358626916|
|           INL|0.5327937649880096|
|           DLG|0.8359307359307364|
|         12888|0.4413461538461539|
+--------------+------------------+
only showing top 5 rows
```

- **.join(tableName, on = “columnNameJoin”, how = “leftouter”):** Join 2 bảng với nhau tương tự như lệnh left join trong SQL. Kết quả trả về sẽ là các trường mới trong bảng `tableName` kết hợp với các trường cũ trong bảng gốc thông qua key là. Lưu ý rằng `columnNameJoin` phải trùng nhau giữa 2 bảng.

2.6. các lệnh biến đổi dữ liệu của spark.sql()

Một trong những lợi thế khi đưa bảng lên cluster đó là có thể sử dụng các câu lệnh biến đổi SQL từ phương thức `.sql()` của spark để transform dữ liệu. Bên dưới là các lệnh cơ bản mà chúng sẽ có thể gặp:

- **SELECT**: Có cú pháp chung là: `SELECT * FROM TABLENAME WHERE CONDTION` Lệnh này sẽ lựa chọn các trường trong bảng theo điều kiện tại WHERE. Đây là lệnh rất quen thuộc trong SQL.

Chẳng hạn bên dưới chúng ta lấy ra những chuyến bay có thời gian bay > 10 phút.

```
flights_10 = my_spark.sql('SELECT * FROM flights_temp WHERE AIR_TIME >
flights_10.show(5)
```

Top

```
+----+-----+---+-----------+-------+-------------+-----------+-------
|YEAR|MONTH|DAY|DAY_OF_WEEK|AIRLINE|FLIGHT_NUMBER|TAIL_NUMBER|ORIGIN_
+----+-----+---+-----------+-------+-------------+-----------+-------
|2015|    1|  1|          4|     AS|           98|     N407AS|
|2015|    1|  1|          4|     AA|         2336|     N3KUAA|
|2015|    1|  1|          4|     US|          840|     N171US|
|2015|    1|  1|          4|     AA|          258|     N3HYAA|
|2015|    1|  1|          4|     AS|          135|     N527AS|
+----+-----+---+-----------+-------+-------------+-----------+-------
only showing top 5 rows
```

- **GROUP BY:** Lệnh này sẽ thống kê các các trường measurement (các trường dùng để tính toán) theo một nhóm các trường dimension (các trường dùng để phân loại) dựa trên các aggregation function như `sum()`, `avg()`, `min()`, `max()`, `mean()`, `median()`, `count()`, `count(distinct())` của SQL.

Chẳng hạn chúng ta muốn tính số phút bay trung bình của mỗi hãng bay trong năm sẽ sử dụng hàm GROUP BY như sau:

```
print(my_spark.catalog.listTables())
agg_arr_time = my_spark.sql("SELECT ORIGIN_AIRPORT, DESTINATION_AIRPOR
agg_arr_time.show(5)
```

```
[Table(name='flights_temp', database=None, description=None, tableTyp
+--------------+-------------------+-----------+------------------+
|ORIGIN_AIRPORT|DESTINATION_AIRPORT|TAIL_NUMBER|         avg_speed|
+--------------+-------------------+-----------+------------------+
|           IAG|                FLL|     N630NK|             158.0|
|           RIC|                ATL|     N947DN|  75.6470588235294|
|           EWR|                ATL|     N970AT|108.02777777777777|
|           MSN|                ORD|     N703SK|              28.0|
|           AVL|                ATL|     N994AT|              30.5|
+--------------+-------------------+-----------+------------------+
only showing top 5 rows
```

Ngoài ra ta còn có thể sử dụng vô số các lệnh SQL khác đối với các bảng xuất hiện trong catalog của clusters.

- **UPDATE**: Cập nhật một trường theo một công thức nào đó.
- **INSERT**: Insert thêm row mới cho bảng.
- **DELETE**: Xóa các records của bảng theo điều kiện.
- **Nhóm các lệnh join**: gồm các lệnh `left join`, `right join`, `inner join`, `outer join` .

Các ví dụ về lệnh này trên SQL các bạn có thể xem SQL tutorial - Website W3School (https://www.w3schools.com/sql/).

Top

3. Xây dựng pipeline End-to-End model trên pyspark

3.1. Xây dựng pipeline biến đổi dữ liệu.

pyspark cho phép xây dựng các end-to-end model mà dữ liệu truyền vào là các raw data và kết quả trả ra là nhãn, xác xuất hoặc giá trị được dự báo của model. Các end-to-end model này được đi qua một pipe line của

pyspark.ml bao gồm 2 class cơ bản là `Transformer` cho phép biến đổi dữ liệu và `Estimator` ước lượng mô hình dự báo.

- Transfromer sử dụng hàm `.transform()` nhận đầu vào là 1 DataFrame và trả ra một DataFrame mới có các trường đã biến đổi theo Transform. Các bạn sẽ hiểu hơn qua thực hành ở ví dụ bên dưới.
- Estimator sử dụng hàm `.fit()` để huấn luyện model. Chúng cũng nhận đầu vào là một DataFrame nhưng kết quả được trả ở đầu ra là 1 model object. Hiện tại spark hỗ trợ khá nhiều các lớp model cơ bản trong machine learning. Các lớp model xuất hiện trong Esimator bao gồm:
    - **Đối với bài toán phân loại:** `LogisticRegression, DecisionTreeClassifier, RandomForestModel, GBTClassifier (gradient bosting tree), MultilayerPerceptronClassifier, LinearSVC (Linear Support Vector Machine), NaiveBayes .`
    - **Đối với bài toán dự báo:** `GeneralizedLinearRegression, DecisionTreeRegressor, RandomForestRegressor, GBTRegressor (gradient boosting Tree), AFTSurvivalRegression (Hồi qui đối với các lớp bài toán estimate survival).`

Cách thức áp dụng các model này các bạn có thể xem hướng dẫn rất chi tiết tại trang chủ của Spark apache (https://spark.apache.org/docs/latest/ml-classification-regression.html).

Bên dưới chúng ta sẽ cùng đi xây dựng 1 pipeline cho model dự báo khả năng trễ chuyến bay dựa trên dữ liệu đầu vào là bảng flights. Trước tiên chúng ta cùng tìm hiểu các trường trong bảng dữ liệu.

```
1     flights.printSchema()
```

Top

```
root
 |-- YEAR: string (nullable = true)
 |-- MONTH: string (nullable = true)
 |-- DAY: string (nullable = true)
 |-- DAY_OF_WEEK: string (nullable = true)
 |-- AIRLINE: string (nullable = true)
 |-- FLIGHT_NUMBER: string (nullable = true)
 |-- TAIL_NUMBER: string (nullable = true)
 |-- ORIGIN_AIRPORT: string (nullable = true)
 |-- DESTINATION_AIRPORT: string (nullable = true)
 |-- SCHEDULED_DEPARTURE: string (nullable = true)
 |-- DEPARTURE_TIME: string (nullable = true)
 |-- DEPARTURE_DELAY: string (nullable = true)
 |-- TAXI_OUT: string (nullable = true)
 |-- WHEELS_OFF: string (nullable = true)
 |-- SCHEDULED_TIME: string (nullable = true)
 |-- ELAPSED_TIME: string (nullable = true)
 |-- AIR_TIME: string (nullable = true)
 |-- DISTANCE: string (nullable = true)
 |-- WHEELS_ON: string (nullable = true)
 |-- TAXI_IN: string (nullable = true)
 |-- SCHEDULED_ARRIVAL: string (nullable = true)
 |-- ARRIVAL_TIME: string (nullable = true)
 |-- ARRIVAL_DELAY: string (nullable = true)
 |-- DIVERTED: string (nullable = true)
 |-- CANCELLED: string (nullable = true)
 |-- CANCELLATION_REASON: string (nullable = true)
 |-- AIR_SYSTEM_DELAY: string (nullable = true)
 |-- SECURITY_DELAY: string (nullable = true)
 |-- AIRLINE_DELAY: string (nullable = true)
 |-- LATE_AIRCRAFT_DELAY: string (nullable = true)
 |-- WEATHER_DELAY: string (nullable = true)
 |-- HOUR_ARR: double (nullable = true)
```

Ý nghĩa các trường như sau:

- YEAR: string (nullable = true): Năm.
- MONTH: string (nullable = true): Tháng.
- DAY: string (nullable = true): Ngày.
- DAY_OF_WEEK: string (nullable = true): Ngày trong tuần.
- AIRLINE: string (nullable = true): Hãng hàng không.
- FLIGHT_NUMBER: string (nullable = true): Mã chuyến bay.
- TAIL_NUMBER: string (nullable = true): Số hiệu máy bay.
- ORIGIN_AIRPORT: string (nullable = true): Nơi xuất phát.
- DESTINATION_AIRPORT: string (nullable = true): Điểm đến.
- SCHEDULED_DEPARTURE: string (nullable = true): Lịch trình xuất phát.
- DEPARTURE_TIME: string (nullable = true): Thời gian xuất phát thực tế.
- DEPARTURE_DELAY: string (nullable = true): Thời gian bị trễ.
- TAXI_OUT: string (nullable = true): Thời gian taxi ra.
- WHEELS_OFF: string (nullable = true): Thời gian lăn bánh cất cánh.
- SCHEDULED_TIME: string (nullable = true): Thời gian theo lịch trình.
- ELAPSED_TIME: string (nullable = true): Không rõ.
- AIR_TIME: string (nullable = true): Thời gian cất cánh.
- DISTANCE: string (nullable = true): Khoảng cách.
- WHEELS_ON: string (nullable = true): Thời gian lăn bánh hạ cánh.

Top

- TAXI_IN: string (nullable = true): thời gian taxi vào
- SCHEDULED_ARRIVAL: string (nullable = true): Thời gian theo lịch di chuyển.
- ARRIVAL_TIME: string (nullable = true): Thời gian di chuyển.
- ARRIVAL_DELAY: string (nullable = true): Thời gian trễ.
- DIVERTED: string (nullable = true): Chuyển hướng.
- CANCELLED: string (nullable = true): Hủy chuyến.
- CANCELLATION_REASON: string (nullable = true): Lý do hủy.
- AIR_SYSTEM_DELAY: string (nullable = true): Trễ vì hệ thống hàng không.
- SECURITY_DELAY: string (nullable = true): Trễ vì lý do an ninh.
- AIRLINE_DELAY: string (nullable = true): Trễ vì lý do từ hãng.
- LATE_AIRCRAFT_DELAY: string (nullable = true): Trễ vì phi cơ.
- WEATHER_DELAY: string (nullable = true): Trễ vì thời tiết.
- HOUR_ARR: double (nullable = true): Số h di chuyển.

Vì là ví dụ demo nên để giảm thiểu thời gian tính toán tôi sẽ chỉ lấy dữ liệu của những chuyến bay xuất phát từ 'SEA' của hãng delta airline và american airlines ('DA' và 'AA'). Dữ liệu dự báo bao gồm các trường: ARRIVAL_DELAY, ARRIVAL_TIME, MONTH, YEAR, DAY_OF_WEEK, DESTINATION_AIRPORT, AIRLINE. Trong đó trường ARRIVAL_DELAY > 0 xác định chuyến bay trễ và ARRIVAL_DELAY = 0 chuyến bay không bị trễ.

Do các mô hình chỉ nhận đầu vào là các biến numeric nên ta phải có các bước xử lý dữ liệu từ biến string, boolean sang biến numeric.

Chúng ta sẽ phân các biến trên thành 2 nhóm biến là:

- **Các biến numeric:** ARRIVAL_TIME, MONTH, YEAR, DAY_OF_WEEK. Không cần phải qua biến đổi và được sử dụng trực tiếp làm đầu vào của model hồi qui. Tuy nhiên các biến này đang được để ở dạng string nên phải chuyển qua numeric bằng hàm CAST.
- **Các biến string:** DESTINATION_AIRPORT, AIRLINE là các biến dạng category cần được đánh index và biến đổi sang dạng biến dummies (chỉ nhận giá trị 0 và 1) để có thể đưa vào model hồi qui. Khi đó mỗi một biến sẽ được phân thành nhiều features mà mỗi một features đại diện cho 1 nhóm của biến. Quá trình biến đổi dummies sẽ trải qua 2 bước: Đánh index cho biến bằng class `StringIndexer()`. Một index sẽ được gán cho 1 nhóm biến. Biểu diễn one-hot vector thông qua class `OneHotEncoder()`: Từ index của biến sẽ biểu diễn các biến dưới dạng one-hot vector sao cho vị trí bằng 1 sẽ là phần tử có thứ tự trùng với index. Cả 2 class `StringIndexer()` và `OneHotEncoder()` đều là các object của `pyspark.ml.feature`.

```
print('Shape of previous data: ({}, {})'.format(flights.count(), len(f
flights_SEA = my_spark.sql("select ARRIVAL_DELAY, ARRIVAL_TIME, MONTH,
print('Shape of flights_SEA data: ({}, {})'.format(flights_SEA.count()
```

```
Shape of previous data: (5819079, 32)
Shape of flights_SEA data: (19956, 7)
```

Top

```
# Create boolean variable IS_DELAY variable as Target
flights_SEA = flights_SEA.withColumn("IS_DELAY", flights_SEA.ARRIVAL_
# Now Convert Boolean variable into integer
flights_SEA = flights_SEA.withColumn("label", flights_SEA.IS_DELAY.ca
# Remove missing value
model_data = flights_SEA.filter("ARRIVAL_DELAY is not null \
                                and ARRIVAL_TIME is not null \
                                and MONTH is not null \
                                and YEAR is not null  \
                                and DAY_OF_WEEK is not null \
                                and DESTINATION_AIRPORT is not null \
                                and AIRLINE is not null")

print('Shape of model_data data: ({}, {})'.format(model_data.count(),
```

```
Shape of model_data data: (19823, 9)
```

Một chú ý quan trọng đó là các model phân loại của pyspark luôn mặc định nhận biến dự báo là label. Do đó trong bất kì model nào chúng ta cũng cần tạo ra biến integer là nhãn của model dưới tên label.

Convert các biến string sang numeric bằng hàm `.withColumn()`

```
# ARRIVAL_TIME, MONTH, YEAR, DAY_OF_WEEK
model_data = model_data.withColumn("ARRIVAL_TIME", model_data.ARRIVAL_
model_data = model_data.withColumn("MONTH", model_data.MONTH.cast("int
model_data = model_data.withColumn("YEAR", model_data.YEAR.cast("integ
model_data = model_data.withColumn("DAY_OF_WEEK", model_data.DAY_OF_WE
model_data.printSchema()
```

```
root
 |-- ARRIVAL_DELAY: string (nullable = true)
 |-- ARRIVAL_TIME: integer (nullable = true)
 |-- MONTH: integer (nullable = true)
 |-- YEAR: integer (nullable = true)
 |-- DAY_OF_WEEK: integer (nullable = true)
 |-- DESTINATION_AIRPORT: string (nullable = true)
 |-- AIRLINE: string (nullable = true)
 |-- IS_DELAY: boolean (nullable = true)
 |-- label: integer (nullable = true)
```

Biến đổi các biến String bằng StringIndexer và OneHotEncoder.

Top

```
from pyspark.ml.feature import StringIndexer, OneHotEncoder
# I. With DESTINATION_AIRPORT
# Create StringIndexer
dest_indexer = StringIndexer(inputCol = "DESTINATION_AIRPORT", \
                             outputCol = "DESTINATION_INDEX")

# Create OneHotEncoder
dest_onehot = OneHotEncoder(inputCol = "DESTINATION_INDEX", \
                            outputCol = "DESTINATION_FACT")

# II. With AIRLINE
# Create StringIndexer
airline_indexer = StringIndexer(inputCol = "AIRLINE", \
                                outputCol = "AIRLINE_INDEX")

# Create OneHotEncoder
airline_onehot = OneHotEncoder(inputCol = "AIRLINE_INDEX", \
                               outputCol = "AIRLINE_FACT")
```

Đầu ra của quá trình biến đổi trên 2 biến DESTINATION_AIRPORT và AIRLINE chính là biến DESTINATION_FACT và AIRLINE_FACT. Hai biến này sẽ cần được đưa vào vector tổng hợp sẽ được giới thiệu biên dưới. Các objects: dest_indexer, dest_onehot, airline_indexer, airline_onehot sẽ được truyền vào pipeline theo thứ tự để thể hiện quá trình transform dữ liệu tuần tự từ trái qua phải.

Bước cuối cùng của pipeline là kết hợp toàn bộ các columns chứa các features thành một column duy nhất. Bước này phải được thực hiện trước khi model training bởi spark model sẽ chỉ chấp nhận đầu vào ở định dạng này. Chúng ta có thể lưu mỗi một giá trị từ một cột như một phần tử của vector. Khi đó vector sẽ chứa toàn bộ các thông tin cần thiết của 1 quan sát để xác định nhãn hoặc giá trị dự báo ở đầu ra của quan sát đó. Class `VectorAssembler` của pyspark.ml.feature sẽ tạo ra vector tổng hợp đại diện cho toàn bộ các chiều của quan sát đầu vào. Việc chúng ta cần thực hiện chỉ là truyền vào class list string tên các trường thành phần của vector tổng hợp.

```
# Make a VectorAssembler
from pyspark.ml.feature import VectorAssembler
vec_assembler = VectorAssembler(inputCols = ["ARRIVAL_TIME", "MONTH",
                                             "DAY_OF_WEEK", "DESTINATI
                                             "AIRLINE_FACT"],
                                outputCol = "features")
```

Sau stage này chúng ta sẽ tổng hợp các biến trong inputCols thành một vector dự báo ở outputCol được lưu dưới tên `features` . Nhãn của model luôn mặc định là biến `label` đã khởi tạo từ đầu.

Tiếp theo chúng ta sẽ khởi tạo pipeline biến đổi dữ liệu cho model thông qua class `Pipeline` của `pyspark.ml` . Các transformer biến đổi dữ liệu sẽ được sắp xếp trong 1 list và truyền vào tham số stages như bên dưới.

Top

```
from pyspark.ml import Pipeline

# Make a pipeline
flights_sea_pipe  = Pipeline(stages = [dest_indexer, dest_onehot, airl
                                       airline_onehot, vec_assembler])
```

Sau khi dữ liệu được `fit()` qua pipeline sẽ thu được đầu ra là vector tổng hợp các biến dự báo `features` và nhãn của quan sát `label` .

3.2. Huấn luyện và đánh giá model.

3.2.1. Phân chia tập train/test.

Sau khi đi qua pipe line ở bước Transform dữ liệu chúng ta sẽ thu được vector tổng hợp (Vector Assemble) và nhãn (LABLE) ở đầu ra. Tiếp theo tại bước này sẽ thực hiện phân chia tập train/test theo tỷ lệ 80%/20%. Trong đó model được xây dựng trên tập Train và được đánh giá trên tập Test. Tại sao lại phải để riêng 1 tập test mà không hồi qui trên toàn bộ dữ liệu? Đó là vì model thường phân loại hoặc dự báo tốt trên dữ liệu nó đã được huấn luyện mà không phân loại, dự báo tốt đối với dữ liệu mới. Chính vì thế tập test được coi như dữ liệu mới không được sử dụng trong huấn luyện và dùng để đánh giá khả năng dự báo của model.

```
# create pipe_data from pipeline
pipe_data = flights_sea_pipe.fit(model_data).transform(model_data)
```

```
# Split train/test data
train, test = pipe_data.randomSplit([0.8, 0.2])
```

3.2.2. Huấn luyện và đánh giá model.

Có rất nhiều các lớp model phân loại đã được giới thiệu trong mục 3.1. Để đơn giản hóa chúng ta sẽ chọn ra model LogisticRegression trong ví dụ demo này làm model phân loại.

```
from pyspark.ml.classification import LogisticRegression

# Create logistic regression
lr = LogisticRegression()
```

Để đánh giá model chúng ta cần sử dụng các metric như `ROC, Accuracy, F1, precision hoặc recal` . Do dữ liệu không có hiện tượng mất cân bằng giữa 2 lớp nên ta sẽ sử dụng ROC làm metric đánh giá model. Trong trường hợp mẫu mất cân bằng thì các chỉ số `F1, precision hoặc recal` nên được sử dụng thay thế vì trong tình huống này ROC, Accuracy thường mặc định là rất cao. Chúng ta sử dụng class `BinaryClassificationEvaluator` trong `pyspark.ml.evaluation` module để tính toán các metrics đánh giá model.

Top

```
# Import the evaluation submodule
import pyspark.ml.evaluation as evals

# Create a BinaryClassificationEvaluator
evaluator = evals.BinaryClassificationEvaluator(metricName = "areaUnde
```

3.2.3. Tuning model thông qua param gridSearch.

pyspark cho phép chúng ta tuning model trên một gridSearch các params. Tuning model được hiểu là tìm kiếm trong 1 không gian các tham số của model để thu được model mà giá trị của metric khai báo ở bước 3.2.2 thu được là nhỏ nhất trên tập test. Để xây dựng một gridSearch chúng ta sử dụng class ParamGridBuilder của submodule pyspark.ml.tuning. Để thêm các gridSearch chúng ta sử dụng phương thức `.addGrid()` và sau đó sử dụng hàm `.build()` để khởi tạo gridSearch. Mỗi lớp model sẽ có các tham số khác nhau. Do đó để tuning được model cần phải xác định xem model có những tham số gì. Việc này đòi hỏi bạn phải nắm vững về thuật toán và phương pháp tối ưu của model. Mình sẽ không đi sâu vào phần này. Đối với LogisticRegression mình sẽ lựa chọn lr.regParam là tham số tuning. Một lưu ý khác là việc tuning có thể rất tốt thời gian và tài nguyên của máy tính do thực hiện nhiều model của cùng 1 lớp model nhưng với các tham số khau. Do đó bạn cần cân nhắc cấu hình máy trước khi thực hiện.

```
# Import the tuning submodule
import pyspark.ml.tuning as tune
import numpy as np

# Create the parameter grid
grid = tune.ParamGridBuilder()

# Add the hyperparameter, we can add more than one hyperparameter
grid = grid.addGrid(lr.regParam, np.arange(0, 0.1, 0.01))

# Build the grid
grid = grid.build()
```

Sau khi build xong gridSearch chúng ta cần Cross Validate toàn bộ các model trên tập các tham số khởi tạo ở `grid` . Nếu bạn đọc chưa nắm rõ về Cross Validate là gì xin mời đọc bài viết Cross validation tuning threshold (http://rpubs.com/phamdinhkhanh/383309) của mình. Chúng ta cần khởi tạo một cross validator từ class CrossValidator của pyspark.ml.tuning.

```
# Create the CrossValidator
cv = tune.CrossValidator(estimator=lr,
                         estimatorParamMaps=grid,
                         evaluator=evaluator
                         )

# Fit cross validation on models
models = cv.fit(train)
```

Mình sẽ không chạy lệnh này do máy cấu hình yếu. Sau khi tuning xong chúng ta sẽ thu được model tốt nhất thông qua hàm:

Top

```
best_lr = models.bestModel
```

Do không chạy tuning nên mình sẽ gán cho best_lr model chính là model LogisticRegression với giá trị mặc định của regParam = 0.

```
best_lr = lr.fit(train)
print(best_lr)
```

```
LogisticRegression_44d7b66a6ac3918984b4
```

Lưu ý để dự báo từ các model của pyspark, chúng ta không sử dụng hàm `predict()` như thông thường mà sử dụng hàm `transform()`. Kiểm tra mức độ chính xác của model trên tập test bằng hàm `evaluate()`.

```
# Use the model to predict the test set
test_results = best_lr.transform(test)

# Evaluate the predictions
print(evaluator.evaluate(test_results))
```

```
0.5890717985162973
```

Như vậy trên tập test model đạt mức độ chính xác khoảng 58.9%. Phương án tốt được cân nhắc là thêm biến để cải thiện model.

4. Tổng kết

Như vậy thông qua bài viết này chúng ta đã được nắm bắt được hiệu quả của spark trong xử lý dữ liệu lớn. Tôi xin tổng hợp lại các kiến thức thu được trong bài:

1. Cách thức khởi tạo một spark context và quản lý các table trên catalog của cluster.
2. Các lệnh cơ bản biến đổi và tổng hợp dữ liệu trên spark DataFrame.
3. Các lệnh SQL được tích hợp trên spark.
4. Tạo pipeline biến đổi dữ liệu cho model phân loại.
5. Xây dựng và đánh giá model trên pyspark.

Ngoài ra còn rất nhiều các kiến thức về thiết lập cụm và xử lý tính toán khác tôi sẽ giới thiệu ở những bài tiếp theo. Hi vọng những kiến thức này sẽ giúp ích bạn đọc trong việc xây dựng các model với dữ liệu lớn. Bài viết được tổng hợp từ rất nhiều các nguồn kiến thức trên mạng mà tôi sẽ liệt kê bên đưới.

5. Tài liệu

1. SparkSQL guideline - Spark homepage (https://spark.apache.org/docs/2.3.1/sql-programming-guide.html)
2. Spark Machine Learning Lib - Spark homepage (https://spark.apache.org/docs/2.3.1/ml-guide.html)
3. Spark structured streaming data - Spark homepage (https://spark.apache.org/docs/2.3.1/structured-streaming-programming-guide.html)
4. Bigdata essentials - coursera (https://www.coursera.org/lecture/big-data-essentials/getting-started-with-spark-python-o6oKt)

Top

5. PySpark Tutorial for Beginners - youtube (https://www.youtube.com/watch?v=639JCua-bQU)
6. CCA 175 - Spark and Hadoop Developer - udemy (https://www.udemy.com/cca-175-spark-and-hadoop-developer-python-pyspark/)
7. Introduction to spark - datacamp (https://campus.datacamp.com/courses/introduction-to-pyspark)
8. Spark and Python for Big Data with PySpark - udemy (https://www.udemy.com/course/spark-and-python-for-big-data-with-pyspark)

Top