Spark & sparklyr part II

Programming for Statistical Science

Shawn Santo

Supplementary materials

Full video lecture available in Zoom Cloud Recordings

Additional resources

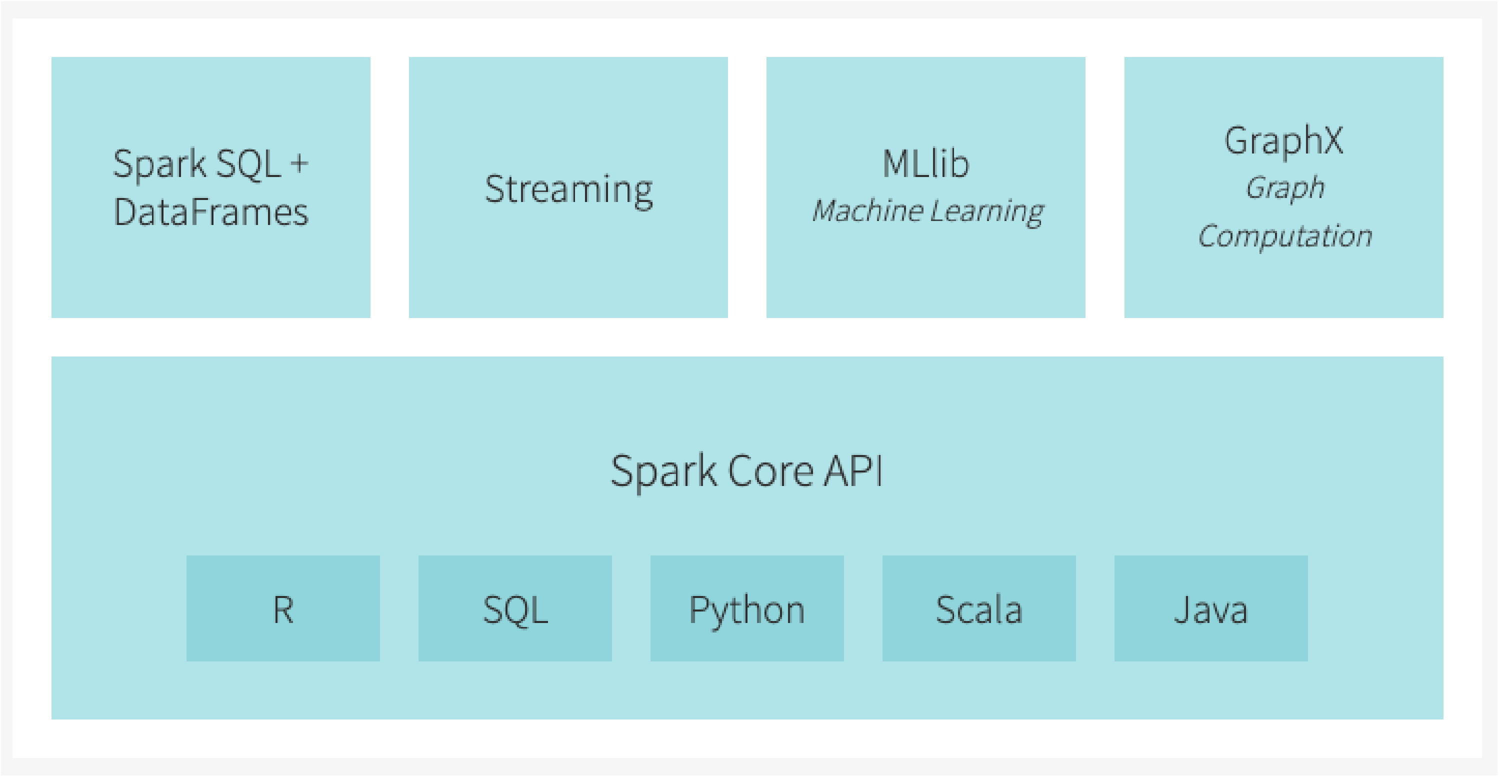
- `sparklyr`: R interface for Apache Spark
- R Front End for Apache Spark
- Mastering Spark with R

Recall

What is Apache Spark?

- As described by Databricks, "Spark is a unified computing engine and a set of libraries for parallel data processing on computing clusters".
- Spark's goal is to support data analytics tasks within a single ecosystem: data loading, SQL queries, machine learning, and streaming computations.
- Spark is written in Scala and runs on Java. However, Spark can be used from R, Python, SQL, Scala, or Java.

The Spark ecosystem



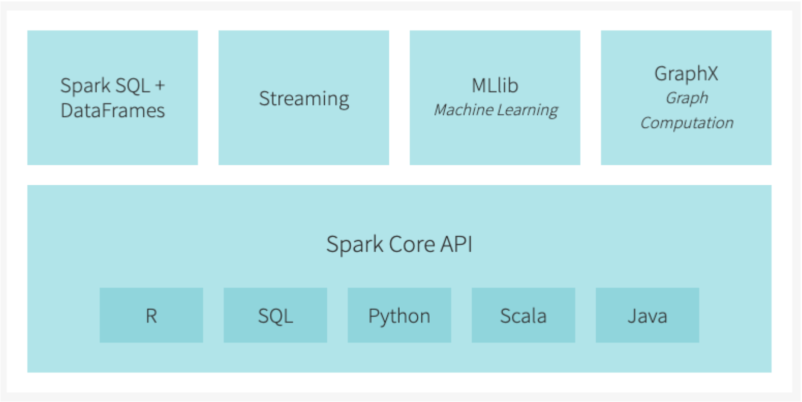
Spark's key features

- In-memory computation
- Fast and scalable
  - Efficiently scale up from one to many thousands of compute nodes
- Access data on a multitude of platforms
  - SQL and NoSQL databses
  - Cloud storage
  - Hadoop Distributed File System
- Real-time stream processing
- Libraries
  - Spark SQL
  - MLlib
  - Spark streaming
  - GraphX

What is `sparklyr`?

Package `sparklyr` provides an R interface for Spark. It works with any version of Spark.

- Use `dplyr` to translate R code into Spark SQL
- Work with Spark's MLlib
- Interact with a stream of data



The interface between R and Spark is young. If you know Scala, a great project would be to contribute to this R and Spark interaction by making Spark libraries available as an R package.

Connecting to Spark

Configure and connect

```r
library(tidyverse)
library(sparklyr)
```

```r
# add some custom configurations
conf <- list(
  sparklyr.cores.local = 4,
  `sparklyr.shell.driver-memory` = "16G",
  spark.memory.fraction = 0.5
)
```

```r
# create a spark connection
sc <- spark_connect(master = "local", version = "3.1", config = conf)
```

R functions and Spark

Distrubted R

We've seen that in our data wrangling we can use `dplyr`, some base R functions, `sparklyr` functions, and Hive functions. If none of these options are available for what you need, it is possible to apply an R function to a Spark DataFrame.

```
diamonds_tbl <- copy_to(sc, diamonds)
```

```
diamonds_tbl %>%
  select(carat, price) %>%
  scale()
```

```
Error: Unable to retrieve a spark_connection from object of class NULL
```

Last resort: `spark_apply()`

```
start <- Sys.time()
diamonds_tbl %>%
  select(carat, price) %>%
  spark_apply(function(x) scale(x))
end <- Sys.time()

end - start
```

```
Time difference of 2.62 mins
```

```
start <- Sys.time()
diamonds_tbl %>%
  select(carat, price) %>%
  mutate(carat = (carat - mean(carat, na.rm = TRUE)) / sd(carat, na.rm =
         price = as.double(price),
         price = (price - mean(price, na.rm = TRUE)) / sd(price, na.rm =
end <- Sys.time()

end - start
```

```
Time difference of 0.58 secs
```

Why so slow?

Since we are using an R function, the data is not processed by Spark.

What happens:

1. chunks of the data are moved from Spark to R
2. data is converted to an appropriate R format -- data.frame
3. the R function is applied
4. the results are converted back to a format for Spark and sent back to Spark

If you can, try to use `dplyr` or code Spark can understand.

Group DataFrame partitions

```r
diamonds_tbl %>%
  spark_apply(
    function(x) summary(lm(price ~ carat, x))$r.squared,
    names = "r.squared",
    group_by = "cut"
    )
```

```
# Source: spark<?> [?? x 2]
  cut       r.squared
  <chr>         <dbl>
1 Premium       0.856
2 Ideal         0.867
3 Good          0.851
4 Fair          0.738
5 Very Good     0.858
```

Check that this is correct.

ML Pipelines

What is an `ml_pipeline`?

Spark's ML Pipelines provide a way to easily combine multiple transformations and algorithms into a single workflow, or pipeline.

Some Spark terminology:

- **Transformer**: a transformer is an algorithm which can transform one DataFrame into another DataFrame
- **Estimator**: an estimator is an algorithm which can be fit on a DataFrame to produce a Transformer.
- **Pipeline**: a pipeline chains multiple Transformers and Estimators together to specify a machine learning workflow
- **Pipeline model**: a pipeline that has been trained on data so all of its components have been converted to transformers

Example: estimator

```
standardizer <- ft_standard_scaler(sc, input_col = "predictors",
                                   output_col = "predictors_standardized"
                                   with_mean = TRUE)
standardizer
```

```
StandardScaler (Estimator)
<standard_scaler__cad4bfc6_f41a_4bd4_bd6b_90cd54d4c071>
 (Parameters -- Column Names)
  input_col: predictors
  output_col: predictors_standardized
 (Parameters)
  with_mean: TRUE
  with_std: TRUE
```

Example: transformer

```
random_df <- copy_to(sc, data.frame(value = rpois(100000, 9))) %>%
  ft_vector_assembler(input_cols = "value", output_col = "predictors")
```

```
standardizer_algo <- ml_fit(standardizer, random_df)
standardizer_algo
```

```
StandardScalerModel (Transformer)
<standard_scaler__cad4bfc6_f41a_4bd4_bd6b_90cd54d4c071>
 (Parameters -- Column Names)
  input_col: predictors
  output_col: predictors_standardized
 (Transformer Info)
  mean:   num 9
  std:  num 3.01
```

Example: transformer

We can now feed the transformer some data. This could be our `random_df` or a new dataset (think train / test).

```
standardizer_algo %>%
  ml_transform(random_df) %>%
  glimpse()
```

```
Rows: ??
Columns: 3
Database: spark_connection
$ value                   <int> 6, 12, 9, 7, 5, 8, 13, 10, 12, …
$ predictors              <list> 6, 12, 9, 7, 5, 8, 13, 10, 12,…
$ predictors_standardized <list> -0.9982689, 0.9979628, -0.0001…
```

NC flights data

Let's create a ML pipeline to classify if a flight is delayed in February 2020 for all NC airports.

```
url <- str_c("http://www2.stat.duke.edu/~sms185/data/",
             "flights/nc_flights_feb_20.csv")

download.file(url = url, destfile = "data/nc_flights.csv")
```

```
nc_flights_tbl <- spark_read_csv(sc, name = "nc_flights",
                                 path = "data/nc_flights.csv")
```

Data is available from the Bureau of Transportation Statistics.

```r
df <- nc_flights_tbl %>%
  mutate(DEP_DELAY   = as.numeric(DEP_DELAY),
         ARR_DELAY   = as.numeric(ARR_DELAY),
         MONTH       = as.character(MONTH),
         DAY_OF_WEEK = as.character(DAY_OF_WEEK)
  ) %>%
  filter(!is.na(DEP_DELAY)) %>%
  select(DEP_DELAY, CRS_DEP_TIME, MONTH, DAY_OF_WEEK, DISTANCE)

df
```

```
# Source: spark<?> [?? x 5]
  DEP_DELAY CRS_DEP_TIME MONTH DAY_OF_WEEK DISTANCE
      <dbl>        <int> <chr> <chr>          <dbl>
1        -7          830 2     5                365
2         7          835 2     6                365
3        -8          830 2     7                365
4       -10          830 2     1                365
5        -3          830 2     2                365
# … with more rows
```

Pipeline

```r
nc_flights_pipe <- ml_pipeline(sc) %>%
  ft_dplyr_transformer(tbl = df) %>%
  ft_binarizer(input_col    = "DEP_DELAY",
               output_col   = "DELAYED",
               threshold    = 15) %>%
  ft_bucketizer(input_col  = "CRS_DEP_TIME",
                output_col = "HOURS",
                splits     = seq(0, 2400, 400)) %>%
  ft_r_formula(DELAYED ~ DAY_OF_WEEK + HOURS + DISTANCE) %>%
  ml_logistic_regression()
```

```
Pipeline (Estimator) with 5 stages
<pipeline_187aa28dcf960>
  Stages
  |--1 SQLTransformer (Transformer)
  |    <dplyr_transformer_187aaca3f397>
  |     (Parameters -- Column Names)
```

`ft_dplyr_transformer()` extracts the dplyr transformations used to generate object `tbl` as a SQL statement then passes it on to `ft_sql_transformer()`. The result is a `ml_pipeline` object.

```r
nc_flights_pipe <- ml_pipeline(sc) %>%
  ft_dplyr_transformer(tbl = df) %>%
  ft_binarizer(input_col    = "DEP_DELAY",
               output_col   = "DELAYED",
               threshold    = 15) %>%
  ft_bucketizer(input_col   = "CRS_DEP_TIME",
                output_col  = "HOURS",
                splits      = seq(0, 2400, 400)) %>%
  ft_r_formula(DELAYED ~ DAY_OF_WEEK + HOURS + DISTANCE) %>%
  ml_logistic_regression()
```

```
  |--2 Binarizer (Transformer)
  |    <binarizer_187aa5412bd32>
  |     (Parameters -- Column Names)
  |      input_col: DEP_DELAY
  |      output_col: DELAYED
```

```r
nc_flights_pipe <- ml_pipeline(sc) %>%
  ft_dplyr_transformer(tbl = df) %>%
  ft_binarizer(input_col    = "DEP_DELAY",
               output_col   = "DELAYED",
               threshold    = 15) %>%
  ft_bucketizer(input_col   = "CRS_DEP_TIME",
                output_col = "HOURS",
                splits      = seq(0, 2400, 400)) %>%
  ft_r_formula(DELAYED ~ DAY_OF_WEEK + HOURS + DISTANCE) %>%
  ml_logistic_regression()
```

```
  |--3 Bucketizer (Transformer)
  |    <bucketizer_187aa90f07cf>
  |     (Parameters -- Column Names)
  |      input_col: CRS_DEP_TIME
  |      output_col: HOURS
```

```
nc_flights_pipe <- ml_pipeline(sc) %>%
  ft_dplyr_transformer(tbl = df) %>%
  ft_binarizer(input_col    = "DEP_DELAY",
               output_col   = "DELAYED",
               threshold    = 15) %>%
  ft_bucketizer(input_col   = "CRS_DEP_TIME",
                output_col  = "HOURS",
                splits      = seq(0, 2400, 400)) %>%
  ft_r_formula(DELAYED ~ DAY_OF_WEEK + HOURS + DISTANCE) %>%
  ml_logistic_regression()
```

```
  |--4 RFormula (Estimator)
  |    <r_formula_187aa79a9bb9b>
  |     (Parameters -- Column Names)
  |      features_col: features
  |      label_col: label
  |     (Parameters)
  |      force_index_label: FALSE
  |      formula: DELAYED ~ DAY_OF_WEEK + HOURS + DISTANCE
  |      handle_invalid: error
  |      stringIndexerOrderType: frequencyDesc
```

```r
nc_flights_pipe <- ml_pipeline(sc) %>%
  ft_dplyr_transformer(tbl = df) %>%
  ft_binarizer(input_col    = "DEP_DELAY",
               output_col   = "DELAYED",
               threshold    = 15) %>%
  ft_bucketizer(input_col   = "CRS_DEP_TIME",
                output_col = "HOURS",
                splits     = seq(0, 2400, 400)) %>%
  ft_r_formula(DELAYED ~ DAY_OF_WEEK + HOURS + DISTANCE) %>%
  ml_logistic_regression()
```

```
  |--5 LogisticRegression (Estimator)
  |    <logistic_regression_187aa3ccd7a92>
  |     (Parameters -- Column Names)
  |      features_col: features
  |      label_col: label
  |      prediction_col: prediction
  |      probability_col: probability
  |      raw_prediction_col: rawPrediction
  |     (Parameters)
  |      aggregation_depth: 2
  |      elastic_net_param: 0
  |      family: auto
  |      fit_intercept: TRUE
  |      max_iter: 100
  |      reg_param: 0
  |      standardization: TRUE
  |      threshold: 0.5
  |      tol: 1e-06
```

Printed pipeline

```
Pipeline (Estimator) with 5 stages
<pipeline_187aa28dcf960>
  Stages
  |--1 SQLTransformer (Transformer)
  |    <dplyr_transformer_187aaca3f397>
  |     (Parameters -- Column Names)
  |--2 Binarizer (Transformer)
  |    <binarizer_187aa5412bd32>
  |     (Parameters -- Column Names)
  |      input_col: DEP_DELAY
  |      output_col: DELAYED
  |--3 Bucketizer (Transformer)
  |    <bucketizer_187aa90f07cf>
  |     (Parameters -- Column Names)
  |      input_col: CRS_DEP_TIME
  |      output_col: HOURS
  |--4 RFormula (Estimator)
  |    <r_formula_187aa79a9bb9b>
  |     (Parameters -- Column Names)
  |      features_col: features
  |      label_col: label
  |     (Parameters)
  |      force_index_label: FALSE
  |      formula: DELAYED ~ DAY_OF_WEEK + HOU
  |      handle_invalid: error
  |      stringIndexerOrderType: frequencyDes
```

```
  |--5 LogisticRegression (Estimator)
  |    <logistic_regression_187aa3ccd7a92>
  |     (Parameters -- Column Names)
  |      features_col: features
  |      label_col: label
  |      prediction_col: prediction
  |      probability_col: probability
  |      raw_prediction_col: rawPrediction
  |     (Parameters)
  |      aggregation_depth: 2
  |      elastic_net_param: 0
  |      family: auto
  |      fit_intercept: TRUE
  |      max_iter: 100
  |      reg_param: 0
  |      standardization: TRUE
  |      threshold: 0.5
  |      tol: 1e-06
```

What can we do with this pipeline?

1. Easily fit data with `ml_fit()`.
2. Make predictions with a fitted pipeline and `ml_transform()`.
3. Save pipelines that result in Scala scripts with `ml_save()` and can be read back into `sparklyr` (with `ml_load()`) or by the Scala or PySpark APIs.

Pipeline model

Partition the data into train and test sets.

```
nc_flights_partition <- nc_flights_tbl %>%
  sdf_random_split(training = 0.80, testing = 0.20)
```

Train the model

```
fitted_pipeline <- ml_fit(
  nc_flights_pipe,
  nc_flights_partition$training
)
```

Predictions

```
predictions <- ml_transform(
  fitted_pipeline,
  nc_flights_partition$training
)
```

```
sdf_crosstab(predictions, "label", "prediction") %>%
  arrange(label_prediction)
```

```
# Source:     spark<?> [?? x 3]
# Ordered by: label_prediction
  label_prediction `0.0` `1.0`
  <chr>            <dbl> <dbl>
1 0.0              32193   177
2 1.0               7244   211
```

Save pipeline objects

Save the pipeline:

```
ml_save(x = nc_flights_pipe, path = "nc_flights_pipeline")
```

Save the pipeline model (fitted pipeline with data):

```
ml_save(x = fitted_pipeline, path = "nc_flights_model")
```

The `ml_load()` command can be used to re-load these objects. You could then create a new pipeline model with new training data or make new predictions with the fitted pipeline model.

Exercise

Use `bike_tbl` to create an `ml_pipeline` object. Consider classification with member type as the response. Also, consider creating buckets for duration and a binary variable for round trips (bike starts and ends at the same location).

```
download.file(url = "http://www2.stat.duke.edu/~sms185/data/bike/cbs_2017
              destfile = "data/cbs_bike_2017.csv")
```

```
bike_tbl <- spark_read_csv(sc, path = "data/cbs_bike_2017.csv")
```

References

1. A Gentle Introduction to Apache Spark. (2021). http://www.dcs.bbk.ac.uk/~dell/teaching/cc/book/databricks/spark-intro.pdf.
2. Javier Luraschi, E. (2021). Mastering Spark with R. https://therinspark.com/.
3. OST_R | BTS | Transtats. (2020). Transtats.bts.gov. https://www.bts.gov
4. R Front End for Apache Spark. (2021). http://spark.apache.org/docs/latest/api/R/index.html.
5. sparklyr. (2021). https://spark.rstudio.com/.