Distributed Machine Learning in R with Apache Spark

An Introduction to the sparklyr and rsparkling packages

Brandon M. Greenwell

2018-08-01

Contents

Preface

This is a *sample* book written in **Markdown**. You can use anything that Pandoc's Markdown supports, e.g., a math equation $a^2 + b^2 = c^2$.

The **bookdown** package can be installed from CRAN or Github:

```
install.packages("bookdown")
# or the development version
# devtools::install_github("rstudio/bookdown")
```

Remember each Rmd file contains one and only one chapter, and a chapter is defined by the first-level heading `#`.

To compile this example to PDF, you need XeLaTeX. You are recommended to install TinyTeX (which includes XeLaTeX): https://yihui.name/tinytex/.

Part I

Part I

TBD.

Chapter 1

Introduction to Apache Spark

1.1 What is Spark?

- Apache Spark (https://spark.apache.org/) is a unified analytics engine and cluster computing framework for large-scale data processing.
- Does not use *MapReduce* as an execution engine. Rather, Spark uses its own distributed runtime to execute work on a cluster.
- It is designed to perform both batch processing (similar to MapReduce) and new workloads like streaming, interactive queries, and machine learning
- Spark is closely integrated with *Hadoop*, an open-source software framework for storing data and running applications on clusters.
  - Spark can run on *YARN*. This is the most convenient way to use Spark when you have an existing Hadoop cluster.
  - Spark is compatible with Hadoop data (e.g., Spark works with Hadoop file formats and storage backends like *parquet*).
  - To be clear, you do not need Hadoop to run Spark, but you'll need some form of shared file system in order to run Spark on a cluster.

1.2 Why use Spark?

- The most notable feature of Spark is its abaility to keep large working data sets in memory *between jobs*.
- There are two types of preccesses that benefit greatly from Sparks framework:
  - Iterative algorithms.
  - Interactive analysis.
- Spark uses a DAG (*directed acyclic graph*) engine which can process arbitrary pipelines of operators that can be translated into a single job for the user.
- Spark has a large community of experienced users.

- Spark provides APIs to three languages: Scala, Java, Python, and R (through the `SparkR` package). An R interface to Apache Spark is also provided by the `sparklyr` package (which we'll primarily be using in this book). Spark also has several built-in SQL functions (Spark SQL).

1.3 Spark DataFrames

A Spark DataFrame is an optimized columnar data structure similar in spirit to R/Pandas data frames. They can be constructed from a wide variety of different including: CSV files, JSON files, parquet files, Hive tables, external databases, data stored on HDFS, and many more. The DataFrame API is available in Scala, Java, Python, and R. The `sparklyr` package provides a `dplyr` backend that works seemlessly with Spark DataFrames. There is also the concept of Spark Datasets, but it will not be discussed in this book.

1.4 APIs to Spark

1.4.1 Scala

In a terminal, you can start a Spark session by typing

```
spark-shell
```

1.4.2 Java

1.4.3 Python

1.4.4 R

- The `sparklyr` package:
  - Allows you to connect to Spark from R.
  - Provides a complete `dplyr` backend.
  - Provdes a SQL interface via the `DBI` package.
  - Allows you to leverage Spark's distributed machine learning library (MLlib) from R.
  - Allows you to leverage H2O' machine learning library (SparkingWater).
  - Allows you to create extensions that call the full Spark API.
  - Provides interfaces to Spark packages.
  - For more information visit http://spark.rstudio.com/.
- The `sparkR` package:

1.4.5 Spark SQL

- You can use of Spark SQL to execute SQL queries.
- Spark SQL can also be used to read data from an existing *Hive* installation.

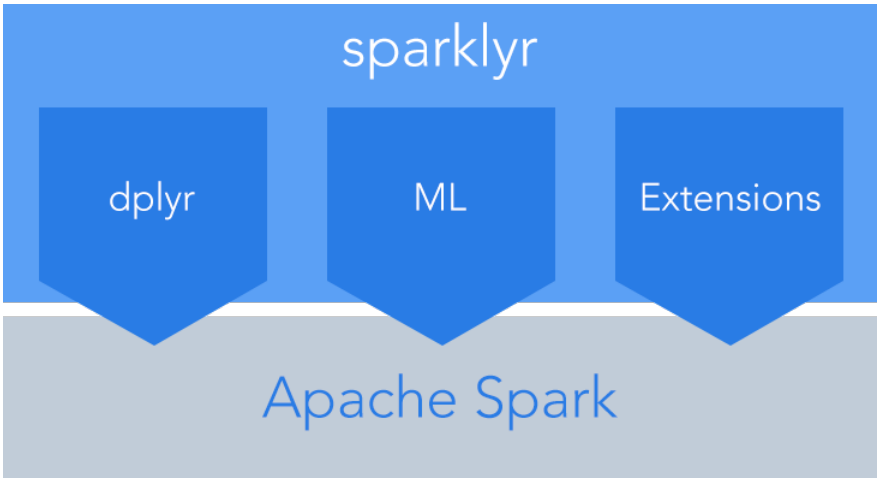

Figure 1.1: Source: http://spark.rstudio.com/tools/readme/sparklyr-illustration.png.

1.5 Installing Spark

The easiest way to install spark is to use the `sparklyr::spark_install()` function. For example, to install Spark version 2.3.0, use

```
sparklyr::spark_install(
  version = "2.3.0",
  hadoop_version = NULL
)
```

For details, see `?sparklyr::spark_install`. To see what Spark installations are available (along with there associated Hadoop versions), use

```
library(dplyr)
sparklyr::spark_available_versions(show_hadoop = TRUE) %>%
  arrange(desc(spark), desc(hadoop)) %>%
  slice(1:5)  # only look at five latest versions
```

```
##   spark hadoop
## 1 2.3.0    2.7
## 2 2.3.0    2.6
## 3 2.2.1    2.7
## 4 2.2.1    2.6
## 5 2.2.0    2.7
```

1.5.1 Linux

1.5.2 MacOS

1.5.3 Windows

- By a Unix machine .

1.6 Glossary of useful terms

1.7 Further reading

TBD.

Chapter 2

Interfacing R with Spark

TBD.

2.1 The sparklyr package

```
library(sparklyr)
sc <- spark_connect("local[4]")
class(sc)
```

```
## [1] "spark_connection"       "spark_shell_connection"
## [3] "DBIConnection"
```

```
pi_est <- function() {  # Monte-Carlo estimate of pi
  x <- runif(100000)
  y <- runif(100000)
  4 * mean(x^2 + y^2 < 1)
}

library(dplyr)

sdf_len(sc, length = 10, repartition = 10) %>%
  spark_apply(pi_est) %>%
  summarize(`Pi estimate` = mean(id, na.rm = TRUE))
```

```
## # Source:   lazy query [?? x 1]
## # Database: spark_connection
##   `Pi estimate`
##           <dbl>
## 1          3.14
```

2.2 Data wrangling in Spark with dplyr

2.3 The sparkR package

Chapter 3

Machine Learning Essentials

TBD.

Part II

Part II

TBD.

Chapter 4

Machine learning in Spark via MLlib

TBD.

Chapter 5

Machine learning in Spark via rsparkling

TBD.