Sparklyr Apply

Jim Harner

6/25/2018

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
## The following objects are masked from 'package:stats':
##
##     filter, lag
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(sparklyr)
sc <- spark_connect(master = "local")
```

6.4 Sparklyr Apply

`sparklyr` provides support to run arbitrary R code at scale within your Spark through the function `spark_apply`. Thus, any of R's functionality can be distributed across an R cluster. Apache Spark, even with Spark Packages, has a limited range of functions available.

`spark_apply` applies an R function to a Spark DataFrame, typically. Spark objects are partitioned so they can be distributed across a cluster. You can use `spark_apply` with the default partitions or you can define your own partitions with the `group_by` argument. Your R function must return another Spark DataFrame. `spark_apply` will run your R function on each partition and output a single Spark DataFrame.

6.4.1 Apply an R function to a Spark Object

Let's apply the identify function, `I()`, over a list of numbers we created with the `sdf_len` function.

```
sdf_len(sc, length = 5, repartition = 1) %>%
  spark_apply(function(e) I(e))
```

```
## # Source:   table<sparklyr_tmp_1ffcc08a5ea> [?? x 1]
## # Database: spark_connection
##      id
##   <dbl>
## 1    1.
## 2    2.
## 3    3.
## 4    4.
## 5    5.
```

6.4.2 Group By

A common task is to apply your R function to specific groups in your data, e.g., computing a regression model for each group. This is done by specifying a `group_by` argument.

The following example initially counts the number of rows in the iris data frame for each species.

```
iris_tbl <- sdf_copy_to(sc, iris)
iris_tbl %>%
  spark_apply(nrow, group_by = "Species")
```

```
## # Source:   table<sparklyr_tmp_1ffc1b7c24ea> [?? x 2]
## # Database: spark_connection
##   Species    Sepal_Length
##   <chr>             <int>
## 1 versicolor           50
## 2 virginica            50
## 3 setosa               50
```

Now compute the R^2 for a linear model for each species.

```
iris_tbl %>%
  spark_apply(
    function(e) summary(lm(Petal_Length ~ Petal_Width, e))$r.squared,
    names = "r.squared",
    group_by = "Species")
```

```
## # Source:   table<sparklyr_tmp_1ffc5282f57e> [?? x 2]
## # Database: spark_connection
##   Species    r.squared
##   <chr>          <dbl>
## 1 versicolor     0.619
## 2 virginica      0.104
## 3 setosa         0.110
```

6.4.3 Distributed Packages

With spark_apply you can use nearly any R package inside Spark.

As an example, we use the broom package to create a tidy data frame from the linear regression output.

```
spark_apply(
  iris_tbl,
  function(e) broom::tidy(lm(Petal_Length ~ Petal_Width, e)),
  names = c("term", "estimate", "std.error", "statistic", "p.value"),
  group_by = "Species")
```

```
## # Source:   table<sparklyr_tmp_1ffc55e17ec8> [?? x 6]
## # Database: spark_connection
##   Species    term        estimate std.error statistic  p.value
##   <chr>      <chr>          <dbl>     <dbl>     <dbl>    <dbl>
## 1 versicolor (Intercept)    1.78     0.284       6.28 9.48e- 8
## 2 versicolor Petal_Width    1.87     0.212       8.83 1.27e-11
## 3 virginica  (Intercept)    4.24     0.561       7.56 1.04e- 9
## 4 virginica  Petal_Width    0.647    0.275       2.36 2.25e- 2
## 5 setosa     (Intercept)    1.33     0.0600     22.1  7.68e-27
## 6 setosa     Petal_Width    0.546    0.224       2.44 1.86e- 2
```

The ability to use R packages in Spark is a killer feature, i.e., it expands the capability of Spark to most of the 13,000+ R packages. There are limitations, however. For example, referencing free variables using closures will not work.

```r
spark_disconnect(sc)
```