Ciencia de Datos en Spark

con sparklyr

Guía Rápida

RStudio

Cadena de herramientas en Ciencia de Datos con Spark + sparklyr

Importar
- Exportar en R DataFrame
- Leer un fichero
- Leer una tabla Hive

Ordenar
- verbos dplyr
- Acceso Spark SQL (DBI)
- Función SDF (Scala API)

Entender

Transformar
Función Transformar

Visualizar
Incluir datos en R para traficar

Manipular

Modelizar
- Spark MLib
- Extensión H2O

Comunicar
- Incluir datos en R
- Compartir gráficos, documentos y apps

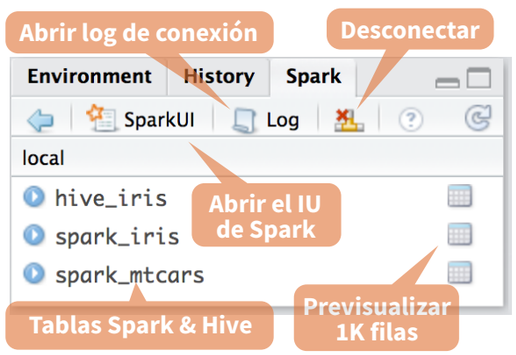
R for Data Science, Grolemund & Wickham

Intro

sparklyr es un interfaz de R a **Apache Spark™,** proporciona una integración con **dplyr** y la opción de lanzar búsquedas directamente usando sentencias **Spark SQL**. Con sparklyr, se puede orquestar aprendizaje automático de forma distribuida usando tanto **Spark's MLlib** como **H2O** Sparkling Water.

Empezando en la **versión 1.044, RStudio Desktop, Server y Pro incluyen un soporte integrado al paquete sparklyr**. Se pueden crear y gestionar conexiones a clusters Spark e instancias de Spark locales desde el IDE.

RStudio integrado con sparklyr



Getting started

Modo Local

Configuración fácil; no requiere cluster

1. Instalar en local una versión de Spark:
   spark_install ("2.0.1")
2. Abrir una conexión:
   sc <- spark_connect (master = "local")

En un cluster de Mesos

1. Instalar RStudio Server o Pro en uno de los nodos existentes
2. Localizar el path al directorio del cluster Spark
3. Abrir una conexión:
   spark_connect(master="[mesos URL]", version = "1.6.2", spark_home = [Cluster's Spark path])

Usando Livy (Experimental)

1. La aplicación Livy REST debe de estar corriendo en el cluster
2. Conectar al cluster:
   sc <- spark_connect(master = "http://host:port" , method = "livy")

En un cluster YARN

1. Instalar RStudio Server or RStudio Pro en uno de los nodos existentes, preferiblemente en un nodo frontera
2. Localizar el path al directorio Home de Spark en el cluster, normalmente es "/usr/lib/spark"
3. Abrir una conexión:
   spark_connect(master="yarn-client", version = "1.6.2", spark_home = [Cluster's Spark path])

En un cluster solo de Spark

1. Instalar RStudio Server o RStudio Pro en uno de los nodos existentes o un servido en la misma LAN
2. Instalar en local una versión de Spark:
   spark_install (version = "2.0.1")
3. Abrir una conexión:
   spark_connect(master="spark://host:port", version = "2.0.1", spark_home = spark_home_dir())

Despliegue del Cluster

Opciones para el despliegue del Cluster

Ajustando Spark

Configuración Ejemplo

```
config <- spark_config()
config$spark.executor.cores <- 2
config$spark.executor.memory <- "4G"
sc <- spark_connect (master = "yarn-client", config = config, version = "2.0.1")
```

Parámetros Importantes de Ajuste con valores por defecto

- spark.yarn.am.cores
- spark.yarn.am.memory *512m*

Parámetros Importantes de Ajuste con valores por defecto *continuación*

- spark.executor.heartbeatInterval *10s*
- spark.network.timeout *120s*
- spark.executor.memory *1g*
- spark.executor.cores *1*
- spark.executor.extraJavaOptions
- spark.executor.instances
- sparklyr.shell.executor-memory
- sparklyr.shell.driver-memory

Usando sparklyr

Un ejemplo breve de análisis de datos utilizando Apache Spark, R y sparklyr en local

```
library(sparklyr); library(dplyr); library(ggplot2);
library(tidyr);
set.seed(100)
spark_install("2.0.1")
sc <- spark_connect(master = "local")

import_iris <- copy_to(sc, iris, "spark_iris",
  overwrite = TRUE)

partition_iris <- sdf_partition(
  import_iris,training=0.5, testing=0.5)

sdf_register(partition_iris,
c("spark_iris_training","spark_iris_test"))

tidy_iris <- tbl(sc,"spark_iris_training") %>%
  select(Species, Petal_Length, Petal_Width)

model_iris <- tidy_iris %>%
  ml_decision_tree(response="Species",
  features=c("Petal_Length","Petal_Width"))

test_iris <- tbl(sc,"spark_iris_test")

pred_iris <- sdf_predict(
  model_iris,  test_iris) %>%
  collect

pred_iris %>%
  inner_join(data.frame(prediction=0:2,
  lab=model_iris$model.parameters$labels)) %>%
  ggplot(aes(Petal_Length, Petal_Width, col=lab)) +
  geom_point()

spark_disconnect(sc)
```

InstalaSpark en local

Conectar a versión local

Copia datos memoria de Spark

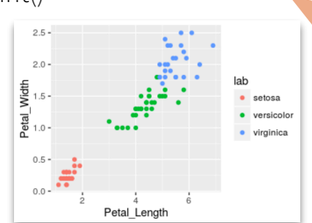
Partición data

Crea metados Hive para cada partición

Modelo árbol de decisión Spark ML

Crea referencia a tabla Spark

Trae datos de vuelta a memoria de R para crear gráfico



Crea gráfico con dplyr y pipes

Desconexión

Importar

Copiar un DataFrame en Spark

sdf_copy_to(sc, iris, "spark_iris")

sdf_copy_to(sc, x, name, memory, repartition, overwrite**)**

Importar en Spark desde fichero

Argumentos que aplican a todas las funciones:
sc, name, path, options = list(), repartition = 0, memory = TRUE, overwrite = TRUE

CSV **spark_read_csv(** header = TRUE, columns = NULL, infer_schema = TRUE, delimiter = ",", quote = "\"", escape = "\\", charset = "UTF-8", null_value = NULL**)**

JSON **spark_read_json()**

PARQUET **spark_read_parquet()**

Comandos Spark SQL

DBI::**dbWriteTable(**
sc, "spark_iris", iris**)**

DBI::**dbWriteTable(**conn, name, value**)**

Desde una table en Hive

my_var <- **tbl_cache**(sc, name= "hive_iris")

tbl_cache(sc, name, force = TRUE**)**
Carga la tabla en memoria

my_var <- **dplyr::tbl**(sc, name= "hive_iris")

dplyr::**tbl(**scr, …**)**
Crea una referencia a la tabla sin cargarla en memoria

Manipular

Spark SQL usand dplyr

Traduce a sentencias Spark SQL

```
my_table <- my_var %>%
  filter(Species=="setosa") %>%
  sample_n(10)
```

Comandos Directos Spark SQL

my_table <- DBI::**dbGetQuery**(sc , "SELECT * FROM iris LIMIT 10")

DBI::**dbGetQuery(**conn, statement**)**

API de Scala via funciones SDF

sdf_mutate(.data)
Funciona como la función mutate de dplyr

sdf_partition(x, ..., weights = NULL, seed = sample (.Machine$integer.max, 1)**)**

sdf_partition*(x, training = 0.5, test = 0.5)*

sdf_register(x, name = NULL**)**
Proporciona a un Spark Data Frame un table name

sdf_sample(x, fraction = 1, replacement = TRUE, seed = NULL**)**

sdf_sort(x, columns**)**
Ordena por >=1 columnas en orden ascendente

sdf_with_unique_id(x, id = "id"**)**
Añade IDs únicos a una columna

sdf_predict(object, newdata**)**
Spark DataFrame con valores predichos

Transformadores ML

ft_binarizer(my_table,input.col="Petal_Length", output.col="petal_large", threshold=1.2)

Argumentos que aplican a todas las funciones:
x, input.col = NULL, output.col = NULL

ft_binarizer(threshold = 0.5**)**
Asigna valores basados en un umbral

ft_bucketizer(splits**)**
Columna numérica a columna discreta

ft_discrete_cosine_transform(inverse = FALSE**)**
Dominio tiempo a dominio frecuencia

ft_elementwise_product(scaling.col**)**
Producto de 2 columnas elemento a elemento

ft_index_to_string()
Etiquetas de índices de vuelta a etiquetas como cadenas

ft_one_hot_encoder()
Vectores continuos a binarios

ft_quantile_discretizer(n.buckets = 5L**)**
Valores continuos a valores discretos categóricos

ft_sql_transformer(sql)

ft_string_indexer(params = NULL**)**
Columnas de etiquetas a columna de índices de etiquetas.

ft_vector_assembler()
Combina vectores a un vector de una sola fila

Visualizar & Comunicar

Descargar datos en memoria de R

r_table <- **collect**(my_table)
plot(Petal_Width~Petal_Length, data=r_table)

dplyr::**collect(**x**)**
Descarga un Spark DataFrame a un R DataFrame

sdf_read_column(x, column**)**
Devuelve el contenido de una sola columna a R

Salvar desde Spark al Sistema de Ficheros

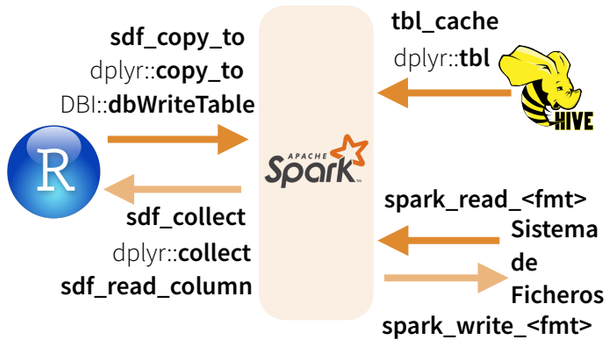
Argumentos que aplican a todas las funciones: x, path

CSV **spark_read_csv(** header = TRUE, delimiter = ",", quote = "\"", escape = "\\", charset = "UTF-8", null_value = NULL**)**

JSON **spark_read_json(**mode = NULL**)**

PARQUET **spark_read_parquet(**mode = NULL**)**

Lectura & Escritura desde Apache Spark



Extensiones

Crear un paquete de R que llame de forma completa al API de Spark & proporcione un interfaz a paquetes Spark

Tipos Core

spark_connection() Conexión entre R y el proceso shell de Spark

spark_jobj() Instancia de un objeto remoto de Spark

spark_dataframe() Instancia un objeto remoto Spark DataFrame

Llamada a Spark desde R

invoke() Llamada a un método en un objeto Java

invoke_new() Crea un nuevo objeto invocando a un constructor

invoke_static() Llamada a un método estático en un objeto

Extensiones Aprendizaje Automático

ml_create_dummy_variables()

ml_prepare_dataframe()

ml_prepare_response_features_intercept()

ml_options()

ml_model()

Modelizar (MLlib)

ml_decision_tree(my_table , response="Species", features= c("Petal_Length" , "Petal_Width"))

ml_als_factorization(x, rating.column = "rating", user.column = "user", item.column = "item", rank = 10L, regularization.parameter = 0.1, iter.max = 10L, ml.options = ml_options()**)**

ml_decision_tree(x, response, features, max.bins = 32L, max.depth = 5L, type = c("auto", "regression", "classification"), ml.options = ml_options()**)**
Same options for: **ml_gradient_boosted_trees**

ml_generalized_linear_regression(x, response, features, intercept = TRUE, family = gaussian(link = "identity"), iter.max = 100L, ml.options = ml_options()**)**

ml_kmeans(x, centers, iter.max = 100, features = dplyr::tbl_vars(x), compute.cost = TRUE, tolerance = 1e-04, ml.options = ml_options()**)**

ml_lda(x, features = dplyr::tbl_vars(x), k = length(features), alpha = (50/k) + 1, beta = 0.1 + 1, ml.options = ml_options()**)**

ml_linear_regression(x, response, features, intercept = TRUE, alpha = 0, lambda = 0, iter.max = 100L, ml.options = ml_options()**)**
Same options for: **ml_logistic_regression**

ml_multilayer_perceptron(x, response, features, layers, iter.max = 100, seed = sample(.Machine$integer.max, 1), ml.options = ml_options()**)**

ml_naive_bayes(x, response, features, lambda = 0, ml.options = ml_options()**)**

ml_one_vs_rest(x, classifier, response, features, ml.options = ml_options()**)**

ml_pca(x, features = dplyr::tbl_vars(x), ml.options = ml_options()**)**

ml_random_forest(x, response, features, max.bins = 32L, max.depth = 5L, num.trees = 20L, type = c("auto", "regression", "classification"), ml.options = ml_options()**)**

ml_survival_regression(x, response, features, intercept = TRUE,censor = "censor", iter.max = 100L, ml.options = ml_options()**)**

ml_binary_classification_eval(predicted_tbl_spark, label, score, metric = "areaUnderROC"**)**

ml_classification_eval(predicted_tbl_spark, label, predicted_lbl, metric = "f1"**)**

ml_tree_feature_importance(sc, model**)**

sparklyr es un interfaz R para Apache Spark