Spark數據科學之 sparklyr 速查表

Spark與sparklyr下的數據科學工具鏈

理解

導入
- 導出R DataFrame
- 讀取文件
- 讀取現有Hive表格

清理
- dplyr語句
- Direct Spark SQL (DBI)
- SDF fFnktion (Scala API)

數據整理

變換
變換函數

演示
在R裏綁定繪圖用數據

建模
- Spark MLlib
- H2O擴展包

交流
- 收集數據到R
- 分享繪圖，文件及應用

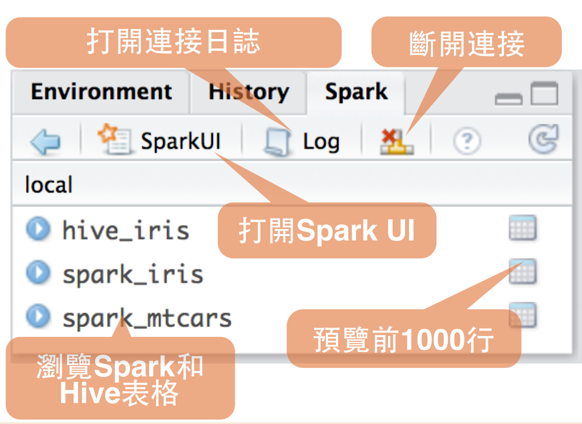
R for Data Science, Grolemund & Wickham

簡介

*sparklyr*是Apache Spark™與R接的口。它提供了一個完整的**dplyr**後端並且直接支持對**Spark SQL**語句的調用。它還能夠基於**Spark MLlib**或者**H2O** Sparkling Water實現分布式機器學習算法。

從版本**1.044**開始，**RStudio**軟件在**Desktop**，**Server**和**Pro**版本中集成了對**sparklyr**軟件包的支持。從IDE內我們可以直接創建或管理與Spark遠程集群或本地Spark的連接。

RStudio平臺中的sparklyr集成



如何開始

本地模式
輕松設置；無需群集

1. 安裝本地模式Spark：
   spark_install ("2.0.1")
2. 建立連接：
   sc <- spark_connect (master = "local")

使用Mesos集群管理模式

1. 在現有節點上安裝RStudio Server或Pro
2. 確定Spark Home的集群路徑
3. 建立集群連接：

   spark_connect(master="[Mesos-URL]", version = "1.6.2", spark_home = [Spark Home集群路徑])

使用Livy（測試階段）

1. Livy REST應用需已在集群上運行
2. 建立集群連接：

   sc <- spark_connect(master = "http://host:port" , method = "livy")

使用YARN集群管理模式

1. 在現有節點上安裝RStudio Server或Pro
2. 確定Spark Home集群路徑（通常位於"/usr/lib/spark"）
3. 建立集群連接：

   spark_connect(master="yarn-client", version = "1.6.2", spark_home = [Spark Home集群路徑])

使用Spark單例模式

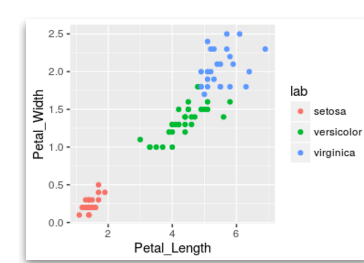
1. 在現有節點上或在同個局域網的服務器上安裝RStudio Server或Pro
2. 安裝本地模式Spark：
   spark_install (version = "2.0.1")
3. 建立連接：

   spark_connect(master="spark://host:port", version = "2.0.1", spark_home = spark_home_dir())

使用sparklyr

簡單示例：在本地模式下使用Apache Spark，R和sparklyr進行數據分析

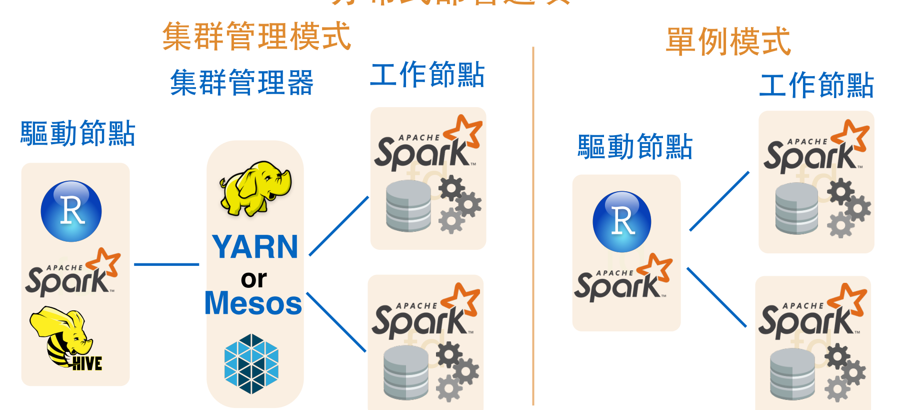
```
library(sparklyr); library(dplyr); library(ggplot2);
library(tidyr);
set.seed(100)
```
安裝本地模式Spark
```
spark_install("2.0.1")
```
連接本地Spark
```
sc <- spark_connect(master = "local")
```
```
import_iris <- copy_to(sc, iris, "spark_iris",
  overwrite = TRUE)
```
復制數據集到Spark
```
partition_iris <- sdf_partition(
  import_iris, training=0.5,
  testing=0.5)
```
分割Spark數據集
```
sdf_register(partition_iris,
c("spark_iris_training","spark_iris_test"))
```
為每個分割區註冊Hive表格
```
tidy_iris <- tbl(sc,"spark_iris_training") %>%
  select(Species, Petal_Length, Petal_Width)
```
創建Spark ML決策樹模型
```
model_iris <- tidy_iris %>%
  ml_decision_tree(response="Species",
  features=c("Petal_Length","Petal_Width"))
```
```
test_iris <- tbl(sc,"spark_iris_test")
```
建立Spark表格指針
```
pred_iris <- sdf_predict(
  model_iris, test_iris) %>%
  collect
```
讀取預測數據至R中用於之後演示之用
```
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)
```
斷開連接

分布式部署

分布式部署選項



Spark調試

配置示例

```
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")
```

調試參數
附默認值

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

調試參數
附默認值（續）

- 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

更多相關內容：spark.rstudio.com • 軟件包版本：0.5 • 更新於：2016-12-16

導入

復制DataFrame到Spark

sdf_copy_to(sc, iris, "spark_iris")

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

Spark SQL指令

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

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

在Spark中讀取文件

適用所有函數的參數：
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()**

讀取Hive表格

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

tbl_cache(sc, name, force = TRUE)
將整個表導入內存

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

dplyr::**tbl**(scr, …)
創建指向表的延遲加載指針

可視化與交流

將數據載入R內存

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

dplyr::**collect(**x**)**
將Spark DataFrame轉為R DataFrame

sdf_read_column(x, column**)**
返回R單個列下的內容

從Spark儲存數據到文件系統

適用所有函數的參數： 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**)**

建模（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()**)**
相同選項也適用於： **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()**)**
相同選項也適用於： **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**)**

數據整理

通過dplyr語句使用Spark SQL

轉換為Spark SQL語句

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

Direct Spark SQL指令

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

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

通過SDF函數使用Scala API

sdf_mutate(.data**)**
工作原理於dplyr mutate函數類似

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**)**
給Spark DataFrame命名

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

sdf_sort(x, columns**)**
按升序排列1個或多個數據列

sdf_with_unique_id(x, id = "id"**)**
添加唯一ID列

sdf_predict(object, newdata**)**
創建含有預測值的Spark DataFrame

ML轉換器

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

適用所有函數的參數：
x, input.col = NULL, output.col = NULL

ft_binarizer(threshold = 0.5**)**
基於閾值把數值列二元化

ft_bucketizer(splits**)**
把數字列轉為離散列

ft_discrete_cosine_transform(inverse = FALSE**)**
把時域轉為頻域

ft_elementwise_product(scaling.col**)**
計算二個數列的元素乘積

ft_index_to_string()
把索引列轉為字符串列

ft_one_hot_encoder()
把連續值轉為二進制向量

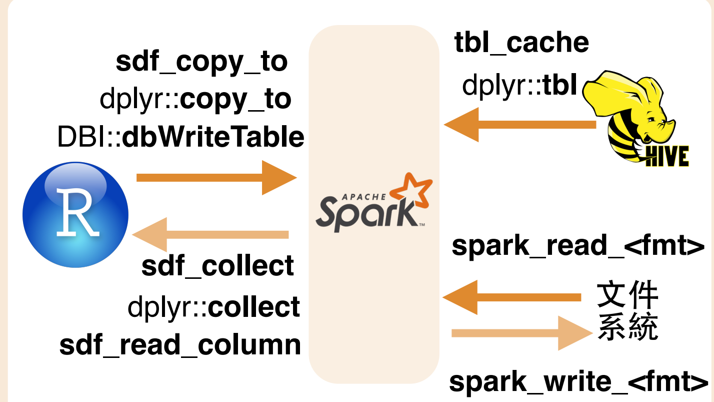
ft_quantile_discretizer(n.buckets = 5L**)**
把連續值轉為二進制向量

ft_sql_transformer(sql**)**

ft_string_indexer(params = NULL**)**
把字符串列轉為索引列

ft_vector_assembler()
合並向量為單行向量

從Apache Spark讀入或寫出



擴展包

創建一個R軟件包，用於調用完整的Spark API
並為Spark軟件包提供接口

核心類型

spark_connection() R與Spark Shell進程之間的連接對象

spark_jobj() 遠程Spark對象實例

spark_dataframe() 遠程Spark DataFrame對象實例

在R中調用Spark

invoke() 調用Java對象函數

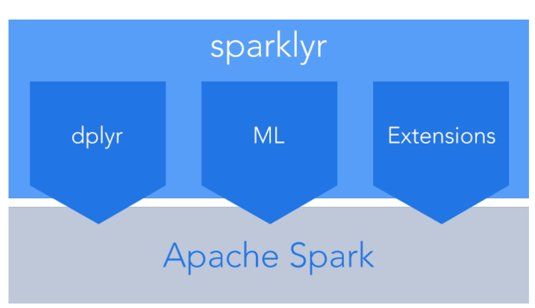
invoke_new() 通過構造函數創建一個新對象

invoke_static() 調用靜態函數

機器學習擴展包

ml_create_dummy_variables() **ml_options()**
ml_prepare_dataframe() **ml_model()**
ml_prepare_response_features_intercept()

sparklyr **是實現** Spark **與R的接口**



更多相關內容：spark.rstudio.com • 軟件包版本：0.5 • 更新於：2016-12-16