Spark数据科学之 sparklyr 速查表

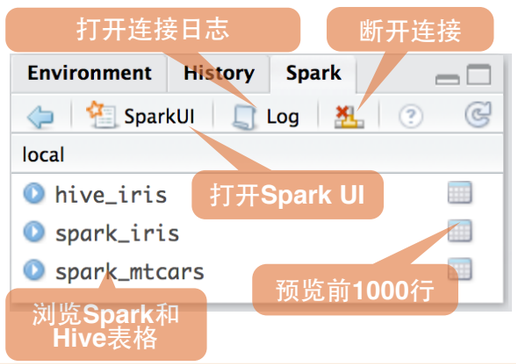
RStudio

简介

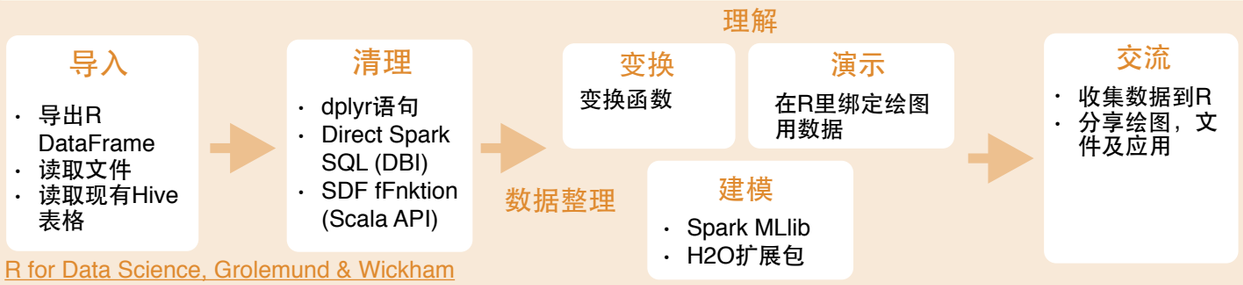
sparklyr是Apache Spark™与R接的口。它提供了一个完整的**dplyr**后端并且直接支持对**Spark SQL**语句的调用。它还能够基于**Spark MLlib**或者**H2O** Sparkling Water实现分布式机器学习算法。

从版本**1.044**开始，**RStudio**软件在**Desktop**，**Server**和**Pro**版本中集成了对**sparklyr**软件包的支持。从IDE内我们可以直接创建或管理与Spark远程集群或本地Spark的连接。

RStudio平台中的sparklyr集成



Spark与sparklyr下的数据科学工具链



R for Data Science, Grolemund & Wickham

如何开始

本地模式

轻松设置；无需群集

1. 安装本地模式Spark：
   spark_install ("2.0.1")
2. 建立连接：
   sc <- spark_connect (master = "local")

使用Mesos集群管理模式

1. 在现有节点上安装RStudio Server或Pro
2. 确定Spark Home的集群路径
3. 建立集群连接：

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

使用Livy（测试阶段）

1. Livy REST应用需已在集群上运行
2. 建立集群连接：

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

使用YARN集群管理模式

1. 在现有节点上安装RStudio Server或Pro
2. 确定Spark Home集群路径（通常位于"/usr/lib/spark"）
3. 建立集群连接：

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

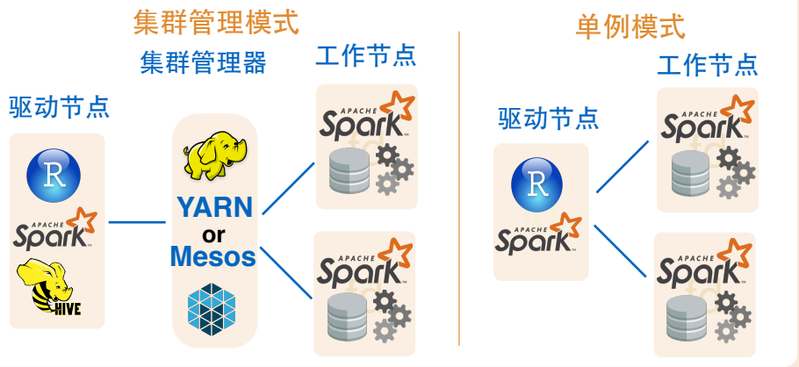
使用Spark单例模式

1. 在现有节点上或在同个局域网的服务器上安装RStudio Server或Pro
2. 安装本地模式Spark：
   spark_install (version = "2.0.1")
3. 建立连接：

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

分布式部署

分布式部署选项



Spark调试

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")
```

调试参数
附默认值

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

调试参数
附默认值（续）

- 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

使用sparklyr

简单示例：在本地模式下使用Apache Spark，R和sparklyr进行数据分析

```
library(sparklyr); library(dplyr); library(ggplot2);
library(tidyr);
set.seed(100)
```

安装本地模式Spark

```
spark_install("2.0.1")
```

连接本地Spark

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

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

复制数据集到Spark

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

分割Spark数据集

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

为每个分割区注册Hive表格

```
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"))
```

创建Spark ML决策树模型

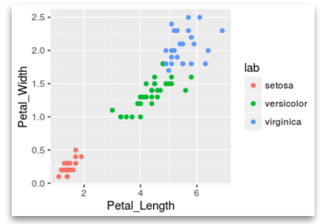
```
test_iris <- tbl(sc,"spark_iris_test")
```

建立Spark表格指针

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

读取预测数据至R中用于之后演示之用

```
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)
```

断开连接

更多相关内容：spark.rstudio.com • 软件包版本：0.5 • 更新于：2016-12-16
翻译：张克 • ke@2cn.de

导入

复制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中读取文件

适用所有函数的参数：
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()

读取Hive表格

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

tbl_cache(sc, name, force = TRUE)
将整个表导入内存

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

dplyr::tbl(scr, …)
创建指向表的延迟加载指针

数据整理

通过dplyr语句使用Spark SQL

转换为Spark SQL语句

```
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)

通过SDF函数使用Scala API

sdf_mutate(.data)
工作原理于dplyr mutate函数类似

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)
给Spark DataFrame命名

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

sdf_sort(x, columns)
按升序排列1个或多个数据列

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

sdf_predict(object, newdata)
创建含有预测值的Spark DataFrame

ML转换器

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

适用于所有函数的参数：
x, input.col = NULL, output.col = NULL

ft_binarizer(threshold = 0.5)
基于阈值把数值列二元化

ft_bucketizer(splits)
把数字列转为离散列

ft_discrete_cosine_transform(inverse = FALSE)
把时域转为频域

ft_elementwise_product(scaling.col)
计算二个数列的元素乘积

ft_index_to_string()
把索引列转为字符串列

ft_one_hot_encoder()
把连续值转为二进制向量

ft_quantile_discretizer(n.buckets = 5L)
把连续值转为分箱分类值

ft_sql_transformer(sql)

ft_string_indexer(params = NULL)
把字符串列转为索引列

ft_vector_assembler()
合并向量为单行向量

可视化与交流

将数据载入R内存

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

dplyr::collect(x)
将Spark DataFrame转为R DataFrame

sdf_read_column(x, column)
返回R单个列下的内容

从Spark储存数据到文件系统

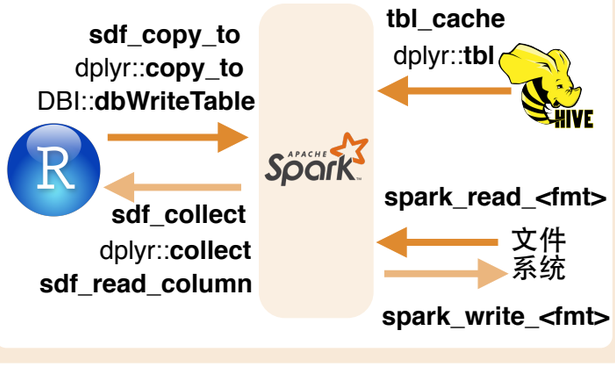
适用所有函数的参数：x, path

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)

从Apache Spark读入或写出



扩展包

创建一个R软件包，用于调用完整的Spark API
并为Spark软件包提供接口

核心类型

spark_connection() R与Spark Shell进程之间的连接对象

spark_jobj() 远程Spark对象实例

spark_dataframe() 远程Spark DataFrame对象实例

在R中调用Spark

invoke() 调用Java对象函数

invoke_new() 通过构造函数创建一个新对象

invoke_static() 调用静态函数

机器学习扩展包

ml_create_dummy_variables()

ml_prepare_dataframe()

ml_prepare_response_features_intercept()

ml_options()

ml_model()

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())
相同选项也适用于：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())
相同选项也适用于：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())

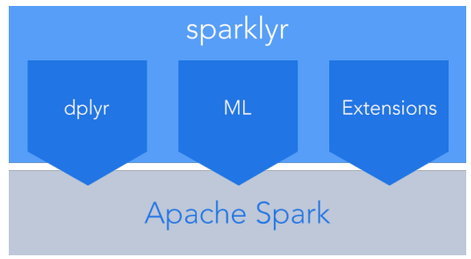
ml_survival_regression(x, response, features, intercept = TRUE,censor = "censor", iter.max = 100L, ml.options = ml_options())

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 是实现 Spark 与R的接口



翻译：张克 • ke@2cn.de

更多相关内容：spark.rstudio.com • 软件包版本：0.5 • 更新于：2016-12-16