Data Science in Spark

with sparklyr

Cheat Sheet

RStudio

Spark + sparklyrを用いたデータサイエンス ツールチェーン

理解

インポート
- Rデータフレームをエクスポート
- ファイル読み込み
- 既存のHiveテーブルの読み込み

整形
- dplyrの手法
- Spark SQLを直接操作(DBI)
- SDF関数(Scala API)

加工

変形
変形関数

可視化
プロット用にRへデータを整形

モデリング
- Spark MLib
- H_2Oエクステンション

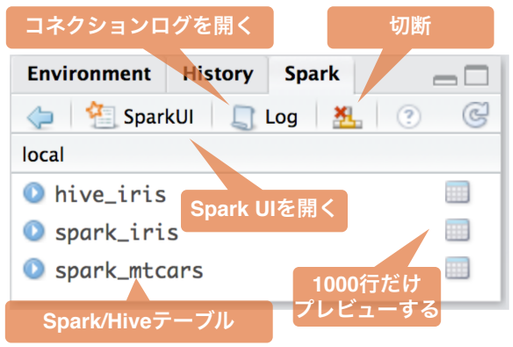
コミュニケーション
- Rへデータを収集
- プロット、ドキュメント、アプリの共有

R for Data Science, Grolemund & Wickham

イントロ

sparklyrとはApache Spark™へのインタフェースです. これはdplyrの完全なバックエンド機能や, Spark SQLステートメントを用いたクエリの直接発行もオプションとして提供します. sparklyrによってSpark MLibやH2O Sparkling Waterといった分散機械学習をオーケストレーションする事ができます.
RStudioデスクトップ版, サーバ版, Pro版のversion 1.044からsparklyrパッケージへの統合サポートが開始されます. これによってSparkクラスタやローカルのSparkインスタンスの作成や接続管理をIDE内で行うことができます.

sparklyrとRStudioの統合



事始め

ローカルモデル

クラスター不要で容易にセットアップが可能

1. Sparkのローカル版をインストール
   spark_install("2.0.1")
2. コネクションを開く
   sc <- spark_connect(master = "local")

Mesos管理クラスタ

1. RStudio ServerまたはProを既存のノードへインストール
2. クラスタのSparkディレクトリを指定する
3. コネクションを開く

```
spark_connect(master="[mesos URL]",
version = "1.6.2", spark_home =
[Cluster's Spark path])
```

Liby使った場合(実験的)

1. Livy RESTアプリケーションをクラスタ上で起動させておく
2. コネクションを開く

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

YARN管理クラスタ

1. RStudio ServerまたはProを既存のノードへインストールし, 可能であればエッジノードへもインストールする.
2. クラスタのSparkディレクトリを指定する. 通常、パスは次の場所である. "/usr/lib/spark"
3. コネクションを開く

```
spark_connect(master="yarn-client",
version = "1.6.2", spark_home = [Cluster's
Spark path])
```

Sparkスタンドアローンクラスタ

1. RStudio ServerまたはProを既存のノード, または同じLAN上のサーバへインストールする
2. ローカルバージョンのSparkを次のコマンドでインストールする

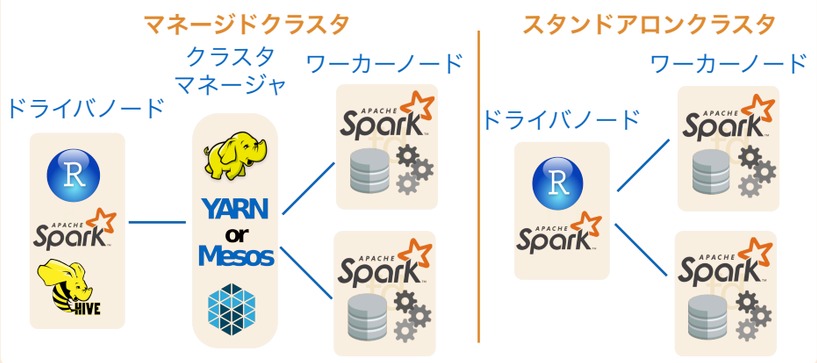
```
spark_install(version = "2.0.1")
```

3. コネクションを開く

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

クラスタを展開

クラスタの展開オプション



チューニング

設定例

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

重要なチューニングパラメータ

オレンジ文字はデフォルト値

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

重要なチューニングパラメータ

続き

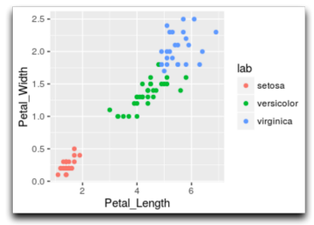
- spark.executor.heartbeatinterval *10s*
- spark.network.timeout *120s*
- spark.executor.memory *1g*
- spark.executor.cores *1*
- spark.executor.extraJavaOptions
- spark.executor.instances
- sparklyr.shell.executor-memory
- sparklyr.shell.driver-memory

sparklyrの活用

Apache Spark, R, sparklyrをローカルモードで用いてデータ分析を行う例

```
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)
```

Sparkをローカルにインストール
ローカルのSparkに接続
データをSparkメモリへコピー
データのパーティション設定
各パーティションにhiveメタデータを作成
Spark ML 決定木モデル
Sparkテーブルへ参照を作成
プロットするためRのメモリへデータを戻す
切断



Learn more at spark.rstudio.com • package version 0.5 • Updated: 12/21/16

Translated by Masato Takahashi // @masato_takahash

インポート

Sparkへデータをコピー

```
sfd_copy_to(sc, iris, "spark_iris")
sdf_copy_to(sc,x,name,memory,repartition,
overwrite)
```

Spark SQLコマンド

```
DBI::dbWriteTable(
  sc, "spark_iris", iris)
DBI::dbWriteTable(conn,name,
value)
```

ファイルからSparkへインポート

全関数共通の引数:
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()`

Hiveテーブルの操作

```
my_var <- tbl_cache(sc,
  name="hive_iris")
tbl_cache(sc,name,force=TRUE)
```
メモリにテーブルを読み込む

```
my_var <- dplyr::tbl(sc,
  name="hive_iris")
dplyr::tbl(src,…)
```
メモリに読み込まずに
テーブルへの参照を作成する

加工

dplyrの文法でSpark SQLを使用

Spark SQLへ翻訳されます

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

Spark SQLコマンドの直接発行

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

SDF関数を用いたScala APIの使用

sdf_mutate(.data)
dplyrのmutate関数と等価

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)
Spark DataFrameに名前を付ける

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

sdf_sort(x, columns)
1つ以上のカラムについて昇順にソート

sdf_with_unique_id(x, id="id")
ユニークIDカラムを追加

sdf_predict(object, newdata)
予測値を含むSpark DataFrame

ML 変換関数

```
ft_binarizer(my_table,input.col=
"Petal_Length",output.col="petal
_large",threshold=1.2)
```

全関数共通の引数:
sc, name, path, options=list(), repartition=0, memory=TRUE, overwrite=TRUE

ft_binarizer(threshold = 0.5)
閾値に基いて値を割り当て

ft_bucketizer(splits)
数値カラムを離散値カラムへ

ft_discrete_cosine_transform(inverse = FALSE)
時間領域から周波数領域へ変換
(離散コサイン変換)

ft_elementwise_product(scaling.col)
要素ごとの積を求める

ft_index_to_string()
インデックスを文字列に変換

ft_one_hot_encoder()
ラベルインデックスのカラムをバイナリベクトルのカラムに変換

ft_quantile_discretizer(n.buckets = 5L)
連続値からビン幅毎のカテゴリ値へ変換

ft_sql_transformer(sql)
ft_string_indexer(params = NULL)
ラベルカラムをラベルインデックスのカラムに変換

ft_vector_assembler()
複数のベクトルを1つに連結する

可視化 & 通信

Rのメモリへデータをダウンロード

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

dplyr::collect(x)
R DataFrameにSpark DataFrameをダウンロードする

sdf_read_column(x, column)
Rへ1カラムの要素を返す

Sparkからファイルシステムへ保存

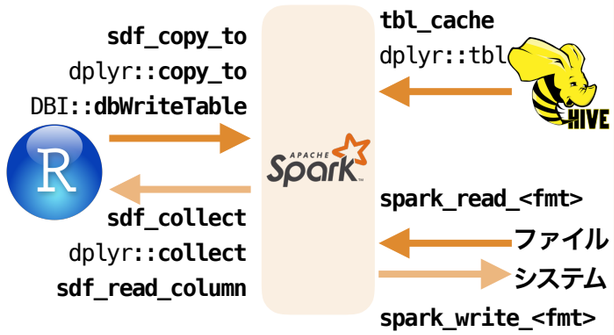
全関数へ適用される引数: x, path

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)`

Apache Sparkからの読み書き



エクステンション

全Spark APIを呼び出しSparkパッケージへのインタフェースを提供するRパッケージを作成

ファイルからSparkへインポート

spark_connection() RとSparkシェルプロセスとの接続
spark_jobj() リモートのSparkオブジェクトへのインタフェース
spark_dataframe() リモートのSpark DataFrameオブジェクトへのインタフェース

RからSparkの呼び出し

invoke() Javaオブジェクトのメソッドを呼び出す
invoke_new() コンストラクタを呼び出して新しいオブジェクトを生成する
invoke_static() オブジェクトの静的メソッドを呼び出す

機械学習エクステンション

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

モデリング(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())
ml_gradient_boosted_treesと同じオプション

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())
ml_logistic_regressionと同じオプション

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())

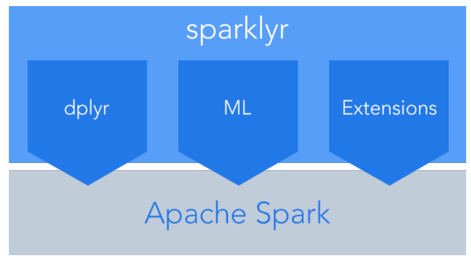
ml_survival_regression(x, response, features, intercept = TRUE,censor = "censor", iter.max = 100L, ml.options = ml_options())

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
Apache Spark
とRとの
インタフェース



Translated by Masato Takahashi // @masato_takahash