Hector Alvaro Rojas *| Data Science, Visualizations and Applied Statistics | May 31, 2020*

Big Data: MLearning Classifiers with Spark, Sparklyr, and H2O-Sparkling Water

WHAT IS THIS ALL ABOUT?

H2O is an open source, distributed machine learning platform for Everyone. H2O is an alternative open-source cross-platform for machine learning that supports the most widely used statistical & machine learning algorithms including gradient boosted machines, generalized linear models, deep learning and more.

The rsparkling extension package lets us access H2O's distributed machine learning functions via sparklyr. It provides bindings to H2O's distributed machine learning algorithms via sparklyr. In particular, rsparkling allows you to access the machine learning routines provided by the Sparkling Water Spark package.

This project is about an application of some common supervised ML classifiers in Apache Spark using Sparklyr and H2O-Sparkling Water. The main results obtained are presented here, providing a clear guideline that identifies the classic steps in the modeling process, possible to adapt to any other equivalent database. The key sparklyr and H2O-Sparkling Water functions used are identified, allowing the reader to consult the appropriate bibliography and tutorial examples clearly and directly.

This project has been developed using sparklyr considering a local Spark cluster environment. The excellent online publications from the University of Chicago "MACS 305001 - Computing for the Social Sciences" in its derived chapter "Spark and sparklyr" and the great "Documentation About Sparkling Water" are both used as bibliographic source of support and consultation base. Some of the scripts they used have been adequated or modified a little bit, meantime others not.

DATA SOURCE

In this project I use the Bank Marketing Data Set from UCI Machine Learning Repository. You can download the dataset here. The classification goal is to predict if the client will subscribe a term deposit (variable y).

The data is packed in zip format. They consist of two files bank-full.csv and bank.csv. In this project, we work with the file "bank.csv".

Note that, the dataset is not significant and you may think that the computation takes a long time. Spark is designed to process a considerable amount of data. Spark's performances increase relative to other machine learning libraries when the dataset processed grows larger.

LOADING THE DATA

```
# Clean memory and remove all fies
rm(list=ls())

# set working directory
setwd("mypath")

# get current working directory
getwd()

# Load required minimum packages
library(rsparkling)
library(sparklyr)
library(h2o)
library(tidyverse) # to get the whole tidyverse: dplyr, ggplot2 tibble, readr,
tidyr, purrr

# Download the zipped folder
download.file("https://archive.ics.uci.edu/ml/machine-learning-
databases/00222/bank.zip", "data/bank.zip")

# unzip the folder
unzip(zipfile = "data/bank.zip",
      exdir = "data")

df_raw <- read_delim("data/bank.csv", delim = ";") %>%
  map_if(is.character, as.factor) %>%
  as_tibble()

glimpse(df_raw)
Observations: 4,521
Variables: 17
$ age        30, 33, 35, 30, 59, 35, 36, 39, 41, 43, 39, 43, 36, 20, 3...
$ job        unemployed, services, management, management, blue-collar...
$ marital    married, married, single, married, married, single, marri...
$ education  primary, secondary, tertiary, tertiary, secondary, tertia...
$ default    no, no, no, no, no, no, no, no, no, no, no, no, no, no, n...
$ balance    1787, 4789, 1350, 1476, 0, 747, 307, 147, 221, -88, 9374,...
$ housing    no, yes, yes, yes, yes, no, yes, yes, yes, yes, yes, yes,...
$ loan       no, yes, no, yes, no, no, no, no, no, yes, no, no, no, no...
$ contact    cellular, cellular, cellular, unknown, unknown, cellular,...
$ day        19, 11, 16, 3, 5, 23, 14, 6, 14, 17, 20, 17, 13, 30, 29, ...
$ month      oct, may, apr, jun, may, feb, may, may, may, apr, may, ap...
$ duration   79, 220, 185, 199, 226, 141, 341, 151, 57, 313, 273, 113,...
$ campaign   1, 1, 1, 4, 1, 2, 1, 2, 2, 1, 1, 2, 2, 1, 1, 2, 5, 1, 1, ...
$ pdays      -1, 339, 330, -1, -1, 176, 330, -1, -1, 147, -1, -1, -1, ...
$ previous   0, 4, 1, 0, 0, 3, 2, 0, 0, 2, 0, 0, 0, 0, 1, 0, 0, 2, 0, ...
$ poutcome   unknown, failure, failure, unknown, unknown, failure, oth...
$ y          no, no, no, no, no, no, no, no, no, no, no, no, no, yes, ...
# copy our R data frame onto spark
df <- copy_to(sc, df_raw, name = "df")
# to see tables in "sc"
src_tbls(sc)
Output:
[1] "df"
```

FEATURE ENGINEERING

BUCKETIZING

By bucketing, we can create categories out of numerical data. We transform the variable "age" by "age_new" defining the splits “0, 30, 40, 50, 70, 90”. We use "ft_bucketizer" by sparklyr to let’s as get the categories where we have defined our splits.

The final values and distribution for the new variable"age_new" is:

```
  age_new   label    n
1       0 -Inf-30  482
2       1   30-40 1808
3       2   40-50 1203
4       3   50-70  967
5       4  70-Inf   61
```

Once the redundant information has been removed from the dataset, its updated structure is as follows:

```
Output:
Observations: ??
Variables: 17
Database: spark_connection
$ job         "unemployed", "services", "management", "management", "bl...
$ marital     "married", "married", "single", "married", "married", "si...
$ education   "primary", "secondary", "tertiary", "tertiary", "secondar...
$ default     "no", "no", "no", "no", "no", "no", "no", "no", "no", "no...
$ balance     1787, 4789, 1350, 1476, 0, 747, 307, 147, 221, -88, 9374,...
$ housing     "no", "yes", "yes", "yes", "yes", "no", "yes", "yes", "ye...
$ loan        "no", "yes", "no", "yes", "no", "no", "no", "no", "no", "...
$ contact     "cellular", "cellular", "cellular", "unknown", "unknown",...
$ day         19, 11, 16, 3, 5, 23, 14, 6, 14, 17, 20, 17, 13, 30, 29, ...
$ month       "oct", "may", "apr", "jun", "may", "feb", "may", "may", "...
$ duration    79, 220, 185, 199, 226, 141, 341, 151, 57, 313, 273, 113,...
$ campaign    1, 1, 1, 4, 1, 2, 1, 2, 2, 1, 1, 2, 2, 1, 1, 2, 5, 1, 1, ...
$ pdays       -1, 339, 330, -1, -1, 176, 330, -1, -1, 147, -1, -1, -1, ...
$ previous    0, 4, 1, 0, 0, 3, 2, 0, 0, 2, 0, 0, 0, 0, 1, 0, 0, 2, 0, ...
$ poutcome    "unknown", "failure", "failure", "unknown", "unknown", "f...
$ y           "no", "no", "no", "no", "no", "no", "no", "no", "no", "no...
$ age         1, 1, 1, 1, 3, 1, 1, 1, 2, 2, 1, 2, 1, 0, 1, 2, 3, 1, 0, ...
```

RE_GROUPING QUALITATIVE FEATURES

No qualitative variable was regrouped since the categories presented enough frequency to develop the models.

FACTORS INTO INTEGERS

Now, we need to turn all the characters into integers, as this is how MLlib likes it. Further, the labels for a categorical outcomes have to be in {0,1}. We use "ft_string_indexer()" by sparklyr to let’s as get the numerical transformation. Finally, the gotten dataset (mydata) is copied to the Spark cluster to reflect a situation where we’re dealing with data already in Spark.

```
glimpse(mydata)

Output:
Observations: ??
Variables: 17
Database: spark_connection
$ y             0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0...
$ age           1, 1, 1, 1, 3, 1, 1, 1, 2, 2, 1, 2, 1, 0, 1, 2, 3, 1, 0...
```

```
$ balance      1787, 4789, 1350, 1476, 0, 747, 307, 147, 221, -88, 937...
$ day          19, 11, 16, 3, 5, 23, 14, 6, 14, 17, 20, 17, 13, 30, 29...
$ duration     79, 220, 185, 199, 226, 141, 341, 151, 57, 313, 273, 11...
$ campaign_i   0, 0, 0, 3, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 4, 0, 0...
$ pdays        -1, 339, 330, -1, -1, 176, 330, -1, -1, 147, -1, -1, -1...
$ previous_i   0, 4, 1, 0, 0, 3, 2, 0, 0, 2, 0, 0, 0, 0, 1, 0, 0, 2, 0...
$ job_i        8, 4, 0, 0, 1, 0, 6, 2, 7, 4, 4, 3, 2, 10, 1, 0, 2, 3, ...
$ marital_i    0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1...
$ education_i  2, 0, 1, 1, 0, 1, 1, 0, 1, 2, 0, 0, 1, 0, 0, 1, 0, 1, 2...
$ default_i    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
$ housing_i    1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0...
$ loan_i       0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0...
$ contact_i    0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1...
$ month_i      8, 0, 5, 3, 0, 6, 0, 0, 0, 5, 0, 5, 2, 5, 7, 2, 2, 5, 0...
$ poutcome_i   0, 1, 1, 0, 0, 1, 2, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0...
mydata_tbl <- mydata %>%
                 #as_tibble() %>%
  copy_to(sc, ., name = "mydata",overwrite = TRUE) # copy our R data frame onto
spark
# to see tables in "sc"
src_tbls(sc)
Output:
[1] "df"     "mydata"
```

GETTING THE TRAIN AND TEST DATASET

GETTING TRAIN AND TEST DATASETS

```
# split our data set into 'training', 'test'
h2o.init()
partitions <- h2o.splitFrame(as.h2o(mydata_tbl), 0.7, seed = 2020)
# Create table references
train <-partitions[[1]]
test <-partitions[[2]]
```

READJUSTMENT OF TRAIN AND TEST DATASETS

Now we have readjusted train and test datasets to be incorporated into the H2o format for machine learning models. We isolate the model’s dependent variable (y) and turn all the categorical features into numbers that represent factors.

```
y <- "y"
x <- setdiff(names(train), y)
train[,y] <- as.factor(train[,y])
train[,"job_i" ] <- as.factor(train[,"job_i" ])
train[,"marital_i" ] <- as.factor(train[,"marital_i" ])
train[,"education_i" ] <- as.factor(train[,"education_i" ])
train[,"default_i" ] <- as.factor(train[,"default_i" ])
train[,"housing_i" ] <- as.factor(train[,"housing_i" ])
train[,"loan_i" ] <- as.factor(train[,"loan_i" ])
train[,"contact_i" ] <- as.factor(train[,"contact_i" ])
train[,"month_i" ] <- as.factor(train[,"month_i" ])
train[,"poutcome_i" ] <- as.factor(train[,"poutcome_i" ])
test[,y] <- as.factor(test[,y])
test[,"job_i" ] <- as.factor(test[,"job_i" ])
test[,"marital_i" ] <- as.factor(test[,"marital_i" ])
test[,"education_i" ] <- as.factor(test[,"education_i" ])
test[,"default_i" ] <- as.factor(test[,"default_i" ])
```

```
test[,"housing_i" ] <- as.factor(test[,"housing_i" ])
test[,"loan_i" ] <- as.factor(test[,"loan_i" ])
test[,"contact_i" ] <- as.factor(test[,"contact_i" ])
test[,"month_i" ] <- as.factor(test[,"month_i" ])
test[,"poutcome_i" ] <- as.factor(test[,"poutcome_i" ])
```

MACHINE LEARNING MODELS

FITTING THE MODELS

Model: Logistic Regression (LR_model)

Build and train the model

```
LR_model <- h2o.glm(x = x,
                 y = y,
                 family = "binomial",
                 training_frame = train,
                 lambda_search = TRUE,
                 seed = 2020)
```

Model summary

```
LR_model@ model$ model_summary
GLM Model: summary
    family  link                                regularization
1 binomial logit Elastic Net (alpha = 0.5, lambda = 1.248E-4 )
                                                                 lambda_search
1 nlambda = 100, lambda.max = 0.2566, lambda.min = 1.248E-4, lambda.1se = -1.0
  number_of_predictors_total number_of_active_predictors number_of_iterations
1                         51                          46                  136
    training_frame
1 RTMP_sid_9a68_24
```

Evaluate performance on a test set

```
perf_test <- h2o.performance(LR_model, newdata = test)
perf_test
H2OBinomialMetrics: glm

MSE:  0.06334832
RMSE:  0.2516909
LogLoss:  0.2170386
Mean Per-Class Error:  0.1785608
AUC:  0.916097
AUCPR:  0.5326796
Gini:  0.832194
R^2:  0.3191915
Residual Deviance:  589.4768
AIC:  683.4768

Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
          0   1    Error       Rate
0      1119  98 0.080526   =98/1217
1        39 102 0.276596    =39/141
Totals 1158 200 0.100884  =137/1358

Maximum Metrics: Maximum metrics at their respective thresholds
                        metric threshold       value idx
1                       max f1  0.207712    0.598240 162
2                       max f2  0.138268    0.699052 206
```

```
3                 max f0point5  0.321153    0.597907 114
4                 max accuracy  0.322458    0.916789 113
5                max precision  0.999449    1.000000   0
6                   max recall  0.010593    1.000000 381
7              max specificity  0.999449    1.000000   0
8             max absolute_mcc  0.207712    0.553368 162
9   max min_per_class_accuracy  0.132347    0.851064 213
10 max mean_per_class_accuracy  0.124923    0.855644 219
11                     max tns  0.999449 1217.000000   0
12                     max fns  0.999449  140.000000   0
13                     max fps  0.000568 1217.000000 399
14                     max tps  0.010593  141.000000 381
15                     max tnr  0.999449    1.000000   0
16                     max fnr  0.999449    0.992908   0
17                     max fpr  0.000568    1.000000 399
18                     max tpr  0.010593    1.000000 381

Gains/Lift Table: Extract with `h2o.gainsLift(, )` or `h2o.gainsLift(, valid=,
xval=)`>
```

Model: Random Forest (RForest_model)

Build and train the model

```
RForest_model <- h2o.randomForest(x = x,
                             y = y,
                             training_frame = train,
                             nbins = 32,
                             max_depth = 5,
                             ntrees = 20,
                             seed = 2020)
```

Model summary

```
LR_model@ model$ model_summary
Model Summary:
  number_of_trees number_of_internal_trees model_size_in_bytes min_depth
1              20                       20                7715         5
  max_depth mean_depth min_leaves max_leaves mean_leaves
1         5    5.00000         20         30    26.05000
```

Evaluate performance on a test set

```
perf_test <- h2o.performance(RForest_model, newdata = test)
perf_test
H2OBinomialMetrics: drf

MSE:  0.06412041
RMSE:  0.2532201
LogLoss:  0.2232473
Mean Per-Class Error:  0.2186052
AUC:  0.9145731
AUCPR:  0.5764669
Gini:  0.8291462
R^2:  0.3108939

Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
          0   1    Error       Rate
0      1151  66 0.054232   =66/1217
1        54  87 0.382979    =54/141
Totals 1205 153 0.088365  =120/1358
```

```
Maximum Metrics: Maximum metrics at their respective thresholds
                        metric threshold       value idx
1                       max f1  0.284890    0.591837 122
2                       max f2  0.175376    0.682635 191
3                 max f0point5  0.406916    0.594966  63
4                 max accuracy  0.406916    0.918262  63
5                max precision  0.828950    1.000000   0
6                   max recall  0.042489    1.000000 358
7              max specificity  0.828950    1.000000   0
8             max absolute_mcc  0.284890    0.542952 122
9   max min_per_class_accuracy  0.139152    0.830731 220
10 max mean_per_class_accuracy  0.175376    0.839752 191
11                     max tns  0.828950 1217.000000   0
12                     max fns  0.828950  140.000000   0
13                     max fps  0.026066 1217.000000 399
14                     max tps  0.042489  141.000000 358
15                     max tnr  0.828950    1.000000   0
16                     max fnr  0.828950    0.992908   0
17                     max fpr  0.026066    1.000000 399
18                     max tpr  0.042489    1.000000 358

Gains/Lift Table: Extract with `h2o.gainsLift(, )` or `h2o.gainsLift(, valid=,
xval=)`>
```

Model: Gradient Boosting Machine (GBM_model)

Build and train the model

```
GBM_model <- h2o.gbm(x = x,
                     y = y,
                     training_frame = train,
                     ntrees = 20,
                     max_depth = 3,
                     learn_rate = 0.01,
                     col_sample_rate = 0.7,
                     seed = 2020)
```

Model summary

```
GBM_model@ model$ model_summary
Model Summary:
  number_of_trees number_of_internal_trees model_size_in_bytes min_depth
1              20                       20                3143         3
  max_depth mean_depth min_leaves max_leaves mean_leaves
1         3    3.00000          7          8     7.90000
```

Evaluate performance on a test set

```
perf_test <- h2o.performance(GBM_model_model, newdata = test)
perf_test
H2OBinomialMetrics: gbm

MSE:  0.0854276
RMSE:  0.29228
LogLoss:  0.3022196
Mean Per-Class Error:  0.2252866
AUC:  0.8891968
AUCPR:  0.5287879
Gini:  0.7783936
R^2:  0.08190414

Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
```

```
          0   1    Error       Rate
0      1152  65 0.053410   =65/1217
1        56  85 0.397163    =56/141
Totals 1208 150 0.089102  =121/1358

Maximum Metrics: Maximum metrics at their respective thresholds
                        metric threshold       value idx
1                       max f1  0.152677    0.584192  89
2                       max f2  0.140807    0.631313 109
3                 max f0point5  0.172130    0.590476  65
4                 max accuracy  0.172130    0.916789  65
5                max precision  0.237878    0.800000   3
6                   max recall  0.105602    1.000000 147
7              max specificity  0.246654    0.999178   0
8             max absolute_mcc  0.152677    0.534670  89
9   max min_per_class_accuracy  0.112943    0.792933 138
10 max mean_per_class_accuracy  0.136973    0.804702 119
11                     max tns  0.246654 1216.000000   0
12                     max fns  0.246654  141.000000   0
13                     max fps  0.105157 1217.000000 148
14                     max tps  0.105602  141.000000 147
15                     max tnr  0.246654    0.999178   0
16                     max fnr  0.246654    1.000000   0
17                     max fpr  0.105157    1.000000 148
18                     max tpr  0.105602    1.000000 147

Gains/Lift Table: Extract with `h2o.gainsLift(, )` or `h2o.gainsLift(, valid=,
xval=)`>
```

Model: Neural Networks (DL_model)

Build and train the model

```
DL_Model <- h2o.deeplearning(x = x, y = y,
                           training_frame = train,
                           epochs = 15,
                           activation = "Rectifier",
                           hidden = c(10, 5, 10),
                           input_dropout_ratio = 0.7,
                             seed = 2020)
```

Model summary

```
DL_Model@ model$ model_summary
Status of Neuron Layers: predicting y, 2-class classification, bernoulli
distribution, CrossEntropy loss, 747 weights/biases, 14,9 KB, 47.516 training
samples, mini-batch size 1
  layer units      type dropout       l1       l2 mean_rate rate_rms momentum
1     1    60     Input 70.00 %       NA       NA        NA       NA       NA
2     2    10 Rectifier  0.00 % 0.000000 0.000000  0.152447 0.370787 0.000000
3     3     5 Rectifier  0.00 % 0.000000 0.000000  0.001085 0.000795 0.000000
4     4    10 Rectifier  0.00 % 0.000000 0.000000  0.001007 0.000833 0.000000
5     5     2   Softmax      NA 0.000000 0.000000  0.001583 0.001760 0.000000
  mean_weight weight_rms mean_bias bias_rms
1          NA         NA        NA       NA
2    0.001186   0.174422  0.258896 0.167770
3    0.124316   0.369663  0.949611 0.083277
4    0.081648   0.392130  1.001566 0.018298
5   -0.740970   1.583561  0.006417 0.000452
```

Evaluate performance on a test set

```
perf_test <- h2o.performance(DL_Model, newdata = test)
perf_test
H2OBinomialMetrics: deeplearning

MSE:  0.07145651
RMSE:  0.2673135
LogLoss:  0.2378484
Mean Per-Class Error:  0.254413
AUC:  0.8779408
AUCPR:  0.4431717
Gini:  0.7558815

Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
          0   1    Error       Rate
0      1107 110 0.090386  =110/1217
1        59  82 0.418440    =59/141
Totals 1166 192 0.124448  =169/1358

Maximum Metrics: Maximum metrics at their respective thresholds
                        metric threshold       value idx
1                       max f1  0.231749    0.492492 140
2                       max f2  0.131472    0.627729 212
3                 max f0point5  0.341961    0.499139  91
4                 max accuracy  0.608495    0.906480  26
5                max precision  0.968073    1.000000   0
6                   max recall  0.019295    1.000000 358
7              max specificity  0.968073    1.000000   0
8             max absolute_mcc  0.131472    0.432195 212
9   max min_per_class_accuracy  0.131472    0.805259 212
10 max mean_per_class_accuracy  0.131472    0.810431 212
11                     max tns  0.968073 1217.000000   0
12                     max fns  0.968073  140.000000   0
13                     max fps  0.000364 1217.000000 399
14                     max tps  0.019295  141.000000 358
15                     max tnr  0.968073    1.000000   0
16                     max fnr  0.968073    0.992908   0
17                     max fpr  0.000364    1.000000 399
18                     max tpr  0.019295    1.000000 358

Gains/Lift Table: Extract with `h2o.gainsLift(, )` or `h2o.gainsLift(, valid=,
xval=)`>
```

Model: Naive Bayes (NB_model)

Build and train the model

```
NB_Model <- h2o.naiveBayes(x = x, y = y,
                          training_frame = train,
                               laplace = 0,
                          seed = 2020)
```

Model summary

```
NB_Model@ model$ model_summary
Model Summary:
  number_of_response_levels min_apriori_probability max_apriori_probability
1                         2                 0.12014                 0.87986
```

Evaluate performance on a test set

```
perf_test <- h2o.performance(NB_Model, newdata = test)
perf_test
```

```
H2OBinomialMetrics: naivebayes

MSE:  0.09213555
RMSE:  0.3035384
LogLoss:  0.4303034
Mean Per-Class Error:  0.2258285
AUC:  0.8679027
AUCPR:  0.3829924
Gini:  0.7358054

Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
          0   1    Error       Rate
0      1073 144 0.118324  =144/1217
1        47  94 0.333333    =47/141
Totals 1120 238 0.140648  =191/1358

Maximum Metrics: Maximum metrics at their respective thresholds
                        metric threshold       value idx
1                       max f1  0.275154    0.496042 167
2                       max f2  0.119273    0.631749 242
3                 max f0point5  0.614295    0.470756  89
4                 max accuracy  0.997865    0.897644   3
5                max precision  0.997865    0.545455   3
6                   max recall  0.000039    1.000000 399
7              max specificity  0.999964    0.993426   0
8             max absolute_mcc  0.155694    0.446703 219
9   max min_per_class_accuracy  0.128474    0.808511 235
10 max mean_per_class_accuracy  0.104429    0.814755 251
11                     max tns  0.999964 1209.000000   0
12                     max fns  0.999964  135.000000   0
13                     max fps  0.000039 1217.000000 399
14                     max tps  0.000039  141.000000 399
15                     max tnr  0.999964    0.993426   0
16                     max fnr  0.999964    0.957447   0
17                     max fpr  0.000039    1.000000 399
18                     max tpr  0.000039    1.000000 399

Gains/Lift Table: Extract with `h2o.gainsLift(, )` or `h2o.gainsLift(, valid=, xval=)`>
```

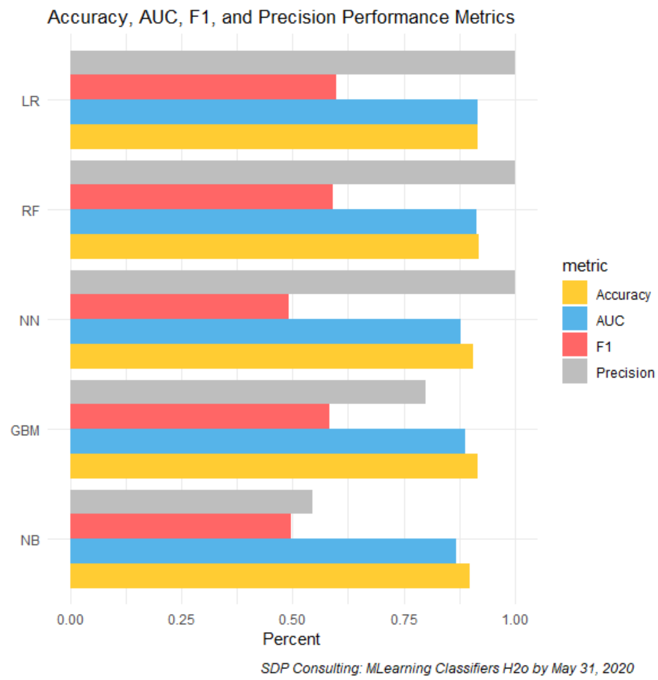
MODEL SELECTION, CHARACTERIZATION AND PREDICTION

Select the Best Model

The models are compared based on the metrics obtained in each case. These are: AUC, F1 (max f1), Accuracy (max accuracy), and Precision (max precision).

These metrics are arranged in a table and then arranged in a possible format to process with ggplot2, generating the respective comparative visualization.

	Model	AUC	F1	Accuracy	Precision
1	LR	0.9160970	0.5982405	0.9167894	1.0000000
2	RF	0.9145731	0.5918367	0.9182622	1.0000000
3	GBM	0.8891968	0.5841924	0.9167894	0.8000000
4	NN	0.8779408	0.4924925	0.9064801	1.0000000
5	NB	0.8679027	0.4960422	0.8976436	0.5454545



The Logistic Regression (LR) and Random Forest (RF) algorithms performed the best and had comparatively better results in each of the metrics used to compare all models: AUC, F1, Accuracy, and Precision. Finally, any of them can be considered as the best model. Among both, finally, I choose the Random Forest as the best model.

Characterization of the best model

characterization of the best model can be obtained using the print command as follows.

```
print(RForest_model)
Output:
Model Details:
==============

H2OBinomialModel: drf
Model ID:  DRF_model_R_1591139718732_1
Model Summary:
  number_of_trees number_of_internal_trees model_size_in_bytes min_depth
1              20                       20                7715         5
  max_depth mean_depth min_leaves max_leaves mean_leaves
1         5    5.00000         20         30    26.05000


H2OBinomialMetrics: drf
** Reported on training data. **
** Metrics reported on Out-Of-Bag training samples **

MSE:  0.07994675
RMSE:  0.2827486
LogLoss:  0.266595
Mean Per-Class Error:  0.2464692
AUC:  0.8733263
AUCPR:  0.4756788
Gini:  0.7466526

R^2:  0.2436856
```

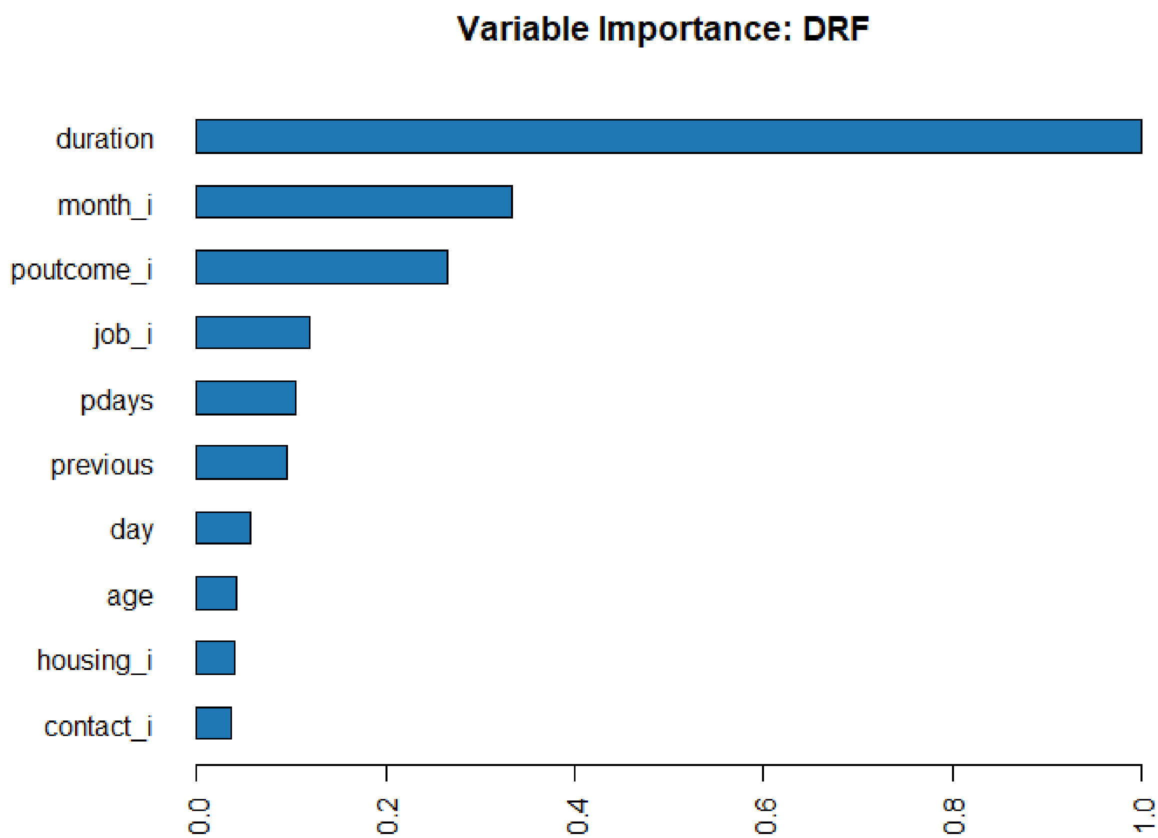
```
Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
          0   1    Error       Rate
0      2539 244 0.087675  =244/2783
1       154 226 0.405263   =154/380
Totals 2693 470 0.125830  =398/3163

Maximum Metrics: Maximum metrics at their respective thresholds
                        metric threshold       value idx
1                       max f1  0.244523    0.531765 172
2                       max f2  0.108434    0.653582 262
3                 max f0point5  0.300456    0.507642 144
4                 max accuracy  0.497186    0.887449  55
5                max precision  0.980392    1.000000   0
6                   max recall  0.019630    1.000000 389
7              max specificity  0.980392    1.000000   0
8             max absolute_mcc  0.244523    0.463492 172
9   max min_per_class_accuracy  0.124687    0.802632 248
10 max mean_per_class_accuracy  0.107437    0.812230 263
11                     max tns  0.980392 2783.000000   0
12                     max fns  0.980392  379.000000   0
13                     max fps  0.000000 2783.000000 399
14                     max tps  0.019630  380.000000 389
15                     max tnr  0.980392    1.000000   0
16                     max fnr  0.980392    0.997368   0
17                     max fpr  0.000000    1.000000 399
18                     max tpr  0.019630    1.000000 389

Gains/Lift Table: Extract with `h2o.gainsLift(, )` or `h2o.gainsLift(, valid=,
xval=)`
```

We can also obtain the importance of the variables in the model. To view the variable importance computed from an H2O model, we can use either the h2o.varimp() or h2o.varimp_plot() functions:

```
#Variable Importance Table
h2o.varimp(RForest_model)
Output:
Variable Importances:
      variable relative_importance scaled_importance percentage
1     duration          639.763550          1.000000   0.455695
2      month_i          212.961884          0.332876   0.151690
3   poutcome_i          169.828339          0.265455   0.120966
4        job_i           76.510735          0.119592   0.054498
5        pdays           67.276260          0.105158   0.047920
6     previous           60.525105          0.094605   0.043111
7          day           35.933807          0.056167   0.025595
8          age           26.424019          0.041303   0.018821
9    housing_i           25.123405          0.039270   0.017895
10   contact_i           23.249025          0.036340   0.016560
11     balance           22.433128          0.035065   0.015979
12    campaign           15.473489          0.024186   0.011022
13   marital_i           13.509902          0.021117   0.009623
14 education_i           10.381807          0.016228   0.007395
#Variable Importance Plot
h2o.varimp_plot(RForest_model)
```



Making Predictions

Below is a brief view of the predictions obtained based on the test dataset.

```
# Generate the predictions on a test set (if necessary):
pred <- h2o.predict(RForest_model, newdata = test)
pred
Output:
  predict        p0         p1
1       0 0.8625073 0.13749275
2       0 0.9584258 0.04157423
3       0 0.9564851 0.04351491
4       0 0.9301738 0.06982616
5       0 0.9190784 0.08092161
6       0 0.9070115 0.09298850
```

FINAL WORDS

This project is about a way to apply supervised machine learning models for classification in Apache Spark using H2O-Sparkling Water through Sparklyr.

You can use H2O-Sparkling Water through Sparklyr to run a variety of classifiers in Apache Spark. For the bank data, the best performing models were both Logistic Regression and Random Forest.

While these models ran on a small data set in a local spark cluster, these methods can be scaled, for the most part, for data analysis in a distributed Apache Spark cluster.