***Hector Alvaro Rojas*** | *Data Science, Visualizations and Applied Statistics* | *August 30, 2020*

# Big Data: MLearning Classifiers Grid Search with Sparklyr and H2O-SWater

## *WHAT IS THIS ALL ABOUT?*

H2O is a product created by the H2O.ai company to combine the main algorithms of machine learning and statistical learning with **Big Data**. Thanks to its way of compressing and storing data, H2O can work with millions of records in a single computer (it uses all its cores) or in a cluster of many computers. Internally, H2O is written in Java and follows the Key / Value paradigm for storing data and Map / Reduce for its algorithms. Thanks to its APIs, it is possible to access all its functions from R, Python, or Scala, as well as through a web interface called Flow.

Although H2O's main advantage over other tools is its scalability, its algorithms are equally useful when working with a small volume of data. 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 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 H2O Sparkling Water Spark package.The main purpose of rsparkling is to provide a connector between sparklyr and H2O's machine learning algorithms. The rsparkling package uses sparklyr for Spark job deployment and initialization of Sparkling Water. After that, user can use the regular h2o R package for modeling.

This present work is about an application and practical exposition of how to apply two classic Grid Search methods for tunning four (Glm, RandomForest, Gbm, Deeplearning) of the most frequently used machine learning models for classification, by using Sparklyr, rsparkling 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 functions and scripts used are proportionated, allowing the reader to consult the appropriate bibliography and tutorial examples clearly and directly.

You can get more specific information about the H2O machine learning models for classification used in this project in the following links:

Generalized Linear Model (Glm)
H2O documentation
R documentation
Random Forest (RandomForest)
H2O documentation
R documentation
Gradient Boosting Machine (Gbm) H2O documentation R documentation

Deep Learning - Neural Networks (Deeplearning)
H2O documentation
R documentation

An excellent clear practical explanation of the Grid Search methods used in this project can be found here. Even though in this video Kevin (the author) use examples based on Python, his concepts explanations are so clearly exposed that I decided to include it here without a doubt.

While in this project models are run 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.

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.

---

# *DATA SOURCE*

In this project I use a prostate cancer dataset which is included in the H2O package. The data was collected by Dr. Donn Young at the Ohio State University Comprehensive Cancer Center for a study of patients with varying degrees of prostate cancer. You can download the dataset here. The classification goal is to predict the incidence of penetration of the prostatic capsule (CAPSULE) (variable y).

Note that, the dataset is not significant [380 rows x 8 columns] 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 location for our project
mypath <- "path"

# set working directory
setwd(mypath)

# get current working directory
getwd()

# Load required minimum packages
library(rsparkling)
library(sparklyr)
library(tidyverse) # to get the whole tidyverse: dplyr, ggplot2 tibble, readr,
tidyr, purrr
library(reshape2)
library(gridExtra) #viewing multiple plots together
library(grid) # To use with tables
library(h2o)
```

```
h2o.init(ip = "localhost",
# Number of threads -1 means use all cores on your machine.
nthreads = -1)
# max_mem_size = "12G")  #max mem size is the maximum memory to allocate to H2O


#Cluster data is deleted in case it has already been started.
h2o.removeAll()

# Spark connection
sc <- spark_connect(master = "local")


# Read data from web
path <- "http://h2o-public-test-
data.s3.amazonaws.com/smalldata/prostate/prostate.csv"
df_raw <- read.csv(path)


glimpse(df_raw)
Observations: 380
Variables: 9
$ ID       1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, ...
$ CAPSULE  0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1,...
$ AGE      65, 72, 70, 76, 69, 71, 68, 61, 69, 68, 68, 72, 72, 65, 75,...
$ RACE     1, 1, 1, 2, 1, 1, 2, 2, 1, 2, 2, 1, 1, 1, 1, 1, 2, 1, 1, 2,...
$ DPROS    2, 3, 1, 2, 1, 3, 4, 4, 1, 1, 4, 2, 4, 4, 1, 2, 1, 2, 1, 3,...
$ DCAPS    1, 2, 2, 1, 1, 2, 2, 2, 1, 2, 2, 2, 2, 2, 1, 1, 1, 1, 2, 2,...
$ PSA      1.4, 6.7, 4.9, 51.2, 12.3, 3.3, 31.9, 66.7, 3.9, 13.0, 4.0,...
$ VOL      0.0, 0.0, 0.0, 20.0, 55.9, 0.0, 0.0, 27.2, 24.0, 0.0, 0.0, ...
$ GLEASON  6, 7, 6, 7, 6, 8, 7, 7, 7, 6, 7, 7, 9, 7, 5, 5, 5, 5, 6, 7,...
# 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"
```

---

# *READJUSTMENT DATASET*

## *GETTING DATASET INTO H2O FORMAT*

```
# Getting mydata_tbl into H2O format.
h2o_df <- as.h2o(df, destination_frame = "h2o_df")

h2o_df
Output:
  ID CAPSULE AGE RACE DPROS DCAPS  PSA  VOL GLEASON
1  1       0  65    1     2     1  1.4  0.0       6
2  2       0  72    1     3     2  6.7  0.0       7
3  3       0  70    1     1     2  4.9  0.0       6
4  4       0  76    2     2     1 51.2 20.0       7
5  5       0  69    1     1     1 12.3 55.9       6
6  6       1  71    1     3     2  3.3  0.0       8

[380 rows x 9 columns]
```

---

## *TRANSFORM CATEGORICAL FEATURES INTO FACTORS*

Now we have readjusted mydata_tbl dataset 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.

When fitting a model, H2O automatically identifies which variables are categorical and internally creates the corresponding dummy variables. It is highly recommended to allow H2O to carry out this process instead of doing it externally since its implementation is highly optimized. The behavior of the encoding of categorical variables can be controlled with the argument categorical_encoding, by default, its value is "AUTO".

By default, H2O standardizes numerical predictors before adjusting the models so that they all have mean zero and variance one. This behavior can be controlled with the "standardize" argument

```
y <- "CAPSULE"
x <- setdiff(names(h2o_df), y)
h2o_df[,y] <- as.factor(h2o_df[,y])
h2o_df[,"RACE" ] <- as.factor(h2o_df[,"RACE" ])
h2o_df[,"DCAPS" ] <- as.factor(h2o_df[,"DCAPS" ])
h2o_df[,"DPROS" ] <- as.factor(h2o_df[,"DPROS" ])
```

Although the data set used in this project is small enough to load everything in memory and directly use the magnificent possibilities of exploratory analysis that R offers, to better represent the way to proceed with large volumes of data, now, by way of example, we use some of H2O's own functions.

---

## *EXPLORING THE DATA*

Although the data set used in this example is small enough to load everything into memory and directly use the magnificent possibilities of exploratory analysis that R offers, to better represent the way to proceed with large volumes of data, H2O's functions are used.

```
# Data set dimensions
h2o.dim(h2o_df)
Output:
[1] 380   9
# Columns name
h2o.colnames(h2o_df)
Output:
[1] "ID"      "CAPSULE" "AGE"     "RACE"    "DPROS"   "DCAPS"   "PSA"
[8] "VOL"     "GLEASON"
```

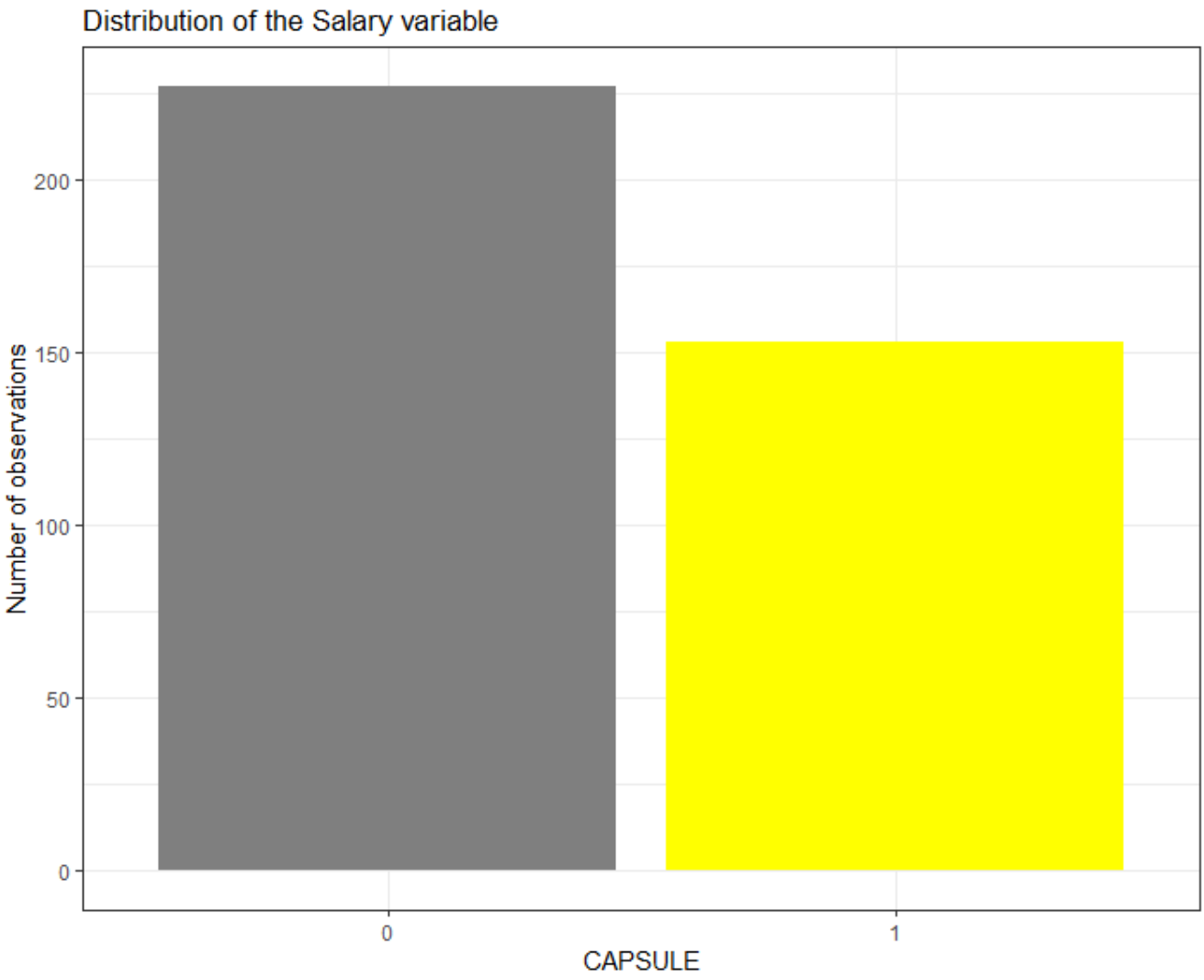
We can get a quick overview of the data and its structure. The "h2o.describe ()" function is very useful to obtain a quick analysis that shows the data type, the number of missing values, the minimum, maximum, mean value, standard deviation, and the number of categories (Cardinality) of each one of variables. H2O uses the name "enum" for data of type factor or character.

```
h2o.describe(h2o_df)
Output:
```

```
    Label Type Missing Zeros PosInf NegInf  Min   Max        Mean       Sigma
Cardinality
1      ID  int       0     0      0      0  1.0 380.0 190.5000000 109.8407939
NA
2 CAPSULE enum       0   227      0      0  0.0   1.0   0.4026316   0.4910743
2
3     AGE  int       0     0      0      0 43.0  79.0  66.0394737   6.5270713
NA
4    RACE enum       0     3      0      0  0.0   2.0          NA          NA
3
5   DPROS enum       0    99      0      0  0.0   3.0          NA          NA
4
6   DCAPS enum       0   339      0      0  0.0   1.0   0.1078947   0.3106564
2
7     PSA real       0     0      0      0  0.3 139.7  15.4086316  19.9975727
NA
8     VOL real       0   167      0      0  0.0  97.6  15.8129211  18.3476200
NA
9 GLEASON  int       0     2      0      0  0.0   9.0   6.3842105   1.0919534
NA
```

To count the number of observations of each class in a categorical variable, as in this case the response variable CAPSULE, the h2o.table () function is used.

```
# A table is created with the number of observations of each type.
tabla_CAPSULE <- as.data.frame(h2o.table(h2o_df$CAPSULE))
tabla_CAPSULE
Output:
  CAPSULE Count
1       0   227
2       1   153
# Once the table is created, it can be loaded into the R environment to be able
to graph.
ggplot(data = tabla_CAPSULE,
        aes(x = as.factor(CAPSULE), y = Count, fill = as.factor(CAPSULE))) +
        geom_col() +
        scale_fill_manual(values = c("gray50", "yellow")) +
        theme_bw() +
        labs(x = "CAPSULE", y = "Number of observations", title = "Distribution
of the Salary variable") +
        theme(legend.position = "none")
```




# *GETTING THE TRAIN AND TEST DATASET*

## *GETTING TRAIN AND TEST DATASETS*

The h2o.splitFrame () function performs random partitions, but it does not allow them to do them in a stratified way, so it does not ensure that the distribution of response variable classes is the same in all partitions. This can be problematic with very unbalanced data (some of the groups are very minority).

In the case of unbalanced data, the base should be adjusted in advance according to traditional methods that allow more effective modeling.

```
# split our data set into 'training', 'test'
partitions <- h2o.splitFrame(h2o_df, 0.7, seed = 2020)
# Create table references
train <-partitions[[1]]
test <-partitions[[2]]
```

# GRID SEARCH GENERALITIES

## INTRODUCTION

Grid-searching is the process of scanning the data to configure optimal parameters for a given model. Grid-searching can be applied across machine learning to calculate the best parameters to use for any given model. Grid-Search will build a model on each parameter combination possible. It iterates through every parameter combination and stores a model for each combination.

Even though grid-search is used to find the optimal hyperparameters of a model which results in the most 'accurate' predictions, It is important to note that it can be extremely computationally expensive and may take your machine quite a long time to run.

There are many hyperparameter optimization/tuning algorithms, but the two most common strategies are: cartesian grid search (or grid search) and random grid search.

## CARTESIAN AND RANDOM GRID SEARCHING OF HYPERPARAMETERS

The cartesian grid search (or grid search, by short) methodically build and evaluate a model for each combination of algorithm parameters specified in a grid.

For instance, if you have three hyperparameters H1, H2, and H3, and each of them can take on 10, 20, and 10 values, respectively, your grid will contain a total of 10 * 20 * 10 = 2000 models.
Cartesian Grid Search, the default grid search method in H2o, exhaustively searches over all possible combinations of the hyperparameters. ***If the search space is small, this should be your method of choice***.

Random search differs from a cartesian search in that you no longer provide a discrete set of values to explore for each hyperparameter; rather, you provide a statistical distribution for each hyperparameter from which values may be randomly sampled. That is to say, Random Search replaces the exhaustive enumeration of all combinations by selecting them randomly.

The idea of random searching of hyperparameters was proposed by James Bergstra & Yoshua Bengio. You can check the original paper here.

In a random grid search a random combination of hyperparameters (H2O will sample uniformly from the set of all possible hyperparameter value combinations) are tested instead of exhaustively testing all possible combinations. Also, the user specifies a stopping criterion, which controls when the random grid search is completed. The user can tell the random grid search to stop by specifying a maximum number of models or the maximum number of seconds allowed for the search. The user may also specify a performance-metric-based stopping criterion, which will stop the random grid search when the performance stops improving by a specified amount. ***If your search space is large, this should be your method of choice***.

# GRID SEARCH MODELING

## MODEL: LOGISTIC REGRESSION (LR_model)

## *CARTESIAN GRID SEARCH*

### *Grid Search Train / Test dataset procedure*

#### *GLM hyperparameters*

```
alpha_opts = list(list(0), list(.25), list(.5), list(.75), list(1))
lambda_opts = list(list(1), list(.5), list(.1), list(.01), list(.001),
list(.0001), list(.00001), list(0))
glm_params <- list(alpha = alpha_opts, lambda =lambda_opts)
```

#### *Train and validate a grid of GLMs*

```
glm_grid <- h2o.grid("glm", y = "CAPSULE", x = c("AGE", "RACE", "PSA",
"GLEASON"),
                     grid_id = "glm_grid1",
                     training_frame= train,
                     family = "binomial",
                     ignore_const_cols = TRUE,
                     seed = 2020,
                     hyper_params = glm_params)
```

#### *Get the grid results, sorted by validation AUC*

```
glm_gridperf1 <- h2o.getGrid(grid_id = "glm_grid1",
                             sort_by = "auc",
                             decreasing = TRUE)
glm_gridperf1
Output:
H2O Grid Details
================

Grid ID: glm_grid1
Used hyper parameters:
  -  alpha
  -  lambda
Number of models: 40
Number of failed models: 0

Hyper-Parameter Search Summary: ordered by decreasing auc
   alpha  lambda           model_ids                 auc
1  [0.0]  [0.01] glm_grid1_model_16 0.7852855477855478
2 [0.75] [0.001] glm_grid1_model_24 0.7852272727272728
3  [1.0] [0.001] glm_grid1_model_25 0.7852272727272728
4  [0.5] [0.001] glm_grid1_model_23 0.7851689976689976
5  [0.5]  [0.01] glm_grid1_model_18 0.7848776223776223

---
    alpha lambda         model_ids auc
35 [0.25]  [1.0] glm_grid1_model_2 0.5
36  [0.5]  [1.0] glm_grid1_model_3 0.5
37 [0.75]  [1.0] glm_grid1_model_4 0.5
38  [1.0]  [1.0] glm_grid1_model_5 0.5
39  [0.5]  [0.5] glm_grid1_model_8 0.5
40 [0.75]  [0.5] glm_grid1_model_9 0.5
```

### *Grid Search KFold Cross Validation procedure*

#### *Train and validate a grid of GLMs*

```
glm_grid <- h2o.grid("glm", y = "CAPSULE", x = c("AGE", "RACE", "PSA",
"GLEASON"),
                     grid_id = "glm_grid2",
                     training_frame= h2o_df,
                     family = "binomial",
```

```
                      seed = 2020,
                      nfolds=10,
                      fold_assignment="Stratified", # can be "AUTO", "Modulo",
"Random" or "Stratified"
                      hyper_params = glm_params)
```

*Get the grid results, sorted by validation AUC*

```
glm_gridperf2 <- h2o.getGrid(grid_id = "glm_grid2",
                             sort_by = "auc",
                             decreasing = TRUE)
glm_gridperf2
Output:
H2O Grid Details
================

Grid ID: glm_grid2
Used hyper parameters:
  -  alpha
  -  lambda
Number of models: 40
Number of failed models: 0

Hyper-Parameter Search Summary: ordered by decreasing auc
   alpha  lambda           model_ids                 auc
1  [0.0]  [0.01] glm_grid2_model_16 0.7859549105986007
2  [0.0]   [0.1] glm_grid2_model_11 0.7848895799141977
3 [0.75] [0.001] glm_grid2_model_24  0.784659238144597
4  [0.5] [0.001] glm_grid2_model_23 0.7845728599809968
5  [1.0] [0.001] glm_grid2_model_25 0.7845440672597966

---
    alpha lambda         model_ids                  auc
35 [0.25]  [1.0] glm_grid2_model_2 0.43675678788402295
36  [0.5]  [1.0] glm_grid2_model_3 0.43675678788402295
37 [0.75]  [1.0] glm_grid2_model_4 0.43675678788402295
38  [1.0]  [1.0] glm_grid2_model_5 0.43675678788402295
39  [0.5]  [0.5] glm_grid2_model_8 0.43675678788402295
40 [0.75]  [0.5] glm_grid2_model_9 0.43675678788402295
```

---

## *RANDOM GRID SEARCH*

## *Random Grid Search Train / Test dataset procedure*

### *GLM hyperparameters*

```
search_criteria <- list(strategy = "RandomDiscrete",
                        max_models = 50)
```

*Train and validate a grid of GLMs*

```
glm_grid <- h2o.grid("glm", y = "CAPSULE", x = c("AGE", "RACE", "PSA",
"GLEASON"),
                      grid_id = "glm_grid3",
                      training_frame= train,
                      family = "binomial",
                      ignore_const_cols = TRUE,
                      seed = 2020,
                      hyper_params = glm_params,
                      search_criteria = search_criteria)
```

*Get the grid results, sorted by validation AUC*

```
# Get the grid results, sorted by validation auc
glm_gridperf3 <- h2o.getGrid(grid_id = "glm_grid3",
                             sort_by = "auc",
                             decreasing = TRUE)
glm_gridperf3
# Output
H2O Grid Details
================

Grid ID: glm_grid3
Used hyper parameters:
  -  alpha
  -  lambda
Number of models: 40
Number of failed models: 0

Hyper-Parameter Search Summary: ordered by decreasing auc
   alpha  lambda          model_ids                auc
1   [0.0]  [0.01] glm_grid3_model_12 0.7852855477855478
2 [0.75] [0.001] glm_grid3_model_22 0.7852272727272728
3   [1.0] [0.001] glm_grid3_model_35 0.7852272727272728
4   [0.5] [0.001]  glm_grid3_model_1 0.7851689976689976
5   [0.5]  [0.01] glm_grid3_model_28 0.7848776223776223

---
    alpha lambda          model_ids auc
35  [0.5]  [1.0] glm_grid3_model_19 0.5
36  [1.0]  [1.0] glm_grid3_model_24 0.5
37 [0.75]  [1.0] glm_grid3_model_32 0.5
38 [0.75]  [0.5] glm_grid3_model_34 0.5
39  [1.0]  [0.5]  glm_grid3_model_4 0.5
40  [0.5]  [0.5]  glm_grid3_model_5 0.5
```

## *Grid Search KFold Cross Validation procedure*

*Train and validate a grid of GLMs*

```
glm_grid <- h2o.grid("glm", y = "CAPSULE", x = c("AGE", "RACE", "PSA",
"GLEASON"),
                     grid_id = "glm_grid4",
                     training_frame= h2o_df,
                     family = "binomial",
                     ignore_const_cols = TRUE,
                     seed = 2020,
                     nfolds=10,
                     fold_assignment="Stratified", # can be "AUTO", "Modulo",
"Random" or "Stratified"
                     hyper_params = glm_params,
                     search_criteria = search_criteria)
```

*Get the grid results, sorted by validation AUC*

```
glm_gridperf4 <- h2o.getGrid(grid_id = "glm_grid4",
                             sort_by = "auc",
                             decreasing = TRUE)
glm_gridperf4
Output:
H2O Grid Details
================

Grid ID: glm_grid4
Used hyper parameters:
```

```
  -  alpha
  -  lambda
Number of models: 40
Number of failed models: 0

Hyper-Parameter Search Summary: ordered by decreasing auc
   alpha  lambda          model_ids                auc
1  [0.0]  [0.01] glm_grid4_model_13 0.7859549105986007
2  [0.0]   [0.1]  glm_grid4_model_8 0.7848895799141977
3 [0.75] [0.001] glm_grid4_model_36  0.784659238144597
4  [0.5] [0.001] glm_grid4_model_33 0.7845728599809968
5  [1.0] [0.001] glm_grid4_model_30 0.7845440672597966

---
    alpha lambda          model_ids                 auc
35  [0.5]  [1.0] glm_grid4_model_17 0.43675678788402295
36 [0.75]  [1.0] glm_grid4_model_25 0.43675678788402295
37  [0.5]  [0.5]  glm_grid4_model_3 0.43675678788402295
38 [0.75]  [0.5] glm_grid4_model_31 0.43675678788402295
39 [0.25]  [1.0] glm_grid4_model_32 0.43675678788402295
40  [1.0]  [0.5] glm_grid4_model_37 0.43675678788402295
```

## *MODEL: RANDOM FOREST (RForest_model)*

## *CARTESIAN GRID SEARCH*

## *Grid Search Train / Test dataset procedure*

### *RForest hyperparameters*

```
rf_params <- list(ntrees = seq(50, 200, by = 50),
                       mtries = seq(2, 4, by = 1),
                       # max_depth = seq(10, 30, by = 10),
                       # min_rows = seq(1, 3, by = 1),
                       # nbins = seq(20, 30, by = 10),
                       sample_rate = c(0.55, 0.632, 0.75))
```

### *Train and validate a grid of RFs*

```
rf_grid <- h2o.grid("randomForest", y = "CAPSULE", x = c("AGE", "RACE", "PSA",
"GLEASON"),
                      grid_id = "rf_grid1",
                      training_frame= train,
                      ignore_const_cols = TRUE,
                      seed = 2020,
                      hyper_params = rf_params)
```

### *Get the grid results, sorted by validation AUC*

```
rf_gridperf1 <- h2o.getGrid(grid_id = "rf_grid1",
                            sort_by = "auc",
                            decreasing = TRUE)
rf_gridperf1
Output:
H2O Grid Details
================

Grid ID: rf_grid1
Used hyper parameters:
  -  mtries
  -  ntrees
  -  sample_rate
Number of models: 36
```

```
Number of failed models: 0

Hyper-Parameter Search Summary: ordered by decreasing auc
  mtries ntrees sample_rate         model_ids                auc
1      3     50        0.55  rf_grid1_model_2 0.6836247086247086
2      3    100        0.55  rf_grid1_model_5 0.6831876456876457
3      2    100       0.632 rf_grid1_model_16 0.6830710955710956
4      2     50       0.632 rf_grid1_model_13 0.6821095571095571
5      3    200        0.55 rf_grid1_model_11 0.6809440559440559

---
   mtries ntrees sample_rate         model_ids                 auc
31      4    100        0.75 rf_grid1_model_30 0.6580419580419581
32      2     50        0.55  rf_grid1_model_1 0.6574009324009323
33      3    150        0.75 rf_grid1_model_32 0.6561771561771562
34      4    200        0.75 rf_grid1_model_36 0.6552738927738928
35      4    150        0.75 rf_grid1_model_33 0.6528846153846154
36      2     50        0.75 rf_grid1_model_25 0.6500874125874126
```

## *Grid Search KFold Cross Validation procedure*

### *Train and validate a grid of RFs*

```
rf_grid <- h2o.grid("randomForest", y = "CAPSULE", x = c("AGE", "RACE", "PSA",
"GLEASON"),
                    grid_id = "rf_grid2",
                    training_frame= h2o_df,
                    ignore_const_cols = TRUE,
                    seed = 2020,
                    nfolds = 10,
                    fold_assignment="Stratified", # can be "AUTO", "Modulo",
"Random" or "Stratified"
                    hyper_params = rf_params)
```

### *Get the grid results, sorted by validation AUC*

```
rf_gridperf2 <- h2o.getGrid(grid_id = "rf_grid2",
                            sort_by = "auc",
                            decreasing = TRUE)
rf_gridperf2
Output:
H2O Grid Details
================

Grid ID: rf_grid2
Used hyper parameters:
  -  mtries
  -  ntrees
  -  sample_rate
Number of models: 36
Number of failed models: 0

Hyper-Parameter Search Summary: ordered by decreasing auc
  mtries ntrees sample_rate         model_ids                auc
1      2    200        0.55 rf_grid2_model_10 0.7319973510696496
2      2    150        0.55  rf_grid2_model_7 0.7283694681984395
3      2    100        0.55  rf_grid2_model_4 0.7271025884656359
4      2     50        0.55  rf_grid2_model_1 0.7248279634908296
5      3    200        0.55 rf_grid2_model_11 0.7229420402522242

---
   mtries ntrees sample_rate         model_ids                auc
```

```
31      3    200         0.75 rf_grid2_model_35 0.7043563387175722
32      3    150         0.75 rf_grid2_model_32 0.7037372952117704
33      4     50         0.75 rf_grid2_model_27  0.701793786530765
34      4    100         0.75 rf_grid2_model_30 0.7011315539431632
35      4    150         0.75 rf_grid2_model_33 0.7003973395525611
36      4    200         0.75 rf_grid2_model_36 0.6989577034925571
```

---

## *RANDOM GRID SEARCH*

## *Random Grid Search Train / Test dataset procedure*

```
search_criteria <- list(strategy = "RandomDiscrete",
                        max_models = 50)
```

### *Train and validate a grid of RFs*

```
rf_grid <- h2o.grid("randomForest", y = "CAPSULE", x = c("AGE", "RACE", "PSA",
"GLEASON"),
                    grid_id = "rf_grid3",
                    training_frame= train,
                    ignore_const_cols = TRUE,
                    seed = 2020,
                    hyper_params = rf_params,
                    search_criteria = search_criteria)
```

### *Get the grid results, sorted by validation AUC*

```
# Get the grid results, sorted by validation auc
rf_gridperf3 <- h2o.getGrid(grid_id = "rf_grid3",
                            sort_by = "auc",
                            decreasing = TRUE)
rf_gridperf3
# Output
H2O Grid Details
================

Grid ID: rf_grid3
Used hyper parameters:
  -  mtries
  -  ntrees
  -  sample_rate
Number of models: 36
Number of failed models: 0

Hyper-Parameter Search Summary: ordered by decreasing auc
  mtries ntrees sample_rate         model_ids                auc
1      3     50        0.55 rf_grid3_model_18 0.6836247086247086
2      3    100        0.55  rf_grid3_model_4 0.6831876456876457
3      2    100       0.632 rf_grid3_model_30 0.6830710955710956
4      2     50       0.632  rf_grid3_model_8 0.6821095571095571
5      3    200        0.55 rf_grid3_model_15 0.6809440559440559

---
   mtries ntrees sample_rate         model_ids                auc
31      4    100        0.75 rf_grid3_model_12 0.6580419580419581
32      2     50        0.55 rf_grid3_model_26 0.6574009324009323
33      3    150        0.75 rf_grid3_model_16 0.6561771561771562
34      4    200        0.75 rf_grid3_model_13 0.6552738927738928
35      4    150        0.75  rf_grid3_model_6 0.6528846153846154
```

```
36      2     50         0.75  rf_grid3_model_5 0.6500874125874126
```

## *Grid Search KFold Cross Validation procedure*

### *Train and validate a grid of RFs*

```
rf_grid <- h2o.grid("randomForest", y = "CAPSULE", x = c("AGE", "RACE", "PSA",
"GLEASON"),
                    grid_id = "rf_grid4",
                    training_frame= h2o_df,
                    ignore_const_cols = TRUE,
                    seed = 2020,
                    nfolds=10,
                    fold_assignment="Stratified", # can be "AUTO", "Modulo",
"Random" or "Stratified"
                    hyper_params = rf_params,
                    search_criteria = search_criteria)
```

### *Get the grid results, sorted by validation AUC*

```
rf_gridperf4 <- h2o.getGrid(grid_id = "rf_grid4",
                            sort_by = "auc",
                            decreasing = TRUE)
rf_gridperf4
Output:
H2O Grid Details
================

Grid ID: rf_grid4
Used hyper parameters:
  -  mtries
  -  ntrees
  -  sample_rate
Number of models: 36
Number of failed models: 0

Hyper-Parameter Search Summary: ordered by decreasing auc
  mtries ntrees sample_rate         model_ids                auc
1      2    200        0.55 rf_grid4_model_14 0.7319973510696496
2      2    150        0.55 rf_grid4_model_35 0.7283694681984395
3      2    100        0.55 rf_grid4_model_28 0.7271025884656359
4      2     50        0.55 rf_grid4_model_30 0.7248279634908296
5      3    200        0.55 rf_grid4_model_36 0.7229420402522242

---
   mtries ntrees sample_rate         model_ids                 auc
31      3    200        0.75 rf_grid4_model_15 0.7043563387175722
32      3    150        0.75 rf_grid4_model_32 0.7037372952117704
33      4     50        0.75  rf_grid4_model_2  0.701793786530765
34      4    100        0.75 rf_grid4_model_19 0.7011315539431632
35      4    150        0.75 rf_grid4_model_26 0.7003973395525611
36      4    200        0.75  rf_grid4_model_7 0.6989577034925571
```

---

## *MODEL: GRADIENT BOOSTING MACHINE (GBM_model)*

## *CARTESIAN GRID SEARCH*

## *Grid Search Train / Test dataset procedure*

### *GBM hyperparameters*

```
gbm_params <- list(learn_rate = c(0.001, 0.05, 0.1),
                   max_depth = c(5, 10, 15, 20),
                   sample_rate = c(0.8, 1.0))
```

### *Train and validate a grid of GBMs*

```
gbm_grid <- h2o.grid("gbm", y = "CAPSULE", x = c("AGE", "RACE", "PSA",
"GLEASON"),
                     grid_id = "gbm_grid1",
                     training_frame= train,
                     ignore_const_cols = TRUE,
                     # Early stop
                     score_tree_interval = 100,
                     stopping_rounds = 3,
                     stopping_tolerance = 0.001,
                     # Fixed hyperparameters
                     ntrees   = 100,
                     # Optimized hyperparameters
                     hyper_params = gbm_params,
                     # Search type
                     search_criteria = list(strategy = "Cartesian"),
                     seed = 2020)
```

### *Get the grid results, sorted by validation AUC*

```
gbm_gridperf1 <- h2o.getGrid(grid_id = "gbm_grid1",
                             sort_by = "auc",
                             decreasing = TRUE)
gbm_gridperf1
Output
H2O Grid Details
================

Grid ID: gbm_grid1
Used hyper parameters:
  -  learn_rate
  -  max_depth
  -  sample_rate
Number of models: 24
Number of failed models: 0

Hyper-Parameter Search Summary: ordered by decreasing auc
  learn_rate max_depth sample_rate          model_ids                 auc
1        0.1        20         1.0 gbm_grid1_model_24 0.9996794871794872
2        0.1        15         1.0 gbm_grid1_model_21 0.9994463869463869
3        0.1        10         1.0 gbm_grid1_model_18  0.999067599067599
4        0.1        20         0.8 gbm_grid1_model_12  0.995425407925408
5        0.1        15         0.8  gbm_grid1_model_9  0.995425407925408

---
   learn_rate max_depth sample_rate          model_ids                 auc
19      0.001        15         0.8  gbm_grid1_model_7 0.8703088578088579
20      0.001         5         0.8  gbm_grid1_model_1  0.867511655011655
21      0.001         5         1.0 gbm_grid1_model_13 0.8423368298368299
22      0.001        10         1.0 gbm_grid1_model_16 0.8423368298368299
23      0.001        15         1.0 gbm_grid1_model_19 0.8423368298368299
24      0.001        20         1.0 gbm_grid1_model_22 0.8423368298368299
```

## *Grid Search KFold Cross Validation procedure*

### *Train and validate a grid of GBMs*

```
gbm_grid <- h2o.grid("gbm", y = "CAPSULE", x = c("AGE", "RACE", "PSA",
"GLEASON"),
                     grid_id = "gbm_grid2",
                     training_frame= h2o_df,
                     ignore_const_cols = TRUE,
                     nfolds = 10,
                     fold_assignment="Stratified", # can be "AUTO", "Modulo",
"Random" or "Stratified"
                     # Early stop
                     score_tree_interval = 100,
                     stopping_rounds = 3,
                     stopping_tolerance = 0.001,
                     # Fixed hyperparameters
                     ntrees   = 100,
                     # Optimized hyperparameters
                     hyper_params = gbm_params,
                     # Search type
                     search_criteria = list(strategy = "Cartesian"),
                     seed = 2020)
```

### *Get the grid results, sorted by validation AUC*

```
gbm_gridperf2 <- h2o.getGrid(grid_id = "gbm_grid2",
                             sort_by = "auc",
                             decreasing = TRUE)
gbm_gridperf2
Output
H2O Grid Details
================

Grid ID: gbm_grid2
Used hyper parameters:
  -  learn_rate
  -  max_depth
  -  sample_rate
Number of models: 24
Number of failed models: 0

Hyper-Parameter Search Summary: ordered by decreasing auc
  learn_rate max_depth sample_rate          model_ids                 auc
1       0.05         5         1.0 gbm_grid2_model_14  0.741081454608275
2      0.001         5         0.8  gbm_grid2_model_1 0.7371224554432639
3       0.05         5         0.8  gbm_grid2_model_2 0.7367193573464629
4       0.05        10         0.8  gbm_grid2_model_5 0.7359995393164609
5       0.05        20         0.8 gbm_grid2_model_11 0.7358843684316605

---
   learn_rate max_depth sample_rate          model_ids                 auc
19      0.001        15         1.0 gbm_grid2_model_19 0.7140882784831994
20      0.001        20         1.0 gbm_grid2_model_22 0.7140882784831994
21        0.1        10         1.0 gbm_grid2_model_18 0.7123607152111947
22        0.1        20         1.0 gbm_grid2_model_24 0.7085888687339841
23        0.1        10         0.8  gbm_grid2_model_6 0.7085888687339841
24        0.1        15         1.0 gbm_grid2_model_21 0.7068037200195791
```

---

## *RANDOM GRID SEARCH*

### *Random Grid Search Train / Test dataset procedure*

```
search_criteria <- list(strategy = "RandomDiscrete",
```

```
                          max_models = 50)
```

*Train and validate a grid of GBMs*

```
gbm_grid <- h2o.grid("gbm", y = "CAPSULE", x = c("AGE", "RACE", "PSA",
"GLEASON"),
                     grid_id = "gbm_grid3",
                     training_frame= train,
                     ignore_const_cols = TRUE,
                     #nfolds = 10,
                     #fold_assignment="Stratified", # can be "AUTO", "Modulo",
"Random" or "Stratified"
                     # Early stop
                     score_tree_interval = 100,
                     stopping_rounds = 3,
                     stopping_tolerance = 0.001,
                     # Fixed hyperparameters
                     ntrees   = 100,
                     # Optimized hyperparameters
                     hyper_params = gbm_params,
                     # Search type
                     search_criteria = search_criteria,
                     seed = 2020)
```

*Get the grid results, sorted by validation AUC*

```
gbm_gridperf3 <- h2o.getGrid(grid_id = "gbm_grid3",
                             sort_by = "auc",
                             decreasing = TRUE)
gbm_gridperf3
Output
H2O Grid Details
================

Grid ID: gbm_grid3
Used hyper parameters:
  -  learn_rate
  -  max_depth
  -  sample_rate
Number of models: 24
Number of failed models: 0

Hyper-Parameter Search Summary: ordered by decreasing auc
  learn_rate max_depth sample_rate          model_ids                 auc
1        0.1        20         1.0 gbm_grid3_model_13 0.9996794871794872
2        0.1        15         1.0 gbm_grid3_model_9  0.9994463869463869
3        0.1        10         1.0 gbm_grid3_model_10  0.999067599067599
4        0.1        15         0.8 gbm_grid3_model_14  0.995425407925408
5        0.1        20         0.8 gbm_grid3_model_15  0.995425407925408

---
   learn_rate max_depth sample_rate          model_ids                 auc
19      0.001        10         0.8  gbm_grid3_model_6 0.8703088578088579
20      0.001         5         0.8  gbm_grid3_model_3  0.867511655011655
21      0.001         5         1.0 gbm_grid3_model_11 0.8423368298368299
22      0.001        10         1.0 gbm_grid3_model_12 0.8423368298368299
23      0.001        15         1.0 gbm_grid3_model_16 0.8423368298368299
24      0.001        20         1.0 gbm_grid3_model_23 0.8423368298368299
```

## *Random Grid Search KFold Cross Validation procedure*

```
search_criteria <- list(strategy = "RandomDiscrete",
                        max_models = 50)
```

### *Train and validate a grid of GBMs*

```
gbm_grid <- h2o.grid("gbm", y = "CAPSULE", x = c("AGE", "RACE", "PSA",
"GLEASON"),
                     grid_id = "gbm_grid4",
                     training_frame= h2o_df,
                     ignore_const_cols = TRUE,
                     nfolds = 10,
                     fold_assignment="Stratified", # can be "AUTO", "Modulo",
"Random" or "Stratified"
                     # Early stop
                     score_tree_interval = 100,
                     stopping_rounds = 3,
                     stopping_tolerance = 0.001,
                     # Fixed hyperparameters
                     ntrees   = 100,
                     # Optimized hyperparameters
                     hyper_params = gbm_params,
                     # Search type
                     search_criteria = search_criteria,
                     seed = 2020)
```

### *Get the grid results, sorted by validation AUC*

```
gbm_gridperf4 <- h2o.getGrid(grid_id = "gbm_grid4",
                             sort_by = "auc",
                             decreasing = TRUE)
gbm_gridperf4
Output
H2O Grid Details
================

Grid ID: gbm_gri4
Used hyper parameters:
  -  learn_rate
  -  max_depth
  -  sample_rate
Number of models: 24
Number of failed models: 0

Hyper-Parameter Search Summary: ordered by decreasing auc
  learn_rate max_depth sample_rate          model_ids                auc
1       0.05         5         1.0 gbm_grid4_model_15  0.741081454608275
2      0.001         5         0.8 gbm_grid4_model_23 0.7371224554432639
3       0.05         5         0.8  gbm_grid4_model_9 0.7367193573464629
4       0.05        10         0.8  gbm_grid4_model_6 0.7359995393164609
5       0.05        20         0.8 gbm_grid4_model_17 0.7358843684316605

---
   learn_rate max_depth sample_rate          model_ids                 auc
19      0.001        15         1.0 gbm_grid4_model_18 0.7140882784831994
20      0.001        20         1.0 gbm_grid4_model_22 0.7140882784831994
21        0.1        10         1.0  gbm_grid4_model_5 0.7123607152111947
22        0.1        20         1.0 gbm_grid4_model_19 0.7085888687339841
23        0.1        10         0.8  gbm_grid4_model_7 0.7085888687339841
24        0.1        15         1.0 gbm_grid4_model_14 0.7068037200195791
```

## *MODEL: DEEP LEARNING - NEURAL NETWORKS (DL_model)*
## *CARTESIAN GRID SEARCH*
### *Grid Search Train / Test dataset procedure*
#### *DL hyperparameters*

```
dl_params <- list(hidden = list(c(64), c(128), c(256), c(512), c(1024),
                                      c(64,64), c(128,128), c(256,256), c(512,
512)))
```

#### *Train and validate a grid of DLs*

```
dl_grid <- h2o.grid(
                      # Algorithm
                      algorithm="deeplearning",
                      activation=c("RectifierWithDropout"),
                      epochs=500,
                      #adaptive_rate = FALSE,
                      # Response variable and predictors
                      y = "CAPSULE",
                      x = c("AGE", "RACE", "PSA", "GLEASON"),
                      grid_id = "dl_grid1",
                      # Training data
                      training_frame= train,
                      shuffle_training_data = FALSE,
                      # Preprocessed
                      standardize = TRUE,
                      missing_values_handling = "Skip",
                      ignore_const_cols = TRUE,
                      # Early stop
                      stopping_metric="AUC",
                      stopping_tolerance= 0.01,
                      stopping_rounds = 3,
                      # Regularization
                      l1=1e-5,
                      l2=1e-5,
                      # Optimized hyperparameters
                      hyper_params = dl_params,
                      # Search type
                      search_criteria = list(strategy = "Cartesian"),
                      seed = 2020)
```

#### *Get the grid results, sorted by validation AUC*

```
dl_gridperf1 <- h2o.getGrid(grid_id = "dl_grid1",
                            sort_by = "auc",
                            decreasing = TRUE)
dl_gridperf1
Output
H2O Grid Details
================

Grid ID: dl_grid1
Used hyper parameters:
  -  hidden
Number of models: 9
Number of failed models: 0

Hyper-Parameter Search Summary: ordered by decreasing auc
      hidden        model_ids                auc
1     [1024] dl_grid6_model_5 0.8193181818181818
2      [512] dl_grid6_model_4 0.8131410256410256
```

```
3      [256] dl_grid6_model_3 0.8083041958041959
4 [256, 256] dl_grid6_model_8 0.8052156177156178
5      [128] dl_grid6_model_2 0.8037004662004663
6 [512, 512] dl_grid6_model_9 0.8035256410256411
7 [128, 128] dl_grid6_model_7 0.8032342657342657
8       [64] dl_grid6_model_1 0.8026515151515151
9   [64, 64] dl_grid6_model_6 0.8010780885780886
```

## *Grid Search KFold Cross Validation procedure*

### *Train and validate a grid of GBMs*

```
dl_grid <- h2o.grid(
                      # Algorithm
                      algorithm="deeplearning",
                      activation=c("RectifierWithDropout"),
                      epochs=500,
                      #adaptive_rate = FALSE,
                      # Response variable and predictors
                      y = "CAPSULE",
                      x = c("AGE", "RACE", "PSA", "GLEASON"),
                      grid_id = "dl_grid2",
                      # Training data
                      training_frame= h2o_df,
                      shuffle_training_data = FALSE,
                      # Cross Validation
                      nfolds = 10,
                      fold_assignment="Stratified", # can be "AUTO", "Modulo",
"Random" or "Stratified"
                      # Preprocessed
                      standardize = TRUE,
                      missing_values_handling = "Skip",
                      ignore_const_cols = TRUE,
                      # Early stop
                      stopping_metric="AUC",
                      stopping_tolerance= 0.01,
                      stopping_rounds = 3,
                      # Regularization
                      l1=1e-5,
                      l2=1e-5,
                      # Optimized hyperparameters
                      hyper_params = dl_params,
                      # Search type
                      search_criteria = list(strategy = "Cartesian"),
                      seed = 2020)
```

### *Get the grid results, sorted by validation AUC*

```
dl_gridperf2 <- h2o.getGrid(grid_id = "dl_grid2",
                            sort_by = "auc",
                            decreasing = TRUE)
dl_gridperf2
Output
H2O Grid Details
================

Grid ID: dl_grid2
Used hyper parameters:
  -  hidden
Number of models: 9
Number of failed models: 0
```

```
Hyper-Parameter Search Summary: ordered by decreasing auc
      hidden        model_ids                auc
1   [64, 64] dl_grid2_model_6 0.7866171431862025
2       [64] dl_grid2_model_1 0.7848319944717975
3      [256] dl_grid2_model_3 0.7813192824853876
4      [128] dl_grid2_model_2 0.7697446085629552
5      [512] dl_grid2_model_4  0.765022602286142
6 [128, 128] dl_grid2_model_7 0.7602718032881288
7 [512, 512] dl_grid2_model_9 0.7557513460597161
8     [1024] dl_grid2_model_5 0.7545996372117129
9 [256, 256] dl_grid2_model_8 0.7353373067288589
```

---

## *RANDOM GRID SEARCH*

## *Random Grid Search Train / Test dataset procedure*

```
search_criteria <- list(strategy = "RandomDiscrete",
                        max_models = 50)
```

### *Train and validate a grid of DLs*

```
dl_grid <- h2o.grid(
                      # Algorithm
                      algorithm="deeplearning",
                      activation=c("RectifierWithDropout"),
                      epochs=500,
                      #adaptive_rate = FALSE,
                      # Response variable and predictors
                      y = "CAPSULE",
                      x = c("AGE", "RACE", "PSA", "GLEASON"),
                      grid_id = "dl_grid3",
                      # Training data
                      training_frame= train,
                      shuffle_training_data = FALSE,
                      # Preprocessed
                      standardize = TRUE,
                      missing_values_handling = "Skip",
                      ignore_const_cols = TRUE,
                      # Early stop
                      stopping_metric="AUC",
                      stopping_tolerance= 0.01,
                      stopping_rounds = 3,
                      # Regularization
                      l1=1e-5,
                      l2=1e-5,
                      # Optimized hyperparameters
                      hyper_params = dl_params,
                      # Search type
                      search_criteria = search_criteria,
                      seed = 2020)
```

### *Get the grid results, sorted by validation AUC*

```
dl_gridperf3 <- h2o.getGrid(grid_id = "dl_grid3",
                            sort_by = "auc",
                            decreasing = TRUE)
dl_gridperf3
Output
H2O Grid Details
```

```
================

Grid ID: dl_grid3
Used hyper parameters:
  -  hidden
Number of models: 9
Number of failed models: 0

Hyper-Parameter Search Summary: ordered by decreasing auc
      hidden        model_ids                auc
1     [1024] dl_grid6_model_5 0.8193181818181818
2      [512] dl_grid6_model_4 0.8131410256410256
3      [256] dl_grid6_model_3 0.8083041958041959
4 [256, 256] dl_grid6_model_8 0.8052156177156178
5      [128] dl_grid6_model_2 0.8037004662004663
6 [512, 512] dl_grid6_model_9 0.8035256410256411
7 [128, 128] dl_grid6_model_7 0.8032342657342657
8       [64] dl_grid6_model_1 0.8026515151515151
9   [64, 64] dl_grid6_model_6 0.8010780885780886
```

## *Random Grid Search KFold Cross Validation procedure*

```
search_criteria <- list(strategy = "RandomDiscrete",
                        max_models = 50)
```

### *Train and validate a grid of DLs*

```
dl_grid <- h2o.grid(
                      # Algorithm
                      algorithm="deeplearning",
                      activation=c("RectifierWithDropout"),
                      epochs=500,
                      #adaptive_rate = FALSE,
                      # Response variable and predictors
                      y = "CAPSULE",
                      x = c("AGE", "RACE", "PSA", "GLEASON"),
                      grid_id = "dl_grid4",
                      # Training data
                      training_frame= h2o_df,
                      shuffle_training_data = FALSE,
                      # Cross Validation
                      nfolds = 10,
                      fold_assignment="Stratified", # can be "AUTO", "Modulo",
"Random" or "Stratified"
                      # Preprocessed
                      standardize = TRUE,
                      missing_values_handling = "Skip",
                      ignore_const_cols = TRUE,
                      # Early stop
                      stopping_metric="AUC",
                      stopping_tolerance= 0.01,
                      stopping_rounds = 3,
                      # Regularization
                      l1=1e-5,
                      l2=1e-5,
                      # Optimized hyperparameters
                      hyper_params = dl_params,
                      # Search type
                      search_criteria = search_criteria,
                      seed = 2020)
```

*Get the grid results, sorted by validation AUC*

```
dl_gridperf4 <- h2o.getGrid(grid_id = "dl_grid4",
                            sort_by = "auc",
                            decreasing = TRUE)
dl_gridperf4
Output
H2O Grid Details
================

Grid ID: dl_grid4
Used hyper parameters:
  -  hidden
Number of models: 9
Number of failed models: 0

Hyper-Parameter Search Summary: ordered by decreasing auc
      hidden        model_ids                 auc
1     [1024] dl_grid4_model_5 0.8020500417494457
2   [64, 64] dl_grid4_model_2 0.7836227001813941
3       [64] dl_grid4_model_3 0.7834787365753938
4 [128, 128] dl_grid4_model_9 0.7688808269269529
5 [512, 512] dl_grid4_model_8 0.7673260199821486
6      [128] dl_grid4_model_1 0.7659151766433446
7      [512] dl_grid4_model_4  0.763237453571737
8      [256] dl_grid4_model_7 0.7584578618525237
9 [256, 256] dl_grid4_model_6 0.7539661973453111
```

---

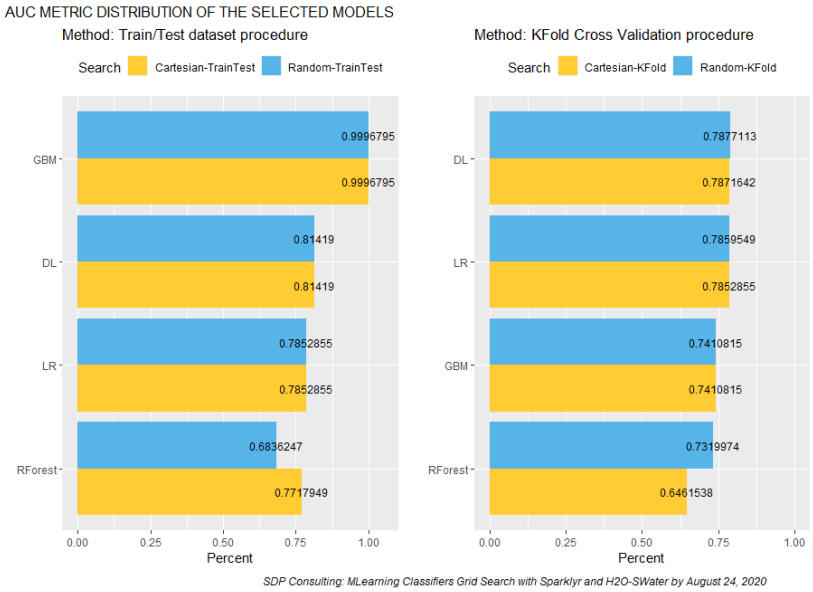
# *SELECT BEST MODEL AND MAKE PREDICTIONS*

## *SELECT BEST MODEL*

The models are compared based on the method (Train/Test or KFold) and the AUC metric obtained in each case.

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

| | Search | LR | RForest | GBM | DL |
|---|---|---|---|---|---|
| 1 | Cartesian-TrainTest | 0.7852855 | 0.7717949 | 0.9996795 | 0.8141900 |
| 2 | Cartesian-KFold | 0.7852855 | 0.6461538 | 0.7410815 | 0.7871642 |
| 3 | Random-TrainTest | 0.7852855 | 0.6836247 | 0.9996795 | 0.8141900 |
| 4 | Random-KFold | 0.7859549 | 0.7319974 | 0.7410815 | 0.7877113 |




The Gradient Boosting Machine (GBM) algorithm performed the best and had a comparatively better AUC metric in the Train/Test method for both the Cartesian and Random procedures. Meantime, on the other hand, the Deep Learning-Neural Network (DL) model got a better AUC result for the KFold method in the two procedures applied.

---

## *BEST MODELS BY TRAIN/TEST METHOD*

### *Selected model*

The selected model is the Gradient Boosting Machine (GBM) algorithm with AUC = 0.9996794871794872 and Hyper-Parameter:
learn_rate = 0.1
max_depth = 20
sample_rate = 1.0

### *Model characterization*

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

```
#Train your model using all data and the best known parameters
GBM_model <- h2o.gbm(x = c("AGE", "RACE", "PSA", "GLEASON"),
                     y = "CAPSULE",
                     training_frame = train,
                     learn_rate = 0.1,
                     max_depth = 20,
                     sample_rate = 1.0,
```

```
                 ntrees   = 100,
                 seed = 2020)

print(GBM_model)
Output:
Model Details:
==============

H2OBinomialModel: gbm
Model ID:  GBM_model_R_1598310950573_1
Model Summary:
  number_of_trees number_of_internal_trees model_size_in_bytes min_depth
1             100                      100               32264         5
  max_depth mean_depth min_leaves max_leaves mean_leaves
1        16   10.04000         19         24    20.96000


H2OBinomialMetrics: gbm
** Reported on training data. **

MSE:  0.03192855
RMSE:  0.1786856
LogLoss:  0.1498627
Mean Per-Class Error:  0.009090909
AUC:  0.9996795
AUCPR:  0.9898817
Gini:  0.999359
R^2:  0.8653624

Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
         0   1    Error    Rate
0      162   3 0.018182  =3/165
1        0 104 0.000000  =0/104
Totals 162 107 0.011152  =3/269

Maximum Metrics: Maximum metrics at their respective thresholds
                        metric threshold      value idx
1                       max f1  0.389477   0.985782 104
2                       max f2  0.389477   0.994264 104
3                 max f0point5  0.469360   0.994094  99
4                 max accuracy  0.469360   0.988848  99
5                max precision  0.999511   1.000000   0
6                   max recall  0.389477   1.000000 104
7              max specificity  0.999511   1.000000   0
8             max absolute_mcc  0.389477   0.976878 104
9   max min_per_class_accuracy  0.430621   0.987879 102
10 max mean_per_class_accuracy  0.389477   0.990909 104
11                      max tns  0.999511 165.000000   0
12                      max fns  0.999511 103.000000   0
13                      max fps  0.001804 165.000000 265
14                      max tps  0.389477 104.000000 104
15                      max tnr  0.999511   1.000000   0
16                      max fnr  0.999511   0.990385   0
17                      max fpr  0.001804   1.000000 265
18                      max tpr  0.389477   1.000000 104

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

## *Making Predictions*

The Train/Test method train the models using the "train" data as the training data. So, to use the model to predict we have to train the model using all data (h2o_df) and the best-known parameters.

```
GBM_model <- h2o.gbm(x = c("AGE", "RACE", "PSA", "GLEASON"),
                     y = "CAPSULE",
                     training_frame = h2o_df,
                     learn_rate = 0.1,
                     max_depth = 20,
                     sample_rate = 1.0,
                     ntrees   = 100,
                     seed = 2020)
```

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

```
# Generate the predictions on a test set (if necessary):
pred <- h2o.predict(GBM_model, newdata = test)
pred
Output:
  predict        p0          p1
1       1 0.5893065 0.410693521
2       1 0.3543368 0.645663223
3       1 0.5505129 0.449487099
4       0 0.9556726 0.044327425
5       0 0.9905264 0.009473623
6       0 0.8623784 0.137621574

[111 rows x 3 columns]
```

---

## *BEST MODELS BY KFOLD METHOD*

### *Selected model*

The selected model is the Deep Learning-Neural Network (DL) algorithm with AUC = 0.7920589674930177 and Hyper-Parameter:

hidden = [1024]

### *Model characterization*

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

```
DL_model <- h2o.deeplearning(
                      # Algorithm
                      activation=c("RectifierWithDropout"),
                      epochs=500,
                      #adaptive_rate = FALSE,
                      hidden = c(1024),
```

```
                      input_dropout_ratio = 0.0,
                      # Response variable and predictors
                      y = "CAPSULE",
                      x = c("AGE", "RACE", "PSA", "GLEASON"),
                      # Training data
                      training_frame=  h2o_df,
                      # Preprocessed
                      standardize = TRUE,
                      missing_values_handling = "Skip",
                      ignore_const_cols = TRUE,
                      # Early stop
                      stopping_metric="AUC",
                      stopping_tolerance= 0.01,
                      stopping_rounds = 3,
                      # Regularization
                      l1=1e-5,
                      l2=1e-5,
                      seed = 2020)

print(DL_model)

Output:
Model Details:
==============

H2OBinomialModel: deeplearning
Model ID:  DeepLearning_model_R_1598718661326_672
Status of Neuron Layers: predicting CAPSULE, 2-class classification, bernoulli
distribution, CrossEntropy loss, 10.242 weights/biases, 135,0 KB, 190.000
training samples, mini-batch size 1
  layer units             type dropout       l1       l2 mean_rate rate_rms
1     1     7            Input  0.00 %       NA       NA        NA       NA
2     2  1024 RectifierDropout 50.00 % 0.000010 0.000010  0.193687 0.343031
3     3     2          Softmax      NA 0.000010 0.000010  0.009714 0.009128
  momentum mean_weight weight_rms mean_bias bias_rms
1       NA          NA         NA        NA       NA
2 0.000000   -0.052502   0.262709 -0.056548 0.279210
3 0.000000    0.008416   0.148099  0.056109 0.140763


H2OBinomialMetrics: deeplearning
** Reported on training data. **
** Metrics reported on full training frame **

MSE:  0.1659942
RMSE:  0.4074238
LogLoss:  0.4883401
Mean Per-Class Error:  0.2623161
AUC:  0.825329
AUCPR:  0.7819455
Gini:  0.6506579

Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
         0   1    Error      Rate
0      145  82 0.361233   =82/227
1       25 128 0.163399   =25/153
Totals 170 210 0.281579  =107/380

Maximum Metrics: Maximum metrics at their respective thresholds
                       metric threshold      value idx
1                      max f1  0.319374   0.705234 207
2                      max f2  0.134448   0.821545 304
```

```
3                  max f0point5  0.737796   0.724947  77
4                  max accuracy  0.554822   0.771053 136
5                 max precision  0.999995   1.000000   0
6                    max recall  0.036747   1.000000 339
7                max specificity  0.999995   1.000000   0
8               max absolute_mcc  0.554822   0.518132 136
9    max min_per_class_accuracy  0.443643   0.726872 171
10 max mean_per_class_accuracy  0.554822   0.754038 136
11                     max tns  0.999995 227.000000   0
12                     max fns  0.999995 152.000000   0
13                     max fps  0.000000 227.000000 376
14                     max tps  0.036747 153.000000 339
15                     max tnr  0.999995   1.000000   0
16                     max fnr  0.999995   0.993464   0
17                     max fpr  0.000000   1.000000 376
18                     max tpr  0.036747   1.000000 339

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

### *Making Predictions*

The KFold method trains the models using all data (h2o_df) as the training data. So we can use directly the selected model which already considers the best-known parameters.

```
# Best model chosen by AUC:
best_model <- h2o.getModel(dl_gridperf4@model_ids[[1]])
```

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

# Generate the predictions on a test set (if necessary):

```
pred <- h2o.predict(best_model, newdata = test)
pred
Output:
  predict         p0           p1
1       1 0.66265631 3.373437e-01
2       1 0.36200723 6.379928e-01
3       1 0.08907328 9.109267e-01
4       0 0.99992892 7.107996e-05
5       0 0.98164801 1.835199e-02
6       1 0.20583521 7.941648e-01

[111 rows x 3 columns]
```

## *FINAL WORDS*

This project is about a practical exposition of how to apply two classic Grid Search methods for tunning four of the most frequently used machine learning models for classification, by using Sparklyr and H2O-SWater.
The tune processes can obviously be optimized either by modifying the current hyperparameter values or by incorporating others. However, it should be noted that incorporating more hyperparameter into the tune will require a greater amount of process time and, in some cases, will not be feasible to carry out.

Data can be loaded directly into the H2O cluster, or -as was done in this project- by first loading it into memory in the R session and then transferring it. The second option is not recommended if the data volume is very large.

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.