***Hector Alvaro Rojas** | Data Science, Visualizations and Applied Statistics | May 11, 2020*

Big Data: K-Fold CV ML Classifiers with Apache Spark using Sparklyr

WHAT IS THIS ALL ABOUT?

This project is about an application of how to use K-Fold CV ML Classifiers with Apache Spark using Sparklyr. 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 functions and scripts used are proportionated, allowing the reader to consult the appropriate bibliography and tutorial examples clearly and directly.

Cross-validation is a statistical method used to estimate the skill of machine learning models. It is used in machine learning to compare and select a model because it is easy to understand, easy to implement, and results in skill estimates that generally have a lower bias than other methods.

Cross-validation evaluates predictive models by partitioning the original sample into a training set to train the model, and a test set to evaluate it. In particular, a good cross-validation method gives us a comprehensive measure of our model's performance throughout the whole dataset.

In k-fold cross-validation, the original sample is randomly partitioned into k equal size subsamples. Of the k subsamples, a single subsample is retained as the validation data for testing the model, and the remaining k-1 subsamples are used as training data. The cross-validation process is then repeated k times (the folds), with each of the k subsamples used exactly once as the validation data. The k results from the folds can then be averaged (or otherwise combined) to produce a single estimation. The advantage of this method is that all observations are used for both training and validation, and each observation is used for validation exactly once.

Aside from selection bias, cross-validation also helps us with avoiding over fitting. By dividing the dataset into a train and validation set, we can concretely check that our model performs well on data seen during training and not.

You can get an excellent clear practical explanation of this method 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.

This project has been developed using sparklyr considering a local Spark cluster environment.

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".

To use the selected models, we proceed to predict values for the dependent variable (y) based on new data that are not part of the base used to generate the respective models. This base can be downloaded from here.

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 AND PREPARING THE DATA

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(tidyverse) # to get the whole tidyverse: dplyr, ggplot2 tibble, readr,
tidyr, purrr
library(sparklyr)
library(caret)
library(e1071)

# 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, ...
```

PREPARING THE DATA

Changing characters to numbers

The first thing we need to do is turn all the factors into integers, as this is how MLlib likes it. Further, the labels for a categorical outcomes (variable "y" in y = f(x)) have to be in {0,1}. Finally, the data is copied to the Spark cluster to reflect a situation where we're dealing with data already in Spark.

```
df <- df_raw %>%
  map_if(is.factor, as.integer) %>%
  as_tibble() %>%
  mutate(y = y - 1) %>% # because labels must be in {0,1} for MLlib
  copy_to(sc, ., name = "df") # copy our R data frame onto spark
glimpse(df)
Observations: ??
Variables: 17
Database: spark_connection
$ age        30, 33, 35, 30, 59, 35, 36, 39, 41, 43, 39, 43, 36, 20, 3...
$ job        11, 8, 5, 5, 2, 5, 7, 10, 3, 8, 8, 1, 10, 9, 2, 5, 10, 1,...
$ marital    2, 2, 3, 2, 2, 3, 2, 2, 2, 2, 2, 2, 2, 3, 2, 2, 2, 3, 3, ...
$ education  1, 2, 3, 3, 2, 3, 3, 2, 3, 1, 2, 2, 3, 2, 2, 3, 2, 3, 1, ...
$ default    1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
$ balance    1787, 4789, 1350, 1476, 0, 747, 307, 147, 221, -88, 9374,...
$ housing    1, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1, 1, 2, 1, 1, 2, 2, ...
$ loan       1, 2, 1, 2, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 2, 2, 1, 1, 1, ...
$ contact    1, 1, 1, 3, 3, 1, 1, 1, 3, 1, 3, 1, 1, 1, 1, 1, 1, 1, 3, ...
$ day        19, 11, 16, 3, 5, 23, 14, 6, 14, 17, 20, 17, 13, 30, 29, ...
$ month      11, 9, 1, 7, 9, 4, 9, 9, 9, 1, 9, 1, 2, 1, 5, 2, 2, 1, 9,...
$ 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   4, 1, 1, 4, 4, 1, 2, 4, 4, 1, 4, 4, 4, 4, 1, 4, 4, 1, 4, ...
$ y          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, ...
```

The dimensions related to the dataset acan be easily gotten by using "sdf_dim()".

```
sdf_dim(df)
Output:
[1] 4521   17
```

How about the distribution of NA's?

```
# short way
any(is.na(df))
Output:
[1] FALSE
```

So there is no missing values in the dataset.

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         11, 8, 5, 5, 2, 5, 7, 10, 3, 8, 8, 1, 10, 9, 2, 5, 10, 1,...
$ marital     2, 2, 3, 2, 2, 3, 2, 2, 2, 2, 2, 2, 2, 3, 2, 2, 2, 3, 3, ...
$ education   1, 2, 3, 3, 2, 3, 3, 2, 3, 1, 2, 2, 3, 2, 2, 3, 2, 3, 1, ...
$ default     1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
$ balance     1787, 4789, 1350, 1476, 0, 747, 307, 147, 221, -88, 9374,...
$ housing     1, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1, 1, 2, 1, 1, 2, 2, ...
$ loan        1, 2, 1, 2, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 2, 2, 1, 1, 1, ...
$ contact     1, 1, 1, 3, 3, 1, 1, 1, 3, 1, 3, 1, 1, 1, 1, 1, 1, 1, 3, ...
$ day         19, 11, 16, 3, 5, 23, 14, 6, 14, 17, 20, 17, 13, 30, 29, ...
$ month       11, 9, 1, 7, 9, 4, 9, 9, 9, 1, 9, 1, 2, 1, 5, 2, 2, 1, 9,...
$ 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    4, 1, 1, 4, 4, 1, 2, 4, 4, 1, 4, 4, 4, 4, 1, 4, 4, 1, 4, ...
$ 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, ...
```

MODELING BY K FOLD CROSS VALIDATION (CV)

K FOLD CV USING USER-DEFINED FUNCTIONS

Getting K Fold CV for three models

The scripts are delivered only for the first model (RForest). For the rest, the respective model and its parameters must be adjusted accordingly.

Model: Random Forest (RForest)

10 Fold CV:

```
vfolds <- sdf_random_split(
  df,
  weights = purrr::set_names(rep(0.1, 10), paste0("fold", 1:10)),
  seed = 2020
)

cv_resultsRF <- purrr::map_df(1:10, function(v) {
  analysis_set <- do.call(rbind, vfolds[setdiff(1:10, v)]) %>% compute()
  assessment_set <- vfolds[[v]]
  model <- ml_random_forest_classifier(
    analysis_set, y ~ ., impurity = "gini"
  )
  s <- ml_predict(model, assessment_set)
  Accuracy <-  round(ml_multiclass_classification_evaluator(s, metric_name =
"accuracy"),8)
  F1 <- round(ml_multiclass_classification_evaluator(s, metric_name = "f1"),8)
  WPrecision <- round(ml_multiclass_classification_evaluator(s, metric_name =
"weightedPrecision"),8)
  WRecall <- round(ml_multiclass_classification_evaluator(s, metric_name =
"weightedRecall"),8)

  tibble(
    Resample = paste0("Fold", stringr::str_pad(v, width = 2, pad = "0")),
    Accuracy = round(Accuracy,8),
    F1 = round(F1,8),
    WeightedPrecision = round(WPrecision,8),
    WeightedRecall = round(WRecall,8)
  )
})
# A tibble: 10 x 5
   Resample Accuracy    F1 WeightedPrecision WeightedRecall

 1 Fold01      0.865 0.818             0.840          0.865
 2 Fold02      0.901 0.867             0.898          0.901
 3 Fold03      0.913 0.876             0.879          0.913
 4 Fold04      0.878 0.837             0.828          0.878
 5 Fold05      0.888 0.850             0.855          0.888
 6 Fold06      0.873 0.823             0.889          0.873
 7 Fold07      0.900 0.872             0.874          0.900
 8 Fold08      0.898 0.866             0.854          0.898
 9 Fold09      0.887 0.854             0.873          0.887
10 Fold10      0.886 0.847             0.848          0.886
```

Getting mean of accuracy:

```
# mean of accuracy:
mean(cv_resultsRF$Accuracy, na.rm=TRUE)
Output:
[1] 0.8887489
```

Model: Decision Tree (DTree)

10 Fold CV:

```
# A tibble: 10 x 5
   Resample Accuracy    F1 WeightedPrecision WeightedRecall

 1 Fold01      0.869 0.851             0.847          0.869
 2 Fold02      0.897 0.885             0.881          0.897
 3 Fold03      0.913 0.903             0.898          0.913
 4 Fold04      0.864 0.838             0.823          0.864
 5 Fold05      0.892 0.871             0.869          0.892
 6 Fold06      0.866 0.837             0.836          0.866
 7 Fold07      0.881 0.868             0.860          0.881
 8 Fold08      0.902 0.889             0.882          0.902
 9 Fold09      0.881 0.862             0.858          0.881
10 Fold10      0.886 0.875             0.869          0.886
```

Getting mean of accuracy:

```
# mean of accuracy:
mean(cv_resultsDT$Accuracy, na.rm=TRUE)
```

```
Output:
[1] 0.8849242
```

Model: Logistic Regression (RLogistic)
10 Fold CV:

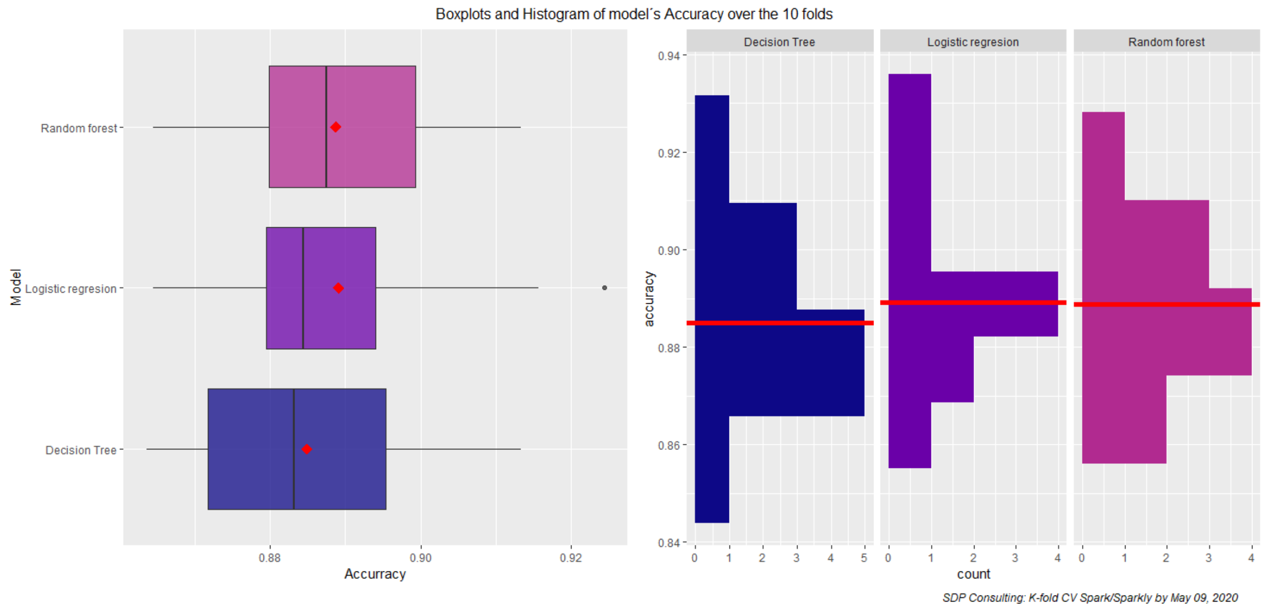
```
# A tibble: 10 x 5
   Resample Accuracy    F1 WeightedPrecision WeightedRecall

 1 Fold01      0.865 0.831             0.834          0.865
 2 Fold02      0.886 0.859             0.853          0.886
 3 Fold03      0.916 0.894             0.893          0.916
 4 Fold04      0.883 0.860             0.855          0.883
 5 Fold05      0.888 0.860             0.858          0.888
 6 Fold06      0.873 0.832             0.856          0.873
 7 Fold07      0.883 0.860             0.848          0.883
 8 Fold08      0.924 0.912             0.914          0.924
 9 Fold09      0.896 0.872             0.886          0.896
10 Fold10      0.879 0.852             0.842          0.879
```

Getting mean of accuracy:

```
# mean of accuracy:
mean(cv_resultsRL$Accuracy, na.rm=TRUE)
Output:
[1] 0.8891277
```

Result's Visualization

Below is a visualization of the accuracy through the 10 fold for each of the models. The visualization contemplates the box-plot and histogram graphs. For graphics, the ggplot2 package is used. The average is highlighted in red in each case.



K FOLD CV USING BUILT INTO SPARK / SPARKLYR FUNCTIONS

The cross-validation functions built into Spark / sparklyr are designed for parameter tuning. The in-built functions are designed to be more effective by using Spark pipelines. Pipelines describe several stages we want to apply to input data, such as feature transformations and fitting a model.

To develop this fundamental part of the project we will base ourselves on the rf model.

Getting K Fold CV with almost not tuning

We can use the in-built spark / sparklyr functions to get a cross-validated estimate of performance for a single model. Now we get away how to do so.

CV process:

```
# create pipeline
pipeline <- ml_pipeline(sc) %>%
  ft_r_formula(y ~ .) %>%
  ml_random_forest_classifier()

# Create grid [Specify hyperparameter grid]
# As we only want to fit a single model, we can just provide a single grid of
parameters containing the default values.
grid <-
  list(random_forest = list(impurity = "gini"))

# create cross validator [Create the cross validator object]
# We are not actually doing the cross-validation here, just specifying its
parameters.
cv <- ml_cross_validator(
  sc,
  estimator = pipeline, # use our pipeline to estimate the model
  estimator_param_maps = grid, # use the params in grid
  evaluator = ml_multiclass_classification_evaluator(sc, metric_name =
"accuracy"), # how to evaluate the CV
  num_folds = 10, # number of CV folds
  seed = 2020
)
```

So far, till now we have not carried ot a CV method yet. We've only just laid out a number of step that we ultimately want to be carried out.
1. **ml_pipeline(sc)**: create a pipeline associated with our Spark connection (sc)
2. **ft_r_formula**: use the Spark equivalent of this R formula when it comes to fitting the model
3. **ml_random_forest_classifier**: use random forest when fitting the model
4. **grid**: these are parameters to try out when doing the cross-validation
5. **ml_cross_validator**: create a cross-validator where the estimator is our pipeline, combining a model formula and algorithm. Try out the parameters in grid and use mml_multiclass_classification_evaluator to evaluation performance using "accuracy" measure.

Now we have all the steps done. Then, we can carry out the cross-validation.

```
# fit models with k-fold cv [Train the models]
cv_model <- ml_fit(cv, df)
names(cv_model)
[1] "uid"                  "param_map"             "estimator"
 [4] "evaluator"            "estimator_param_maps" "best_model"
 [7] "num_folds"            "metric_name"           "avg_metrics"
[10] "avg_metrics_df"       "sub_models"            ".jobj"
# get metric
cv_model$avg_metrics_df
Output:
   accuracy impurity_1
```

```
1 0.8907471       gini
```

Predict new values on another dataframe:Predict new values on another dataframe:

```
#Read new data
NewData <- read.csv("data/bank-NewData.csv") %>%
  map_if(is.character, as.factor) %>%
  as_tibble()

# all the factors into integers and the labels for a categorical outcomes in
{0,1}
NewData <- NewData %>%
  map_if(is.factor, as.integer) %>%
  as_tibble() %>%
  copy_to(sc, ., name = "NewData", overwrite = TRUE) # copy our R data frame onto
spark

glimpse(NewData)
Observations: ??
Variables: 16
Database: spark_connection
$ job         1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
$ marital     1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, ...
$ education   2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, ...
$ default     1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
$ balance     -427, 0, 362, 512, 286, 0, 51, 67, 168, 190, 205, 404, 56...
$ housing     2, 2, 1, 1, 1, 2, 2, 1, 2, 1, 1, 2, 2, 2, 1, 2, 1, 2, 2, ...
$ loan        1, 2, 1, 1, 1, 1, 2, 2, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, ...
$ contact     1, 1, 1, 1, 1, 1, 1, 3, 1, 1, 1, 3, 1, 1, 1, 1, 1, 1, 3, ...
$ day         13, 14, 13, 4, 5, 24, 14, 20, 7, 27, 4, 29, 4, 29, 5, 4, ...
$ month       9, 9, 2, 4, 2, 1, 5, 7, 9, 2, 2, 9, 9, 5, 2, 2, 2, 10, 9,...
$ duration    8, 17, 529, 275, 100, 83, 203, 70, 249, 276, 180, 539, 19...
$ campaign    5, 7, 1, 1, 3, 2, 1, 5, 2, 3, 1, 1, 3, 1, 2, 1, 2, 1, 1, ...
$ pdays       -1, -1, -1, -1, -1, -1, 87, -1, 169, -1, 92, -1, -1, 245,...
$ previous    0, 0, 0, 0, 0, 0, 2, 0, 1, 0, 3, 0, 0, 2, 0, 0, 0, 0, 0, ...
$ poutcome    4, 4, 4, 4, 4, 4, 2, 4, 1, 4, 3, 4, 4, 2, 4, 4, 4, 4, 4, ...
$ age         1, 1, 2, 1, 3, 1, 1, 2, 2, 3, 2, 1, 1, 0, 2, 3, 2, 2, 1, ...
# get prediction done
df %>%
        ml_random_forest_classifier(y ~ .,impurity = "gini") %>%
        ml_predict(NewData) %>%
        select(prediction)
Output:
# Source: spark [?? x 1]
   prediction

 1           0
 2           0
 3           0
 4           0
 5           0
 6           0
 7           0
 8           0
 9           0
10           0
# ... with more rows
```

Getting K Fold CV with more explicit tuning

This is the more efficient way to occupy the in-built spark / sparklyr functions to get cross-validated modeling done. For instance, the specification of hyperparameter in the grid lets us get more efficient results or, at least, to try more options and combinations to get those done.

CV process:

```
# create pipeline
pipeline <- ml_pipeline(sc) %>%
  ft_r_formula(y ~ .) %>%
  ml_random_forest_classifier()

# create grid [Specify hyperparameter grid]
grid <- list(
  random_forest = list(
    num_trees = c(5,10),
    max_depth = c(5,10),
    impurity = c("entropy", "gini")
  )
)

# create cross validator [Create the cross validator object]
cv <- ml_cross_validator(
  sc,
  estimator = pipeline, # use our pipeline to estimate the model
  estimator_param_maps = grid, # use the params in grid
  evaluator = ml_multiclass_classification_evaluator(sc, metric_name =
"accuracy"), # how to evaluate the CV
  num_folds = 10, # number of CV folds
  seed = 2020
)

# fit models with k-fold cv [Train the models]
cv_model <- ml_fit(cv, df)
# get metric
cv_model$avg_metrics_df
Output:
   accuracy impurity_1 num_trees_1 max_depth_1
1 0.8901779    entropy           5           5
2 0.8902403    entropy          10           5
3 0.8908325    entropy           5          10
4 0.8978620    entropy          10          10
5 0.8913678       gini           5           5
6 0.8915958       gini          10           5
7 0.8923794       gini           5          10
8 0.8940359       gini          10          10
# get selected model parameters
max(ml_validation_metrics(cv_model)[[1]])  # max(
ml_validation_metrics(cv_model)$accuracy)
Output:
[1] 0.897862
```

Then, model "4" is the one selected.

Predict new values on another dataframe:Predict new values on another dataframe:

```
# get prediction done
df %>%
        ml_random_forest_classifier(y ~ ., impurity = "entropy", num_trees = 10 ,
max_depth = 10) %>%
        ml_predict(NewData) %>%
        select(prediction)
Output:
```

```
# Source: spark [?? x 1]
   prediction

 1          0
 2          0
 3          0
 4          0
 5          0
 6          0
 7          0
 8          0
 9          0
10          0
```

FINAL WORDS

In this project, an application of how to use k-fold CV ML Classifiers with Apache Spark using Sparklyr has been carried out. Here, we use 10 folds and we managed examples by emplying Random Forest, Decision Tree, and Logistic Regression models
.
The cross-validation function built into Spark / sparklyr (in-built cross-validation function) is a lot faster than the user-defined function. However, it does not return performance metrics for each fold yet. Till today, we can just get an overall average. Anyway, if we were fitting a lot of models we'd have to use the in-built functions for performance reasons.

Also, since the advent of spark 2.3 and sparklyr 0.8 parallel cross-validation is supported. This, for sure, should generate considerable performance gains, if you have access to a computer cluster environment. This project was carried out on a spark 2.3.3 and sparklyr 1.2.0.

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.