```r
# Script:
#   des algorithmes Random Forest (RF) Gradient Boosting (GBM),
#   extrem Gradient Boosting (XGB) et Deep learning (DL) Sous H2O avec R

# Description:
# Ce script est dédié au développement des algorithme RF, GBM, XGB et DL,
# le développement de ces algorithme suit étapes suivant:
#  1- Importation du Paquet H2O et le démarré
#  2- Importation des données et spécification des variables
#     indépendants et dépendant;
#  3- Création de la fonction d'évaluation
#      (Pour calculer les indice de pérformance);
#  4- Développement et l'évaluation des 4 algorithmes:
#      4-1) Avec les paramètres par défaut;
#      4-2) Avec les paramètres optimisé par RandomDiscrit
#    (Random Searche);
#      4-3) Avec les paramètres optimisé par Cartesien
#    (Grid Search).

# Version:
#     Mohammed AMEKSA:Juin 2019Script Original
```

2.1 Importer la bibliothèque H2O et le démarré

2.2 Puis l'importation des données

```r
library(h2o)
# Démarage de h2o
h2o.init(
  ip = 'localhost', # addresse local du machine
  port = 54321,
  nthreads = -1,      ## -1: pour l'utilisation de tous les threads
  max_mem_size = "8g" ## Spécification de la taille mémoire H2O cloud
)
h2o.removeAll() # Au cas où le cluster fonctionnait déjà

##importer les deux fichiers de données
#spécifier le chemine ou se trouvent
meteo_train_path <- "Stage DMN/Train-Equil-Lon-Lat-Hour-Month-RedVisi.csv"
meteo_test_path <- "Stage DMN/Test-Equil-Lon-Lat-Hour-Month-RedVisi.csv"

# importer les données de disque au h2o
meteo_train<-h2o.importFile(path = meteo_train_path,
              destination_frame = "meteo_train.hex")
meteo_test<-h2o.importFile(path = meteo_test_path,
             destination_frame = "meteo_test.hex")

# variable target (Visibilite)
y.dep <- 35
```

```
# variables independent
x.indep <- c(1:34)
```

2.3 Création des fonctions d'évaluation

```
Evaliation <- function(model) {
    print("-----  H2o model : -----")
    print("Comparaison de la performance pour les donnée Train & Test")
    PredTest <- as.data.frame(h2o.predict(model, meteo_test))
    PredTrain <- as.data.frame(h2o.predict(model, meteo_train))

    ActualTrain=as.data.frame(meteo_train['Visibilite'])
    ActualTest=as.data.frame(meteo_test['Visibilite'])

    TestbothPA=cbind(PredTest,ActualTest)
    BIAS <- round(mean(TestbothPA[,1] - TestbothPA[,2]),4)
    RMSE <-  round(sqrt(mean((TestbothPA[,1] - TestbothPA[,2])^2)),4)
    MAE <- round(mean(abs(TestbothPA[,1] - TestbothPA[,2])),4)
    CC <- round(cor(TestbothPA[,1],TestbothPA[,2]),4)
    set <- "Test"
    errors.test <- data.frame(set,BIAS,RMSE,MAE,CC)

    TrainbothPA=cbind(PredTrain,ActualTrain)
    BIAS <- round(mean(TrainbothPA[,1] - TrainbothPA[,2]),4)
    RMSE <-  round(sqrt(mean((TrainbothPA[,1] - TrainbothPA[,2])^2)),4)
    MAE <- round(mean(abs(TrainbothPA[,1] - TrainbothPA[,2])),4)
    CC <- round(cor(TrainbothPA[,1],TrainbothPA[,2]),4)
    set <- "Train"
    errors.train <- data.frame(set,BIAS,RMSE,MAE,CC)

    Eval.metrics=rbind(errors.train,errors.test)
    print('====================================')
    print(Eval.metrics)

    print("----- H2o model : check variable importance -----")
    #check variable importance
    VarImpo <- h2o.varimp(model)
    #print(VarImpo)
    h2o.varimp_plot(model)
}
```

2.4 1- Méthodes Ensemblistes

2.5 Random Forest

2.5.1 Par défaut

```
# Création du modèle RF par défauts
rforest.defaut <- h2o.randomForest(y=y.dep,
```

```
    x=x.indep,
    training_frame = meteo_train,
    validation_frame = meteo_test,
    model_id="rf_defaut",
    seed = 42, )
# évaluation du modèle
Evaliation(rforest.defaut)
```

2.5.2 Optimisation des hyperparamètres avec Grid Search et Random Search

les hyperparamètres les plus importants choisis pour Random forest sont:
* ntrees = nombre d'arbres,
* mtries = le nombre de colonnes à sélectionner au hasard à chaque niveau,
* max_depth = profondeur maximale de chaque arbre,
* min_rows = le nombre minimum d'observations pour une feuille,

```
rf_params <- list(ntrees = seq(50, 100, 10),
   max_depth = seq(10, 17, 1),
   min_rows= seq(1, 5, 1),
   mtries = seq(-1, 3, 1))
search_criteria <- list(strategy = "RandomDiscrete",
   max_models = 100)
```

Random Search

```
rf_RandomDiscrete <- h2o.grid("drf",
                y=y.dep,
                x=x.indep,
                grid_id = "rf_RandomDiscrete",
                training_frame = meteo_train,
                validation_frame = meteo_test,
                seed = 1,
               hyper_params = rf_params,
                search_criteria = search_criteria)
# on trie les résultats par mae
perf_rf_RandomDiscrete <- h2o.getGrid(grid_id = "rf_RandomDiscrete",
sort_by = "mae",
decreasing = FALSE)
# Puis on garder le premier (c-à-d qu'il a le minimum des erreurs)
rf_RandomDiscrete_selectione = h2o.getModel(perf_rf_RandomDiscrete@model_ids[[1]])
# et on évaluer le modèle
Evaliation(rf_RandomDiscrete_selectione)
```

Grid Search

```
# gride search et par défauts
rf_Cartesient <- h2o.grid("drf", x=1:34, y=35,
             grid_id = "rf_Cartesient",
             training_frame = meteo_train,
```

```
              validation_frame = meteo_test,
              seed = 1,
              hyper_params = rf_params)
 # on trie les résultats par mae
 perf_rf_Cartesient <- h2o.getGrid(grid_id = "rf_Cartesient",
                                   sort_by = "mae",
                                   decreasing = FALSE)
 # Puis on garder le premier (c-à-d qu'il a le minimum des erreurs)
 rf_Cartesient_selectione = h2o.getModel(perf_rf_Cartesient@model_ids[[1]])
 # et on évaluer le modèle
 Evaliation(rf_Cartesient_selectione)
```

2.6 Gradient Boosting Machine

2.6.1 Par défaut

```
 gbm.defaut <- h2o.gbm(y=y.dep,
                x=x.indep,
                training_frame = meteo_train,
                validation_frame = meteo_test,
                model_id="gbm_defaut",
                seed = 42, )
 # évaluation du modèle
 Evaliation(gbm.defaut)
```

2.6.2 Optimisation des hyperparamètres avec Grid Search et Random Search

les hyperparamètres les plus importants choisis pour Gradient Boosting Machine sont: * ntrees = nombre d'arbres, * max_depth = profondeur maximale de chaque arbre, * sample_rate = spécifier le taux d'échantillonnage de la ligne (sans remplacement). * min_rows = le nombre minimum d'observations pour une feuille, * learn_rate = taux d'apprentissage

```
 gbm_params <- list(ntrees = seq(200, 400 , 50),
max_depth = seq(11, 17, 2),
sample_rate = c(0.5, 0.8, 0.95, 1.0),
min_rows = c(2, 5, 10),
learn_rate = c(0.1))
 search_criteria <- list(strategy = "RandomDiscrete",
    max_models = 100)
```

Random Search

```
 gbm_RandomDiscrete <- h2o.grid("gbm", x=x.indep, y=y.dep,
  grid_id = "gbm_RandomDiscrete",
  training_frame = meteo_train,
  validation_frame = meteo_test,
  seed = 1,
  hyper_params = gbm_params,
  search_criteria = search_criteria,
```

```
 stopping_tolerance = 0.001,
 stopping_rounds=3,
 score_tree_interval = 10 )
# on trie les résultats par mae
perf_gbm_RandomDiscrete <- h2o.getGrid(grid_id = "gbm_RandomDiscrete",
 sort_by = "mae",
 decreasing = FALSE)
# Puis on garder le premier (c-à-d qu'il a le minimum des erreurs)
gbm_gs_selected=h2o.getModel(perf_gbm_RandomDiscrete@model_ids[[1]])
# et on évaluer le modèle
Evaliation(gbm_gs_selected)
```

Grid Search

```
# Train and validate a grid of GBM
gbm_cartesien<- h2o.grid("gbm", x=x.indep, y=y.dep,
 grid_id = "gbm_cartesien",
 training_frame = meteo_train,
 validation_frame = meteo_test,
 seed = 1,
 hyper_params = gbm_params)
# on trie les résultats par mae
perf_gbm_cartesien <- h2o.getGrid(grid_id = "gbm_cartesien",
 sort_by = "mae",
 decreasing = FALSE)
# Puis on garder le premier (c-à-d qu'il a le minimum des erreurs)
gbm_cartesien=h2o.getModel(perf_gbm_random_search@model_ids[[1]])
# et on évaluer le modèle
Evaliation(gbm_cartesien)
```

2.7 eXtrem Gradient Boosting

```
# Pour xgboost nous avons le développé dans une autre machine
# pour cela nous avons lancé h2o dans une machine
# Puis nous avons fait l'accé a cette machine via l'adresse IP
library(h2o)
#Start H2O
h2o.init(
  ip = 192.168.132.128,
  port = 54321,
  nthreads = -1,
  max_mem_size = "8g"
)
h2o.removeAll()

# les chemines des deux fichiers de données
meteo_train_path <- "C:/Users/Stage DMN/Train-Equil-Lon-Lat-Hour-Month-RedVisi.csv"
meteo_test_path <- "C:/Users/Stage DMN/Test-Equil-Lon-Lat-Hour-Month-RedVisi.csv"
```

```
 # puisque nous avons lancé h2o dans une autre machine
 # ici nous avons utilisé uploadFile au lieu d'importFile
 meteo_train<-h2o.uploadFile(path = meteo_train_path,
 destination_frame = "meteo_train.hex")
 meteo_test<-h2o.uploadFile(path = meteo_test_path,
destination_frame = "meteo_test.hex")

 # variable target (Visibilite)
 y.dep <- 35
 # variables independents
 x.indep <- c(1:34)
```

2.7.1 Par défaut

```
 xgb.defaut<-h2o.xgboost(y=y.dep,
  x=x.indep,
  training_frame =train,
  validation_frame=test,
  model_id="Xgboost_defaut",
  seed = 1)
 # évaluation du modèle
 Evaliation(xgb.defaut)
```

2.7.2 Optimisation des hyperparamètres avec Grid Search et Random Search

les hyperparamètres les plus importants choisis pour Gradient Boosting Machine sont: * max_depth = profondeur maximale de chaque arbre * min_rows = le nombre minimum d'observations pour une feuille, * sample_rate = spécifier le taux d'échantillonnage de la ligne (axe des x) (sans remplacement). * col_sample_rate et col_sample_rate_per_tree

```
 xgb_params <- list(max_depth = c(5, 10, 17),
   min_rows = c(2, 5, 11),
   sample_rate = c(0.5, 0.8, 0.95, 1.0),
   col_sample_rate = c(0.5, 0.8, 0.95, 1.0),
   col_sample_rate_per_tree = c(0.8, 0.99, 1.0))
 xgb_search_criteria <- list(strategy = "RandomDiscrete",)
```

Random Search

```
 xgb_RandomDiscrete = h2o.grid(grid_id ="xgb_randomDiscrete",
x=x.indep, y=y.dep,
hyper_params = xgb_params,
search_criteria = xgb_search_criteria,
training_frame = meteo_train,
validation_frame = meteo_test,

stopping_tolerance = 0.001,
stopping_rounds=3,
score_tree_interval = 10,
```

```
ntrees = 400)
 # on trie les résultats par mae
 xgb_perf_RandomDiscrete <- h2o.getGrid(grid_id = "xgb_randomDiscrete",
  sort_by = "mae",
  decreasing = FALSE)
 # Puis on garder le premier (c-à-d qu'il a le minimum des erreurs)
 RS_xgb <- h2o.getModel(xgb_perf_RandomDiscrete@model_ids[[1]])
 # évaluation du modèle
 Evaliation(RS_xgb)
```

Grid Search

```
 xgb_cartesien = h2o.grid(grid_id ="xgb_cartesien",
x=x.indep, y=y.dep,
hyper_params = xgb_params,
training_frame = meteo_train,
validation_frame = meteo_test,

stopping_tolerance = 0.001,
stopping_rounds=3,
score_tree_interval = 10,
ntrees = 400)
 # on trie les résultats par mae
 xgb_perf_cartesien <- h2o.getGrid(grid_id = "xgb_cartesien",
  sort_by = "mae",
  decreasing = FALSE)
 # Puis on garder le premier (c-à-d qu'il a le minimum des erreurs)
 GS_xgb <- h2o.getModel(xgb_perf_cartesien@model_ids[[1]])
 # évaluation du modèle
 Evaliation(GS_xgb)
```

2.8 2- Apprentissage profond

2.8.1 Par défaut

```
 dlearning.defaut <- h2o.deeplearning(model_id="dlearning_defaut",
   x=x.indep, y=y.dep,
     training_frame =meteo_train,
     validation_frame = meteo_test,
     model_id="dlearning_defaut", )
 # évaluation du modèle
 Evalution(dlearning.defaut)
```

2.8.2 Manuellement

```
 dlearning_manuel <- h2o.deeplearning(model_id="dlearning_manuel",
   x=x.indep, y=y.dep,
     training_frame =meteo_train,
     validation_frame = meteo_test,
```

```
    epoch = 100,
    hidden = c(68,68), # 2 couches cachées
    activation = "Rectifier", )
# évaluation du modèle
Evaliation(dlearning_manuel)
```