# MODEL 1 - GRADIENT BOOSTING

FENDAWN F. RECENTES

12/14/2022

## HELPER PACKAGES

```
library(dplyr)
```

```
## Warning: package 'dplyr' was built under R version 4.1.3

##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(gbm)
```

```
## Warning: package 'gbm' was built under R version 4.1.3

## Loaded gbm 2.1.8.1
```

```
library(h2o)
```

```
## Warning: package 'h2o' was built under R version 4.1.3

##
## ----------------------------------------------------------------------
##
## Your next step is to start H2O:
##     > h2o.init()
##
## For H2O package documentation, ask for help:
##     > ??h2o
##
## After starting H2O, you can use the Web UI at http://localhost:54321
## For more information visit https://docs.h2o.ai
##
## ----------------------------------------------------------------------
```

```
##
## Attaching package: 'h2o'

## The following objects are masked from 'package:stats':
##
##     cor, sd, var

## The following objects are masked from 'package:base':
##
##     %*%, %in%, &&, ||, apply, as.factor, as.numeric, colnames,
##     colnames<-, ifelse, is.character, is.factor, is.numeric, log,
##     log10, log1p, log2, round, signif, trunc
```

```
library(xgboost)
```

```
## Warning: package 'xgboost' was built under R version 4.1.3

##
## Attaching package: 'xgboost'

## The following object is masked from 'package:dplyr':
##
##     slice
```

```
library(modeldata)
```

```
## Warning: package 'modeldata' was built under R version 4.1.3
```

```
library(rsample)
```

```
## Warning: package 'rsample' was built under R version 4.1.3
```

```
library(recipes)
```

```
##
## Attaching package: 'recipes'

## The following object is masked from 'package:stats':
##
##     step
```

```
library(caret)
```

```
## Warning: package 'caret' was built under R version 4.1.3

## Loading required package: ggplot2

## Warning: package 'ggplot2' was built under R version 4.1.3

## Loading required package: lattice
```

```r
library(ROCR)
```

```
## Warning: package 'ROCR' was built under R version 4.1.3
```

```r
library(pROC)
```

```
## Warning: package 'pROC' was built under R version 4.1.3

## Type 'citation("pROC")' for a citation.

##
## Attaching package: 'pROC'

## The following object is masked from 'package:h2o':
##
##     var

## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```r
library(readr)
```

```
## Warning: package 'readr' was built under R version 4.1.3
```

```r
library(tidyverse)
```

```
## Warning: package 'tidyverse' was built under R version 4.1.3

## -- Attaching packages --------------------------------------- tidyverse 1.3.2 --

## v tibble  3.1.8     v stringr 1.4.1
## v tidyr   1.2.0     v forcats 0.5.2
## v purrr   0.3.4

## Warning: package 'tibble' was built under R version 4.1.3

## Warning: package 'tidyr' was built under R version 4.1.3

## Warning: package 'purrr' was built under R version 4.1.3

## Warning: package 'stringr' was built under R version 4.1.3

## Warning: package 'forcats' was built under R version 4.1.3

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter()  masks stats::filter()
## x stringr::fixed() masks recipes::fixed()
## x dplyr::lag()     masks stats::lag()
## x purrr::lift()    masks caret::lift()
## x xgboost::slice() masks dplyr::slice()
```

```
library(bestNormalize)
```

```
## Warning: package 'bestNormalize' was built under R version 4.1.3
```

### Load and view radiomatics dataset

```
radiomics = read_csv("C:\\Users\\MSU-TCTO OVCAA\\Documents\\radiomics_completedata.csv")
```

```
## Rows: 197 Columns: 431
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr   (1): Institution
## dbl (430): Failure.binary, Failure, Entropy_cooc.W.ADC, GLNU_align.H.PET, Mi...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

## DATA PRE-PROCESSING

### Check for null and missing values

```
sum(is.na(radiomics))
```

```
## [1] 0
```

### Check for normality

```
df <- radiomics%>%select_if(is.numeric)
df <- df[,-1]
test_df <- apply(df,2,function(x){shapiro.test(x)})
```

### Convert a list to vector

```
pvalue_list <- unlist(lapply(test_df, function(x) x$p.value))
```

### Compute and identify variables that are not normally distributed

### We have 428 variables that are not normally distributed

```
sum(pvalue_list < 0.05)
```

```
## [1] 428
```

### We have one variable that is normally distributed and that is Entropy_cooc.W.ADC

```
sum(pvalue_list > 0.05)
```

```
## [1] 1
```

```
which.max(pvalue_list)
```

```
## Entropy_cooc.W.ADC
##                  2
```

### Test for normality, the variable Entropy_cooc.W.ADC is normally distributed

```
shapiro.test(radiomics$Entropy_cooc.W.ADC)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  radiomics$Entropy_cooc.W.ADC
## W = 0.98903, p-value = 0.135
```

### To normalized the data, we remove the categorical, binary and Entropy_cooc.W.ADC variable

```
newdf1 = radiomics[,c(3,5:length(names(radiomics)))]

newdf1 = apply(newdf1,2,orderNorm)
newdf1 = lapply(newdf1, function(x) x$x.t)
newdf1 = newdf1%>%as.data.frame()

test_newdf1 = apply(newdf1,2,shapiro.test)
pval_list2 = unlist(lapply(test_newdf1, function(x) x$p.value))
```

```
sum(pval_list2 < 0.05)
```

```
## [1] 0
```

```
sum(pval_list2 > 0.05)
```

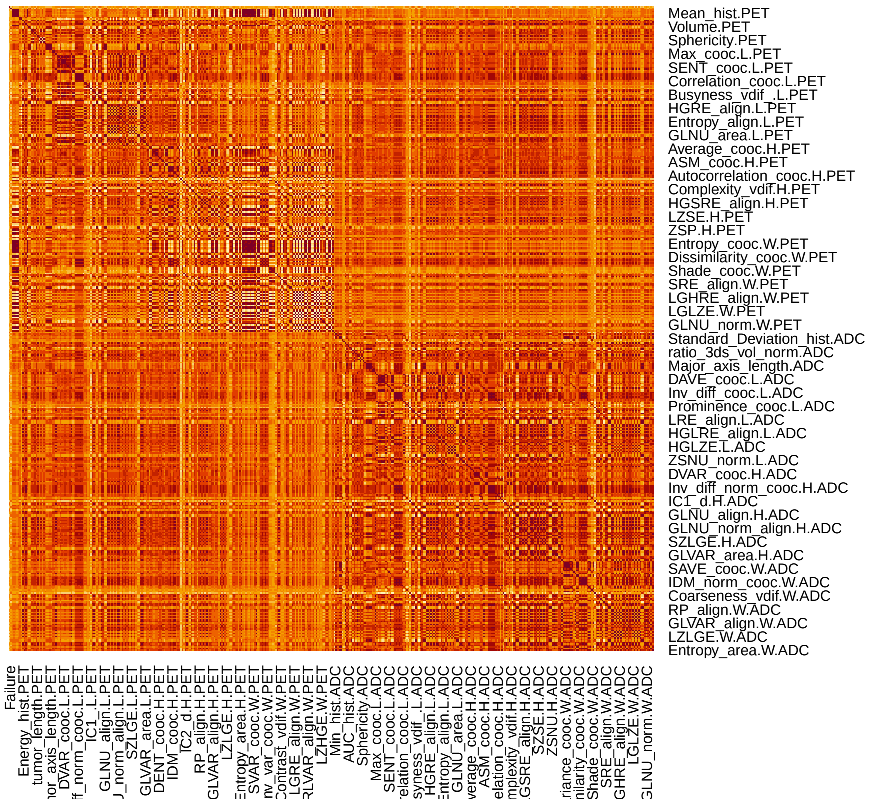
```
## [1] 428
```

### New data is created

```
newdata = select(radiomics, c("Failure.binary",  "Entropy_cooc.W.ADC"))
new_radiomics = cbind(newdata,newdf1)
```

### Get the correlation of the whole data except the categorical variables

```
CorMatrix=cor(new_radiomics[,-c(1,2)])
heatmap(CorMatrix,Rowv=NA,Colv=NA,scale="none",revC = T)
```



## DATA PREPARATION AND SPLITTING

## We use here the new normalized radiomics dataset

### Load new normalized radiomics data

```
radiomics = read_csv("C:\\Users\\MSU-TCTO OVCAA\\Documents\\normalRad.csv")
```

```
## Rows: 197 Columns: 431
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr   (1): Institution
## dbl (430): Failure.binary, Failure, Entropy_cooc.W.ADC, GLNU_align.H.PET, Mi...
##
## i Use ‘spec()‘ to retrieve the full column specification for this data.
## i Specify the column types or set ‘show_col_types = FALSE‘ to quiet this message.
```

### Split the data intro training (80%) and testing (20%) stratified in Failure.binary column

```
set.seed(123)
split <- initial_time_split(radiomics, prop = 0.8, strata = "Failure.binary")
radiomics_train <- training(split)
radiomics_test <- testing(split)
```

### Training the dataset

```
xgb_prep <- recipe(Failure.binary~ ., data = radiomics_train) %>%
  step_integer(all_nominal()) %>%
  prep(training = radiomics_train, retain = TRUE) %>%
  juice()

X <- as.matrix(xgb_prep[setdiff(names(xgb_prep), "Failure.binary")])
Y <- xgb_prep$Failure.binary
```

### Hypergrid parameter

```
hyper_grid <- expand.grid(
  eta = 0.01,
  max_depth = 3,
  min_child_weight = 3,
  subsample = 0.5,
  colsample_bytree = 0.5,
  gamma = c(0, 1, 10, 100, 1000),
  lambda = c(0, 1e-2, 0.1, 1, 100, 1000, 10000),
  alpha = c(0, 1e-2, 0.1, 1, 100, 1000, 10000)
)
```

### Grid Search

```
for(i in seq_len(nrow(hyper_grid))) {
  set.seed(123)
  m <- xgb.cv(
    data = X,
    label = Y,
    nrounds = 100,
    objective = "binary:logistic",
    early_stopping_rounds = 5,
    nfold = 2,
    verbose = 0,
    params = list(
      eta = hyper_grid$eta[i],
      max_depth = hyper_grid$max_depth[i],
      min_child_weight = hyper_grid$min_child_weight[i],
      subsample = hyper_grid$subsample[i],
      colsample_bytree = hyper_grid$colsample_bytree[i],
      gamma = hyper_grid$gamma[i],
      lambda = hyper_grid$lambda[i],
      alpha = hyper_grid$alpha[i]
    )
  )
  hyper_grid$rmse[i] <- min(m$evaluation_log$test_rmse_mean)
  hyper_grid$trees[i] <- m$best_iteration
}

hyper_grid %>%
  filter(rmse > 0) %>%
  arrange(rmse) %>%
  glimpse()

# optimal parameter list
params <- list(
  eta = 0.01,
  max_depth = 3,
  min_child_weight = 3,
  subsample = 0.5,
  colsample_bytree = 0.5
)
```

### Optimal parameter list

```
params <- list(
  eta = 0.01,
  max_depth = 3,
  min_child_weight = 3,
  subsample = 0.5,
  colsample_bytree = 0.5
)
```

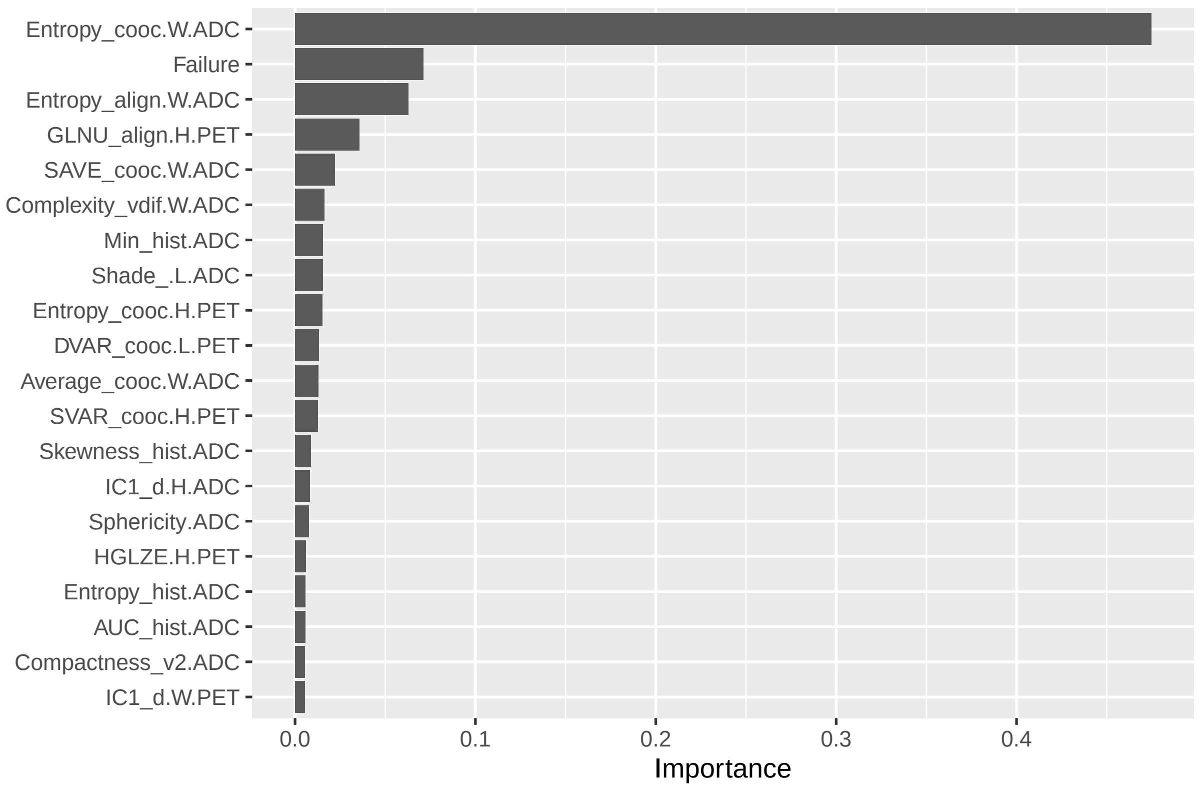
### Modelling the training data using XGBoost

```
xgb.fit.final <- xgboost(
  params = params,
  data = X,
  label = Y,
  nrounds = 394,
  objective = "binary:logistic",
  verbose = 0
)
summary(xgb.fit.final)
```

```
##                Length Class              Mode
## handle              1 xgb.Booster.handle externalptr
## raw            328181 -none-             raw
## niter               1 -none-             numeric
## evaluation_log      2 data.table         list
## call               14 -none-             call
## params              7 -none-             list
## callbacks           1 -none-             list
## feature_names     430 -none-             character
## nfeatures           1 -none-             numeric
```

### Plot the top 20 important features during training

```
vip::vip(xgb.fit.final, num_features = 20)
```



### Prediction performance of the model using training data set

```r
pred_xgboosttrain<- predict(xgb.fit.final, X, type = "prob")
pred_xgboosttrain
```

```
##   [1] 0.28875238 0.30152425 0.25602970 0.92629713 0.19158913 0.89647108
##   [7] 0.15258946 0.11095142 0.76877522 0.40700099 0.83128464 0.87070680
##  [13] 0.05034275 0.71686786 0.73426938 0.14770316 0.86017066 0.92190468
##  [19] 0.79525447 0.91963732 0.91917479 0.03644962 0.04278171 0.91484690
##  [25] 0.18570501 0.90313697 0.06638998 0.06558927 0.35646239 0.04697568
##  [31] 0.04117190 0.92019469 0.91875321 0.05389850 0.12956952 0.13396433
##  [37] 0.91463023 0.04486260 0.17500959 0.06216494 0.08394852 0.05856422
##  [43] 0.91931665 0.04526942 0.03153244 0.33592409 0.15302026 0.05646370
##  [49] 0.04025634 0.06908067 0.06857249 0.07497264 0.89442128 0.03849119
##  [55] 0.04015504 0.05165524 0.86540991 0.04548761 0.04735329 0.09144320
##  [61] 0.85765636 0.07284794 0.09609137 0.05816139 0.04363082 0.04978577
##  [67] 0.06926540 0.93451840 0.06013738 0.09982964 0.05192779 0.90535688
##  [73] 0.04028035 0.05166441 0.09987261 0.08139315 0.05203830 0.06554573
##  [79] 0.17735453 0.94690728 0.75249511 0.91254407 0.08138330 0.04603196
##  [85] 0.05000319 0.81480581 0.13473703 0.08985078 0.84237385 0.05736211
##  [91] 0.14540972 0.07876454 0.12444738 0.70572263 0.21564540 0.10242965
##  [97] 0.75710547 0.10351728 0.12288468 0.08107327 0.06177349 0.09683329
## [103] 0.31482378 0.27047104 0.08786308 0.92919528 0.66459125 0.85611993
## [109] 0.52401751 0.10979064 0.16040021 0.08576418 0.11877970 0.09171715
## [115] 0.11712329 0.05936291 0.86777848 0.80973506 0.58933258 0.68868154
## [121] 0.46976453 0.20701872 0.70986509 0.48012668 0.86052591 0.07634561
## [127] 0.08452407 0.06065508 0.05392676 0.09222873 0.04385296 0.64608473
## [133] 0.76712281 0.82446569 0.08518016 0.92197174 0.89314121 0.13330054
## [139] 0.88874727 0.14768700 0.78173810 0.13287185 0.30474433 0.11704161
## [145] 0.52451754 0.80776584 0.30594414 0.21836394 0.08907196 0.85396904
## [151] 0.09733468 0.77617955 0.06194000 0.06227546 0.28544852 0.11334136
## [157] 0.85894907
```

### Compute AUC metrics

```r
perf1 <- prediction(pred_xgboosttrain, radiomics_train$Failure.binary) %>%
  performance(measure = "tpr", x.measure = "fpr")
```

### Prediction performance of the model in testing dataset

```r
xgb_preptest <- recipe(Failure.binary~ ., data = radiomics_test) %>%
  step_integer(all_nominal()) %>%
  prep(training = radiomics_test, retain = TRUE) %>%
  juice()

X1 <- as.matrix(xgb_preptest[setdiff(names(xgb_preptest), "Failure.binary")])
```

### Prediction performance of the model using testing data set

```
pred_xgboosttest<- predict(xgb.fit.final, X1, type = "prob")
pred_xgboosttest
```

```
##  [1] 0.06346330 0.87782639 0.90982735 0.12173807 0.53001356 0.65260023
##  [7] 0.06886759 0.16263841 0.13524383 0.88837379 0.08139928 0.10966636
## [13] 0.07884696 0.85393512 0.10967711 0.84386265 0.83606106 0.62201697
## [19] 0.72121668 0.09942771 0.65444428 0.19247118 0.66308051 0.75610620
## [25] 0.79082209 0.13557769 0.11162146 0.79597753 0.14485182 0.78792340
## [31] 0.18671314 0.16007788 0.14810665 0.13562724 0.13927166 0.17027317
## [37] 0.21817867 0.67547232 0.24571458 0.22485881
```

### Compute AUC metrics

```
perf2 <- prediction(pred_xgboosttest, radiomics_test$Failure.binary) %>%
  performance(measure = "tpr", x.measure = "fpr")
```

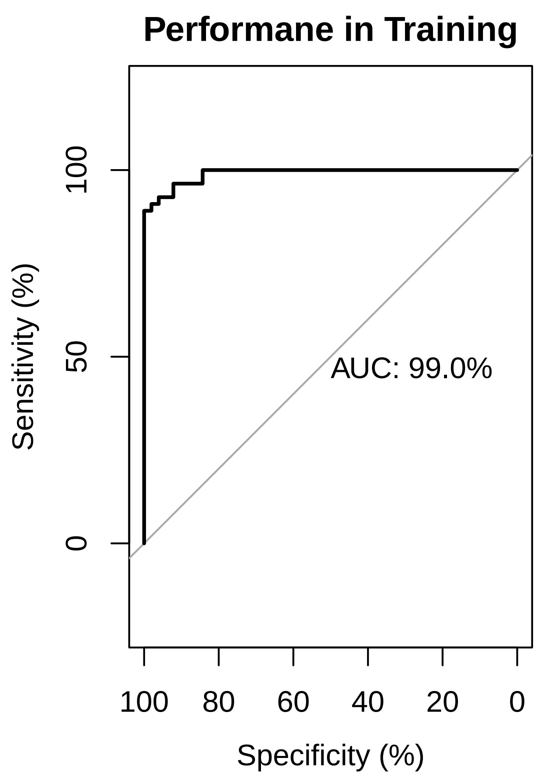
### Training and Testing data performance plot

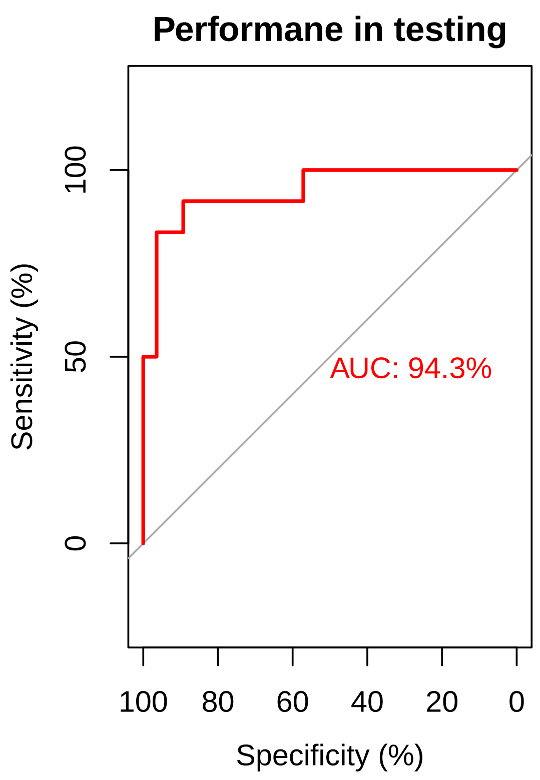
```
par(mfrow = c(1,2))

# Training prediction performance
roc(radiomics_train$Failure.binary ~ pred_xgboosttrain, plot=TRUE, legacy.axes=FALSE,
    percent=TRUE, col="black", lwd=2, print.auc=TRUE, main = "Performane in Training")
```

```
## Setting levels: control = 0, case = 1

## Setting direction: controls < cases

##
## Call:
## roc.formula(formula = radiomics_train$Failure.binary ~ pred_xgboosttrain,     plot = TRUE, legacy.axe
##
## Data: pred_xgboosttrain in 102 controls (radiomics_train$Failure.binary 0) < 55 cases (radiomics_trai
## Area under the curve: 99.04%
```

```
# Testing set prediction performance
roc(radiomics_test$Failure.binary ~ pred_xgboosttest, plot=TRUE, legacy.axes=FALSE,
    percent=TRUE, col="red", lwd=2, print.auc=TRUE, main = "Performane in testing")
```

```
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
```





```
##
## Call:
## roc.formula(formula = radiomics_test$Failure.binary ~ pred_xgboosttest,     plot = TRUE, legacy.axes
##
## Data: pred_xgboosttest in 28 controls (radiomics_test$Failure.binary 0) < 12 cases (radiomics_test$Fa
## Area under the curve: 94.35%
```

The performance during training has the highest AUC of 0.99 which indicates that it has the highest area under the curve compared to the performance during testing which has 0.943 AUC.