MODEL 1 GRADIENT BOOSTING MODEL

REY P. PENDANG

2022-12-16

```r
# Load Packages
library(dplyr)    # for general data wrangling needs
```

```
##
## 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
```

```r
library(tidyverse)# for filtering
```

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

## v ggplot2 3.4.0     v purrr   0.3.5
## v tibble  3.1.8     v stringr 1.4.1
## v tidyr   1.2.1     v forcats 0.5.2
## v readr   2.1.3
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```r
library(rsample)   # for creating validation splits
library(h2o)      # for a java-based implementation of GBM variants
```

```
##
## ----------------------------------------------------------------------
##
## 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)  # for fitting extreme gradient boosting
```

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

```
library(gbm)      # for original implementation of regular and stochastic GBMs
```

```
## Loaded gbm 2.1.8.1
```

```
library(ROCR)
library(pROC)
```

```
## 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
```

```
library(h2o)        # a java-based implementation of random forest
h2o.init()
```

```
##  Connection successful!
##
## R is connected to the H2O cluster:
##     H2O cluster uptime:         7 hours 44 minutes
##     H2O cluster timezone:       Asia/Taipei
##     H2O data parsing timezone:  UTC
##     H2O cluster version:        3.38.0.1
##     H2O cluster version age:    2 months and 27 days
##     H2O cluster name:           H2O_started_from_R_REY_hvw787
##     H2O cluster total nodes:    1
##     H2O cluster total memory:   3.76 GB
##     H2O cluster total cores:    16
##     H2O cluster allowed cores:  16
##     H2O cluster healthy:        TRUE
##     H2O Connection ip:          localhost
##     H2O Connection port:        54321
##     H2O Connection proxy:       NA
##     H2O Internal Security:      FALSE
##     R Version:                  R version 4.2.2 (2022-10-31 ucrt)
```

RUNNING BASIC GBM MODEL

```r
set.seed(123)  # for reproducibility
dt<- read_csv("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.
```

```r
dt$Institution=as.factor(dt$Institution)
split <- initial_split(dt, strata = "Failure.binary")
traindt <- training(split)
testdt <- testing(split)
```

```r
gradientBoostingModel_1 <- gbm(
  formula = Failure.binary ~ .,
  data = traindt,
  distribution = "bernoulli",  # SSE loss function
  n.trees = 5000,
  shrinkage = 0.1,
  n.minobsinnode = 10,
  cv.folds = 10

)
```

FIND INDEX FOR NUMBER TREES WITH MINIMUM CV ERROR

```r
best <- which.min(gradientBoostingModel_1$cv.error)
```

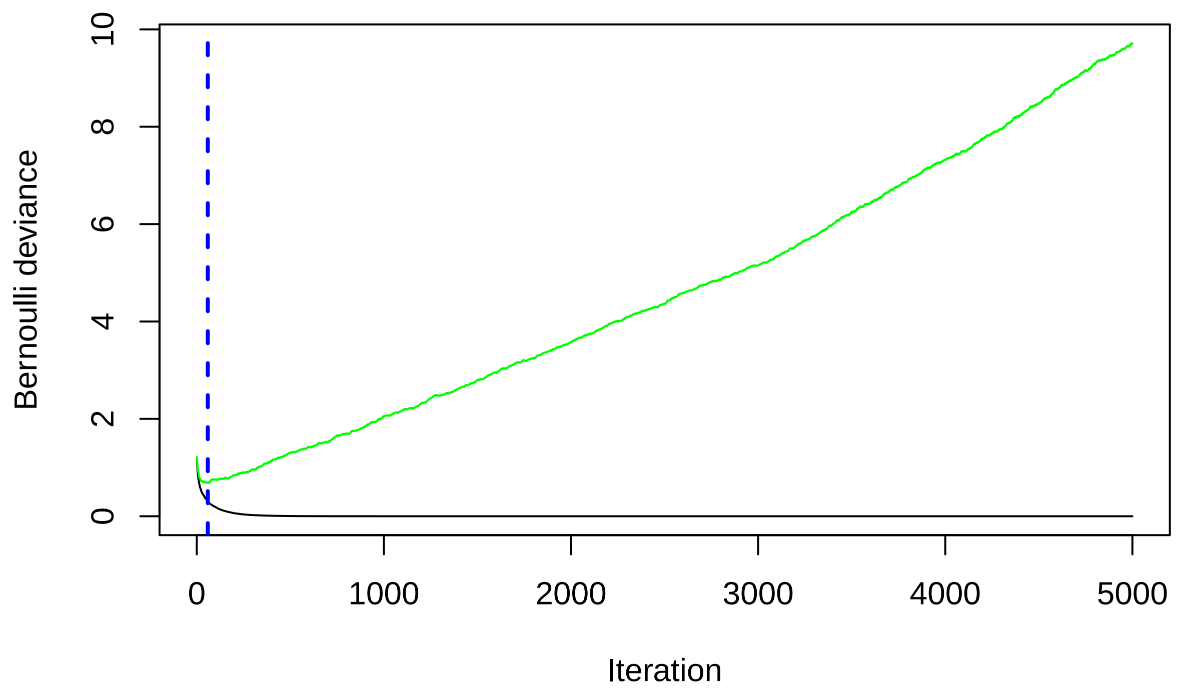
GET MSE AND COMPUTE RMSE

```r
sqrt(gradientBoostingModel_1$cv.error[best])
```

```
## [1] 0.8234636
```

PLOTTING THE ERROR CURVE

```r
gbm.perf(gradientBoostingModel_1, method = "cv")
```



```
## [1] 59
```

CREATE GRID SEARCH

```
hyper_grid <- expand.grid(
  learning_rate = c(0.3, 0.1, 0.05, 0.01, 0.005),
  logloss = NA,
  trees = NA,
  time = NA
)

# EXECUTE GRID SEARCH
for(i in seq_len(nrow(hyper_grid))) {

  # fit gbm
  set.seed(123)  # for reproducibility
  train_time <- system.time({
    m <- gbm(
      formula = Failure.binary ~ .,
      data = traindt,
      distribution = "bernoulli",
      n.trees = 5000,
      shrinkage = hyper_grid$learning_rate[i],
      interaction.depth = 3,
      n.minobsinnode = 10,
      cv.folds = 10
    )
  })

  # adding SSE, trees, and training time to results
  hyper_grid$logloss[i]  <- sqrt(min(m$cv.error))
  hyper_grid$trees[i] <- which.min(m$cv.error)
  hyper_grid$Time[i]  <- train_time[["elapsed"]]

}
```

RESULTS

```
arrange(hyper_grid, logloss)
```

```
##   learning_rate   logloss trees time  Time
## 1         0.100 0.7656695    32   NA 27.83
## 2         0.005 0.7782019   905   NA 28.64
## 3         0.050 0.7850738    82   NA 29.51
## 4         0.010 0.7916756   400   NA 28.71
## 5         0.300 0.8043208    17   NA 27.39
```

SEARCH GRID

```
hyper_grid <- expand.grid(
  n.trees = 6000,
  shrinkage = 0.01,
  interaction.depth = c(3, 5, 7),
  n.minobsinnode = c(5, 10, 15)

)
```

CREATING THE MODEL FIT FUNCTION

```
model_fit <- function(n.trees, shrinkage, interaction.depth, n.minobsinnode) {
  set.seed(123)
  m <- gbm(
    formula = Failure.binary ~ .,
    data = traindt,
    distribution = "bernoulli",
    n.trees = n.trees,
    shrinkage = shrinkage,
    interaction.depth = interaction.depth,
    n.minobsinnode = n.minobsinnode,
    cv.folds = 10
  )
  # compute RMSE
  sqrt(min(m$cv.error))

}
```

PERFORMING SEARCH GRID WITH FUNCTIONAL PROGRAMMING

```
hyper_grid$logloss <- purrr::pmap_dbl(
  hyper_grid,
  ~ model_fit(
    n.trees = ..1,
    shrinkage = ..2,
    interaction.depth = ..3,
    n.minobsinnode = ..4
  )
)

# RESULTS
arrange(hyper_grid, logloss)
```

```
##   n.trees shrinkage interaction.depth n.minobsinnode   logloss
## 1    6000      0.01                 3             15 0.7628452
## 2    6000      0.01                 5             15 0.7628452
## 3    6000      0.01                 7             15 0.7628452
## 4    6000      0.01                 3             10 0.7916756
## 5    6000      0.01                 5             10 0.7917035
## 6    6000      0.01                 7             10 0.7917035
## 7    6000      0.01                 3              5 0.7958123
## 8    6000      0.01                 5              5 0.7971165
## 9    6000      0.01                 7              5 0.7971459
```

REFINED HYPERPARAMETER GRID

```
hyper_grid <- list(
  sample_rate = c(0.5, 0.75, 1),              # row subsampling
  col_sample_rate = c(0.5, 0.75, 1),          # col subsampling for each split
  col_sample_rate_per_tree = c(0.5, 0.75, 1)  # col subsampling for each tree
)

# random grid search strategy
search_criteria <- list(
  strategy = "RandomDiscrete",
  stopping_metric = "logloss",
  stopping_tolerance = 0.001,
  stopping_rounds = 10,
  max_runtime_secs = 60*60
)
```

PERFORMING GRID SEARCH

```
traindt$Failure.binary=as.factor(traindt$Failure.binary)

h2o.init()
```

```
##  Connection successful!
```

```
##
## R is connected to the H2O cluster:
##     H2O cluster uptime:         7 hours 53 minutes
##     H2O cluster timezone:       Asia/Taipei
##     H2O data parsing timezone:  UTC
##     H2O cluster version:        3.38.0.1
##     H2O cluster version age:    2 months and 27 days
##     H2O cluster name:           H2O_started_from_R_REY_hvw787
##     H2O cluster total nodes:    1
##     H2O cluster total memory:   3.76 GB
##     H2O cluster total cores:    16
##     H2O cluster allowed cores:  16
##     H2O cluster healthy:        TRUE
##     H2O Connection ip:          localhost
##     H2O Connection port:        54321
##     H2O Connection proxy:       NA
##     H2O Internal Security:      FALSE
##     R Version:                  R version 4.2.2 (2022-10-31 ucrt)
```

```
grid <- h2o.grid(
  algorithm = "gbm",
  grid_id = "gbm_grid",
  y = "Failure.binary",
  training_frame = as.h2o(traindt),
  hyper_params = hyper_grid,
  ntrees = 10,#supposedly 6000
  learn_rate = 0.01,
  max_depth = 7,
  min_rows = 5,
  nfolds = 10,
  stopping_rounds = 10,
  stopping_tolerance = 0,
  stopping_metric="logloss",
  search_criteria = search_criteria,
  seed = 123

)
```

```
##   |                                                                              |
##   |                                                                              |
```

COLLECT THE RESULTS AND SORT BY OUR MODEL PERFORMANCE METRIC OF CHOICE

```
grid_perf <- h2o.getGrid(
  grid_id = "gbm_grid",
  sort_by = "logloss",
  decreasing = FALSE
)
```

```
grid_perf
```

```
## H2O Grid Details
## ================
##
## Grid ID: gbm_grid
## Used hyper parameters:
##   -  col_sample_rate
##   -  col_sample_rate_per_tree
##   -  sample_rate
## Number of models: 27
## Number of failed models: 0
##
## Hyper-Parameter Search Summary: ordered by increasing logloss
##   col_sample_rate col_sample_rate_per_tree sample_rate         model_ids
## 1         1.00000                  1.00000     1.00000 gbm_grid_model_11
## 2         0.75000                  1.00000     1.00000 gbm_grid_model_18
## 3         1.00000                  1.00000     0.75000 gbm_grid_model_22
## 4         0.75000                  1.00000     0.75000  gbm_grid_model_5
## 5         1.00000                  1.00000     0.50000  gbm_grid_model_4
##   logloss
## 1 0.59521
## 2 0.59742
## 3 0.59989
## 4 0.60071
## 5 0.60091
##
## ---
##    col_sample_rate col_sample_rate_per_tree sample_rate         model_ids
## 22         0.50000                  0.50000     1.00000 gbm_grid_model_12
## 23         1.00000                  0.50000     0.50000 gbm_grid_model_15
## 24         0.50000                  1.00000     0.50000  gbm_grid_model_7
## 25         0.75000                  0.50000     0.50000 gbm_grid_model_19
## 26         0.50000                  0.50000     0.75000 gbm_grid_model_17
## 27         0.50000                  0.50000     0.50000  gbm_grid_model_9
##    logloss
## 22 0.60916
## 23 0.60926
## 24 0.60976
## 25 0.61094
## 26 0.61317
## 27 0.61769
```

GRAB THE MODEL_ID FOR THE TOP MODEL, CHOSEN BY CROSS VALIDATION ERROR

```
best_model_id <- grid_perf@model_ids[[1]]
best_model <- h2o.getModel(best_model_id)

# GETTING THE PERFORMANCE METRICS ON THE BEST MODEL

h2o.performance(model = best_model, xval = TRUE)
```

```
## H2OBinomialMetrics: gbm
## ** Reported on cross-validation data. **
## ** 10-fold cross-validation on training data (Metrics computed for combined holdout predictions) **
##
```

```
## MSE:  0.2035836
## RMSE:  0.4512024
## LogLoss:  0.595207
## Mean Per-Class Error:  0.156701
## AUC:  0.8316495
## AUCPR:  0.6940474
## Gini:  0.663299
## R^2:  0.09294073
##
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
##         0  1    Error     Rate
## 0      86 11 0.113402   =11/97
## 1      10 40 0.200000   =10/50
## Totals 96 51 0.142857  =21/147
##
## Maximum Metrics: Maximum metrics at their respective thresholds
##                         metric threshold     value idx
## 1                       max f1  0.358378  0.792079  42
## 2                       max f2  0.324552  0.820896  56
## 3                 max f0point5  0.358378  0.787402  42
## 4                 max accuracy  0.358378  0.857143  42
## 5                max precision  0.417564  1.000000   0
## 6                   max recall  0.287644  1.000000  98
## 7              max specificity  0.417564  1.000000   0
## 8             max absolute_mcc  0.358378  0.683365  42
## 9   max min_per_class_accuracy  0.342764  0.820000  47
## 10 max mean_per_class_accuracy  0.358378  0.843299  42
## 11                     max tns  0.417564 97.000000   0
## 12                     max fns  0.417564 49.000000   0
## 13                     max fps  0.281445 97.000000 100
## 14                     max tps  0.287644 50.000000  98
## 15                     max tnr  0.417564  1.000000   0
## 16                     max fnr  0.417564  0.980000   0
## 17                     max fpr  0.281445  1.000000 100
## 18                     max tpr  0.287644  1.000000  98
##
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/
```

```r
library(recipes)
```

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

## The following object is masked from 'package:stringr':
##
##     fixed

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

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

```r
X <- as.matrix(xgb_prep[setdiff(names(xgb_prep), "Failure.binary")])
Y <- xgb_prep$Failure.binary
Y=as.numeric(Y)-1
```

```r
set.seed(123)
ames_xgb <- xgb.cv(
  data = X,
  label = Y,
  nrounds = 6000,
  objective = "binary:logistic",
  early_stopping_rounds = 50,
  nfold = 10,
  params = list(
    eta = 0.1,
    max_depth = 3,
    min_child_weight = 3,
    subsample = 0.8,
    colsample_bytree = 1.0),
  verbose = 0
)
```

MINIMUM TEST CV RMSE

```r
min(ames_xgb$evaluation_log$test_logloss_mean)
```

```
## [1] 0.3090401
```

```r
# hyperparameter grid
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),
  logloss = 0,          # a place to dump RMSE results
  trees = 0          # a place to dump required number of trees
)

# grid search
for(i in seq_len(nrow(hyper_grid))) {
  set.seed(123)
  m <- xgb.cv(
    data = X,
    label = Y,
    nrounds = 100,#supposedly 4000
    objective = "binary:logistic",
    early_stopping_rounds = 50,
    nfold = 10,
    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$logloss[i] <- min(m$evaluation_log$test_logloss_mean)
  hyper_grid$trees[i] <- m$best_iteration
}
```

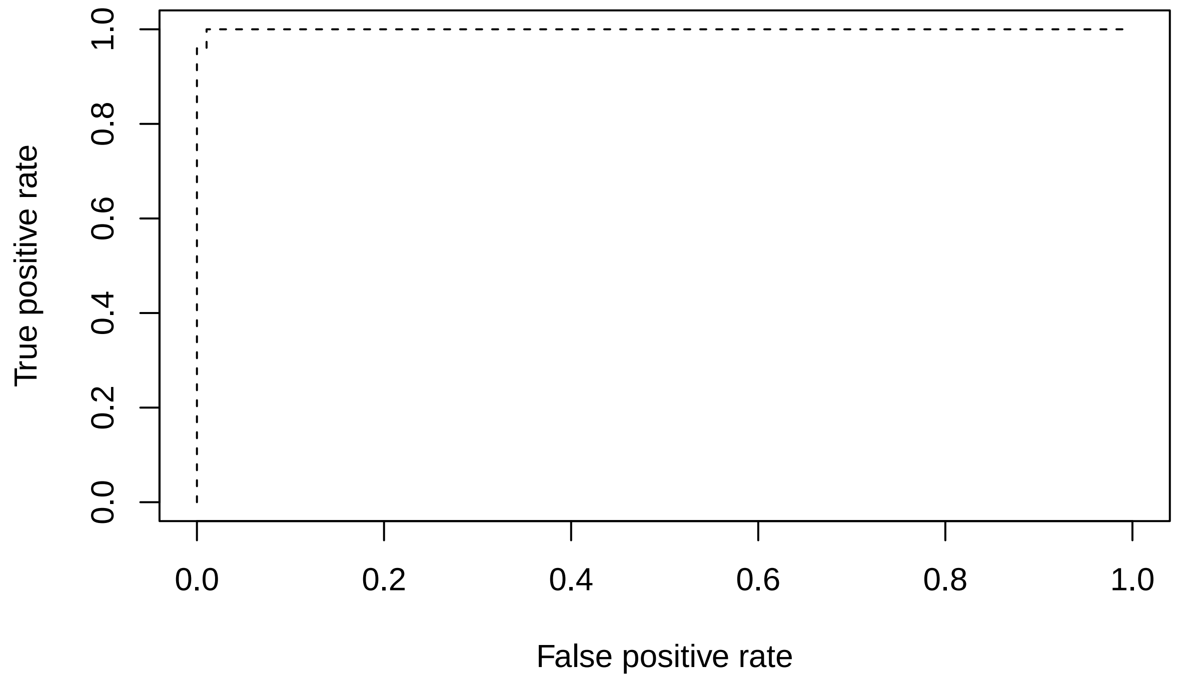
```
# results
hyper_grid %>%
  filter(logloss > 0) %>%
  arrange(logloss) %>%
  glimpse()
```

```
## Rows: 245
## Columns: 10
## $ eta              <dbl> 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01,~
## $ max_depth        <dbl> 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,~
## $ min_child_weight <dbl> 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,~
## $ subsample        <dbl> 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5~
## $ colsample_bytree <dbl> 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5~
## $ gamma            <dbl> 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0,~
## $ lambda           <dbl> 0.00, 0.00, 0.00, 0.01, 0.00, 0.01, 0.01, 0.01, 0.00,~
## $ alpha            <dbl> 0.00, 0.00, 0.01, 0.00, 0.01, 0.00, 0.01, 0.01, 0.10,~
## $ logloss          <dbl> 0.4500955, 0.4503004, 0.4505360, 0.4505707, 0.4506906~
## $ trees            <dbl> 100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100~
```

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

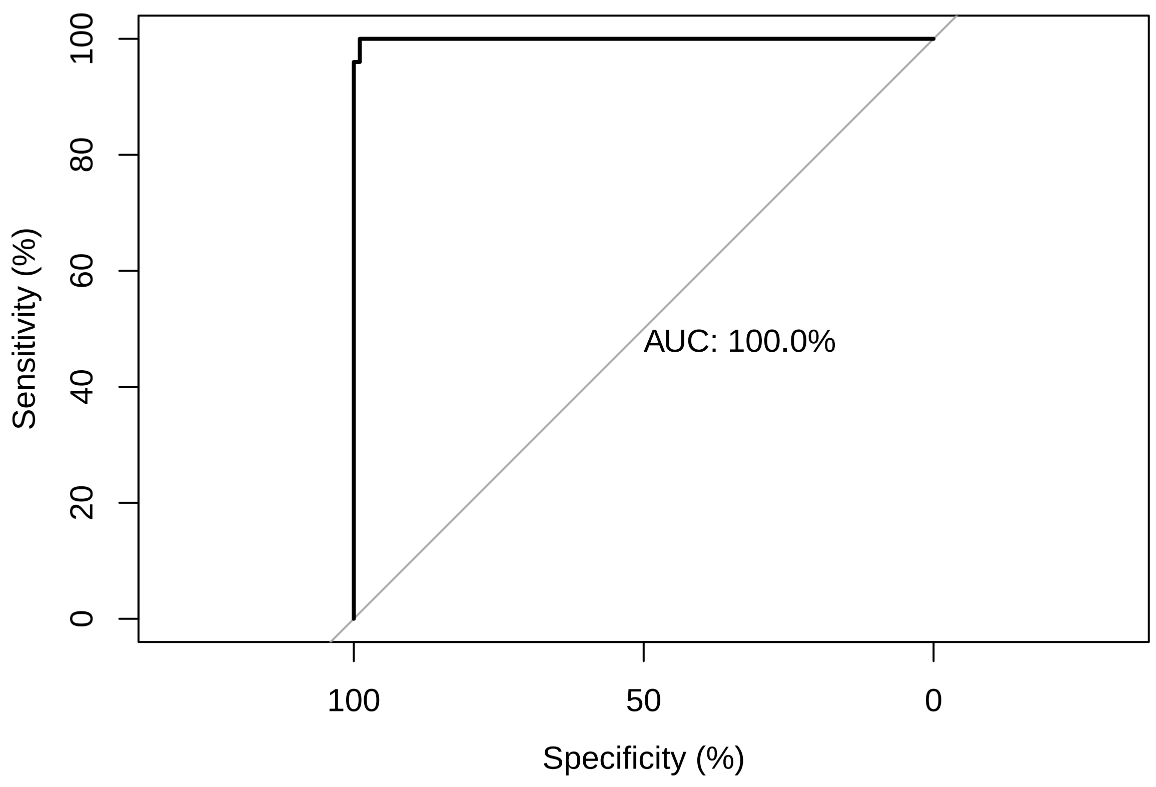
```
#
# # train final model
# traindt$Institution=fct_recode(traindt$Institution, "1" = "A", "2" ="B","3"="C","4"="D")
# traindt$Institution=as.numeric(traindt$Institution)
# traindt=as.matrix(traindt)

xgb.fit.final <- xgboost(
  params = params,
  data = X,
  label = Y,
  nrounds = 3944,
  objective = "binary:logistic",
  verbose = 0
)
```

```
# Compute predicted probabilities on training data
m1_prob <- predict(xgb.fit.final, X, type = "prob")

# Compute AUC metrics for cv_model1,2 and 3
perf1 <- prediction(m1_prob,traindt$Failure.binary) %>%
  performance(measure = "tpr", x.measure = "fpr")


# Plot ROC curves for cv_model1,2 and 3
plot(perf1, col = "black", lty = 2)
```



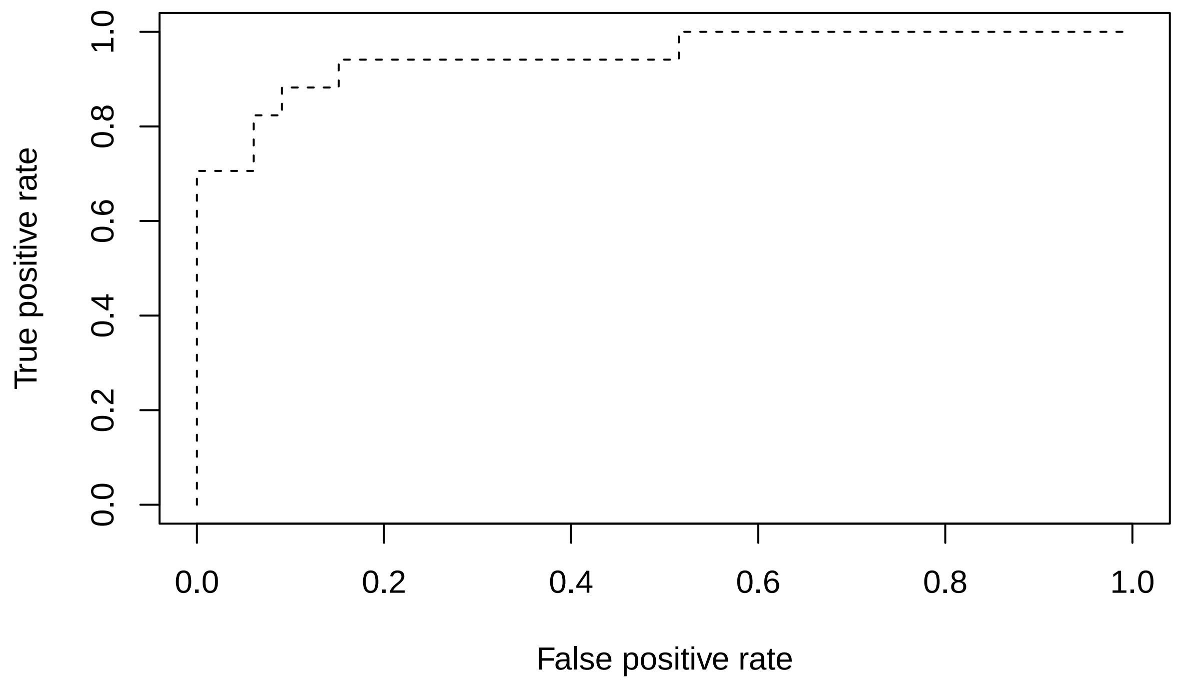
```
# ROC plot for training data
roc( traindt$Failure.binary ~ m1_prob, plot=TRUE, legacy.axes=FALSE,
    percent=TRUE, col="black", lwd=2, print.auc=TRUE)
```

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

## Setting direction: controls < cases
```



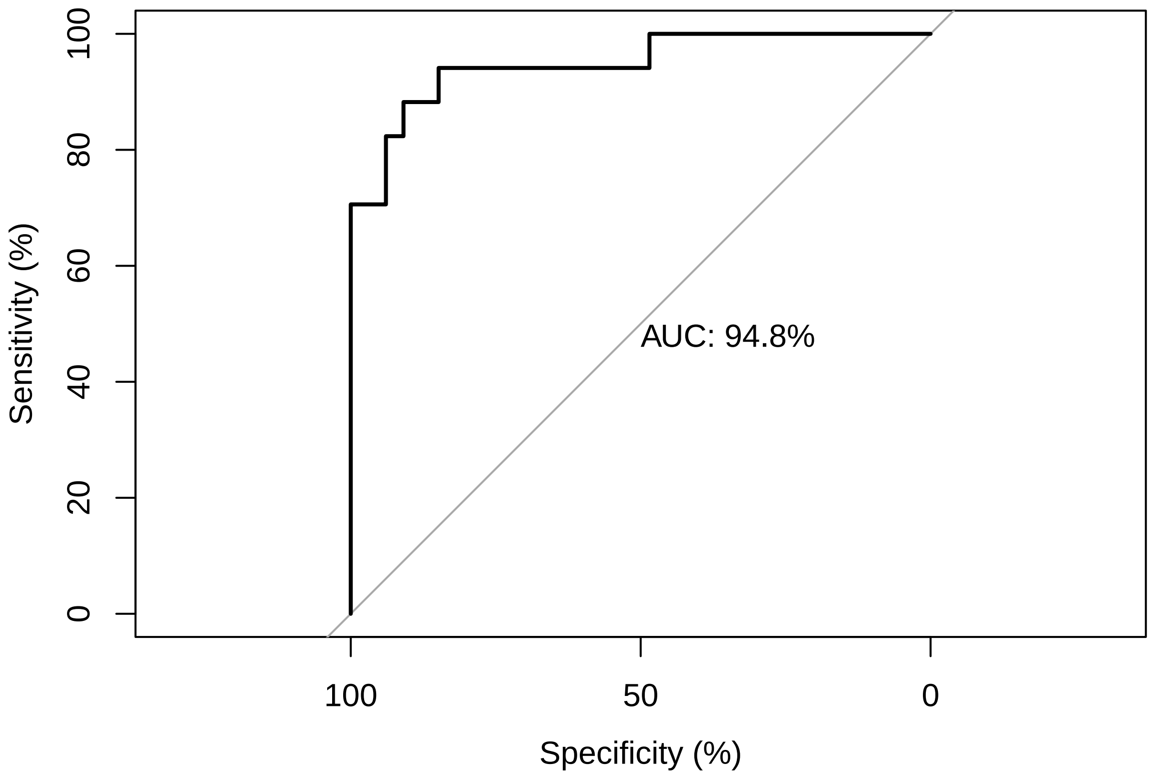
```
##
## Call:
## roc.formula(formula = traindt$Failure.binary ~ m1_prob, plot = TRUE,     legacy.axes = FALSE, percen
##
## Data: m1_prob in 97 controls (traindt$Failure.binary 0) < 50 cases (traindt$Failure.binary 1).
## Area under the curve: 99.96%
```

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

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

# Compute predicted probabilities on training data
m2_prob <- predict(xgb.fit.final, X, type = "prob")

# Compute AUC metrics for cv_model1,2 and 3
perf2 <- prediction(m2_prob,testdt$Failure.binary) %>%
  performance(measure = "tpr", x.measure = "fpr")


# Plot ROC curves for cv_model1,2 and 3
plot(perf2, col = "black", lty = 2)
```



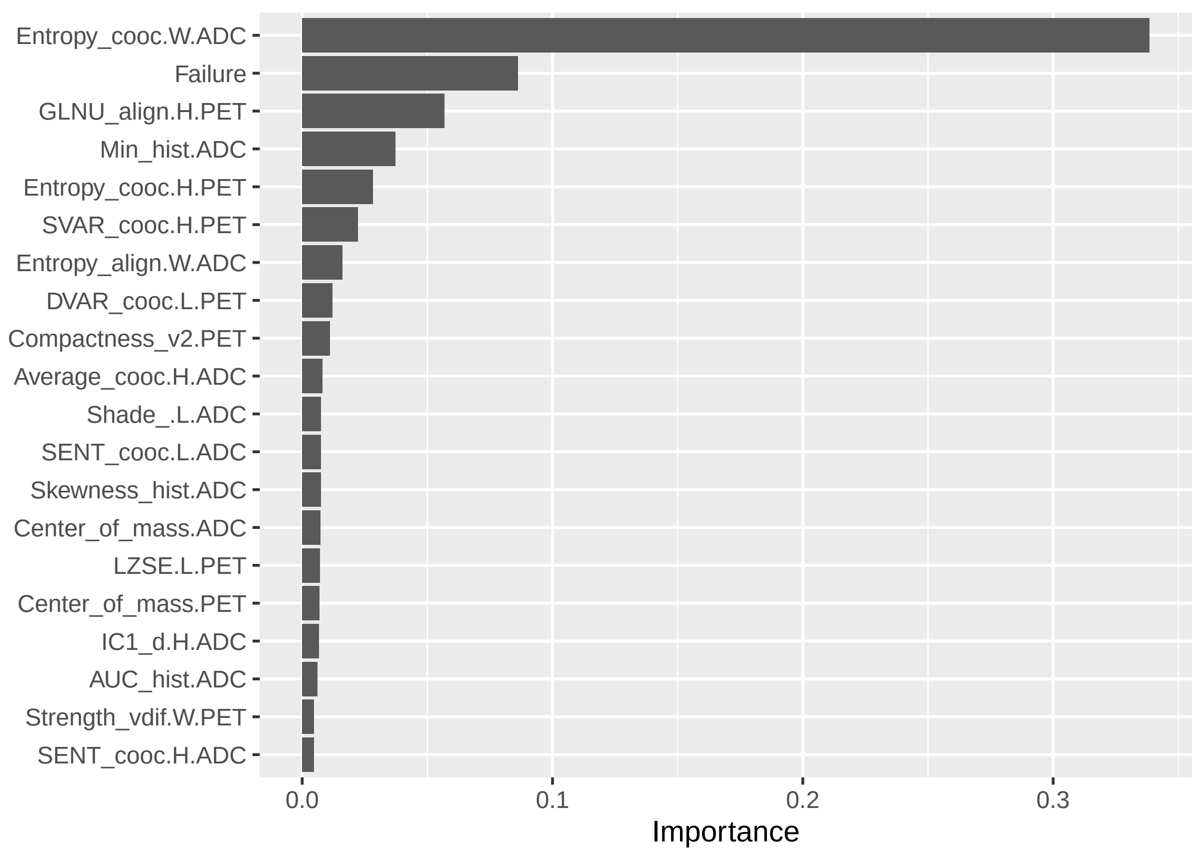
```
# ROC plot for training data
roc( testdt$Failure.binary ~ m2_prob, plot=TRUE, legacy.axes=FALSE,
    percent=TRUE, col="black", lwd=2, print.auc=TRUE)
```

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



```
##
## Call:
## roc.formula(formula = testdt$Failure.binary ~ m2_prob, plot = TRUE,     legacy.axes = FALSE, percent
##
## Data: m2_prob in 33 controls (testdt$Failure.binary 0) < 17 cases (testdt$Failure.binary 1).
## Area under the curve: 94.83%
```

```
# variable importance plot
vip::vip(xgb.fit.final,num_features=20)
```



Entropy_cooc.W.ADC
Failure
GLNU_align.H.PET
Min_hist.ADC
Entropy_cooc.H.PET
SVAR_cooc.H.PET
Entropy_align.W.ADC
DVAR_cooc.L.PET
Compactness_v2.PET
Average_cooc.H.ADC
Shade_.L.ADC
SENT_cooc.L.ADC
Skewness_hist.ADC
Center_of_mass.ADC
LZSE.L.PET
Center_of_mass.PET
IC1_d.H.ADC
AUC_hist.ADC
Strength_vdif.W.PET
SENT_cooc.H.ADC
0.0
0.1
0.2
0.3
Importance