MODEL 1 - Stacked Model

FENDAWN F. RECENTES

12/16/2022

Helper and Modeling Packages

```
library(rsample)
```

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

```
library(recipes)
```

```
## Loading required package: 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
```

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

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

```
library(tidyverse)
```

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

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

```
## v ggplot2 3.3.6     v purrr   0.3.4
## v tibble  3.1.8     v stringr 1.4.1
## v tidyr   1.2.0     v forcats 0.5.2
## v readr   2.1.2

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

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

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

## Warning: package 'readr' 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()
```

```
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
```

```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
h2o.init()
```

```
##  Connection successful!
##
## R is connected to the H2O cluster:
##     H2O cluster uptime:         2 hours 51 minutes
##     H2O cluster timezone:       Asia/Manila
##     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_MSU-TCTO_OVCAA_mvc880
##     H2O cluster total nodes:    1
##     H2O cluster total memory:   3.70 GB
##     H2O cluster total cores:    8
##     H2O cluster allowed cores:  8
##     H2O cluster healthy:        TRUE
##     H2O Connection ip:          localhost
##     H2O Connection port:        54321
##     H2O Connection proxy:       NA
##     H2O Internal Security:      FALSE
##     R Version:                  R version 4.1.2 (2021-11-01)
```

Load and view radiomics data set

```r
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.
```

```
view(radiomics)
```

Convert target variable to a factor form

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

DATA PREPARATION AND SPLITTING

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

```
set.seed(123)  # for reproducibility
split <- initial_split(radiomics, strata = "Failure.binary")
radiomics_train <- training(split)
radiomics_test <- testing(split)
```

Make sure we have consistent categorical levels

```
blueprint <- recipe(Failure.binary ~ ., data = radiomics_train) %>%
  step_other(all_nominal(), threshold = 0.005)
```

Create training & test sets for h2o

```
h2o.init()
```

```
##  Connection successful!
##
## R is connected to the H2O cluster:
##     H2O cluster uptime:         2 hours 51 minutes
##     H2O cluster timezone:       Asia/Manila
##     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_MSU-TCTO_OVCAA_mvc880
##     H2O cluster total nodes:    1
##     H2O cluster total memory:   3.70 GB
```

```
##     H2O cluster total cores:    8
##     H2O cluster allowed cores:  8
##     H2O cluster healthy:        TRUE
##     H2O Connection ip:          localhost
##     H2O Connection port:        54321
##     H2O Connection proxy:       NA
##     H2O Internal Security:      FALSE
##     R Version:                  R version 4.1.2 (2021-11-01)
```

```
train_h2o <- prep(blueprint, training = radiomics_train, retain = TRUE) %>%
  juice() %>%
  as.h2o()
```

```
##   |                                                                              |
```

```
test_h2o <- prep(blueprint, training = radiomics_train) %>%
  bake(new_data = radiomics_test) %>%
  as.h2o()
```

```
##   |                                                                              |
```

Get response and feature names

```
Y <- "Failure.binary"
X <- setdiff(names(radiomics_train), Y)
```

Train & cross-validate a GLM model

```
best_glm <- h2o.glm(
  x = X, y = Y, training_frame = train_h2o, alpha = 0.1,
  remove_collinear_columns = TRUE, nfolds = 10, fold_assignment = "Modulo",
  keep_cross_validation_predictions = TRUE, seed = 123
)
```

```
##   |                                                                              |
```

Train & cross-validate a RF model

```
best_rf <- h2o.randomForest(
  x = X, y = Y, training_frame = train_h2o, ntrees = 100, mtries = 20,
  max_depth = 30, min_rows = 1, sample_rate = 0.8, nfolds = 10,
  fold_assignment = "Modulo", keep_cross_validation_predictions = TRUE,
  seed = 123, stopping_rounds = 50, stopping_metric = "logloss",
  stopping_tolerance = 0
)
```

```
## Warning in .h2o.processResponseWarnings(res): early stopping is enabled but neither score_tree_inter
```

```
##   |                                                                              |
```

Train & cross-validate a GBM model

```
best_gbm <- h2o.gbm(
  x = X, y = Y, training_frame = train_h2o, ntrees = 100, learn_rate = 0.01,
  max_depth = 7, min_rows = 5, sample_rate = 0.8, nfolds = 10,
  fold_assignment = "Modulo", keep_cross_validation_predictions = TRUE,
  seed = 123, stopping_rounds = 50, stopping_metric = "logloss",
  stopping_tolerance = 0
)
```

```
## Warning in .h2o.processResponseWarnings(res): early stopping is enabled but neither score_tree_inter

##   |                                                                              |
```

Get results from base learners

```
get_rmse <- function(model) {
  results <- h2o.performance(model, newdata = test_h2o)
  results@metrics$RMSE
}
list(best_glm, best_rf, best_gbm) %>%
  purrr::map_dbl(get_rmse)
```

```
## [1] 0.4737088 0.3992117 0.3338713
```

Define GBM hyperparameter grid

```
hyper_grid <- list(
  max_depth = c(1, 3, 5),
  min_rows = c(1, 5, 10),
  learn_rate = c(0.01, 0.05, 0.1),
  learn_rate_annealing = c(0.99, 1),
  sample_rate = c(0.5, 0.75, 1),
  col_sample_rate = c(0.8, 0.9, 1)
)

# Define random grid search criteria
search_criteria <- list(
  strategy = "RandomDiscrete",
  max_models = 25
)

# Build random grid search
random_grid <- h2o.grid(
  algorithm = "gbm", grid_id = "gbm_grid", x = X, y = Y,
  training_frame = train_h2o, hyper_params = hyper_grid,
  search_criteria = search_criteria, ntrees = 20, stopping_metric = "logloss",
  stopping_rounds = 10, stopping_tolerance = 0, nfolds = 10,
```

```
  fold_assignment = "Modulo", keep_cross_validation_predictions = TRUE,
  seed = 123
)
```

```
##   |                                                                              |
```

```
ensemble_tree <- h2o.stackedEnsemble(
  x = X, y = Y, training_frame = train_h2o, model_id = "ensemble_gbm_grid",
  base_models = random_grid@model_ids, metalearner_algorithm = "gbm",
)
```

```
##   |                                                                              |
```

Stacked results

```
h2o.performance(ensemble_tree, newdata = test_h2o)@metrics$RMSE
```

```
## [1] 0.3668616
```

```
data.frame(
  GLM_pred = as.vector(h2o.getFrame(best_glm@model$cross_validation_holdout_predictions_frame_id$name))%
  RF_pred = as.vector(h2o.getFrame(best_rf@model$cross_validation_holdout_predictions_frame_id$name))%>%
  GBM_pred = as.vector(h2o.getFrame(best_gbm@model$cross_validation_holdout_predictions_frame_id$name))%
) %>% cor()
```

```
##            GLM_pred    RF_pred  GBM_pred
## GLM_pred 1.00000000 0.08062095 0.0323378
## RF_pred  0.08062095 1.00000000 0.7063735
## GBM_pred 0.03233780 0.70637346 1.0000000
```

Sort results by RMSE

```
h2o.getGrid(
  grid_id = "gbm_grid",
  sort_by = "logloss"
)
```

```
## H2O Grid Details
## ================
##
## Grid ID: gbm_grid
## Used hyper parameters:
##   -  col_sample_rate
##   -  learn_rate
##   -  learn_rate_annealing
##   -  max_depth
##   -  min_rows
```

```
##   -  sample_rate
## Number of models: 25
## Number of failed models: 0
##
## Hyper-Parameter Search Summary: ordered by increasing logloss
##   col_sample_rate learn_rate learn_rate_annealing max_depth min_rows
## 1         1.00000    0.10000              0.99000   3.00000  1.00000
## 2         0.90000    0.10000              1.00000   3.00000 10.00000
## 3         0.80000    0.10000              1.00000   3.00000 10.00000
## 4         0.90000    0.10000              1.00000   5.00000  5.00000
## 5         0.80000    0.10000              1.00000   3.00000 10.00000
##   sample_rate         model_ids logloss
## 1     1.00000 gbm_grid_model_22 0.28791
## 2     1.00000  gbm_grid_model_6 0.32343
## 3     1.00000 gbm_grid_model_21 0.32762
## 4     0.50000 gbm_grid_model_18 0.33001
## 5     0.75000  gbm_grid_model_7 0.33064
##
## ---
##    col_sample_rate learn_rate learn_rate_annealing max_depth min_rows
## 20         1.00000    0.01000              1.00000   5.00000 10.00000
## 21         0.90000    0.01000              1.00000   3.00000 10.00000
## 22         0.90000    0.01000              1.00000   5.00000 10.00000
## 23         0.80000    0.01000              0.99000   5.00000 10.00000
## 24         0.80000    0.01000              0.99000   5.00000 10.00000
## 25         1.00000    0.01000              0.99000   1.00000  1.00000
##    sample_rate         model_ids logloss
## 20     0.75000  gbm_grid_model_9 0.54848
## 21     0.75000 gbm_grid_model_20 0.54942
## 22     1.00000 gbm_grid_model_11 0.55124
## 23     1.00000 gbm_grid_model_10 0.55677
## 24     0.50000 gbm_grid_model_24 0.55821
## 25     1.00000  gbm_grid_model_8 0.56397
```

```r
random_grid_perf <- h2o.getGrid(
  grid_id = "gbm_grid",
  sort_by = "logloss"
)
```

Grab the model_id for the top model, chosen by validation error

```r
best_model_id <- random_grid_perf@model_ids[[1]]
best_model <- h2o.getModel(best_model_id)
h2o.performance(best_model, newdata = test_h2o)
```

```
## H2OBinomialMetrics: gbm
##
## MSE:  0.1269313
## RMSE:  0.3562742
## LogLoss:  0.392992
## Mean Per-Class Error:  0.1203209
```

```
## AUC:  0.899287
## AUCPR:  0.8621024
## Gini:  0.798574
## R^2:  0.4343524
##
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
##         0  1    Error   Rate
## 0      27  6 0.181818  =6/33
## 1       1 16 0.058824  =1/17
## Totals 28 22 0.140000  =7/50
##
## Maximum Metrics: Maximum metrics at their respective thresholds
##                         metric threshold     value idx
## 1                       max f1  0.187380  0.820513  16
## 2                       max f2  0.187380  0.888889  16
## 3                 max f0point5  0.851785  0.849057   3
## 4                 max accuracy  0.187380  0.860000  16
## 5                max precision  0.879403  1.000000   0
## 6                   max recall  0.062365  1.000000  22
## 7              max specificity  0.879403  1.000000   0
## 8             max absolute_mcc  0.187380  0.724666  16
## 9   max min_per_class_accuracy  0.371182  0.818182  14
## 10 max mean_per_class_accuracy  0.187380  0.879679  16
## 11                     max tns  0.879403 33.000000   0
## 12                     max fns  0.879403 11.000000   0
## 13                     max fps  0.062365 33.000000  22
## 14                     max tps  0.062365 17.000000  22
## 15                     max tnr  0.879403  1.000000   0
## 16                     max fnr  0.879403  0.647059   0
## 17                     max fpr  0.062365  1.000000  22
## 18                     max tpr  0.062365  1.000000  22
##
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/
```

Train a stacked ensemble using the GBM grid

```
ensemble <- h2o.stackedEnsemble(
  x = X, y = Y, training_frame = train_h2o, model_id = "ensemble_gbm_grid",
  base_models = random_grid@model_ids, metalearner_algorithm = "gbm"
)
```

```
##   |                                                                              |
```

Eval ensemble performance on a test set

```
h2o.performance(ensemble, newdata = test_h2o)
```

```
## H2OBinomialMetrics: stackedensemble
##
```

```
## MSE:  0.1345874
## RMSE:  0.3668616
## LogLoss:  0.5044706
## Mean Per-Class Error:  0.1203209
## AUC:  0.8823529
## AUCPR:  0.7572798
## Gini:  0.7647059
##
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
##         0  1    Error   Rate
## 0      27  6 0.181818  =6/33
## 1       1 16 0.058824  =1/17
## Totals 28 22 0.140000  =7/50
##
## Maximum Metrics: Maximum metrics at their respective thresholds
##                         metric threshold     value idx
## 1                       max f1  0.369886  0.820513  21
## 2                       max f2  0.369886  0.888889  21
## 3                 max f0point5  0.369886  0.761905  21
## 4                 max accuracy  0.369886  0.860000  21
## 5                max precision  0.995542  1.000000   0
## 6                   max recall  0.009455  1.000000  31
## 7              max specificity  0.995542  1.000000   0
## 8             max absolute_mcc  0.369886  0.724666  21
## 9   max min_per_class_accuracy  0.569875  0.818182  19
## 10 max mean_per_class_accuracy  0.369886  0.879679  21
## 11                     max tns  0.995542 33.000000   0
## 12                     max fns  0.995542 16.000000   0
## 13                     max fps  0.002901 33.000000  49
## 14                     max tps  0.009455 17.000000  31
## 15                     max tnr  0.995542  1.000000   0
## 16                     max fnr  0.995542  0.941176   0
## 17                     max fpr  0.002901  1.000000  49
## 18                     max tpr  0.009455  1.000000  31
##
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/
```

Use AutoML to find a list of candidate models (i.e., leaderboard)

```
auto_ml <- h2o.automl(
  x = X, y = Y, training_frame = train_h2o, nfolds = 5,
  max_runtime_secs = 60 * 120, max_models = 10,#max_models=50
  keep_cross_validation_predictions = TRUE, sort_metric = "logloss", seed = 123,
  stopping_rounds = 50, stopping_metric = "logloss", stopping_tolerance = 0
)
```

```
##   |                                                                         |
## 23:59:11.760: Stopping tolerance set by the user is < 70% of the recommended default of 0.05, so mode
## 23:59:11.761: AutoML: XGBoost is not available; skipping it.  |
## 23:59:29.104: _min_rows param, The dataset size is too small to split for min_rows=100.0: must have a
```

Assess the leader board; the following truncates the results to show the top

and bottom 15 models. You can get the top model with auto_ml@leader

```
auto_ml@leaderboard %>%
  as.data.frame() %>%
  dplyr::select(model_id, logloss) %>%
  dplyr::slice(1:25)
```

```
##                                                  model_id   logloss
## 1     StackedEnsemble_AllModels_1_AutoML_6_20221216_235911 0.2671887
## 2  StackedEnsemble_BestOfFamily_1_AutoML_6_20221216_235911 0.3137664
## 3                             GLM_1_AutoML_6_20221216_235911 0.3475365
## 4                             XRT_1_AutoML_6_20221216_235911 0.4576169
## 5                             DRF_1_AutoML_6_20221216_235911 0.4682705
## 6                    DeepLearning_1_AutoML_6_20221216_235911 0.5612953
## 7     DeepLearning_grid_1_AutoML_6_20221216_235911_model_1 0.6325772
## 8              GBM_grid_1_AutoML_6_20221216_235911_model_1 0.6358521
## 9                             GBM_4_AutoML_6_20221216_235911 0.7124804
## 10                            GBM_2_AutoML_6_20221216_235911 0.8188022
## 11                            GBM_3_AutoML_6_20221216_235911 0.8235714
## 12                            GBM_5_AutoML_6_20221216_235911 1.1416549
```

Compute predicted probabilities on training data

```
train_h2o = as.h2o(radiomics_train)
```

```
##   |                                                                      |
```

```
m1_prob <- predict(auto_ml@leader, train_h2o, type = "prob")
```

```
##   |                                                                      |
```

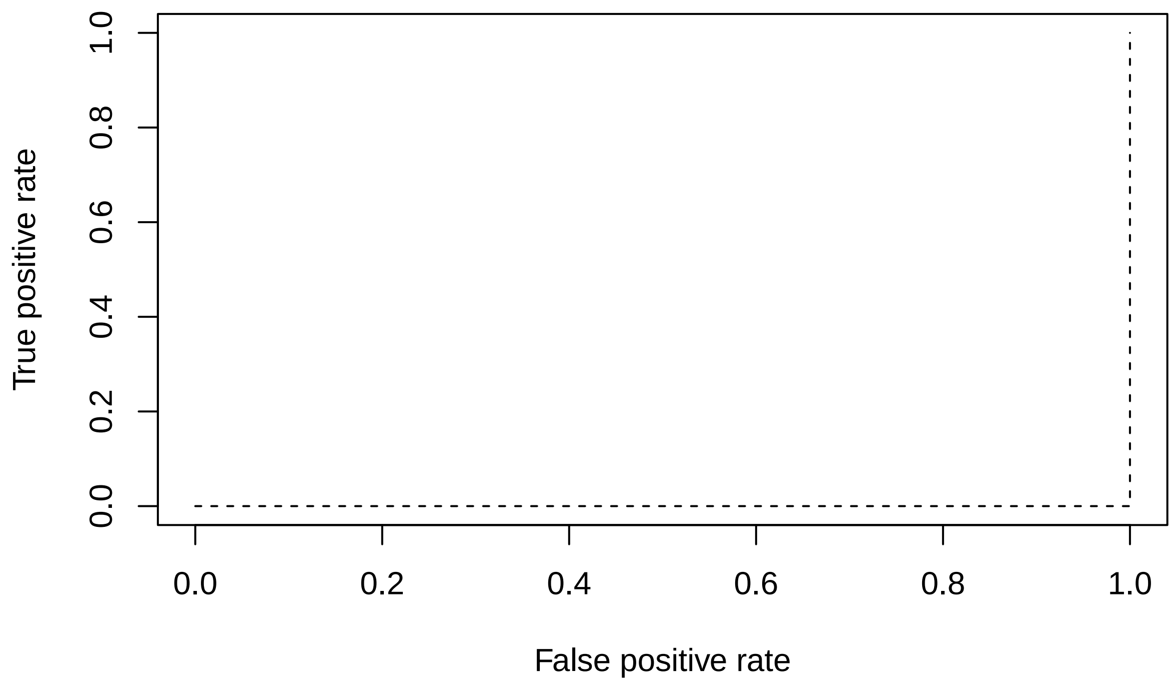
```
m1_prob = as.data.frame(m1_prob)[,2]

train_h2o = as.data.frame(train_h2o)
```

Compute AUC metrics

```
perf1 <- prediction(m1_prob,train_h2o$Failure.binary) %>%
  performance(measure = "tpr", x.measure = "fpr")
```

Plot AUC

```
plot(perf1, col = "black", lty = 2)
```

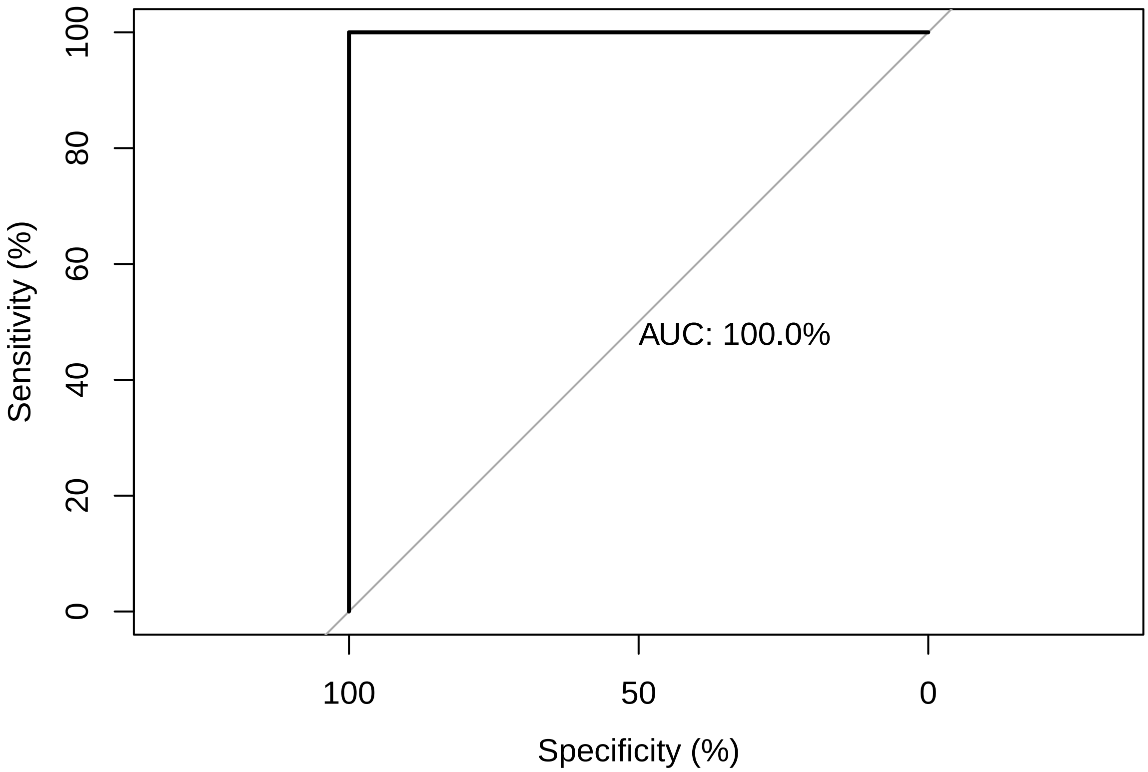


ROC plot for training data

```
roc(train_h2o$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 = train_h2o$Failure.binary ~ m1_prob, plot = TRUE,     legacy.axes = FALSE, perc
##
## Data: m1_prob in 97 controls (train_h2o$Failure.binary 0) > 50 cases (train_h2o$Failure.binary 1).
## Area under the curve: 100%
```

The performance during training has an AUC of 1.0 whose predictions are 100% correct.

Compute predicted probabilities on testing data

```
test_h2o = as.h2o(radiomics_test)
```

```
##   |                                                                              |
```

```
m2_prob <- predict(auto_ml@leader, test_h2o, type = "prob")
```

```
##   |                                                                              |
```

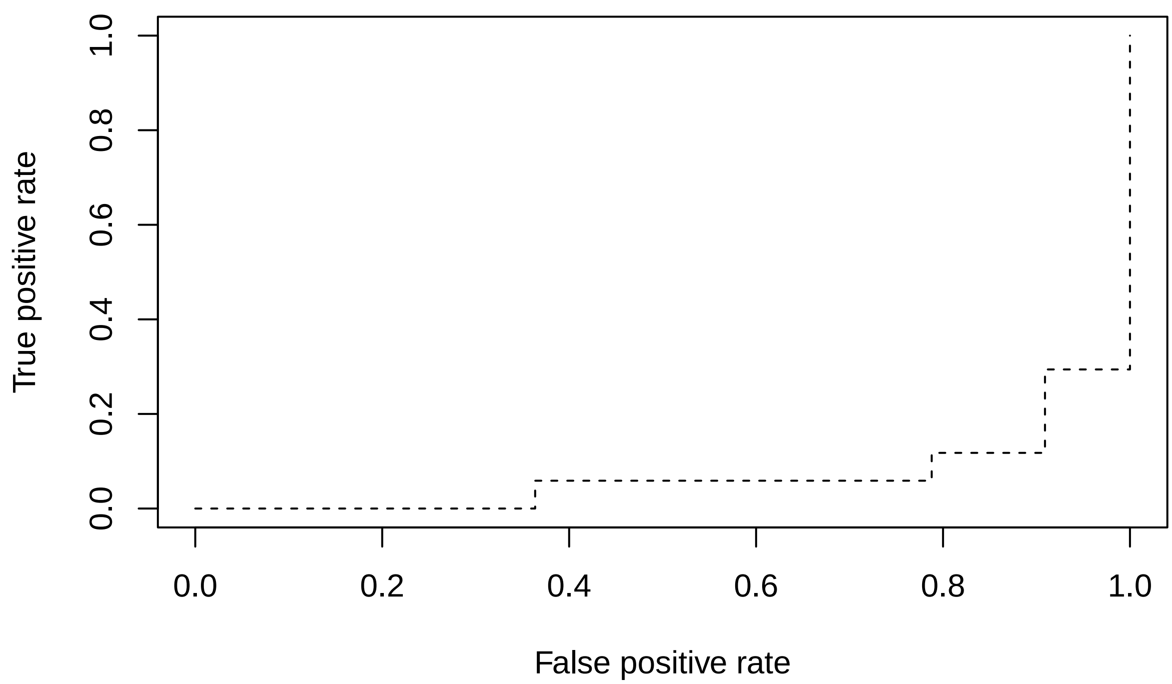
```
m2_prob=as.data.frame(m2_prob)[,2]

test_h2o=as.data.frame(test_h2o)
```

Compute AUC metrics

```
perf2 <- prediction(m2_prob,test_h2o$Failure.binary) %>%
  performance(measure = "tpr", x.measure = "fpr")
```

Plot AUC

```
plot(perf2, col = "black", lty = 2)
```

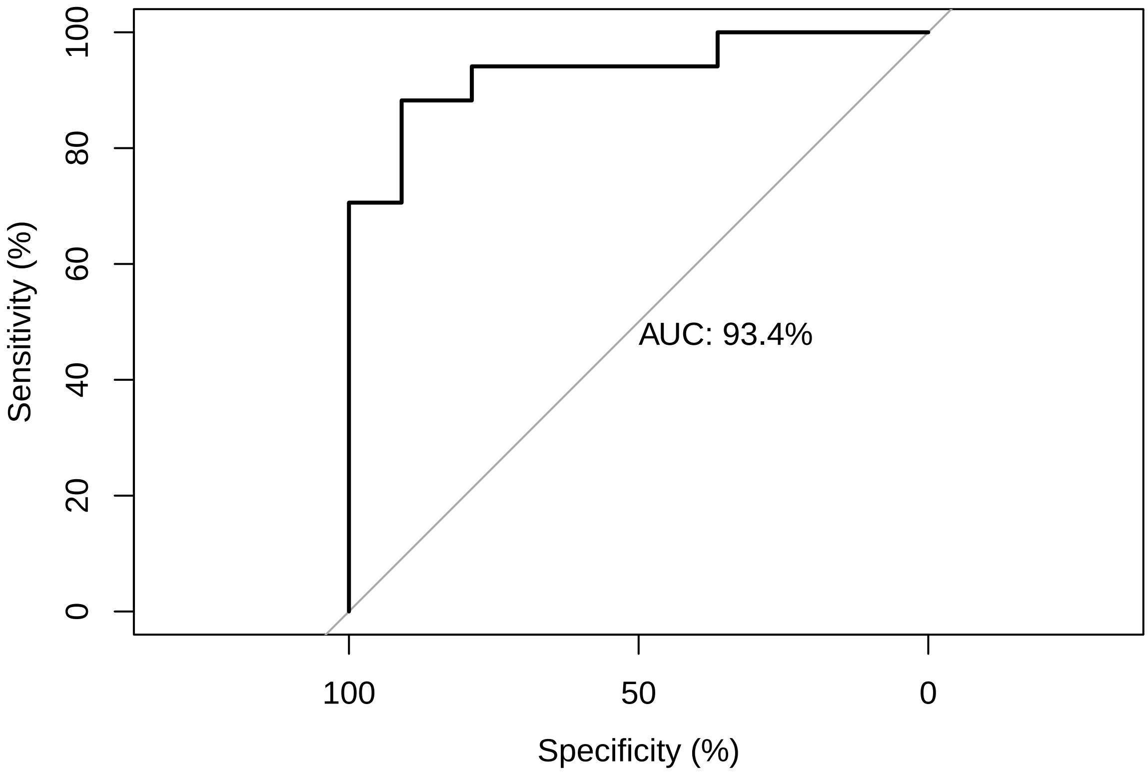


ROC plot for testing data

```
roc(test_h2o$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 = test_h2o$Failure.binary ~ m2_prob, plot = TRUE,     legacy.axes = FALSE, percen
##
## Data: m2_prob in 33 controls (test_h2o$Failure.binary 0) > 17 cases (test_h2o$Failure.binary 1).
## Area under the curve: 93.4%
```

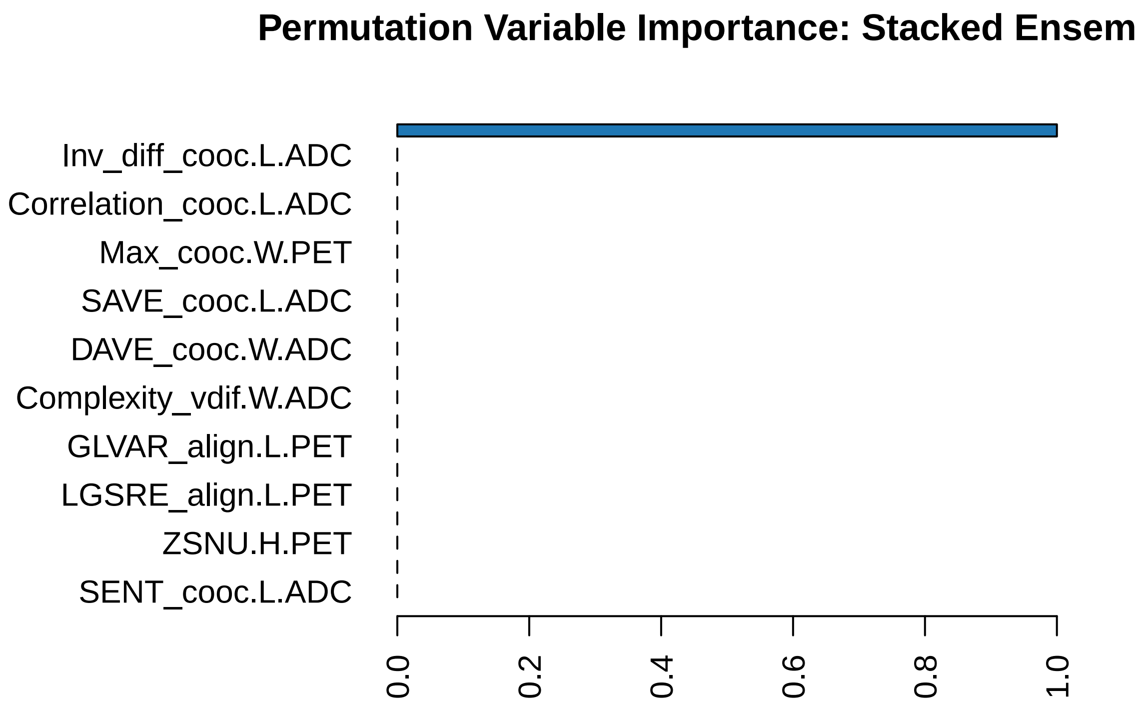
The performance during testing has the AUC of 92.9% which indicates that its area under the curve is high.

Plot the top 20 feature importance during training

```
train_h2o = as.h2o(train_h2o)
```

```
##   |                                                                              |
```

```
h2o.permutation_importance_plot(auto_ml@leader,train_h2o,num_of_features = 20)
```

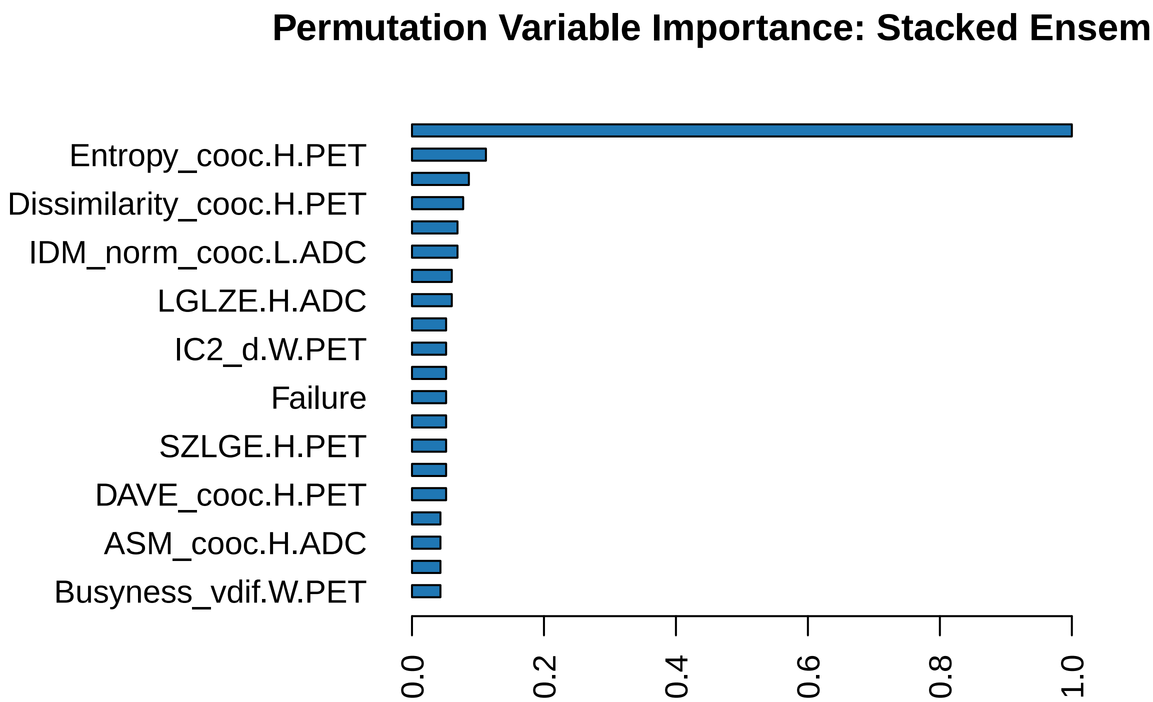


Plot the top 20 feature importance during testing

```
test_h2o = as.h2o(test_h2o)
```

```
##   |                                                                              |
```

```
h2o.permutation_importance_plot(auto_ml@leader,test_h2o,num_of_features = 20)
```


Permutation Variable Importance: Stacked Ensem
Entropy_cooc.H.PET
Dissimilarity_cooc.H.PET
IDM_norm_cooc.L.ADC
LGLZE.H.ADC
IC2_d.W.PET
Failure
SZLGE.H.PET
DAVE_cooc.H.PET
ASM_cooc.H.ADC
Busyness_vdif.W.PET
0.0
0.2
0.4
0.6
0.8
1.0