MODEL 1 STACKING MODEL

JUDISMA SALI

2022-12-16

```r
# Helper packages
library(rsample)   # for creating our train-test splits
library(recipes)   # for minor feature engineering tasks
```

```
## Loading required package: dplyr

##
## 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
```

```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 stringr::fixed() masks recipes::fixed()
## x dplyr::lag()     masks stats::lag()
```

```r
library(readr)     #load dataset

# Modeling packages
library(ROCR)
library(pROC)
```

```
## Type 'citation("pROC")' for a citation.
##
## Attaching package: 'pROC'
##
```

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

```r
library(h2o)       # for fitting stacked models
```

```
##
## ----------------------------------------------------------------------
##
## 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 object is masked from 'package:pROC':
##
##     var
##
## 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
h2o.init()
```

```
##  Connection successful!
##
## R is connected to the H2O cluster:
##     H2O cluster uptime:         8 hours 43 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:   3.59 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)
```

```
h2o.init()
```

```
##  Connection successful!
##
## R is connected to the H2O cluster:
##     H2O cluster uptime:         8 hours 43 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:   3.59 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)
```

MODEL 1 STACKING”

Stacking is a process where the data is transformed, and variables (columns) can be rearranged to act as cases (rows). This is sometimes called hierarchical data.

LOAD THE REPROCESSED DATASET

Note that we used the reprocessed data of radiomics_complete.csv (*RAD. NORMAL DATA.CSV*) in performing stacking.

Radiomics Dataset 197 Rows (Observations) of 431 Columns (Variables) Failure.binary: binary property to predict

```
radiomicsdt <- read_csv("RAD. NORMAL DATA.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(radiomicsdt)
head(radiomicsdt)
```

```
## # A tibble: 6 x 431
##   Institution Failure.~1 Failure Entro~2 GLNU_~3 Min_h~4 Max_h~5 Mean_~6 Varia~7
##   <chr>            <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
## 1 A                      0   1.15     12.9  -0.433  -0.270 -0.257   -0.192  0.0509
## 2 A                      1  -0.533    12.2  -1.02    0.671  0.405    0.490  0.687
## 3 A                      0   2.24     12.8   0.179  -1.41  -1.57    -1.53  -1.57
## 4 A                      1  -0.140    13.5   2.00   -0.218  0.0764  -0.153  0.0127
## 5 A                      0   0.787    12.6   0.153  -1.06  -1.15    -1.45  -1.91
## 6 A                      1  -2.80     13.2   0.391  -1.57  -1.91    -1.72  -1.84
## # ... with 422 more variables: Standard_Deviation_hist.PET <dbl>,
## #   Skewness_hist.PET <dbl>, Kurtosis_hist.PET <dbl>, Energy_hist.PET <dbl>,
## #   Entropy_hist.PET <dbl>, AUC_hist.PET <dbl>, H_suv.PET <dbl>,
## #   Volume.PET <dbl>, X3D_surface.PET <dbl>, ratio_3ds_vol.PET <dbl>,
## #   ratio_3ds_vol_norm.PET <dbl>, irregularity.PET <dbl>,
## #   tumor_length.PET <dbl>, Compactness_v1.PET <dbl>, Compactness_v2.PET <dbl>,
## #   Spherical_disproportion.PET <dbl>, Sphericity.PET <dbl>, ...
```

CHECKING FOR NULL AND MISSING VALUES

The result for checking null and missing values is 0 using *sum(is.n())*. Thus, there is no null and missing values.

```
sum(is.na(radiomicsdt))
```

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

```
set.seed(123)  # for reproducibility

radiomicsdt<- read_csv("RAD. NORMAL DATA.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.
```

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


split <- initial_split(radiomicsdt, strata = "Failure.binary")
traindt <- training(split)
testdt <- testing(split)
```

```
# Make sure we have consistent categorical levels
blueprint <- recipe(Failure.binary ~ ., data = traindt) %>%
  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:         8 hours 43 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:   3.59 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)
```

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

```
##   |                                                                      |
```

```r
test_h2o <- prep(blueprint, training = traindt) %>%
  bake(new_data = testdt) %>%
  as.h2o()
```

```
##   |                                                                      |
```

```r
# Get response and feature names
Y <- "Failure.binary"
```

```r
X <- setdiff(names(traindt), 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
)
```

```
##   |                                                                      |
```

```r
# Train & cross-validate a RF model
best_rf <- h2o.randomForest(
  x = X, y = Y, training_frame = train_h2o, ntrees = 1000, 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
```

```
##   |                                                                      |
```

```r
# Train & cross-validate a GBM model
best_gbm <- h2o.gbm(
  x = X, y = Y, training_frame = train_h2o, ntrees = 1000, 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 0.3918207 0.3060996
```

```
## [1] 30024.67 23075.24 20859.92 21391.20
```

```
# 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] 0.3831863
## [1] 20664.56

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  1.0000000000 0.04363592 -0.0004822025
## RF_pred   0.0436359202 1.00000000  0.6638666821
## GBM_pred -0.0004822025 0.66386668  1.0000000000
```

```
# 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              1.00000   5.00000  1.00000
## 2         0.90000    0.10000              1.00000   5.00000  5.00000
## 3         0.80000    0.10000              1.00000   3.00000  5.00000
## 4         0.90000    0.10000              0.99000   5.00000  5.00000
## 5         0.80000    0.10000              0.99000   5.00000  5.00000
##   sample_rate         model_ids logloss
## 1     0.50000 gbm_grid_model_10 0.29671
## 2     1.00000 gbm_grid_model_11 0.30636
## 3     1.00000 gbm_grid_model_20 0.30718
## 4     1.00000 gbm_grid_model_15 0.31242
## 5     0.75000 gbm_grid_model_21 0.32082
##
## ---
##    col_sample_rate learn_rate learn_rate_annealing max_depth min_rows
## 20         0.90000    0.01000              0.99000   5.00000  5.00000
## 21         1.00000    0.01000              1.00000   1.00000  1.00000
## 22         1.00000    0.01000              1.00000   1.00000  5.00000
## 23         0.80000    0.01000              1.00000   1.00000  1.00000
## 24         1.00000    0.01000              0.99000   1.00000 10.00000
## 25         0.90000    0.01000              0.99000   1.00000  5.00000
##    sample_rate         model_ids logloss
## 20     0.50000  gbm_grid_model_8 0.55749
## 21     1.00000 gbm_grid_model_19 0.55862
## 22     0.50000  gbm_grid_model_3 0.55875
## 23     0.75000  gbm_grid_model_7 0.55954
## 24     0.50000  gbm_grid_model_4 0.56408
## 25     0.75000 gbm_grid_model_18 0.56479
```

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

```
# Grab the model_id for the top model, chosen by validation error
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:  0.07905221
## RMSE:  0.2811623
## LogLoss:  0.2720538
## Mean Per-Class Error:  0.1033868
## AUC:  0.9625668
## AUCPR:  0.9332294
## Gini:  0.9251337
## R^2:  0.6477174
##
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
##         0  1    Error   Rate
## 0      32  1 0.030303  =1/33
## 1       3 14 0.176471  =3/17
## Totals 35 15 0.080000  =4/50
##
## Maximum Metrics: Maximum metrics at their respective thresholds
##                         metric threshold     value idx
## 1                       max f1  0.555940  0.875000  14
## 2                       max f2  0.262300  0.909091  19
## 3                 max f0point5  0.555940  0.909091  14
## 4                 max accuracy  0.555940  0.920000  14
## 5                max precision  0.922680  1.000000   0
## 6                   max recall  0.099787  1.000000  25
## 7              max specificity  0.922680  1.000000   0
## 8             max absolute_mcc  0.555940  0.819972  14
## 9   max min_per_class_accuracy  0.465786  0.882353  17
## 10 max mean_per_class_accuracy  0.262300  0.909982  19
## 11                     max tns  0.922680 33.000000   0
## 12                     max fns  0.922680 16.000000   0
## 13                     max fps  0.042504 33.000000  47
## 14                     max tps  0.099787 17.000000  25
## 15                     max tnr  0.922680  1.000000   0
## 16                     max fnr  0.922680  0.941176   0
## 17                     max fpr  0.042504  1.000000  47
## 18                     max tpr  0.099787  1.000000  25
```

```
##
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/
```

```r
# 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"
)
```

```
##   |                                                                              |
```

```r
# Eval ensemble performance on a test set
h2o.performance(ensemble, newdata = test_h2o)
```

```
## H2OBinomialMetrics: stackedensemble
##
## MSE:  0.1468317
## RMSE:  0.3831863
## LogLoss:  0.4519484
## Mean Per-Class Error:  0.09090909
## AUC:  0.9269162
## AUCPR:  0.8525781
## Gini:  0.8538324
##
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
##         0  1    Error   Rate
## 0      27  6 0.181818  =6/33
## 1       0 17 0.000000  =0/17
## Totals 27 23 0.120000  =6/50
##
## Maximum Metrics: Maximum metrics at their respective thresholds
##                         metric threshold     value idx
## 1                       max f1  0.050958  0.850000  22
## 2                       max f2  0.050958  0.934066  22
## 3                 max f0point5  0.947350  0.819672  10
## 4                 max accuracy  0.050958  0.880000  22
## 5                max precision  0.993694  1.000000   0
## 6                   max recall  0.050958  1.000000  22
## 7              max specificity  0.993694  1.000000   0
## 8             max absolute_mcc  0.050958  0.777652  22
## 9   max min_per_class_accuracy  0.276801  0.818182  19
## 10 max mean_per_class_accuracy  0.050958  0.909091  22
## 11                     max tns  0.993694 33.000000   0
## 12                     max fns  0.993694 16.000000   0
## 13                     max fps  0.003454 33.000000  49
## 14                     max tps  0.050958 17.000000  22
## 15                     max tnr  0.993694  1.000000   0
## 16                     max fnr  0.993694  0.941176   0
## 17                     max fpr  0.003454  1.000000  49
## 18                     max tpr  0.050958  1.000000  22
##
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/
```

```r
# 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 = 10, stopping_metric = "logloss", stopping_tolerance = 0
)
```

```
##   |                                                                              |
## 23:32:01.77: Stopping tolerance set by the user is < 70% of the recommended default of 0.05, so model
## 23:32:01.79: AutoML: XGBoost is not available; skipping it.  |
## 23:32:04.511: _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_4_20221216_233201 0.2684350
## 2  StackedEnsemble_BestOfFamily_1_AutoML_4_20221216_233201 0.2741078
## 3                            GBM_4_AutoML_4_20221216_233201 0.3046628
## 4                            GBM_3_AutoML_4_20221216_233201 0.3304561
## 5               GBM_grid_1_AutoML_4_20221216_233201_model_1 0.3308175
## 6                            GLM_1_AutoML_4_20221216_233201 0.3475365
## 7                            GBM_2_AutoML_4_20221216_233201 0.3626937
## 8                            GBM_5_AutoML_4_20221216_233201 0.4063628
## 9                            XRT_1_AutoML_4_20221216_233201 0.4576169
## 10                           DRF_1_AutoML_4_20221216_233201 0.4682705
## 11                  DeepLearning_1_AutoML_4_20221216_233201 0.6543593
## 12   DeepLearning_grid_1_AutoML_4_20221216_233201_model_1 0.9730544
```

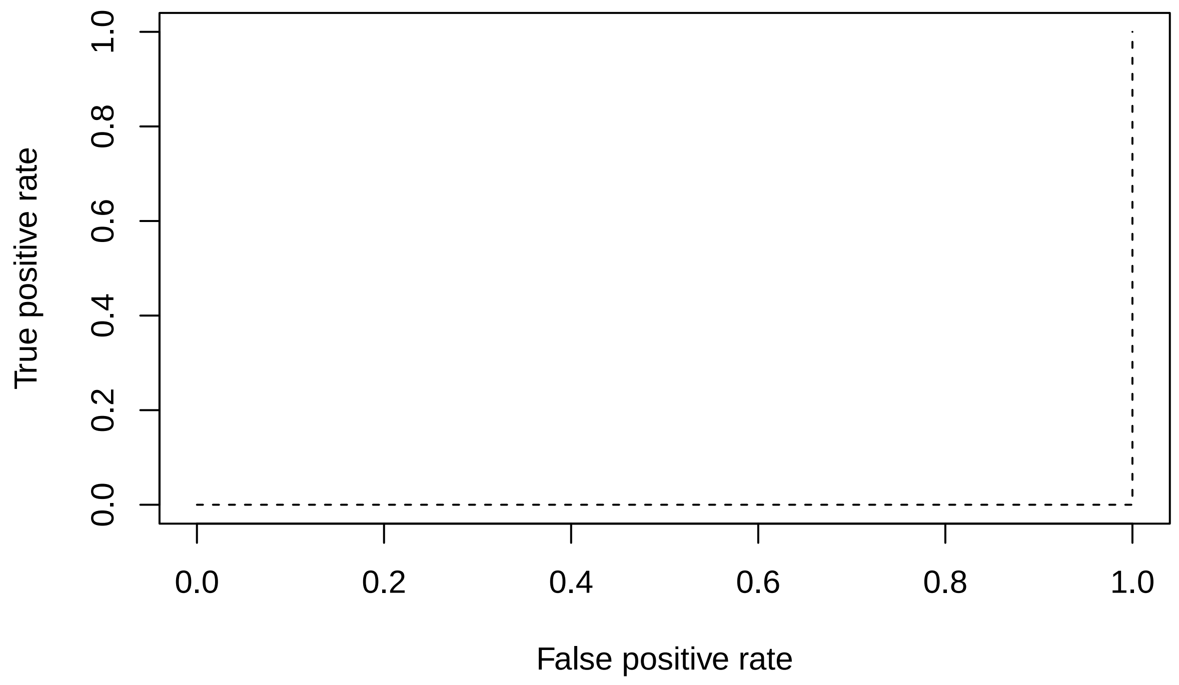
```
# Compute predicted probabilities on training data
train_h2o=as.h2o(traindt)
```

```
##   |                                                                              |
```

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

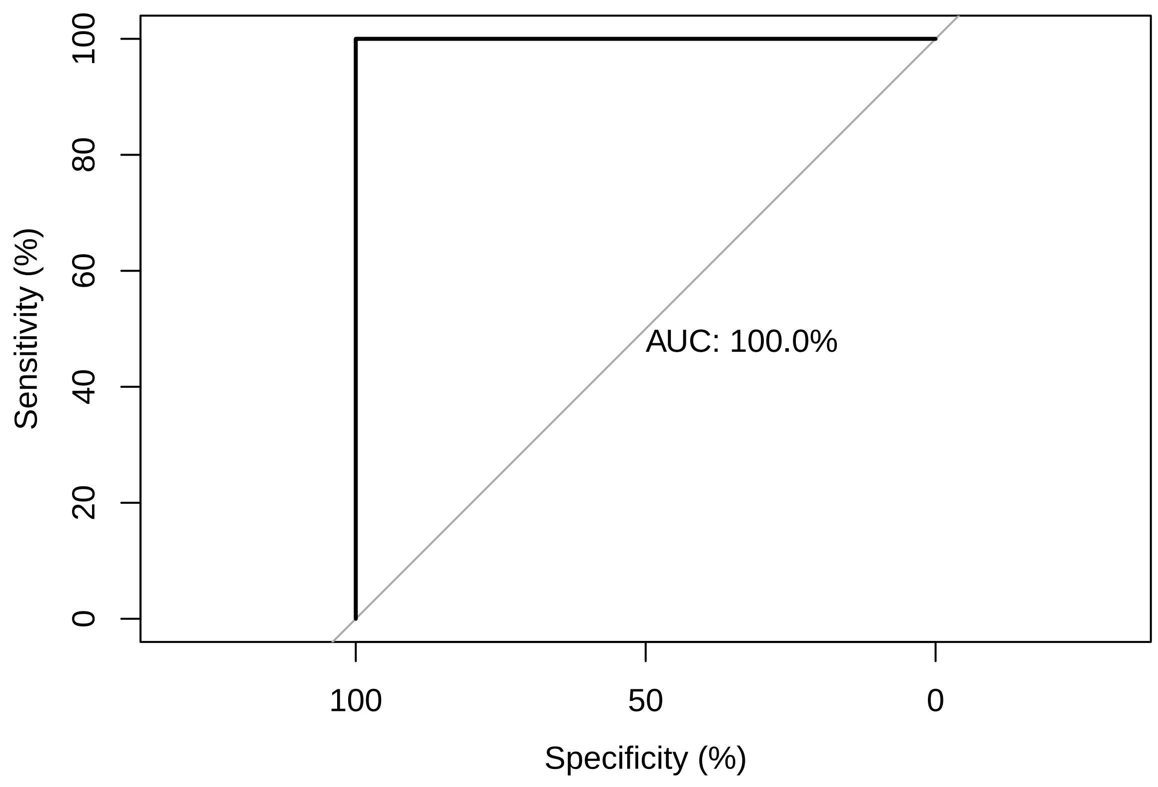
```
##   |                                                                              |
```

```
m1_prob=as.data.frame(m1_prob)[,2]
train_h2o=as.data.frame(train_h2o)
# Compute AUC metrics for cv_model1,2 and 3
perf1 <- prediction(m1_prob,train_h2o$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( 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%
```

```
#
# #Feature Interpretation
# vip(cv_model3, num_features = 20)

# Compute predicted probabilities on training data
test_h2o=as.h2o(testdt)
```

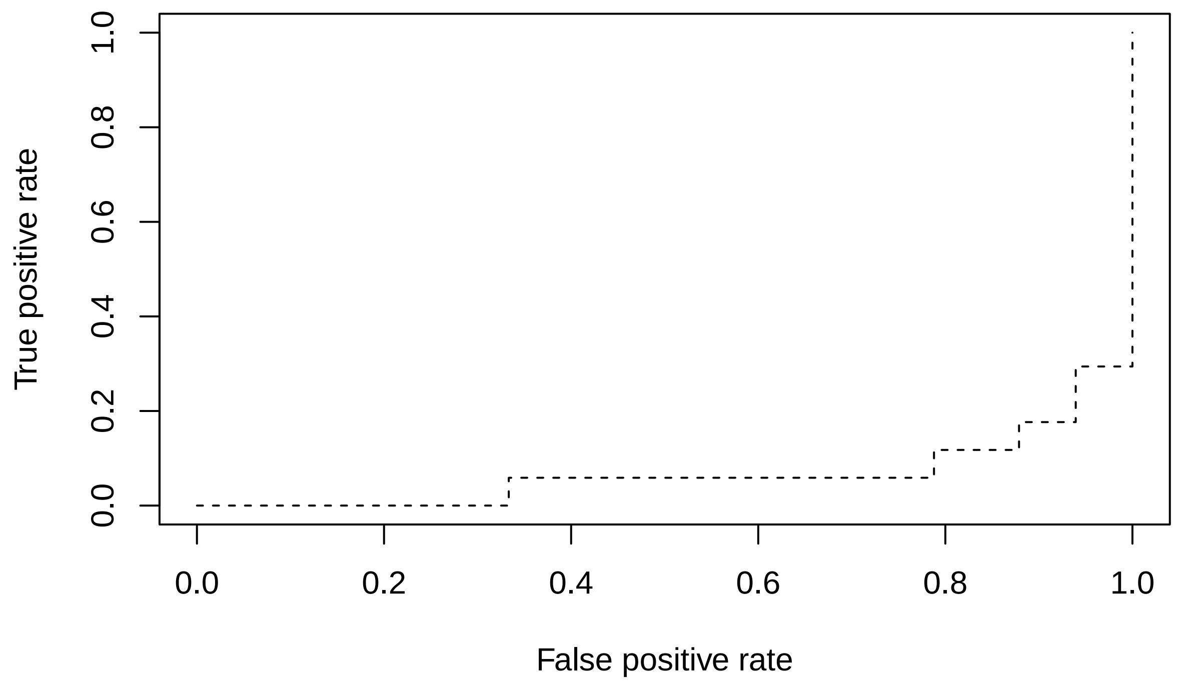
```
##   |                                                                                                    |
```

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

```
##   |                                                                                                    |
```

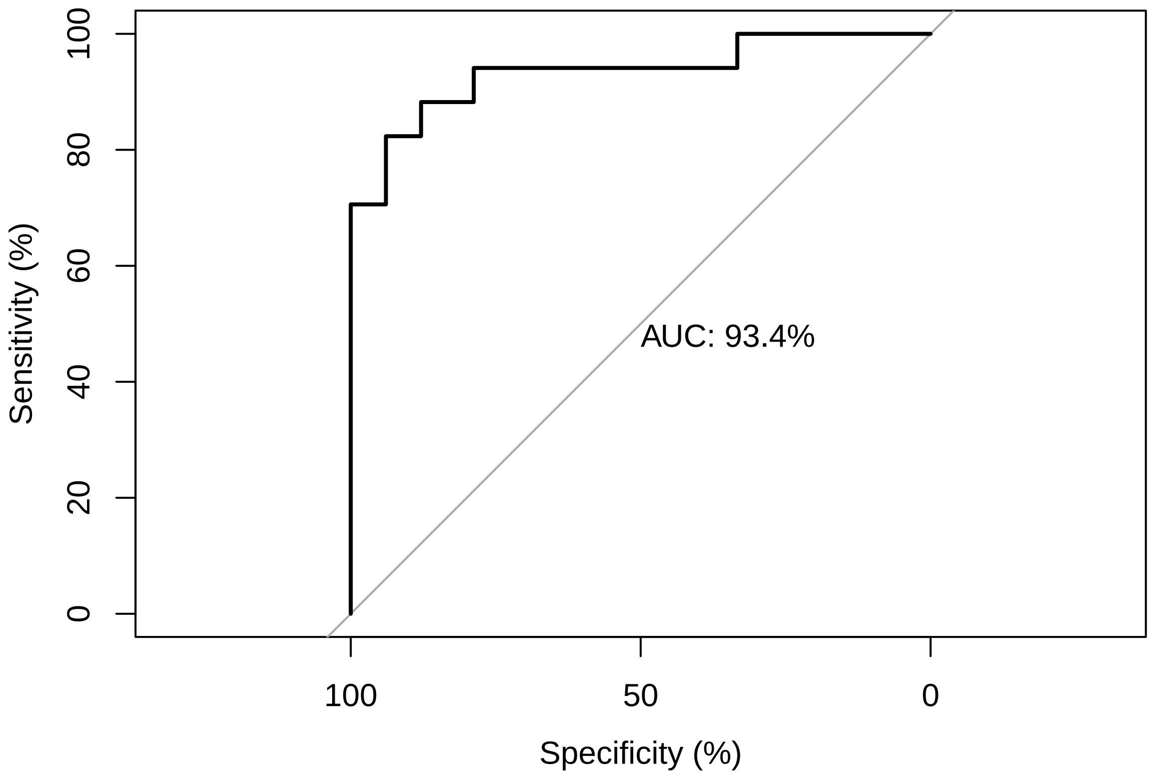
```
m2_prob=as.data.frame(m2_prob)[,2]

test_h2o=as.data.frame(test_h2o)

# Compute AUC metrics for cv_model1,2 and 3
perf2 <- prediction(m2_prob,test_h2o$Failure.binary) %>%
  performance(measure = "tpr", x.measure = "fpr")
```

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



```r
# ROC plot for training 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
```

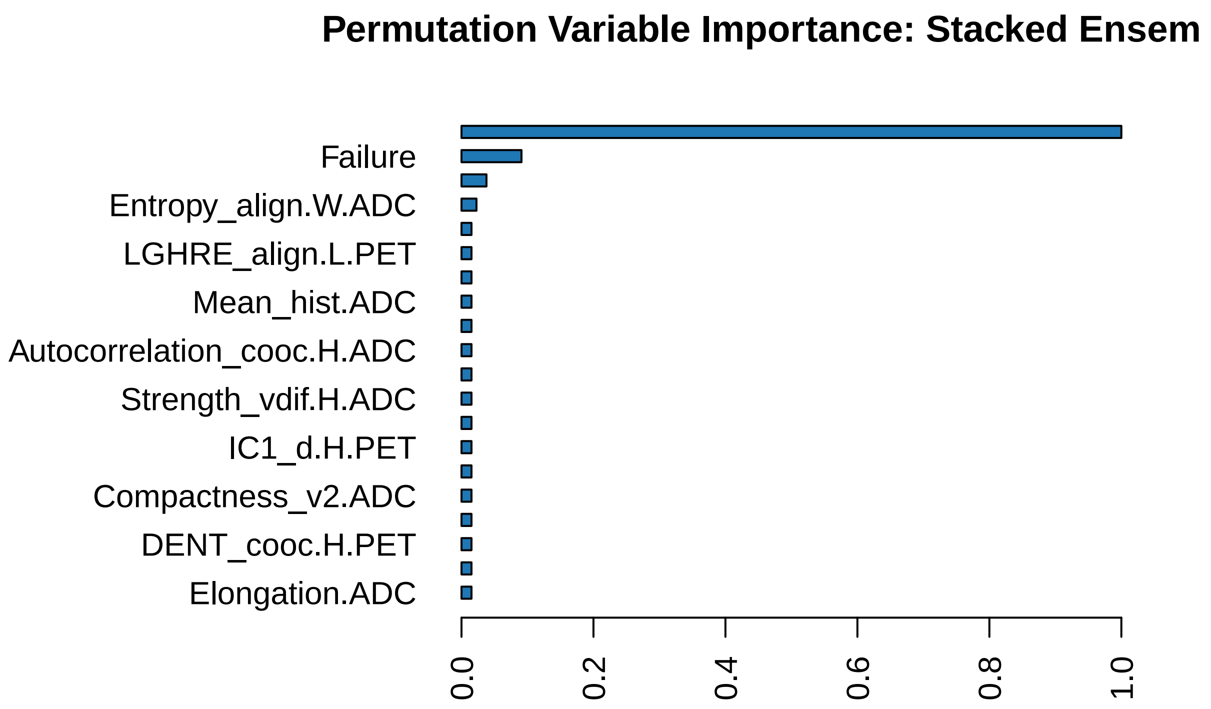


```
##
## Call:
## roc.formula(formula = test_h2o$Failure.binary ~ m2_prob, plot = TRUE,     legacy.axes = FALSE, percen
##
## Data: m2_prob in 33 controls (test_h2o$Failure.binary 0) > 17 cases (test_h2o$Failure.binary 1).
## Area under the curve: 93.4%
```

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

```
##   |                                                                              |
```

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


Permutation Variable Importance: Stacked Ensem
Failure
Entropy_align.W.ADC
LGHRE_align.L.PET
Mean_hist.ADC
Autocorrelation_cooc.H.ADC
Strength_vdif.H.ADC
IC1_d.H.PET
Compactness_v2.ADC
DENT_cooc.H.PET
Elongation.ADC
0.0
0.2
0.4
0.6
0.8
1.0