INFS 692 Data Science Final Project Model1

Ivan Huang

2022-12-05

A total of *three models* should be should be performed for this final project. You should find the detailed codes and answers underneath

Model 1

Step 1: Create an ensemble classification model (at least 3 models of your choice).

Answer: For this first answer, after some scrutinized research, only *stacking* among all ensemble methods can be performed in three different approaches. Therefore, I intend to use stacking models for ensemble classification model.

First, we need to implement all the essential R libraries so that we can run the codes afterwards.

```
# 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
```

```
# Modeling packages
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 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
```

```
# Other packages
library(ggplot2)
library(rpart)       # direct engine for decision tree application
library(caret) # meta engine for decision tree application
```

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

```
library(recipes)
library(dslabs)
library(purrr)
```

```
##
## Attaching package: 'purrr'
```

```
## The following object is masked from 'package:caret':
##
##     lift
```

```
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(rpart.plot)  # for plotting decision trees
library(vip)         # for feature importance
```

```
##
## Attaching package: 'vip'

## The following object is masked from 'package:utils':
##
##     vi
```

```
library(pdp)         # for feature effects
```

```
##
## Attaching package: 'pdp'

## The following object is masked from 'package:purrr':
##
##     partial
```

Step 2: Data Preprocessing

1. Introduce the sample data; *check for* ***null*** *and* ***missing values***

```
# Load data from local environment (supposedly from the same directory)
# split the data
data <- read.csv('./radiomics_completedata.csv')
which(is.na(data)) # returns integer(0) meaning there is no null or missing value
```

```
## integer(0)
```

2. Remove the categorical and binary data

```
institution <- data$Institution
i1 <- sapply(data, is.numeric) #remove all the categorical data
data <- data[i1]
Failure_binary <- data$Failure.binary
data <- Filter(function(x) !all(x %in% c(0, 1)), data) # remove all the binary data
final_data <- scale(data)

final_data <- as.data.frame(final_data)
```

3. Get the correlation of the whole data

```
cor(final_data)
```

Step 3: Split the data into training (80%) and testing (20%)

```
set.seed(123) # for reproducibility
final_data2 <- cbind(final_data, Failure_binary)
final_data3 <- cbind(final_data2, institution)
split <- initial_split(final_data3, prop = 0.8, strata = 'Failure') #prop = 0.8 as in training vs. test
data_train <- training(split)
data_test <- testing(split)
```

Step 4: Print the AUC values during Training

But before that, we need to create different training models and stack them together

```
# Make sure we have consistent categorical levels
blueprint <- recipe(Failure_binary ~ ., data = data_train) %>%
  step_other(all_nominal(), threshold = 0.005)

# Create training & test sets for h2o
h2o.init()
```

```
##
## H2O is not running yet, starting it now...
##
## Note:  In case of errors look at the following log files:
##     /var/folders/65/1mdmld216fg1cxwf27_8rzs80000gn/T//RtmpvsYd72/file71896a66f37/h2o_yifanhuang_star
##     /var/folders/65/1mdmld216fg1cxwf27_8rzs80000gn/T//RtmpvsYd72/file718918cf874a/h2o_yifanhuang_sta
##
##
## Starting H2O JVM and connecting: .... Connection successful!
##
## R is connected to the H2O cluster:
##     H2O cluster uptime:         3 seconds 56 milliseconds
##     H2O cluster timezone:       America/Toronto
##     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_yifanhuang_pxv372
##     H2O cluster total nodes:    1
##     H2O cluster total memory:   4.00 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 = data_train, retain = TRUE) %>%
  juice() %>%
  as.h2o()
```

```
##   |                                                                      |
```

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

```
##   |                                                                      |
```

```
# Get response and feature names
Y <- "Failure_binary"
X <- setdiff(names(data_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
)
```

```
## Warning in .h2o.processResponseWarnings(res): We have detected that your response column has only 2 u
```

```
##   |                                                                      |
```

```
# Train & cross-validate a RF model
best_rf <- h2o.randomForest(
  x = X, y = Y, training_frame = train_h2o, ntrees = 500, 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 = "RMSE",
  stopping_tolerance = 0
)
```

```
## Warning in .h2o.processResponseWarnings(res): We have detected that your response column has only 2 u
## early stopping is enabled but neither score_tree_interval or score_each_iteration are defined. Early
```

```
##   |                                                                      |
```

```r
# Train & cross-validate a GBM model
best_gbm <- h2o.gbm(
  x = X, y = Y, training_frame = train_h2o, ntrees = 500, 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 = "RMSE",
  stopping_tolerance = 0
)
```

```
## Warning in .h2o.processResponseWarnings(res): We have detected that your response column has only 2 u
## early stopping is enabled but neither score_tree_interval or score_each_iteration are defined. Early
```

```
##   |                                                                              |
```

```r
# Train a stacked ensemble using all the previous models

ensemble <- h2o.stackedEnsemble(x = X, y = Y, training_frame = train_h2o, base_models = list(best_glm, b
```

```
## Warning in .h2o.processResponseWarnings(res): We have detected that your response column has only 2 u
```

```
##   |                                                                              |
```

```r
# Compute predicted probabilities on training data
df_train <- as.data.frame(train_h2o)
m1_prob <- predict(ensemble, train_h2o, type = "prob")
```

```
##   |                                                                              |
```

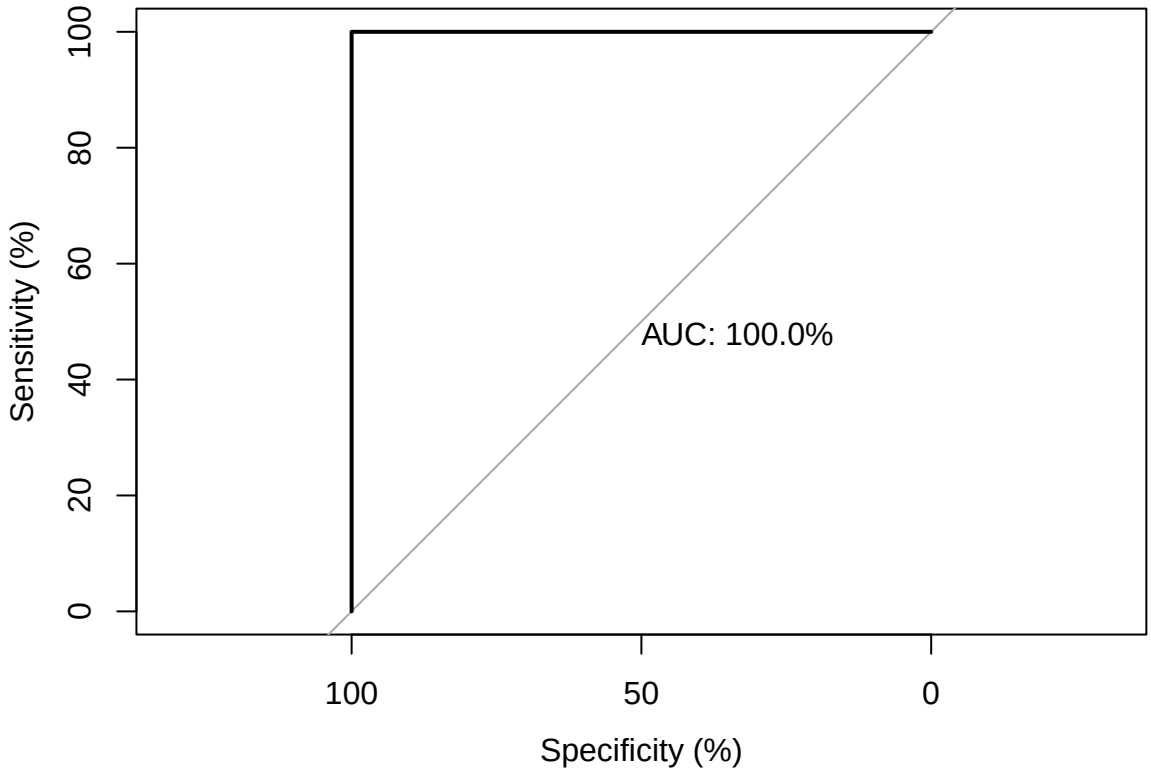
```r
df_m1prob <- as.data.frame(m1_prob)

# ROC plot for training data

roc(df_train$Failure_binary~ df_m1prob[,1], plot=TRUE, legacy.axes=FALSE,
    percent=TRUE, col="black", lwd=2, print.auc=TRUE)
```

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

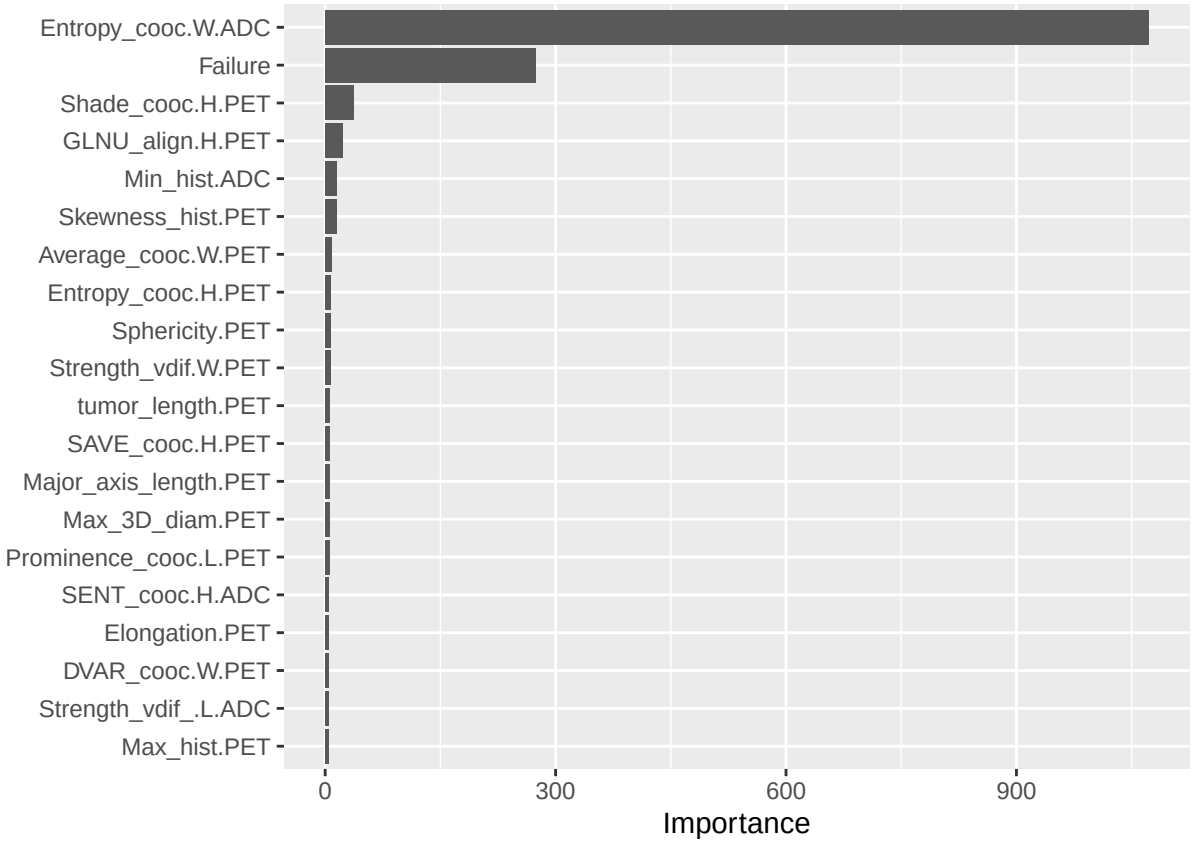
```
## Setting direction: controls < cases
```



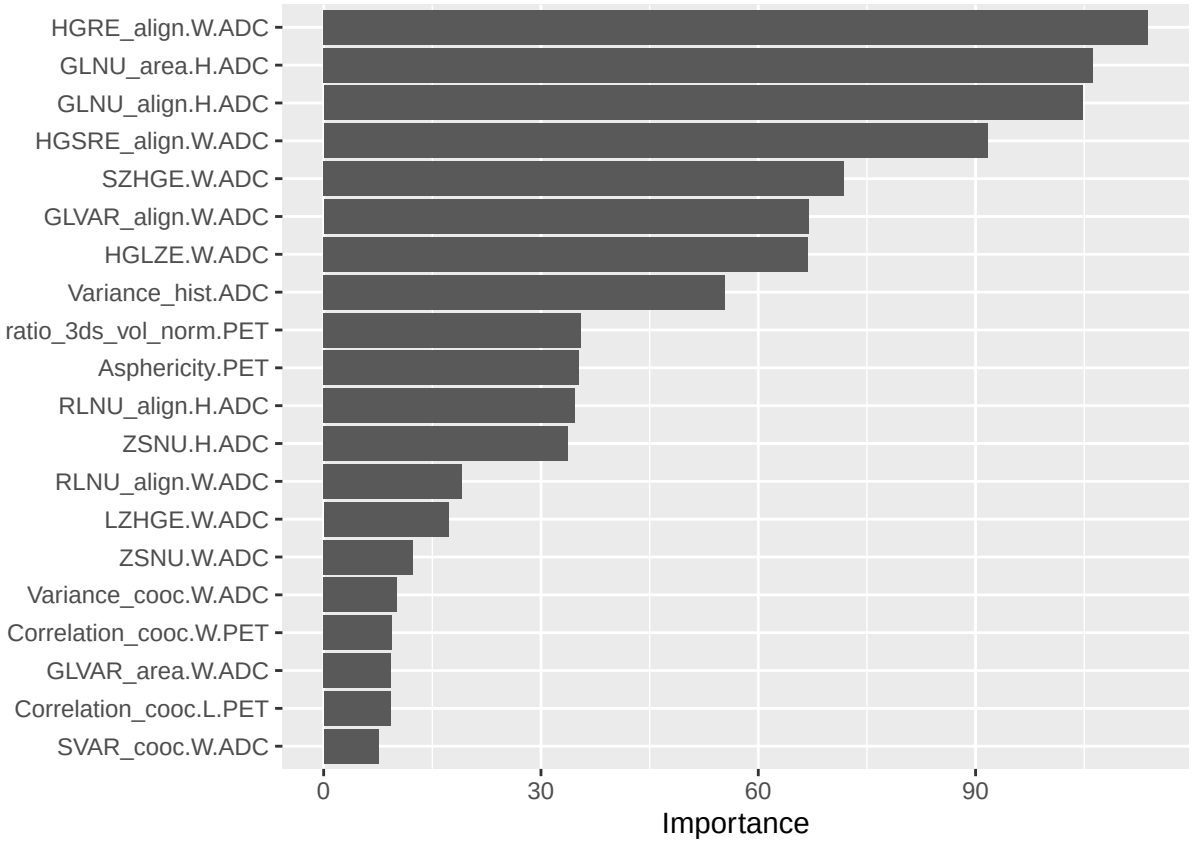

```
##
## Call:
## roc.formula(formula = df_train$Failure_binary ~ df_m1prob[, 1],     plot = TRUE, legacy.axes = FALSE
##
## Data: df_m1prob[, 1] in 105 controls (df_train$Failure_binary 0) < 52 cases (df_train$Failure_binary
## Area under the curve: 100%
```

Step 5: Print the Top 20 important features during Training

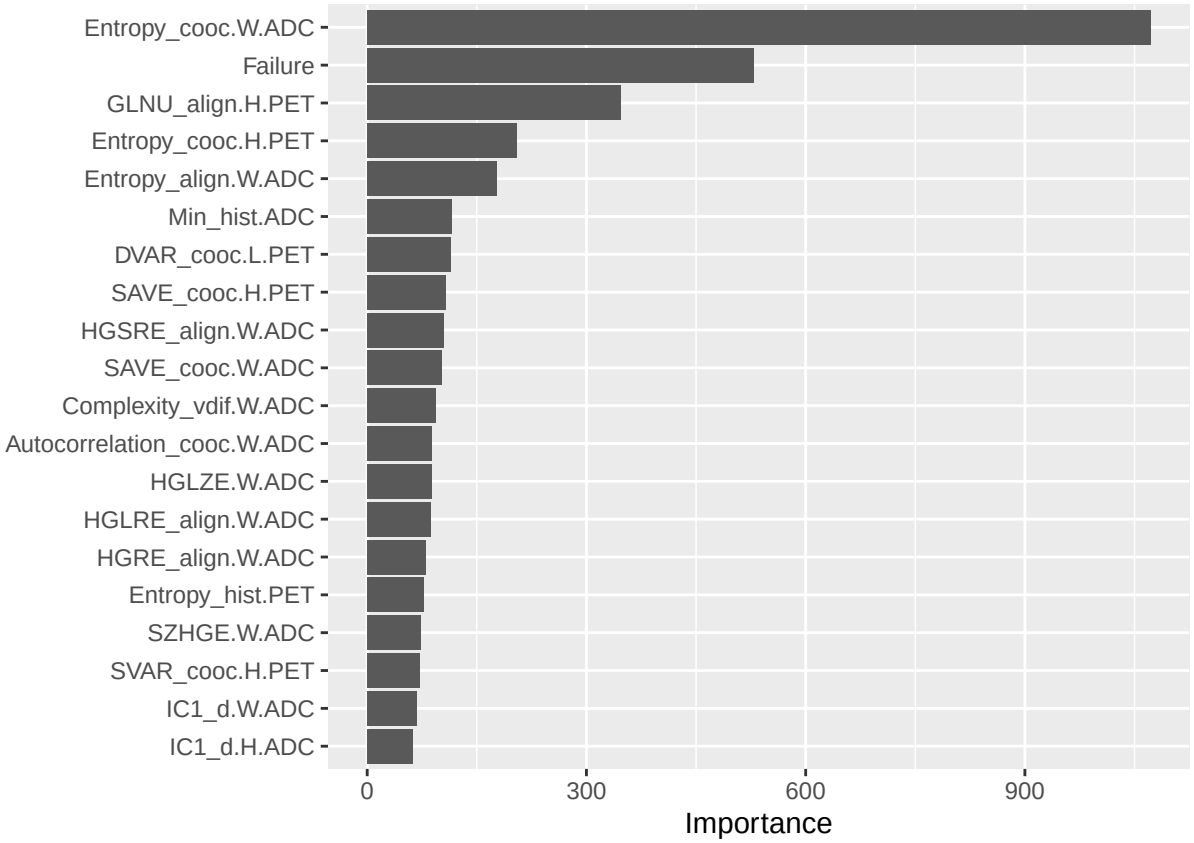
```
#feature importance
vip::vip(best_gbm, num_features = 20, bar = FALSE)
```




```
vip::vip(best_glm, num_features = 20, bar = FALSE)
```




```
vip::vip(best_rf, num_features = 20, bar = FALSE)
```



Step 6: Print the AUC values during Testing

```
# Compute predicted probabilities on testing data
df_test <- as.data.frame(test_h2o)
m2_prob <- predict(ensemble, test_h2o, type = "prob")
```

```
##   |                                                                      |
```

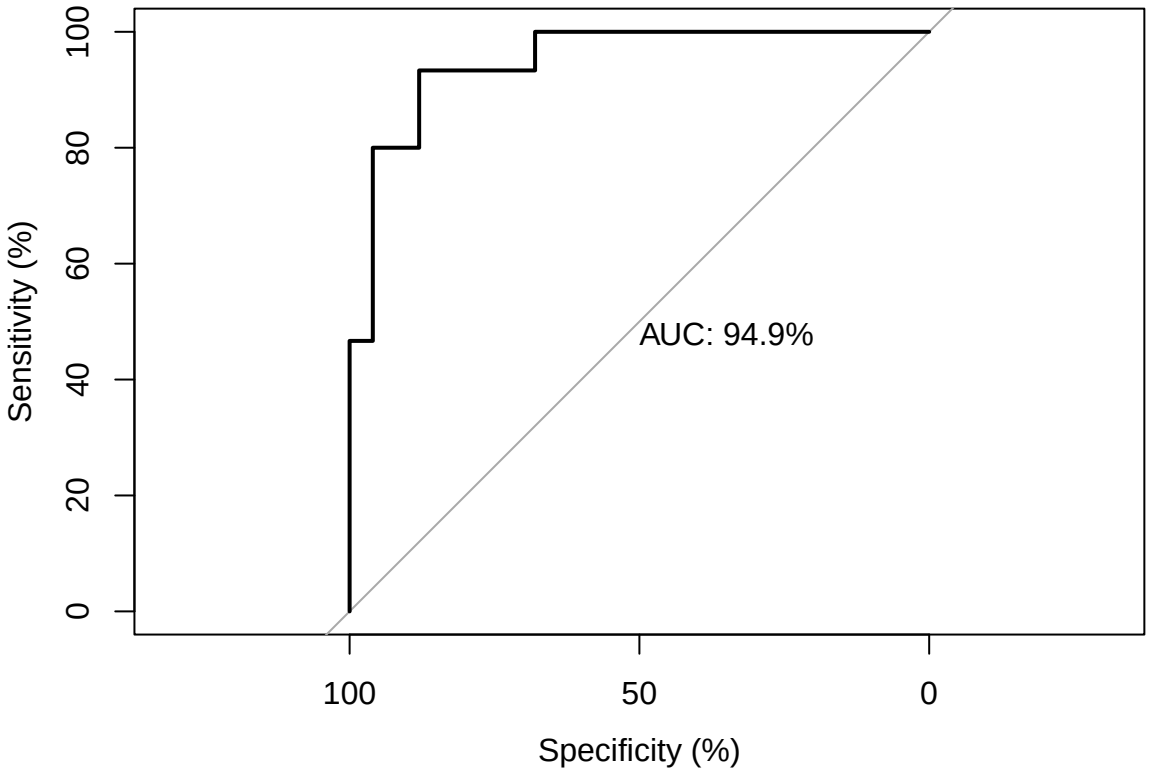
```
df_m2prob <- as.data.frame(m2_prob)

# ROC plot for testing data

roc(df_test$Failure_binary~ df_m2prob[,1], 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 = df_test$Failure_binary ~ df_m2prob[, 1],     plot = TRUE, legacy.axes = FALSE,
##
## Data: df_m2prob[, 1] in 25 controls (df_test$Failure_binary 0) < 15 cases (df_test$Failure_binary 1)
## Area under the curve: 94.93%
```