Model 1 - Stacking

Basic packages will be loaded to help with package loading, splitting the data into test and training set, the use of h2o to utilize java in the machine learning method and finaly with performance checks with the use or ROCR and and pROC

```
# Helper packages
pacman::p_load(
  pacman,
  rsample,
  recipes,
  tidyverse,
  readr,
  h2o,
  ROCR,
  pROC
               )
```

We will have to initiallize the h2o connection, allow permission incase a pop-up appears for Java. Also may require the user to run h2o.removeAll() to clear h2o environment incase of h2o errors.

```
h2o.init()
```

```
##
## H2O is not running yet, starting it now...
##
## Note:  In case of errors look at the following log files:
##     C:\Users\USER\AppData\Local\Temp\RtmpgnOwyL\filefc6230b8a/h2o_USER_started_from_r.out
##     C:\Users\USER\AppData\Local\Temp\RtmpgnOwyL\filefc282f369c/h2o_USER_started_from_r.err
##
##
## Starting H2O JVM and connecting:  Connection successful!
##
## R is connected to the H2O cluster:
##     H2O cluster uptime:         4 seconds 133 milliseconds
##     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_USER_dgp320
##     H2O cluster total nodes:    1
##     H2O cluster total memory:   1.95 GB
##     H2O cluster total cores:    4
##     H2O cluster allowed cores:  4
##     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)
```

Loading data, factoring target variable for classification and splittign data based on the classifier as strata into training and testing set.

```
set.seed(123)  # for reproducibility
DF= read_csv("CleanedDF.csv")
```

```
## New names:
## Rows: 197 Columns: 432
## -- Column specification
## -------------------------------------------------------- Delimiter: "," chr
## (1): Institution dbl (431): ...1, Failure.binary, Failure, Entropy_cooc.W.ADC,
## GLNU_align.H.P...
## 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.
## * `` -> `...1`
```

```
DF$Failure.binary=DF$Failure.binary%>%as.factor()
split = DF%>%initial_split( strata = "Failure.binary")
trn_df = training(split)
tst_df = testing(split)
```

Creating a blueprint of the model for the training and testing set converted into h2o objects since we will use h2o for modelling.

```
# Make sure we have consistent categorical levels
blueprint = recipe(Failure.binary ~ ., data = trn_df) %>%
  step_other(all_nominal(), threshold = 0.005)

# Create training & test sets for h2o
trn_h2o = prep(blueprint, training = trn_df, retain = TRUE) %>%
  juice() %>%
  as.h2o()
```

```
##   |                                                                      |
```

```
tst_h2o = prep(blueprint, training = trn_df) %>%
  bake(new_data = tst_df) %>%
  as.h2o()
```

```
##   |                                                                      |
```

Extracting response and feature names for easy access

```
# Get response and feature names
Y = "Failure.binary"

X = setdiff(names(trn_df), Y)
```

Training best candidate glm model for ensemble

```
best_glm = h2o.glm(
  x = X, y = Y, training_frame = trn_h2o, alpha = 0.1,
  remove_collinear_columns = TRUE, nfolds = 10, fold_assignment = "Modulo",stopping_metric = "logloss",
  keep_cross_validation_predictions = TRUE, seed = 123
)
```

```
##   |                                                                      |
```

Training best candidate rf model for ensemble

```r
best_rf = h2o.randomForest(
  x = X, y = Y, training_frame = trn_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
)
```

```
##   |                                                                      |
```

Training best candidate glm model for ensemble

```r
# Train & cross-validate a GBM model
best_gbm = h2o.gbm(
  x = X, y = Y, training_frame = trn_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
)
```

```
##   |                                                                      |
```

Getting logloss of each candidate model using custom function, and it seems glm had the largest logloss while gbm had the smallest. this makes gbm a good candidate for stacked ensemble

```r
get_logloss = function(model) {
  results = h2o.performance(model, newdata = tst_h2o)
  results@metrics$logloss
}
list(best_glm, best_rf, best_gbm) %>%
  purrr::map_dbl(get_logloss)
```

```
## [1] 0.6410355 0.4439697 0.3944326
```

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

Defining a hyper parameter tuning gread and the search criteria for the stacked ensemble algorithm

```r
# 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
)
```

creaing the grid in h2o

```r
# Build random grid search
random_grid = h2o.grid(
```

```r
  algorithm = "gbm", grid_id = "gbm_grid", x = X, y = Y,
  training_frame = trn_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
)
```

```
##   |                                                                      |
```

now creating the stacked ensemble with the gbm as base model

```r
ensemble_tree = h2o.stackedEnsemble(
  x = X, y = Y, training_frame = trn_h2o, model_id = "ensemble_gbm_grid",
  base_models = random_grid@model_ids, metalearner_algorithm = "gbm",
)
```

```
##   |                                                                      |
```

```r
# Stacked results
h2o.performance(ensemble_tree, newdata = tst_h2o)@metrics$logloss
```

```
## [1] 0.3255857
```

```r
## [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 1.00000000 0.04449903 0.00551532
## RF_pred  0.04449903 1.00000000 0.77011314
## GBM_pred 0.00551532 0.77011314 1.00000000
```

We will now sort the stacking result by their corresponding logloss.

```r
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         0.90000    0.10000              1.00000   5.00000  1.00000
## 2         0.90000    0.10000              0.99000   3.00000  5.00000
## 3         0.80000    0.10000              1.00000   3.00000 10.00000
## 4         0.80000    0.10000              0.99000   5.00000 10.00000
## 5         1.00000    0.05000              1.00000   5.00000  1.00000
##   sample_rate         model_ids logloss
## 1     0.75000 gbm_grid_model_16 0.30529
## 2     0.50000  gbm_grid_model_7 0.33244
## 3     0.50000 gbm_grid_model_13 0.33501
## 4     0.50000 gbm_grid_model_19 0.33587
## 5     0.75000  gbm_grid_model_8 0.34808
##
## ---
##    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   5.00000  5.00000
## 22         0.80000    0.01000              1.00000   3.00000  5.00000
## 23         0.90000    0.01000              0.99000   3.00000 10.00000
## 24         0.90000    0.01000              1.00000   1.00000  5.00000
## 25         0.90000    0.01000              0.99000   1.00000  1.00000
##    sample_rate         model_ids logloss
## 20     1.00000 gbm_grid_model_23 0.55116
## 21     0.50000 gbm_grid_model_17 0.55121
## 22     0.50000  gbm_grid_model_6 0.55349
## 23     0.50000 gbm_grid_model_11 0.55889
## 24     0.50000 gbm_grid_model_18 0.56170
## 25     0.50000 gbm_grid_model_21 0.56695
```

Retreiving the sorted grid.

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

Retrieving the best model from the grid and checking its perfomrmance on testing data.

```r
# 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 = tst_h2o)
```

```
## H2OBinomialMetrics: gbm
##
## MSE:  0.1363061
## RMSE:  0.3691966
## LogLoss:  0.4227819
## Mean Per-Class Error:  0.1497326
## AUC:  0.868984
## AUCPR:  0.7944527
## Gini:  0.7379679
## R^2:  0.3925753
##
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
```

```
##         0  1    Error   Rate
## 0      27  6 0.181818  =6/33
## 1       2 15 0.117647  =2/17
## Totals 29 21 0.160000  =8/50
##
## Maximum Metrics: Maximum metrics at their respective thresholds
##                         metric threshold     value idx
## 1                       max f1  0.242441  0.789474  19
## 2                       max f2  0.152062  0.869565  22
## 3                 max f0point5  0.837327  0.789474   8
## 4                 max accuracy  0.449568  0.840000  17
## 5                max precision  0.925744  1.000000   0
## 6                   max recall  0.044943  1.000000  41
## 7              max specificity  0.925744  1.000000   0
## 8             max absolute_mcc  0.242441  0.672361  19
## 9   max min_per_class_accuracy  0.449568  0.823529  17
## 10 max mean_per_class_accuracy  0.242441  0.850267  19
## 11                     max tns  0.925744 33.000000   0
## 12                     max fns  0.925744 16.000000   0
## 13                     max fps  0.043734 33.000000  42
## 14                     max tps  0.044943 17.000000  41
## 15                     max tnr  0.925744  1.000000   0
## 16                     max fnr  0.925744  0.941176   0
## 17                     max fpr  0.043734  1.000000  42
## 18                     max tpr  0.044943  1.000000  41
##
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/
```

Best model is finally stacked.

```
# Train a stacked ensemble using the GBM grid
ensemble = h2o.stackedEnsemble(
  x = X, y = Y, training_frame = trn_h2o, model_id = "ensemble_gbm_grid",
  base_models = random_grid@model_ids, metalearner_algorithm = "gbm"
)
```

```
##   |                                                                              |
```

And its prediction performance on test data is:

```
# Eval ensemble performance on a test set
h2o.performance(ensemble, newdata = tst_h2o)
```

```
## H2OBinomialMetrics: stackedensemble
##
## MSE:  0.09124263
## RMSE:  0.302064
## LogLoss:  0.3255857
## Mean Per-Class Error:  0.08823529
## AUC:  0.9286988
## AUCPR:  0.9329073
## Gini:  0.8573975
##
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
##         0  1    Error   Rate
## 0      33  0 0.000000  =0/33
## 1       3 14 0.176471  =3/17
```

```
## Totals 36 14 0.060000  =3/50
##
## Maximum Metrics: Maximum metrics at their respective thresholds
##                         metric threshold     value idx
## 1                       max f1  0.841542  0.903226  13
## 2                       max f2  0.063842  0.888889  21
## 3                 max f0point5  0.841542  0.958904  13
## 4                 max accuracy  0.841542  0.940000  13
## 5                max precision  0.994287  1.000000   0
## 6                   max recall  0.003365  1.000000  44
## 7              max specificity  0.994287  1.000000   0
## 8             max absolute_mcc  0.841542  0.868851  13
## 9   max min_per_class_accuracy  0.747444  0.878788  18
## 10 max mean_per_class_accuracy  0.841542  0.911765  13
## 11                     max tns  0.994287 33.000000   0
## 12                     max fns  0.994287 16.000000   0
## 13                     max fps  0.003225 33.000000  47
## 14                     max tps  0.003365 17.000000  44
## 15                     max tnr  0.994287  1.000000   0
## 16                     max fnr  0.994287  0.941176   0
## 17                     max fpr  0.003225  1.000000  47
## 18                     max tpr  0.003365  1.000000  44
##
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/F
```

```
h2o.performance(ensemble)
```

```
## H2OBinomialMetrics: stackedensemble
## ** Reported on training data. **
##
## MSE:  0.02100334
## RMSE:  0.1449253
## LogLoss:  0.0757774
## Mean Per-Class Error:  0.01546392
## AUC:  0.996701
## AUCPR:  0.9934776
## Gini:  0.9934021
##
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
##         0  1    Error    Rate
## 0      94  3 0.030928   =3/97
## 1       0 50 0.000000   =0/50
## Totals 94 53 0.020408  =3/147
##
## Maximum Metrics: Maximum metrics at their respective thresholds
##                         metric threshold     value idx
## 1                       max f1  0.358162  0.970874  44
## 2                       max f2  0.358162  0.988142  44
## 3                 max f0point5  0.964032  0.968468  34
## 4                 max accuracy  0.358162  0.979592  44
## 5                max precision  0.996176  1.000000   0
## 6                   max recall  0.358162  1.000000  44
## 7              max specificity  0.996176  1.000000   0
## 8             max absolute_mcc  0.358162  0.956148  44
## 9   max min_per_class_accuracy  0.528741  0.969072  43
```

```
## 10 max mean_per_class_accuracy  0.358162  0.984536  44
## 11                     max tns  0.996176 97.000000   0
## 12                     max fns  0.996176 49.000000   0
## 13                     max fps  0.002930 97.000000 121
## 14                     max tps  0.358162 50.000000  44
## 15                     max tnr  0.996176  1.000000   0
## 16                     max fnr  0.996176  0.980000   0
## 17                     max fpr  0.002930  1.000000 121
## 18                     max tpr  0.358162  1.000000  44
##
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/F
```

To further validate the model, we will do an AutoML and be able to see how the stacked ensemble performs well overall

```
# Use AutoML to find a list of candidate models (i.e., leaderboard)
auto_ml = h2o.automl(
  x = X, y = Y, training_frame = trn_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
)
```

```
##   |                                                                              |
## 23:43:55.799: Stopping tolerance set by the user is < 70% of the recommended default of 0.05, so mode
## 23:43:55.814: AutoML: XGBoost is not available; skipping it.  |
## 23:44:05.399: _min_rows param, The dataset size is too small to split for min_rows=100.0: must have a
```

The resulting list of best models are then sorted by logloss

```
# 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_1_20221216_234355 0.2777050
## 2  StackedEnsemble_BestOfFamily_1_AutoML_1_20221216_234355 0.2791236
## 3                             GBM_4_AutoML_1_20221216_234355 0.2953036
## 4                             GBM_3_AutoML_1_20221216_234355 0.3077520
## 5              GBM_grid_1_AutoML_1_20221216_234355_model_1 0.3105793
## 6                             GBM_2_AutoML_1_20221216_234355 0.3342698
## 7                             GLM_1_AutoML_1_20221216_234355 0.3436027
## 8                             GBM_5_AutoML_1_20221216_234355 0.3673880
## 9                             XRT_1_AutoML_1_20221216_234355 0.4280780
## 10                            DRF_1_AutoML_1_20221216_234355 0.4559723
## 11                   DeepLearning_1_AutoML_1_20221216_234355 0.5817020
## 12    DeepLearning_grid_1_AutoML_1_20221216_234355_model_1 0.8299525
```

Now testing the stacked best mode's prediction performane on training data which again shows strong accuracy.

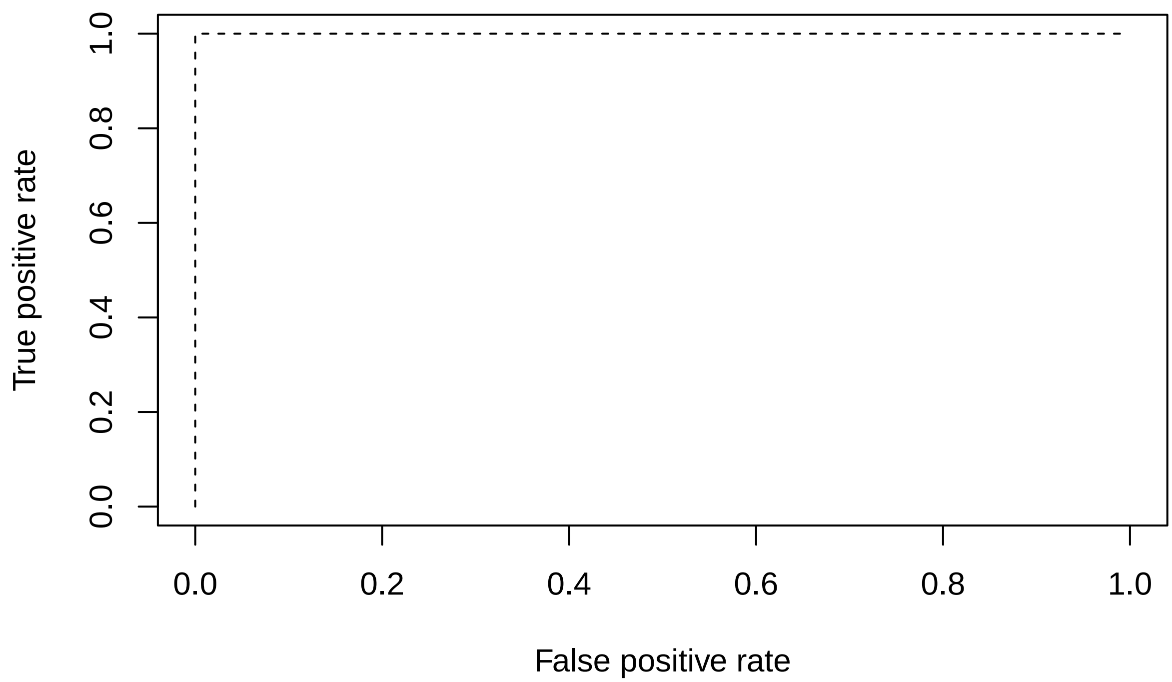
```
# Compute predicted probabilities on training data
trn_h2o=as.h2o(trn_df)
```

```
##   |                                                                              |
```

```r
m1_prob = predict(auto_ml@leader, trn_h2o, type = "prob")
```

```
##   |                                                                              |
```

```r
m1_prob=as.data.frame(m1_prob)[,1]%>%as.numeric()
trn_h2o=as.data.frame(trn_h2o)
# Compute AUC metrics for cv_model1,2 and 3
perf1 = prediction(m1_prob,trn_h2o$Failure.binary) %>%
  performance(measure = "tpr", x.measure = "fpr")
# Plot ROC curves for cv_model1,2 and 3
plot(perf1, col = "black", lty = 2)
```

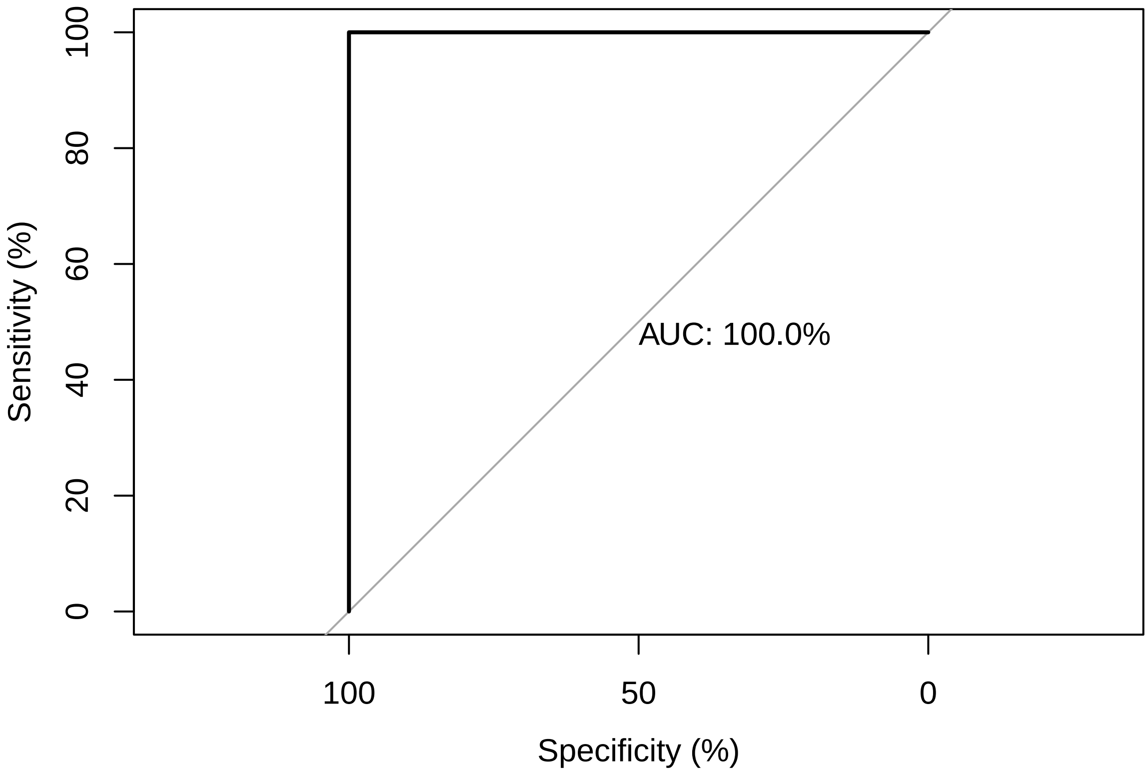


Strong accuracy is dpicted by the high AUC value.

```r
# ROC plot for training data
roc( trn_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 = trn_h2o$Failure.binary ~ m1_prob, plot = TRUE,     legacy.axes = FALSE, percent
##
## Data: m1_prob in 97 controls (trn_h2o$Failure.binary 0) < 50 cases (trn_h2o$Failure.binary 1).
## Area under the curve: 100%
```

The model performed well on the testing data as well so it is not considerably an overfitted model

```
# Compute predicted probabilities on training data
tst_h2o=as.h2o(tst_df)
```

```
##   |                                                                              |
```

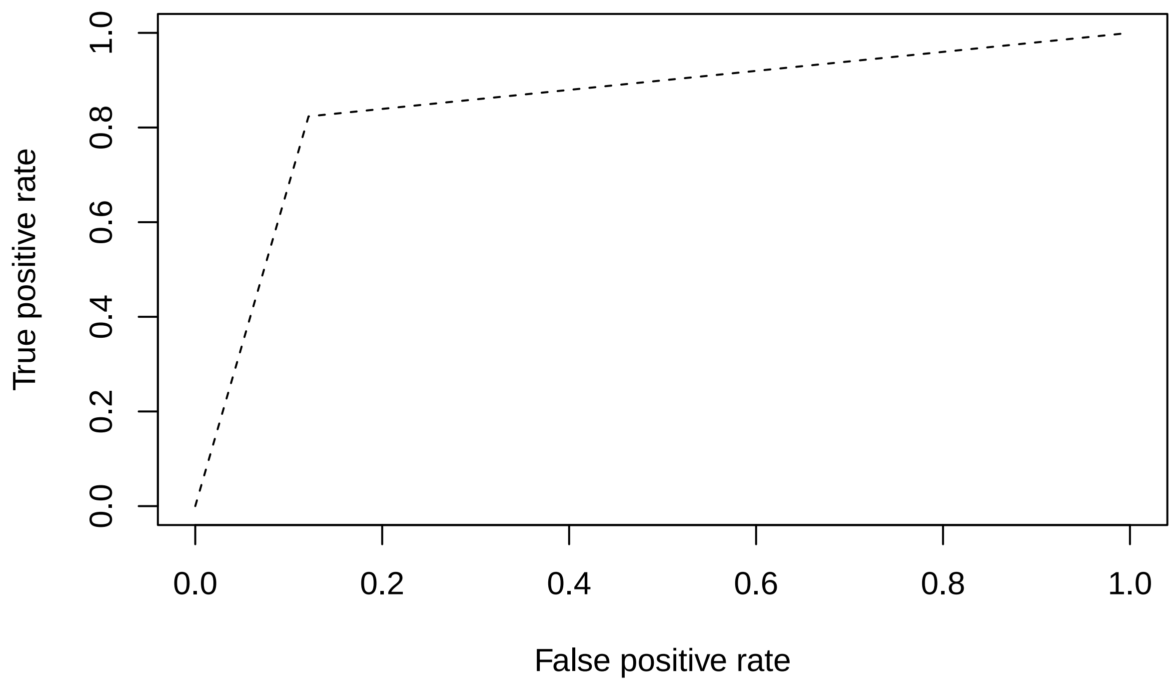
```
m2_prob = predict(auto_ml@leader, tst_h2o, type = "prob")
```

```
##   |                                                                              |
```

```
m2_prob=as.data.frame(m2_prob)[,1]%>%as.numeric()
tst_h2o=as.data.frame(tst_h2o)

# Compute AUC metrics for cv_model1,2 and 3
perf2 = prediction(m2_prob,tst_h2o$Failure.binary) %>%
  performance(measure = "tpr", x.measure = "fpr")


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

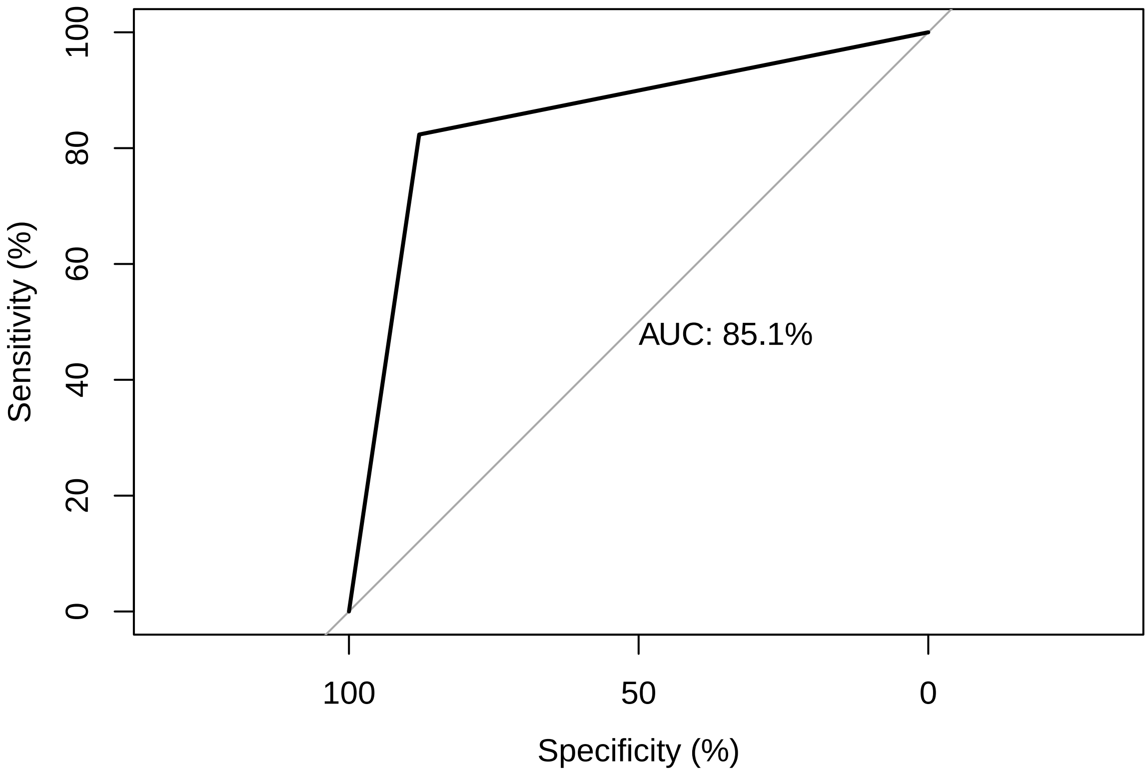


The AUC shown below is high enough to provide a good prediction.

```
# ROC plot for training data
roc( tst_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
```



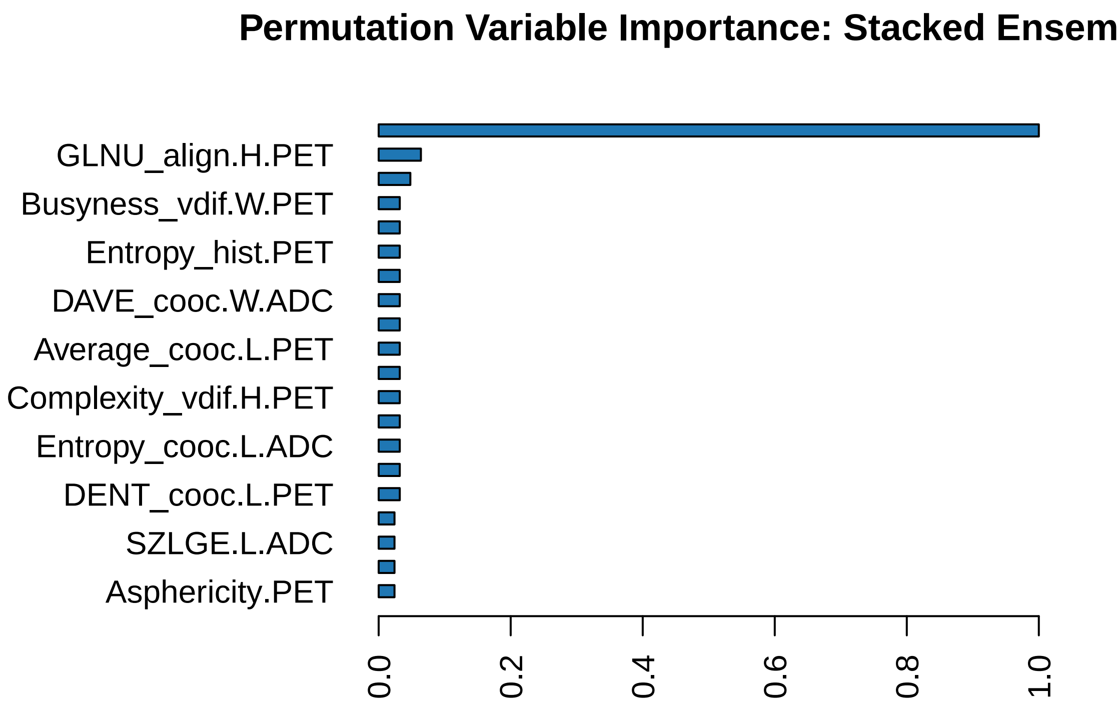

```
##
## Call:
## roc.formula(formula = tst_h2o$Failure.binary ~ m2_prob, plot = TRUE,     legacy.axes = FALSE, percen
##
## Data: m2_prob in 33 controls (tst_h2o$Failure.binary 0) < 17 cases (tst_h2o$Failure.binary 1).
## Area under the curve: 85.12%
```

The Entropy is again the ighest interms of the variable importance but this time interms of permutatino importance since vip does not exist for stacked models.

```
tst_h2o=as.h2o(tst_h2o)
```

```
##   |                                                                             |
```

```
h2o.permutation_importance_plot(auto_ml@leader,tst_h2o,num_of_features = 20)
```


Permutation Variable Importance: Stacked Ensem
GLNU_align.H.PET
Busyness_vdif.W.PET
Entropy_hist.PET
DAVE_cooc.W.ADC
Average_cooc.L.PET
Complexity_vdif.H.PET
Entropy_cooc.L.ADC
DENT_cooc.L.PET
SZLGE.L.ADC
Asphericity.PET
0.0
0.2
0.4
0.6
0.8
1.0