CKME136_Stacked_Ensembles

Ebunoluwa Odeniyi

Load required packages

```
require(plyr)
```

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

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:plyr':
##
##     arrange, count, desc, failwith, id, mutate, rename, summarise,
##     summarize
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(corrplot)
```

```
## corrplot 0.84 loaded
```

Load data

```
accs <- read.csv("C:/Users/YENN/Desktop/UST/FARS2016N/accident2016.csv", header = T, s
tringsAsFactors = F)
```

```r
library(h2o)
```

```
##
## ----------------------------------------------------------------------
##
## 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 http://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
localH2O = h2o.init(ip = 'localhost', port = 54321, nthreads = -1,max_mem_size = "8G")
```

```
##
## H2O is not running yet, starting it now...
##
## Note:  In case of errors look at the following log files:
##     C:\Users\YENN\AppData\Local\Temp\RtmpYDc8yJ/h2o_YENN_started_from_r.out
##     C:\Users\YENN\AppData\Local\Temp\RtmpYDc8yJ/h2o_YENN_started_from_r.err
##
##
## Starting H2O JVM and connecting:  Connection successful!
##
## R is connected to the H2O cluster:
##     H2O cluster uptime:         5 seconds 783 milliseconds
##     H2O cluster timezone:       America/New_York
##     H2O data parsing timezone:  UTC
##     H2O cluster version:        3.20.0.2
##     H2O cluster version age:    26 days
##     H2O cluster name:           H2O_started_from_R_YENN_ahr209
##     H2O cluster total nodes:    1
##     H2O cluster total memory:   7.11 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
##     H2O API Extensions:         Algos, AutoML, Core V3, Core V4
##     R Version:                  R version 3.5.1 (2018-07-02)
```

convert data to H2OFrame

```
accs_h2o <- as.h2o(accs)
```

```
##
  |
  |                                                                 |   0%
  |
  |=================================================================| 100%
```

```
splits <- h2o.splitFrame(accs_h2o,
                         ratios = c(0.6, 0.2),
                         seed = 148)   #partition data into 60%, 20%, 20% chunks
train <- splits[[1]]
validation <- splits[[2]]
test <- splits[[3]]
y <- "y"
x <- setdiff(colnames(train), y)
```

```
library(h2o)
h2o.init()
```

```
##  Connection successful!
##
## R is connected to the H2O cluster:
##     H2O cluster uptime:         10 seconds 368 milliseconds
##     H2O cluster timezone:       America/New_York
##     H2O data parsing timezone:  UTC
##     H2O cluster version:        3.20.0.2
##     H2O cluster version age:    26 days
##     H2O cluster name:           H2O_started_from_R_YENN_ahr209
##     H2O cluster total nodes:    1
##     H2O cluster total memory:   7.09 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
##     H2O API Extensions:         Algos, AutoML, Core V3, Core V4
##     R Version:                  R version 3.5.1 (2018-07-02)
```

```
# # Import a sample binary outcome train/test set into H2O
train <- h2o.importFile("https://s3.amazonaws.com/erin-data/higgs/higgs_train_10k.cs
v")
```

```
##
  |
  |                                                                 |   0%
  |
  |=================================================================| 100%
```

```
test <- h2o.importFile("https://s3.amazonaws.com/erin-data/higgs/higgs_test_5k.csv")
```

```
##
  |
  |                                                                 |   0%
  |
  |=================================================================| 100%
```

```
#
# # Identify predictors and response
y <- "response"
x <- setdiff(names(train), y)
#
# # For binary classification, response should be a factor
train[,y] <- as.factor(train[,y])
test[,y] <- as.factor(test[,y])
# Number of CV folds (to generate level-one data for stacking)

nfolds <- 5

# There are a few ways to assemble a list of models to stack toegether:
# 1. Train individual models and put them in a list
# 2. Train a grid of models
# 3. Train several grids of models
# Note: All base models must have the same cross-validation folds and
# the cross-validated predicted values must be kept.
# 1. Generate a 2-model ensemble (GBM + RF)
# Train & Cross-validate a GBM

my_gbm <- h2o.gbm(x = x,
                  y = y,
                  training_frame = train,
                  distribution = "bernoulli",
                  ntrees = 10,
                  max_depth = 3,
                  min_rows = 2,
                  learn_rate = 0.2,
                  nfolds = nfolds,
                  fold_assignment = "Modulo",
                  keep_cross_validation_predictions = TRUE,
                  seed = 1)
```

```
##
  |
  |                                                                 |   0%
  |
  |===========================================                      |  67%
  |
  |=================================================================| 100%
```

```
# Train & Cross-validate a RF
my_rf <- h2o.randomForest(x = x,
                          y = y,
                          training_frame = train,
                          ntrees = 50,
                          nfolds = nfolds,
                          fold_assignment = "Modulo",
                          keep_cross_validation_predictions = TRUE,
                          seed = 1)
```

```
##
  |
  |                                                                 |   0%
  |
  |==                                                               |   4%
  |
  |======                                                           |   9%
  |
  |========                                                         |  12%
  |
  |============                                                     |  18%
  |
  |================                                                 |  25%
  |
  |====================                                             |  31%
  |
  |========================                                         |  38%
  |
  |=============================                                    |  44%
  |
  |================================                                 |  50%
  |
  |======================================                           |  58%
  |
  |==========================================                       |  64%
  |
  |===============================================                  |  72%
  |
  |===================================================              |  78%
  |
  |=====================================================            |  82%
  |
  |========================================================         |  86%
  |
  |===============================================================  |  96%
  |
  |=================================================================| 100%
```

```
# Train a stacked ensemble using the GBM and RF above
ensemble <- h2o.stackedEnsemble(x = x,
                                y = y,
                                training_frame = train,
                                model_id = "my_ensemble_binomial",
                                base_models = list(my_gbm@model_id, my_rf@model_id))
```

```
##
  |
  |                                                                 |   0%
  |
  |=================================================================| 100%
```

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

# Compare to base learner performance on the test set
perf_gbm_test <- h2o.performance(my_gbm, newdata = test)
perf_rf_test <- h2o.performance(my_rf, newdata = test)
baselearner_best_auc_test <- max(h2o.auc(perf_gbm_test), h2o.auc(perf_rf_test))
ensemble_auc_test <- h2o.auc(perf)
print(sprintf("Best Base-learner Test AUC:  %s", baselearner_best_auc_test))
```

```
## [1] "Best Base-learner Test AUC:  0.769805454669772"
```

```
print(sprintf("Ensemble Test AUC:  %s", ensemble_auc_test))
```

```
## [1] "Ensemble Test AUC:  0.773516978976877"
```

```
# Generate predictions on a test set (if neccessary)
pred <- h2o.predict(ensemble, newdata = test)
```

```
##
  |
  |                                                                 |   0%
  |
  |=================================================================| 100%
```

```
pred%>%head()
```

	predict <fctr>	p0 <dbl>	p1 <dbl>
1	0	0.6670590	0.3329410
2	1	0.5833897	0.4166103
3	1	0.6058676	0.3941324
4	1	0.1909705	0.8090295

	predict	p0	p1
	<fctr>	<dbl>	<dbl>
5	1	0.4533609	0.5466391
6	1	0.3145294	0.6854706

6 rows

0 is the probability (between 0 and 1) that class 0 is chosen.

1 is the probability (between 0 and 1) that class 1 is chosen.

The predict is made by applying a threshold to 0/1. That threshold point is chosen depending on whether you want to reduce false positives or false negatives. It's not just 0.5.

The threshold chosen for "the prediction" is max-F1. But you can extract out p1 yourself and threshold it any way you like.

Parameter Tuning

```
# 2. Generate a random grid of models and stack them together

# GBM Hyperparamters
learn_rate_opt <- c(0.01, 0.03)
max_depth_opt <- c(3, 4, 5, 6, 9)
sample_rate_opt <- c(0.7, 0.8, 0.9, 1.0)
col_sample_rate_opt <- c(0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8)
hyper_params <- list(learn_rate = learn_rate_opt,
                     max_depth = max_depth_opt,
                     sample_rate = sample_rate_opt,
                     col_sample_rate = col_sample_rate_opt)
search_criteria <- list(strategy = "RandomDiscrete",
                        max_models = 3,
                        seed = 1)
gbm_grid <- h2o.grid(algorithm = "gbm",
                     grid_id = "gbm_grid_binomial",
                     x = x,
                     y = y,
                     training_frame = train,
                     ntrees = 50,
                     seed = 1,
                     nfolds = nfolds,
                     fold_assignment = "Modulo",
                     keep_cross_validation_predictions = TRUE,
                     hyper_params = hyper_params,
                     search_criteria = search_criteria)
```

```
##
  |
  |                                                                 |   0%
  |
  |===                                                              |   5%
  |
  |========                                                         |  12%
  |
  |=============                                                    |  20%
  |
  |================                                                 |  25%
  |
  |===================                                              |  29%
  |
  |=======================                                          |  35%
  |
  |============================                                     |  43%
  |
  |=================================                                |  50%
  |
  |=====================================                            |  57%
  |
  |========================================                         |  61%
  |
  |==============================================                   |  71%
  |
  |===================================================              |  79%
  |
  |=========================================================        |  88%
  |
  |=============================================================    |  94%
  |
  |=================================================================| 100%
```

```
# Train a stacked ensemble using the GBM grid
ensemble2 <- h2o.stackedEnsemble(x = x,
                                y = y,
                                training_frame = train,
                                model_id = "ensemble_gbm_grid_binomial",
                                base_models = gbm_grid@model_ids)
```

```
##
  |
  |                                                                 |   0%
  |
  |=================================================================| 100%
```

```r
# storing and loading the model
path <- h2o.saveModel(ensemble2, path = "ensemble2", force = TRUE)
print(path)
```

```
## [1] "C:\\Users\\YENN\\Desktop\\UST\\ensemble2\\ensemble_gbm_grid_binomial"
```

```r
loaded <- h2o.loadModel(path)
```

```r
# Eval ensemble performance on a test set
perf <- h2o.performance(ensemble2, newdata = test)

# Compare to base learner performance on the test set
.getauc <- function(mm) h2o.auc(h2o.performance(h2o.getModel(mm), newdata = test))
baselearner_aucs <- sapply(gbm_grid@model_ids, .getauc)
baselearner_best_auc_test <- max(baselearner_aucs)
ensemble_auc_test <- h2o.auc(perf)
print(sprintf("Best Base-learner Test AUC:  %s", baselearner_best_auc_test))
```

```
## [1] "Best Base-learner Test AUC:  0.756244072541236"
```

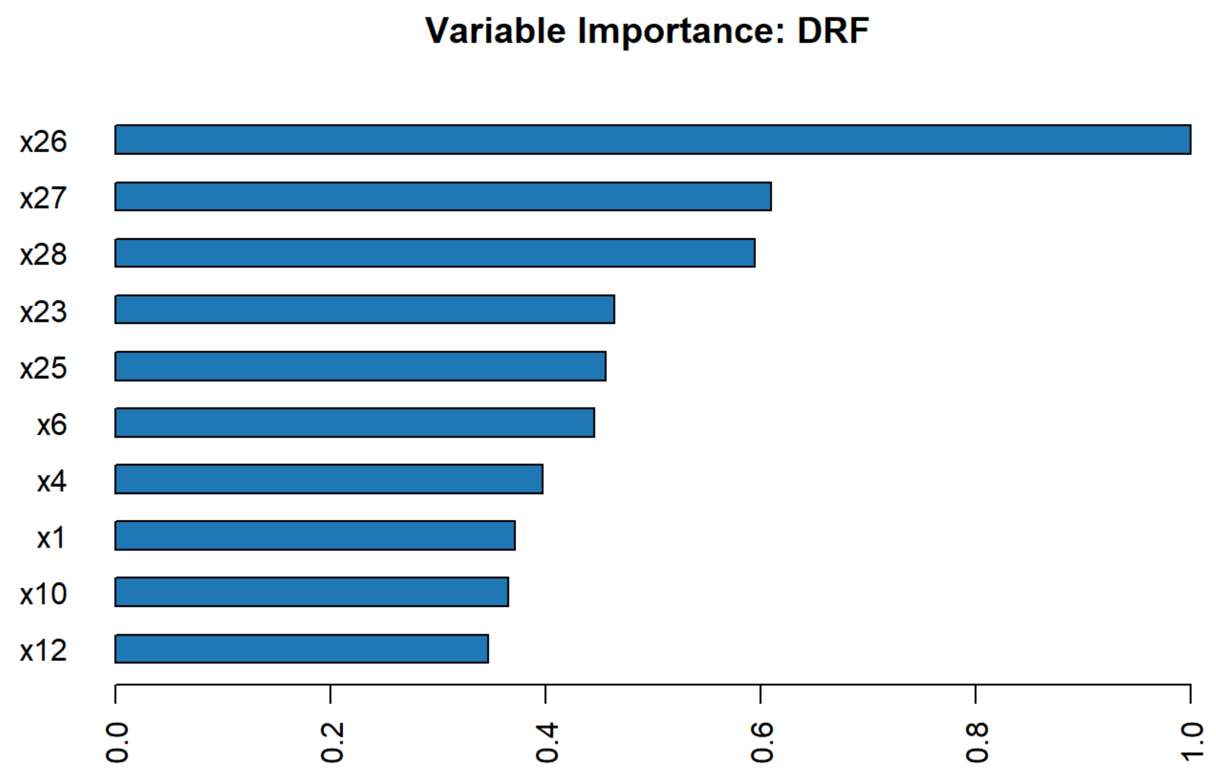
```r
print(sprintf("Ensemble Test AUC:  %s", ensemble_auc_test))
```

```
## [1] "Ensemble Test AUC:  0.759037851273574"
```

```r
# Generate predictions on a test set (if neccessary)
pred <- h2o.predict(ensemble, newdata = test)
```

```
## 
  |                                                                 
  |                                                                 |   0%
  |                                                                 
  |=================================================================| 100%
```

```r
h2o.varimp_plot(my_rf)
```

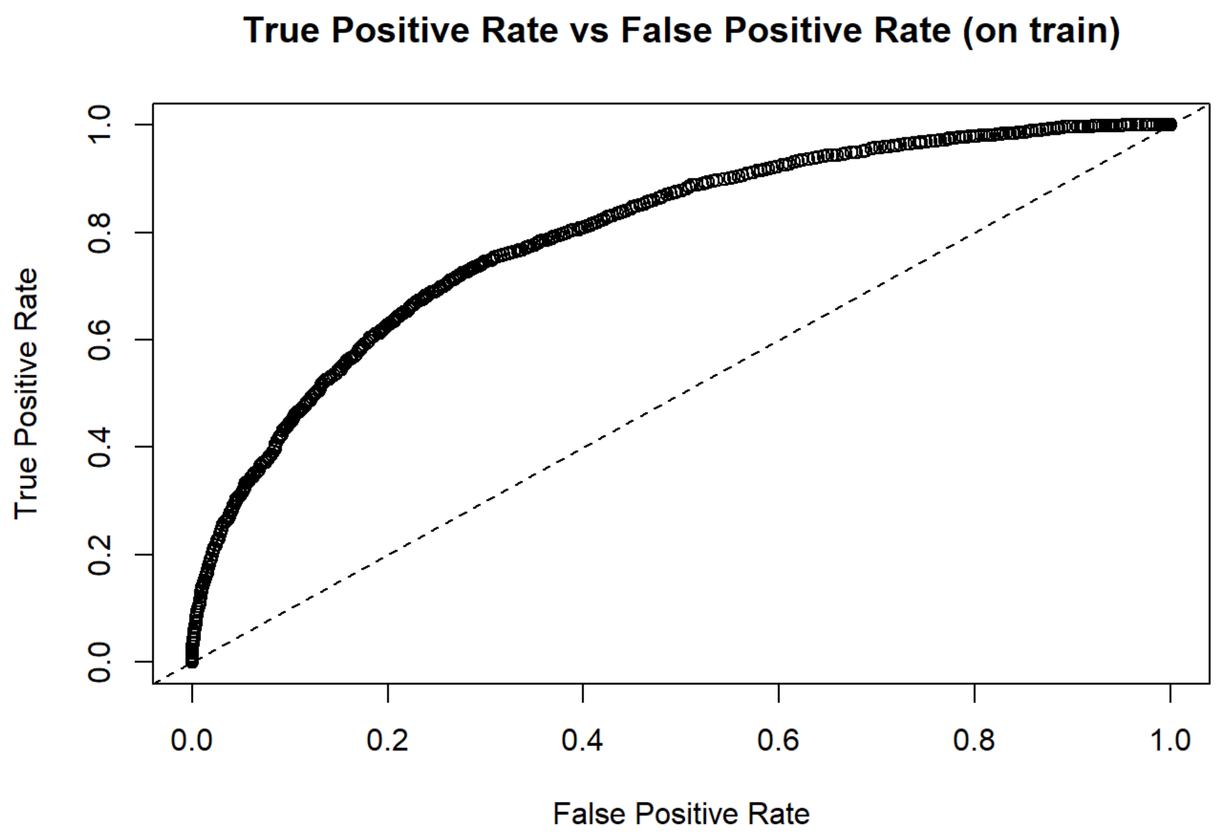


```
h2o.varimp(my_rf)%>%as_tibble()
```

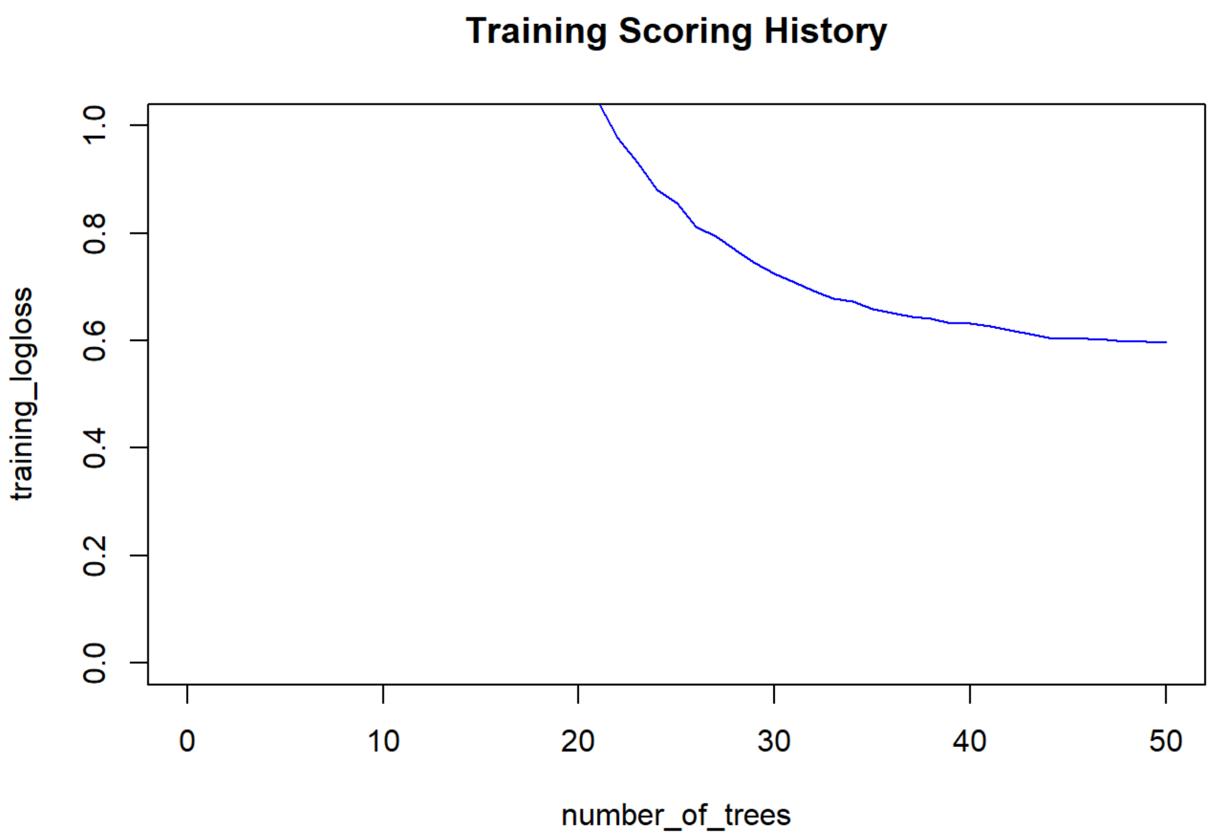
variable <chr>	relative_importance <dbl>	scaled_importance <dbl>	percentage <dbl>
x26	7796.9424	1.00000000	0.101277344
x27	4754.2627	0.60975989	0.061754861
x28	4638.1064	0.59486222	0.060246065
x23	3618.0005	0.46402812	0.046995535
x25	3555.4287	0.45600295	0.046182767
x6	3471.5066	0.44523948	0.045092672
x4	3101.8667	0.39783117	0.040291284
x1	2901.7019	0.37215895	0.037691270
x10	2851.9561	0.36577878	0.037045103
x12	2708.3020	0.34735437	0.035179128

1-10 of 28 rows Previous **1** 2 3 Next

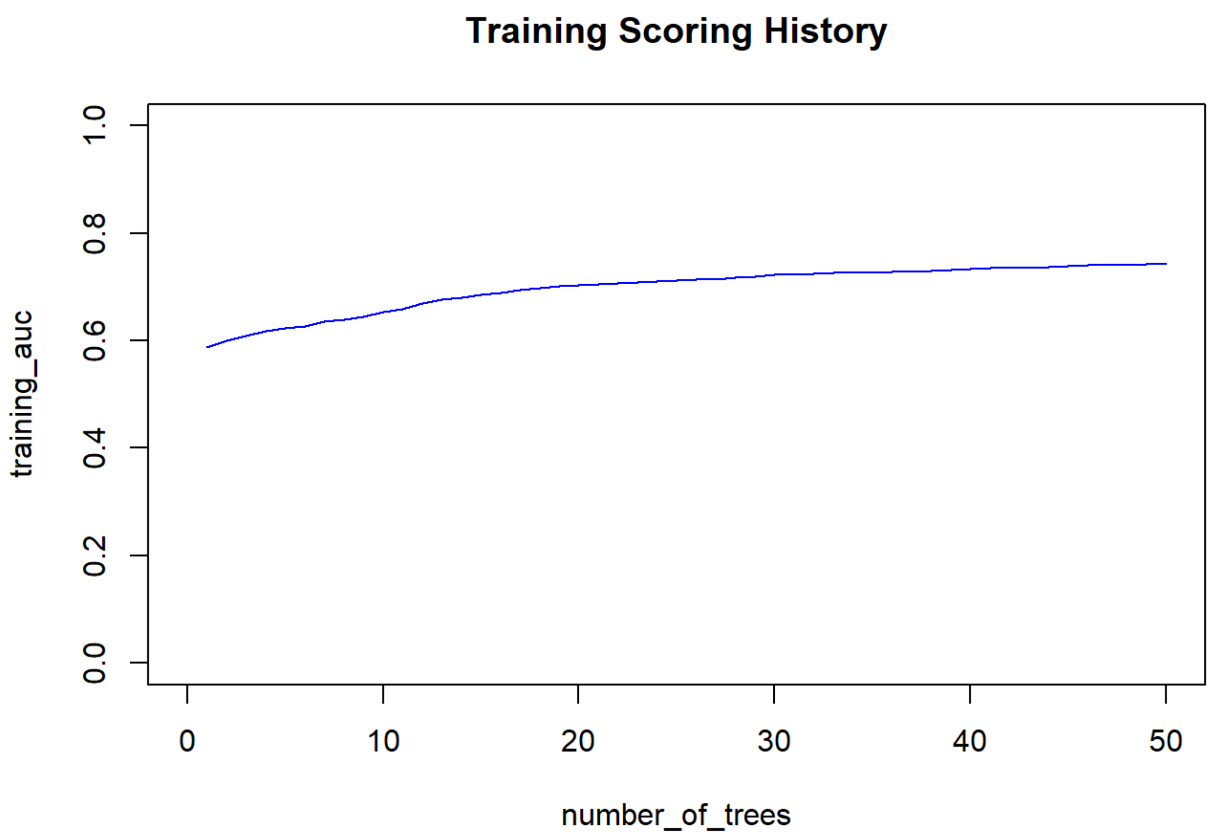
```
# for deep learning set the variable_importance parameter to TRUE eg.
#iris.hex <- as.h2o(iris)
#iris.dl <- h2o.deeplearning(x = 1:4, y = 5, training_frame = iris.hex,
#variable_importances = TRUE)
#h2o.varimp_plot(iris.dl)

plot(h2o.performance(ensemble2)) ## display ROC curve
```



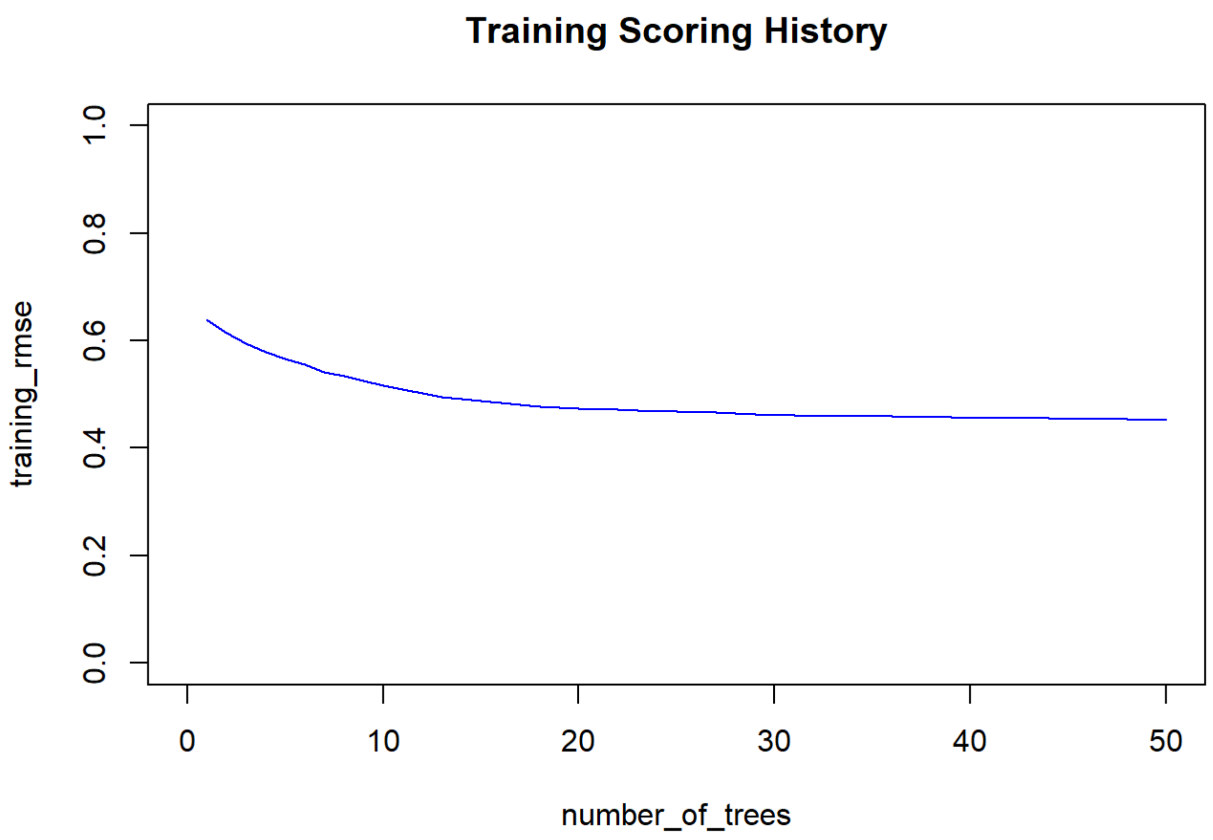
```
plot(my_rf)
```



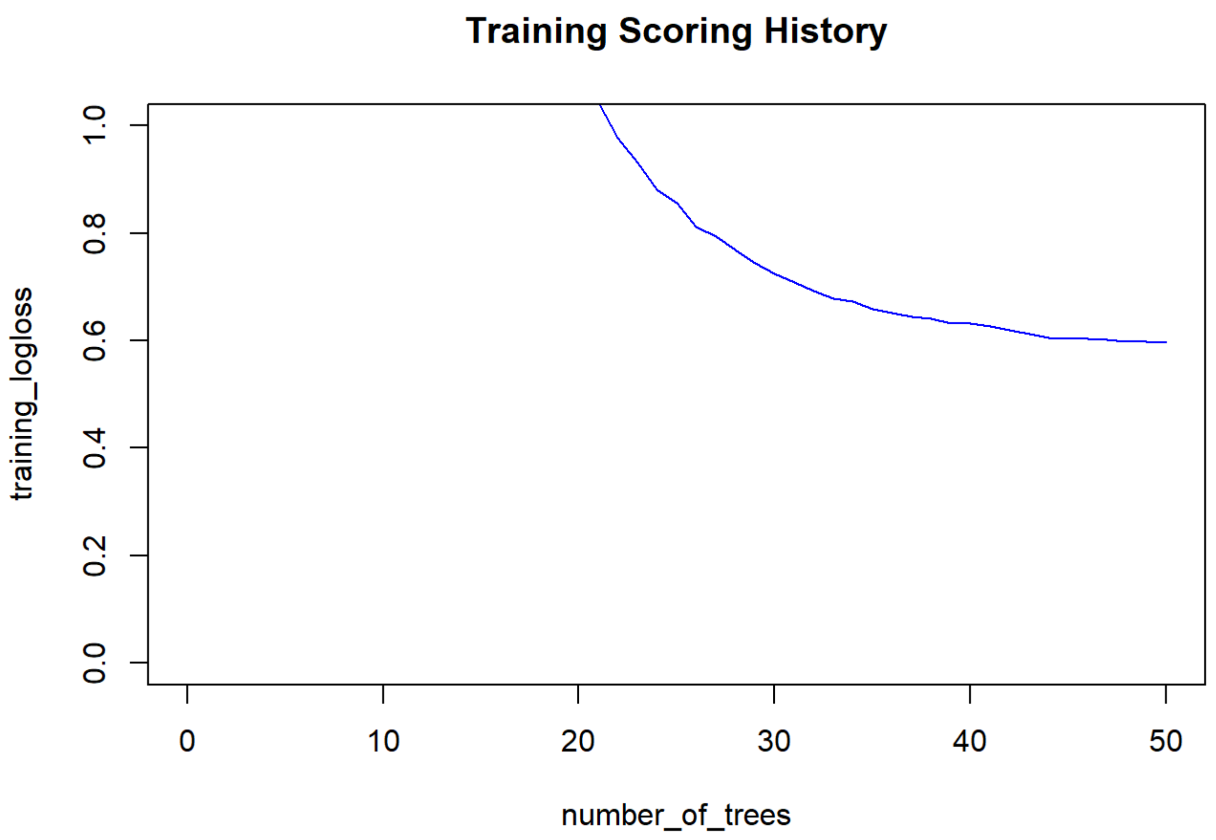
```
#plot(my_rf, timestep = "duration", metric = "deviance")
plot(my_rf, timestep = "number_of_trees", metric = "auc")
```



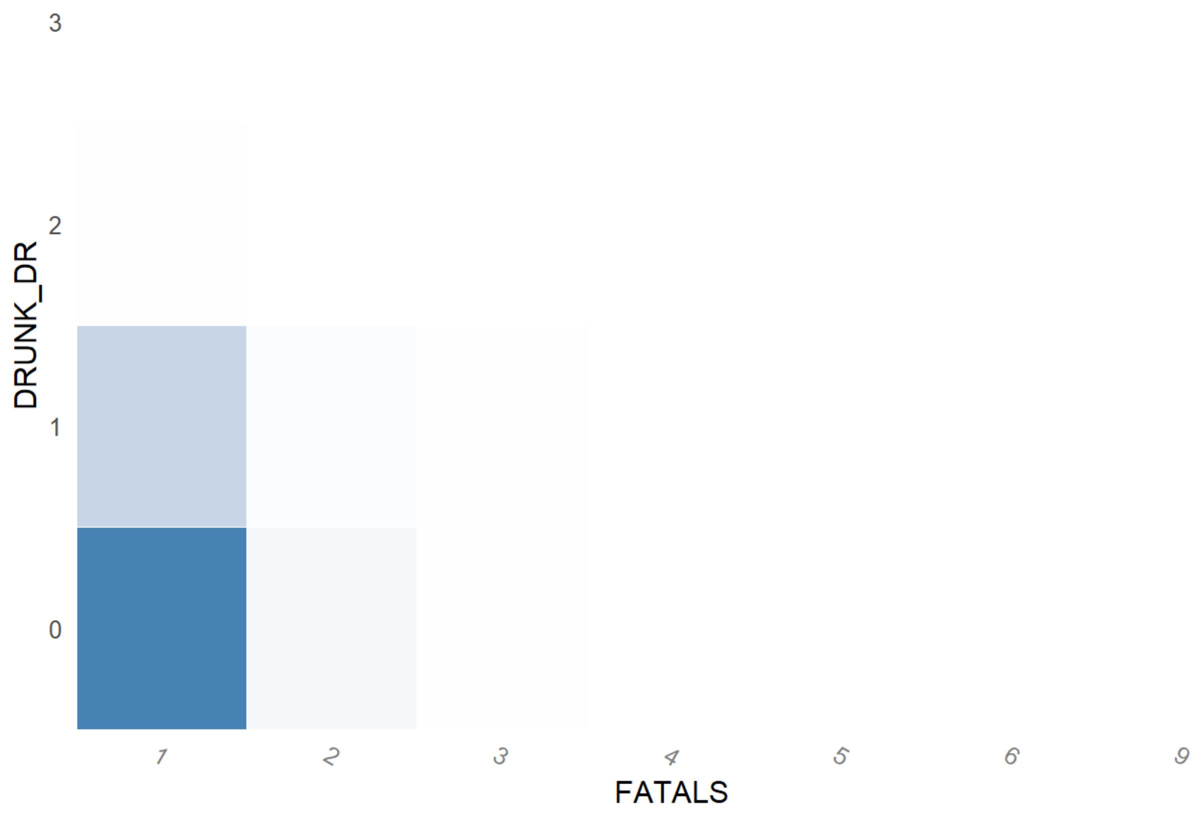
```
plot(my_rf, timestep = "number_of_trees", metric = "rmse")
```



```
plot(my_rf, timestep = "number_of_trees", metric = "logloss")
```



```
tab <- h2o.tabulate(data = accs_h2o, x = "FATALS", y = "DRUNK_DR",
weights_column = NULL, nbins_x = 10, nbins_y = 10)
plot(tab)
```



```
# storing and loading the model
# path <- h2o.saveModel(model, path = "mybest_deeplearning_covtype_model", force = TRU
E)
# print(path)
# loaded <- h2o.loadModel(path)
h2o.shutdown(prompt = FALSE)
```

```
## [1] TRUE
```