APAN5420 — HW 9, Credit Card Transactions

Megan Wilder

7/28/18

Contents

---

1 Down-Sampling for the Majority Class

Fraud transactions (the positive class) represent 0.173% of the data set, resulting in a highly imbalanced data set. Were I to run my model on the data set as is, it would bias the prediction model towards the more common non-fraudulent class. It is therefore necessary to balance the data set. I choose to use down-sampling, which creates a more balanced data set by selecting a random sample from the majority class. After down-sampling, fraud transactions represent 10% of the training data set. (source: http://www.simafore.com/blog/handling-unbalanced-data-machine-learning-models).

```
#split dataset into training and test sets
#set seed
set.seed(123)

# Sample into 3 sets. 60% train, 20% validation and 20% test
idx <-
sample(
seq(1, 3),
size = nrow(ccard),
replace = TRUE,
prob = c(.6, .2, .2)
)
train <- ccard[idx == 1, ]
test <- ccard[idx == 2, ]
val <- ccard[idx == 3, ]

#check classes distribution
kable(prop.table(table(train$Class)))
```

Var1	Freq
0	0.9982625
1	0.0017375

```
kable(prop.table(table(test$Class)))
```

Var1	Freq
0	0.9983397
1	0.0016603

```
kable(prop.table(table(val$Class)))
```

Var1	Freq
0	0.998235
1	0.001765

```
#down-sampling, sample so that fraud represents about 10% of data set
#a typical range for resampling is to make fraud 5-20% of the training set.
#want to make it a significant amount of the training set but not
#amplyify the noise too much.
data_balanced_under <-
ovun.sample(
Class ~ .,
data = train,
method = "under",
p = 0.1,
seed = 1
)$data

#view table of class variable in rebalanced training set
kable(table(data_balanced_under$Class))
```

Var1	Freq
0	2661
1	297

```
#view classes distribution in rebalanced training set
kable(prop.table(table(data_balanced_under$Class)))
```

Var1	Freq
0	0.8995943
1	0.1004057

```
#Start H2O
h2o.init(nthreads = -1, max_mem_size = '8G')

# clean slate in case the cluster was already running
h2o.removeAll()
```

2 Random Forest

Random Forest technique: The random forest classifier is a supervised learning technique. The model creates a set of decision trees from randomly selected subsets of the training set, it then aggregates the votes from different decision trees to decide the final class of the test object (source: https://medium.com/machine-learning-101/chapter-5-random-forest-classifier-56dc7425c3e1).

```
# make h2o data.frame, loads into H2O service
train.hex <- as.h2o(data_balanced_under)
```

```
##
  |
  |                                                                      |   0%
  |
```

```
  |======================================================================| 100%
```

```
test.hex <- as.h2o(test)
```

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

```
val.hex <- as.h2o(val)
```

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

```
# Summary
#summary(train.hex, exact_quantiles = TRUE)
#summary(test.hex, exact_quantiles = TRUE)
#summary(val.hex, exact_quantiles = TRUE)

# Response and predictors to use
resp <- "Class"
pred <- setdiff(names(train.hex), 'Class')

# train model
rf.1 = h2o.randomForest(
x = pred,
y = resp,
training_frame = train.hex,
validation_frame = val.hex,
model_id = "rf.1",
ntrees = 200,
## use a maximum of 200 trees to create the
##  random forest model. Will let
##  the early stopping criteria decide when
##  the random forest is sufficiently accurate
max_depth = 30,
stopping_rounds = 2,
## Stop fitting new trees when the 2-tree
##  average is within 0.001 (default) of
##  the prior two 2-tree averages.
##  Can be thought of as a convergence setting
stopping_tolerance = 1e-2,
score_each_iteration = T,
seed = 123 ## Set the random seed so that this can be reproduced
)
```

```
##
  |
  |                                                                      |   0%
  |
  |==                                                                    |   3%
  |
  |=====                                                                 |   7%
```

```
  |
  |=================================================================| 100%
```

```r
## Get the AUC on the test set
h2o.auc(h2o.performance(rf.1, newdata = test.hex)) #0.9435885 AUC
```

```
## [1] 0.9435885
```

```r
# predict response variable
rf.pred = h2o.predict(object = rf.1, newdata = test.hex)
```

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

3 Tune Hyperparameters

Hyperparameters:
ntrees = Number of trees. I used 200, 500, 1000, 1500, 2000.
max_depth = Maximum tree depth. I used 5, 10, 15, 20, 25, 30.

Grid Search: I used H2o's grid search to train and validate numerous models at once based on different hyper-parameter levels.

Performance Metrics: In order to evaluate the performance of a model on a given data set, it is necessary to measure how well the model's predictions actually match the observed data.

MSE = Mean squared error, it measures the square of the errors. The MSE will be small if the predicted responses are very close to the true responses, and it will be large if the predicted and true responses differ substantially. MSE is vulnerable to outliers and is in a different scale than the measured units. Used in regression (continuous output).

RMSE = Root mean squared error. It is the square root of the average of squared differences between prediction and actual observation (MSE). Lower values are better. It is scale dependent, therefore if the scales of the dependent variables differ across models, you can't compare RMSEs. Used in regression (continuous output).

Log Loss = Logarithmic loss measures the performance of a classification model where the prediction input is a probability value between 0 and 1. Lower values are better. "Log Loss takes into account the uncertainty of your prediction based on how much it varies from the actual label. This gives us a more nuanced view into the performance of our model."

AUC = The overall performance of a classifier, summarized over all possible thresholds, is given by the area under the (ROC) curve (AUC). It is used in classification analysis to determine which model predicted the classes best. It is typically used with binary classification. Not very useful for imbalanced data as it doesn't place more emphasis on one class over the other (i.e. it does not reflect the minority class well).

Gini = The Gini coefficient can be used to evaluate the performance of a classifier. It is the ratio between area between the ROC curve and the diagonal line and the area of the above triangle (Gini = 2*AUC – 1). Gini above 60% is viewed as a good model.

Precision = Measures that fraction of examples classified as positive that are truly positive (i.e. when the model predicts positive, how often is it correct?)

Recall = True positive rate (i.e. when it's actually positive, how often does it predict positive?).

F1 = Measure of a model's accuracy. It's the harmonic average of precision and recall, where an F1 score reaches its best value at 1 (perfect precision and recall) and worst at 0.

(sources: https://cran.r-project.org/web/packages/zFactor/vignettes/statistics.html http://wiki.fast.ai/index.php/Log_Loss
https://stats.stackexchange.com/questions/132777/what-does-auc-stand-for-and-what-is-it
https://www.analyticsvidhya.com/blog/2016/02/7-important-model-evaluation-error-metrics/
https://en.wikipedia.org/wiki/F1_score https://www.biostat.wisc.edu/~page/rocpr.pdf)

```r
#create list of hyperparameters to tune
hyper_params = list(ntrees = seq(500, 2000, 500),
max_depth = seq(5, 30, 5))

#Cartesian Grid Search
grid <- h2o.grid(
hyper_params = hyper_params,
search_criteria = list(strategy = "Cartesian"),
algorithm = "randomForest",
grid_id = "rf_grid",
x = pred,
y = resp,
training_frame = train.hex,
validation_frame = val.hex,
seed = 123,
stopping_rounds = 2,
stopping_tolerance = 1e-2,
stopping_metric = "AUC",
score_tree_interval = 10
)
```

```
##
  |
  |                                                                 |   0%
  |
  |=                                                                |   2%
  |
  |==                                                               |   3%
  |
  |==                                                               |   4%
  |
  |===                                                              |   5%
  |
  |====                                                             |   7%
  |
  |=====                                                            |   8%
  |
  |======                                                           |   9%
  |
  |=======                                                          |  10%
  |
  |=========                                                        |  13%
  |
  |=========                                                        |  14%
  |
  |===========                                                      |  17%
```

```
  |
  |=============                                                    |  20%
  |
  |===============                                                  |  24%
  |
  |=================                                                |  27%
  |
  |====================                                             |  30%
  |
  |=======================                                          |  35%
  |
  |==========================                                       |  40%
  |
  |=============================                                    |  45%
  |
  |=================================                                |  50%
  |
  |====================================                             |  55%
  |
  |=======================================                          |  60%
  |
  |===========================================                      |  67%
  |
  |================================================                 |  73%
  |
  |================================================                 |  74%
  |
  |====================================================             |  80%
  |
  |========================================================         |  87%
  |
  |=============================================================    |  94%
  |
  |=================================================================| 100%
```

```r
#view grid
grid
```

```
## H2O Grid Details
## ================
##
## Grid ID: rf_grid
## Used hyper parameters:
##   -  max_depth
##   -  ntrees
## Number of models: 24
## Number of failed models: 0
##
## Hyper-Parameter Search Summary: ordered by increasing logloss
##   max_depth ntrees        model_ids              logloss
## 1        10    500  rf_grid_model_1  0.02217957017223756
## 2        10   1000  rf_grid_model_7  0.02217957017223756
## 3        10   2000 rf_grid_model_19  0.02217957017223756
## 4        10   1500 rf_grid_model_13  0.02217957017223756
## 5         5   2000 rf_grid_model_18 0.022899041231277683
```

```
##
## ---
##    max_depth ntrees        model_ids              logloss
## 19        30   1000 rf_grid_model_11  0.02463165239446585
## 20        25   1000 rf_grid_model_10  0.02463165239446585
## 21        20   2000 rf_grid_model_21 0.024659055392621115
## 22        20   1000  rf_grid_model_9 0.024659055392621115
## 23        20    500  rf_grid_model_3 0.024659055392621115
## 24        20   1500 rf_grid_model_15 0.024659055392621115
```

```
## sort the grid models by decreasing AUC
sortedGrid <-
h2o.getGrid("rf_grid", sort_by = "auc", decreasing = TRUE)
print(sortedGrid)
```

```
## H2O Grid Details
## ================
##
## Grid ID: rf_grid
## Used hyper parameters:
##   -  max_depth
##   -  ntrees
## Number of models: 24
## Number of failed models: 0
##
## Hyper-Parameter Search Summary: ordered by decreasing auc
##   max_depth ntrees        model_ids                auc
## 1        15   1500 rf_grid_model_14 0.9911798716339268
## 2        15   1000  rf_grid_model_8 0.9911798716339268
## 3        15   2000 rf_grid_model_20 0.9911798716339268
## 4        15    500  rf_grid_model_2 0.9911798716339268
## 5        25    500  rf_grid_model_4 0.9894407411991443
##
## ---
##    max_depth ntrees        model_ids                auc
## 19        10   2000 rf_grid_model_19 0.9858226214261718
## 20        10   1500 rf_grid_model_13 0.9858226214261718
## 21         5   2000 rf_grid_model_18 0.9848034726028608
## 22         5   1000  rf_grid_model_6 0.9848034726028608
## 23         5    500  rf_grid_model_0 0.9848034726028608
## 24         5   1500 rf_grid_model_12 0.9848034726028608
```

```
#print AUC for 10 best models
for (i in 1:10) {
topModels <- h2o.getModel(sortedGrid@model_ids[[i]])
print(h2o.auc(h2o.performance(topModels, valid = TRUE)))
}
```

```
## [1] 0.9911799
## [1] 0.9911799
## [1] 0.9911799
## [1] 0.9911799
## [1] 0.9894407
## [1] 0.9894407
## [1] 0.9894407
## [1] 0.9894407
```

```
## [1] 0.9894407
## [1] 0.9894407
```

```
#name model with highest AUC best model
best_model <-
h2o.getModel(sortedGrid@model_ids[[1]]) #better than my original model, which had an AUC of 0.94358853

#view best model
summary(best_model)
```

```
## Model Details:
## ==============
##
## H2OBinomialModel: drf
## Model Key:  rf_grid_model_14
## Model Summary:
##   number_of_trees number_of_internal_trees model_size_in_bytes min_depth
## 1              50                       50               40243        12
##   max_depth mean_depth min_leaves max_leaves mean_leaves
## 1        15   14.84000         48         77    59.00000
##
## H2OBinomialMetrics: drf
## ** Reported on training data. **
## ** Metrics reported on Out-Of-Bag training samples **
##
## MSE:  0.01534102
## RMSE:  0.1238589
## LogLoss:  0.09097733
## Mean Per-Class Error:  0.07632127
## AUC:  0.9732082
## Gini:  0.9464164
##
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
##           0   1    Error      Rate
## 0      2658   3 0.001127   =3/2661
## 1        45 252 0.151515   =45/297
## Totals 2703 255 0.016227  =48/2958
##
## Maximum Metrics: Maximum metrics at their respective thresholds
##                         metric threshold    value idx
## 1                       max f1  0.631107 0.913043  46
## 2                       max f2  0.318794 0.884354  71
## 3                 max f0point5  0.768421 0.961089  38
## 4                 max accuracy  0.668254 0.983773  42
## 5                max precision  1.000000 1.000000   0
## 6                   max recall  0.000000 1.000000 399
## 7              max specificity  1.000000 1.000000   0
## 8             max absolute_mcc  0.668254 0.907422  42
## 9   max min_per_class_accuracy  0.078849 0.930101 190
## 10 max mean_per_class_accuracy  0.205338 0.938416  95
##
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/F
## H2OBinomialMetrics: drf
## ** Reported on validation data. **
##
```

```
## MSE:  0.003426902
## RMSE:  0.05853975
## LogLoss:  0.02373937
## Mean Per-Class Error:  0.1151149
## AUC:  0.9911799
## Gini:  0.9823597
##
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
##            0  1    Error       Rate
## 0      56544 13 0.000230  =13/56557
## 1         23 77 0.230000    =23/100
## Totals 56567 90 0.000635  =36/56657
##
## Maximum Metrics: Maximum metrics at their respective thresholds
##                         metric threshold    value idx
## 1                       max f1  0.899940 0.810526   5
## 2                       max f2  0.740717 0.821918  13
## 3                 max f0point5  0.959930 0.853365   2
## 4                 max accuracy  0.899940 0.999365   5
## 5                max precision  0.959930 0.898734   2
## 6                   max recall  0.000880 1.000000 389
## 7              max specificity  0.999832 0.999876   0
## 8             max absolute_mcc  0.899940 0.811337   5
## 9   max min_per_class_accuracy  0.124032 0.960000 186
## 10 max mean_per_class_accuracy  0.120083 0.963031 191
##
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/
##
##
## Scoring History:
##             timestamp   duration number_of_trees training_rmse
## 1 2018-07-27 09:14:19 23.606 sec               0
## 2 2018-07-27 09:14:19 23.664 sec              10       0.13963
## 3 2018-07-27 09:14:20 23.787 sec              20       0.12987
## 4 2018-07-27 09:14:20 23.987 sec              30       0.12637
## 5 2018-07-27 09:14:20 24.254 sec              40       0.12502
## 6 2018-07-27 09:14:20 24.597 sec              50       0.12386
##   training_logloss training_auc training_lift
## 1
## 2          0.32702      0.94750       9.83025
## 3          0.20956      0.95569       9.91370
## 4          0.15526      0.96181       9.95960
## 5          0.12349      0.96773       9.95960
## 6          0.09098      0.97321       9.95960
##   training_classification_error validation_rmse validation_logloss
## 1
## 2                       0.01844         0.06954            0.03272
## 3                       0.01690         0.06308            0.02873
## 4                       0.01521         0.06109            0.02798
## 5                       0.01623         0.05950            0.02766
## 6                       0.01623         0.05854            0.02374
##   validation_auc validation_lift validation_classification_error
## 1
## 2        0.99155        91.79618                         0.00072
```

```
## 3        0.99390        79.09627                         0.00069
## 4        0.99266        90.93099                         0.00065
## 5        0.99169        89.30175                         0.00065
## 6        0.99118        88.93250                         0.00064
##
## Variable Importances: (Extract with `h2o.varimp`)
## =================================================
##
## Variable Importances:
##   variable relative_importance scaled_importance percentage
## 1      V14         2498.389404          1.000000   0.221581
## 2      V10         1603.708008          0.641897   0.142232
## 3      V17         1556.449585          0.622981   0.138041
## 4      V12          964.447693          0.386028   0.085536
## 5      V16          905.607727          0.362477   0.080318
##
## ---
##    variable relative_importance scaled_importance percentage
## 25       V1           44.570545          0.017840   0.003953
## 26      V25           42.996765          0.017210   0.003813
## 27     Time           28.612165          0.011452   0.002538
## 28   Amount           24.659983          0.009870   0.002187
## 29      V28           23.468691          0.009394   0.002081
## 30       V8           20.460659          0.008190   0.001815
```

```r
#get the actual number of trees
ntrees <- best_model@model$model_summary$number_of_trees
ntrees
```

```
## [1] 50
```

```r
#get the actual max depth
mdepth <- best_model@model$model_summary$max_depth
mdepth
```

```
## [1] 15
```

```r
#Validation set used to select the best model
#Evaluate the model performance on test set to get honest estimate of model performance
best_model_perf <- h2o.performance(model = best_model,
newdata = test.hex)

#model performance metrics on test set
best_model_perf
```

```
## H2OBinomialMetrics: drf
##
## MSE:  0.003808079
## RMSE:  0.06170964
## LogLoss:  0.02584733
## Mean Per-Class Error:  0.09494692
## AUC:  0.9709288
## Gini:  0.9418577
##
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
##            0    1    Error        Rate
## 0      57099   24 0.000420  =24/57123
```

```
## 1         18  77 0.189474     =18/95
## Totals 57117 101 0.000734  =42/57218
##
## Maximum Metrics: Maximum metrics at their respective thresholds
##                         metric threshold    value idx
## 1                       max f1  0.825196 0.785714  10
## 2                       max f2  0.825196 0.800416  10
## 3                 max f0point5  0.825196 0.771543  10
## 4                 max accuracy  0.825196 0.999266  10
## 5                max precision  0.959883 0.792208   2
## 6                   max recall  0.000172 1.000000 397
## 7              max specificity  0.999832 0.999755   0
## 8             max absolute_mcc  0.825196 0.785716  10
## 9   max min_per_class_accuracy  0.064022 0.914150 257
## 10 max mean_per_class_accuracy  0.128552 0.928628 184
##
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/
```

```r
#MSE
h2o.mse(best_model_perf) #0.003808079, not really relevant for classification problems
```

```
## [1] 0.003808079
```

```r
#RMSE
h2o.rmse(best_model_perf) #0.06170964, not really relevant for classification problems
```

```
## [1] 0.06170964
```

```r
#Log Loss
h2o.logloss(best_model_perf) #0.02584733
```

```
## [1] 0.02584733
```

```r
#AUC
h2o.auc(best_model_perf)  #0.9709288, slighly less than the AUC on the validation set
```

```
## [1] 0.9709288
```

```r
#Gini
h2o.giniCoef(best_model_perf) #0.9418577
```

```
## [1] 0.9418577
```

```r
#best model performance metrics at all thresholds
test.scores <- best_model_perf@metrics$thresholds_and_metric_scores

#find best threshold that maximizes F1
best.thresh <- test.scores$threshold[which.max(test.scores$f1)]

#create dataframe with performance metrics of model on test data at
#threshold that maximizes F1
metrics <- data_frame(
Precision = h2o.precision(best_model_perf, best.thresh),
Recall = h2o.recall(best_model_perf, best.thresh),
F1 = h2o.F1(best_model_perf, best.thresh),
AUC = h2o.auc(best_model_perf),
LogLoss = h2o.logloss(best_model_perf),
Gini = h2o.giniCoef(best_model_perf),
```

```
Accuracy = h2o.accuracy(best_model_perf, best.thresh),
Mean_Accuracy = h2o.mean_per_class_accuracy(best_model_perf, best.thresh)
)

#view metrics
kable(metrics) %>%
kable_styling(bootstrap_options = "striped", full_width = F)
```

Precision	Recall	F1	AUC	LogLoss	Gini	Accuracy	Mean_Accuracy
0.7623762	0.8105263	0.7857143	0.9709288	0.0258473	0.9418577	0.999266	0.9050531

```
#overall it appears that my model performed well
```

Results: Out of all the models with varying number of trees and maximum tree depths, I choose the model with the highest AUC on the validation set as the best model. This model used 50 trees and had a max depth of 15. I then evaluated the model performance on my test set.

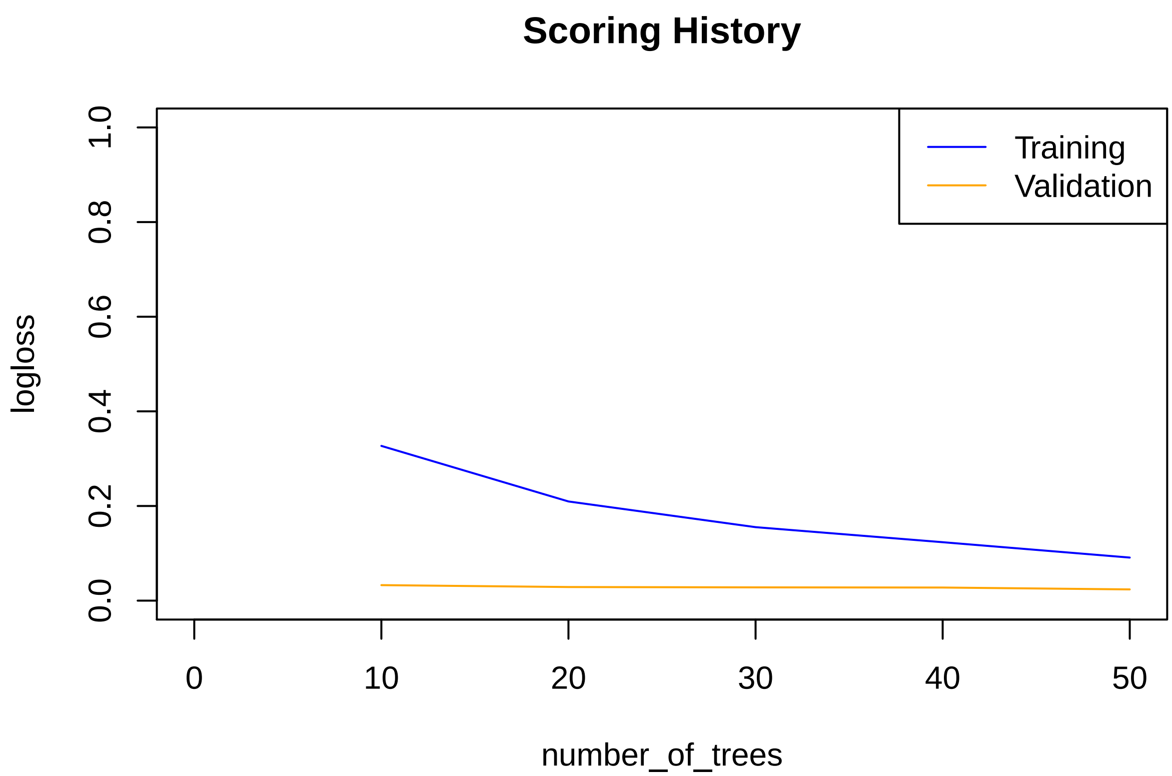
The test set performance metrics at the threshold that maximizes the F-statistic:
LogLoss: 0.02584733
AUC: 0.9709288
Gini: 0.9418577
Precision: 0.762
Recall: 0.811
F1: 0.786

I used the above metrics to determine my model's performance. As my dataset was imbalanced I primarily used Precision, Recall and the F-score to evaluate my model performance. All of which indicate that the model is good.

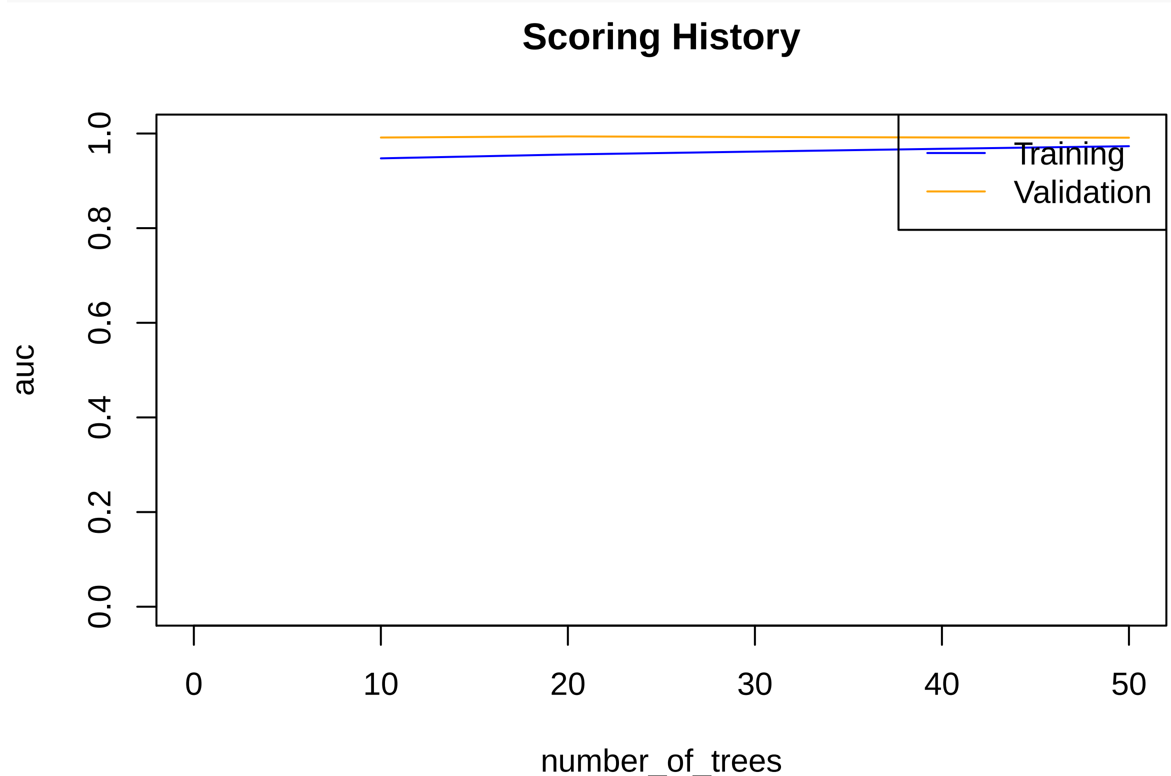
In a business use case a credit card company would prefer more false positives than false negatives. That is the company would rather incorrectly identify a transaction as fraud than identify a fraudulent transaction as legitimate. Therefore, for my performance metrics I preferred high Recall, which is a low false negative rate rather than high Precision, which is a low false positive rate.

4 Plot Performance Metrics

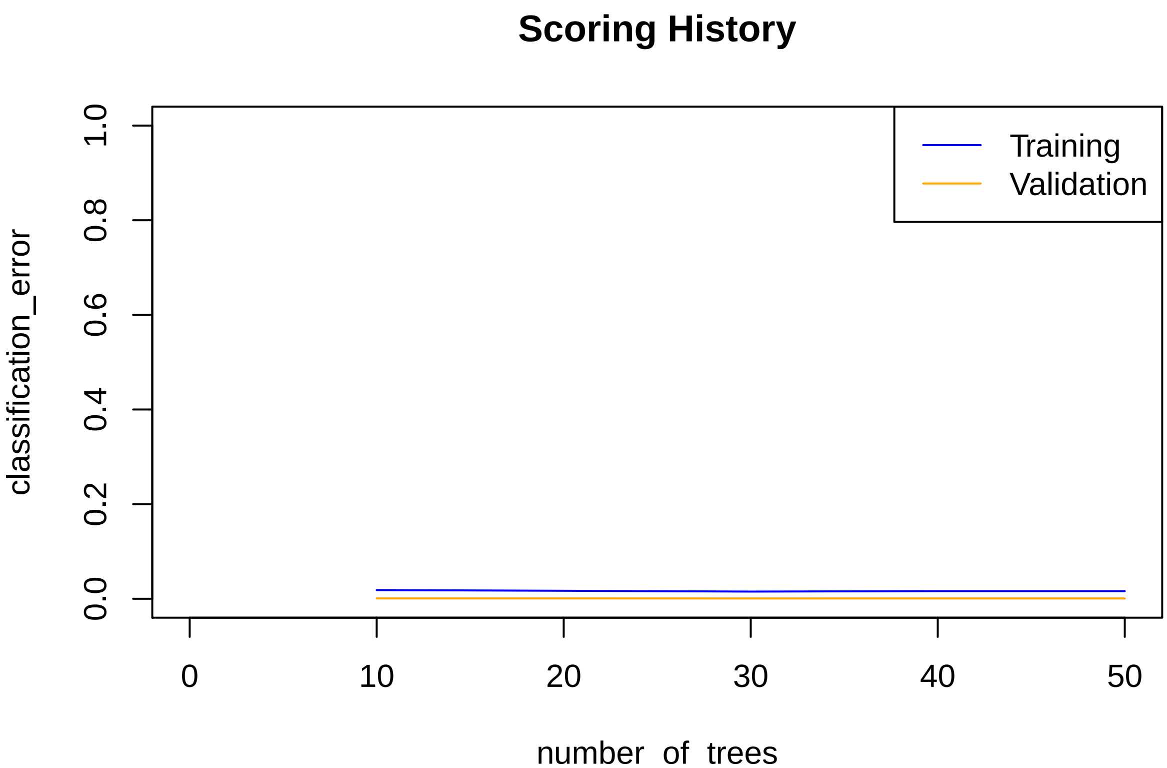
```
#scoring history of train and validation set
scoring_history <- as.data.frame(best_model@model$scoring_history)

#LogLoss
plot(best_model,
timestep = "number_of_trees",
metric = "logloss")
```




```r
#AUC
plot(best_model,
timestep = "number_of_trees",
metric = "AUC")
```



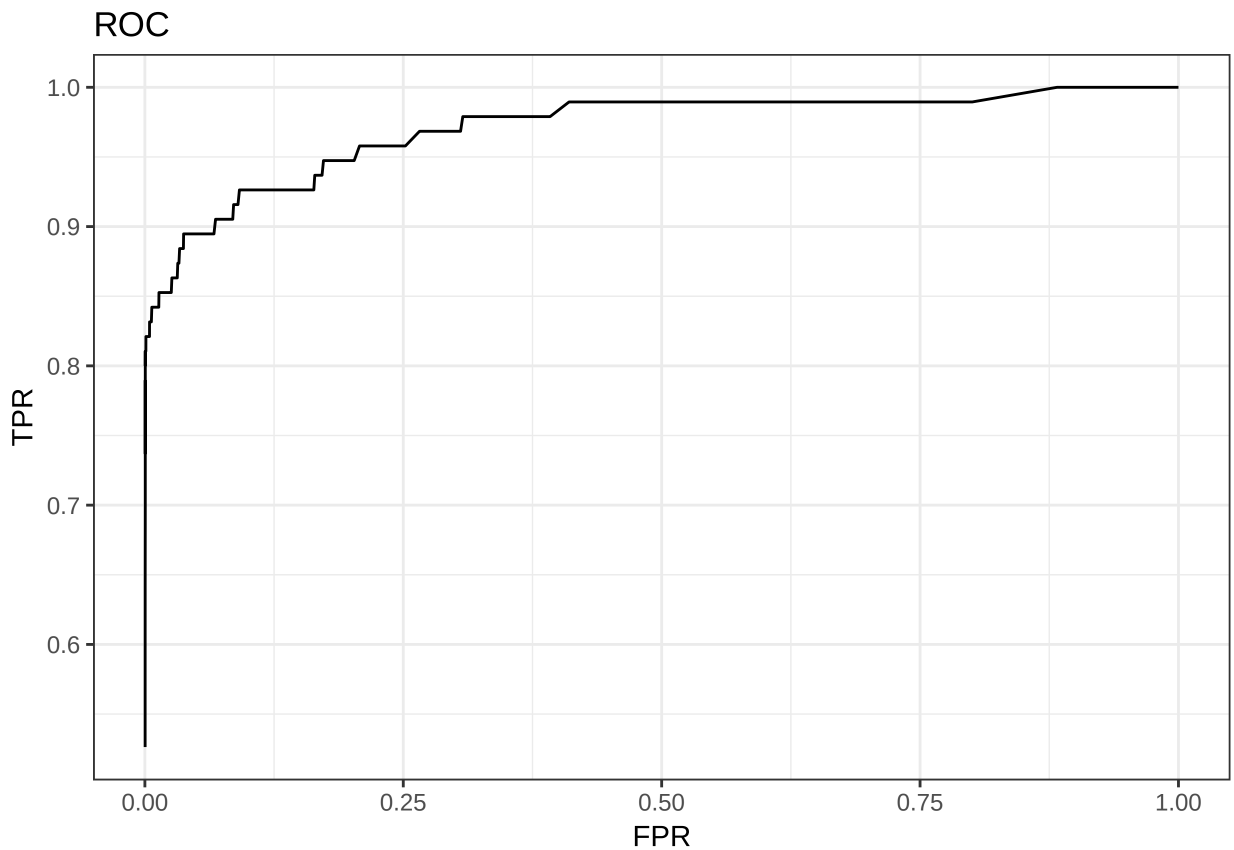
```r
#Classification Error
plot(best_model,
timestep = "number_of_trees",
metric = "classification_error")
```




```r
#ROC
tpr = as.data.frame(h2o.tpr(best_model_perf))
fpr = as.data.frame(h2o.fpr(best_model_perf))
ROC_out <- merge(tpr, fpr, by = 'threshold')
head(ROC_out)
```

```
##      threshold       tpr       fpr
## 1 6.761139e-05 1.0000000 1.0000000
## 2 1.254214e-04 1.0000000 0.9529261
## 3 1.719925e-04 1.0000000 0.8825517
## 4 2.271066e-04 0.9894737 0.8005357
## 5 3.016923e-04 0.9894737 0.7027817
## 6 3.750610e-04 0.9894737 0.6383943
```

```r
#Plot ROC
ggplot(ROC_out, aes(x = fpr, y = tpr)) +
theme_bw() +
geom_line() +
ggtitle("ROC") + ylab("TPR") + xlab("FPR")
```




```r
#ROC curves plot the tradeoff between recall and false positive rates

#Precision Recall
#Evaluate the predictive performance of model based on precision and recall
head(h2o.F1(best_model_perf))
```

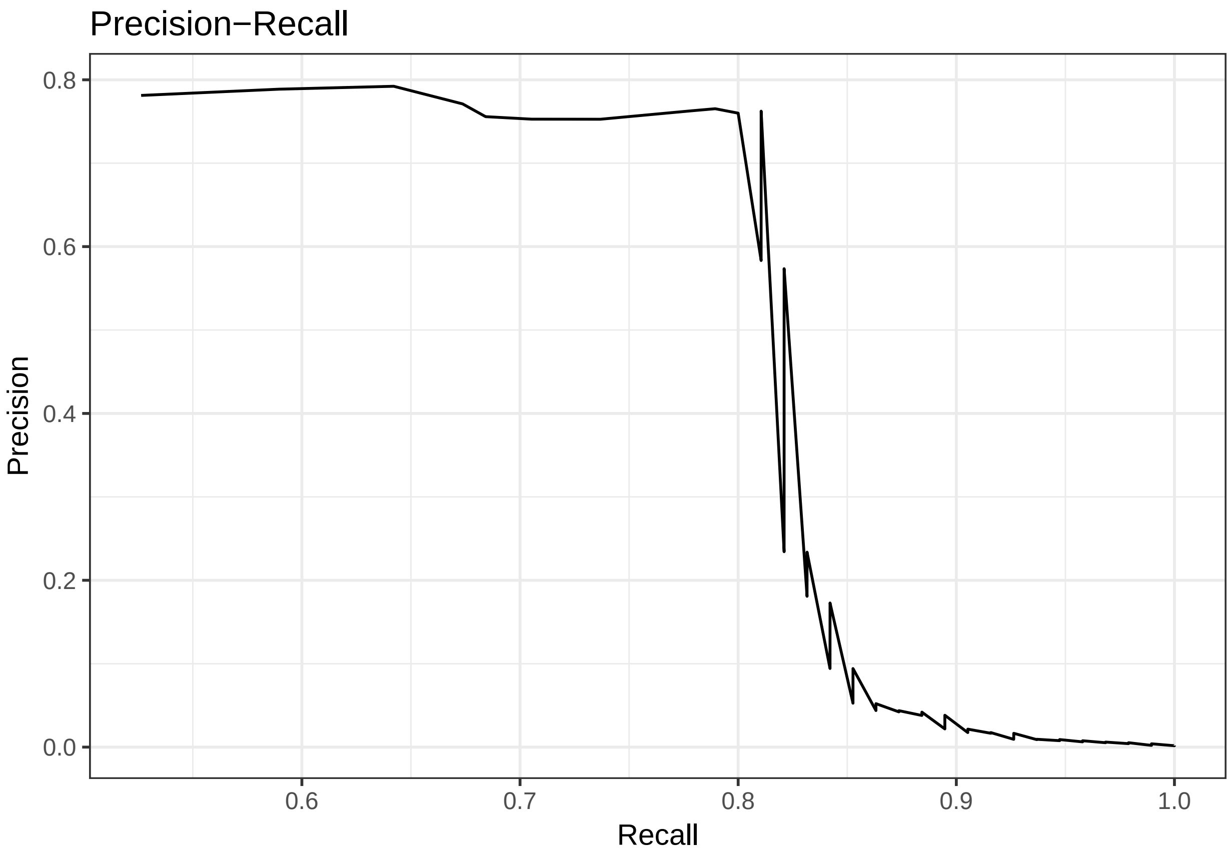
```
##   threshold        f1
## 1 0.9998318 0.6289308
## 2 0.9799143 0.6746988
## 3 0.9598832 0.7093023
## 4 0.9399402 0.7191011
## 5 0.9199402 0.7182320
## 6 0.8998804 0.7282609
```

```r
precision = as.data.frame(h2o.precision(best_model_perf))
recall = as.data.frame(h2o.recall(best_model_perf))
PR_out <- merge(precision, recall, by = 'threshold')
head(PR_out)
```

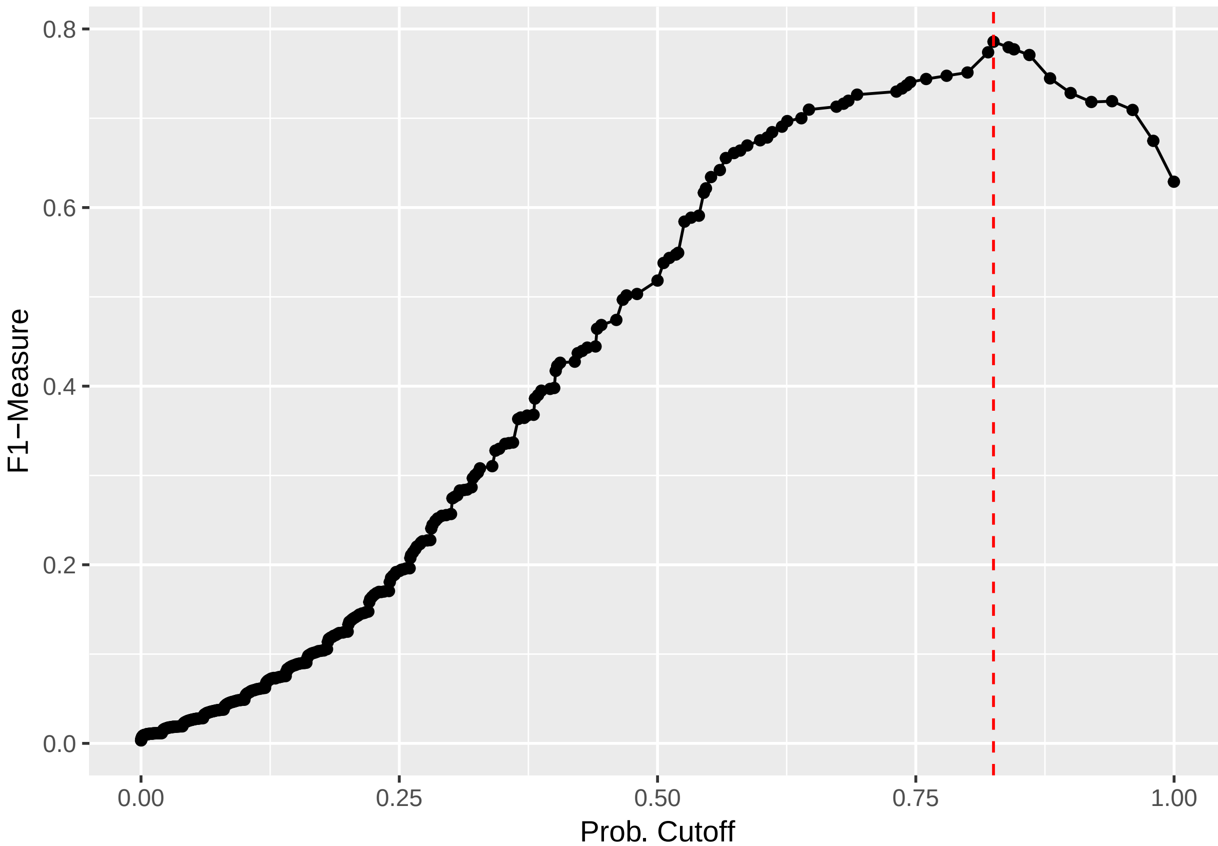
```
##      threshold   precision       tpr
## 1 6.761139e-05 0.001660317 1.0000000
## 2 1.254214e-04 0.001742192 1.0000000
## 3 1.719925e-04 0.001880853 1.0000000
## 4 2.271066e-04 0.002051372 0.9894737
## 5 3.016923e-04 0.002336042 0.9894737
## 6 3.750610e-04 0.002571046 0.9894737
```

```r
#Plot Precision - Recall
ggplot(PR_out, aes(x = tpr, y = precision)) +
```

```r
theme_bw() +
geom_line() +
ggtitle("Precision-Recall") + ylab("Precision") + xlab("Recall")
```




```r
#Precision-recall curves shows the tradeoff between precision and recall
#for different thresholds. Useful measure of prediction success when
#modeling rare events (classes very imbalanced).
#High precision relates to a low false positive rate,
#and high recall relates to a low false negative rate. High scores
#for both show that the classifier is returning accurate results
#(high precision), as well as returning a majority of all
#positive results (high recall).
#(source: http://scikit-learn.org/stable/auto_examples/model_selection/plot_precision_recall.html)

#plot threshold that maximizes F1
ggplot(test.scores, aes(x = threshold, y = f1)) +
geom_line() +
geom_point() +
geom_vline(xintercept = best.thresh,
linetype = "dashed",
color = "red") +
labs(x = "Prob. Cutoff", y = "F1-Measure")
```




```r
# All done. Shut down H2O.
h2o.shutdown(prompt = FALSE)
```

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