Credit Card Fraud Detection, in R

Joshua Bela

May 4, 2020

import libraries

library(magrittr)  
library(caret)

## Loading required package: lattice

## Loading required package: ggplot2

library(xgboost)

## Warning: package 'xgboost' was built under R version 3.6.2

import data

data = read.csv('data/creditcard.csv')

For each variable, determine if it’s eligible to be a predictor using T-tests. The Pearson correlation must be non-zero, with a 95% confidence interval, for eligibility. Create a new data set containing only eligible predictors.

eligible\_predictors = NULL  
for(predictor in names(data)){  
 confidence\_interval = (cor.test(  
 data$Class,  
 data[,predictor],  
 method = "pearson"  
 ))$conf.int  
 if(confidence\_interval[1]\*confidence\_interval[2] > 0)  
 eligible\_predictors = c(eligible\_predictors, predictor)  
}  
  
data0 = data[,eligible\_predictors]

Data Engineering Ensure all variables have the correct data types.

data1 = data0  
data1$Class = ifelse(  
 data0$Class == 1,  
 "fraud",  
 "clean"  
) %>% as.factor()

Apply regression using extreme gradient boosting in Caret.

# create the tuning grid  
xgb\_grid\_1 = expand.grid(  
 nrounds = 10,  
 eta=c(.3),  
 gamma = c(0, .1, 1, 10),  
 max\_depth = seq(2, 10, by = 2),  
 min\_child\_weight=1,  
 subsample=0.5,  
 colsample\_bytree=0.5  
)  
  
# modify more settings  
xgb\_trcontrol\_1 = trainControl(  
 method = "cv",  
 number = 2,  
 verboseIter = TRUE,  
 returnData = FALSE,  
 returnResamp = "all", # save losses across all models  
 classProbs = TRUE, # set to TRUE for AUC to be computed  
 summaryFunction = twoClassSummary,  
 allowParallel = TRUE  
)  
  
# train the models on the data  
xgb\_train\_1 = train(  
 x = data1 %>% subset(select = -Class),  
 y = data1$Class,  
 trControl = xgb\_trcontrol\_1,  
 tuneGrid = xgb\_grid\_1,  
 method = "xgbTree",  
 scale\_pos\_weight = 5  
)

# print out the parameters for the best model  
print(xgb\_train\_1$bestTune)

## nrounds max\_depth eta gamma colsample\_bytree min\_child\_weight subsample  
## 17 10 10 0.3 0 0.5 1 0.5

# print out parameters for all models in descending order of ROC  
xgb\_train\_1$results[order(-xgb\_train\_1$results$ROC),]

## eta max\_depth gamma colsample\_bytree min\_child\_weight subsample nrounds  
## 17 0.3 10 0.0 0.5 1 0.5 10  
## 15 0.3 8 1.0 0.5 1 0.5 10  
## 5 0.3 4 0.0 0.5 1 0.5 10  
## 2 0.3 2 0.1 0.5 1 0.5 10  
## 19 0.3 10 1.0 0.5 1 0.5 10  
## 13 0.3 8 0.0 0.5 1 0.5 10  
## 18 0.3 10 0.1 0.5 1 0.5 10  
## 20 0.3 10 10.0 0.5 1 0.5 10  
## 1 0.3 2 0.0 0.5 1 0.5 10  
## 7 0.3 4 1.0 0.5 1 0.5 10  
## 14 0.3 8 0.1 0.5 1 0.5 10  
## 10 0.3 6 0.1 0.5 1 0.5 10  
## 6 0.3 4 0.1 0.5 1 0.5 10  
## 16 0.3 8 10.0 0.5 1 0.5 10  
## 11 0.3 6 1.0 0.5 1 0.5 10  
## 8 0.3 4 10.0 0.5 1 0.5 10  
## 4 0.3 2 10.0 0.5 1 0.5 10  
## 12 0.3 6 10.0 0.5 1 0.5 10  
## 9 0.3 6 0.0 0.5 1 0.5 10  
## 3 0.3 2 1.0 0.5 1 0.5 10  
## ROC Sens Spec ROCSD SensSD SpecSD  
## 17 0.9234855 0.9999402 0.6686992 0.0043749470 2.487084e-05 0.014372089  
## 15 0.9234520 0.9999437 0.6971545 0.0042556899 2.984493e-05 0.020120925  
## 5 0.9229388 0.9999578 0.6727642 0.0047715034 4.974129e-05 0.025869760  
## 2 0.9223477 0.9999648 0.5081301 0.0026965652 1.989661e-05 0.040241849  
## 19 0.9223364 0.9999472 0.6483740 0.0028339396 2.487080e-05 0.037367432  
## 13 0.9223271 0.9999261 0.6768293 0.0001409858 2.487091e-05 0.020120925  
## 18 0.9215078 0.9999437 0.6951220 0.0014661853 9.948496e-06 0.028744178  
## 20 0.9214547 0.9999508 0.6239837 0.0015908618 9.948461e-06 0.037367432  
## 1 0.9214268 0.9999683 0.4756098 0.0014720899 4.974266e-06 0.034493014  
## 7 0.9214178 0.9999543 0.6991870 0.0014184464 3.481898e-05 0.022995342  
## 14 0.9213934 0.9999332 0.6849593 0.0015187982 6.466374e-05 0.002874418  
## 10 0.9213706 0.9999472 0.6829268 0.0016097476 3.481902e-05 0.045990685  
## 6 0.9204355 0.9999437 0.6829268 0.0001109281 3.979315e-05 0.022995342  
## 16 0.9203912 0.9999472 0.5914634 0.0086149548 1.492259e-05 0.008623253  
## 11 0.9203759 0.9999437 0.6686992 0.0001176718 1.989671e-05 0.077609281  
## 8 0.9194454 0.9999472 0.6341463 0.0015808022 2.487080e-05 0.011497671  
## 4 0.9194181 0.9999648 0.5020325 0.0015796195 9.948041e-06 0.008623253  
## 12 0.9174345 0.9999613 0.6077236 0.0042508039 3.481895e-05 0.043116267  
## 9 0.9173751 0.9999543 0.6402439 0.0101964430 3.481898e-05 0.002874418  
## 3 0.9152302 0.9999297 0.5264228 0.0073059265 9.947866e-06 0.077609281

# generate predictions using final model  
predictions = predict(  
 xgb\_train\_1,  
 data1 %>% subset(select = -Class)  
)  
  
# generate confusion matrix based on predictions and true values  
confusionMatrix(  
 predictions,  
 data1$Class  
) %>% print()

*Final Results*

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction clean fraud  
## clean 284314 110  
## fraud 1 382  
##   
## Accuracy : 0.9996   
## 95% CI : (0.9995, 0.9997)  
## No Information Rate : 0.9983   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.873   
##   
## Mcnemar's Test P-Value : < 2.2e-16   
##   
## Sensitivity : 1.0000   
## Specificity : 0.7764   
## Pos Pred Value : 0.9996   
## Neg Pred Value : 0.9974   
## Prevalence : 0.9983   
## Detection Rate : 0.9983   
## Detection Prevalence : 0.9987   
## Balanced Accuracy : 0.8882   
##