Various Detection Methods for Unbalanced Credit Card Fraud Data

Jamel M. Thomas

April 2017

1 Summary

In this paper, we obtain credit card transactions from September 2013 of European card holders. These transactions were recorded over the span of two days. The data contains $284,807$ transactions, with only 492 cases of fraud. Therefore, this data set is highly unbalanced. Moreover, a considerable drawback to the data is due to a PCA transformation of 28 features. Unfortunately, due to confidentiality issues, there is no way to extract more information about these features. There are only three features that have not been transformed: Time, Amount, and Class. These are the time in seconds from the current transaction to the last transaction, the amount of purchase at the time of transaction, and whether the transaction was fraud or not, respectively.

We address the issues of decision tree algorithms when classifying highly unbalanced data sets. We consider Random Forest and Adaptive Boosting as a generalized solution to the issue and determine which if one is better suited for the class of data over the other. We also attempt to identify variables that significantly contribute to the productivity of the model. We find that with the tree algorithm with the smallest out of bag error still performs significantly worse than boosting and random forests. However, to our surprise, random forests outperformed boosting in all cases. We also consider the accuracy and complexity of the two ensemble methods. We determine, however, that with a proper threshold value we are able to detect credit card fraud in virtually all instances of our test data.

2 Introduction

The convenience of credit cards is apparent. Today, individuals have the ability to purchase items from across the globe with a few valid numbers and dates. However, some people are able to take advantage of this convenience by stealing credit card information. Therefore, it is of great significance that banks, who issue these cards, are able to detect fraud quickly, accurately, and precisely. This will not only save their customers time, it saves money.

The data includes $284,807$ variables with 28 features. Of these observations, 492 are classified as fraud. Therefore, the data is highly unbalanced and is biased towards classifying an observation as not fraud. Moreover, there are only three variables that are not anonymous. With this data, we want attempt to accurately predict the class of a transaction. Assuming we obtain an algorithm that has a low type II error - predicting no fraud when it truly is - we can minimize rampant fraudulent credit card usage. Although Type I error is a nuisance for the consumer, we recognize that the impacts of Type II error carry a greater weight when undetected. Therefore, our goal in this paper is to find a prediction algorithm, based on an ensemble trees, to accurately predict the classification of fraud.

3 Methods

The data is obtained from the research collaboration of Worldline and the Machine Learning Group of ULB (Université Libre de Bruxelles) [1]. There is no further information on the collection of the data. All of the following algorithms were done in R version 1.0.136.

We begin our analysis by splitting the data into training and testing sets. We conduct a stratified sample such that 80% of the data is trained and 20% tested. All evaluation of the various methods are based on the testing set. Even though the decision tree algorithms conduct cross validation, we perform this on the training set. We build an original tree with 25 - fold cross validation. This is because the plot of the minimum cross validated error value as a function of V obtains the minimizer at $V = 30$. After pruning based on the one standard error rule, we obtain a tree with 11 terminal nodes. We consider the loss matrix that penalizes type II error by factor of 50 then 100. However, after pruning we obtain trees with abhorrent classification of fraud. Therefore, we proceed with two ensemble methods: random forests and boosting. In our random forest algorithm, we build 1000 trees. There were four boosting algorithms: discrete, real, and gentle, and gentle stumps. Real converged at 199 iterations, while discrete and gentle only contained 89 and 140 respectively. The gentle stump boosting algorithm was built on 1000 iterations, as was necessary to uncover decent predictive accuracy. We also build a boosting algorithm via real that boosted stumps, not shown in Figure 2. This converged at 414 iterations, however did not perform nearly as well as gentle stump boosting. All are very similar in concept, but the implementation varies: build a multiple decision trees iteratively, each time giving more importance to the misclassified observations. Finally, based on random forests and boosting, we estimated density plot of the variable importance to visualize the differences among the two types of transactions.

4 Results

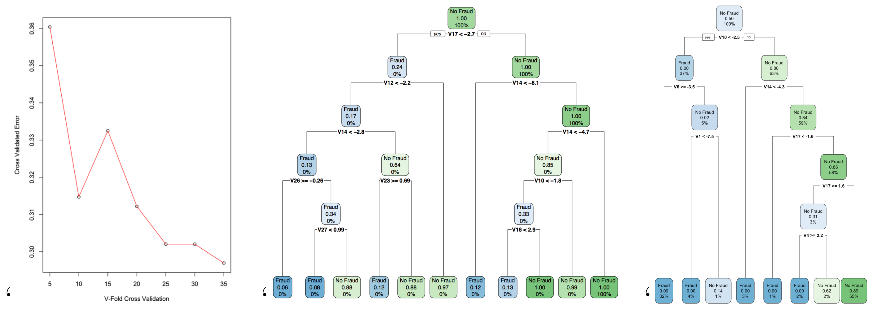


Figure 1: Unweighted Decision Tree (left) & Under sampled tree with loss matrix

Figure 1 shows the minimum cross validated error as a function of V on the left. In the middle, it shows an unweighted decision tree based on 25-fold cross validation after pruning via the 1 standard error rule. The plot on the right shows the tree built on under-sampled data. This tree also includes a loss matrix that penalizes failing to classify fraud, with a cost of 100.

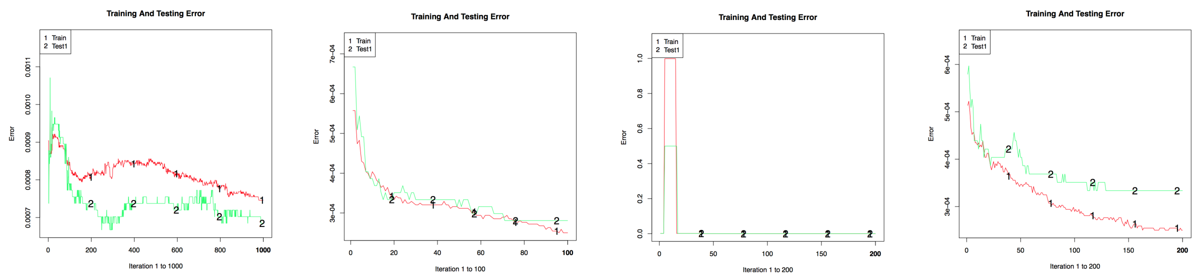


Figure 2: Training and Testing error: Gentle Stump Boost, Discrete, Real, Gentle Boost

Figure 2 shows the training error as a function of iteration. The training error is in red, and testing error in green. There were different iterations per boosting algorithm. The left-most is the training and testing error for gentle stump boosting built on 1000 iterations. The middle-left shows a plot of discrete boosting. The middle-right shows real boost. The far right plot of figure 2 shows gentle boost.

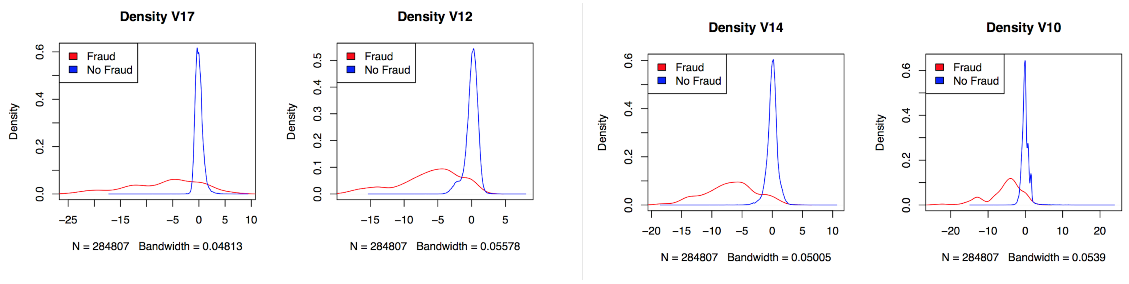


Figure 3: Probability Distribution of Variables by Importance

Finally, the last figure shows variable densities of fraud and no fraud based on the most important variables in random forest. This plot shows the differences in distributions of those individuals who were classified as fraud versus those classified as not fraud.

5 Conclusion

Based on our work we recognize tat decision trees are not the most accurate method of classification. Moreover, in order to reduce over-training bias, we pruned the tree. Even with this, our results were not very accurate. Of individuals that were truly fraud, the tree misclassified 22.4% of them. In order to attempt to reduce the type two error, we consider a 0.05 threshold value. That is, if the probability of fraud is more than this threshold, we classify as fraud. This brought the same misclassification as defined above down to 15%. The estimated Cohen's kappa value is 0.76. When weighting for type two error, we obtained a tree that misclassified 24% of transactions truly classified as fraud. Therefore, we proceeded with boosting.

There were four boosting algorithms implemented. Boosting via Discrete stumps, based on 414 iterations, did not perform as well as originally hypothesized. It contained a smaller type one error, but a similar type two error as the decision tree at about 15%. The misclassification of discrete, real and gentle only increased at 16%, 18%, and 16% respectively. In order to improve performance, there is not much to try. Other than manipulating the threshold, we could attempt more methods such as gradient boosting. The plot of the gentle stumps contains a lower testing error than training as the number of iterations climb. This seems to indicate that this particular method is more generalizable to testing data than the others. Discrete boosting and gentle boosting performed similarly, with discrete boosting slightly outshining gentle. Real boosting, however, contained massive updates in pure regions, as seen by the plots. Overall, these methods may still not perform well in practice.

Random forests performed the best of all methods. Random forests misclassified two observations of the testing set. With a threshold of only 0.45, we are able to bring this testing misclassification down to one observation. The estimated Cohen's kappa value is 0.99. This method, out of the box, performed better than all other methods hands down.

In conclusion, random forest outperformed all other algorithms. This is not surprising, for the most part. Our expectations of various boosting algorithms were not met, however this is more so due to a lack of tuning. Decision trees met expectations and did no better than random guessing. Therefore, if we only use random forests, we can correctly classify virtually all instances of credit card fraud with minimal computational cost [2].

References

[1] Andrea Dal Pozzolo. "Credit Card Fraud Detection". In: (2016). URL: `https://www.kaggle.com/dalpozz/creditcardfraud`.

[2] Robert E. Schapire. "Experiments with a New Boosting Algorithm". In: (1996). URL: `http://web.eecs.utk.edu/~leparker/Courses/CS425-528-fall10/Handouts/AdaBoost.M1.pdf`.

Appendix

Jamel M. Thomas

12 April 2017

```
oldw <- getOption("warn")
options(warn = -1) #Supress warnings
################################################################
card <- read.csv("~/Desktop/DataMiningProj/creditcard.csv")


################################################################
# Packages
library(rpart) #Decision tree
library(rpart.plot) #Decision tree plotting.
library(randomForest) #Random Forest

## randomForest 4.6-12
## Type rfNews() to see new features/changes/bug fixes.

library(caret) #Data Processing

## Loading required package:  lattice
## Loading required package:  ggplot2
##
## Attaching package:  'ggplot2'
## The following object is masked from 'package:randomForest':
##
##    margin

library(ada) #Boosting
library(rlist) #Save and load data
################################################################
# Create a Factor


card$Class <- factor(card$Class,
                     levels = c("1", "0"),
                     labels = c("Fraud", "No Fraud")) #Create a Fac-
tor
```

```
################################################################
# Summary Stats
table(card$Class)

##
##    Fraud No Fraud
##      492   284315

table(card$Class) / length(card$Class)

##
##       Fraud    No Fraud
## 0.001727486 0.998272514


##################################################################
#First, let's split the data
set.seed(702)
# From the cart package: Split the data into 80% Train - 20% test
trainIndex <- createDataPartition(card$Class, p = .8,
                                                list = FALSE,
                                                times = 1) #Strat-
ified
cardTrain <- card[trainIndex, ]; cardTest <- card[-trainIndex, ]

table(cardTrain$Class); table(cardTest$Class) #New data proportions

##
##    Fraud No Fraud
##      394   227452
##
##    Fraud No Fraud
##       98    56863

############### Undersampling to lower No Fraud Size
undersample <- sample(which(cardTrain$Class == "No Fraud"), 1000)
unTrain <- rbind(cardTrain[cardTrain$Class == "Fraud",]
                , cardTrain[undersample,])
table(unTrain$Class)

##
##    Fraud No Fraud
##      394     1000
```

```r
########################################################################
# The original unweighted tree
#########################################################################
#Without xval
#Determine a decent cross validation exhaustively
c.5 <- rpart.control(xval = 5, cp = 0); c.10 <- rpart.control(xval = 10, cp = 0);
c.15 <- rpart.control(xval = 15, cp = 0); c.20 <- rpart.control(xval = 20, cp = 0);
c.25 <- rpart.control(xval = 25, cp = 0); c.30 <- rpart.control(xval = 30, cp = 0);
c.35 <- rpart.control(xval = 35, cp = 0)
#Build the trees
no.c5 <- rpart(cardTrain$Class ~., data = cardTrain, control = c.5)
no.c10 <- rpart(cardTrain$Class ~., data = cardTrain, control = c.10)
no.c15 <- rpart(cardTrain$Class ~., data = cardTrain, control = c.15)
no.c20 <- rpart(cardTrain$Class ~., data = cardTrain, control = c.20)
no.c25 <- rpart(cardTrain$Class ~., data = cardTrain, control = c.25)
no.c30 <- rpart(cardTrain$Class ~., data = cardTrain, control = c.30)
no.c35 <- rpart(cardTrain$Class ~., data = cardTrain, control = c.35)


fold25 <- predict(no.c25, newdata = cardTest[,-31], type = "prob")
#Consider two thresholds .5 and .05
no.c25$cptable
```

```
##             CP nsplit rel error    xerror       xstd
## 1 0.4695431472      0 1.0000000 1.0000000 0.05033569
## 2 0.0786802030      1 0.5304569 0.5355330 0.03685054
## 3 0.0507614213      2 0.4517766 0.4568528 0.03403834
## 4 0.0219966159      3 0.4010152 0.4111675 0.03229288
## 5 0.0164974619      6 0.3350254 0.3832487 0.03117800
## 6 0.0152284264      9 0.2791878 0.3553299 0.03002163
## 7 0.0012690355     10 0.2639594 0.3020305 0.02767986
## 8 0.0008460237     12 0.2614213 0.3147208 0.02825507
## 9 0.0000000000     15 0.2588832 0.3147208 0.02825507
```

```r
no.c25p <- prune(no.c25, cp = 0.0013) #Prune
fold25p <- predict(no.c25p, newdata = cardTest[,-31], type = "prob")
#Generate Pr
Prediction.25 <- ifelse(fold25p[,1] > .5, "Fraud", "No Fraud")
Prediction.25.t <- ifelse(fold25p[,1] > .05, "Fraud", "No Fraud")

table(Truth = cardTest$Class, Prediction.25)
```

```
##           Prediction.25
## Truth      Fraud No Fraud
##   Fraud       78       20
##   No Fraud     8    56855
```

```
table(Truth = cardTest$Class, Prediction.25.t)
```

```
##            Prediction.25.t
## Truth       Fraud No Fraud
##   Fraud        81       17
##   No Fraud     20    56843
```

```
lmat <- matrix(c(0,50, 1,0), nrow = 2) #Misclassification with 25 fold CV
tree1 <- rpart(Class ~. , data = cardTrain, control = c.25,
               parms = list(loss = lmat))

###############################################################
# We can possibly increse the weights
# Increase weights
###############################################################
#Consider the following loss matrix
print(lmat2 <- matrix(c(0,100, 1,0), nrow = 2) )

tree2 <- rpart(Class ~. , data = cardTrain, control = c.25,
               parms = list(loss = lmat2))


tree2$cptable #CP Table and ptuning
```

```
##            CP nsplit rel error    xerror     xstd
## 1 0.14974619      0 1.0000000 100.00000 5.033569
## 2 0.04145516      1 0.8502538  87.56853 4.710688
## 3 0.03722504      6 0.6218274  64.48731 4.042769
## 4 0.02538071      9 0.5101523  50.02538 3.560820
## 5 0.02030457     10 0.4847716  45.71827 3.403845
## 6 0.01099831     12 0.4441624  43.43909 3.317725
## 7 0.00676819     15 0.4111675  40.39594 3.199283
## 8 0.00000000     18 0.3908629  39.63706 3.168979
```

```
tree2.p <- prune(tree2, cp = 0.007)
tree2.pred <- predict(tree2.p, newdata = cardTest[,-31])
Prediction2 <- ifelse(tree2.pred[,1] > .5, "Fraud", "No Fraud")
Prediction2 <- factor(Prediction2,
                      levels = c("Fraud", "No Fraud"))

table(Truth = cardTest$Class, Prediction2) #Prediction
```

```
##            Prediction2
## Truth       Fraud No Fraud
##   Fraud        74       24
##   No Fraud     15    56848
```

```
#Therefore, we need another method besides trees
#####################################################
# Random forests
#####################################################
```

```
rF1 <- randomForest(Class ~., data = cardTrain, ntree = 1000) #Ran-
dom Forests
```

```
####################################################################
# Boosting Algorithm
#Stumps via Discrete
stump <- rpart.control(maxdepth=1, cp=-1, minsplit=0, xval=0) #No cross val-
idation, only split once
boost1 <- ada(Class ~., data = cardTrain, iter = 1000, type="discrete",
              control = stump) #Build the 6.3GB model
boost1.5 <- ada(Class ~., data = cardTrain, iter = 1000, type="gentle",
              control = stump) #Compare two types of stumps
```

```
boost1; boost1.5 #Look at error

## Call:
## ada(Class ~ ., data = cardTrain, iter = 1000, type = "discrete",
##     control = stump)
##
## Loss: exponential Method: discrete   Iteration: 1000
##
## Final Confusion Matrix for Data:
##           Final Prediction
## True value  Fraud No Fraud
##   Fraud       283      111
##   No Fraud     59   227393
##
## Train Error: 0.001
##
## Out-Of-Bag Error:  0.001  iteration= 414
##
## Additional Estimates of number of iterations:
##
## $train.err1
## [1] 979
##
## $train.kap1
## integer(0)
## Call:
## ada(Class ~ ., data = cardTrain, iter = 1000, type = "gentle",
##     control = stump)
##
## Loss: exponential Method: gentle   Iteration: 1000
##
## Final Confusion Matrix for Data:
##           Final Prediction
## True value  Fraud No Fraud
##   Fraud       285      109
##   No Fraud     40   227412
##
## Train Error: 0.001
##
## Out-Of-Bag Error:  0.001  iteration= 957
##
## Additional Estimates of number of iterations:
##
## $train.err1
## [1] 919
##
## $train.kap1
## integer(0)
```

```
boost.compare <- predict(boost1, newdata = cardTest[,-31])

boost.compare <- factor(boost.compare, c( "Fraud", "No Fraud"))
table(True = cardTest$Class, boost.compare)
```

```
##           boost.compare
## True        Fraud No Fraud
##   Fraud       77       21
##   No Fraud    18    56845
```

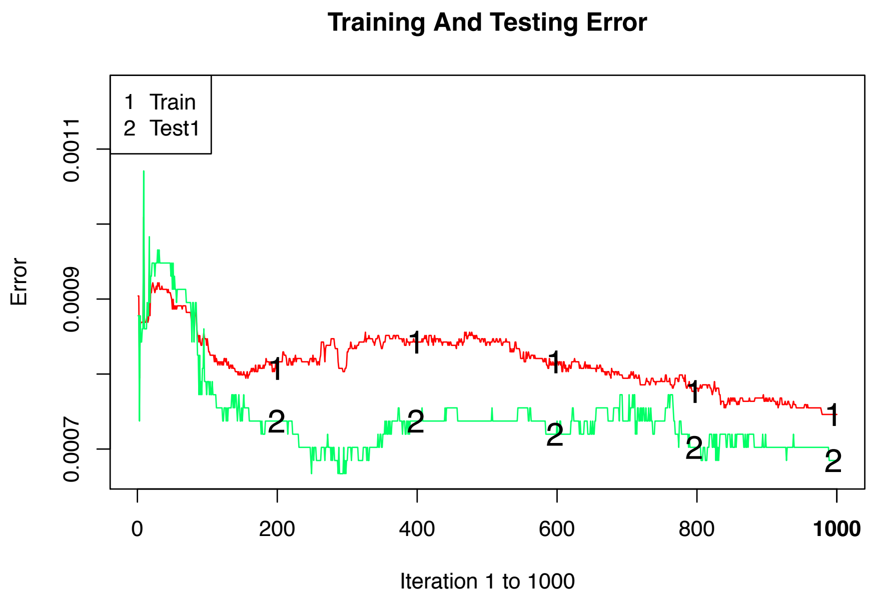
```
boost.compare.prob <- predict(boost1, newdata = cardTest[,-31], type = "prob")
boost.compare.prob.t <- ifelse(boost.compare.prob[,1] > .01, "Fraud", "No Fraud")
boost.compare.prob.t <- factor(boost.compare.prob.t, c("Fraud", "No Fraud"))

table(True = cardTest$Class, boost.compare.prob.t)
```

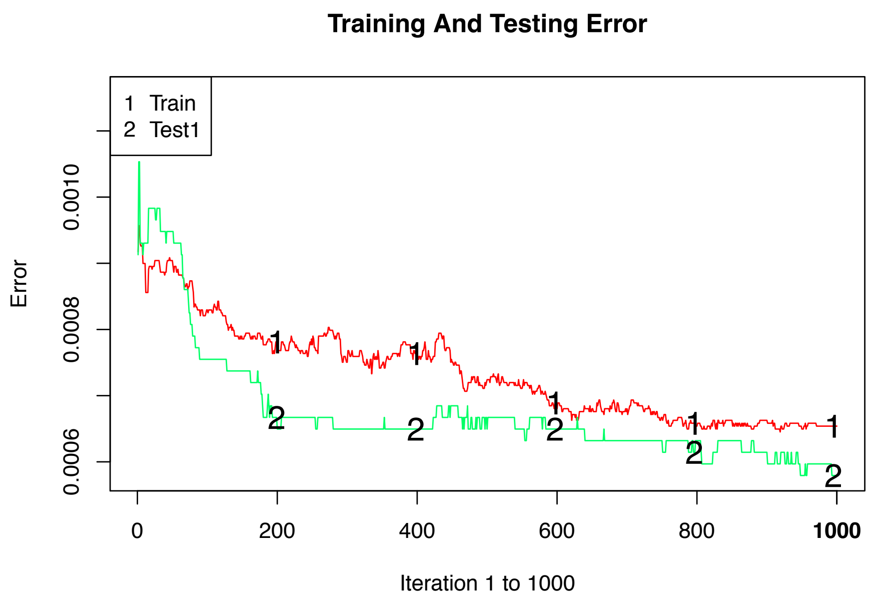
```
##           boost.compare.prob.t
## True        Fraud No Fraud
##   Fraud       90        8
##   No Fraud   216    56647
```

```
newboost1 <- addtest(boost1, cardTest[,-31], cardTest[,31]) #Plot test-
ing and training errors
```

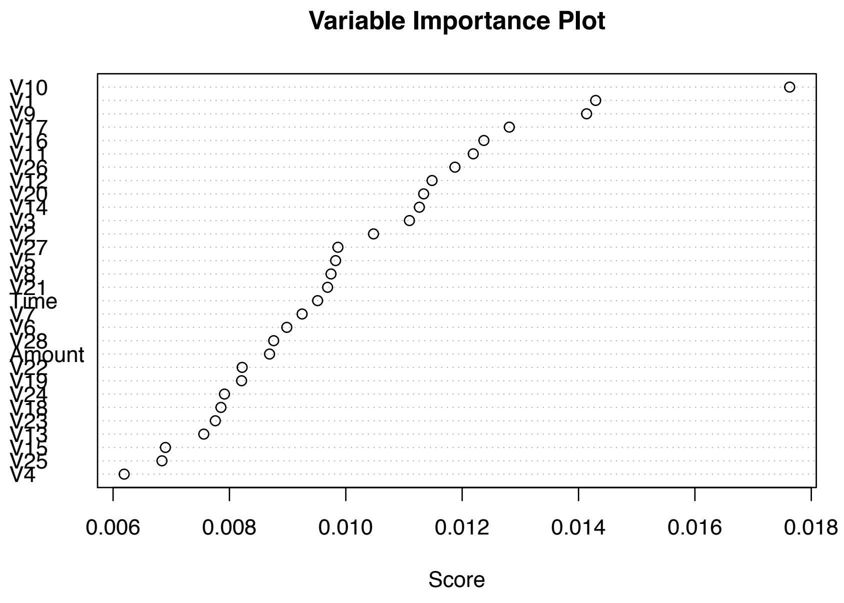
```
newboost <- addtest(boost1.5, cardTest[,-31], cardTest[, 31]) #Look at test-
ing and training error
plot(newboost1, test = T)
```



```
plot(newboost, test = T)
```



```
varplot(boost1) #Var importance
```



```
rm(un.boost1)
rm(boost1.5)


####################################################################
# Discrete ada Boost
boost2 <- ada(Class ~., data = cardTrain, iter = 100, type="discrete") #Dis-
crete Boosting


boost2 #

## Call:
## ada(Class ~ ., data = cardTrain, iter = 100, type = "discrete")
##
## Loss: exponential Method: discrete   Iteration: 100
##
## Final Confusion Matrix for Data:
##              Final Prediction
## True value  Fraud No Fraud
##   Fraud       339       55
##   No Fraud      2   227450
##
## Train Error: 0
##
## Out-Of-Bag Error:  0  iteration= 93
##
## Additional Estimates of number of iterations:
##
## $train.err1
## [1] 96
##
## $train.kap1
## integer(0)

boost2.pred <- predict(boost2, newdata = cardTest[,-31])
boost2.pred.prob <- predict(boost2, newdata = cardTest[,-31], type = "prob")

boost2.pred <- factor(boost2.pred, c("Fraud", "No Fraud"))

boost2.pred.prob <- ifelse(boost2.pred.prob[,1] > 0.01, "Fraud", "No Fraud")
boost2.pred.prob <- factor(boost2.pred.prob, c("Fraud", "No Fraud"))

table(True = cardTest$Class, boost2.pred) #Original

##             boost2.pred
## True       Fraud No Fraud
##   Fraud       85       13
##   No Fraud     3    56860
```

```
table(True = cardTest$Class, boost2.pred.prob) #Lower threshold
```

```
##           boost2.pred.prob
## True       Fraud No Fraud
##   Fraud       86       12
##   No Fraud    20    56843
```

```
newboost <- addtest(boost2, cardTest[,-31], cardTest[,31]) #Testing and train-
ing err


####################################################################
# Real ada Boost
boost3 <- ada(Class ~., data = cardTrain, iter = 200, type="real")


####################################################################
# Gentle ada Boost
boost4 <- ada(Class ~., data = cardTrain, iter = 100, type="gentle")


boost4
```

```
## Call:
## ada(Class ~ ., data = cardTrain, iter = 200, type = "gentle")
##
## Loss: exponential Method: gentle   Iteration: 200
##
## Final Confusion Matrix for Data:
##           Final Prediction
## True value  Fraud No Fraud
##   Fraud       340       54
##   No Fraud      3   227449
##
## Train Error: 0
##
## Out-Of-Bag Error:  0  iteration= 182
##
## Additional Estimates of number of iterations:
##
## $train.err1
## [1] 163
```

```
##
## $train.kap1
## integer(0)
```

```
boost4.pred <- predict(boost4, newdata = cardTest[,-31])

boost4.pred <- factor(boost4.pred, c("Fraud", "No Fraud"))
table(True = cardTest$Class, boost4.pred)
```

```
##           boost4.pred
## True       Fraud No Fraud
##   Fraud       83       15
##   No Fraud     4    56859
```

```
newboost <- addtest(boost4, cardTest[,-31], cardTest[,31])


options(warn = oldw) #Reactivate warnings
```