Exam 3

EricLi

April 22, 2018

```
require(randomForest)
```

```
## Loading required package: randomForest
```

```
## randomForest 4.6-14
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
require(rpart)
```

```
## Loading required package: rpart
```

```
require(gbm)
```

```
## Loading required package: gbm
```

```
## Loading required package: survival
```

```
## Loading required package: lattice
```

```
## Loading required package: splines
```

```
## Loading required package: parallel
```

```
## Loaded gbm 2.1.3
```

```
require(xgboost)
```

```
## Loading required package: xgboost
```

```
require(e1071)
```

```
## Loading required package: e1071
```

```
#Load data
load("C:/Users/Eric Li/Documents/School/SMU/3rd Year Second Semester/STAT 630
6 - Data Science/] HW/Exam3/Xtrain.Rdata")
load("C:/Users/Eric Li/Documents/School/SMU/3rd Year Second Semester/STAT 630
6 - Data Science/] HW/Exam3/Xvalidate.Rdata")

load("C:/Users/Eric Li/Documents/School/SMU/3rd Year Second Semester/STAT 630
6 - Data Science/] HW/Exam3/Ytrain.Rdata")
load("C:/Users/Eric Li/Documents/School/SMU/3rd Year Second Semester/STAT 630
6 - Data Science/] HW/Exam3/Yvalidate.Rdata")
```

```r
# function for replacing NAs with medians found from (https://github.com/mlampros/FeatureSelection/blob/master/R/feature_selection.R)
func_replace_NAs = function(data, which_isna) {       # function to replace NAs
    for (i in which_isna) {
      tmp_median = median(data[, i], na.rm = T)
      data[which(is.na(data[, i])), i] = tmp_median
    }
    return(data)
 }

misClass =function(pred.class,true.class,produceOutput=FALSE){
  confusion.mat = table(pred.class,true.class)
  if(produceOutput){
    return(1-sum(diag(confusion.mat))/sum(confusion.mat))
  }
  else{
    print('miss-class')
    print(1-sum(diag(confusion.mat))/sum(confusion.mat))
    print('confusion mat')
    print(confusion.mat)
    print('')
    print('sensitivity or recall')
    sens = confusion.mat[2,2]/(confusion.mat[1,2]+confusion.mat[2,2])
    print(sens)
    print('')
    print('specificity')
    spec = confusion.mat[1,1]/(confusion.mat[1,1]+confusion.mat[2,1])
    print(spec)
    print('')
    print('precision')
    pres = confusion.mat[2,2]/(confusion.mat[2,2]+confusion.mat[2,1])
    print(pres)
    print('')
    print('F1 score')
    f1 = 2 * (pres * sens) / (pres + sens)
    print(f1)
  }
}

checkNumberItersRF = function(ntrees = 5, tolParm = 1, maxIter = 10, verbose = 0){
  ###
  # tolParm: iterations will continue until the percent decrease
  #          is less than tolParm
  ###
  misClass_out   = list()
  totalTrees_out = list()

  n              = nrow(X)
```

```
  votes          = matrix(0,nrow=n,ncol=2)
  totalTrees     = 0
  iterations     = 0
  misClass_old   = 1
  while(iterations < maxIter){
    votes[is.nan(votes)] = 0
    iterations    = iterations + 1
    totalTrees    = totalTrees + ntrees
    if(verbose >= 2){cat('Total trees: ',totalTrees,'\n')}
    out.rf        = randomForest(X, Y,ntree = ntrees)

    oob.times        = out.rf$oob.times
    votes_iterations = out.rf$votes*oob.times
    votes[oob.times>0,] = matrix(votes + votes_iterations,nrow=n)[oob.times>0
,]
    if(min(apply(votes,1,sum)) == 0){next}

    Yhat          = apply(votes,1,which.max) - 1

# apply function (X, 1..2, function).
# X is what you're working on
# 1..2, where 1 is rows and 2 is columns
# function, what you are trying to do.... in this instance you are trying to
max the row

    misClass_new  = misClass(Yhat,Y,produceOutput = TRUE)
    misClass_out[[iterations]]   = misClass_new
    totalTrees_out[[iterations]] = totalTrees
    percentChange = 100*(misClass_new - misClass_old)/misClass_old

    if(verbose >= 1){cat('% change: ',percentChange,'\n')}
    if(percentChange > -tolParm){break}
    misClass_old = misClass_new
  }
  if(iterations == maxIter){
    stop("too many iterations, try a larger ntrees or maxIter value")
  }
  return(list('misClass' = unlist(misClass_out),
              'totalTree' = unlist(totalTrees_out)))
}
```

Feature Selection and Visualizations for randomForest, bagging, pruned tree, adaboost, and logistic boost:

We use all of the variables when conducting these procedures for feature selection and cross validation purposes.

```r
#Remove NA values of V255 and V256
NAindex.1 = which(is.na(Xtrain$V255))
NAindex.2 = which(is.na(Xtrain$V256))
NAindex = c(NAindex.1,NAindex.2)

Xclean.train = Xtrain[-NAindex,]

# Impute missing data with median values (from https://github.com/mlampros/FeatureSelection/blob/master/R/feature_selection.R)
isna = as.vector(Matrix::colSums(is.na(Xclean.train)))

if (sum(isna) > 0) {
  Xclean.train = func_replace_NAs(Xclean.train, which(isna > 0))
}


X = model.matrix(~0+.+V255+V256,Xclean.train,contrasts.arg = lapply(Xclean.train[,255:256],contrasts,contrasts=FALSE))

Y = Ytrain[-NAindex]
Y = factor(Y,level=c("not fraud","fraud"))

#randomForest
set.seed(42)
rf.out = checkNumberItersRF(ntrees=5,tolParm=1,maxIter=12,verbose=0)
rf.out
```

```
## $misClass
## [1] 0.2847125 0.2758966 0.2732919
##
## $totalTree
## [1] 20 25 30
```

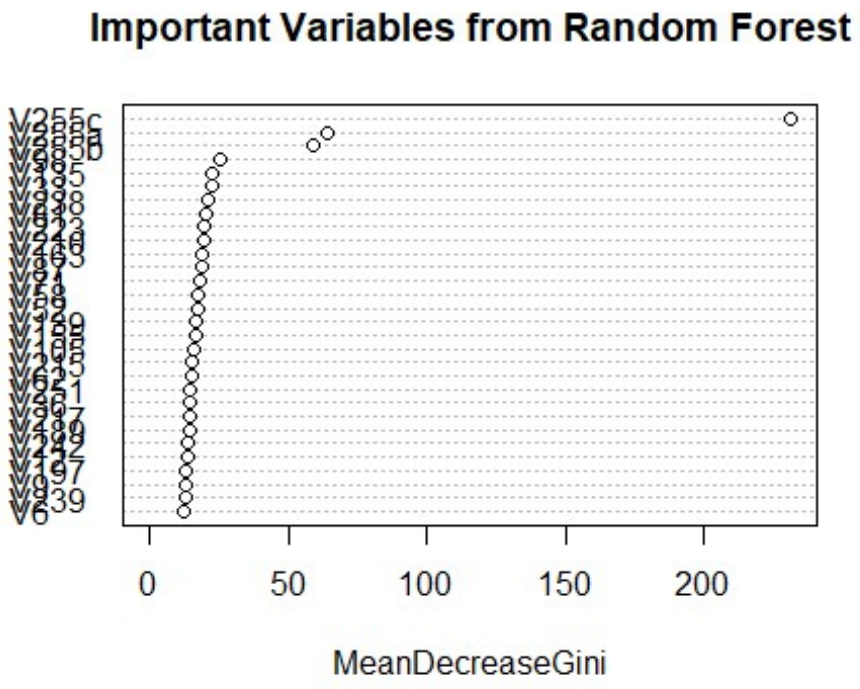
```r
p = lengths(attributes(X)[[2]])[2]
ntree = max(rf.out[[2]])

# mtree = sqrt(p) for classification
out.rf2 = randomForest(X, Y,proximity=TRUE,ntree = ntree,mtree=round(sqrt(p)))
out.rf2
```

```
##
## Call:
##  randomForest(x = X, y = Y, ntree = ntree, proximity = TRUE, mtree = round(sqrt(p)))
##                Type of random forest: classification
```

```
##                       Number of trees: 30
## No. of variables tried at each split: 16
##
##         OOB estimate of  error rate: 26.03%
## Confusion matrix:
##           not fraud fraud class.error
## not fraud      2991   430   0.1256942
## fraud           869   701   0.5535032
```

We use randomForest for feature/procedure selection. The cross validation for OOB misclassification is ~26%. Our estimate of risk is the 0-1 loss function (misclassification rate).

```
varImpPlot(out.rf2,main="Important Variables from Random Forest")
```



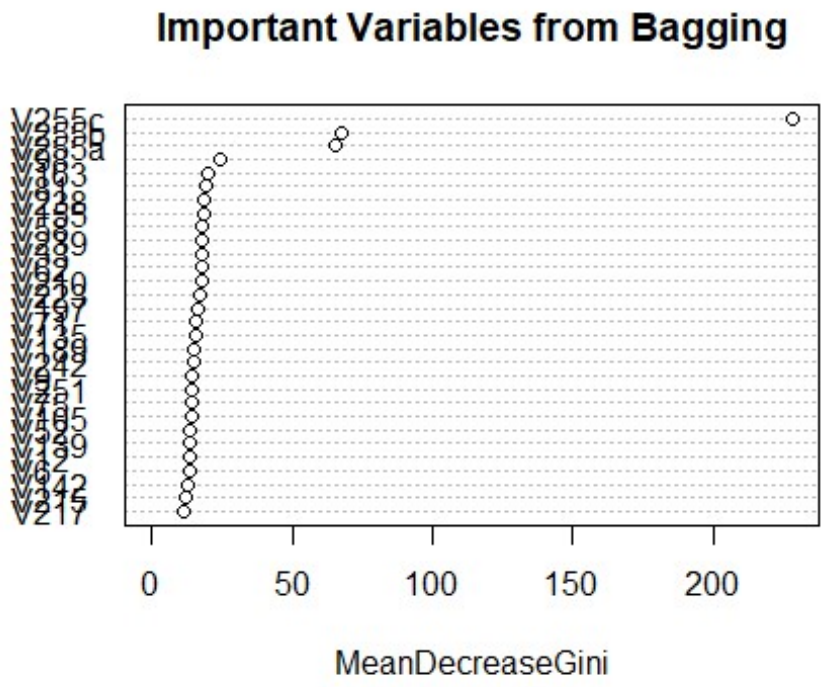
The Variable Importance Plot shows the average reduction in the Gini index by splitting on that feature (over all trees that were bagged). The importance of the feature (in randomForest and bagging) is defined as the greatest average decrease in the Gini index (in descending order).

```
out.bag = randomForest(X, Y,proximity=TRUE,ntree = ntree,mtree=p) # mtree = p
out.bag

##
## Call:
##  randomForest(x = X, y = Y, ntree = ntree, proximity = TRUE, mtree = p)
##                Type of random forest: classification
```

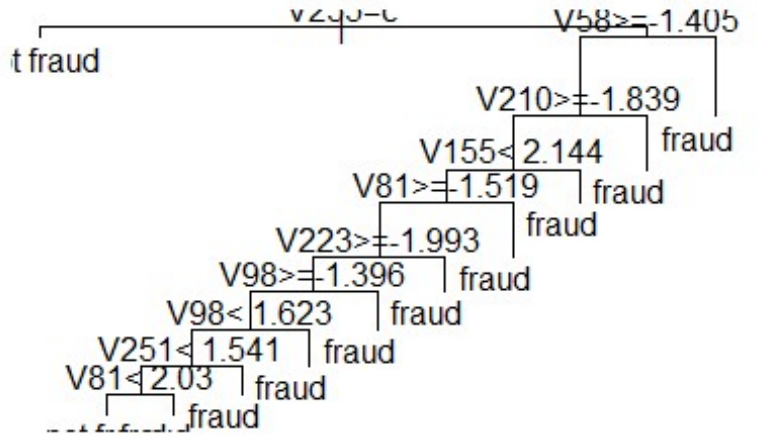
```
##                       Number of trees: 30
## No. of variables tried at each split: 16
##
##         OOB estimate of  error rate: 27.19%
## Confusion matrix:
##           not fraud fraud class.error
## not fraud      2974   447   0.1306635
## fraud           910   660   0.5796178
```

We also use bagging for feature/procedure selection. The cv for OOB misclassification in bagging is ~27%. Our estimate of risk is the 0-1 loss function (misclassification rate).

```
varImpPlot(out.bag,main="Important Variables from Bagging")
```




```
#pruned tree
unprune.tree = rpart(Y~.,data=Xclean.train,method="class",parms=list(split="g
ini"))
bestcp = unprune.tree$cptable[which.min(unprune.tree$cptable[,"xerror"]),"CP"
]
prune.tree = prune(unprune.tree,cp=bestcp)
plot(prune.tree);text(prune.tree)
```



Based on: https://cran.r-project.org/web/packages/rpart/vignettes/longintro.pdf, we make a pruned tree based on best cp. Important features are those determined to be split nodes. We observed that for our setseed, the pruned and unpruned trees are the same.

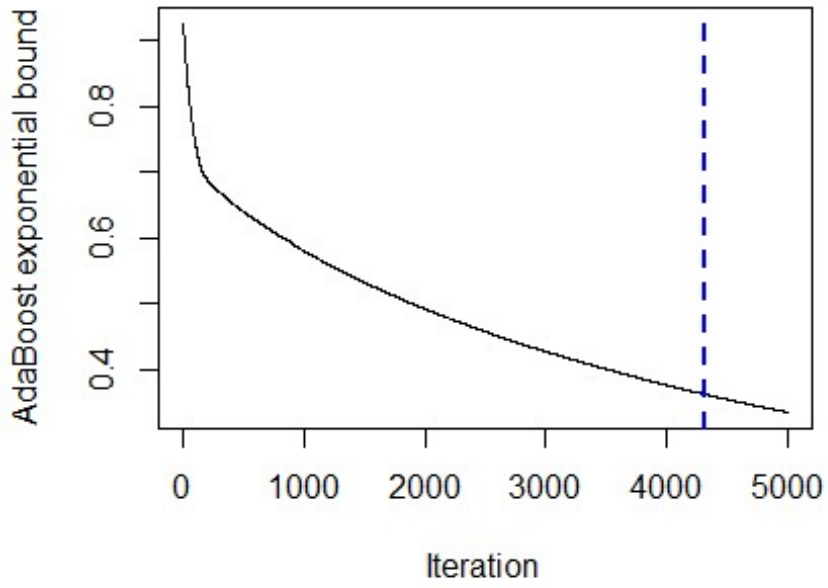
```
printcp(prune.tree)
```

```
##
## Classification tree:
## rpart(formula = Y ~ ., data = Xclean.train, method = "class",
##     parms = list(split = "gini"))
##
## Variables actually used in tree construction:
## [1] V155 V210 V223 V251 V255 V58  V81  V98
##
## Root node error: 1570/4991 = 0.31457
##
## n= 4991
##
##         CP nsplit rel error  xerror     xstd
## 1 0.030892      0   1.00000 1.00000 0.020895
## 2 0.028662      5   0.82803 0.92611 0.020446
## 3 0.021656      8   0.74204 0.90573 0.020311
## 4 0.015924      9   0.72038 0.89108 0.020211
## 5 0.012739     10   0.70446 0.87261 0.020081
```

The misclassification error is 0.31457 * 0.87261 = ~28%, which is the same as the unpruned tree (the 0-1 loss function).

```
#adaboost
Yclean.adatrain = (Y=="fraud")
ada.boost = gbm(Yclean.adatrain~.,data=Xclean.train,distribution="adaboost",n
.trees=5000,shrinkage=0.01,interaction.depth=2,n.cores=3,cv.folds=5)
best.iter = gbm.perf(ada.boost,method="OOB")
```

```
## Warning in gbm.perf(ada.boost, method = "OOB"): OOB generally
## underestimates the optimal number of iterations although predictive
## performance is reasonably competitive. Using cv.folds>0 when calling gbm
## usually results in improved predictive performance.
```



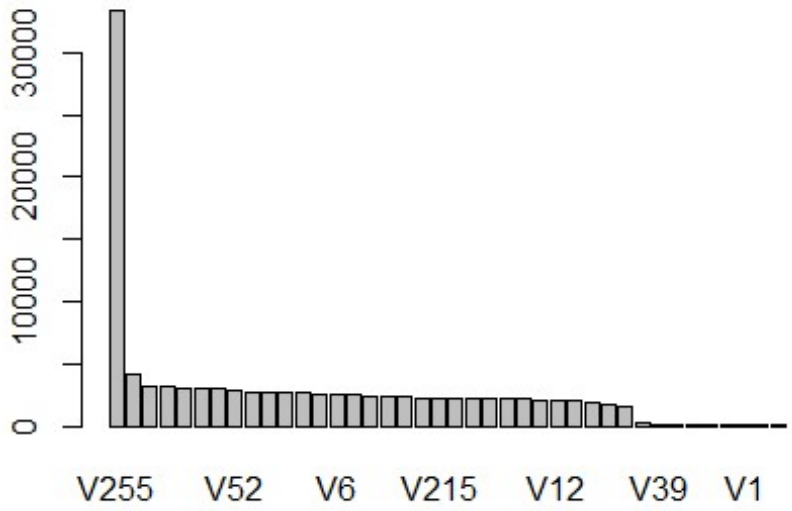

```
min(ada.boost$cv.error)
```

```
## [1] 0.4267269
```

From (https://cran.r-project.org/web/packages/gbm/gbm.pdf): We perform adaptive boosting with an interaction depth of 3 and a learning rate of 0.01 to iteratively fit a classifier on the reweighted training data such that the misclassified observations are upweighted. We arbitrarily chose k = 5 and number of trees to be 5000.

Under 5-fold cross-validation, we get a risk estimate of ~43%.

```
barplot(relative.influence(ada.boost,n.trees = best.iter,sort. = TRUE)[1:40])
```



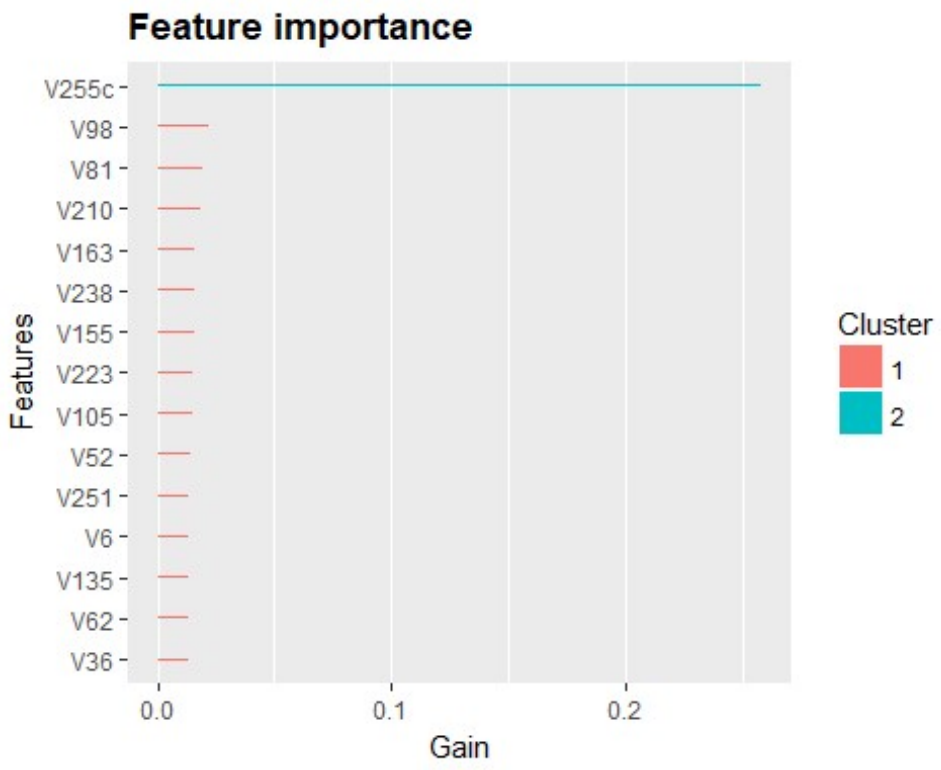
Again, from (https://cran.r-project.org/web/packages/gbm/gbm.pdf) and from (http://avesbiodiv.mncn.csic.es/estadistica/bt1.pdf): We show the Variable Importance Plot (based on estimations of relative influence). Relative influence is based on the number of times a feature was selected for splitting, weighted by the squared improvement in the model (by 0-1 loss function) and average over all the trees.

```
#logistic boost
X = model.matrix(~0+.+V255+V256,Xclean.train,contrasts.arg = lapply(Xclean.tr
ain[,255:256],contrasts,contrasts=FALSE))

boost.train = xgb.DMatrix(X,label=Yclean.adatrain)
logistic.boost = xgb.cv(data=boost.train,nfold=5,nrounds=1000,objective="bina
ry:logistic",metrics=list("error"),showsd=FALSE,verbose=FALSE)
best.nrounds = which(grepl(min(logistic.boost$test.error.mean),logistic.boost
$test.error.mean))

min(logistic.boost$test.error.mean)

## [1] 0.150471
```

Based on (https://xgboost.readthedocs.io/en/latest/): We perform logistic boosting. We arbitrarily chose k = 5 and number of iterations to be 1000.

The cross validation misclassification rate estimate of boosted logistic regression is ~15% (0-1 loss function)

```
names = unlist(dimnames(X)[2],use.names=FALSE)

best.logistic.boost = xgboost(data = X,label=Yclean.adatrain,objective="binar
y:logistic",nround=best.nrounds,verbose=0)

important.logistic.boost = xgb.importance(names,model=best.logistic.boost)
xgb.plot.importance(importance_matrix = important.logistic.boost[1:15])
```



The Variable Importance Plot shows the Gain – the reduction in misclassification error (0-1 loss function) attributed to the split at the feature.

Validation for procedure selection

```
# Again, Impute missing data with median values (from https://github.com/mlam
pros/FeatureSelection/blob/master/R/feature_selection.R)
isna2 = as.vector(Matrix::colSums(is.na(Xvalidate)))

if (sum(isna2) > 0) {
  Xvalidate = func_replace_NAs(Xvalidate, which(isna2 > 0))
}

use.Xvalidate = model.matrix(~0+.+V255+V256,Xvalidate,contrasts.arg = lapply(
Xvalidate[,255:256],contrasts,contrasts=FALSE))
```

```
use.Yvalidate = factor(Yvalidate,level=c("not fraud","fraud"))

#randomForest predictions (mtry = default)
rf.p      = predict(out.rf2,use.Xvalidate,type="class")

#bagging predictions (mtry = p)
bag.p       = predict(out.bag,use.Xvalidate,type='class')

#pruned tree predictions
prune.p = predict(prune.tree,Xvalidate,type='class')

#adaboost predictions
ada.boost.p = round(predict(ada.boost,Xvalidate,n.trees=best.iter,type="respo
nse"))
ada.boost.p = factor(ifelse(ada.boost.p==1,"fraud","not fraud"),levels=c("not
fraud","fraud"))

#logistic boost predictions
logistic.boost.p = round(predict(best.logistic.boost,use.Xvalidate))
logistic.boost.p = factor(ifelse(logistic.boost.p==1,"fraud","not fraud"),lev
els=c("not fraud","fraud"))
```

There are 209 empty observations in the Xvalidate data set. Again, we compute the missing continuous observations with the median of the data set.

For our ada.boost and logistic boost predictions, our decision boundary was the 0.5 mark. For predictions that are greater than or equal to 0.5, they will be labeled as “fraud.” For predictions that are less than 0.5, they will be labeled as “not fraud.”

```
misClass(rf.p,use.Yvalidate)
```

```
## [1] "miss-class"
## [1] 0.236
## [1] "confusion mat"
##            true.class
## pred.class  not fraud fraud
##   not fraud       635   177
##   fraud            59   129
## [1] ""
## [1] "sensitivity or recall"
## [1] 0.4215686
## [1] ""
## [1] "specificity"
## [1] 0.9149856
## [1] ""
## [1] "precision"
## [1] 0.6861702
## [1] ""
```

```
## [1] "F1 score"
## [1] 0.5222672
```

```
misClass(bag.p,use.Yvalidate)
```

```
## [1] "miss-class"
## [1] 0.255
## [1] "confusion mat"
##            true.class
## pred.class  not fraud fraud
##   not fraud       628   189
##   fraud            66   117
## [1] ""
## [1] "sensitivity or recall"
## [1] 0.3823529
## [1] ""
## [1] "specificity"
## [1] 0.9048991
## [1] ""
## [1] "precision"
## [1] 0.6393443
## [1] ""
## [1] "F1 score"
## [1] 0.4785276
```

```
misClass(prune.p,use.Yvalidate)
```

```
## [1] "miss-class"
## [1] 0.256
## [1] "confusion mat"
##            true.class
## pred.class  not fraud fraud
##   not fraud       598   160
##   fraud            96   146
## [1] ""
## [1] "sensitivity or recall"
## [1] 0.4771242
## [1] ""
## [1] "specificity"
## [1] 0.8616715
## [1] ""
## [1] "precision"
## [1] 0.6033058
## [1] ""
## [1] "F1 score"
## [1] 0.5328467
```

```
misClass(logistic.boost.p,use.Yvalidate)
```

```
## [1] "miss-class"
## [1] 0.152
```

```
## [1] "confusion mat"
##            true.class
## pred.class  not fraud fraud
##   not fraud       630    88
##   fraud            64   218
## [1] ""
## [1] "sensitivity or recall"
## [1] 0.7124183
## [1] ""
## [1] "specificity"
## [1] 0.907781
## [1] ""
## [1] "precision"
## [1] 0.7730496
## [1] ""
## [1] "F1 score"
## [1] 0.7414966
```

```
misClass(ada.boost.p,use.Yvalidate)
```

```
## [1] "miss-class"
## [1] 0.092
## [1] "confusion mat"
##            true.class
## pred.class  not fraud fraud
##   not fraud       668    66
##   fraud            26   240
## [1] ""
## [1] "sensitivity or recall"
## [1] 0.7843137
## [1] ""
## [1] "specificity"
## [1] 0.962536
## [1] ""
## [1] "precision"
## [1] 0.9022556
## [1] ""
## [1] "F1 score"
## [1] 0.8391608
```

The adapative boosting algorithm had the highest sensitivity, recall, precision, and f1score compared with other procedures. It also had the lowest misclassification rate across all the procedures.

Thus, it appears as though the classifier has high bias and low variance.

```
load("C:/Users/Eric Li/Documents/School/SMU/3rd Year Second Semester/STAT 630
6 - Data Science/] HW/Exam3/Xtest.Rdata")

predictions = round(predict(ada.boost,Xtest,n.trees=best.iter,type="response"
))
```

```
predictions = factor(ifelse(predictions==1,"fraud","not fraud"),levels=c("not
fraud","fraud"))

Yhat = data.frame('Yhat' = predictions)

#write.table
yourName="EricLi"

fName = paste(c(yourName,'_Predictions.txt'),collapse='')
write.table(t(Yhat),file=fName,row.names=FALSE,col.names=FALSE)
```

There are 230 missing data points in the test data set.

We note from (https://www.rdocumentation.org/packages/gbm/versions/2.1.1/topics/gbm.object) that gbm builds trees with three splits, left node, right node, and missing node. Thus, the algorithm treats missing values as a separate group, so the missing data points will not be an issue for the chosen algorithm and procedure.