Ensemble Methods

Aditi Chaudhari and Abigail Solomon

2022-10-21

Ensemble Methods

We will first perform machine learning with decision tree as a baseline, and then we will use Random Forest, XGBoost, and AdaBoost as a point of comparison.

First, let's read our data into a data frame. The data is from https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud. We will attempt to classify if a credit card is fraudulent or not using a variety of variables.

```
df <- read.csv("creditcard.csv")
```

We need to convert the Class attribute to a factor with two levels.

```
df$Class <- as.factor(df$Class)
```

We also want to change the size of the data frame to 10,000 observations instead of the original 284,807 observations. This is done because my laptop cannot handle a data set that large.

First, we create a data frame that only contains the data associated with a fraudulent credit card. There were 492 fraudulent credit cards out of the original 284,807 observations. Next, we create a data frame that only contains the data associated with non-fraudulent credit cards. 9508 observations are randomly chosen from the newly-created non-fraudulent credit cards data frame. The fraudulent credit cards data frame is merged with the non-fraudulent one and then the resulting data frame is shuffled. We now have 10,000 observations of credit card data where 492 out of the 10,000 are fraudulent.

```
library(dplyr)
```

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

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

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

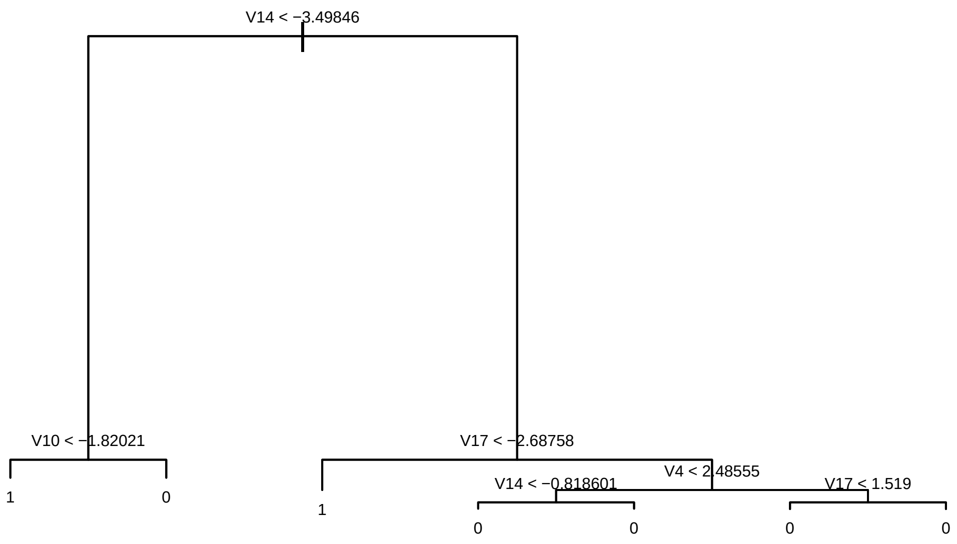
```
frauds <- subset(df, df$Class==1)
nonfrauds <- subset(df, df$Class==0)

set.seed(123)
ind <- sample(1:nrow(nonfrauds), 10000 - nrow(frauds), replace=FALSE)
nonfrauds <-  nonfrauds[ind,]

df <- rbind(frauds, nonfrauds)
df <- df[sample(1:nrow(df)),]
```

Now we can use different ensemble methods to explore the data. We will compare the ensemble methods using accuracy, the Matthews Correlation Coefficient (mcc), and the time it takes to run the algorithm.

Decision Trees

First, let's create a decision tree with the entire data set.

```
library(tree)
tree_creditcard <- tree(Class~., data=df)
plot(tree_creditcard)
text(tree_creditcard, cex=0.5, pretty=0)
```



Here is a summary of the model we created. Our model has a misclassification error rate of 0.91%. which is very good.

```
summary(tree_creditcard)
```

```
##
## Classification tree:
## tree(formula = Class ~ ., data = df)
## Variables actually used in tree construction:
## [1] "V14" "V10" "V17" "V4"
## Number of terminal nodes:  7
## Residual mean deviance:  0.07812 = 780.7 / 9993
## Misclassification error rate: 0.0091 = 91 / 10000
```

Next, we split the original data frame into a training data frame and a testing data frame. We created a new decision tree with the training data, and then used the pred() function to evaluate the model with the testing data. Our model misidentified 23 credit cards that were non-fraudulent to be fraudulent and 5 credit cards that were fraudulent to be non-fraudulent. Aside from these misclassifications, our model is fairly accurate with an accuracy rate of 98.9% and an mcc score of 88.3%. The algorithm ran fairly fast.

```r
set.seed(1111)
i <- sample(1:nrow(df), nrow(df) * 0.75, replace=FALSE)
train <- df[i,]
test <- df[-i,]
start_time <-Sys.time()
tree_cc2 <- tree(Class~., data=train)
end_time <- Sys.time()
pred_tree <- predict(tree_cc2, newdata=test, type="class")
table(pred_tree, test$Class)
```

```
##
## pred_tree    0    1
##         0 2363   23
##         1    5  109
```

```r
library(mltools)
acc_tree <- mean(pred_tree==test$Class)
mcc_tree <- mcc(pred_tree, test$Class)
print(paste("accuracy for decision tree: ", acc_tree))
```

```
## [1] "accuracy for decision tree:  0.9888"
```

```r
print(paste("mcc for decision tree: ", mcc_tree))
```

```
## [1] "mcc for decision tree:  0.88293922299399"
```

```r
total_time_tree <- end_time - start_time
print(paste("time taking for Decision Tree algorithm: ", total_time_tree))
```

```
## [1] "time taking for Decision Tree algorithm:  0.808484077453613"
```

Random Forest

Next, let's perform classification using a random forest. The training data set is used to train the random forest model, and then the testing data set is used to evaluate how accurate the model is. Running the random forest algorithm shows that 500 trees were created, trying 5 variables at each split. our model is quite accurate with an accuracy rate of 99.1% and an mcc score of 90.4%. Although, the random forest algorithm took longer to run than other ensemble methods.

```r
library(randomForest)
```

```
## randomForest 4.7-1.1

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

##
## Attaching package: 'randomForest'

## The following object is masked from 'package:dplyr':
##
##     combine
```

```r
set.seed(1234)
start_time <-Sys.time()
rf <- randomForest(Class~., data=train, importance=TRUE)
end_time <- Sys.time()
```

```
total_time_RF <- end_time - start_time
summary(rf)
```

```
##                 Length Class  Mode
## call                4  -none- call
## type                1  -none- character
## predicted        7500  factor numeric
## err.rate         1500  -none- numeric
## confusion           6  -none- numeric
## votes           15000  matrix numeric
## oob.times        7500  -none- numeric
## classes             2  -none- character
## importance        120  -none- numeric
## importanceSD       90  -none- numeric
## localImportance     0  -none- NULL
## proximity           0  -none- NULL
## ntree               1  -none- numeric
## mtry                1  -none- numeric
## forest             14  -none- list
## y                7500  factor numeric
## test                0  -none- NULL
## inbag               0  -none- NULL
## terms               3  terms  call
```

```
pred_rf <- predict(rf, newdata = test)

acc_rf <- mean(pred_rf==test$Class)
mcc_rf <- mcc(pred_rf, test$Class)

print(paste("accuracy for random forest: ", acc_rf))
```

```
## [1] "accuracy for random forest:  0.9908"
```

```
print(paste("mcc for random forest: ", mcc_rf))
```

```
## [1] "mcc for random forest:  0.904440194540115"
```

```
print(paste("time taking for Random Forest algorithm: ", total_time_RF))
```

```
## [1] "time taking for Random Forest algorithm:  38.7954380512238"
```

XGBoost

Next, let's perform classification using the XGBoost algorithm. Our model is accurate with an accuracy rate of 99.1% and a mcc score of 90.4%. It also ran very quickly.

```
library(xgboost)
```

```
##
## Attaching package: 'xgboost'

## The following object is masked from 'package:dplyr':
##
##     slice
```

```
train_label <- ifelse(train$Class==1,1,0)
test_label <- ifelse(test$Class==1,1,0)
```

```
train_matrix <- data.matrix(train[1:30])
test_matrix <- data.matrix(test[1:30])


start_time <- Sys.time()
XGBoostModel <- xgboost(data=train_matrix, label = train_label, nrounds=100, objective='binary:logistic
```

```
## [1]  train-logloss:0.444423
## [2]  train-logloss:0.306827
## [3]  train-logloss:0.220202
## [4]  train-logloss:0.162770
## [5]  train-logloss:0.122497
## [6]  train-logloss:0.094103
## [7]  train-logloss:0.073697
## [8]  train-logloss:0.058630
## [9]  train-logloss:0.047448
## [10] train-logloss:0.038691
## [11] train-logloss:0.031657
## [12] train-logloss:0.026337
## [13] train-logloss:0.022043
## [14] train-logloss:0.018745
## [15] train-logloss:0.016032
## [16] train-logloss:0.013896
## [17] train-logloss:0.012040
## [18] train-logloss:0.010594
## [19] train-logloss:0.009154
## [20] train-logloss:0.008027
## [21] train-logloss:0.007190
## [22] train-logloss:0.006419
## [23] train-logloss:0.005729
## [24] train-logloss:0.005235
## [25] train-logloss:0.004830
## [26] train-logloss:0.004450
## [27] train-logloss:0.004111
## [28] train-logloss:0.003786
## [29] train-logloss:0.003547
## [30] train-logloss:0.003323
## [31] train-logloss:0.003104
## [32] train-logloss:0.002911
## [33] train-logloss:0.002748
## [34] train-logloss:0.002597
## [35] train-logloss:0.002459
## [36] train-logloss:0.002349
## [37] train-logloss:0.002241
## [38] train-logloss:0.002163
## [39] train-logloss:0.002077
## [40] train-logloss:0.001994
## [41] train-logloss:0.001910
## [42] train-logloss:0.001843
## [43] train-logloss:0.001787
## [44] train-logloss:0.001718
## [45] train-logloss:0.001667
## [46] train-logloss:0.001602
## [47] train-logloss:0.001554
## [48] train-logloss:0.001502
## [49] train-logloss:0.001460
## [50] train-logloss:0.001418
## [51] train-logloss:0.001388
## [52] train-logloss:0.001353
## [53] train-logloss:0.001321
## [54] train-logloss:0.001294
## [55] train-logloss:0.001268
## [56] train-logloss:0.001242
## [57] train-logloss:0.001221
## [58] train-logloss:0.001200
## [59] train-logloss:0.001179
## [60] train-logloss:0.001159
## [61] train-logloss:0.001138
## [62] train-logloss:0.001115
## [63] train-logloss:0.001097
## [64] train-logloss:0.001078
## [65] train-logloss:0.001064
## [66] train-logloss:0.001050
## [67] train-logloss:0.001032
## [68] train-logloss:0.001015
## [69] train-logloss:0.001002
## [70] train-logloss:0.000987
## [71] train-logloss:0.000972
## [72] train-logloss:0.000960
## [73] train-logloss:0.000947
## [74] train-logloss:0.000934
## [75] train-logloss:0.000925
## [76] train-logloss:0.000911
## [77] train-logloss:0.000902
## [78] train-logloss:0.000890
## [79] train-logloss:0.000881
## [80] train-logloss:0.000874
## [81] train-logloss:0.000866
## [82] train-logloss:0.000858
## [83] train-logloss:0.000846
## [84] train-logloss:0.000838
## [85] train-logloss:0.000828
## [86] train-logloss:0.000820
## [87] train-logloss:0.000812
## [88] train-logloss:0.000806
## [89] train-logloss:0.000798
## [90] train-logloss:0.000792
## [91] train-logloss:0.000786
## [92] train-logloss:0.000778
## [93] train-logloss:0.000772
## [94] train-logloss:0.000768
## [95] train-logloss:0.000761
## [96] train-logloss:0.000756
## [97] train-logloss:0.000752
## [98] train-logloss:0.000747
## [99] train-logloss:0.000743
## [100]    train-logloss:0.000737
```

```
end_time = Sys.time()
XG_time <- end_time - start_time

probs <- predict(XGBoostModel, test_matrix)
XGpred <- ifelse(probs>0.5, 1, 0)
```

```
acc_xg <- mean(XGpred==test_label)
mcc_xg <- mcc(XGpred, test_label)
print(paste("accuracy for xgboost=", acc_xg))
```

```
## [1] "accuracy for xgboost= 0.9908"
```

```
print(paste("mcc for xgboost=", mcc_xg))
```

```
## [1] "mcc for xgboost= 0.904440194540115"
```

```
print(paste("Time taking for XGBoost algorithm: ", XG_time))
```

```
## [1] "Time taking for XGBoost algorithm:  3.04436707496643"
```

AdaBoost

Finally, let's perform classification with AdaBoost. Our model is very accurate with an accuracy rate of 99.1% and a mcc score of 90.1%. However, it takes longer to run than the XGBoost algorithm.

```
library(adabag)
```

```
## Loading required package: rpart

## Loading required package: caret

## Loading required package: ggplot2

##
## Attaching package: 'ggplot2'

## The following object is masked from 'package:randomForest':
##
##     margin

## Loading required package: lattice

## Loading required package: foreach

## Loading required package: doParallel

## Loading required package: iterators

## Loading required package: parallel
```

```
start_time <- Sys.time()
adab <-boosting(Class~., data=train, boos=TRUE, mfinal=20, coeflearn = 'Breiman')
end_time = Sys.time()
summary(adab)
```

```
##            Length Class   Mode
## formula        3  formula call
## trees         20  -none-  list
## weights       20  -none-  numeric
## votes      15000  -none-  numeric
## prob       15000  -none-  numeric
## class       7500  -none-  character
## importance    30  -none-  numeric
## terms          3  terms   call
## call           6  -none-  call
```

```
pred_ada <- predict(adab, newdata=test, type="response")

acc_ada <- mean(pred_ada$class == test$Class)
mcc_ada <- mcc(factor(pred_ada$class), test$Class)

print(paste("accuracy= for adaboost", acc_ada))
```

```
## [1] "accuracy= for adaboost 0.9912"
```

```
print(paste("mcc for adaboost=", mcc_ada))
```

```
## [1] "mcc for adaboost= 0.90866069407386"
```

```
ada_time <- end_time - start_time
print(paste("Time taking for AdaBoost algorithm: ", ada_time))
```

```
## [1] "Time taking for AdaBoost algorithm:  20.2681441307068"
```