Ensemble

Meinhard Capucao, Khang Thai

Introduction

In this project, we want to see if we can classify credit card fraud based on 28 numeric input variables. The important column here is the **'class'** column. If the class is 1, the credit card transaction is detected as fraud.

The dataset contains only numerical input variables because of confidentiality issues. The only features not transformed are 'Time' and 'Amount'.

Here, we will use different ensemble techniques. First, we will use decision tree as a baseline. We can keep the training/test data split from the last notebook.

Installing Packages and Reading Data

First, we are going to install the necessary libraries and packages. Then, we will put the credit card data into a dataframe named 'creditcard'.

```
install.packages("tidyverse",repos = "http://cran.us.r-project.org")
```

```
## Installing package into 'C:/Users/meinc/AppData/Local/R/win-library/4.2'
## (as 'lib' is unspecified)
```

```
## package 'tidyverse' successfully unpacked and MD5 sums checked
##
## The downloaded binary packages are in
##  C:\Users\meinc\AppData\Local\Temp\Rtmp8GfoZk\downloaded_packages
```

```
install.packages("corrr",repos = "http://cran.us.r-project.org")
```

```
## Installing package into 'C:/Users/meinc/AppData/Local/R/win-library/4.2'
## (as 'lib' is unspecified)
```

```
## package 'corrr' successfully unpacked and MD5 sums checked
##
## The downloaded binary packages are in
##  C:\Users\meinc\AppData\Local\Temp\Rtmp8GfoZk\downloaded_packages
```

```
install.packages("e1071",repos = "http://cran.us.r-project.org")
```

```
## Installing package into 'C:/Users/meinc/AppData/Local/R/win-library/4.2'
## (as 'lib' is unspecified)
```

```
## package 'e1071' successfully unpacked and MD5 sums checked
```

```
## Warning: cannot remove prior installation of package 'e1071'
```

```
## Warning in file.copy(savedcopy, lib, recursive = TRUE): problem copying C:
## \Users\meinc\AppData\Local\R\win-library\4.2\00LOCK\e1071\libs\x64\e1071.dll
## to C:\Users\meinc\AppData\Local\R\win-library\4.2\e1071\libs\x64\e1071.dll:
## Permission denied
```

```
## Warning: restored 'e1071'
```

```
##
## The downloaded binary packages are in
##  C:\Users\meinc\AppData\Local\Temp\Rtmp8GfoZk\downloaded_packages
```

```
install.packages("rpart",repos = "http://cran.us.r-project.org")
```

```
## Installing package into 'C:/Users/meinc/AppData/Local/R/win-library/4.2'
## (as 'lib' is unspecified)
```

```
##
##   There is a binary version available but the source version is later:
##       binary source needs_compilation
## rpart 4.1.16 4.1.19              TRUE
```

```
## installing the source package 'rpart'
```

```
## Warning in install.packages("rpart", repos = "http://cran.us.r-project.org"):
## installation of package 'rpart' had non-zero exit status
```

```
install.packages("tree",repos = "http://cran.us.r-project.org")
```

```
## Installing package into 'C:/Users/meinc/AppData/Local/R/win-library/4.2'
## (as 'lib' is unspecified)
```

```
## package 'tree' successfully unpacked and MD5 sums checked
```

```
## Warning: cannot remove prior installation of package 'tree'
```

```
## Warning in file.copy(savedcopy, lib, recursive = TRUE): problem copying C:
## \Users\meinc\AppData\Local\R\win-library\4.2\00LOCK\tree\libs\x64\tree.dll to C:
## \Users\meinc\AppData\Local\R\win-library\4.2\tree\libs\x64\tree.dll: Permission
## denied
```

```
## Warning: restored 'tree'
```

```
##
## The downloaded binary packages are in
##  C:\Users\meinc\AppData\Local\Temp\Rtmp8GfoZk\downloaded_packages
```

```
install.packages("randomForest",repos = "http://cran.us.r-project.org")
```

```
## Installing package into 'C:/Users/meinc/AppData/Local/R/win-library/4.2'
## (as 'lib' is unspecified)
```

```
## package 'randomForest' successfully unpacked and MD5 sums checked
```

```
## Warning: cannot remove prior installation of package 'randomForest'
```

```
## Warning in file.copy(savedcopy, lib, recursive = TRUE):
## problem copying C:\Users\meinc\AppData\Local\R\win-
## library\4.2\00LOCK\randomForest\libs\x64\randomForest.dll
## to C:\Users\meinc\AppData\Local\R\win-
## library\4.2\randomForest\libs\x64\randomForest.dll: Permission denied
```

```
## Warning: restored 'randomForest'
```

```
##
## The downloaded binary packages are in
##  C:\Users\meinc\AppData\Local\Temp\Rtmp8GfoZk\downloaded_packages
```

```
install.packages("xgboost",repos = "http://cran.us.r-project.org")
```

```
## Installing package into 'C:/Users/meinc/AppData/Local/R/win-library/4.2'
## (as 'lib' is unspecified)
```

```
## package 'xgboost' successfully unpacked and MD5 sums checked
```

```
## Warning: cannot remove prior installation of package 'xgboost'
```

```
## Warning in file.copy(savedcopy, lib, recursive = TRUE): problem copying C:
## \Users\meinc\AppData\Local\R\win-library\4.2\00LOCK\xgboost\libs\x64\xgboost.dll
## to C:\Users\meinc\AppData\Local\R\win-library\4.2\xgboost\libs\x64\xgboost.dll:
## Permission denied
```

```
## Warning: restored 'xgboost'
```

```
##
## The downloaded binary packages are in
##  C:\Users\meinc\AppData\Local\Temp\Rtmp8GfoZk\downloaded_packages
```

```
install.packages("SuperLearner",repos = "http://cran.us.r-project.org")
```

```
## Installing package into 'C:/Users/meinc/AppData/Local/R/win-library/4.2'
## (as 'lib' is unspecified)
```

```
## package 'SuperLearner' successfully unpacked and MD5 sums checked
##
## The downloaded binary packages are in
##  C:\Users\meinc\AppData\Local\Temp\Rtmp8GfoZk\downloaded_packages
```

```
library(SuperLearner)
```

```
## Loading required package: nnls
```

```
## Loading required package: gam
```

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

```
## Loading required package: foreach
```

```
## Loaded gam 1.20.2
```

```
## Super Learner
```

```
## Version: 2.0-28
```

```
## Package created on 2021-05-04
```

```
library(randomForest)
```

```
## randomForest 4.7-1.1
```

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

```
library(rpart)
library(e1071)
library(corrr)
library(tidyverse)
```

```
## ── Attaching packages
## ───────────────────────────────────────
## tidyverse 1.3.2 ──
```

```
## ✔ ggplot2 3.3.6      ✔ purrr   0.3.4
## ✔ tibble  3.1.8      ✔ dplyr   1.0.10
## ✔ tidyr   1.2.1      ✔ stringr 1.4.1
## ✔ readr   2.1.2      ✔ forcats 0.5.2
## ── Conflicts ──────────────────────────────────────────── tidyverse_conflicts() ──
## ✖ purrr::accumulate() masks foreach::accumulate()
## ✖ dplyr::combine()    masks randomForest::combine()
## ✖ dplyr::filter()     masks stats::filter()
## ✖ dplyr::lag()        masks stats::lag()
## ✖ ggplot2::margin()   masks randomForest::margin()
## ✖ purrr::when()       masks foreach::when()
```

```
library(tree)
library(xgboost)
```

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

```
creditcard <- read_csv("creditcard.csv")
```

```
## Rows: 284807 Columns: 31
## ── Column specification ────────────────────────────────────────────────────────
## Delimiter: ","
## dbl (31): Time, V1, V2, V3, V4, V5, V6, V7, V8, V9, V10, V11, V12, V13, V14,...
##
## ℹ Use `spec()` to retrieve the full column specification for this data.
## ℹ Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

Train / Test Split

Lets split our data into 75% train and 25% test. We need to use 10k data points only since it would take too long for this dataset to load all of it. We use the sample function to randomize. Also, we add all of the fraud data points since we need to make sure there is fraudulent data. It is very important to scale the data (not including the class) to make sure the data is uniform and can be put into one dimension.

We can see that after scaling, our shuffled data has all rows as numeric except the last one, which is class. It is important to write this as a factor.

```
random= creditcard[sample(1:nrow(creditcard)), ]
shuffled_data <- (random[, 0:31])
shuffled_data[c(1,30)] <- scale(shuffled_data[c(1, 30)])
shuffled_data$Class <- factor(shuffled_data$Class)

str(shuffled_data)
```

```
## tibble [284,807 × 31] (S3: tbl_df/tbl/data.frame)
##  $ Time  : num [1:284807] 0.9283 -0.6769 0.0666 -1.0127 1.5603 ...
##  $ V1    : num [1:284807] 1.992 -2.391 2.004 -1.469 -0.968 ...
##  $ V2    : num [1:284807] 0.221 -0.794 -0.224 -5.27 1.857 ...
##  $ V3    : num [1:284807] -2.31 1.92 -0.3 -1.61 -1.8 ...
##  $ V4    : num [1:284807] 0.739 1.338 0.394 0.637 -1.53 ...
##  $ V5    : num [1:284807] 0.572 -0.662 -0.297 -2.213 1.215 ...
##  $ V6    : num [1:284807] -0.867 0.468 -0.118 -0.122 -0.933 ...
##  $ V7    : num [1:284807] 0.0983 -0.0605 -0.6208 1.5315 1.2285 ...
##  $ V8    : num [1:284807] -0.117 0.8502 -0.0871 -0.4765 0.1542 ...
##  $ V9    : num [1:284807] 0.546 -0.52 2.252 -0.766 -0.219 ...
##  $ V10   : num [1:284807] -0.887 -0.637 -0.416 -0.116 0.389 ...
##  $ V11   : num [1:284807] 1.066 1.301 0.207 -0.44 -0.22 ...
##  $ V12   : num [1:284807] 0.268 1.344 -1.751 -0.937 0.515 ...
##  $ V13   : num [1:284807] -0.6877 0.7655 2.592 -0.8674 0.0921 ...
##  $ V14   : num [1:284807] -2.375 0.165 1.156 0.384 0.918 ...
##  $ V15   : num [1:284807] -0.977 0.348 -0.189 0.328 -0.823 ...
##  $ V16   : num [1:284807] 0.596 -0.156 0.383 0.479 -0.11 ...
##  $ V17   : num [1:284807] 1.739 0.13 0.162 1.007 -0.912 ...
##  $ V18   : num [1:284807] 1.193 0.533 -0.259 -1.674 0.427 ...
##  $ V19   : num [1:284807] 0.07933 0.19197 -0.29585 -0.00752 0.46349 ...
##  $ V20   : num [1:284807] -0.153 1.02 -0.145 2.947 0.257 ...
##  $ V21   : num [1:284807] 0.0139 0.5366 -0.3633 1.0591 0.1834 ...
##  $ V22   : num [1:284807] 0.321 0.841 -0.687 -0.266 0.782 ...
##  $ V23   : num [1:284807] -0.0927 0.3372 0.4472 -1.4214 -0.3125 ...
##  $ V24   : num [1:284807] -0.57 0.241 0.557 0.184 -1.051 ...
##  $ V25   : num [1:284807] 0.1921 0.4148 -0.6156 0.1367 -0.0107 ...
##  $ V26   : num [1:284807] 0.735 -0.128 0.209 -0.189 0.151 ...
##  $ V27   : num [1:284807] -0.041 0.1916 -0.0591 -0.2789 0.5806 ...
##  $ V28   : num [1:284807] -0.0265 -0.082 -0.0413 0.2551 0.366 ...
##  $ Amount: num [1:284807] -0.304 0.735 -0.301 5.588 -0.352 ...
##  $ Class : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
```

```
set.seed(1234)
i <- sample(1:nrow(shuffled_data), 0.75*nrow(shuffled_data), replace = FALSE)
train <- shuffled_data[i,]
test <- shuffled_data[-i,]
```

```
set.seed(1234)
creditcard2 <- shuffled_data[1:10000, ]
fraud <- subset(creditcard, (creditcard$Class == 1))
fraud[c(1,30)] <- scale(fraud[c(1, 30)])
creditcard2 <- rbind(creditcard2, fraud)

i <- sample(1:nrow(creditcard2), 0.75*nrow(creditcard2), replace = FALSE)
train2 <- creditcard2[i,]
test2 <- creditcard2[-i,]
```

Decision Tree

Decision trees use recursion to split input observations until the observations are uniform.

Let's use the tree package for decision trees. According to our textbook, it tends to make better trees and is just different compared to using rpart.

```
tree_creditcard2 <- tree(Class~., data=creditcard2)
tree_creditcard2
```

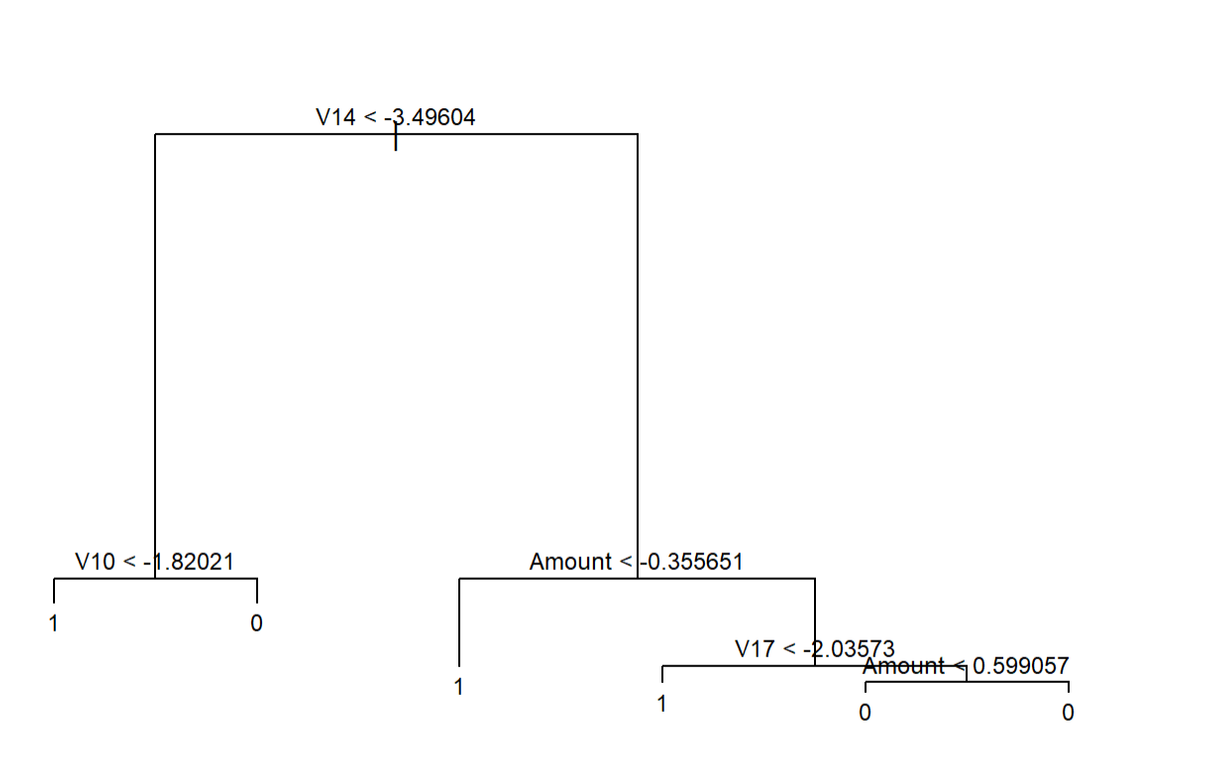
```
## node), split, n, deviance, yval, (yprob)
##       * denotes terminal node
##
##  1) root 10492 4032.00 0 ( 0.952154 0.047846 )
##    2) V14 < -3.49604 443  249.70 1 ( 0.081264 0.918736 )
##      4) V10 < -1.82021 395   44.70 1 ( 0.010127 0.989873 ) *
##      5) V10 > -1.82021 48   61.11 0 ( 0.666667 0.333333 ) *
##    3) V14 > -3.49604 10049 1075.00 0 ( 0.990546 0.009454 )
##      6) Amount < -0.355651 53    0.00 1 ( 0.000000 1.000000 ) *
##      7) Amount > -0.355651 9996  543.50 0 ( 0.995798 0.004202 )
##       14) V17 < -2.03573 12   10.81 1 ( 0.166667 0.833333 ) *
##       15) V17 > -2.03573 9984  431.40 0 ( 0.996795 0.003205 )
##         30) Amount < 0.599057 9136  183.20 0 ( 0.998687 0.001313 ) *
##         31) Amount > 0.599057 848  189.40 0 ( 0.976415 0.023585 ) *
```

```
summary(tree_creditcard2)
```

```
##
## Classification tree:
## tree(formula = Class ~ ., data = creditcard2)
## Variables actually used in tree construction:
## [1] "V14"    "V10"    "Amount" "V17"
## Number of terminal nodes:  6
## Residual mean deviance:  0.04666 = 489.3 / 10490
## Misclassification error rate: 0.005147 = 54 / 10492
```

The fraction of the predictsns that were wrong was 40/10492, which is pretty good.

```
plot(tree_creditcard2)
text(tree_creditcard2, cex=0.75, pretty=0)
```




```
pred <- predict(tree_creditcard2, newdata = test, type = "class")
table(pred, test$Class)
```

```
##
## pred     0     1
##    0 71036    17
##    1    49   100
```

```
mean(pred==test$Class)
```

```
## [1] 0.9990731
```

We have a 99.8% accuracy for the decision tree.

Random Forest

In random forests, trees are de-corellated and at each split of the tree, a random subset of predictors is selected from all the predictors and one is chosen. This approach prevents trees from chosing the same predictors in the same order.

```
set.seed(1234)
rf <- randomForest(Class~., data=train2, importance = TRUE)
```

```
pred <- predict(rf, newdata = test, type = "response")
table(pred, test$Class)
```

```
##
## pred     0     1
##    0 71061    12
##    1    24   105
```

```
mean(pred==test$Class)
```

```
## [1] 0.9994944
```

The random forest algorithm predicted more credit card frauds than the decision tree. It had an accuracy of 0.999%, which is amazing!

XGBoost

The XGBoost algorithm runs up to 10 times faster than earlier tree algorithms. It was developed by Tianqi Chen. However, for XGBoost to work, the training data needs to be converted into a numeric matrix.

```
train_label <- ifelse(train$Class==1, 1, 0)
train_matrix <- data.matrix(train[,])
xgboostmodel <- xgboost(data=train_matrix, label=train_label, nrounds = 100, objective = 'binary:logistic')
```

```
## [1]  train-logloss:0.437496
## [2]  train-logloss:0.296293
## [3]  train-logloss:0.207319
## [4]  train-logloss:0.147793
## [5]  train-logloss:0.106601
## [6]  train-logloss:0.077491
## [7]  train-logloss:0.056632
## [8]  train-logloss:0.041544
## [9]  train-logloss:0.030558
## [10] train-logloss:0.022521
## [11] train-logloss:0.016622
## [12] train-logloss:0.012281
## [13] train-logloss:0.009081
## [14] train-logloss:0.006719
## [15] train-logloss:0.004973
## [16] train-logloss:0.003683
## [17] train-logloss:0.002729
## [18] train-logloss:0.002022
## [19] train-logloss:0.001499
## [20] train-logloss:0.001112
## [21] train-logloss:0.000825
## [22] train-logloss:0.000614
## [23] train-logloss:0.000459
## [24] train-logloss:0.000344
## [25] train-logloss:0.000256
## [26] train-logloss:0.000193
## [27] train-logloss:0.000148
## [28] train-logloss:0.000111
## [29] train-logloss:0.000086
## [30] train-logloss:0.000066
## [31] train-logloss:0.000053
## [32] train-logloss:0.000040
## [33] train-logloss:0.000034
## [34] train-logloss:0.000029
## [35] train-logloss:0.000023
## [36] train-logloss:0.000020
## [37] train-logloss:0.000018
## [38] train-logloss:0.000015
## [39] train-logloss:0.000014
## [40] train-logloss:0.000013
## [41] train-logloss:0.000013
## [42] train-logloss:0.000012
## [43] train-logloss:0.000012
## [44] train-logloss:0.000010
## [45] train-logloss:0.000010
## [46] train-logloss:0.000010
## [47] train-logloss:0.000009
## [48] train-logloss:0.000009
## [49] train-logloss:0.000009
## [50] train-logloss:0.000009
## [51] train-logloss:0.000009
## [52] train-logloss:0.000009
## [53] train-logloss:0.000009
## [54] train-logloss:0.000009
## [55] train-logloss:0.000009
## [56] train-logloss:0.000009
## [57] train-logloss:0.000009
## [58] train-logloss:0.000009
## [59] train-logloss:0.000009
## [60] train-logloss:0.000009
## [61] train-logloss:0.000009
## [62] train-logloss:0.000009
## [63] train-logloss:0.000009
## [64] train-logloss:0.000009
## [65] train-logloss:0.000009
## [66] train-logloss:0.000009
## [67] train-logloss:0.000009
## [68] train-logloss:0.000009
## [69] train-logloss:0.000009
## [70] train-logloss:0.000009
## [71] train-logloss:0.000009
## [72] train-logloss:0.000009
## [73] train-logloss:0.000009
## [74] train-logloss:0.000009
## [75] train-logloss:0.000009
## [76] train-logloss:0.000009
## [77] train-logloss:0.000009
## [78] train-logloss:0.000009
## [79] train-logloss:0.000009
## [80] train-logloss:0.000009
## [81] train-logloss:0.000009
## [82] train-logloss:0.000009
## [83] train-logloss:0.000009
## [84] train-logloss:0.000009
## [85] train-logloss:0.000009
## [86] train-logloss:0.000009
## [87] train-logloss:0.000009
## [88] train-logloss:0.000009
## [89] train-logloss:0.000009
## [90] train-logloss:0.000009
## [91] train-logloss:0.000009
## [92] train-logloss:0.000009
## [93] train-logloss:0.000009
## [94] train-logloss:0.000009
## [95] train-logloss:0.000009
## [96] train-logloss:0.000009
## [97] train-logloss:0.000009
## [98] train-logloss:0.000009
## [99] train-logloss:0.000009
## [100]    train-logloss:0.000009
```

```
test_label <- ifelse(test$Class==1, 1, 0)
test_matrix <- data.matrix(test[,])

probs <- predict(xgboostmodel,test_matrix)
pred <- ifelse(probs>0.5, 1, 0)

table(pred, test$Class)
```

```
##
## pred     0     1
##    0 71085     0
##    1     0   117
```

```
mean(pred==test$Class)
```

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

Wow. The accuracy for XGBoost is over 100%.

SuperLearner

This isn't the most optimal method, but we could try it. We know it isn't optimal, so we move on to analysis.

Analysis

XGBoost and Random Forest works best. This is expected because it is an optimized algorithm to run fast. Random Forest can also handle large datasets efficiently, and provides a high level of accuracy for predictions because of the emphasis for feature selection. Decision trees performed really good as well (most our algorithms had an accuracy of over 100%), but it was our baseline and isn't as optimized as the other two.