Classification

Here is the data used for classification.

Dividing Data

```
library(readr)
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.2 --
## v ggplot2 3.3.6      v dplyr   1.0.10
## v tibble  3.1.8      v stringr 1.4.1
## v tidyr   1.2.1      v forcats 0.5.2
## v purrr   0.3.4
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(dplyr)
library(ROCR)
library(xgboost)
```

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

```
library(mccr)
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
##
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
library(ISLR)
library(caret)
```

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

```
##
## Attaching package: 'caret'
##
## The following object is masked from 'package:purrr':
##
##     lift
```

```
library(adabag)
```

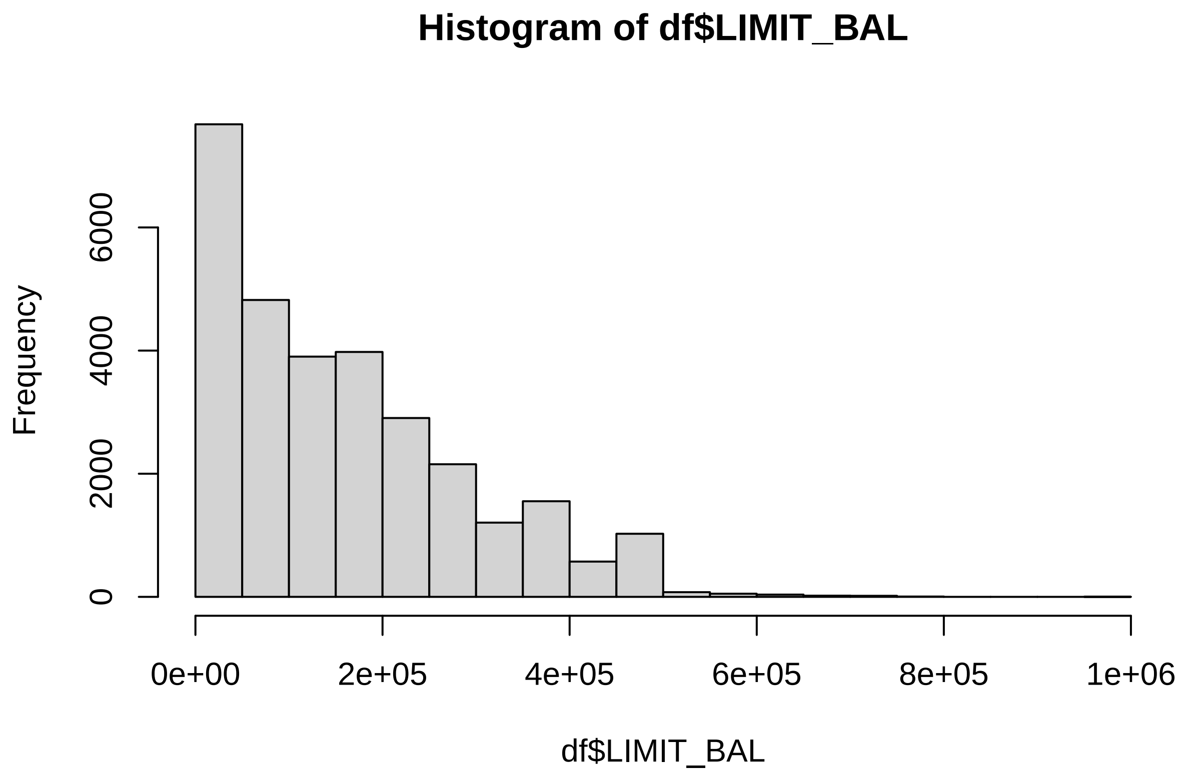
```
## Loading required package: rpart
## Loading required package: foreach
##
## Attaching package: 'foreach'
##
## The following objects are masked from 'package:purrr':
##
##     accumulate, when
##
## Loading required package: doParallel
## Loading required package: iterators
## Loading required package: parallel
```

```
library(tree)
library(rpart)
library(e1071)
library(tree)

df <- read.csv("credit.csv")

set.seed(1234)
i <- sample(1:nrow(df), nrow(df)*0.8, replace=FALSE)
train <- df[i,]
test <- df[-i,]

set.seed(1234)
train_2 <- train
train_2$dpnm <- as.factor(train_2$dpnm)
```

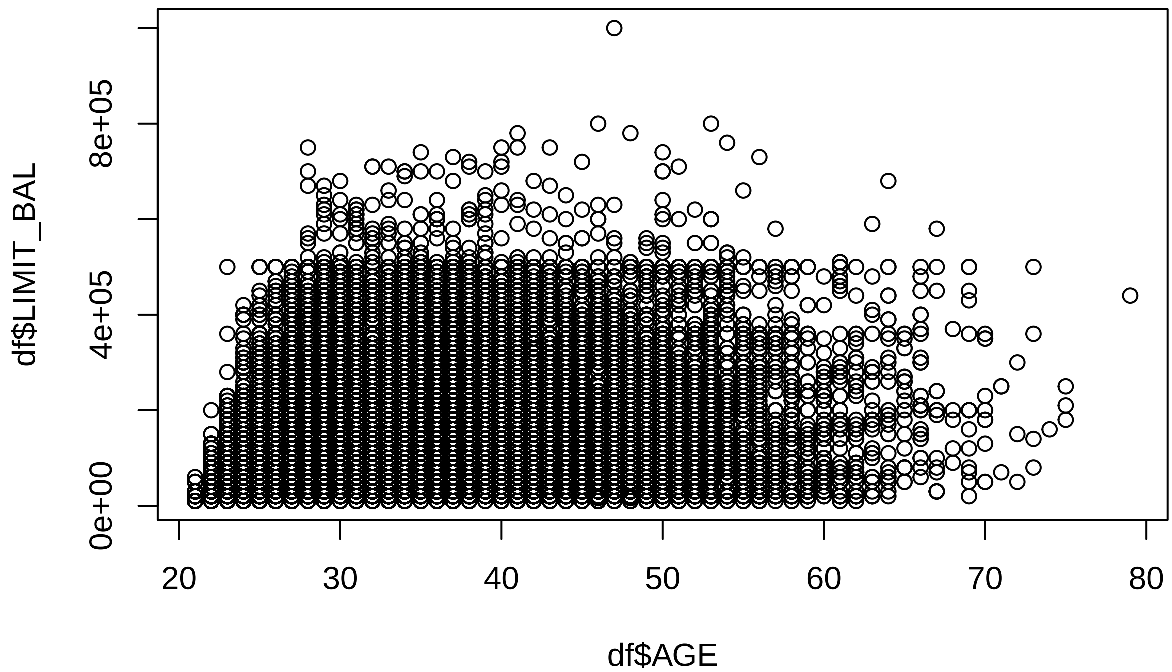
Data exploration

Lets take a look at the data graphically as well as some of the statistics.

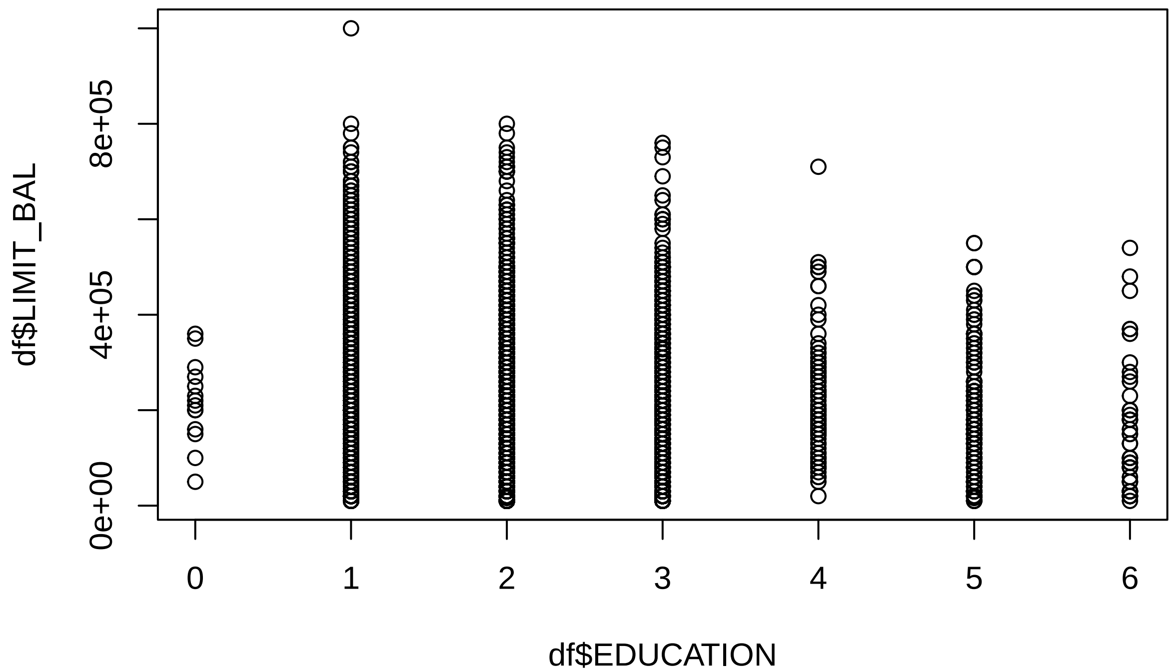
```
hist(df$LIMIT_BAL)
```




```
plot(df$AGE, df$LIMIT_BAL)
```




```
plot(df$EDUCATION, df$LIMIT_BAL)
```




```
cor(df[2:6], use = "complete")
```

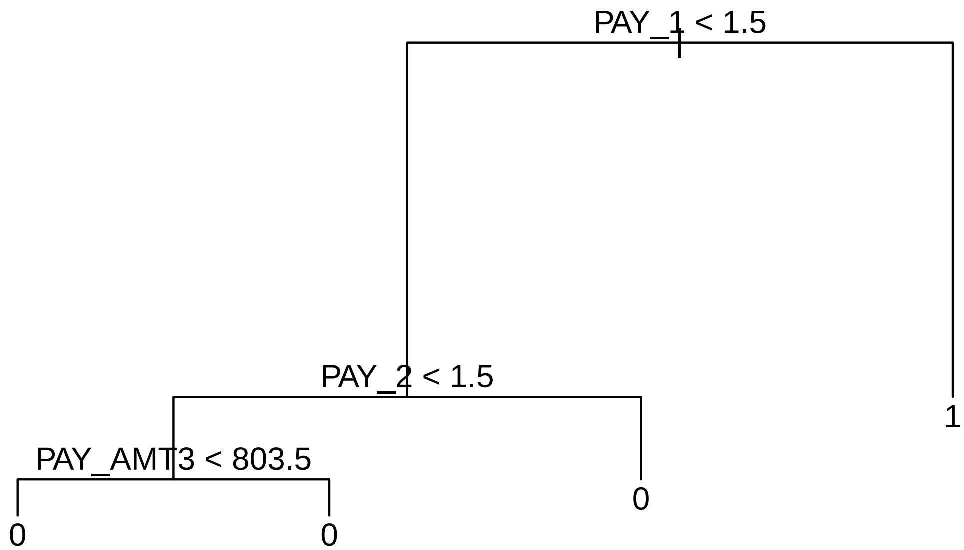
```
##              LIMIT_BAL         SEX   EDUCATION    MARRIAGE         AGE
## LIMIT_BAL  1.00000000  0.02475524 -0.21916070 -0.10813941  0.14471280
## SEX        0.02475524  1.00000000  0.01423194 -0.03138884 -0.09087365
## EDUCATION -0.21916070  0.01423194  1.00000000 -0.14346434  0.17506066
## MARRIAGE  -0.10813941 -0.03138884 -0.14346434  1.00000000 -0.41416992
## AGE        0.14471280 -0.09087365  0.17506066 -0.41416992  1.00000000
```

Decision Tree Baseline

```
start_time <- Sys.time()
tree1 <- tree(as.factor(dpnm) ~., data=train)
end_time <- Sys.time()
summary(tree1)
```

```
##
## Classification tree:
## tree(formula = as.factor(dpnm) ~ ., data = train)
## Variables actually used in tree construction:
## [1] "PAY_1"    "PAY_2"    "PAY_AMT3"
## Number of terminal nodes:  4
## Residual mean deviance:  0.8926 = 21420 / 24000
## Misclassification error rate: 0.1809 = 4341 / 24000
```

```
plot(tree1)
text(tree1, cex=1)
```




```
print(paste("Training time: ", (end_time-start_time), "s"))
```

```
## [1] "Training time:  0.296769142150879 s"
```

```
pred <- predict(tree1,newdata=test,type="class")
table(pred,test$dpnm)
```

```
##
## pred    0    1
##    0 4526  897
##    1  174  403
```

```
acc <- mean(pred==test$dpnm)
mcc <- mccr(factor(pred), test$dpnm)
print(paste("accuracy=", acc))
```

```
## [1] "accuracy= 0.8215"
```

```
print(paste("mcc=", mcc))
```

```
## [1] "mcc= 0.381453579643801"
```

##Random Forest

```
start_time <- Sys.time()
rf <- randomForest(dpnm ~ ., data=train_2, importance = TRUE, proximity = TRUE)
end_time <- Sys.time()

summary(rf)
```

```
##                 Length    Class  Mode
## call                    5 -none- call
## type                    1 -none- character
## predicted           24000 factor numeric
## err.rate             1500 -none- numeric
## confusion               6 -none- numeric
## votes               48000 matrix numeric
## oob.times           24000 -none- numeric
## classes                 2 -none- character
## importance             96 -none- numeric
## importanceSD           72 -none- numeric
## localImportance         0 -none- NULL
## proximity       576000000 -none- numeric
## ntree                   1 -none- numeric
## mtry                    1 -none- numeric
## forest                 14 -none- list
## y                   24000 factor numeric
## test                    0 -none- NULL
## inbag                   0 -none- NULL
## terms                   3 terms  call
```

```
print(paste("Training time: ", (end_time-start_time), "s"))
```

```
## [1] "Training time:  7.65738598108292 s"
```

```
pred <- predict(rf, newdata = test)
acc <- mean(pred == test$dpnm)
mcc <- mccr(factor(pred), test$dpnm)
table(pred, test$dpnm)
```

```
##
## pred    0    1
##    0 4451  840
##    1  249  460
```

```
print(paste("Accuracy = ", acc))
```

```
## [1] "Accuracy =  0.8185"
```

```
print(paste("MCC = ", mcc))
```

```
## [1] "MCC =  0.383975737946668"
```

XGBoost

```
start_time <- Sys.time()
xg <- xgboost(data = data.matrix(train), label = train$dpnm, nrounds = 100)
```

```
## [1]  train-rmse:0.350012
## [2]  train-rmse:0.245017
## [3]  train-rmse:0.171518
## [4]  train-rmse:0.120067
## [5]  train-rmse:0.084050
## [6]  train-rmse:0.058837
## [7]  train-rmse:0.041187
## [8]  train-rmse:0.028832
## [9]  train-rmse:0.020183
```

```
## [10] train-rmse:0.014129
## [11] train-rmse:0.009891
## [12] train-rmse:0.006924
## [13] train-rmse:0.004847
## [14] train-rmse:0.003393
## [15] train-rmse:0.002375
## [16] train-rmse:0.001663
## [17] train-rmse:0.001164
## [18] train-rmse:0.000815
## [19] train-rmse:0.000570
## [20] train-rmse:0.000399
## [21] train-rmse:0.000279
## [22] train-rmse:0.000196
## [23] train-rmse:0.000137
## [24] train-rmse:0.000096
## [25] train-rmse:0.000067
## [26] train-rmse:0.000047
## [27] train-rmse:0.000033
## [28] train-rmse:0.000023
## [29] train-rmse:0.000016
## [30] train-rmse:0.000011
## [31] train-rmse:0.000008
## [32] train-rmse:0.000006
## [33] train-rmse:0.000005
## [34] train-rmse:0.000005
## [35] train-rmse:0.000005
## [36] train-rmse:0.000005
## [37] train-rmse:0.000005
## [38] train-rmse:0.000005
## [39] train-rmse:0.000005
## [40] train-rmse:0.000005
## [41] train-rmse:0.000005
## [42] train-rmse:0.000005
## [43] train-rmse:0.000005
## [44] train-rmse:0.000005
## [45] train-rmse:0.000005
## [46] train-rmse:0.000005
## [47] train-rmse:0.000005
## [48] train-rmse:0.000005
## [49] train-rmse:0.000005
## [50] train-rmse:0.000005
## [51] train-rmse:0.000005
## [52] train-rmse:0.000005
## [53] train-rmse:0.000005
## [54] train-rmse:0.000005
## [55] train-rmse:0.000005
## [56] train-rmse:0.000005
## [57] train-rmse:0.000005
## [58] train-rmse:0.000005
## [59] train-rmse:0.000005
## [60] train-rmse:0.000005
## [61] train-rmse:0.000005
## [62] train-rmse:0.000005
## [63] train-rmse:0.000005
```

```
## [64] train-rmse:0.000005
## [65] train-rmse:0.000005
## [66] train-rmse:0.000005
## [67] train-rmse:0.000005
## [68] train-rmse:0.000005
## [69] train-rmse:0.000005
## [70] train-rmse:0.000005
## [71] train-rmse:0.000005
## [72] train-rmse:0.000005
## [73] train-rmse:0.000005
## [74] train-rmse:0.000005
## [75] train-rmse:0.000005
## [76] train-rmse:0.000005
## [77] train-rmse:0.000005
## [78] train-rmse:0.000005
## [79] train-rmse:0.000005
## [80] train-rmse:0.000005
## [81] train-rmse:0.000005
## [82] train-rmse:0.000005
## [83] train-rmse:0.000005
## [84] train-rmse:0.000005
## [85] train-rmse:0.000005
## [86] train-rmse:0.000005
## [87] train-rmse:0.000005
## [88] train-rmse:0.000005
## [89] train-rmse:0.000005
## [90] train-rmse:0.000005
## [91] train-rmse:0.000005
## [92] train-rmse:0.000005
## [93] train-rmse:0.000005
## [94] train-rmse:0.000005
## [95] train-rmse:0.000005
## [96] train-rmse:0.000005
## [97] train-rmse:0.000005
## [98] train-rmse:0.000005
## [99] train-rmse:0.000005
## [100]    train-rmse:0.000005
```

```
end_time <- Sys.time()
summary(xg)
```

```
##                 Length Class              Mode
## handle              1  xgb.Booster.handle externalptr
## raw             75159  -none-             raw
## niter               1  -none-             numeric
## evaluation_log      2  data.table         list
## call               13  -none-             call
## params              1  -none-             list
## callbacks           2  -none-             list
## feature_names      25  -none-             character
## nfeatures           1  -none-             numeric
```

```
print(paste("Training time: ", (end_time-start_time), "s"))
```

```
## [1] "Training time:  0.669389963150024 s"
```

```r
pred <- predict(xg, data.matrix(test), reshape=TRUE)

prediction <- as.numeric(pred > 0.5)

err <- mean(prediction != test$dpnm)
print(paste("test-error=", err))
```

```
## [1] "test-error= 0"
```

Adabag

```r
start_type <-Sys.time()
adab <- boosting(dpnm~. , data=train_2, boos = TRUE, mfinal=10)
end_time <- Sys.time()
summary(adab)
```

```
##            Length Class   Mode
## formula        3  formula call
## trees         10  -none-  list
## weights       10  -none-  numeric
## votes      48000  -none-  numeric
## prob       48000  -none-  numeric
## class      24000  -none-  character
## importance    24  -none-  numeric
## terms          3  terms   call
## call           5  -none-  call
```

```r
print(paste("Training time: ", (end_time-start_time), "s"))
```

```
## [1] "Training time:  20.3851799964905 s"
```

```r
pred <- predict(adab, newdata=test, type="response")
pred$class <- as.integer(pred$class)
acc<- mean(pred$class==test$dpnm)
print(paste("accuracy=", acc))
```

```
## [1] "accuracy= 0.817166666666667"
```

##Analysis

Decision Tree - As a baseline, the decision tree provides an accuracy of 82.15%, with an MCC of 0.38, making it fairly accurate, considering that it only took 0.33s to compute, making it nearly instantaneous.

Random Forest - The random Forest algorithm, on the other hand has an accuracy of 81.9%, and an mcc of 0.38, but takes over 10x as long, taking a little over 7 minutes to fully create a model. Because there is not even an increase in accuracy in exchange for the computation time.

XGBoost has a accuracy that is around the same as Random Forest, but is able to do it in a fraction of the time, even coming in faster than the Decision Tree algorithm.

Adaboost is the last algorithm, and this completels in 2.65s with an accuracy of 82.3%, which is not significantly higher than any of the others.

Conclusion - While all the models, and methods were able to maintain a similar level of accuracy while predicting, the biggest difference between them was the time that it took to compute. From all the results seen, it seems that XGBoost is the best technique to use for this type of machine learning. Another option for fast binary classification would be Fast Adaboost.