Ensemble Learning

Aleezah Athar

This dataset is from https://archive.ics.uci.edu/ml/datasets/Bank+Marketing (https://archive.ics.uci.edu/ml/datasets/Bank+Marketing) First we set the seed to 1234 to get the same results each time Then we read in the csv file which contains our data

```
set.seed(1234)
df <- read.csv("bank-full.csv", sep = ";", quote = "")
df
```

X.age.	X.job.	X.marital.	X.education.	X.default.	X.balance.	X.housing.	X.loan
<int>	<chr>	<chr>	<chr>	<chr>	<int>	<chr>	<chr>
58	"management"	"married"	"tertiary"	"no"	2143	"yes"	"no"
44	"technician"	"single"	"secondary"	"no"	29	"yes"	"no"
33	"entrepreneur"	"married"	"secondary"	"no"	2	"yes"	"yes"
47	"blue-collar"	"married"	"unknown"	"no"	1506	"yes"	"no"
33	"unknown"	"single"	"unknown"	"no"	1	"no"	"no"
35	"management"	"married"	"tertiary"	"no"	231	"yes"	"no"
28	"management"	"single"	"tertiary"	"no"	447	"yes"	"yes"
42	"entrepreneur"	"divorced"	"tertiary"	"yes"	2	"yes"	"no"
58	"retired"	"married"	"primary"	"no"	121	"yes"	"no"
43	"technician"	"single"	"secondary"	"no"	593	"yes"	"no"

1-10 of 10,000 rows | 1-8 of 17 columns Previous **1** 2 3 4 5 6 … 1000 Next

```
for(i in 1:ncol(df)){
  if(is.character(df[,i])){
    df[,i] <- as.factor(df[,i])
  }
}


i <- sample(nrow(df), 0.8*nrow(df), replace = FALSE)
train <- df[i,]
test <- df[-i,]

str(train)
```

```
## 'data.frame':    36168 obs. of  17 variables:
##  $ X.age.      : int  45 60 77 50 37 48 56 50 37 32 ...
##  $ X.job.      : Factor w/ 12 levels "\"admin.\"","\"blue-collar\"",..: 5 2 5 10 1 2
5 10 5 2 ...
##  $ X.marital.  : Factor w/ 3 levels "\"divorced\"",..: 2 2 2 1 2 2 1 3 2 3 ...
##  $ X.education.: Factor w/ 4 levels "\"primary\"",..: 3 1 4 2 2 2 3 2 3 2 ...
##  $ X.default.  : Factor w/ 2 levels "\"no\"","\"yes\"": 1 1 1 1 1 1 1 1 1 1 ...
##  $ X.balance.  : int  3857 631 1780 8016 749 1794 -59 246 578 1940 ...
##  $ X.housing.  : Factor w/ 2 levels "\"no\"","\"yes\"": 2 1 2 1 1 2 1 2 2 2 ...
##  $ X.loan.     : Factor w/ 2 levels "\"no\"","\"yes\"": 1 1 1 1 1 2 2 1 1 2 ...
##  $ X.contact.  : Factor w/ 3 levels "\"cellular\"",..: 1 1 1 1 1 1 1 1 1 3 ...
##  $ X.day.      : int  11 12 23 17 21 8 28 17 8 13 ...
##  $ X.month.    : Factor w/ 12 levels "\"apr\"","\"aug\"",..: 2 2 11 6 1 9 6 6 2 9 ...
##  $ X.duration. : int  425 429 221 903 219 97 127 174 401 299 ...
##  $ X.campaign. : int  2 2 2 4 1 1 6 2 4 3 ...
##  $ X.pdays.    : int  190 -1 183 -1 -1 343 -1 -1 -1 -1 ...
##  $ X.previous. : int  1 0 3 0 0 2 0 0 0 0 ...
##  $ X.poutcome. : Factor w/ 4 levels "\"failure\"",..: 1 4 3 4 4 1 4 4 4 4 ...
##  $ X.y.        : Factor w/ 2 levels "\"no\"","\"yes\"": 1 2 2 1 1 1 1 1 1 1 ...
```

AdaBoost

```
#installing the necessary packages
#install.packages("adabag")
library(adabag)
```

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

```
## Loading required package: caret
```

```
## Loading required package: ggplot2
```

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

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

```
## Loading required package: doParallel
```

```
## Loading required package: iterators
```

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

```
#Calling the boosting function and passing the response variable and training data
#and calculate time taken
start<-Sys.time()
adab1 <- boosting(X.y.~., data=train, boos=TRUE, mfinal=20, coeflearn='Breiman')
end<-Sys.time()
time_taken<-end-start
#using summary to see results
summary(adab1)
```

```
##            Length Class   Mode
## formula         3  formula call
## trees          20  -none-  list
## weights        20  -none-  numeric
## votes       72336  -none-  numeric
## prob        72336  -none-  numeric
## class       36168  -none-  character
## importance     16  -none-  numeric
## terms           3  terms   call
## call            6  -none-  call
```

```
print("Time taken for Adaboost")
```

```
## [1] "Time taken for Adaboost"
```

```
print(time_taken)
```

```
## Time difference of 21.48767 secs
```

```
#installing the necessary packages
#install.packages("mltools")
library(mltools)
#using predict
pred <- predict(adab1, newdata=test, type="response")
#computing the accuracy and mcc
acc_adabag <- mean(pred$class==test$X.y.)
mcc_adabag <- mcc(factor(pred$class), test$X.y.)
print(paste("accuracy=", acc_adabag))
```

```
## [1] "accuracy= 0.904788233993144"
```

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

```
## [1] "mcc= 0.486859037402595"
```

XGBoost

```
#install.packages("xgboost")
library(xgboost)
#converting the target factor variable back to numeric so we can use XGBoost.
train$X.y.<-as.numeric(train$X.y.)
test$X.y.<-as.numeric(test$X.y.)
#creating a binary outcome variable which checks if the X.y. column is equal to 0 or 1
train_label <- ifelse(train$X.y.==1, 1, 0)
train_matrix <- data.matrix(train[, -17])
#calling the xgboost function and measure time taken

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

```
## [1]  train-logloss:0.509285
## [2]  train-logloss:0.407354
## [3]  train-logloss:0.344175
## [4]  train-logloss:0.302477
## [5]  train-logloss:0.273209
## [6]  train-logloss:0.251951
## [7]  train-logloss:0.237898
## [8]  train-logloss:0.227445
## [9]  train-logloss:0.219607
## [10] train-logloss:0.213445
## [11] train-logloss:0.207265
## [12] train-logloss:0.202050
## [13] train-logloss:0.198404
## [14] train-logloss:0.195709
## [15] train-logloss:0.193672
## [16] train-logloss:0.192047
## [17] train-logloss:0.188974
## [18] train-logloss:0.186765
## [19] train-logloss:0.185610
## [20] train-logloss:0.180432
## [21] train-logloss:0.178809
## [22] train-logloss:0.177191
## [23] train-logloss:0.175593
## [24] train-logloss:0.174245
## [25] train-logloss:0.172242
## [26] train-logloss:0.171367
## [27] train-logloss:0.169883
## [28] train-logloss:0.169011
## [29] train-logloss:0.166981
## [30] train-logloss:0.166537
## [31] train-logloss:0.166370
## [32] train-logloss:0.165642
## [33] train-logloss:0.164142
## [34] train-logloss:0.163995
## [35] train-logloss:0.163286
## [36] train-logloss:0.163157
## [37] train-logloss:0.162553
## [38] train-logloss:0.160729
## [39] train-logloss:0.159718
## [40] train-logloss:0.158977
## [41] train-logloss:0.158209
## [42] train-logloss:0.157710
## [43] train-logloss:0.157391
## [44] train-logloss:0.156892
## [45] train-logloss:0.156818
## [46] train-logloss:0.156133
## [47] train-logloss:0.155863
## [48] train-logloss:0.154601
## [49] train-logloss:0.154039
## [50] train-logloss:0.153621
## [51] train-logloss:0.152947
## [52] train-logloss:0.152491
```

```
## [53] train-logloss:0.151673
## [54] train-logloss:0.150645
## [55] train-logloss:0.150131
## [56] train-logloss:0.149631
## [57] train-logloss:0.148651
## [58] train-logloss:0.148378
## [59] train-logloss:0.148264
## [60] train-logloss:0.148090
## [61] train-logloss:0.147335
## [62] train-logloss:0.147125
## [63] train-logloss:0.146658
## [64] train-logloss:0.145789
## [65] train-logloss:0.145092
## [66] train-logloss:0.144591
## [67] train-logloss:0.144537
## [68] train-logloss:0.144261
## [69] train-logloss:0.144192
## [70] train-logloss:0.143993
## [71] train-logloss:0.143395
## [72] train-logloss:0.142384
## [73] train-logloss:0.141898
## [74] train-logloss:0.141744
## [75] train-logloss:0.141701
## [76] train-logloss:0.140763
## [77] train-logloss:0.140451
## [78] train-logloss:0.140086
## [79] train-logloss:0.139583
## [80] train-logloss:0.139429
## [81] train-logloss:0.139280
## [82] train-logloss:0.138856
## [83] train-logloss:0.138268
## [84] train-logloss:0.138171
## [85] train-logloss:0.137171
## [86] train-logloss:0.136460
## [87] train-logloss:0.135201
## [88] train-logloss:0.134154
## [89] train-logloss:0.132823
## [90] train-logloss:0.132540
## [91] train-logloss:0.132181
## [92] train-logloss:0.131731
## [93] train-logloss:0.131162
## [94] train-logloss:0.131041
## [95] train-logloss:0.130777
## [96] train-logloss:0.130735
## [97] train-logloss:0.130272
## [98] train-logloss:0.129038
## [99] train-logloss:0.128584
## [100]    train-logloss:0.127734
```

```
end<-Sys.time()
time_taken<-end-start
print("Time taken for xgboost: ")
```

```
## [1] "Time taken for xgboost: "
```

```
print(time_taken)
```

```
## Time difference of 2.107458 secs
```

```
#Create binary test labels
test_label <- ifelse(test$X.y.==1, 1, 0)
#Create a test matrix by removing the 17th column from the 'test' data frame
test_matrix <- data.matrix(test[, -17])
#use the predict function
probs <- predict(model, test_matrix)
#check if probability is greater than 0.5
pred <- ifelse(probs>0.5, 1, 0)
#Calculate the accuracy
acc_xg <- mean(pred==test_label)
#Calculate the Matthews correlation coefficient (MCC)
mcc_xg <- mcc(pred, test_label)
#Print the accuracy of the test predictions
print(paste("accuracy=", acc_xg))
```

```
## [1] "accuracy= 0.90677872387482"
```

```
#Print the MCC of the test predictions
print(paste("mcc=", mcc_xg))
```

```
## [1] "mcc= 0.496263646910475"
```

Conclusion

I used AdaBoost and XGBoost to improve the performance of my project. The time taken for AdaBoost was 22 seconds and the time taken for XGBoost was 2 seconds. Therefore, XGBoost was a lot faster. Furthermore, the accuracy for adaboost was 0.905 and mcc was 0.49. The accuracy for XGBoost was 0.907 and mcc was 0.50. XGBoost was faster and slightly more accurate. The mcc was also closer to 1 using XGBoost. Finally, I believe XGBoost was superior and would prefer to use it moving forward.