# Ensemble

Aloksai Choudari, Alekhya Pinnamaneni

2022-10-21

## Load the data

```
avila <- read.csv("avila-tr.txt", header = FALSE)
```

## Clean the data

```
# Add column names to the data set
colnames(avila) <- c("intercolumnar_dist", "upper_margin", "lower_margin", "exploitation", "row_no", "m

# Factor the copyist column
avila$copyist <- as.factor(avila$copyist)
```

## Divide data into train, test, and validate sets

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

## Decision Tree

```
library(tree)
library(mltools)
start <- Sys.time()
tree <- tree(copyist~., data=train)
end <-Sys.time()
rt <- end - start
pred1 <- predict(tree, newdata=test, type="class")
acc1 <- mean(pred1==test$copyist)
print(paste("accuracy = ", acc1))
```

```
## [1] "accuracy =  0.605944391179291"
```

```
mcc1 <- mcc(pred1, test$copyist)
print(paste("MCC = ", mcc1))
```

```
## [1] "MCC =  0.484952025347181"
```

```
print(paste("Runtime: ", rt, " secs"))
```

```
## [1] "Runtime:  0.250961065292358  secs"
```

### Random Forest

```
library(randomForest)
```

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

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

```
set.seed(1234)
start <- Sys.time()
rf <- randomForest(copyist~., data=train, importance=TRUE)
end <-Sys.time()
rt <- end - start
pred2 <- predict(rf, newdata=test, type="response")
acc2 <- mean(pred2==test$copyist)
print(paste("accuracy = ", acc2))
```

```
## [1] "accuracy =  0.980824544582934"
```

```
mcc2 <- mcc(pred2, test$copyist)
print(paste("MCC = ", mcc2))
```

```
## [1] "MCC =  0.975183120838423"
```

```
print(paste("Runtime: ", rt, " secs"))
```

```
## [1] "Runtime:  18.7834229469299  secs"
```

### XGBoost

```
library(xgboost)
train_label <- as.numeric(train$copyist) - 1
start <- Sys.time()
xgb <- xgboost(data=data.matrix(train[, -11]), label=train_label, nrounds=100, objective='multi:softprob
```

```
## [1]  train-mlogloss:1.520024
## [2]  train-mlogloss:1.193194
## [3]  train-mlogloss:0.983192
## [4]  train-mlogloss:0.822472
## [5]  train-mlogloss:0.700460
## [6]  train-mlogloss:0.608003
## [7]  train-mlogloss:0.536446
## [8]  train-mlogloss:0.473718
## [9]  train-mlogloss:0.422334
## [10] train-mlogloss:0.376622
## [11] train-mlogloss:0.340785
## [12] train-mlogloss:0.309715
## [13] train-mlogloss:0.284524
## [14] train-mlogloss:0.259570
## [15] train-mlogloss:0.241138
## [16] train-mlogloss:0.222981
## [17] train-mlogloss:0.207744
## [18] train-mlogloss:0.193145
## [19] train-mlogloss:0.182451
## [20] train-mlogloss:0.164945
## [21] train-mlogloss:0.152143
## [22] train-mlogloss:0.140660
## [23] train-mlogloss:0.128668
## [24] train-mlogloss:0.123155
## [25] train-mlogloss:0.115696
## [26] train-mlogloss:0.108849
## [27] train-mlogloss:0.099612
## [28] train-mlogloss:0.091229
## [29] train-mlogloss:0.082971
## [30] train-mlogloss:0.077507
## [31] train-mlogloss:0.072626
## [32] train-mlogloss:0.069316
## [33] train-mlogloss:0.065299
## [34] train-mlogloss:0.059482
## [35] train-mlogloss:0.055326
## [36] train-mlogloss:0.053088
## [37] train-mlogloss:0.050855
## [38] train-mlogloss:0.047287
## [39] train-mlogloss:0.044611
## [40] train-mlogloss:0.043226
## [41] train-mlogloss:0.040686
## [42] train-mlogloss:0.037749
## [43] train-mlogloss:0.035535
## [44] train-mlogloss:0.033642
## [45] train-mlogloss:0.032270
## [46] train-mlogloss:0.030148
## [47] train-mlogloss:0.028827
## [48] train-mlogloss:0.026718
## [49] train-mlogloss:0.025846
## [50] train-mlogloss:0.024472
## [51] train-mlogloss:0.023543
## [52] train-mlogloss:0.022492
## [53] train-mlogloss:0.021652
## [54] train-mlogloss:0.020584
```

```
## [55] train-mlogloss:0.019613
## [56] train-mlogloss:0.018924
## [57] train-mlogloss:0.018639
## [58] train-mlogloss:0.018009
## [59] train-mlogloss:0.016954
## [60] train-mlogloss:0.016534
## [61] train-mlogloss:0.015706
## [62] train-mlogloss:0.015015
## [63] train-mlogloss:0.014708
## [64] train-mlogloss:0.014414
## [65] train-mlogloss:0.013857
## [66] train-mlogloss:0.013631
## [67] train-mlogloss:0.013370
## [68] train-mlogloss:0.013109
## [69] train-mlogloss:0.012782
## [70] train-mlogloss:0.012577
## [71] train-mlogloss:0.011779
## [72] train-mlogloss:0.011538
## [73] train-mlogloss:0.011405
## [74] train-mlogloss:0.010962
## [75] train-mlogloss:0.010558
## [76] train-mlogloss:0.010420
## [77] train-mlogloss:0.010067
## [78] train-mlogloss:0.009803
## [79] train-mlogloss:0.009546
## [80] train-mlogloss:0.009326
## [81] train-mlogloss:0.009114
## [82] train-mlogloss:0.008956
## [83] train-mlogloss:0.008669
## [84] train-mlogloss:0.008480
## [85] train-mlogloss:0.008326
## [86] train-mlogloss:0.008215
## [87] train-mlogloss:0.008066
## [88] train-mlogloss:0.007867
## [89] train-mlogloss:0.007649
## [90] train-mlogloss:0.007503
## [91] train-mlogloss:0.007409
## [92] train-mlogloss:0.007268
## [93] train-mlogloss:0.007069
## [94] train-mlogloss:0.006985
## [95] train-mlogloss:0.006789
## [96] train-mlogloss:0.006592
## [97] train-mlogloss:0.006447
## [98] train-mlogloss:0.006292
## [99] train-mlogloss:0.006157
## [100]    train-mlogloss:0.006055
```

```
end <-Sys.time()
rt <- end - start
test_label <- as.numeric(test$copyist) - 1
prob <- predict(xgb, data.matrix(test[, -11]))
probs <- split(prob, ceiling(seq_along(prob) / 12))
pred3 <- rep(0, 2086)
for (x in 1:2086) {
```

```
  if (probs[[x]][[2]] > 0.5) {
    pred3[x] <- 1
  }
  else if (probs[[x]][[3]] > 0.5) {
    pred3[x] <- 2
  }
  else if (probs[[x]][[4]] > 0.5) {
    pred3[x] <- 3
  }
  else if (probs[[x]][[5]] > 0.5) {
    pred3[x] <- 4
  }
  else if (probs[[x]][[6]] > 0.5) {
    pred3[x] <- 5
  }
  else if (probs[[x]][[7]] > 0.5) {
    pred3[x] <- 6
  }
  else if (probs[[x]][[8]] > 0.5) {
    pred3[x] <- 7
  }
  else if (probs[[x]][[9]] > 0.5) {
    pred3[x] <- 8
  }
  else if (probs[[x]][[10]] > 0.5) {
    pred3[x] <- 9
  }
  else if (probs[[x]][[11]] > 0.5) {
    pred3[x] <- 10
  }
  else if (probs[[x]][[12]] > 0.5) {
    pred3[x] <- 11
  }
}
acc3 <- mean(pred3==test_label)
print(paste("accuracy = ", acc3))
```

```
## [1] "accuracy =  0.996644295302013"
```

```
mcc3 <- mcc(pred3, test_label)
print(paste("MCC = ", mcc3))
```

```
## [1] "MCC =  0.99566264414749"
```

```
print(paste("Runtime: ", rt, " secs"))
```

```
## [1] "Runtime:  16.596076965332  secs"
```

## Adabag

```
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 <- Sys.time()
ada <- boosting(copyist~., data=train, boos=TRUE, mfinal=50, coeflearn='Breiman')
end <-Sys.time()
rt <- end - start
pred4 <- predict(ada, newdata=test, type="response")
acc4 <- mean(pred4$class==test$copyist)
print(paste("accuracy = ", acc4))
```

```
## [1] "accuracy =  0.644774688398849"
```

```
mcc4 <- mcc(factor(pred4$class), test$copyist)
print(paste("MCC = ", mcc4))
```

```
## [1] "MCC =  0.525371924507201"
```

```
print(paste("Runtime: ", rt, " secs"))
```

```
## [1] "Runtime:  30.9686408042908  secs"
```

## Analysis

Out of all three of the ensemble techniques we used in this notebook, XGBoost had the best performance with an accuracy of 0.9966 and Matthew's correlation coefficient (MCC) of 0.9957. This is because we set the nrounds parameter equal to 100, creating 100 trees and resulting in increased accuracy. XGBoost also had the fastest runtime out of the three algorithms, since it uses multithreading. Adabag had the worst performance (accuracy = 0.6448, MCC = 0.5254), and only had slightly better results compared to the baseline decision tree algorithm (accuracy = 0.6059, MCC = 0.4849). Adabag also had the longest runtime. The random Forest algorithm also had excellent performance with an accuracy of 0.9808 and MCC of 0.9752. However, the runtime of the random forest algorithm was slightly longer than that of the XGBoost algorithm.