Ensemble

Nikilas John

October 23th, 2022

Read in the data & Factor

This section will read in the data, do a little bit of cleaning and then factor the column we will use for predictions

```
library(RWeka)
df <- read.csv("avocado.csv")
df <- na.omit(df)
df$type <- as.factor(df$type)
str(df)
```

```
## 'data.frame':    14869 obs. of  13 variables:
##  $ Sale.ID          : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ Date             : chr  "3/5/2017 0:00" "2/5/2017 0:00" "3/5/2017 0:00"
"2/26/2017 0:00" ...
##  $ AveragePrice     : num  0.44 0.46 0.48 0.49 0.49 0.51 0.51 0.51 0.51
0.51 ...
##  $ Total.Avocados   : int  4973 1750185 4857 4726 1036815 5959 1281938 2272
1269280 1312113 ...
##  $ Small.4046       : int  224 1200633 718 253 738315 225 985040 482
1097285 1037699 ...
##  $ Extra.Large.4770: int  0 18325 0 0 11642 0 6314 0 7534 14567 ...
##  $ Large.4225       : int  4749 531227 4139 4473 286858 5734 290584 1790
164461 259847 ...
##  $ Total.Bags       : int  59085 450366 46034 39299 100892 36028 193803
14864 97565 130860 ...
##  $ Small.Bags       : int  639 113752 1385 600 70749 474 62497 123 44647
76814 ...
##  $ Large.Bags       : int  58446 330583 44649 38699 30143 35554 131306
14741 52918 54046 ...
##  $ XLarge.Bags      : int  0 6031 0 0 0 0 0 0 0 0 ...
##  $ type             : Factor w/ 2 levels "conventional",..: 2 1 2 2 1 2 1 2
1 1 ...
##  $ Cities           : chr  "Cincinnati Dayton" "Phoenix Tucson" "Detroit"
"Cincinnati Dayton" ...
```

Split into train and test

Splits the data into train and test (80/20)

```
set.seed(4444)
i <- sample(nrow(df), .8*nrow(df), replace=FALSE)
```

```
train <- df[i,]
test <- df[-i,]
```

Decision Tree Baseline

Conducts the decision tree baseline

```
library(tree)
tree1 <- tree(type~AveragePrice + Total.Bags, data=train)
summary(tree1)

##
## Classification tree:
## tree(formula = type ~ AveragePrice + Total.Bags, data = train)
## Number of terminal nodes:  6
## Residual mean deviance:  0.3354 = 3988 / 11890
## Misclassification error rate: 0.05994 = 713 / 11895
```

Random Forest

Conducts the random forest algorithm

```
library(randomForest)

## randomForest 4.7-1.1

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

set.seed(1234)
rf <- randomForest(type~AveragePrice + Total.Bags, data=train,
importance=TRUE)
rf

##
## Call:
##  randomForest(formula = type ~ AveragePrice + Total.Bags, data = train,
importance = TRUE)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 1
##
##         OOB estimate of  error rate: 6.11%
## Confusion matrix:
##              conventional organic class.error
## conventional         5570     346  0.05848546
## organic               381    5598  0.06372303
```

Testing Random Forest

Tests the random forest algorithm

```
library(mltools)
pred <- predict(rf, newdata=test, type="response")
```

```
acc_rf <- mean(pred==test$type)
mcc_rf <- mcc(factor(pred), test$type)
print(paste("accuracy=", acc_rf))

## [1] "accuracy= 0.945527908540686"

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

## [1] "mcc= 0.891038215876235"
```

XGBoost

Conducts the XGBoost Algorithm

```
library(xgboost)
train_label <- ifelse(train$type==1, 1, 0)
train_matrix <- data.matrix(train[, -31])
model <- xgboost(data=train_matrix, label=train_label,
                 nrounds=100, objective='binary:logistic')

## [1]  train-logloss:0.437559
## [2]  train-logloss:0.296370
## [3]  train-logloss:0.207400
## [4]  train-logloss:0.147873
## [5]  train-logloss:0.106679
## [6]  train-logloss:0.077567
## [7]  train-logloss:0.056706
## [8]  train-logloss:0.041617
## [9]  train-logloss:0.030629
## [10] train-logloss:0.022591
## [11] train-logloss:0.016690
## [12] train-logloss:0.012348
## [13] train-logloss:0.009148
## [14] train-logloss:0.006785
## [15] train-logloss:0.005039
## [16] train-logloss:0.003749
## [17] train-logloss:0.002794
## [18] train-logloss:0.002087
## [19] train-logloss:0.001564
## [20] train-logloss:0.001176
## [21] train-logloss:0.000889
## [22] train-logloss:0.000676
## [23] train-logloss:0.000517
## [24] train-logloss:0.000400
## [25] train-logloss:0.000312
## [26] train-logloss:0.000246
## [27] train-logloss:0.000197
## [28] train-logloss:0.000160
## [29] train-logloss:0.000131
## [30] train-logloss:0.000109
## [31] train-logloss:0.000092
```

```
## [32] train-logloss:0.000079
## [33] train-logloss:0.000079
## [34] train-logloss:0.000079
## [35] train-logloss:0.000079
## [36] train-logloss:0.000079
## [37] train-logloss:0.000079
## [38] train-logloss:0.000079
## [39] train-logloss:0.000079
## [40] train-logloss:0.000079
## [41] train-logloss:0.000079
## [42] train-logloss:0.000079
## [43] train-logloss:0.000079
## [44] train-logloss:0.000079
## [45] train-logloss:0.000079
## [46] train-logloss:0.000079
## [47] train-logloss:0.000079
## [48] train-logloss:0.000079
## [49] train-logloss:0.000079
## [50] train-logloss:0.000079
## [51] train-logloss:0.000079
## [52] train-logloss:0.000079
## [53] train-logloss:0.000079
## [54] train-logloss:0.000079
## [55] train-logloss:0.000079
## [56] train-logloss:0.000079
## [57] train-logloss:0.000079
## [58] train-logloss:0.000079
## [59] train-logloss:0.000079
## [60] train-logloss:0.000079
## [61] train-logloss:0.000079
## [62] train-logloss:0.000079
## [63] train-logloss:0.000079
## [64] train-logloss:0.000079
## [65] train-logloss:0.000079
## [66] train-logloss:0.000079
## [67] train-logloss:0.000079
## [68] train-logloss:0.000079
## [69] train-logloss:0.000079
## [70] train-logloss:0.000079
## [71] train-logloss:0.000079
## [72] train-logloss:0.000079
## [73] train-logloss:0.000079
## [74] train-logloss:0.000079
## [75] train-logloss:0.000079
## [76] train-logloss:0.000079
## [77] train-logloss:0.000079
## [78] train-logloss:0.000079
## [79] train-logloss:0.000079
## [80] train-logloss:0.000079
## [81] train-logloss:0.000079
```

```
## [82] train-logloss:0.000079
## [83] train-logloss:0.000079
## [84] train-logloss:0.000079
## [85] train-logloss:0.000079
## [86] train-logloss:0.000079
## [87] train-logloss:0.000079
## [88] train-logloss:0.000079
## [89] train-logloss:0.000079
## [90] train-logloss:0.000079
## [91] train-logloss:0.000079
## [92] train-logloss:0.000079
## [93] train-logloss:0.000079
## [94] train-logloss:0.000079
## [95] train-logloss:0.000079
## [96] train-logloss:0.000079
## [97] train-logloss:0.000079
## [98] train-logloss:0.000079
## [99] train-logloss:0.000079
## [100]    train-logloss:0.000079
```

Testing XGBoost

Tests the XGBoost algorithm

```
test_label <- ifelse(test$type==1, 1, 0)
test_matrix <- data.matrix(test[, -31])

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

acc_xg <- mean(pred==test_label)
mcc_xg <- mcc(pred, test_label)
print(paste("accuracy=", acc_xg))

## [1] "accuracy= 1"

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

## [1] "mcc= 0"
```

Adabag

Conducts the adabag 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

adab1 <- boosting(type~AveragePrice + Total.Bags, data=train, boos=TRUE,
mfinal=20, coeflearn='Breiman')
summary(adab1)

##            Length Class   Mode
## formula        3  formula call
## trees         20  -none-  list
## weights       20  -none-  numeric
## votes      23790  -none-  numeric
## prob       23790  -none-  numeric
## class      11895  -none-  character
## importance     2  -none-  numeric
## terms          3  terms   call
## call           6  -none-  call
```

Testing Adabag

Tests the adabag algorithm

```
pred <- predict(adab1, newdata=test, type="response")
acc_adabag <- mean(pred$class==test$type)
mcc_adabag <- mcc(factor(pred$class), test$type)
print(paste("accuracy=", acc_adabag))

## [1] "accuracy= 0.945527908540686"

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

## [1] "mcc= 0.890999687301299"
```

Analysis

These algorithms have always interested me, as boosting weak learners to become strong learners is a complex concept to me. To make a predictor better by simply reducing the errors it makes seems like such a simple task with an insanely difficult solution, and yet it was done. We can see that three algorithms did have an impact on the accuracy and MCC.

For the Random Forest, the accuracy was very high at .945527 and the MCC was 0.891038. However, the most interesting boosting algorithm had to be XGBoost, as it came out with an accuracy of 1 and an MCC of 0. After reading in the textbook about how unique this algorithm was in regard to run time and results, I had some doubts, but these results served to prove me wrong. This boosting algorithm helped improve the results from the decision tree by almost 6% (taken from the misclassification error rate in the decision tree section). On the Adabag testing, we can see that the accuracy stayed the same at 0.945527, but the MCC did decrease by a little bit compared to the random forest algorithm at 0.890999. The run times for each algorithm were decently quick all things considered. The random forest was the slowest, adabag was in the middle, but the big standout was the XGBoost algorithm. It ran extremely quickly, clocking in at 1.18 seconds. With this analysis, I can confidently say that XGBoost is the best algorithm for boosting in regard to metrics and time.