Ensemble Methods

Naomi Zilber

1 April 2023

Overview

In this notebook, I perform ensemble learning models on the data set, including random forest, bagging, Adaboost, and XGBoost. The data set used in this notebook is from this link. (https://www.kaggle.com/datasets/ahsan81/hotel-reservations-classification-dataset)

Load data

Read in the data of hotel reservations

```
df <- read.csv("Hotel_Reservations.csv", header=TRUE)
str(df)
```

```
## 'data.frame':    36275 obs. of  19 variables:
##  $ Booking_ID                          : chr  "INN00001" "INN00002" "INN00003" "INN00004" ...
##  $ no_of_adults                        : int  2 2 1 2 2 2 2 2 3 2 ...
##  $ no_of_children                      : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ no_of_weekend_nights                : int  1 2 2 0 1 0 1 1 0 0 ...
##  $ no_of_week_nights                   : int  2 3 1 2 1 2 3 3 4 5 ...
##  $ type_of_meal_plan                   : chr  "Meal Plan 1" "Not Selected" "Meal Plan 1" "Mea
l Plan 1" ...
##  $ required_car_parking_space          : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ room_type_reserved                  : chr  "Room_Type 1" "Room_Type 1" "Room_Type 1" "Room
_Type 1" ...
##  $ lead_time                           : int  224 5 1 211 48 346 34 83 121 44 ...
##  $ arrival_year                        : int  2017 2018 2018 2018 2018 2018 2017 2018 2018 20
18 ...
##  $ arrival_month                       : int  10 11 2 5 4 9 10 12 7 10 ...
##  $ arrival_date                        : int  2 6 28 20 11 13 15 26 6 18 ...
##  $ market_segment_type                 : chr  "Offline" "Online" "Online" "Online" ...
##  $ repeated_guest                      : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ no_of_previous_cancellations        : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ no_of_previous_bookings_not_canceled: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ avg_price_per_room                  : num  65 106.7 60 100 94.5 ...
##  $ no_of_special_requests              : int  0 1 0 0 0 1 1 1 1 3 ...
##  $ booking_status                      : chr  "Not_Canceled" "Not_Canceled" "Canceled" "Cance
led" ...
```

Data cleaning

Got rid of features that I don’t think will affect the target value (booking status), and converted room_type_reserved, booking_status, and repeated_guest into factors.

```
df <- df[,c(-1,-6,-7,-10,-11,-12,-13)]
df$room_type_reserved <- as.factor(df$room_type_reserved)
df$booking_status <- as.factor(df$booking_status)
df$repeated_guest <- as.factor(df$repeated_guest)
str(df)
```

```
## 'data.frame':    36275 obs. of  12 variables:
##  $ no_of_adults                        : int  2 2 1 2 2 2 2 2 3 2 ...
##  $ no_of_children                      : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ no_of_weekend_nights                : int  1 2 2 0 1 0 1 1 0 0 ...
##  $ no_of_week_nights                   : int  2 3 1 2 1 2 3 3 4 5 ...
##  $ room_type_reserved                  : Factor w/ 7 levels "Room_Type 1",..: 1 1 1 1 1 1 1 4
1 4 ...
##  $ lead_time                           : int  224 5 1 211 48 346 34 83 121 44 ...
##  $ repeated_guest                      : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
##  $ no_of_previous_cancellations        : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ no_of_previous_bookings_not_canceled: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ avg_price_per_room                  : num  65 106.7 60 100 94.5 ...
##  $ no_of_special_requests              : int  0 1 0 0 0 1 1 1 1 3 ...
##  $ booking_status                      : Factor w/ 2 levels "Canceled","Not_Canceled": 2 2 1
1 1 1 2 2 2 2 ...
```

Handle missing values

There are no NAs to handle in this data set

```
sapply(df, function(x) sum(is.na(x)==TRUE))
```

```
##                         no_of_adults                       no_of_children
##                                    0                                    0
##                 no_of_weekend_nights                    no_of_week_nights
##                                    0                                    0
##                   room_type_reserved                            lead_time
##                                    0                                    0
##                       repeated_guest         no_of_previous_cancellations
##                                    0                                    0
## no_of_previous_bookings_not_canceled                   avg_price_per_room
##                                    0                                    0
##               no_of_special_requests                       booking_status
##                                    0                                    0
```

Divide into train and test data

Divide the data to 80% train data and 20% test data

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

Random Forest

Create a random forest model. It took the algorithm roughly 35 seconds to run.

```
library(tictoc)
```

```
## Warning: package 'tictoc' was built under R version 4.2.3
```

```
library(randomForest)
```

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

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

```
set.seed(1234)
tic("random forest")
rf <- randomForest(booking_status~., data=train, importance=TRUE)
toc()
```

```
## random forest: 34.58 sec elapsed
```

```
rf
```

```
##
## Call:
##  randomForest(formula = booking_status ~ ., data = train, importance = TRUE)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 3
##
##         OOB estimate of  error rate: 13.86%
## Confusion matrix:
##              Canceled Not_Canceled class.error
## Canceled         6480         3027  0.31839697
## Not_Canceled      996        18517  0.05104289
```

Predict on the Random Forest

The accuracy is very good. The mcc is also relatively good, which shows that there is relatively strong agreement between the predictions and actual values in the random forest model.

```
library(mltools)
pred_rf <- predict(rf, newdata=test)
acc_rf <- mean(pred_rf==test$booking_status)
mcc_rf <- mcc(pred_rf, test$booking_status)

print(paste("accuracy =", acc_rf))
```

```
## [1] "accuracy = 0.860372157133012"
```

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

```
## [1] "mcc = 0.674429723482604"
```

```
confus_rf <- table(pred_rf, test$booking_status)
confus_rf
```

```
##
## pred_rf        Canceled Not_Canceled
##   Canceled         1612          247
##   Not_Canceled      766         4630
```

Bagging

I set mtry to 11 since there are 11 predictors, which results in bagging. It took the algorithm roughly 28 seconds to run, which is faster than the random forest algorithm.

```
tic("bagging")
bag <- randomForest(booking_status~., data=train, mtry=11)
toc()
```

```
## bagging: 26.82 sec elapsed
```

```
bag
```

```
##
## Call:
##  randomForest(formula = booking_status ~ ., data = train, mtry = 11)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 11
##
##         OOB estimate of  error rate: 12.64%
## Confusion matrix:
##              Canceled Not_Canceled class.error
## Canceled         7333         2174  0.22867361
## Not_Canceled     1493        18020  0.07651309
```

Predict on Bagging Model

Results are slightly better for bagging than the random forest for both the accuracy and mcc.

```
pred_bg <- predict(bag, newdata=test)
acc_bg <- mean(pred_bg==test$booking_status)
mcc_bg <- mcc(pred_bg, test$booking_status)

print(paste("accuracy =", acc_bg))
```

```
## [1] "accuracy = 0.870434183321847"
```

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

```
## [1] "mcc = 0.700657164211758"
```

```
confus_bg <- table(pred_bg, test$booking_status)
confus_bg
```

```
##
## pred_bg        Canceled Not_Canceled
##   Canceled         1800          362
##   Not_Canceled      578         4515
```

Adaboost

Boost using the adabag package and create a model. It took the algorithm roughly 25 seconds to run, which is a bit faster than the bagging algorithm, but 10 seconds faster than the random forest algorithm (a big improvement).

In the boosting() function, the boos=TRUE argument indicates that a bootstrap sample of the training data should be used, the mfinal argument indicates the number of iterations in boosting, and the coeflearn argument control the algorithm selected.

```r
library(adabag)
```

```
## Warning: package 'adabag' was built under R version 4.2.3
```

```
## 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
```

```
## Warning: package 'doParallel' was built under R version 4.2.3
```

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

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

```r
tic("adaboost")
adab1 <- boosting(booking_status~., data=train, boos=TRUE, mfinal=20, coeflearn='Breiman')
toc()
```

```
## adaboost: 25.02 sec elapsed
```

```r
summary(adab1)
```

```
##            Length Class   Mode
## formula        3  formula call
## trees         20  -none-  list
## weights       20  -none-  numeric
## votes      58040  -none-  numeric
## prob       58040  -none-  numeric
## class      29020  -none-  character
## importance    11  -none-  numeric
## terms          3  terms   call
## call           6  -none-  call
```

Predict on the Adaboost Model

The accuracy and mcc are a bit lower than those of the random forest and bagging models.

```
pred_adabag <- predict(adab1, newdata=test, type="response")
acc_adabag <- mean(pred_adabag$class==test$booking_status)
mcc_adabag <- mcc(factor(pred_adabag$class), test$booking_status)

print(paste("accuracy =", acc_adabag))
```

```
## [1] "accuracy = 0.819021364576154"
```

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

```
## [1] "mcc = 0.573447350817863"
```

XGBoost

Convert data into numeric matrices since the data needs to be processed before xgboost can be used.

```
train_label <- ifelse(as.integer(train$booking_status)==2, 1, 0)
train_matrix <- data.matrix(train[, -12])

test_label <- ifelse(as.integer(test$booking_status)==2, 1, 0)
test_matrix <- data.matrix(test[, -12])
```

Create the Model and Predict using XGBoost

Create the model using xgboost package. It took the algorithm roughly 2 seconds to run, which is by far the fastest out of all the other algorithms used in this notebook.

The accuracy and mcc are second best to the bagging model, though they are very close to the accuracy and mcc of the random forest and bagging models, so the xgboost model didn't under/outperform them by much at all.

The nrounds argument specifies the number of decision trees in the final mode.

```r
require(xgboost)
```

```
## Loading required package: xgboost
```

```
## Warning: package 'xgboost' was built under R version 4.2.3
```

```r
tic("xgboost")
model <- xgboost(data=train_matrix, label=train_label, nrounds=100, objective='binary:logistic')
```

```
## [1]  train-logloss:0.569282
## [2]  train-logloss:0.500343
## [3]  train-logloss:0.458338
## [4]  train-logloss:0.431019
## [5]  train-logloss:0.411065
## [6]  train-logloss:0.397968
## [7]  train-logloss:0.388872
## [8]  train-logloss:0.382251
## [9]  train-logloss:0.376410
## [10] train-logloss:0.371534
## [11] train-logloss:0.369152
## [12] train-logloss:0.367232
## [13] train-logloss:0.366208
## [14] train-logloss:0.361560
## [15] train-logloss:0.358742
## [16] train-logloss:0.357635
## [17] train-logloss:0.355203
## [18] train-logloss:0.354426
## [19] train-logloss:0.351400
## [20] train-logloss:0.350490
## [21] train-logloss:0.348411
## [22] train-logloss:0.347109
## [23] train-logloss:0.343117
## [24] train-logloss:0.342367
## [25] train-logloss:0.339570
## [26] train-logloss:0.336209
## [27] train-logloss:0.335586
## [28] train-logloss:0.335232
## [29] train-logloss:0.333699
## [30] train-logloss:0.332118
## [31] train-logloss:0.329487
## [32] train-logloss:0.329026
## [33] train-logloss:0.325670
## [34] train-logloss:0.323786
## [35] train-logloss:0.323449
## [36] train-logloss:0.323115
## [37] train-logloss:0.321200
## [38] train-logloss:0.319257
## [39] train-logloss:0.316865
## [40] train-logloss:0.314811
## [41] train-logloss:0.313316
## [42] train-logloss:0.311039
## [43] train-logloss:0.308071
## [44] train-logloss:0.307246
## [45] train-logloss:0.304797
## [46] train-logloss:0.302404
## [47] train-logloss:0.301466
## [48] train-logloss:0.301291
## [49] train-logloss:0.301213
## [50] train-logloss:0.300330
## [51] train-logloss:0.299491
## [52] train-logloss:0.297492
```

```
## [53] train-logloss:0.296214
## [54] train-logloss:0.295391
## [55] train-logloss:0.293770
## [56] train-logloss:0.293328
## [57] train-logloss:0.291528
## [58] train-logloss:0.290746
## [59] train-logloss:0.289378
## [60] train-logloss:0.288006
## [61] train-logloss:0.287931
## [62] train-logloss:0.287724
## [63] train-logloss:0.286740
## [64] train-logloss:0.285604
## [65] train-logloss:0.284871
## [66] train-logloss:0.283806
## [67] train-logloss:0.282744
## [68] train-logloss:0.281814
## [69] train-logloss:0.281114
## [70] train-logloss:0.280283
## [71] train-logloss:0.278230
## [72] train-logloss:0.277091
## [73] train-logloss:0.275921
## [74] train-logloss:0.274936
## [75] train-logloss:0.273440
## [76] train-logloss:0.273378
## [77] train-logloss:0.273080
## [78] train-logloss:0.272433
## [79] train-logloss:0.271887
## [80] train-logloss:0.271276
## [81] train-logloss:0.270057
## [82] train-logloss:0.269268
## [83] train-logloss:0.268227
## [84] train-logloss:0.266641
## [85] train-logloss:0.266338
## [86] train-logloss:0.264888
## [87] train-logloss:0.264471
## [88] train-logloss:0.264183
## [89] train-logloss:0.263370
## [90] train-logloss:0.262494
## [91] train-logloss:0.261730
## [92] train-logloss:0.260748
## [93] train-logloss:0.260209
## [94] train-logloss:0.258519
## [95] train-logloss:0.257854
## [96] train-logloss:0.256893
## [97] train-logloss:0.256369
## [98] train-logloss:0.255502
## [99] train-logloss:0.255160
## [100]    train-logloss:0.254127
```

```
toc()
```

```
## xgboost: 1.3 sec elapsed
```

```
probs <- predict(model, test_matrix)
pred_xg <- ifelse(probs>0.5, 1, 0)
acc_xg <- mean(pred_xg==test_label)
mcc_xg <- mcc(pred_xg, test_label)

print(paste("accuracy =", acc_xg))
```

```
## [1] "accuracy = 0.86368022053756"
```

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

```
## [1] "mcc = 0.682454761975517"
```

Evaluation

Based on the results, bagging had the highest accuracy and mcc, followed by XGBoost, random forest, and Adaboost Overall, the bagging, XGBoost, and random forest algorithms all had very close accuracy and mcc values, with only Adaboost lagging behind, relatively speaking.

When taking the speed of the algorithm into consideration, XGBoost was the fastest by far, followed by Adaboost and bagging closely after, and random forest being the slowest. This shows that even though bagging had the highest accuracy, it was much slower than XGBoost, whose accuracy was extremely close to that of the bagging model and the random forest model, so XGBoost would be much preferred over these algorithms overall. Bagging and Adaboost had similar run times, but Adaboost had a lower accuracy compared to bagging, so Adaboost being slightly faster than bagging seems to have cost it a bit of accuracy.

Overall, it seems like XGBoost performed the best out of all of these algorithms when both run time and accuracy are taken into account because the algorithm’s speed barely affected its accuracy, and its speed was substantially better than any of the other algorithms.

References

Mazidi, Karen. *Machine Learning Handbook Using R and Python*. 2nd ed., 2020.