Predicting On Base Percentage (OBP)

W. Kerney

2024

Project Overview

This project focuses on designing, building, and evaluating a model to predict a player's on-base percentage (OBP) based on other performance indicators. The MLB Hitting and Pitching stats dataset contains individual statistics for Major League Baseball players. It includes various performance metrics, such as batting averages, on-base percentages, slugging percentages, home runs, RBIs, strikeouts, walks, and more.

```r
### Step 1: Load and Explore the Dataset
install.packages("readr")
```

```
## Installing package into '/cloud/lib/x86_64-pc-linux-gnu-library/4.4'
## (as 'lib' is unspecified)
```

```r
install.packages("tidyverse")
```

```
## Installing package into '/cloud/lib/x86_64-pc-linux-gnu-library/4.4'
## (as 'lib' is unspecified)
```

```r
install.packages("dplyr")
```

```
## Installing package into '/cloud/lib/x86_64-pc-linux-gnu-library/4.4'
## (as 'lib' is unspecified)
```

```r
install.packages("janitor")
```

```
## Installing package into '/cloud/lib/x86_64-pc-linux-gnu-library/4.4'
## (as 'lib' is unspecified)
```

```r
install.packages("caret")
```

```
## Installing package into '/cloud/lib/x86_64-pc-linux-gnu-library/4.4'
## (as 'lib' is unspecified)
```

```r
install.packages("randomForest")
```

```
## Installing package into '/cloud/lib/x86_64-pc-linux-gnu-library/4.4'
## (as 'lib' is unspecified)
```

```r
install.packages("knitr")
```

```
## Installing package into '/cloud/lib/x86_64-pc-linux-gnu-library/4.4'
## (as 'lib' is unspecified)
```

```r
install.packages("Metrics")
```

```
## Installing package into '/cloud/lib/x86_64-pc-linux-gnu-library/4.4'
## (as 'lib' is unspecified)
```

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

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v purrr     1.0.2
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.5.1     v tibble    3.2.1
## v lubridate 1.9.3     v tidyr     1.3.1
```

```
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```r
library(janitor)
```

```
##
## Attaching package: 'janitor'
##
## The following objects are masked from 'package:stats':
##
##     chisq.test, fisher.test
```

```r
library(knitr)
library(Metrics)

#read in data
mlb_data <- read.csv("baseball_hitting.csv") %>%
  clean_names()

# Explore the dataset
head(mlb_data)
```

```
##   player_name position games at_bat runs hits double_2b third_baseman home_run
## 1     B Bonds       LF  2986   9847 2227 2935       601            77      762
## 2     H Aaron       RF  3298  12364 2174 3771       624            98      755
## 3      B Ruth       RF  2504   8399 2174 2873       506           136      714
## 4    A Pujols       1B  3080  11421 1914 3384       686            16      703
## 5 A Rodriguez       SS  2784  10566 2021 3115       548            31      696
## 6      W Mays       CF  2992  10881 2062 3283       523           140      660
##   run_batted_in a_walk strikeouts stolen_base caught_stealing   avg
## 1          1996   2558       1539         514             141 0.298
## 2          2297   1402       1383         240              73 0.305
## 3          2213   2062       1330         123             117 0.342
## 4          2218   1373       1404         117              43 0.296
## 5          2086   1338       2287         329              76 0.295
## 6          1903   1464       1526         338             103 0.302
##   on_base_percentage slugging_percentage on_base_plus_slugging
## 1              0.444               0.607                 1.051
## 2              0.374               0.555                 0.929
## 3              0.474               0.690                 1.164
## 4              0.374               0.544                 0.918
## 5              0.380               0.550                 0.930
## 6              0.384               0.557                 0.941
```

```r
summary(mlb_data)
```

```
##  player_name          position              games             at_bat     
##  Length:2508        Length:2508        Min.   :   2.0   Min.   :  262  
##  Class :character   Class :character   1st Qu.: 616.8   1st Qu.: 1874  
##  Mode  :character   Mode  :character   Median : 998.0   Median : 3266  
##                                        Mean   :1084.6   Mean   : 3715  
##                                        3rd Qu.:1438.2   3rd Qu.: 5106  
##                                        Max.   :3562.0   Max.   :14053  
##                                        NA's   :8        NA's   :8      
##       runs             hits           double_2b     third_baseman   
##  Min.   :  32.0   Min.   :  57.0   Min.   :  7.0   Min.   :  0.00  
##  1st Qu.: 231.8   1st Qu.: 471.2   1st Qu.: 86.0   1st Qu.:  9.00  
##  Median : 423.5   Median : 853.5   Median :154.0   Median : 20.00  
##  Mean   : 521.6   Mean   :1010.9   Mean   :181.9   Mean   : 32.33  
##  3rd Qu.: 719.2   3rd Qu.:1399.2   3rd Qu.:249.0   3rd Qu.: 42.25  
##  Max.   :2295.0   Max.   :4256.0   Max.   :792.0   Max.   :309.00  
##  NA's   :8        NA's   :8        NA's   :8       NA's   :8       
##     home_run     run_batted_in        a_walk        strikeouts       
##  Min.   : 17.0   Min.   :  37.0   Min.   :  19.0   Length:2508       
##  1st Qu.: 33.0   1st Qu.: 222.0   1st Qu.: 162.0   Class :character  
##  Median : 69.0   Median : 404.0   Median : 292.5   Mode  :character  
##  Mean   :100.6   Mean   : 494.2   Mean   : 373.0                     
##  3rd Qu.:125.5   3rd Qu.: 656.2   3rd Qu.: 486.2                     
##  Max.   :762.0   Max.   :2297.0   Max.   :2558.0                     
##  NA's   :8       NA's   :8        NA's   :8                          
##   stolen_base     caught_stealing         avg          on_base_percentage
##  Min.   :   0.0   Length:2508        Min.   :0.1230   Min.   :0.1570    
##  1st Qu.:  11.0   Class :character   1st Qu.:0.2470   1st Qu.:0.3110    
##  Median :  32.0   Mode  :character   Median :0.2620   Median :0.3300    
##  Mean   :  76.1                      Mean   :0.2633   Mean   :0.3316    
##  3rd Qu.:  89.0                      3rd Qu.:0.2780   3rd Qu.:0.3510    
##  Max.   :1406.0                      Max.   :0.3670   Max.   :0.4820    
##  NA's   :8                           NA's   :8        NA's   :8         
##  slugging_percentage on_base_plus_slugging
##  Min.   :0.1970      Min.   :0.3540       
##  1st Qu.:0.3750      1st Qu.:0.6930       
##  Median :0.4070      Median :0.7380       
##  Mean   :0.4099      Mean   :0.7417       
##  3rd Qu.:0.4410      3rd Qu.:0.7840       
##  Max.   :0.6900      Max.   :1.1640       
##  NA's   :8           NA's   :20           
```

Step 2: Data Preprocessing and Feature Engineering

Now we'll extract relevant features and create new ones to use in the model.

```r
#some feature engineering
library(dplyr)

#converting strikeouts to integers
mlb_data$strikeouts <- as.integer(mlb_data$strikeouts)
```

```
## Warning: NAs introduced by coercion
```

```r
str(mlb_data)
```

```
## 'data.frame':    2508 obs. of  18 variables:
##  $ player_name          : chr  "B Bonds" "H Aaron" "B Ruth" "A Pujols" ...
##  $ position             : chr  "LF" "RF" "RF" "1B" ...
##  $ games                : int  2986 3298 2504 3080 2784 2992 2671 2543 2354 2808 ...
##  $ at_bat               : int  9847 12364 8399 11421 10566 10881 9801 8422 8813 10006 ...
##  $ runs                 : int  2227 2174 2174 1914 2021 2062 1662 1583 1475 1829 ...
##  $ hits                 : int  2935 3771 2873 3384 3115 3283 2781 2328 2408 2943 ...
##  $ double_2b            : int  601 624 506 686 548 523 524 451 379 528 ...
##  $ third_baseman        : int  77 98 136 16 31 140 38 26 45 72 ...
##  $ home_run             : int  762 755 714 703 696 660 630 612 609 586 ...
##  $ run_batted_in        : int  1996 2297 2213 2218 2086 1903 1836 1699 1667 1812 ...
##  $ a_walk               : int  2558 1402 2062 1373 1338 1464 1312 1747 929 1420 ...
##  $ strikeouts           : int  1539 1383 1330 1404 2287 1526 1779 2548 2306 1532 ...
##  $ stolen_base          : int  514 240 123 117 329 338 184 19 234 204 ...
##  $ caught_stealing      : chr  "141" "73" "117" "43" ...
##  $ avg                  : num  0.298 0.305 0.342 0.296 0.295 0.302 0.284 0.276 0.273 0.294 ...
##  $ on_base_percentage   : num  0.444 0.374 0.474 0.374 0.38 0.384 0.37 0.402 0.344 0.389 ...
##  $ slugging_percentage  : num  0.607 0.555 0.69 0.544 0.55 0.557 0.538 0.554 0.534 0.537 ...
##  $ on_base_plus_slugging: num  1.051 0.929 1.164 0.918 0.93 ...
```

```r
#creating new features
mlb_features <- mlb_data %>%
  filter(at_bat > 0) %>%
  mutate(ISO = slugging_percentage - ((hits/ at_bat)),
         BABIP = (hits - home_run) / (at_bat - strikeouts - home_run)) %>%
select(player_name, position, on_base_percentage, ISO, BABIP, slugging_percentage, runs, run_batted_in,


# Remove rows with NA values and convert column names.

mlb_features <- mlb_features[complete.cases(mlb_features), ]
colnames(mlb_features)[c (3,6,7,8,9,10,11,12)] <- c("OBP", "SLG", "R", "RBI", "K", "BB", "HR", "OPS")

# Inspect the new dataset
head(mlb_features)
```

```
##    player_name position   OBP       ISO     BABIP   SLG    R  RBI    K   BB  HR
## 1      B Bonds       LF 0.444 0.3089397 0.2879671 0.607 2227 1996 1539 2558 762
## 2      H Aaron       RF 0.374 0.2500016 0.2949345 0.555 2174 2297 1383 1402 755
## 3       B Ruth       RF 0.474 0.3479355 0.3397325 0.690 2174 2213 1330 2062 714
## 4     A Pujols       1B 0.374 0.2477037 0.2878463 0.544 1914 2218 1404 1373 703
## 5 A Rodriguez       SS 0.380 0.2551864 0.3190030 0.550 2021 2086 2287 1338 696
## 6       W Mays       CF 0.384 0.2552814 0.3016676 0.557 2062 1903 1526 1464 660
##     OPS
## 1 1.051
## 2 0.929
## 3 1.164
## 4 0.918
## 5 0.930
## 6 0.941
```

Step 3: Splitting the Data

Now that we have confirmed that the features we created are valid. We'll split the data, creating a training set and a test set for evaluating the model.

```
# Split the data into training and test sets
library(caret)
```

```
## Loading required package: lattice

##
## Attaching package: 'caret'

## The following objects are masked from 'package:Metrics':
##
##     precision, recall

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

```
set.seed(123)
trainIndex <- createDataPartition(mlb_features$OBP, p = .8, list = FALSE)

mlb_train <- mlb_features[trainIndex, ]
mlb_test <- mlb_features[-trainIndex, ]
```

Step 4: Train a Machine Learning Model

After splitting our data, we'll use a random forest model to predict OBP.

```
#creating a random forest model to predict OBP
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
```

```
# Train the random forest model
rf_model <- randomForest(OBP ~ ISO + BABIP + SLG + R + RBI + K + BB + HR, data = mlb_train, ntree = 100

# Check the model's summary
print(rf_model)
```

```
##
## Call:
##  randomForest(formula = OBP ~ ISO + BABIP + SLG + R + RBI + K +      BB + HR, data = mlb_train, ntre
##                Type of random forest: regression
##                      Number of trees: 100
## No. of variables tried at each split: 2
```

```
##
##           Mean of squared residuals: 0.0001547024
##                     % Var explained: 83.93
```

Step 5: Evaluate the Model's Performance

From here we'll assess the model's performance using the test data and visualize the results.

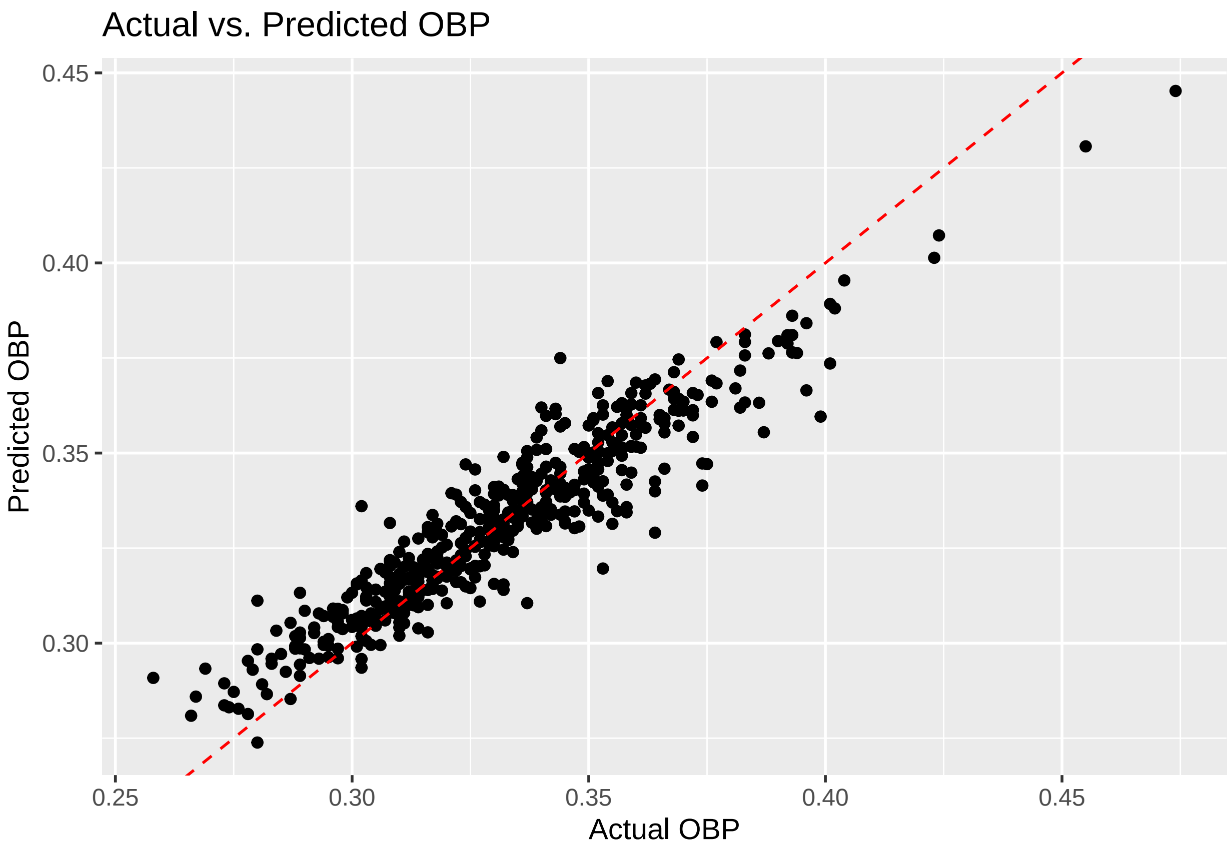
```r
#evaluating model performance
# Make predictions on the test set
obp_predictions <- predict(rf_model, mlb_test)
rmse(predict(rf_model, mlb_test), mlb_test$OBP)
```

```
## [1] 0.01065553
```

```r
# Calculate the R-squared value
rsq <- cor(mlb_test$OBP, obp_predictions)^2
print(paste("R-squared:", round(rsq, 2)))
```

```
## [1] "R-squared: 0.88"
```

```r
# Visualize predicted vs. actual OBP
library(ggplot2)

ggplot(mlb_test, aes(x = OBP, y = obp_predictions)) +
  geom_point() +
  geom_abline(slope = 1, intercept = 0, linetype = "dashed", color = "red") +
  labs(
    title = "Actual vs. Predicted OBP",
    x = "Actual OBP",
    y = "Predicted OBP"
  )
```



Tuning Model

The Mean of squared residuals and % Var explained lets us know we do not have an overfit model, and evaluating the rmse we have our base error so now we will use caret to do some tuning.

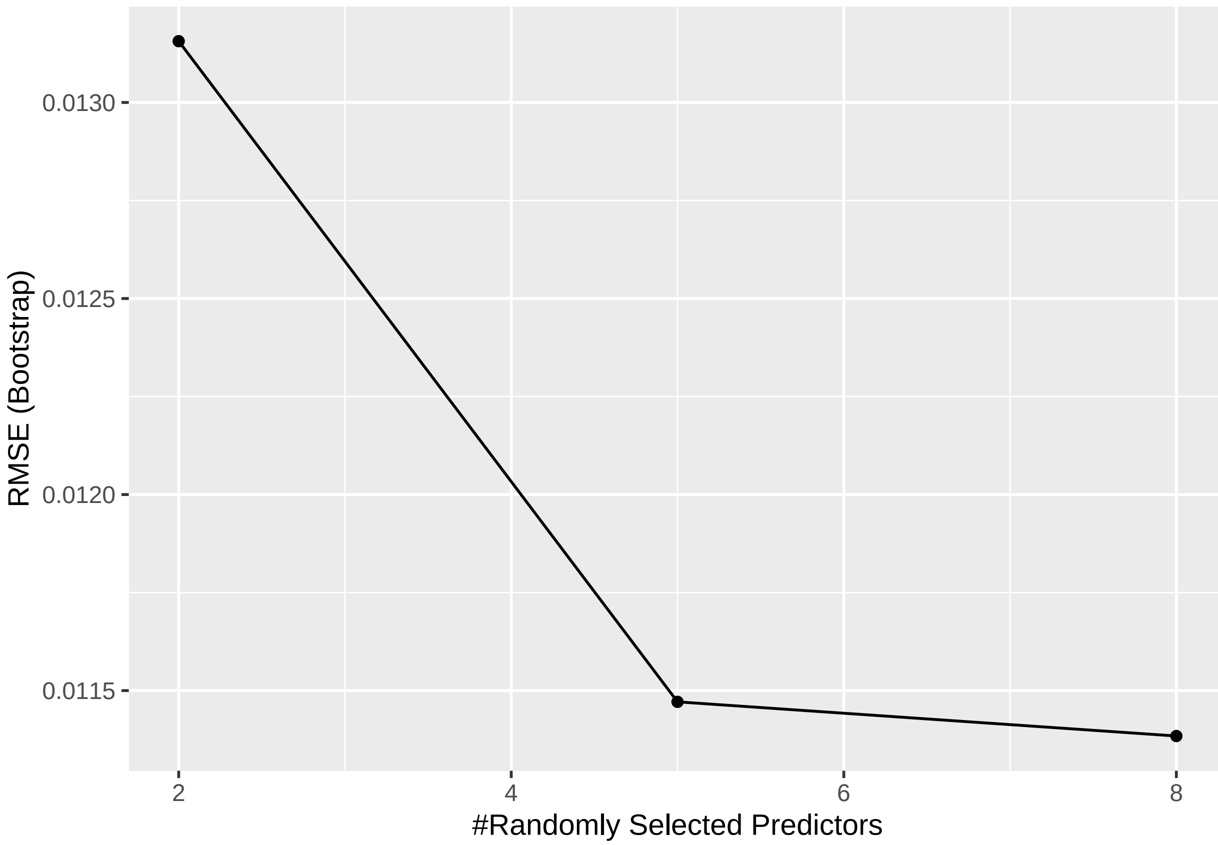
```
# use caret to pick a value for mtry

tuned_model <- train(OBP ~ ISO + BABIP + SLG + R + RBI + K + BB + HR ,data = mlb_train, ntree = 50, #nu
                     method = "rf")

print(tuned_model)
```

```
## Random Forest
##
## 1992 samples
##    8 predictor
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 1992, 1992, 1992, 1992, 1992, 1992, ...
## Resampling results across tuning parameters:
##
##   mtry  RMSE        Rsquared   MAE
##   2     0.01315634  0.8484628  0.009724142
##   5     0.01147130  0.8792759  0.008347823
##   8     0.01138418  0.8757369  0.008278651
##
## RMSE was used to select the optimal model using the smallest value.
```

```
## The final value used for the model was mtry = 8.
```

As we can see, caret picked 8 as the best mtry value, letting us know that the model was most accurate when we use 8 variables for prediction. We can see this very clearly if we plot the rmse for each of the values it tried by passing our tuned model right to ggplot.

```r
# plot the rmse for various possible training values
ggplot(tuned_model)
```



Comparing the Models

From here we will check the model against the test data.

```r
#Checking original against tuned model
print("base model rmse:")
```

```
## [1] "base model rmse:"
```

```r
print(rmse(predict(rf_model, mlb_test), mlb_test$OBP))
```

```
## [1] 0.01065553
```

```r
print("tuned model rmse:")
```

```
## [1] "tuned model rmse:"
```

```r
print(rmse(predict(tuned_model$finalModel, mlb_test), mlb_test$OBP))
```

```
## [1] 0.009343885
```

We can also compare what each model selected as the most important features. In this case, we are going to look at the top five features for each model. In the plots, 'IncNodePurity' is a measure of how important each feature is. A larger value means that feature was more important.

```r
# plot both plots at once
par(mfrow = c(1,2))

varImpPlot(rf_model, n.var = 5)
varImpPlot(tuned_model$finalModel, n.var = 5)
```