MLB Batted Ball Prediction Modeling

Jon Anderson

9/12/2019

Introduction

Major League Baseball (MLB) makes their pitch-by-pitch data set available to the public. This data set contains 85 different columns of data about every pitch thrown in every MLB game. In this study we will look at some data on batted balls to see if we can predict the outcome of that event (hit or not a hit).

I have done this same thing with various algorithms, but this example uses Random Forest. Simply stated, this algorithm takes three inputs (batted ball velocity, batted ball launch angle, and batted ball direction) and predicts if the ball will go for a hit (a 1), or if the ball will end up in an out (a 0).

Define functions

- The colFilter function brings in a dataset and a list of column names and filters the data down to just the requested columns
- The nullFilter goes through each column we are using and gets rid of all rows that have a null value in those columns
- shuffleData simply shuffles the order of the rows
- splitTrainTest will divide any data frame into two different data sets, one for training a model and one for testing
- The colFormat function will convert our columns into our desired data type
- The sumPredictions format summarizes our model after running it on a test data set. It will show us how many of each prediction it make (it should predict about 30% hits (1's) and 70% outs (0's), because that is the hit to out ratio in baseball

```r
colFilter <- function(data, columns) {
  keeps <- columns
  data <- data[keeps]
  return(data)
}

nullFilter <- function(data) {
  data <- na.omit(data)
  data <- data[!data$launch_angle=="null",]
  data <- data[!data$launch_speed=="null",]
  data <- data[!data$hit_distance_sc=="null",]
  data <- data[!data$if_fielding_alignment=="null",]
  data <- data[!data$of_fielding_alignment=="null",]
  data <- data[!data$babip_value=="null",]
  data <- data[!data$hit_location=="null",]
  return(data)
}

shuffleData <- function(data) {
  data <- data[sample(nrow(data)),]
  return(data)
}
```

```r
splitTestTrain <- function(data, rows, pct) {
  data <- data[1:rows,]
  smp_size <- floor(pct * nrow(data))
  set.seed(123)
  train_ind <- sample(seq_len(nrow(data)), size = smp_size)
  train <- data[train_ind, ]
  test <- data[-train_ind, ]
  return(list(train,test))
}

colFormat <- function(data) {
  data$launch_speed <- factor(data$launch_speed)
  data$launch_angle <- factor(data$launch_angle)
  data$hit_distance_sc <- factor(data$hit_distance_sc)

  data$launch_speed <- as.numeric(as.character(data$launch_speed))
  data$launch_angle <- as.numeric(as.character(data$launch_angle))
  data$hit_distance_sc <- as.numeric(as.character(data$hit_distance_sc))
  data$hit_location <- as.factor(factor(data$hit_location))
  data$babip_value <- as.factor(factor(data$babip_value))
  return(data)
}

sumPredictions <- function(data) {
  data$right <- ifelse(data$babip_value==data$pred,1,0)
  data$right <- as.numeric(as.character(data$right))
  accuracy_rate <- sum(data$right) / dim(data)[1]
  print(paste("Accuracy Rate",accuracy_rate))

  print("Actual Results:")
  real_table <- table(data$babip_value)
  print(real_table)
  real_pct <- real_table[2] / (real_table[1] + real_table[2])
  print(paste("Percent of 1's: ", real_pct ))

  print("Predicted Results:")
  pred_table <- table(data$pred)
  print(pred_table)
  pred_pct <- pred_table[2] / (pred_table[1] + pred_table[2])
  print(paste("Percent of 1's: ", pred_pct))
}
```

Prepare the data

- Load the data and then use another data set to merge in the hitter names. Since the baseball savant data set identifies each hitter by an identifying number, if we want their names we have to use a lookup table to merge in those names.
- Choose and format columns and then make our train and test sets

```r
data <- read.csv("g:\\sports\\flb\\savant\\data\\2019pitches.csv")
lookup <- read.csv("g:\\sports\\flb\\savant\\data\\idlookup.csv")
data <- merge(data, lookup, by="batter")
```

```r
keeps <- c("launch_speed", "launch_angle", "hit_distance_sc", "hit_location",
           "if_fielding_alignment", "of_fielding_alignment", "babip_value")
df <- colFilter(data, keeps)

# Remove nulls
df <- nullFilter(df)
df <- colFormat(df)

# Shuffle and split into train and test
df <- shuffleData(df)
train_test <- splitTestTrain(df, 40000, .75)
train <- as.data.frame(train_test[1])
test <- as.data.frame(train_test[2])

# Get these into the correct data types so the model can handle them
train <- colFormat(train)
test <- colFormat(test)
train <- droplevels(train)
test <- droplevels(test)
```

Describe Our Data

Let's learn a little more about our data with some visualizations.

Here is a snapshot of the data frame we are working with, each of these rows represents one ball put in play with the launch speed, the launch angle, the direction the ball was hit, the distance it was hit, the alignment of the fielders, and the binary babip_value column showing if it was a hit or not.

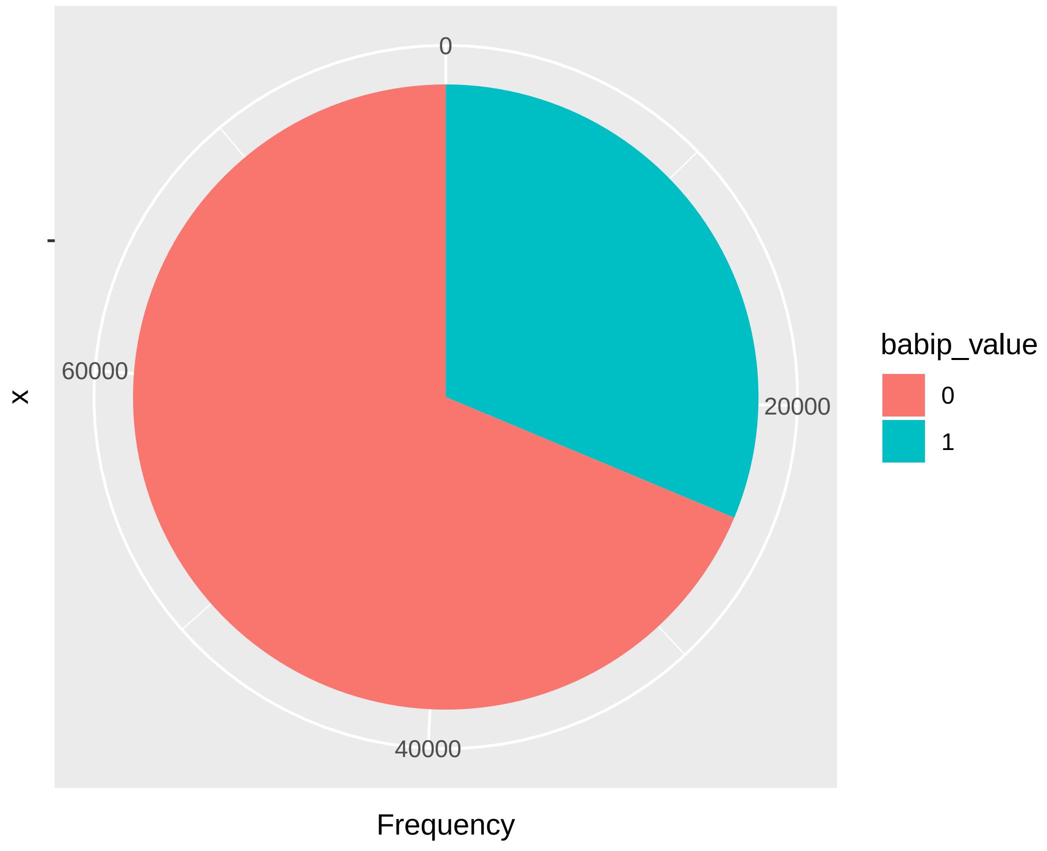
```r
print(head(df,5))
```

```
##        launch_speed launch_angle hit_distance_sc hit_location
## 421142         92.9            3              77            4
## 513676        105.8            1              51            5
## 49691          94.7          -17              10            4
## 315744         92.0           41             308            9
## 151766        100.5            7             112            9
##        if_fielding_alignment of_fielding_alignment babip_value
## 421142         Infield shift              Standard           0
## 513676         Infield shift              Standard           0
## 49691               Standard             Strategic           0
## 315744         Infield shift              Standard           0
## 151766         Infield shift              Standard           1
```

Since we are predicting outs (0's) and hits (1's), let's get a feel for how often these two outcomes happen in real life. We see about 70% outs and 30% hits in this pie graph:

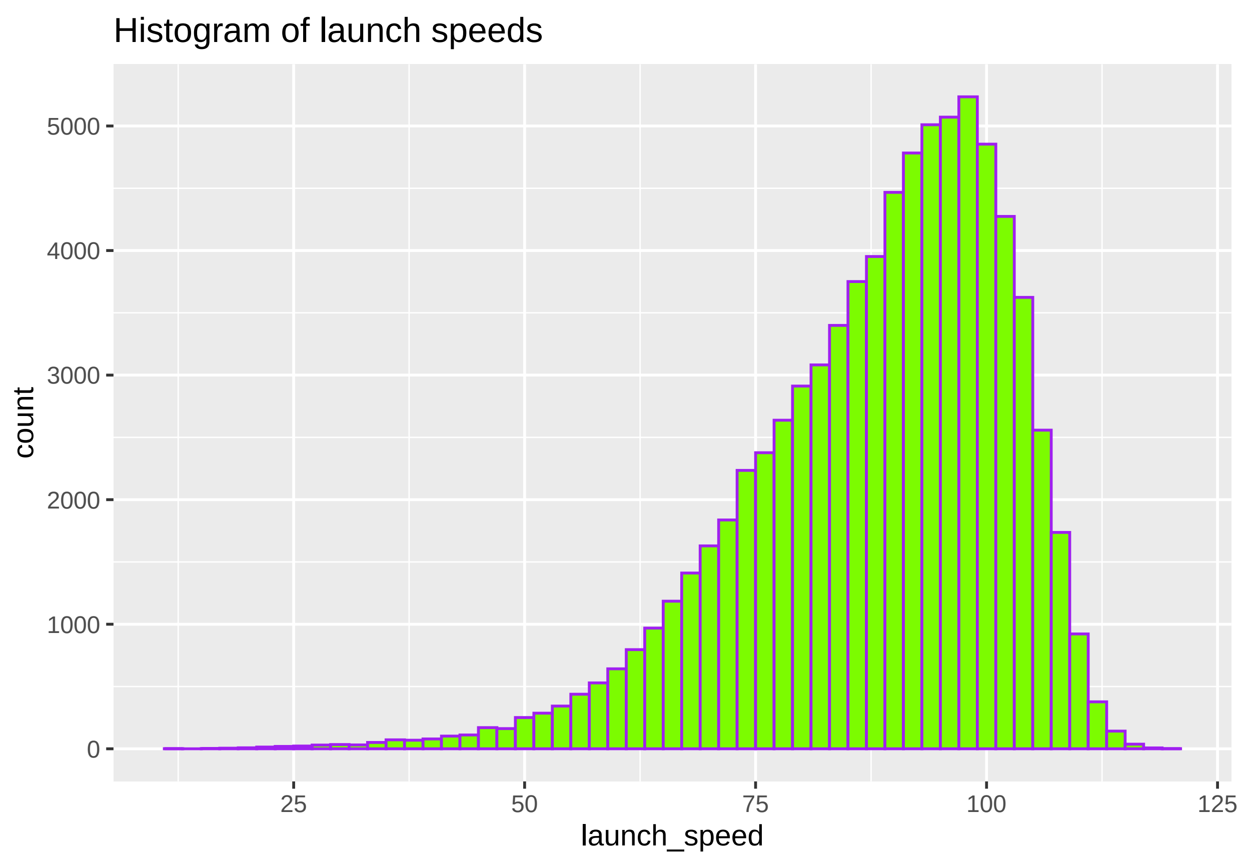
```r
babipfreq <- as.data.frame(table(df$babip_value))
colnames(babipfreq)<-c("babip_value", "Frequency")

bp <- ggplot(babipfreq, aes(x="", y=Frequency, fill=babip_value)) +
geom_bar(width = 1, stat = "identity")
```

```
pie <- bp + coord_polar("y", start=0)
plot(pie)
```

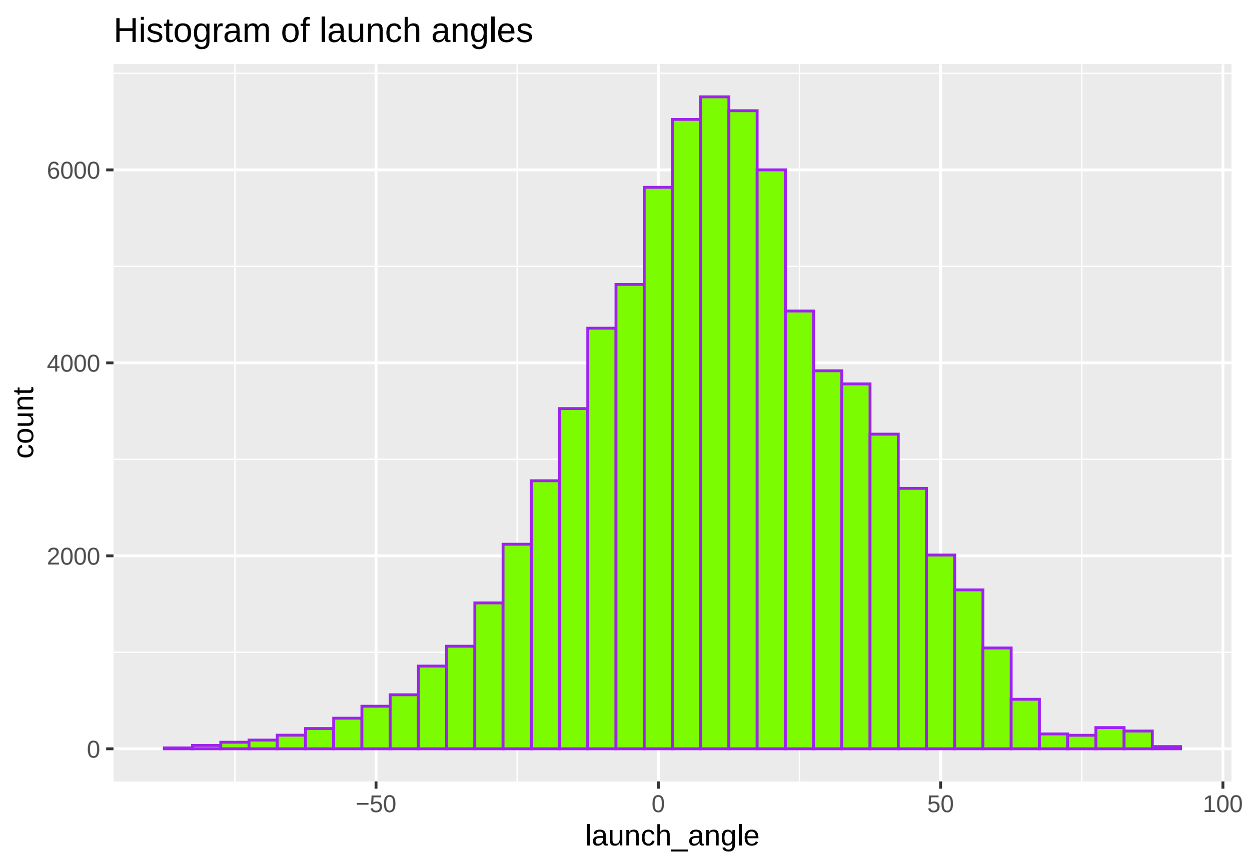


We see from these histograms that our launch speeds are most frequently in the 90's and our launch angle's are most frequently in the 20's.

```
lsplot <- ggplot(df, aes(x=launch_speed)) +
  geom_histogram(binwidth=2, color="purple", fill="lawngreen") +
  ggtitle("Histogram of launch speeds")
plot(lsplot)
```



```
laplot <- ggplot(df, aes(x=launch_angle)) +
  geom_histogram(binwidth=5, color="purple", fill="lawngreen") +
  ggtitle("Histogram of launch angles")
plot(laplot)
```



Random Forest Model

Before publishing this, I messed around with different inputs to the model. The ntree parameter tells the algorithm how many trees to grow, I achieved the best accuracy when I really raised that number up. The mtry parameter specifies the number of variables randomly sampled as candidates at each split. There wasn't much difference in accuracy by changing this parameter, so I settled on three. Since this is a classification problem and not a regression, I used 1 for the nodesize parameter.

```
# Random Forest with LS + LA + hit location
rf <- randomForest(babip_value ~ launch_speed + launch_angle + hit_location,
                   data=train, ntree=150, mtry=3, nodesize=1)
test$pred <- predict(rf, test)
sumPredictions(test)
```

```
## [1] "Accuracy Rate 0.88"
## [1] "Actual Results:"
##
##    0    1
## 6922 3078
## [1] "Percent of 1's:  0.3078"
## [1] "Predicted Results:"
##
##    0    1
## 7190 2810
## [1] "Percent of 1's:  0.281"
```

We have achieved 88% accuracy while predicting a hit 31% of the time. Pretty solid results.

Real World Usage?

I would imagine a baseball team could use something like this to go through each of their players and see how many hits they were predicted for in the last season, and then compare it to how many they actually got. There is a huge element of luck to baseball, for example you can hit a ball really well but just right at a fielder and that will go for an out, but if you hit the ball hard more often you will expect to get more hits. This model could help show which hitters should have had more hits than they actually did, and vice versa. You then may be able to identify a value player to go acquire or give more playing time to.