Baseball Strike Zone Analysis

Jacob Kramp

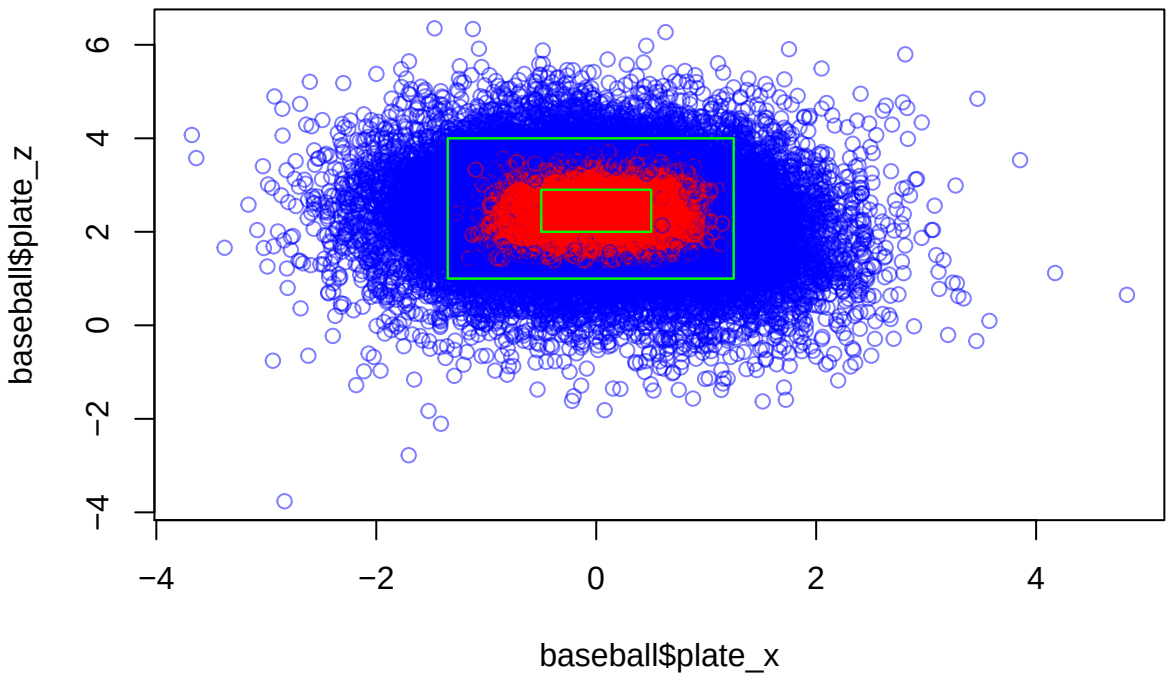
8/8/2021

Introduction

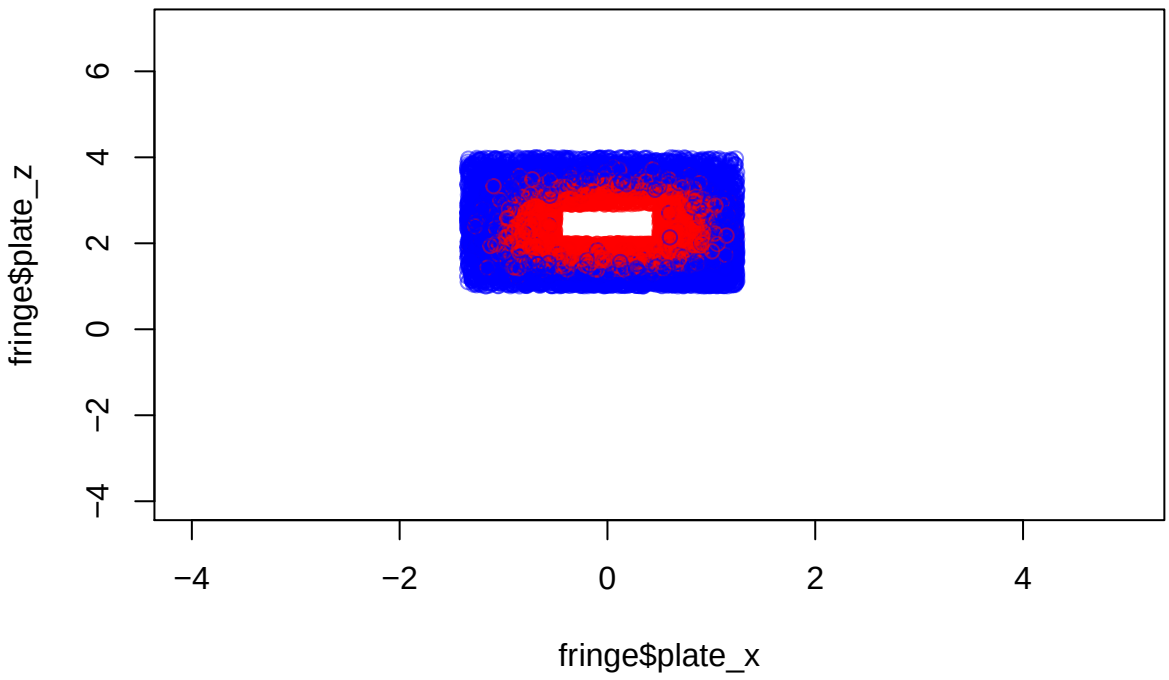
We have been asked to evaluate if the probabaility of an umpire calling a strike is impacted by the number of strikes and balls that currently have been called on the batter. The fringe zone of a pitch is the area of the batter's box in which the call of the pitch varies the most. Obviously the location of the pitch is going to be important no matter which model we decide to use to model this probabaility, but we'd like to evaluate whether or not the number of balls and strikes are also important.

Data Visualization

We'll begin by creating a plot to visualize the call of a pitch based on the location. We'd like to get an idea of where the firnge zone actually is so that we can subset our data based on this assumption.



The fringe zone is indicated by the area between the above two rectangles shown in the plot.The call of the pitch seems to vary in this area. The following is a visual of our subset of data to model.



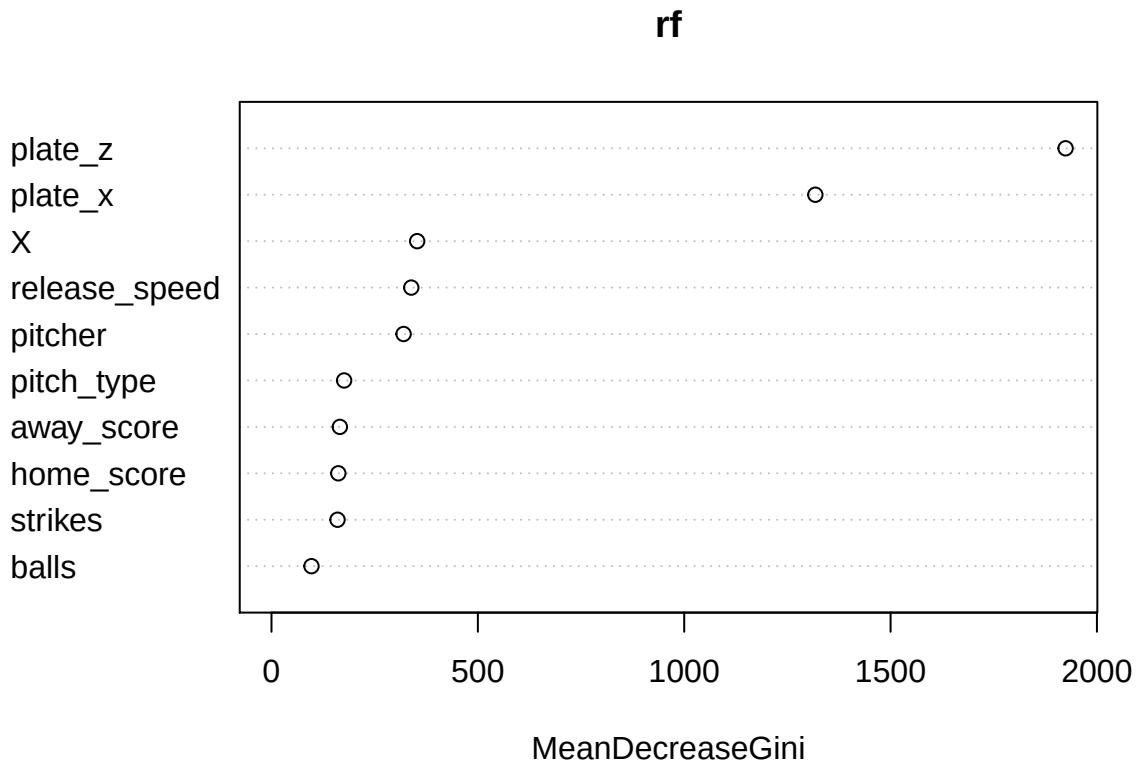
Modeling the data using Random Forests

Given that the pitch being called a strike is not linearly related to the log odds of the call, a logistic regression model would not be a suitable choice. We will instead use a random forest since it tends to be a more powerful non-parametric model compared to a single decision tree for instance. We'll then evaluate the contribution of each variable to node impurity as a subjective measure of variable importance. The data will be split into two sets: a training set to train the model and a test set to evaluate the performance of the model. We obtain the following confusion matrix for predictions made by the model.

```
##                 p.rf
##                  ball called_strike
##   ball            571            84
##   called_strike    69           441
```

The model seems to be performing relatively well, but confusion matrices don't contribute weight to mistakes made by the model based on the severity of the mistake according to the probabailty associated with each prediction. The log likelihood would be a more accurate and trustworthy evaluation of this classification model. By calculating the log-likelihood, we obtain an estimated -4288.9377841. We'll keep this number in mind after we use another model to evaluate variable importance. Before that, we can plot a chart of severity of impact on node impurity of each variable and use that as our measure of importance according to the random forest.

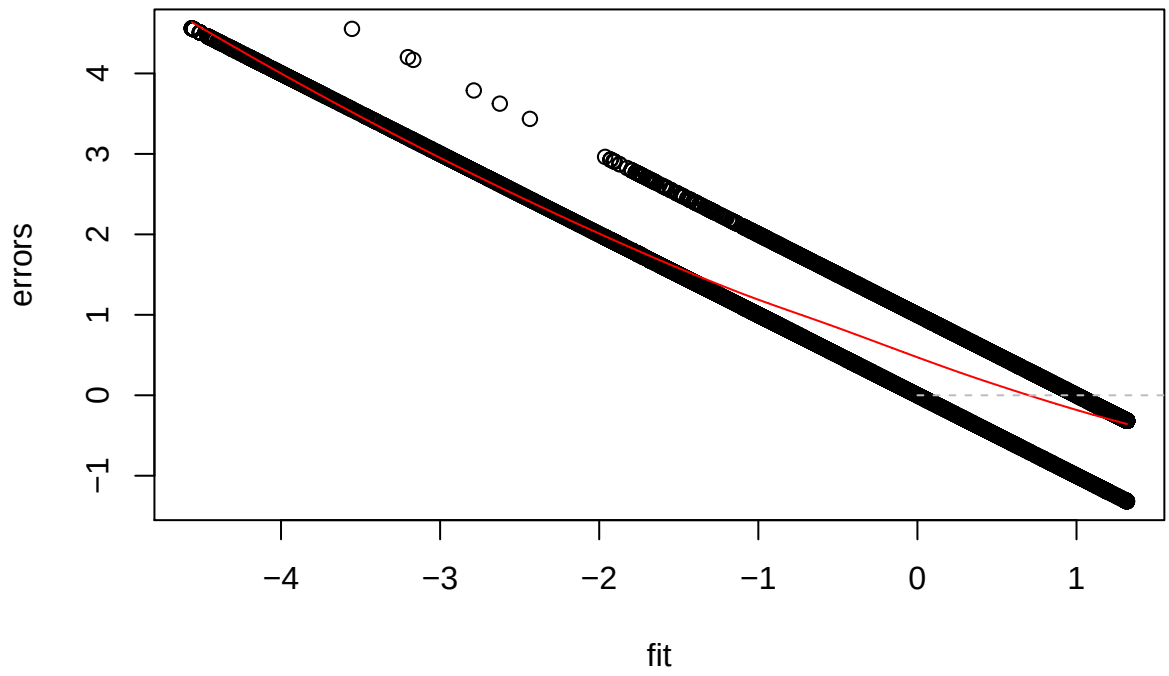
```
##                MeanDecreaseGini
## X                     352.94643
## pitch_type            176.00663
## release_speed         338.71199
## pitcher               319.99070
## balls                  96.83146
## strikes               160.00450
## plate_x              1317.71072
## plate_z              1924.04472
## home_score            161.94757
## away_score            165.67265
```



As it turns out, the number of strikes and balls called by the umpire at the time of the pitch seems to be the least important set of covariates. This is interesting and it may be more reassuring to evalulate the result using a different model.

Gradient Boosting Trees

This method is known for having an edge in performance over just using random forests. The model is similar, but besides leveraging several weaker models to create a well performing model, it also exercises an optimization algorithm on a suitable cost function to maximize performance.



This diagnostic plot is definitely not what we'd like to see.It's underpredicting very often and must not be performing well. There may be some hyper parameter tuning that we could perform to improve this performance, but that is computationally expensive and time consuming. We will assume for the sake of this paper that this is the best fit we can get. We'll compare the performance of this model to the random forest by comparing the calculated log likelihood.

This model is showing a very high log lieklihood of -4412.2900419, which is indicating a performance that is significantly worse than the previous random forest classification model. We will accept the results of the random forest classification model.

Conclusion

Although our instinct is that the number of balls and strikes are possibly a high contributor to the call made by the umpire, our most suitable model is showing that they're among the lowest contributors to the performance of our ability to predict the outcome. This indicates that covariates such as the speed of the pitch and the type of pitch thrown are actually much more important. Our conclusion is that the count of balls and strikes perhaps shouldn't be considered as important

Index (Code)

```r
library(data.table)
library(tidyverse)
library(dummies)
library(randomForest)

baseball <- read.csv('baseball.csv')

baseball$balls <- factor(baseball$balls)
baseball$strikes <- factor(baseball$strikes)
baseball <- baseball %>% mutate(color = case_when(
  description == 'ball' ~ 'blue',
  description == 'called_strike' ~ 'red'))

#Visualize the fringe.
plot(baseball$plate_x,baseball$plate_z,col = alpha(baseball$color,.5))
rect(xleft = c(-1.35,-1.35),xright = c(1.25,1.25),ytop = c(4,4),
     ybottom = c(1,1),border = 'green')
rect(xleft = c(-.5,-.5),xright = c(.5,.5),ytop = c(2.9,2.9),
     ybottom = c(2,2),border = 'green')

#Subset our data to the fringe data visualized above.

fringe <- baseball %>% filter(
  !(plate_x > -.5 & plate_x < .5 & plate_z < 2.9 & plate_z > 2) &
    !((plate_x < -1.35 | plate_x > 1.25) | (plate_z > 4 |
                                              plate_z < 1)))
plot(fringe$plate_x,fringe$plate_z,col = alpha(fringe$color,.5),
     xlim = c(-4,5),ylim = c(-4,7))


#Drop color to not mess with models
fringe <- fringe[,-grep('color',colnames(fringe))]
#Since it's obvious that the log odds are not linearly related to say,
#the x positon and z position, we will not
#Use glm. We will instead use random forrests
fringe <- fringe %>%
  mutate_if(is.character, as.factor)

#Create some test and training data
test <- sample(nrow(fringe),size = floor(.1*nrow(fringe)))
fringe.test <- fringe[test,]
fringe.train <- fringe[-test,]

#Train a random forest model
rf <- randomForest(description ~., data = fringe.train,ntree=100,
                   mtry=2,control=rpart.control(minsplit=10,cp=.05))
#make some predictions
p.rf <- predict(rf,fringe.test,type = 'class')
table(fringe.test$description,p.rf)

#calculate the log like
classes <- list(c(1:2330),
                c(2331:4661),
```

```
                c(4662:6992),
                c(6993:9322),
                c(9322:11653))
pb <- txtProgressBar(min = 0, max = length(classes), style = 3)
actual <- fringe %>%
  mutate(call = case_when(
    description == 'ball' ~ 0,
    description == 'called_strike' ~ 1))

results = lapply(classes,function(r){
  rf <- randomForest(description ~., data = fringe[-r,],ntree=100,
                     mtry=2,control=rpart.control(minsplit=10,cp=.05))
  cv.p <- predict(rf,fringe[r,],type = 'prob')
  data.frame(actual = actual[r,'call'],cv.p = cv.p[,'called_strike'])
  # update progress bar
 # setTxtProgressBar(pb, which(classes == r))
})

results <- Reduce(rbind,results)
results[which(results$cv.p == 1),'cv.p'] <- .9999
results[which(results$cv.p == 0),'cv.p'] <- .0001

LL = sum(results$actual*log(results$cv.p) +
           (1-results$actual)*log(1-results$cv.p))

#Maybe create a visual showing how the likelihood of a called strike increases
importance(rf,type=2)
varImpPlot(rf)


library(gbm)
data = na.omit(fringe)
gbm.model = gbm(call~.,data=actual[,-grep('description',colnames(actual))],
                n.trees=1000,distribution="bernoulli",shrinkage=.01,cv.folds=5)


fit = gbm.model$cv.fitted
errors = actual$call - fit

#Create residual vs. fit plot.

plot(fit,errors)
loess.line = loess(errors~fit)
lines(loess.line$x[order(loess.line$x)],loess.line$fitted[order(loess.line$x)],col=2)
lines(c(0,200),c(0,0),lty=2,col="grey")

#calculate the log like
classes <- list(c(1:2330),
                c(2331:4661),
                c(4662:6992),
                c(6993:9322),
                c(9322:11653))
```

```
results = lapply(classes,function(r){
  rf <- gbm(call~.,data=actual[-r,-grep('description',colnames(actual))],
            n.trees=1000,distribution="bernoulli",shrinkage=.01,cv.folds=5)
  cv.p <- predict(rf,fringe[r,],type = 'response')
  data.frame(actual = actual[r,'call'],cv.p = cv.p)
  # update progress bar
  # setTxtProgressBar(pb, which(classes == r))
})

results <- Reduce(rbind,results)
results[which(results$cv.p == 1),'cv.p'] <- .9999
results[which(results$cv.p == 0),'cv.p'] <- .0001

LL = sum(results$actual*log(results$cv.p) + (1-results$actual)*log(1-results$cv.p))
```