A Machine Learning Approach to MLB Catcher Framing

Nathan Hemenway and Matthew Boyd

12/10/2021

Introduction

- “Catcher framing is the art of a catcher receiving a pitch in a way that makes it more likely for an umpire to call it a strike – whether that’s turning a borderline ball into a strike, or not losing a strike to a ball due to poor framing.” - MLB.com Glossary

Motivation

- Baseball catchers can influence the call of a ball or strike on how they catch the ball
- Some catchers are better than others at this skill
- Baseball teams are aware of this and are acquiring players good at this skill to win more games
- We want to quantify the best catcher's at framing for the 2021 season
- There are several factors that influence whether a pitch will be a strike or ball
- Catchers getting more strikes translates to more outs, and fewer runs for the opposing team

Data

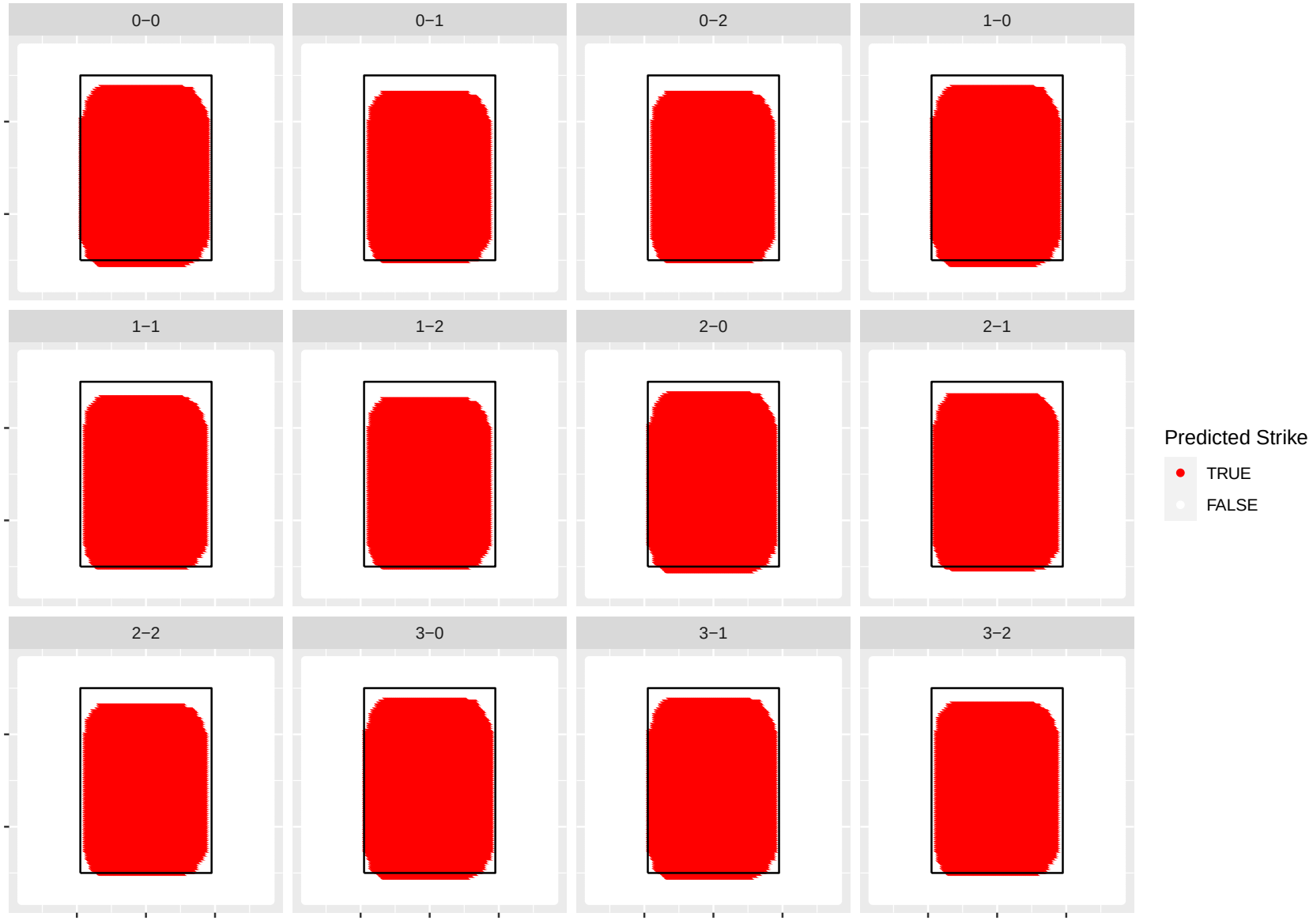
- 2021 pitch data scraped from Baseball Savant through baseballr package
- ~700,000 rows (each for a single pitch)
- Wanted to look at pitches that were not swung at by the batter (called strike or ball)
- ~350,000 rows remain

Variables

- We used:
- Pitch type and pitch release speed, position, and spin rate
- Whether or not the pitcher and batter are right or left handed
- Count, number of outs during the at-bat, and inning number
- Where the pitch landed
- Whether the game was played home or away
- How tall the batter is

Example

- Strike Probability by Location and Count



Logistic Regression Model

```
##          true
## pred          0      1
##   ball   112462  52245
##   strike   5489   5750
```

```
## [1] 0.6718652
```

- The variables with significant results were:
- Pitch type, release speed, z release position
- Whether the batter is a lefty/righty
- Count
- Where the pitch landed
- Most of the innings
- Batter height
- Whether it was a home game

Ridge Regression

```
##      true
## pred      0      1
##    0 113370  52991
##    1   4634   4951
```

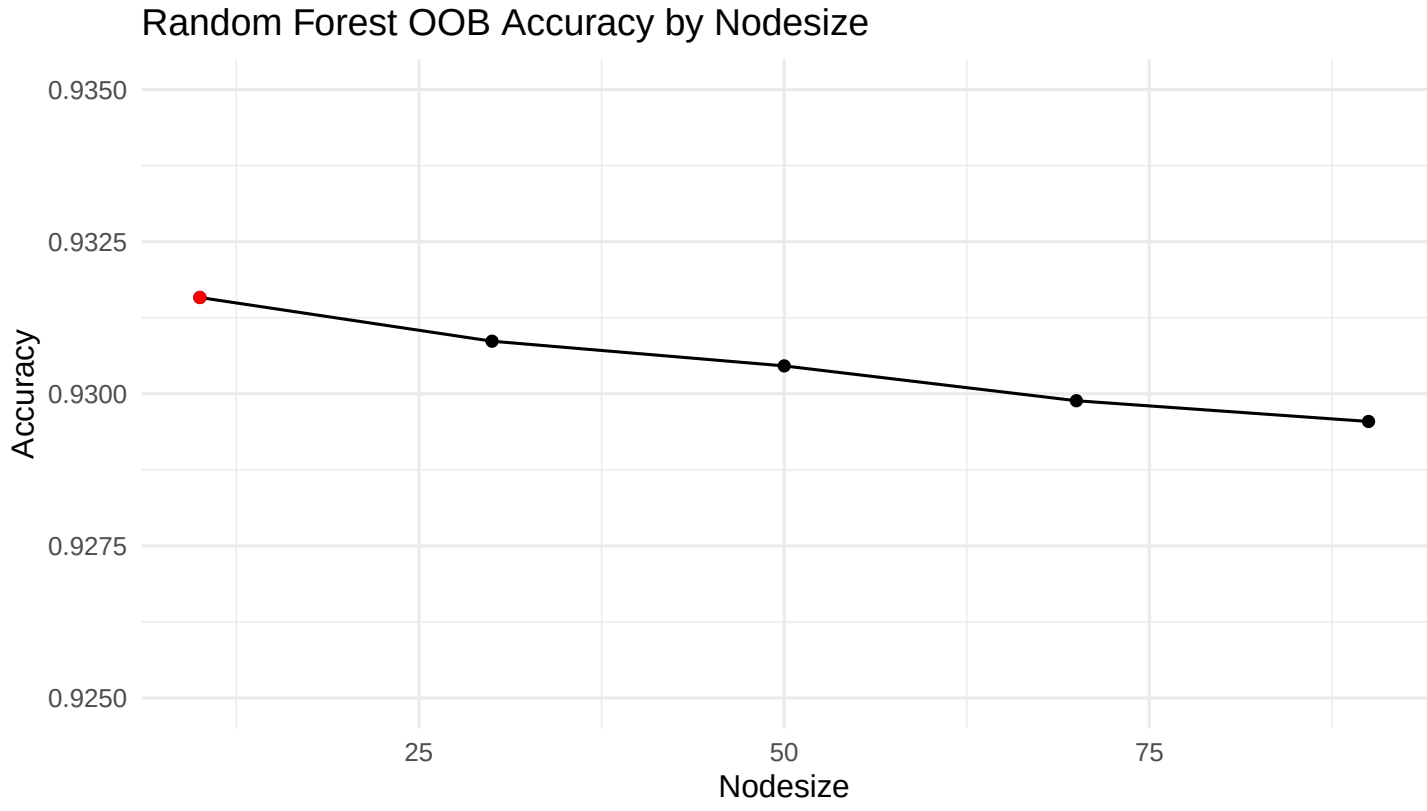
```
## [1] 0.6724847
```

LASSO

```
## <sparse>[ <logic> ] : .M.sub.i.logical() maybe inefficie

##  [1] -0.1106207892  0.2759643490  0.1159028318  0.29184
##  [6]  0.2266853017 -0.0053454197  0.0033333288 -0.01798
## [11] -0.9445639625 -1.9638888147 -0.1027170808 -0.77274
## [16]  0.0518068380 -0.5155705528 -1.4026167245  0.68943
## [21] -0.9723125357  0.0043624505 -0.0297774447  0.04359
## [26] -0.0219640696  0.0762663217  0.0480752968 -0.00764
## [31] -0.0339832302 -0.0578131243 -0.1890927430 -0.19899
## [36] -0.2681536522  0.0165870050 -0.3062763626  0.00023
## [41]  0.0432222285

##      true
## pred      0      1
##    0 114132  53722
##    1   3872   4220

## [1] 0.6726609
```

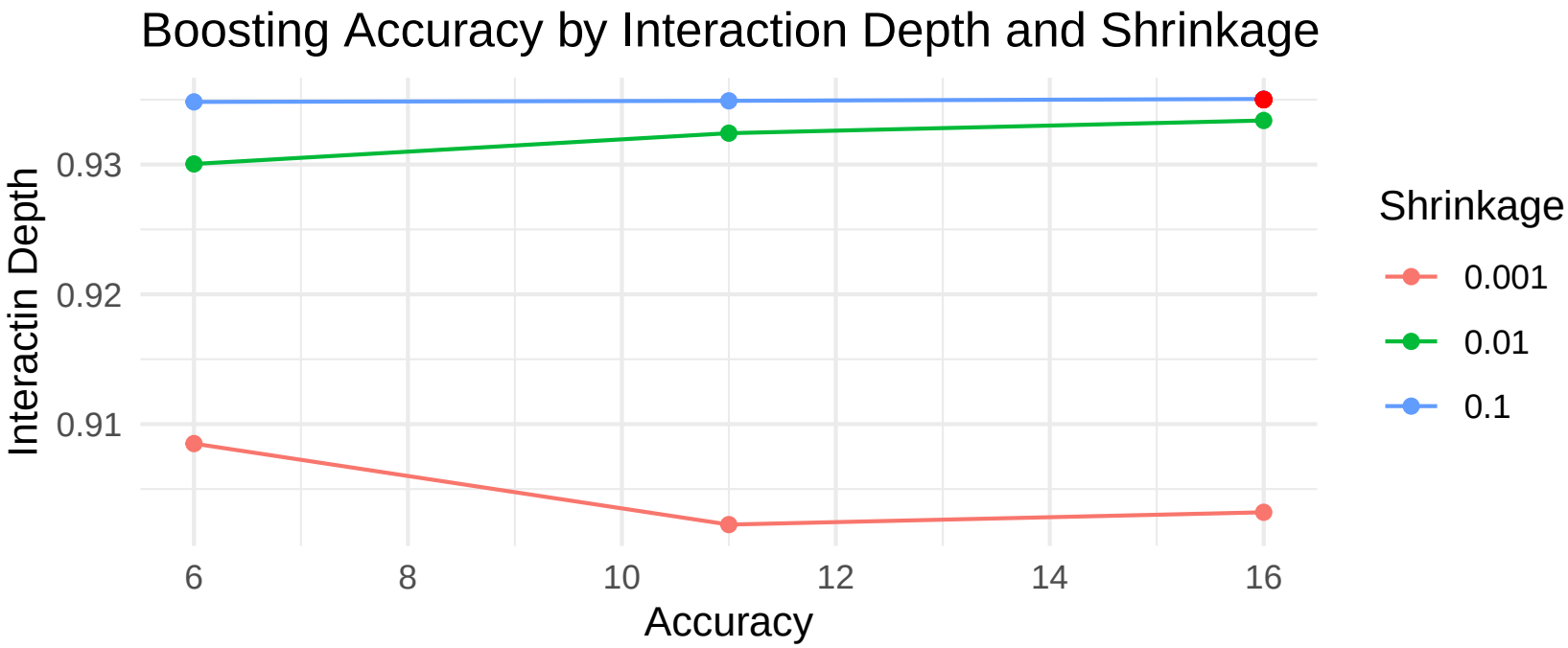
Random Forest

- Tested several Random Forest models
  - Nodesize: 10, 30, 50, 70, 90 | # of Trees: 500
  - Compared Out-Of-Bag Error



Boosting

- 3 Fold Cross Validation on Boosting
  - Interaction depth: 6, 11, 16
  - Shrinkage: 0.1, 0.01, 0.001
  - Number of Trees: 500
  - Compared CV accuracy
- 27 models took over 5 hours to run



Boosting Part 2

- Only tuned shrinkage and interaction depth
- Now tune trees with interaction depth of 16 and shrinkage of 0.1
- Trees: 300, 500, 600, 900, 1200