A Machine Learning Approach to MLB Catcher Framing

Nathan Hemenway and Matthew Boyd

12/16/2021

Motivation

Catcher framing is the art of making a pitch look better than it really is. This is a skill a catcher can have to convince the umpire to call a pitch that would otherwise be called a ball, a strike. If a pitch is for all intents and purposes a ball, then the batter likely will not swing at it. Then if the catcher can make it appear like a strike to the umpire, the umpire will call it a strike and bring the batter one step closer to an out. This is an obvious advantage, since the more strike outs take place, the less of opportunity the opposing team has to score runs while at-bat.

Our idea to measure this phenomenon is to use statistical machine learning methods in order to predict whether a given pitch is a strike. Given the nature of baseball, there will be pitches that are likely strikes and likely balls. If a catcher can convince the umpire to call a likely ball a strike, then that catcher is a good catcher and gets credited for that pitch. On the other hand, if the umpire calls what should have been a strike a ball, then the catcher did not do a good job and is penalized for the pitch. We sought to find out which catchers are best at pitch framing by predicting how likely the pitches thrown are to be strikes, and then seeing how well the catchers can turn them into strikes.

Methodology

Our methodology is to use available data to predict whether a pitch will be a strike for all data points up until the catcher catches the ball. This will give us a good idea of whether any pitch, given a number of characteristics, will be called a strike or called a ball. From the model output, we can get a probability that every pitch will be a strike and compare that to the observed outcome of the pitch. From here we can find out which catchers during the 2021 season excelled at getting more called strikes than their probabilities say they would.

Data

The data contains pitch characteristics for every pitch thrown during the 2021 Major League Baseball season. It was scraped from Baseball Savant using the baseballr package. The scraped data contains ~700,000 rows with ~90 columns. We are only interested in pitches that were a called strike or a ball. This is because if the batter swung at a pitch, how the catcher presents the pitch doesn't matter. This left ~350,000 rows to model from.

Variables

Models

We chose to predict whether a pitch is a strike on several models to find the model with the highest accuracy. The model chosen will be the model used on the results.

Logistic Regression

```
##
## Call:
## glm(formula = strike ~ ., family = "binomial", data = data_no_catchers[train,
##     ])
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.6195  -0.9957  -0.6124   1.2307   2.5036
##
## Coefficients:
##                       Estimate Std. Error z value Pr(>|z|)
## (Intercept)          2.168e+00  6.903e-01   3.141 0.001683 **
## pitch_typeCurveball  3.975e-01  3.838e-02  10.358  < 2e-16 ***
## pitch_typeCutter     3.018e-01  3.177e-02   9.499  < 2e-16 ***
## pitch_typeFastball   4.559e-01  2.885e-02  15.800  < 2e-16 ***
## pitch_typeSinker     6.178e-01  2.723e-02  22.686  < 2e-16 ***
## pitch_typeSlider     3.782e-01  2.812e-02  13.450  < 2e-16 ***
## release_speed       -1.326e-02  1.793e-03  -7.395 1.41e-13 ***
## release_pos_x        4.109e-03  7.465e-03   0.550 0.581991
## release_pos_z       -2.210e-02  1.076e-02  -2.054 0.039987 *
## standR               7.277e-02  1.151e-02   6.322 2.58e-10 ***
## p_throwsR            2.203e-02  3.129e-02   0.704 0.481454
## count0-1            -9.823e-01  1.821e-02 -53.931  < 2e-16 ***
## count0-2            -2.120e+00  3.628e-02 -58.444  < 2e-16 ***
## count1-0            -1.107e-01  1.679e-02  -6.589 4.42e-11 ***
## count1-1            -7.894e-01  2.019e-02 -39.090  < 2e-16 ***
## count1-2            -1.811e+00  2.898e-02 -62.496  < 2e-16 ***
## count2-0             6.527e-02  2.590e-02   2.519 0.011752 *
## count2-1            -5.260e-01  2.648e-02 -19.864  < 2e-16 ***
## count2-2            -1.439e+00  2.954e-02 -48.712  < 2e-16 ***
## count3-0             7.801e-01  3.713e-02  21.013  < 2e-16 ***
## count3-1            -2.314e-01  3.654e-02  -6.332 2.42e-10 ***
## count3-2            -1.041e+00  3.725e-02 -27.958  < 2e-16 ***
## pfx_x               -1.127e-03  7.362e-03  -0.153 0.878341
## pfx_z                1.151e-02  1.533e-02   0.751 0.452829
## plate_x             -3.307e-02  5.904e-03  -5.601 2.13e-08 ***
## plate_z              4.048e-02  5.405e-03   7.488 6.97e-14 ***
## outs_when_up1       -4.848e-02  1.295e-02  -3.744 0.000181 ***
## outs_when_up2       -4.631e-02  1.305e-02  -3.547 0.000389 ***
## inning2              5.466e-02  2.193e-02   2.492 0.012687 *
## inning3              2.922e-02  2.211e-02   1.322 0.186216
## inning4             -3.805e-02  2.231e-02  -1.705 0.088115 .
## inning5             -2.790e-02  2.218e-02  -1.258 0.208531
## inning6             -5.559e-02  2.221e-02  -2.503 0.012328 *
```

```
## inning7             -8.577e-02  2.233e-02  -3.840 0.000123 ***
## inning8             -8.860e-02  2.250e-02  -3.938 8.23e-05 ***
## inning9             -3.205e-02  2.442e-02  -1.313 0.189337
## inning10            -2.418e-01  6.356e-02  -3.804 0.000142 ***
## inning11            -2.764e-01  1.159e-01  -2.384 0.017110 *
## inning12            -2.596e-01  2.272e-01  -1.142 0.253278
## inning13             3.435e-01  4.254e-01   0.808 0.419344
## inning14            -5.641e+00  2.666e+01  -0.212 0.832456
## inning15             1.366e-02  8.908e-01   0.015 0.987763
## inning16             9.462e-01  7.802e-01   1.213 0.225218
## sz_top               1.705e-01  4.128e-02   4.131 3.61e-05 ***
## sz_bot              -5.813e-01  7.501e-02  -7.750 9.16e-15 ***
## release_spin_rate    1.491e-04  2.315e-05   6.438 1.21e-10 ***
## release_pos_y       -2.943e-02  1.234e-02  -2.384 0.017114 *
## locationhome         2.749e-02  1.071e-02   2.568 0.010224 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 223174  on 175944  degrees of freedom
## Residual deviance: 205642  on 175897  degrees of freedom
## AIC: 205738
##
## Number of Fisher Scoring iterations: 6

##         true
## pred          0      1
##   ball   113122  52925
##   strike   4772   5127

## [1] 0.6720755
```

We started with a logistic regression classifier using the selected variables as a preliminary model for predicting strikes. The model did not perform very impressively, with a prediction accuracy of 0.67. The summary for the logistic regression model gives which predictors are statistically significant, which included: - 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 and LASSO

#Ridge Regression

```
##     true
## pred      0      1
##    0 114019  53815
##    1   3875   4237

## [1] 0.6721153
```

The ridge regression model performed modestly with a prediction accuracy of 0.672. This is hardly any different from the logistic regression model, so it also is not the best choice of model to be used to rank the catchers.

#LASSO

```
## 48 x 1 sparse Matrix of class "dgCMatrix"
##                                s1
## (Intercept)          1.0442736664
## pitch_typeCurveball  0.2580520837
## pitch_typeCutter     0.1391181995
## pitch_typeFastball   0.3010434915
## pitch_typeSinker     0.4651378270
## pitch_typeSlider     0.2301523892
## release_speed       -0.0090089326
## release_pos_x        .
## release_pos_z       -0.0079373997
## standR               0.0645863246
## p_throwsR            .
## count0-1            -0.9526621681
## count0-2            -2.0553709629
## count1-0            -0.0850414475
## count1-1            -0.7604104089
## count1-2            -1.7639116626
## count2-0             0.0603334975
## count2-1            -0.4908800365
## count2-2            -1.3957736228
## count3-0             0.7697440768
## count3-1            -0.1841831155
## count3-2            -0.9916946242
## pfx_x                .
## pfx_z                .
## plate_x             -0.0258676977
## plate_z              0.0408917849
## outs_when_up1       -0.0306398199
## outs_when_up2       -0.0298283724
## inning2              0.0714053559
## inning3              0.0437731922
## inning4              .
## inning5              .
## inning6             -0.0132814157
## inning7             -0.0441955536
## inning8             -0.0458348357
## inning9              .
## inning10            -0.1587136678
## inning11            -0.1428440077
## inning12            -0.0196917124
## inning13             .
## inning14             .
## inning15             .
## inning16             0.3396244527
## sz_top               0.0672643911
## sz_bot              -0.3908698178
## release_spin_rate    0.0001977036
## release_pos_y       -0.0163466478
## locationhome         0.0184387463

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

```
##     true
## pred      0      1
##    0 114922  54625
##    1   2972   3427
```

```
## [1] 0.6726439
```

The model obtained using the LASSO gave a similar prediction accuracy as the logistic and ridge regression models at 0.673. The LASSO also performs variable selection, and gave non-zero coefficients for: - Pitch type - Count - Outs and Inning - Batter height - Whether the game is home or away - Where the pitch landed

Random Forest

Boosting

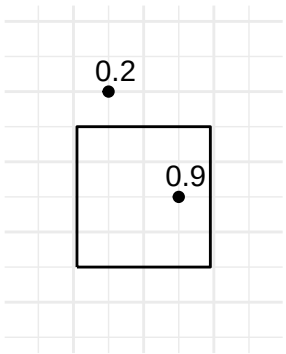
Results

To give credit to each catcher, we use this equation for every pitch:

- If observed strike: 1 - Strike Probability
- If observed ball: Strike Probability * -1

To give an example, below is a plot of the strike zone. Umpires will try to make a call of a strike or ball if the pitch is located inside the rectangle. However, umpires don't have access to a strike zone while actually making calls themselves, so there will be some error as to whether a pitch an umpire calls a strike, is actually a strike (inside the strike zone).

The point located inside the strike zone is a theoretical location of a pitch with a 0.9 strike probability. That's quite high but it makes sense, given the pitch is in the strike zone. The pitch outside of the strike zone has a 0.2 strike probability since the pitch lands outside the strike zone. If the pitch with a 0.9 strike probability gets called a strike, the catcher will earn 1 - 0.9 = 0.1 strikes worth of credit. This is a small contribution given that the pitch is more likely to be a strike anyway. However, if the pitch gets called a ball, he will lose 0.9 strikes worth of credit which is a significant loss. That is because the catcher could have could have presented the pitch to the umpire very poorly, and he deserves to be be given credit with losing a strike for his team. If we look at the pitch out of the zone, if that pitch is an observed called strike, the catcher will receive 1 - 0.2 = 0.8 strikes worth of credit. That's a big gain for the catcher's team because he turned a pitch, more likely going to be a ball, into a strike and he should be rewarded. If that pitch is an observed ball, he will only lose -0.2 strikes worth of credit. That's small because the pitch is likely to be a ball anyway.



Once we have an accurate model this approach is completed for every pitch in th 2021 Major League Baseball season since we will have a strike probability and we will know the observed outcome of the pitch (called strike or ball). This way we can see which catcher's are better at presenting pitches to the umpire to benefit their team.

Conclusion