# P(ball contact | batter swings)

A collection of non-parametric classification models constructed via MLB pitch data.

Emma Godfrey

December 5th, 2020

## Introduction

In recent decades, baseball has drifted towards being a 'power' league, with hitters often swinging for the fences and pitchers trying to throw it past you. Launch angle and spin rates have become mainstream data points to collect in terms of evaluating a swing or pitch. Technology like Rapsodo's has given every team with a budget of a few hundred dollars the ability to track velocity, spin efficiency, and much more, in real time. The rise of organizations like Driveline, which meticulously apply data-driven approaches to improving biomechanics and baseball performance, is undoubtedly related to the state of the baseball we see today. Despite the steroid-era having ended over a decade ago, home run numbers and pitching velocities have sharply increased in recent years. In the 2019 season, 6776 home runs were hit in the entire league, which is more than 1,000 more than were hit in 2000, and more than double the number of home runs hit in 1980.[1] Regarding pitching, the percentage of fastballs thrown above 95 mph in 2008 was 12%; now it is 22%. Unsurprisingly, the strikeout rate has increased from 16.9% in 1998 to 22.3% in 2018.[2] Almost every team has at least one pitcher that throws 100 mph. Even 18-year old kids getting drafted out of high school are displaying these mind-blowing feats of athleticism and explosiveness.. Clearly something is going on, and it does not seem to be going away. With such a large emphasis placed upon pitcher's having "swing-and-miss stuff", a cliche referring to how difficult it is to hit a pitcher's arsenal, it seemed fitting to study what makes hitters swing and miss; collecting all of the obscure data is only helpful insofar as we know what parameters to maximize to get the desired result on the field. In this paper, I attempt to use machine learning techniques to classify and predict whether a pitch will be hit or missed based on statcast pitch-by-pitch data.

## Exploratory Data Analysis

The dataset I will be using to conduct my analysis was sourced from Kaggle and contains pitch level data for the 2015-2018 MLB seasons. Each observation represents a single pitch. Due to the minutiae of baseball and its reliance on statistics, there are over 40 explanatory variables in this dataset which, to the average non-baseball connoisseur, is overwhelming. Here, I will detail the most important variables for the analysis. For a more detailed variable explanation, reference this link.

| **Variable Name** | **Definition/Interpretation** |
|---|---|
| px | The X-location as the pitch crosses the plate. X=0 means the pitch crossed right down the middle of the plate. |

[1] Poindexter, Owen. "Baseball's Historic Home Run Season In One Chart." Forbes. Accessed December 4, 2020. https://www.forbes.com/sites/owenpoindexter/2019/06/28/baseballs-historic-home-run-season-in-one-chart/.
[2] Sullivan, Jeff. "The Velocity Surge Has Plateaued." *FanGraphs Baseball* (blog). Accessed December 4, 2020. https://blogs.fangraphs.com/the-velocity-surge-has-plateaued/.

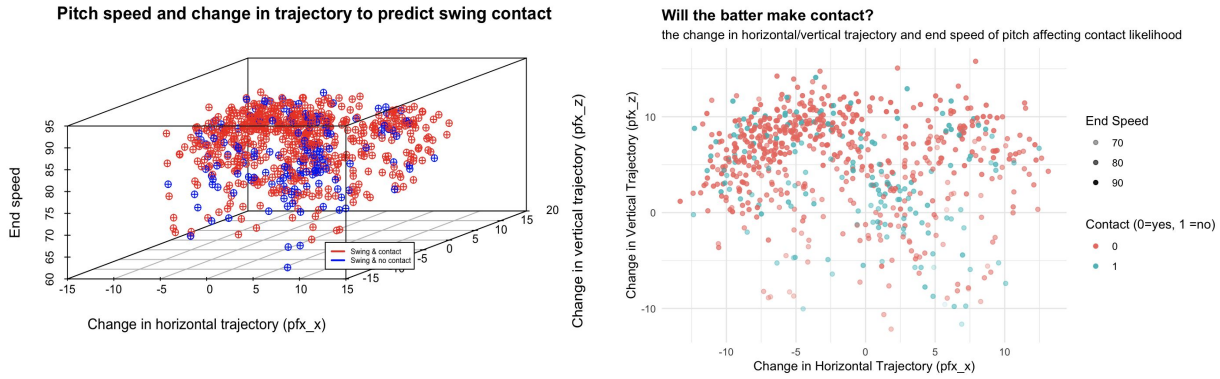
| | |
|---|---|
| pz | The Z-location as the pitch crosses the plate. Z=0 means it crossed at ground level. |
| start_speed/end_speed | Speed of pitch as it was released/crossed the plate (mph). |
| spin_rate | The spin rate of the pitch (rpm). |
| spin_dir | The direction of the spin of the pitch (degrees). |
| break_angle | The angle of the vertical straight line path from the release point to where the pitch crossed the front of home plate (degrees). |
| break_length | The length of the greatest distance between the trajectory of the pitch at any point between the release point and the front of home plate, and the straight line path from the release point and the front of home plate (inches). |
| break_y | Length from break length to where the pitch crosses home plate. |
| ax/ay/az | The acceleration of the pitch (ft/s) in three dimensions, measured at the initial release. |
| vx0/vy0/vz0 | The velocity of the pitch (ft/s) in three dimensions, measured at the initial release. |
| x0 | Horizontal location of pitcher's hand at the release point (ft) with respect to the center of the pitching rubber. |
| z0 | Vertical location of pitcher's hand at the release point (ft) with respect to field level. |
| x/y | The horizontal/vertical location of the pitch as it crossed home plate (ft). |
| pfx_x / pfx_z | The deviation of pitch trajectory of horizontal/vertical location (inches). |
| nasty | A measurement of the greatest distance between the trajectory of the pitch at any point between the release point and the front of home plate, and the straight line path from the release point and front of home plate (inches). |
| zone | PitchFX zone number. 1-9 indicates inside strike zone, 11-14 indicates outside strike zone. |
| code | Result of the pitch (i.e. swinging strike, foul, ball) |
| type | Simplified version of code (strike/ball/in-play) |
| pitch_type | Type of pitch (i.e. four-seam fastball, changeup etc.) |

Prior to any subsetting of the data, there are 2,867,154 pitch observations with 40 features. However, I am only concerned with predicting whether the batter will swing and make contact or swing and no contact. The following values of the variable 'code' will allow me to subset the data accordingly.

| **Swing and contact** | **Swing and no contact** |
|---|---|
| L - Foul Bunt<br>T - Foul Tip<br>F - Foul<br>X - In play, out occurred<br>D - In play, no out occurred<br>E - In play, run occurred | S - Swinging Strike<br>W - Blocked swinging strike<br>M - Missed Bunt |

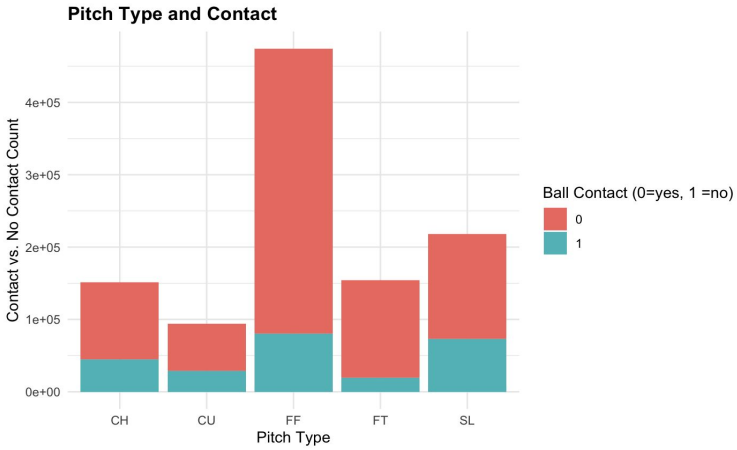
To indicate the presence or absence of ball contact given a swing, I created a binary indicator variable, 'swing.miss'. 'swing.miss' = 1 if 'code' takes on values 'S', 'W' or 'M'. Contrastingly, 'swing.miss' = 0 if 'code' takes on values 'L', 'T', 'F', 'X', 'D', or 'E'. After subsetting the data on the presence of a batter swing, there are 1,334,062 unique pitch observations, all of which are complete observations. With this subsetted dataset, I perform an initial data analysis to visualize patterns and motivate further exploration.

Consider the two graphs below, both attempt to uncover whether change in horizontal and vertical trajectory and end speed of the pitch affect the likelihood of the batter making contact. On the left, I created a 3-dimensional visualization where the color of the points indicates contact presence. Comparatively, on the right, I present the 2-dimensional counterpart. Note, all variables represented in the left plot are also represented in the right. However, instead of a third axis, I denote the pitch's end speed via transparency levels.




I first want to bring attention to the high density of 'swing & contact' (red) points in the upper left quadrant of each plot. Due to increased speed and high vertical trajectory in this area, the majority of these pitches likely represent fastballs. Notable, the high majority of pitches in this quadrant resulted in contact. I suspect this is due to the popularity of a fastball within the MLB, and thus batters are accustomed to typical patterns of fastball movement and adjust accordingly. Next, the density of 'swing & no contact' (blue) points increases as the change in horizontal trajectory shifts from negative to positive.

Next, consider the frequency of contact vs. no contact across the most common pitch types. Here, I am curious whether the type of pitch shows a clear relationship with the frequency of contact vs. no contact. As we can see, the proportion of swing and no contact occurrences vs. swing and contact occurrences is significantly higher for changeups, curveballs, and sliders. The difference in proportion across pitch types is significant and thus suggests pitch type will be an important factor in predicting contact.




Pitch Glossary:

CH: Changeup

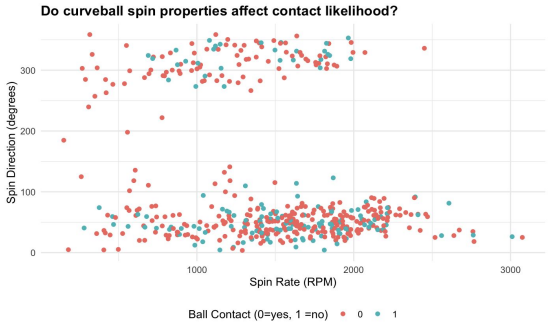
CU: Curveball

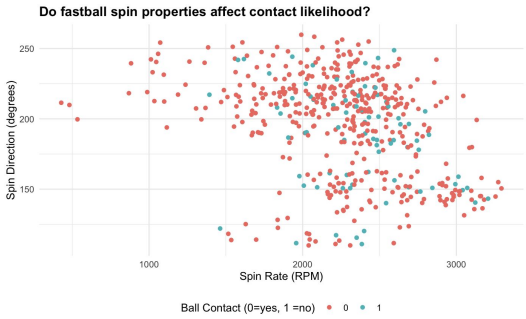
FF: Four-seam fastball

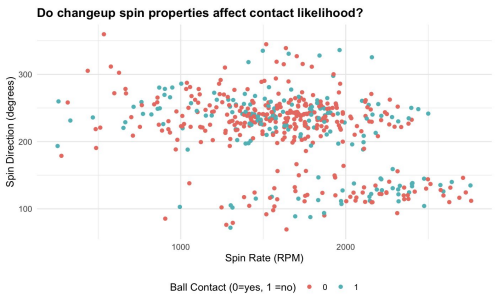
FT: Two -seam fastball

SL: Slider

While pitch type is clearly an important factor when predicting contact, I wonder what specific features of each pitch influence the significant difference in contact vs. no contact likelihood as noted above. Here, I consider spin direction, measured in degrees, and spin rate, measured in rotations per minute, for three different pitch types: curveball, fastball, and changeup. As we see below, there is no discernable pattern across the three pitch types. However, amongst fastballs, almost all of the pitches with a low spin rate resulted in contact; this pattern seems to exist, but less drastically, within the curveball pitches as well. Additionally, I posit that the vertical divide of pitches based on spin direction is due to the handedness of the pitcher.

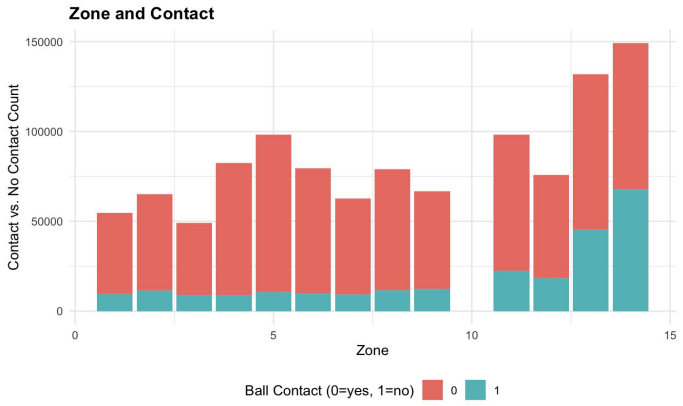


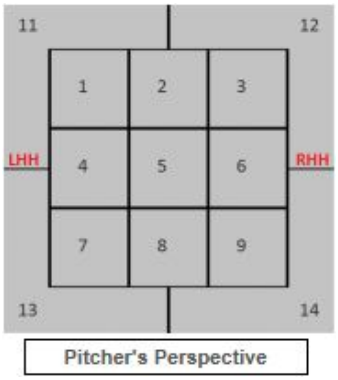




Lastly, I am curious whether the location of the pitch, as determined by the zoning described below, has a significant influence on the proportion of contact vs. no contact swings. Of note is the high

proportion of no contact swing in zones 13 and 14. As pictured on the right, these zones correspond to the lower out-of-strike-zone quadrants. Per my research, it is a common strategy to throw pitches below the strike-zone to entice batters to swing at pitches that are hard to hit.




As shown throughout this section, there are interesting relationships within the data to exploit via thoughtful model construction to maximize predictive power and ultimately provide teams with meaningful insight.

## Model Construction

In this section, I will begin by constructing testing and training sets and accessing the class balances within. Then, I will build two models which seek to predict the variable 'swing.miss'. In other words, the model will attempt to predict whether, on a given pitch where the batter swings, there was contact with the ball. Given the model has descently high predictive capability, I will then look at various variable importance plots to discern which aspects of the pitch matter the most in getting a batter to swing and miss.

As data scientists, the goal is to build models which accurately predict future data via training the model on current data. However, in real life, there exists a single dataset. Thus, to evaluate the predictive capabilities of a single learner or compare across learners, we must deconstruct the single dataset into distinctive training and testing sets. The training set will act as current data, and the testing set will act as never before seen future data. Since the original dataset contains over 1.3 million observations, I randomly sample 50% of observations from the original dataset to create a more manageable dataset (n = 667,031) which inherits similar patterns through random sampling. From this smaller dataset, I randomly sample 80% of observations for the training data and 20% for the testing data. As a result, there are 533,624 observations in the training dataset and 133,407 observations in the testing dataset.

Following the construction of the testing and training sets, I test the balance of contact vs. no contact pitches within the training dataset. Checking for balance prior to training is vital as learners tend to underperform in situations with severe class imbalance. If the unbalance is severe enough, the learner's optimal rule will be to always classify observations as the majority class. In situations where the less frequent class is of utmost predictive importance, the learner which always classifies observations as the majority class is useless. In this case, 77.9% of the training observations belong to the swing with contact class; this leaves only 22.1% of observations belonging to the swing with no contact class. To remedy this imbalance, I use a technique called up-sampling. Up-sampling is the process of randomly duplicating observations in the minority class (i.e. swing with no contact) with replacement until the dataset achieves class balance. The R package 'ROSE' makes this process extraordinarily easy. After implementing up-sampling, the new training data consists of 50/50 balanced classes.

Next, I will build out various classification prediction models and assess their performance. To do so, I will use a common metric known as the Area Under the Receiver Operating Characteristics (AUROC). The ROC curve is constructed by plotting the true positive rate (TPR) against the false positive rate (FPR). An AUROC close to 1 suggests that the model can distinguish between classes quite effectively. Contrastingly, an AUROC of 0.5 indicates that the model is no better as distinguishing classes than chance alone. Given these interpretations, I hope to build a model with the highest possible AUROC.

1. *Random Forest*

The building blocks of a random forest model are decision trees. A single decision tree consists of many recursive binary feature splits. In a simple sense, at every level of the tree, there exists a logical feature test; data points corresponding to 'true' will funnel right, while those corresponding to 'false' will funnel left. Given a feature, there are many logical splits we could consider. However, in building a decision tree, one often decides the split point by minimizing Gini Impurity, a measure of heterogeneity in the resulting leaves. Since the tree tries to minimize heterogeneity in the leaves at each recursive step, this algorithm is known as a greedy algorithm as it tries to find a local optimum/minimum at every split without considering future splits. A decision tree's tendency to get stuck at local minimums/maximums motivates the use of a random forest.

A random forest consists of many decision trees compiled in an ensemble manner. Given a new observation, each individual tree will classify that observation and the class which receives the majority of votes secures that observation's label. The intuition behind a random forest is that many simple, uncorrelated decision trees operating via majority rule will outperform a single, complex decision tree. In other words, a random forest has greater predictive capability than the sum of its individual parts since the random forest will hopefully balance out any errors from individual trees. To ensure higher predictive

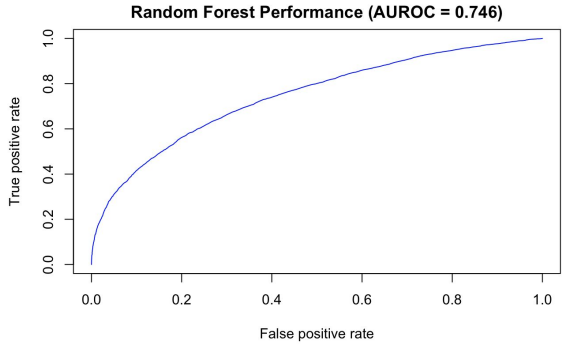
power than the sum of the individual trees, the individual trees cannot be correlated. Creating uncorrelated decision trees is done via two steps:

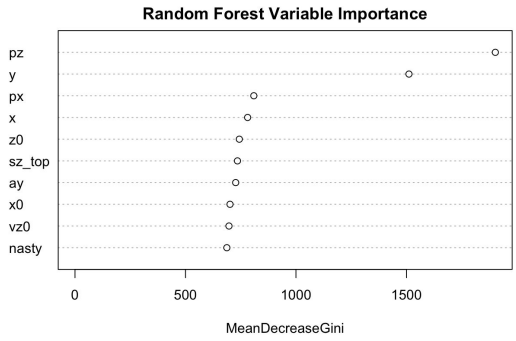
1. Choose a random subset of features for each individual tree in the forest.
2. Employ bootstrap aggregation. Choose a random subset of the training data to train the individual tree on.

Following construction, I will use the random forest model to predict the classes of the test data set and conclude its performance via AUROC.

The R package 'randomForest' simplifies the implementation. First, it is important to find the optimal number of features to consider at each decision tree. To do so, I use the function 'tuneRF' within the 'randomForest' package. The optimal number of randomly subsetted features to build each decision tree with, denoted 'mtry', is 10; however, I will note that there was little out-of-bag error change from mtry=10 to mtry=3. Next, using this optimal number of features, we can build the random forest.

In the plots below, we can visualize the performance of the random forest. The random forest achieved an AUROC of 0.746; this is notably better than chance, however there exists much room for improvement. Moreover, we can see the variables which proved to be most important in the graph to the right. Consider the variables 'pz' and 'y'; these variables cause the greatest mean decrease in Gini Impurity. Both 'pz' and 'y' represent a measure of the vertical location of the pitch as it crosses home plate; 'pz' denotes distance in feet from the field level while 'y' denotes the vertical location as indicated by the Gameday coordinate system. In a similar fashion, 'px' and 'x' are seemingly the next two most important variables, representing the horizontal location of the pitch as it crosses home plate. Interestingly, speed, trajectory shift, and pitch type do not appear within the top few importance features. This indicates that the location of the ball as it crosses home plate is more indicative of the presence of contact than movement features.



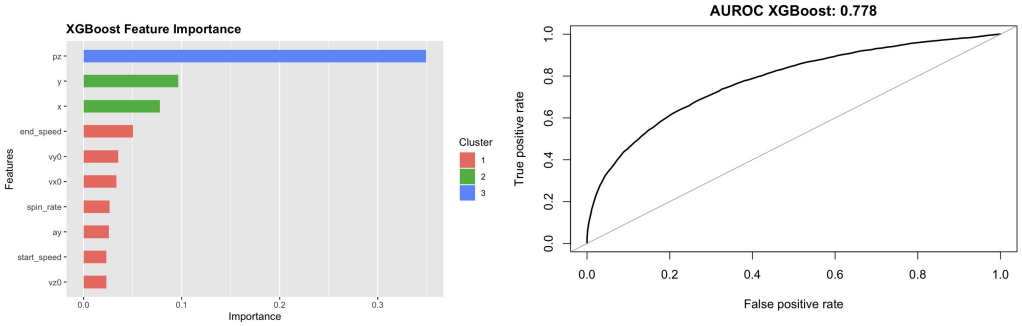

While the location of the pitch seems most important in predicting whether the batter will make contact upon swinging, there is still a hefty amount of unexplained variability as noted by the lower

AUROC. Going forward, I hope to train the random forest on a larger portion of data. However, given time constraints and the computational complexity, I was limited in that regard.

*2. Extreme Gradient Boosting*

Extreme Gradient Boosting (XGBoost), similar to a random forest, is an ensemble learner where the individual components are classification trees. While random forest utilizes a technique called 'bagging', XGBoost utilizes 'boosting'. Boosting is a technique used in ensemble learners where the subsequent weak learner focuses on observations which the previous learner predicted poorly. This contrasts a random forest where subsequent weak learners are completely unrelated to prior weak learners. To construct the subsequent trees, XGBoost uses a gradient descent algorithm to minimize error induced from prior trees. The gradient descent algorithm is used to find a local minimum of the loss function and thus improve the overall error rate. Intuitively, we can imagine choosing a point on a traditional cereal bowl and calculate the derivative of our error function, the so-called gradient, evaluated at our chosen point. The gradient is then used to find the direction in which to tune the parameter(s) to maximally reduce the boosted model error. XGBoost employs various hyperparameters to increase the computational speed and accuracy of the general gradient boosting algorithm, however these are beyond the scope of this paper.

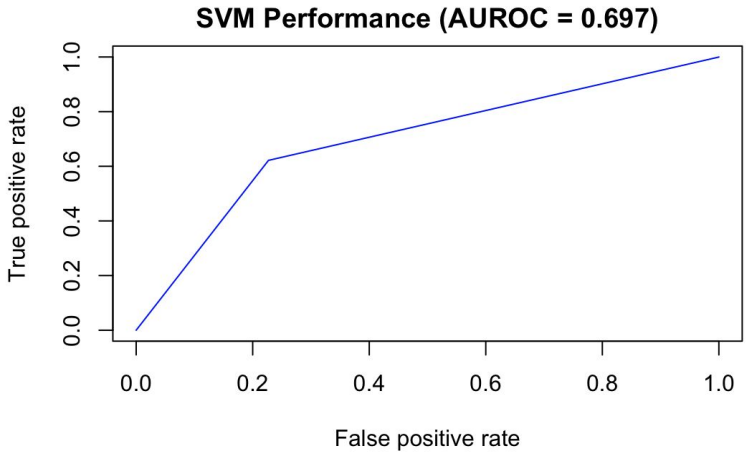
The R package to implement Extreme Gradient Boosting is rightfully called 'xgboost'. The implementation is fairly straightforward, and notably the runtime is drastically shorter compared to random forest and support vector machines. As we can see below, the AUROC improved with XGBoost compared to random forest, however not by leaps and bounds. Similarly, the top three importance features are the vertical and horizontal location of the ball as it crosses home plate. However, XGBoost deems the end speed and velocity of pitch of higher importance than random forest; this ranking of features is more aligned with my a priori beliefs. Again, while the AUROC is not worse than chance, perhaps a larger training set and closer parameter tuning would increase predictive power.



*3. Support Vector Machines*

Lastly, I fit a support vector machine (SVM) model. The objective of a SVM is to fit a hyperplane to the data in the feature space which correctly classifies the observations according to the observation's location with respect to the hyperplane. While there are many hyperplanes that can classify the observations with similar accuracy, in an SVM, we want to find the hyperplane with the greatest distance from the data points of each class. In other words, we want to maximize the margin between the observations in each class and the hyperplane. In making the margin as large as possible, one will have greater confidence in the classifications of future observations. There are two parameters that require tuning in creating an SVM, the cost parameter and the flexibility of the learner. First, the cost parameter, represented by lambda, represents the penalty for misclassification. If we expect our data to be perfectly classifiable, then we would want to make lambda large. However, as the lambda increases, the bias of the learner simultaneously increases. The flexibility of the learner, denoted by gamma, represents the standard deviation of the gaussian normal bumps. A low value of gamma will produce a decision boundary with less curvature, which results in a decreased variance.

For R implementation, the package to use is 'e1071'. While the implementation is fairly straight forward, the parameter tuning is where the real magic happens. Unfortunately, due to the computational complexity of simultaneously tuning the gamma parameter and lambda parameter with such a large dataset, the tuning process was not smoothe. In fact, even after using less than 5% of the original dataset for training data in the tuning process, the process took over 12 hours. After the 12 hour process, the optimal parameters were at the endpoints of the grid, and thus did not provide any further information. Unfortunately, due to time constraints, I was unable to tune the SVM any further. The lack of tuning is likely part of the reason for the poor AUROC metric, 0.697. In the future, I want to perform principal component analysis and build an SVM with the resulting features. This process of dimension reduction will hopefully provide computationally less intensive tuning and construction process.



## Conclusion

Baseball is a sport of strategy consisting of seemingly endless statistics. For pitchers, their goal is to throw pitches that entice the batter to swing. As a hitter, their goal is to successfully recognize the incoming pitch and make good swinging decisions to hopefully get the ball in play. In this paper, I focus mainly on the pitcher's goal of throwing enticing, yet tricky, pitches that the offensive batter swings at. I sought to answer the following question: Given that the batter swings, what influences the likelihood of the batter making contact with the ball? To answer this question, I first subset the data into two groups, ball contact and no ball contact, conditional on an offensive swing. Initially, I looked at pitch type, ball movement, and pitch location as predictors for successful ball contact. Based on my initial EDA, pitch location seemed the most meaningful in predicting successful ball contact. I then built a random forest, extreme gradient booster, and a support vector machine using a training dataset and proceeded to test each model on unseen data. The XGBoost model outperformed the other two models in terms of area under the ROC curve, however there is still significant unexplained variability. Most importantly, though, the location of the pitch (horizontal and vertical) ended up being the most meaningful features in all three models. The high importance of pitch location agrees with my initial EDA and dissection into the likelihood of contact in a given zone; there was a significant increase in no contact swings just below the strike zone.

Next, I want to further tune the support vector machine. I plan to do so via feature reduction to decrease the computational complexity and hopefully increase the speed of the process. Additionally, with a reduced number of features, I will hopefully be able to increase the amount of data for which I train each model on.