Live on national TV, the batter stares into the eyes of the pitcher, waiting for the delivery. The pitch is thrown and the batter puts all of his strength into a swing - looking to drive a home run right out of the ballpark. Instead he finds himself swinging five feet out in front of the pitch with his bat flying into the crowd. Nothing is more embarrassing for a Major League batter than guessing fastball and instead getting thrown hook. Major League batters “sit” on certain pitches during every at bat. Determining the type of pitch that is being thrown is the first step in making solid contact with a swing. However, this guessing game is tricky and is often impossible.

My love for baseball, paired with my recent discovery of machine learning, gave me the idea to attempt and solve the batter's dilemma. The goal was to construct a few machine learning models that predict major league pitches based on the current game scenario. I have a long-term vision of models that will print out a list of pitch predictions before every MLB game, but for now I settled on developing a few simple open-source models that could be easily used by anyone.

Thanks to awesome websites like Baseball Savant, and a beautifully made api by James Ledoux, individual pitch data was readily available for me to download in bulk. After some brief data cleaning, I determined that I wanted to focus my models around predictive decision tree algorithms, specifically gradient boosting and random forest. Later forays into other models like SVM’s, k-nearest, and naive-bayes confirmed that random forest and gradient boosting were the best out-of-the-box models to work with. My dependent variable, *pitch_type*, was already clearly established, and I then determined that my predictive independent variables would be: *ball_strike, stand, outs_when_up, inning, on_3b, on_2b, on_1b, at_bat_number, pitch_number*.

After determining what variables and algorithms to use, I decided to make three different open-source models: a single player model that looked to predict pitches for one pitcher, a multiplayer model that predicts pitches based off of a set of major League pitchers, and a fastball/off-speed pitch model that simply predicts a fastball or off-speed pitch for any player. All three models are posted on my github page and can be easily used by others in jupyter notebook.

The single player pitch model is run with the above mentioned variables and can be easily adjusted to predict any MLB pitcher and any number of his pitches. The test case that I used for the single player model was Felix Hernandez, due to his high number of pitches since his MLB debut in 2005. Setting the needed frequency of a pitch at 15%, Felix Hernandez had four pitches that qualified for the model: slider, four-seam fastball, changeup, and curveball. Therefore, the baseline rate of a random guess is 25%, or a 1 in 4 probability. Both the random forest model and gradient boosting models saw an improvement in this probability. After some light model tuning using gridsearchcv, individual parameter tuning, and roc-curves, both the random forest model and the gradient boosting model saw increased positive prediction rates by 13%.

The multiplayer model uses the same variables and algorithms as the single pitcher model and is designed to predict general trends across all MLB pitchers. The test model used 37 MLB pitchers, all of which are on the MLB’s top 100 most innings pitched list for current players. Using a 5% pitch frequency limitation, seven pitches were selected from the aggregate pitch data - leading to an approximate 14% baseline guess rate. The random forest model, without tuning, showed an increase in this baseline guess rate by 20%, and the gradient boosting model showed an increase of 25%. It is important to note that the gradient boosting model in this situation started to guess only fastballs since the aggregate data had a large selection of fastballs. This is due to the algorithms construction to learn from past trees and improve its future trees. If the pitch frequency for each pitch is fed equally into the gradient boosting model, the increased accuracy drops slightly. Overall, this model showed more promise than the single-player model and could be greatly improved with more experimentation.

The final model is the fastball/off-speed pitch model, which again uses the same variables and algorithms as the previous two models. This binary model looks at both single

player and multiplayer data. Once again, I used Felix Hernandez as my test subject for the single player portion of the binary model and mapped all of his pitches to either fastball or off-speed. The untuned random forest model saw barely any predictive improvement over the baseline 50%, but the gradient boosting model showed an increased predictive rate of over 10%. The gradient boosting model further increased its predictive power with limited parameter tuning and unequal input of pitch types by 26%. The multiplayer portion of this model saw less promising results across the board. The random forest model, without tuning, showed nearly no predictive power improvement, and the gradient boosting model showed an improvement of 6%. Further research into the binary single player model could show very promising results. Intuitively, this is a model that should be able to have decent predictive power with the help of machine learning algorithms.

Even though my results were not as robust as I had hoped they would be, I am not discouraged and will continue to look into predictive pitch models. All of the data that I experimented with was data from top MLB pitchers that are on the active leader list for innings pitched. These pitchers have clearly had success at the Major league level and are most likely very good at being unpredictable. It would be interesting to see if pitchers with higher ERA's, or pitchers with less successful Major League careers are more predictable using the single player model. In addition, further promising research can be done with seasonal data. Without a doubt pitchers change their approach on the mound season to season, and even throughout each individual season. Weighing data based on date or only constructing seasonal models could show higher results. Finally, it is important to note that even though many descriptive variables were used in the models, the only batter-specific variable that was used was whether or not the batter was left or right handed. All of the other variables described the current game scenario. Further variables that represent if the batter is a power hitter, a seasoned veteran, a currently hot hitter, or a batter that has had past success against this pitcher could easily increase the robustness of all of the models.

Overall, I thoroughly enjoyed looking into this machine learning problem and will continue to do so in the future. In the meantime, I hope that many people find my models to be interesting and easy to use on my github page. I would love to gather a vibrant community of data scientists and baseball enthusiasts to tackle this problem together.