Capstone Project Report

Stephen Strosko

November 26, 2017

The Problem:

My capstone project looks to solve the problem of how to successfully use fx/pitch data from MLB pitchers to improve pitch predictability for an MLB batter. When facing an MLB pitcher that has 3 or 4 solid pitches that can vary up to 20 mph in speed, a batter is forced to take an educated guess at the pitch type if they are looking to make solid contact. Even though each batter has a different process for deciding what pitch to “sit on”, every batter is always thinking about what pitch is coming next. My goal is to help this process by using machine learning models to predict major league pitches.

Clientele:

My clientele is two-fold. I hope that the first users of my models are any MLB related organizations that believe my findings will improve their player performance or research. The second user group would be avid fans of baseball, like myself, that enjoy analytics and the art of pitching. These models could give any MLB team the upper edge in a pitching matchup and any avid fan a tool to impress friends with. I am passionate about baseball and sports analytics and would love to expand on my capstone project’s work in the future to suit the need of any MLB related organization.

Retrieving and Cleaning the Data:

Originally, I thought that the only available option to retrieve fx/pitch data in bulk was to download it by hand from baseball savant. The website does not have a feature that allows for a

user to download data for multiple players, so the only available option was to look for an api that worked with the website. Thankfully, an awesome api called *pybaseball* was created to provide the flexibility that was needed for this project. The api interacts with baseball savant and allows for data to be downloaded via command line interface. Because of the api, I was able to write a short wrapper code that allowed me to retrieve data in bulk for a set of players. In addition, I was able to wrap the api in a way that allows individuals that use my jupyter notebooks to retrieve data for any MLB pitcher for any part of their pitching career.

After retrieving the needed data, there was some work that needed to be done to get the data into a form that most out-of-the-box machine learning algorithms through sklearn would accept. To start, I had to isolate all of the variable columns that I wanted to work with, as well as convert them into the correct variable type (integer). After completing that, I worked on mapping many of the variables including: *stand, on_1b, on_2b,* and *on_3b* to values of 0 and 1. This was done by using the map feature to improve runtime. The structure is shown in the example code below from my project.

```
df['on_1b'] = df['on_1b'].map(lambda x: 1 if x != 0 else 0)
```

Two variables *balls* and *strikes* I mapped to a new variable *ball_strike* to allow for a better representation of a pitch count. This decision was made once I determined that the best algorithms to use in the project would be decision tree based. A 3-0 count and a 3-2 both have three balls, but represent very different game situations. Converting balls and strikes to one variable that represents the pitch count allows for amore concise representation of an in-game scenario.

The next step in the data cleaning process was to focus my attention on my dependent variable, *pitch_type*. I built functionality into my code that looks at all of the unique occurrences of *pitch_type* and determines which pitches are outliers in the data. This variable, *percentage*,

can be adjusted based on user preference. By adjusting the *percentage* variable, the user can set the minimal amount that a pitcher has to throw a given pitch, relative to his total amount of thrown pitchers, for that pitch to be included in the model. In addition, all NaN values or rows that contained missing data were dropped from the dataset all together. There were very few rows that were dropped due to these conditions, therefore, no concerns about data integrity are raised.

Finally, a post-model addition to the data cleaning process took the form of an optional block of code that allows the user to give the model an equal amount of data points for each selected pitch. This feature is extremely useful for pitchers that are predominate fastball throwers since it encourages algorithms like gradient boosting to shy away from predicting fastball for every pitch. Even though predicting fastball every pitch for a fastball dominate pitcher may increase overall model accuracy, this does not give us the variance that is desired in the model.

The Models:

The three models I decided to build are the single-player model, the aggregate multi-player model, and the off-speed/fastball binary model. All three models go through virtually the same data cleaning process, the process described in the previous section, but each model uses a different form of the pitch data. The single player model only takes in one pitcher's data and will predict pitches for that individual pitcher. The multi-player model takes in a set of aggregate pitcher data, inputted via a *.txt* file, and will make predictions that can be applied across all MLB pitchers. The binary model is the most unique model and looks to simulate a batter looking for either a fastball or an off-speed pitch. This model can take data for either a single pitcher or a list of pitchers and uses the example code below to map all pitches to either fastball or off-speed.

```
df['pitch_type'] = df['pitch_type'].map(lambda x: 1 if (x != 'FF'
or x != 'FT' or x != 'FC', or x != 'SI') else 0)
```

FF, FT, FC, and SI are abbreviations for the four most common fastballs: four-seam, two-seam, cutter, and sinker, respectfully.

Results:

After cleaning the data and determining the models I wanted to create, it was time to experiment with different out-of-the-box sklearn machine learning algorithms. The following algorithms were all thoroughly tested:

1) K-Nearest
2) Random Forest
3) Gradient Boosting
4) Naïve Bayes
5) Support Vector Machine

The most robust results were shown by the random forest and gradient boosting algorithms. These two algorithms also made sense to pursue logically since I had a large set of labeled data with many descriptive variables. For all of the models, the dependent variable (*y*) was set to the variable *pitch_type*, and my predictive independent variables (*X*) were: *ball_strike, stand, outs_when_up, inning, on_3b, on_2b, on_1b, at_bat_number, pitch_number*.

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 pitcher that I used for the single player model was Felix Hernandez. Hernandez was a perfect test candidate due to his high number of pitches thrown 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. This improvement was increased after some light model tuning using GridSearchCV, individual parameter tuning, and roc-curves. The

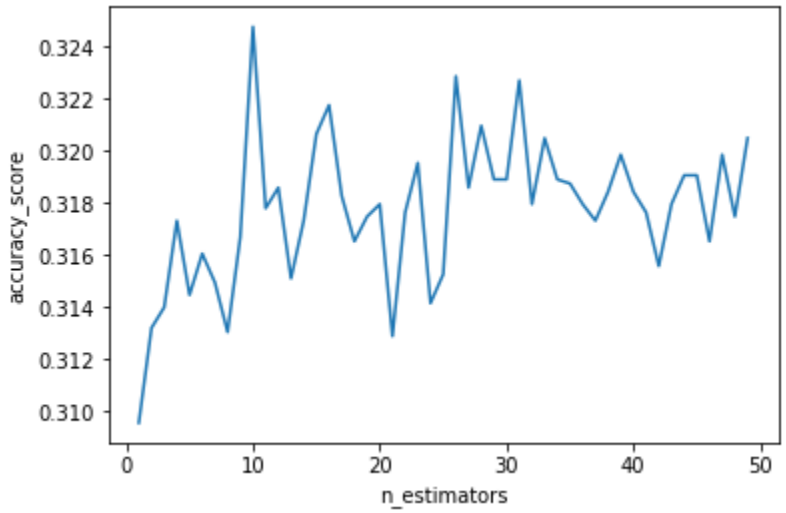
GridSearchCV parameter grids for both the random forest and gradient booster are shown below.

```
param_grid_rf = {
    'n_estimators': [200, 700],
    'max_features': ['sqrt', 'log2'],
    'criterion' : ['gini', 'entropy'],
    'max_depth' : [4, 8, 10]
}
param_grid_gb = {
    'learning_rate' : [0.001, 0.01, 0.1, 0.2, 0.3],
    'n_estimators' : [10, 25, 50, 100],
}
```

The individual parameter tuning took the form of the below block of code that outputs an image like Figure 1. This example shows the parameter tuning of *n*_estimators.

```
score = []
for number in range(1, 50):
    tree = RandomForestClassifier(n_estimators=number)
    tree.fit(X_train, y_train)
    results = tree.predict(X_test)
    score.append(accuracy_score(y_test, results))
plt.plot(np.arange(1,50,1),score)
plt.xlabel('n_estimators')
plt.ylabel('accuracy_score')
plt.show()
```

Figure 1:



Tuning parameters individually is not as effective as tuning parameters via GridSearchCV or RandomSearchCV, but the individual tuning does show trends of how specific variables impact model performance. Post-tuning both the random forest model and the gradient boosting model saw increased positive prediction rates by 12.5% as shown in Figure 2.

Figure 2:

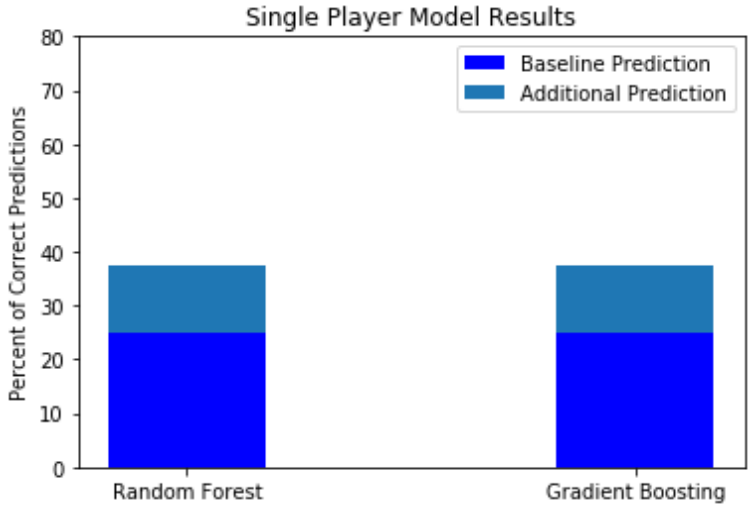
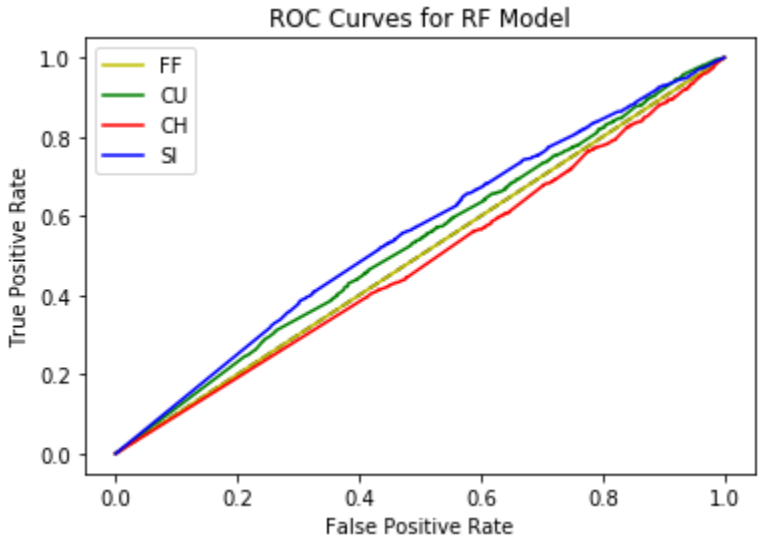


Figure 3 shows the ROC curves for each pitch type for the single player random forest pitch model. ROC curves plot the true positive rate verse the false positive rate for all of the test data.

Figure 3:



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: four-seam fastball, slider, changeup, two-seam fastball, curveball, sinker, and cutter - 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% as shown in Figure 4 and the ROC curves are shown in Figure 5.

Figure 4:

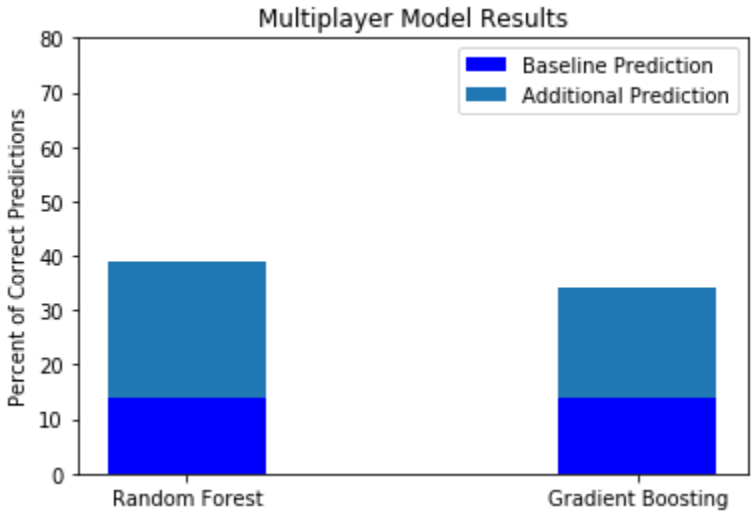
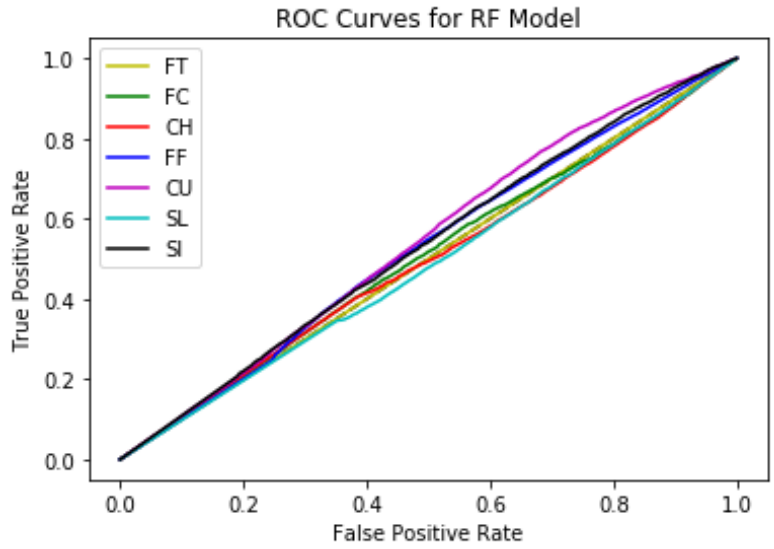


Figure 5:



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 algorithm's 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 significantly. The multi-player model on a whole suffers when fed an equal amount of each pitch. MLB pitchers as a whole rely on the fastball more than any other pitch due to its

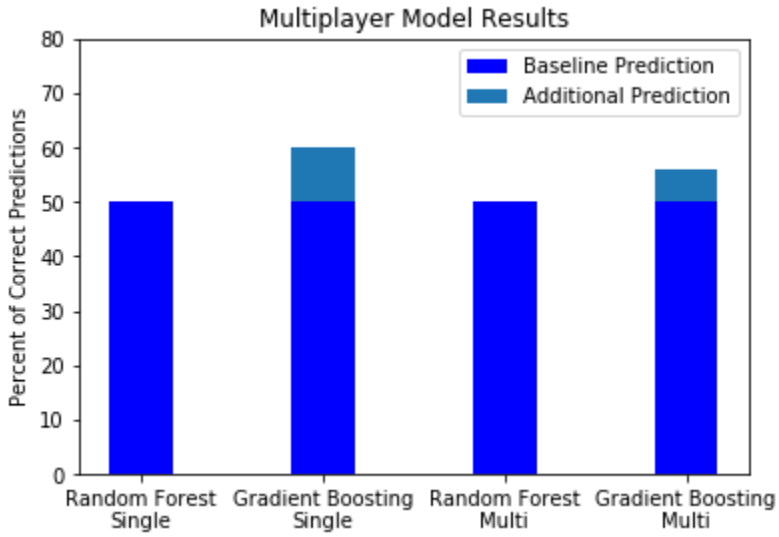
accuracy. This caused the multiplayer model to increase its accuracy by predicting fastball repetitively, but decreased the model's variance.

The final model that I created was the fastball/off-speed pitch model. This model uses the same variables and algorithms as the previous two models and looks at both single player and multiplayer data. For the single player portion of the model Felix Hernandez was again used as the test subject. The model differs from the previous two by mapping all of Felix's pitches to either fastball or off-speed by using the code shown below.

```
df['pitch_type'] = df['pitch_type'].map(lambda x: int((x != 'FF' and x
!= 'FT' and x != 'SI' and x != 'FC')))
```

The single player untuned random forest model saw barely any predictive improvement over the baseline 50%, but the gradient boosting model did show an increased predictive rate of over 10%. 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%. The results are shown in Figure 6.

Figure 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. However, my current model structure had disappointing results.

Conclusion:

I can easily draw the conclusion that predicting pitches at the MLB level using only game scenario information is either truly random or needs to include further information (like batter data). 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 – especially because using only game scenario data is just the tip of the iceberg. 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.