**Scraper**

Pitchfx is a pitch tracking system that records many pitch features including velocity, movement, spin rate etc. for every pitch thrown in a game. The system is installed in every MLB Stadium and has being used since 2006 (Fangraphs, 2016). Given the project aim, Pitchfx Data is the perfect source for building prediction model on. However, although there are lots of online sources in order to scrape Pitchfx Data (e.g Brooks Baseball, Jeff Zimmerman’s website, PitchRx Package in R), perhaps the first challenge we faced as a group was finding the right source. First, we tried to use PitchRx package given some of us has used R, we did not construct the data we wanted. More precisely, **pitch** and **atbat** datasets were not comprehensive enough to merge and use for our later analysis. Second, the dataset itself is publically available through [MLB Gameday Directory](http://gd2.mlb.com/components/game/mlb/), and includes 17 different sub-directories. Within it, there are number of XML files: one for each inning, one that combines all innings, one that only gives (less than useful) hit information, and one for scoring plays. We wanted to scrape the directory using BeautifulSoup, but it did not work either. Last, after browsing through the internet, we came across to a scraper that, fortunately, gives what we want. The scraper reports the below features and then outputs two files called **pitch\_table.csv**, with one line per pitch and **atbat.csv** with descriptive information for each plate appearance in the pitch file (beyondtheboxscore, 2016).

* Game ID
* Flags
  + Spring Training
  + Regular Season
  + Playoffs etc.
* The MLB Game ID
* Game Location Data
* Batter/Pitcher ID data
* Game Situation Data
* Pitch outcome sequence up to that point in the plate appearance
* A flag to designate if the pitch is the last pitch in the plate appearance
* Retrosheet-style event code

http://www.beyondtheboxscore.com/2015/9/24/9374949/a-new-python-based-pitchf-x-parser-scraper

http://www.fangraphs.com/library/misc/pitch-fx/

**Game Type Description**

To recall, Game Type Description feature of the scraped dataset includes *Divisional Series, LCS, Regular Season, Wild-Card Game, World Series, Spring Training* and *Unknown*. In order to keep our analysis limited to official games and lower the number of NaNs, we disregarded the pitches thrown in *Spring Training* and *Unknown* game types*.*

**Creating Dummy Variables**

After cleaning the dataset for running different types of models (especially Logistic Regression Model requires some feature engineering), we ended up with 7 columns that has more than 2 different classes:

* Pitch Type
* Batter Position
* Months
* Year
* Outs ct
* Pa Ball ct
* Pa Strike ct

Using **One Hot Encoding,** we created k dummy variables to be included in the logistic regression model.

**Normalization**

Although many of our features are between 0 and 1 (mostly percentages), we have 7 other columns needed be feature scaled in a scale of 0 to 1. The below features were normalized using **minmax**. As opposed to other normalization approaches, minmax will give us a result with smaller standard deviations, which can suppress the effect of outliers.

* Pitcher Age
* Cumulative Pitch Count
* HR9
* SO/BB
* WHIP
* ERA
* Pitches Per Pa

**Decision Tree**

As our second model option (started to look for other models that could outperform Logistic Regression), we have run several **decision tree** models for different hyper parameters including *maximum depth, minimum leaf size* and *minimum split size*. Arbitrarily selected below parameters, gives us the resulting table for classification score.

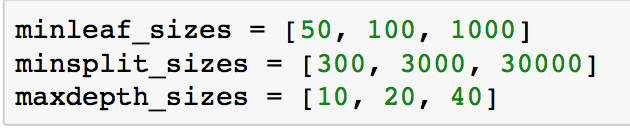


Figure 1

**

Figure 2

For each *min leaf size,* the smaller the *min split size* the better the score. Moreover, higher *depths* tend to give higher scores. Last, for each *min split size,* the smaller the *min leaf size* the better the score. Given above values, the highest score we got was **0.485**.

Let us change the hyper parameters a little bit. Figure 4 shows that the highest score we got so far is **0.486** and it has a *depth* of 30 and *min split size* of 300.

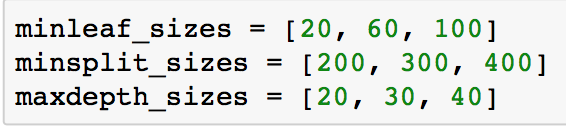


Figure 3



Figure 4

After settling on max depth and min split size as 20, 300 respectively, let us look for the optimal min leaf size. It looks like the smaller *min leaf size* is going to perform better:

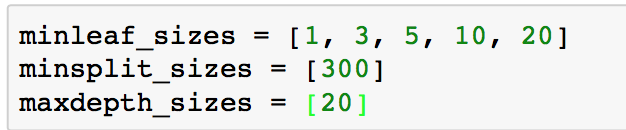


Figure 5



Figure 6

To conclude, the best result is achieved using *max depth size* 20, *min leaf size* 3 and *min split size* 300. The below figure summarizes the feature importance for our final decision tree models. The top three most important features are **pct\_SI, pct\_SL** and **pct\_FT.**

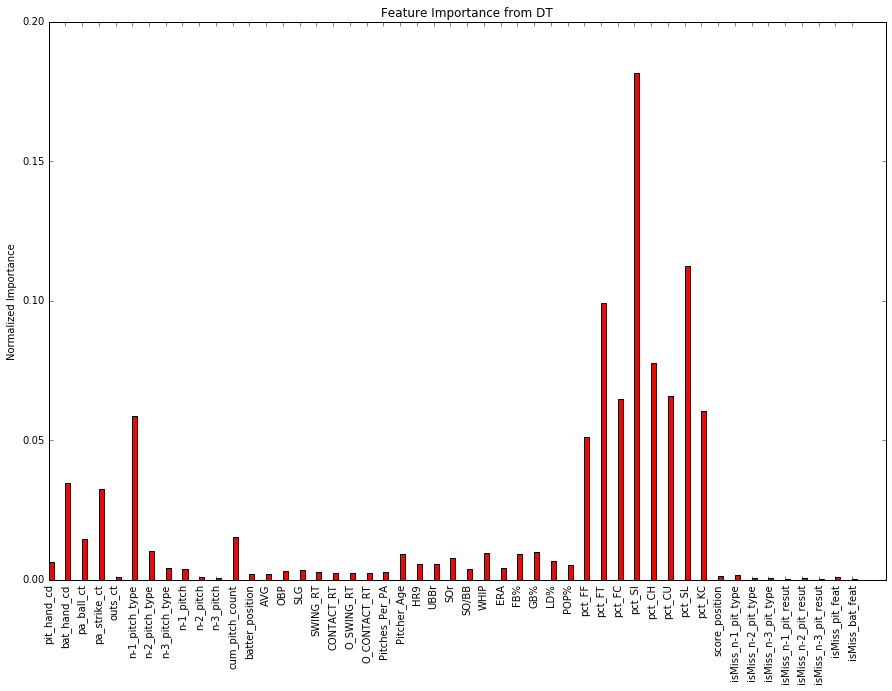


Figure 7