Phillies QA Questionnaire

3/22/2021

Part A

Introduction

This document describes the proposed analysis to answer the question proposed by our Major League infield coach. The question to be answered is below:

"One of our infielders, Player X, seems to be struggling in the field. He's got a great arm, but he's made a few errors this season and is failing to get to some balls. Could you look into this and identify any problem areas that we can target with drills?"

Potential Approaches

To better investigate what is causing our player to struggle fielding I can look at a few different questions, including:

- What are the batted ball profiles of hits toward this player? If we can see which types of balls/hits are not resulting in a play being made we can design drills to attack this deficiency and strengthen it.
- Where is the player positioned prior to and at pitch delivery? If we see that this player is not setting themselves up for success with their positioning (too far to the glove, or arm, side) we can focus on improvement in this regard.
- What plays are not being made? By reviewing the Baseball Savant Outs Above Average (OAA) Leaderboard we can see if our player has less success going in a certain direction and can use drills to work on these plays.

In this analysis I will explore all three of the above approaches.

Data

The data I will use for the analysis of our infielder is a collection of batted ball data, from Trackman, fielder movement data, and fielder positioning data. This data is available via Baseball Savant and Trackman data. An example of some of the data to review and be used in this analysis is below in **Table 1**.

```r
kbl(cbind(Variables, Description), booktabs = T, centering = T,
    caption = "Fielding Data of Interest") %>%
  kable_styling(latex_options =  c("striped", "scale_down"),
                full_width = F) %>%
  column_spec(1, bold = T, color = "black") %>%
  column_spec(2, italic = T, width = "30em")
```

Table 1: Fielding Data of Interest

Variables	Description
Exit Velocity	*The speed of the ball as it leaves the bat in miles per hour*
Launch Angle	*How steeply the ball leaves the bat as an angle (up or down)*
Hit Spin Rate	*How quickly the ball is spinning as it leaves the bat in revolutions per minute*
Distance	*Estimated carry distance (ball in the air) of a batted ball (ft.)*
Bearing	*Where the ball would have landed had it not been obstructed/caught, measured in degrees with 0 being a straight line from home to second base*
OAA	*Outs Above Average: the cumulative effect of all individual plays a fielder has been credited or debited with (helps to measure range)*
In	*OAA on plays made in front of fielder's starting position*
Right	*OAA on plays made to a fielder's right*
Left	*OAA on plays made to a fielder's left*
Back	*OAA on plays made behind a fielder's starting position*
RHB	*OAA on right handed batters*
LHB	*OAA on left handed batters*
Success Rate	*Percentage of plays made successfully*
Depth	*How far from home plate a fielder starts (measured in feet)*
Postion Angle	*Angle of fielder at the start meausred in degrees (0 would be a straight line from the playe to second base)*
First Step	*Time after ball is hit before fielder moves*
Speed	*How fast a fielder moves when pursuing the ball (mph)*
Exchange Time	*Time it takes from catch to throw*
playID	*Unique ID for each play*

Potential Issues

When we review the dataset and the variables of interest I notice that there could be some limitations in how we can review and use the Outs Above Average metric. Since our player is struggling to get to some balls, this variable could be skewed. We could also run into issues with consistency in the player positioning data because this is an average based on the filters applied on the Baseball Savant web page and may not be as accurate as we would like it to be when studying the data.

Some of the Trackman data I hope to use in this analysis also posses an issue because "Distance" and "Bearing" are both estimated and extrapolated based on other variables and could become an issue if the system is not calibrated or measurements are not calculated properly, reducing accuracy. This limitation will be necessary to keep in mind as we analyze and clean the original data pull, but the measurements from Trackman are essential to our analysis and drill design to help our infielder improve.

Methodology

With the potential data issues above the best path forward seems to be analyzing the starting position of our infielder as well as reviewing the movement patterns our player displays. We can review all three questions in greater detail after focusing on the player's positioning.

The first step will be to visualize our positioning data and determine if there are any trends in where our infielder is starting both in depth and angle. I would then look to overlay the batted ball location information and match them to see how far he is having to move to make a play. After reviewing the visual, I would look to include the OAA information to better understand potential directional limitations of our infielder. Finally, if we are not able to identify adjustments and drills to help our infielder we should look at the batted ball information to determine how large of a role the pitcher/batter interaction is affecting our fielder. This analysis could go much deeper and be applied across the team if requested, but should provide a strong first draft of analysis requested by the coach.

Resources

- https://trackman.zendesk.com/hc/en-us/articles/115002776647-Radar-Measurement-Glossary-of-Terms - Information for batted ball statistics
- https://baseballsavant.mlb.com/visuals/fielder-positioning - Information for fielder positioning
- https://baseballsavant.mlb.com/leaderboard/outs_above_average?type=Fielder&year=2020&team=&range=year&min=q&pos=if&roles=&viz=show - Motivation for seeing which plays are made and which ones are not

---

Part B

Summary

I was tasked with predicting a batter's 2018 end of season batting average given their batting performance in March and April of that same year. The model I developed used a stepwise regression approach and ended up predicting Full Season batting average with an error of about 25 points (0.025) on the holdout sample. This model was an improvement over the simplest model I could think of, batting average in March/April predicting end of season model, by about 1 point on the holdout.

Cleaning

- Checked the summaries of the variables in the data frame to ensure no missing values
- Verified the distributions of each variable to ensure they were not skewed
- Corrected Jace Peterson's team for the 2018 season to be the Yankees (verified with mlb.com)
- Split the data into training and holdout samples; using a stratified random 80% proportion by Teams

Model Development

- Started by fitting vanilla linear regression model without interactions
- Stepped up to develop regularized regression, partition models, random forest, and a boosted tree model
- All of those models had about the same final results on the holdout with the regularized regression having the best RMSE but the other models were within one standard deviation and we cannot distinguish which were better
- After running through these more advanced models, I decided to test more basic linear regression models to see if these models struggled or did well
- The base model of March and April Batting Average predicting Full Season, actually predicts just as well if not better than most of the more "advanced" models; this is somewhat disappointing overall but I then wanted to check out regression with interactions and a stepwise algorithm
- Using a stepwise algorithm, with a full model of all predictors and two-way interactions and a naive model with 1 as the predictor, I was able to identify a linear model that reduced RMSE below the "advanced" models and the base linear model
- The best linear model used variables that basically recalculated the batting average from March and April, and added information to help explain how often a batter makes contact and puts the ball in play
- Including interactions greatly improved the model

Conclusions

During this model development I found many different ways to attempt to predict batting average. The best model ended up being a descriptive linear model that included interactions between the variables. Since a majority of our models included variables that were able to recalculate batting average these were the most impressive during my stepwise analysis.

Future Recommendations

- Include more variables that are solely controlled by the batter's actions during plate appearances, such as Exit Velocity and Launch Angle
- Include interactions in more advanced modeling techniques, ie. neural networks, support vector machines, to see how they compared to the linear models developed here
- Include previous seasons' batting averages/metrics (along with player level) to see how they can improve our model via machine learning and time series forecasting

---

```r
batting <- read.csv("batting.csv", header=T)
```

Goal: Predict final batting average in the 2018 season based on various batting statistics from March/April 2018

Step 1: Data Cleaning/Validation

```r
batting.original <- batting
table(batting$Team)
```

```
##
##         - - -       Angels       Astros    Athletics    Blue Jays       Braves
##             1           11           11           10           14            7
##       Brewers    Cardinals         Cubs Diamondbacks      Dodgers       Giants
##            11           11           11           11           10           12
##       Indians     Mariners      Marlins         Mets    Nationals      Orioles
##            10           10            8            8           10            8
##        Padres     Phillies      Pirates      Rangers         Rays      Red Sox
##            10           10           11           11           11           11
##          Reds      Rockies       Royals       Tigers        Twins    White Sox
##            11           10            8           11           10           10
##       Yankees
##            11
```

```
# update Jace Peterson team to be his 2018 team (Yankees)
  # [verified on mlb.com https://www.mlb.com/player/jace-peterson-607054]
batting[which(batting$Team == "- - -"),"Team"] <- "Yankees"
batting[which(batting$Name == "Jace Peterson"),]
```

```
##     ï..playerid          Name    Team MarApr_PA MarApr_AB MarApr_H MarApr_HR
## 211       12325 Jace Peterson Yankees        33        28        6         0
##     MarApr_R MarApr_RBI MarApr_SB MarApr_BB. MarApr_K. MarApr_ISO MarApr_BABIP
## 211        1          4         3      0.121     0.273      0.071        0.316
##     MarApr_AVG MarApr_OBP MarApr_SLG MarApr_LD. MarApr_GB. MarApr_FB.
## 211      0.214      0.333      0.286      0.158      0.579      0.263
##     MarApr_IFFB. MarApr_HR.FB MarApr_O.Swing. MarApr_Z.Swing. MarApr_Swing.
## 211          0.2            0           0.184           0.685         0.413
##     MarApr_O.Contact. MarApr_Z.Contact. MarApr_Contact. FullSeason_AVG
## 211               0.5              0.82           0.742            0.2
```

```
table(batting$Team)
```

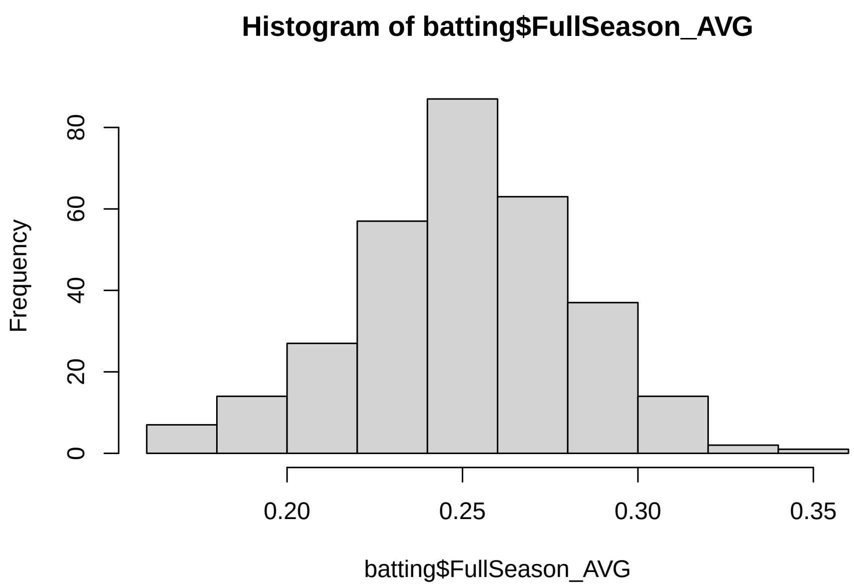
```
##
##       Angels       Astros    Athletics    Blue Jays       Braves      Brewers
##           11           11           10           14            7           11
##    Cardinals         Cubs Diamondbacks      Dodgers       Giants      Indians
##           11           11           11           10           12           10
##     Mariners      Marlins         Mets    Nationals      Orioles       Padres
##           10            8            8           10            8           10
##     Phillies      Pirates      Rangers         Rays      Red Sox         Reds
##           10           11           11           11           11           11
##      Rockies       Royals       Tigers        Twins    White Sox      Yankees
##           10            8           11           10           10           12
```

```
summary(batting) # no NA values to worry about
```

```
##   ï..playerid        Name               Team             MarApr_PA
##  Min.   :  393   Length:309         Length:309         Min.   : 30.00
##  1st Qu.: 5933   Class :character   Class :character   1st Qu.: 61.00
##  Median :10346   Mode  :character   Mode  :character   Median : 91.00
##  Mean   : 9966                                         Mean   : 87.03
##  3rd Qu.:13157                                         3rd Qu.:112.00
```

```
##  Max.   :19755                                         Max.   :137.00  
##    MarApr_AB          MarApr_H       MarApr_HR         MarApr_R    
##  Min.   : 22.00   Min.   : 3.00   Min.   : 0.000   Min.   : 1.00  
##  1st Qu.: 54.00   1st Qu.:13.00   1st Qu.: 1.000   1st Qu.: 6.00  
##  Median : 80.00   Median :19.00   Median : 2.000   Median :10.00  
##  Mean   : 77.05   Mean   :19.47   Mean   : 2.667   Mean   :10.59  
##  3rd Qu.: 99.00   3rd Qu.:26.00   3rd Qu.: 4.000   3rd Qu.:14.00  
##  Max.   :125.00   Max.   :41.00   Max.   :10.000   Max.   :29.00  
##    MarApr_RBI      MarApr_SB         MarApr_BB.         MarApr_K.     
##  Min.   : 0.00   Min.   : 0.000   Min.   :0.00000   Min.   :0.0330  
##  1st Qu.: 6.00   1st Qu.: 0.000   1st Qu.:0.06000   1st Qu.:0.1620  
##  Median : 9.00   Median : 1.000   Median :0.08800   Median :0.2160  
##  Mean   :10.04   Mean   : 1.227   Mean   :0.09127   Mean   :0.2184  
##  3rd Qu.:14.00   3rd Qu.: 2.000   3rd Qu.:0.11800   3rd Qu.:0.2560  
##  Max.   :30.00   Max.   :13.000   Max.   :0.29000   Max.   :0.5230  
##    MarApr_ISO      MarApr_BABIP      MarApr_AVG      MarApr_OBP    
##  Min.   :0.0000   Min.   :0.1190   Min.   :0.106   Min.   :0.1560  
##  1st Qu.:0.0970   1st Qu.:0.2530   1st Qu.:0.213   1st Qu.:0.2810  
##  Median :0.1560   Median :0.2980   Median :0.250   Median :0.3260  
##  Mean   :0.1647   Mean   :0.2973   Mean   :0.250   Mean   :0.3256  
##  3rd Qu.:0.2220   3rd Qu.:0.3420   3rd Qu.:0.290   3rd Qu.:0.3670  
##  Max.   :0.4080   Max.   :0.5000   Max.   :0.484   Max.   :0.5290  
##    MarApr_SLG       MarApr_LD.       MarApr_GB.      MarApr_FB.    
##  Min.   :0.1290   Min.   :0.0000   Min.   :0.194   Min.   :0.0940  
##  1st Qu.:0.3280   1st Qu.:0.1690   1st Qu.:0.358   1st Qu.:0.3060  
##  Median :0.4100   Median :0.2030   Median :0.414   Median :0.3640  
##  Mean   :0.4147   Mean   :0.2088   Mean   :0.423   Mean   :0.3683  
##  3rd Qu.:0.4950   3rd Qu.:0.2430   3rd Qu.:0.488   3rd Qu.:0.4340  
##  Max.   :0.7450   Max.   :0.3850   Max.   :0.733   Max.   :0.6670  
##   MarApr_IFFB.    MarApr_HR.FB    MarApr_O.Swing.  MarApr_Z.Swing. 
##  Min.   :0.000   Min.   :0.0000   Min.   :0.1210   Min.   :0.4350  
##  1st Qu.:0.048   1st Qu.:0.0560   1st Qu.:0.2430   1st Qu.:0.6220  
##  Median :0.100   Median :0.1110   Median :0.2970   Median :0.6620  
##  Mean   :0.109   Mean   :0.1211   Mean   :0.2976   Mean   :0.6657  
##  3rd Qu.:0.154   3rd Qu.:0.1820   3rd Qu.:0.3460   3rd Qu.:0.7150  
##  Max.   :0.625   Max.   :0.5290   Max.   :0.5220   Max.   :0.8490  
##  MarApr_Swing.    MarApr_O.Contact. MarApr_Z.Contact. MarApr_Contact.
##  Min.   :0.2880   Min.   :0.2110    Min.   :0.6530    Min.   :0.5250  
##  1st Qu.:0.4180   1st Qu.:0.5500    1st Qu.:0.8200    1st Qu.:0.7290  
##  Median :0.4560   Median :0.6320    Median :0.8610    Median :0.7750  
##  Mean   :0.4569   Mean   :0.6265    Mean   :0.8569    Mean   :0.7722  
##  3rd Qu.:0.4970   3rd Qu.:0.6980    3rd Qu.:0.9000    3rd Qu.:0.8170  
##  Max.   :0.6150   Max.   :0.9290    Max.   :1.0000    Max.   :0.9610  
##  FullSeason_AVG  
##  Min.   :0.1650  
##  1st Qu.:0.2330  
##  Median :0.2510  
##  Mean   :0.2509  
##  3rd Qu.:0.2710  
##  Max.   :0.3460  
```

```r
hist(batting$FullSeason_AVG)
```




```r
# all histograms appear to be okay to use without transformations!
  # (Stolen Bases may be an issue but for the moment let's leave it alone)
# since Player_id and Name are identifiers we will remove them from batting
batting$ï..playerid <- NULL
batting$Name <- NULL
table(batting$Team)
```

```
##
##       Angels       Astros    Athletics    Blue Jays       Braves      Brewers
##           11           11           10           14            7           11
##    Cardinals         Cubs Diamondbacks      Dodgers       Giants      Indians
##           11           11           11           10           12           10
##     Mariners      Marlins         Mets    Nationals      Orioles       Padres
##           10            8            8           10            8           10
##     Phillies      Pirates      Rangers         Rays      Red Sox         Reds
##           10           11           11           11           11           11
##      Rockies       Royals       Tigers        Twins    White Sox      Yankees
##           10            8           11           10           10           12
```

```r
# since there is limited representation for each team, we will use stratified sampling
# checking for near zero and zero variance columns
infodensity <- nearZeroVar(batting, saveMetrics= TRUE)
infodensity[infodensity$nzv,] # appears we do not have any ZeroVar or nearZeroVar variables
```

```
## [1] freqRatio     percentUnique zeroVar       nzv
## <0 rows> (or 0-length row.names)
```

```
  # because we return no rows, so let's keep moving!
# checking for highly correlated values
highlycorrelated <- findCorrelation(cor_matrix(batting), cutoff = 0.9)
highlycorrelated # Plate Appearances and Slugging % are highly
```

```
## [1] 14  1
```

```
  # correlated with something else, but I don't feel comfortable removing them
```

Step 2: Create Training and Holdout Samples

```
set.seed(23); TRAIN <- stratified(batting, c("Team"), .8, bothSets = T)$SAMP1
# break batting into training rows (80% team representation)
set.seed(23); HOLDOUT <- stratified(batting, c("Team"), .8, bothSets = T)$SAMP2
# we will use 20% of team representation as the holdout
# code found for stratified sampling via StackOverflow
  # https://stackoverflow.com/questions/23479512/stratified-random-sampling-from-data-frame
```

Step 3: Model Building Time

```
# set up generalization error estimation (using 10-fold cross validation here)
fitControl <- trainControl(method="cv",number=10, allowParallel = TRUE)

# vanilla linear regression
set.seed(23); GLM <- train(FullSeason_AVG~.,data=TRAIN,method='glm',
                          trControl=fitControl,preProc=c("center", "scale") )
GLM$results # RMSE: 0.028707
```

```
##   parameter     RMSE  Rsquared        MAE      RMSESD RsquaredSD       MAESD
## 1      none 0.028707 0.2697597 0.02302975 0.004873621  0.1701305 0.003585668
```

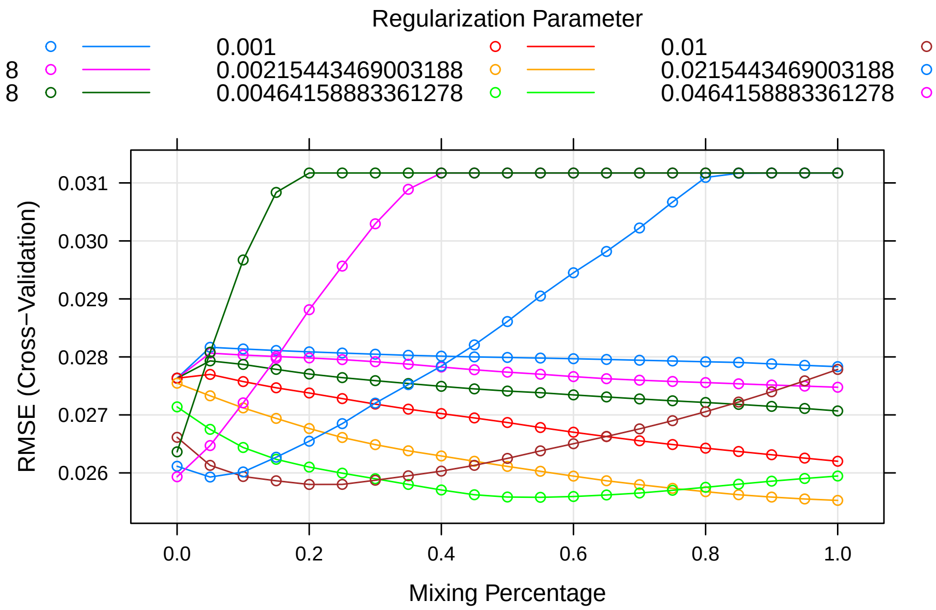
```
postResample(predict(GLM,newdata=HOLDOUT),HOLDOUT$FullSeason_AVG) # 0.02632316
```

```
##       RMSE   Rsquared        MAE
## 0.02632316 0.32723091 0.02186324
```

```
# regularized linear regression
glmnetGrid <- expand.grid(alpha = seq(0,1,.05),lambda = 10^seq(-4,-1,length=10))
set.seed(23); GLMnet <- train(FullSeason_AVG~.,data=TRAIN,method='glmnet', tuneGrid=glmnetGrid,
                             trControl=fitControl, preProc = c("center", "scale"))
```

```
## Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info = trainInfo, :
## There were missing values in resampled performance measures.
```

```
plot(GLMnet) #See how error changes with choices
```




```
GLMnet$bestTune #Gives best parameters
```

```
##     alpha      lambda
## 205     1 0.002154435
```

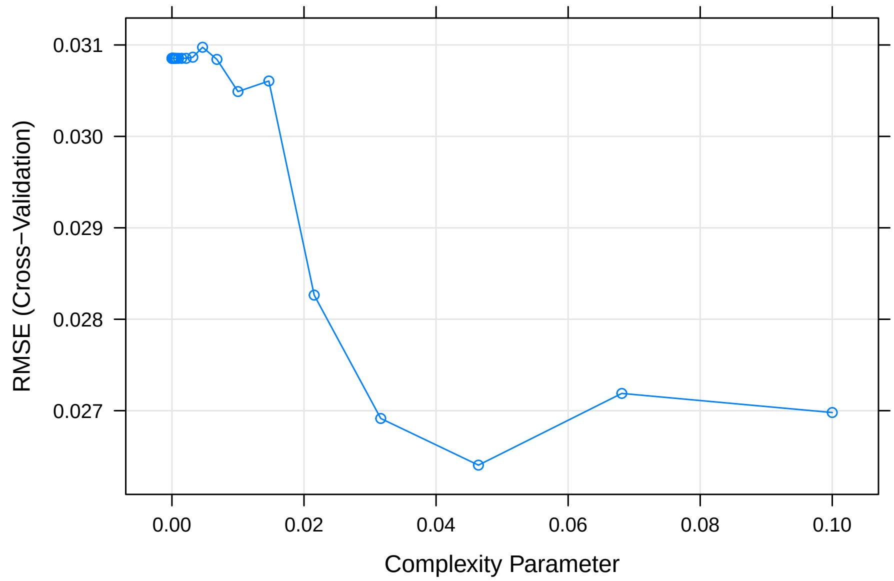
```
GLMnet$results[rownames(GLMnet$bestTune),]  # RMSE: 0.02552582
```

```
##     alpha      lambda       RMSE  Rsquared         MAE      RMSESD RsquaredSD
## 205     1 0.002154435 0.02552582 0.3537571 0.02027919 0.002714302  0.1297308
##           MAESD
## 205 0.002304583
```

```
postResample(predict(GLMnet,newdata=HOLDOUT),HOLDOUT$FullSeason_AVG) # 0.02530690
```

```
##       RMSE   Rsquared        MAE
## 0.02530690 0.39543128 0.02125725
```

```
# vanilla partition
treeGrid <- expand.grid(cp=10^seq(-5,-1,length=25))
set.seed(23); TREE <- train(FullSeason_AVG~.,data=TRAIN,method='rpart', tuneGrid=treeGrid,
                            trControl=fitControl, preProc = c("center", "scale"))
plot(TREE)
```




```
TREE$bestTune
```

```
##            cp
## 23 0.04641589
```

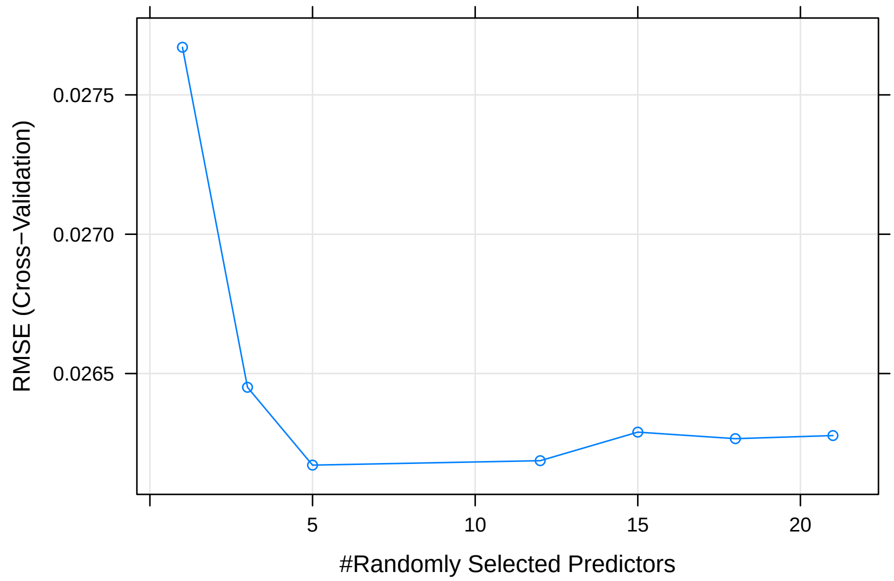
```
TREE$results[rownames(TREE$bestTune),] # RMSE: 0.0264044
```

```
##            cp      RMSE  Rsquared        MAE      RMSESD RsquaredSD       MAESD
## 23 0.04641589 0.0264044 0.3027249 0.02120794 0.002937483  0.1151302 0.002738071
```

```
postResample(predict(TREE,newdata=HOLDOUT),HOLDOUT$FullSeason_AVG) # 0.02675582
```

```
##       RMSE   Rsquared        MAE
## 0.02675582 0.30456307 0.02201361
```

```
# random forest
forestGrid <- expand.grid(mtry=c(1,3,5,12,15,18,21))
cluster <- makeCluster(detectCores() - 1) #parallelization
registerDoParallel(cluster)
FOREST <- train(FullSeason_AVG~.,data=TRAIN,method='rf',tuneGrid=forestGrid,
                               trControl=fitControl, preProc = c("center", "scale"))
stopCluster(cluster)
registerDoSEQ()
plot(FOREST)
```




```
FOREST$bestTune
```

```
##   mtry
## 3    5
```

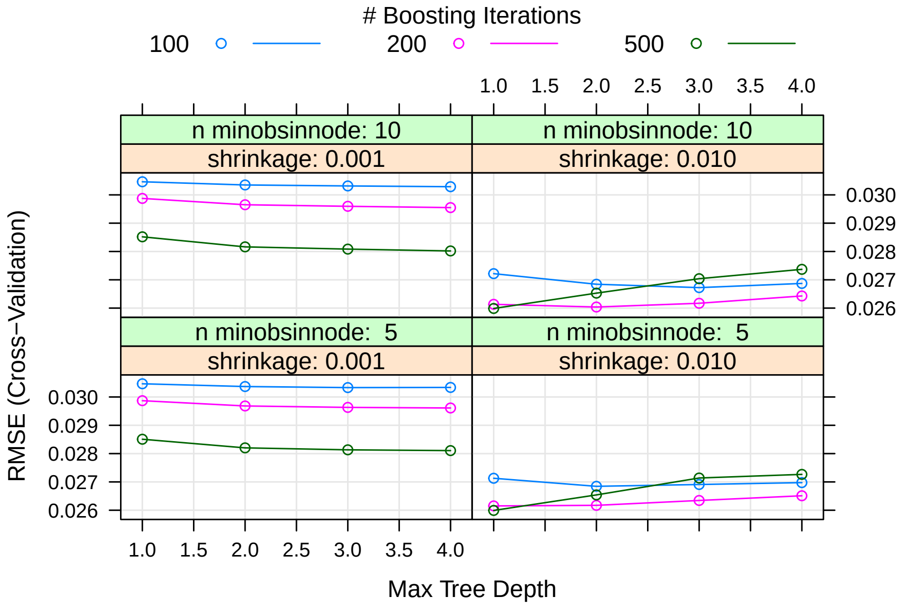
```
FOREST$results[rownames(FOREST$bestTune),] # RMSE: 0.02669615
```

```
##   mtry      RMSE  Rsquared        MAE      RMSESD RsquaredSD       MAESD
## 3    5 0.0261713 0.3271266 0.0207124 0.002594279  0.1451083 0.002541341
```

```
postResample(predict(FOREST,newdata=HOLDOUT),HOLDOUT$FullSeason_AVG) # 0.02663614 with mtry=5
```

```
##       RMSE   Rsquared        MAE
## 0.02660435 0.32759566 0.02208299
```

```
# boosted tree
gbmGrid <- expand.grid(n.trees=c(100,200,500),interaction.depth=1:4,
                       shrinkage=c(.01,.001),n.minobsinnode=c(5,10))
cluster <- makeCluster(detectCores() - 1) # parallelization
registerDoParallel(cluster)
set.seed(23); GBM <- train(FullSeason_AVG~.,data=TRAIN, method='gbm',tuneGrid=gbmGrid,
                           verbose=FALSE,trControl=fitControl, preProc = c("center", "scale"))
stopCluster(cluster)
registerDoSEQ()
plot(GBM)
```




```
GBM$bestTune
```

```
##    n.trees interaction.depth shrinkage n.minobsinnode
## 30     500                 1      0.01             10
```

```
GBM$results[rownames(GBM$bestTune),] # RMSE: 0.02598728
```

```
##    shrinkage interaction.depth n.minobsinnode n.trees       RMSE  Rsquared
## 30      0.01                 1             10     500 0.02598728 0.3270281
##           MAE      RMSESD RsquaredSD       MAESD
## 30 0.02063013 0.00295313  0.1138356 0.002346447
```

```
postResample(predict(GBM,newdata=HOLDOUT,n.trees=500),HOLDOUT$FullSeason_AVG)
```

```
##       RMSE   Rsquared        MAE
## 0.02553700 0.37783132 0.02146369
```

```
# 0.02553700  with ntrees=500
```

Step 4: Choosing the best model of those tested

When we look at the RMSE of the models estimated via cross validation, we see the boosted tree, random forest, vanilla partition, and regularized linear all are within 1 standard deviation of the best RMSE value. Let's check and see how much each model overfit, by calculating % increase in RMSE.

```
# Regularized Linear
(0.02530690 - 0.02552582)/0.02552582 # -0.9%
```

```
## [1] -0.008576414
```

```r
# Vanilla Partition
(0.02675582 - 0.0264044)/0.0264044 # 1.3%
```

```
## [1] 0.01330915
```

```r
# Random Forest
(0.02663614 - 0.02669615)/0.02669615 # -0.2%
```

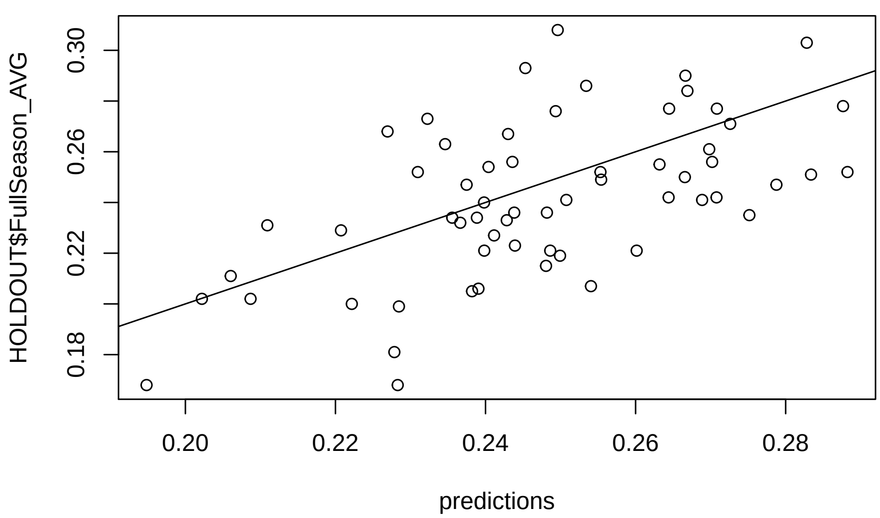
```
## [1] -0.00224789
```

```r
# Boosted Tree
(0.02553700 - 0.02598728)/0.02598728 # -1.7%
```

```
## [1] -0.01732694
```

All the models appear to not overfit the Holdout data, based on the general guideline of a less than 10% increase being solid. Let's check the best models out and test step-wise regression and base models.

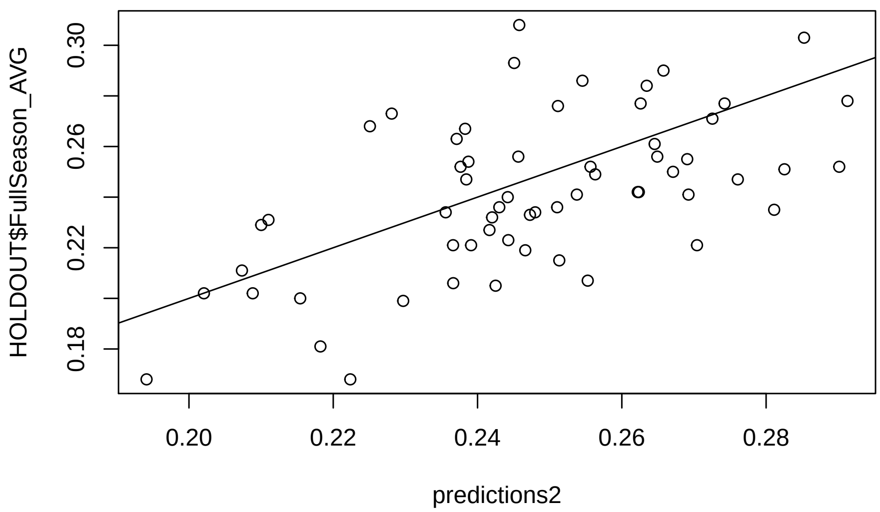
```r
FINAL <- train(FullSeason_AVG~., data=TRAIN, method="glmnet",
               tune.grid=expand.grid(alpha=0.9,lambda=0.004641589),
               trControl=trainControl(method="none"),
               preProc=c("center", "scale"))
predictions <- predict(FINAL, HOLDOUT)
plot(y=HOLDOUT$FullSeason_AVG, x=predictions); abline(0,1)
```




```r
sqrt(mean((predictions-HOLDOUT$FullSeason_AVG)^2)) # RMSE: 0.0261405
```

```
## [1] 0.0261405
```

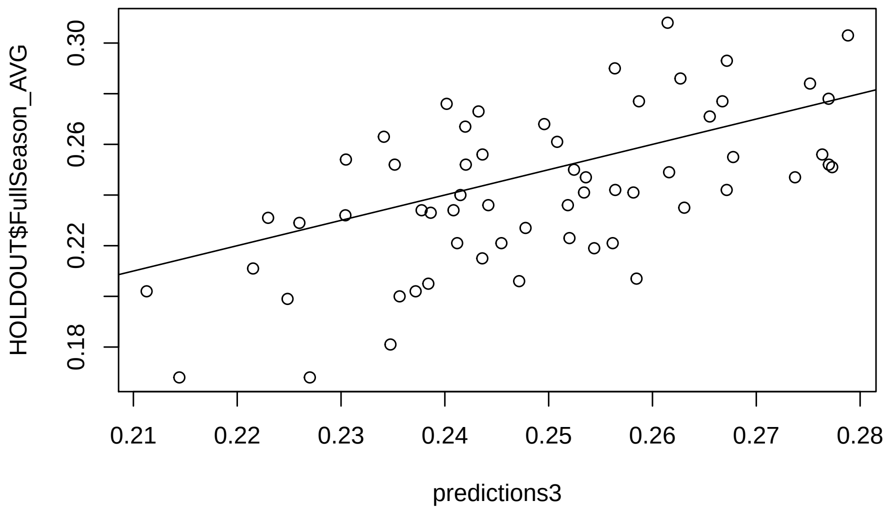
```r
FINAL2 <- train(FullSeason_AVG~., data=TRAIN, method="glm",
                trControl=trainControl(method="none"),
                preProc=c("center", "scale"))
predictions2 <- predict(FINAL2, HOLDOUT)
plot(y=HOLDOUT$FullSeason_AVG, x=predictions2); abline(0,1)
```




```r
sqrt(mean((predictions2-HOLDOUT$FullSeason_AVG)^2)) # RMSE: 0.02632316
```

```
## [1] 0.02632316
```

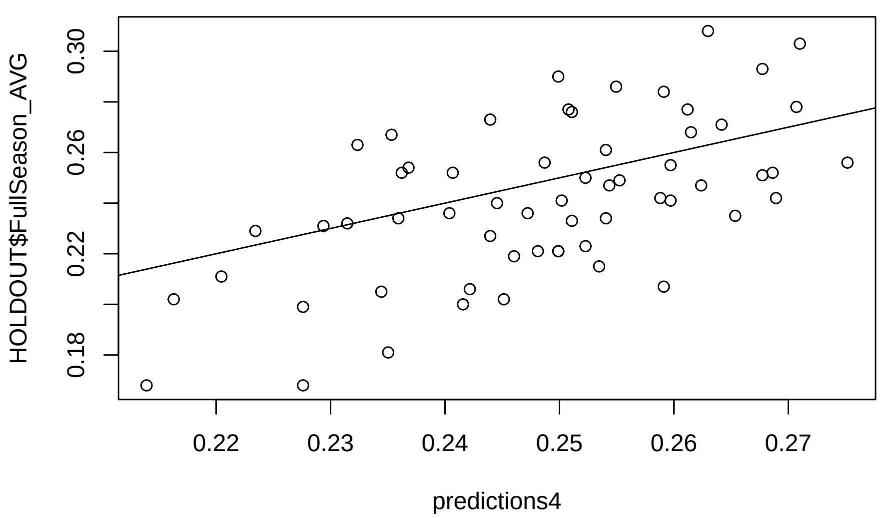
```r
predictions3 <- predict(GBM, HOLDOUT)
plot(y=HOLDOUT$FullSeason_AVG, x=predictions3); abline(0,1)
```




```
sqrt(mean((predictions3-HOLDOUT$FullSeason_AVG)^2)) # RMSE: 0.025537
```

```
## [1] 0.025537
```

```
FINAL4 <- lm(FullSeason_AVG~MarApr_AVG, data = TRAIN)
predictions4 <- predict(FINAL4, HOLDOUT)
plot(y=HOLDOUT$FullSeason_AVG, x=predictions4); abline(0,1)
```

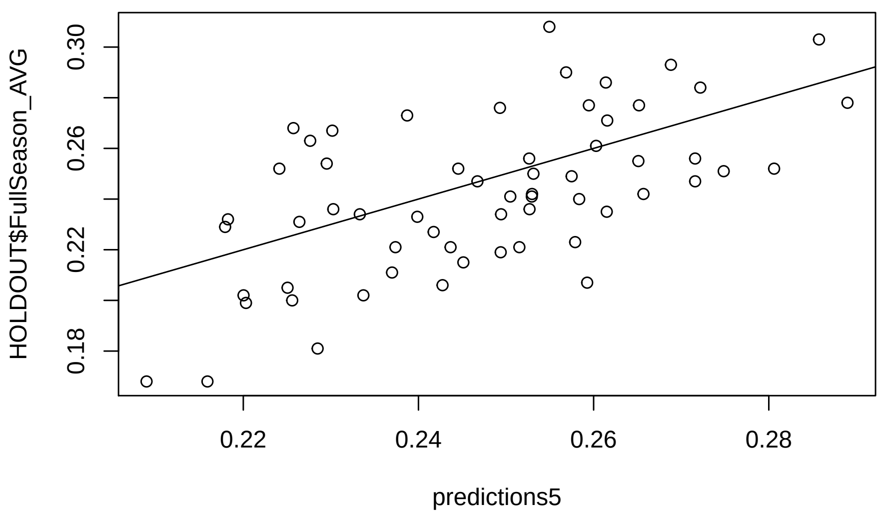


```r
sqrt(mean((predictions4-HOLDOUT$FullSeason_AVG)^2)) # RMSE: 0.02602793
```

```
## [1] 0.02602793
```

```r
full <- lm(FullSeason_AVG~.^2, data=TRAIN)
naive <- lm(FullSeason_AVG~1, data=TRAIN)
S <- step(naive,scope=list(lower=naive,upper=full),direction="both",trace=0)

FINAL5 <- lm(formula(S), data=TRAIN)
predictions5 <- predict(FINAL5, HOLDOUT)
plot(y=HOLDOUT$FullSeason_AVG, x=predictions5); abline(0,1)
```




```r
sqrt(mean((predictions5-HOLDOUT$FullSeason_AVG)^2)) # RMSE: 0.02517612
```

```
## [1] 0.02517612
```

```r
summary(S)
```

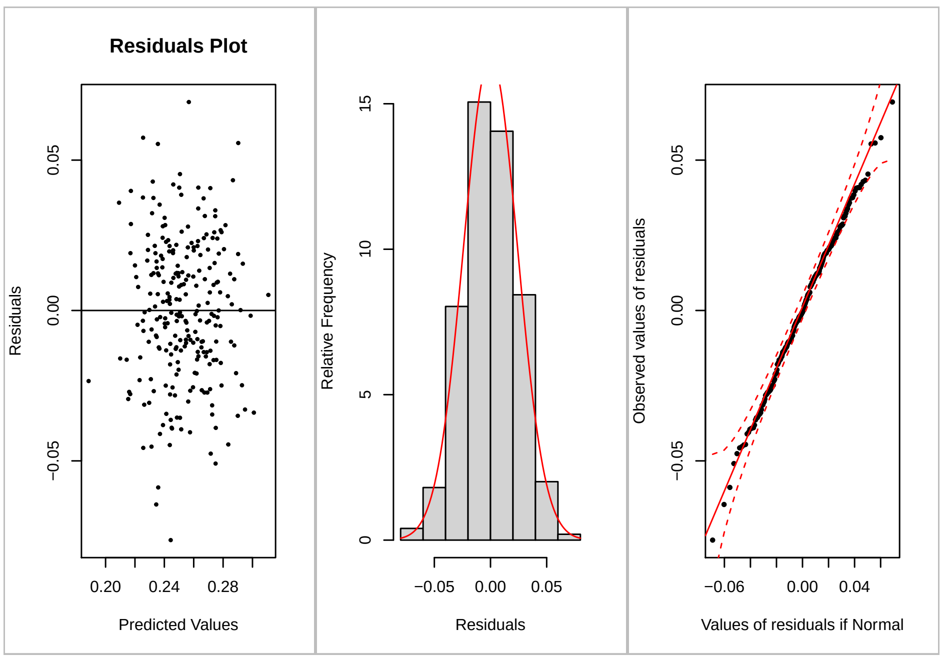
```
##
## Call:
## lm(formula = FullSeason_AVG ~ MarApr_H + MarApr_AB + MarApr_K. +
##     MarApr_IFFB. + MarApr_O.Swing. + MarApr_K.:MarApr_IFFB. +
##     MarApr_K.:MarApr_O.Swing., data = TRAIN)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.076337 -0.015278 -0.000589  0.017750  0.069310
##
## Coefficients:
##                               Estimate Std. Error t value Pr(>|t|)
```

```
## (Intercept)                  0.2926133  0.0190930  15.326  < 2e-16 ***
## MarApr_H                     0.0037914  0.0004143   9.151  < 2e-16 ***
## MarApr_AB                   -0.0008007  0.0001266  -6.323 1.24e-09 ***
## MarApr_K.                   -0.2791495  0.0822038  -3.396   0.0008 ***
## MarApr_IFFB.                -0.1375703  0.0541375  -2.541   0.0117 *
## MarApr_O.Swing.             -0.0802939  0.0617812  -1.300   0.1950
## MarApr_K.:MarApr_IFFB.       0.4136663  0.2290789   1.806   0.0722 .
## MarApr_K.:MarApr_O.Swing.    0.5683067  0.2649847   2.145   0.0330 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.02435 on 241 degrees of freedom
## Multiple R-squared:  0.4117, Adjusted R-squared:  0.3946
## F-statistic: 24.09 on 7 and 241 DF,  p-value: < 2.2e-16
```

```
VIF(S)
```

```
##                  MarApr_H                 MarApr_AB                 MarApr_K.
##                  4.908065                  4.506776                 15.739052
##              MarApr_IFFB.           MarApr_O.Swing.    MarApr_K.:MarApr_IFFB.
##                  8.230021                  7.945356                  8.819492
## MarApr_K.:MarApr_O.Swing.
##                 25.960304
```

```
check_regression(S)
```




```
##
## Tests of Assumptions: ( sample size n = 249 ):
## Linearity
##    p-value for MarApr_H : 0.2444
##    p-value for MarApr_AB : 0.7438
```

```
##    p-value for MarApr_K. : 0.6969
##    p-value for MarApr_IFFB. : 0.4901
##    p-value for MarApr_O.Swing. : 0.8987
##    p-value for MarApr_K.:MarApr_IFFB. : 0.9847
##    p-value for MarApr_K.:MarApr_O.Swing. : 0.3224
##    p-value for overall model : NA (not enough duplicate rows)
## Equal Spread:  p-value is 0.8921
## Normality:  p-value is 0.9476
##
## Advice:  if n<25 then all tests must be passed.
## If n >= 25 and test is failed, refer to diagnostic plot to see if violation is severe
##  or is small enough to be ignored.
```

When we compare our more advanced modeling techniques with the base/simplest model we see they do not do much better. If we include interactions and use a descriptive linear regression, found via the stepwise algorithm, we build a model that makes the best predictions. All of my models appear to have strong residual plots (seen above) because of the randomness of the points compared to the y=x line; this leads to be comfortable with any of the above models in predicting full season batting average, but our best choice is still the stepwise model as its regression check looks picturesque.