Regression and Classification Analysis: Pitcher Performance

Ricky Trujillo

10/14/2020

1 Abstract

This article evaluates a collective group of pitchers' pitching performance during the 2008 to 2019 Major League Baseball(MLB) seasons. A pitcher's Estimated Run Average (ERA) is crucially important to look at when assessing pitching performance as a lower ERA indicates a better assessment of the given pitcher.

For the Regression case, I will be taking a look at which variables from the given dataset causes a pitcher's ERA to fluctuate for better or worse. In doing so, I am immediately assuming non-linearity in the dataset and applying Principal Component Analysis with Partial Least Squares Regression, Neural Networks, and Random Forests to study and analyze the relationships the other predictors have to the ERA response.

For the Classification case, I will be creating quantiles of the data that are split according to 4 intervals dependent on the ERA scale. Each quantile will have an associated label such as Strong Pitcher, Fairly Good Pitcher, Weak Pitcher, and Pitcher in Trouble. Various classification models will be constructed to assess the ability in predicting the same labels in the testing data. The methods of interest will be Naive Bayes, K-Nearest Neighbors, and Support Vector Machines

2 Baseball-Reference Data

2.1 Baseballr package

" baseballr is a package written for R focused on baseball analysis. It includes functions for scraping various data from websites, such as FanGraphs.com, Baseball-Reference.com, and baseballsavant.com. It also includes functions for calculating metrics, such as wOBA, FIP, and team-level consistency over custom time frames."

The latest package, baseballr 0.6.0, published by Bill Petti had its latest version release on January 07, 2020 documentation (http://billpetti.github.io/baseballr/about/)

2.2 Variable Glossary

bbref_id : Baseball Reference ID for Players

season : Respective year to each season time period

Name : Name of Pitcher

Age : Age of Pitcher

Level : League of Pitcher (i.e. MLB National League and/or MLB American League)

Team : MLB Team of Pitcher

G : Number of Games Pitched

GS : Grand Slam

W : Number of games won

L : Number of games lost

SV : Save

IP : Number of Innings Pitched

H : Number of Hits

R : Number of Runs

ER : Earned Run

uBB : unintentional base on ball

BB : Base on balls (or walks)

SO : Strikeouts

HR : Number of Homeruns

HBP : Hit-by-pitch

ERA : Earned Run Average

AB : At-Bat

X1B : Number of Singles

X2B : Number of Doubles

X3B : Number of Triples

IBB : Intentional Base on Ball

GDP : Ground into Double Play

SF : Sacrifice Fly

SB : Stolen Base

CS : Caught Stealing

PO : Putout

BF : Batters Faced

Pit : Number of Pitches

Str : Number of Strikes

StL : Strikes per Loss

StS : Strikes per Save

GB.FB : Ground Ball/ Fly Ball

LD : Line Drives

PU : Popup

WHIP : Walk and Hits per Innings Pitched

BAbip : Batting Average on balls in play

SO9 : Average Strikeout over 9 innings

SO.W : Strikeouts before another Walk

SO_perc : Strikeout Percentage

uBB_perc : Unintentional Base on Ball Percentage

SO_uBB : Strike outs to normal walks ratio

3 Baseballr Data Setup

3.1 Establishing the Dataframe

The data at hand will be obtained, as mentioned, through the 'baseballr' package with the provided function "daily_pitcher_bref()". This function takes in a start date and an end date to web-scrape pitcher data from all MLB teams over the course of the respective time period. Here, the data gathered ranged from the MLB seasons of 2008 until 2019, given that these were the completed seasons prior to Covid-19. After all pitcher stats were web-scraped and stored into their corresponding variable, the observations were then binded together by their rows to create a dataframe called Pitchers. As a final step, all observations that contained a N/A value for any variable was then omitted and thus resulted in a size reduction for the dataframe.

3.2 Looking into Factor Variables

Given the binded dataframe, the compiled variables are presented through the str() function, where it consists of the following data types for simplicity: chr for the character class and num for the numeric class. For now, the numeric types can be disregarded as the discussion of importance is how to incorporate the character values into a regression model because applying a string of characters can be uninterpretable. In a similar fashion to numeric values, the character values must be interpreted numerically and that can be done in the sense of factor variables. Factor variables are established through obtaining a unique set of all possibilities for a given variable. For example, there can be 1000 observations for Gender being "Male" or "Female" but there are only two unique possibilities. The two unique set of possibilities in this case would be "Male" and "Female". In relation to the dataframe, the variable 'Name' can also be considered a factor variable as there are 500+ observations for pitchers, but names in general are usually distinctive and unique. Each name observation would most likely be a factor variable in the set of unique possibilities, but may not be of great significance. The motivation of factor variables is to apply it to a give predictor that can provide unique information to the research question at hand and thus use through contrast encoding or one-hot encoding. With that in mind, I have also made the assumption that the variables bbref_id, season, Name, and Team would have no significant impact in any model as well as becoming factor variables. The only one deemed most meaningful to consider as a factor variable would be the Level variable.

```
str(Pitchers)
```

```
## 'data.frame':    1439 obs. of  45 variables:
##  $ bbref_id: chr  "452298" "112020" "430101" "466918" ...
##  $ season  : chr  "2008" "2008" "2008" "2008" ...
##  $ Name    : chr  "Jarrod Washburn" "Joel Pineiro" "Ryan Rowland-Smith" "Miguel Batista" ...
##  $ Age     : num  33 29 25 37 24 28 35 26 25 30 ...
##  $ Level   : chr  "MLB-AL" "MLB-NL" "MLB-AL" "MLB-AL" ...
##  $ Team    : chr  "Seattle" "St. Louis" "Seattle" "Seattle" ...
##  $ G       : num  28 26 47 44 47 72 54 49 64 71 ...
##  $ GS      : num  26 25 12 20 9 4 5 4 0 0 ...
##  $ W       : num  5 7 5 4 4 3 4 4 6 5 ...
##  $ L       : num  14 7 3 14 7 7 4 4 1 4 ...
##  $ SV      : num  1 1 2 1 1 3 2 1 3 1 ...
##  $ IP      : num  153 148 118 115 108 ...
##  $ H       : num  174 180 114 135 112 86 97 98 62 81 ...
##  $ R       : num  87 89 49 89 53 44 43 55 29 33 ...
##  $ ER      : num  80 85 45 80 49 40 36 48 22 28 ...
##  $ uBB     : num  48 35 48 73 29 41 29 27 38 28 ...
##  $ BB      : num  50 35 48 79 30 47 36 31 39 35 ...
##  $ SO      : num  87 81 77 73 93 68 79 45 92 63 ...
##  $ HR      : num  19 22 13 19 18 5 12 12 6 5 ...
##  $ HBP     : num  7 2 2 6 3 7 4 5 4 4 ...
##  $ ERA     : num  4.69 5.15 3.42 6.26 4.07 3.7 3.4 4.84 2.22 2.87 ...
##  $ AB      : num  607 598 451 457 421 354 368 350 320 319 ...
##  $ X1B     : num  109 106 73 77 75 65 67 66 43 64 ...
##  $ X2B     : num  42 47 27 37 17 15 17 15 13 11 ...
##  $ X3B     : num  4 5 1 2 2 1 1 5 0 1 ...
##  $ IBB     : num  2 0 0 6 1 6 7 4 1 7 ...
##  $ GDP     : num  15 14 11 7 6 13 11 15 7 15 ...
##  $ SF      : num  5 3 3 11 1 5 0 1 1 2 ...
##  $ SB      : num  5 5 4 19 5 7 5 1 10 3 ...
##  $ CS      : num  2 3 2 2 2 2 1 2 3 4 ...
##  $ PO      : num  3 2 0 0 0 0 0 1 1 0 ...
##  $ BF      : num  675 645 506 556 464 419 412 388 370 365 ...
##  $ Pit     : num  2632 2227 1946 2104 1788 ...
##  $ Str     : num  0.61 0.66 0.63 0.57 0.63 0.6 0.64 0.61 0.61 0.6 ...
##  $ StL     : num  0.17 0.17 0.18 0.16 0.17 0.19 0.19 0.15 0.22 0.17 ...
##  $ StS     : num  0.06 0.07 0.09 0.07 0.11 0.08 0.1 0.07 0.1 0.08 ...
##  $ GB.FB   : num  0.36 0.48 0.39 0.46 0.47 0.63 0.51 0.59 0.54 0.46 ...
##  $ LD      : num  0.23 0.23 0.22 0.24 0.2 0.13 0.21 0.15 0.19 0.18 ...
##  $ PU      : num  0.1 0.03 0.1 0.06 0.05 0.04 0.06 0.04 0.07 0.11 ...
##  $ WHIP    : num  1.46 1.45 1.37 1.86 1.31 ...
##  $ BAbip   : num  0.306 0.317 0.277 0.309 0.302 0.283 0.307 0.293 0.251 0.3 ...
##  $ SO9     : num  5.1 4.9 5.9 5.7 7.7 6.3 7.5 4.5 9.3 6.5 ...
##  $ SO.W    : num  1.74 2.31 1.6 0.92 3.1 1.45 2.19 1.45 2.36 1.8 ...
##  $ SO_perc : num  0.129 0.126 0.152 0.131 0.2 0.162 0.192 0.116 0.249 0.173 ...
##  $ uBB_perc: num  0.071 0.054 0.095 0.131 0.062 0.098 0.07 0.07 0.103 0.077 ...
##  - attr(*, "na.action")= 'omit' Named int [1:7068] 1 2 3 4 5 6 7 8 9 10 ...
##   ..- attr(*, "names")= chr [1:7068] "1" "2" "3" "4" ...
```

The variable Level, referring to League, is now the only factor variable in the dataframe. The 4 levels are now denoted as the numbers 1 through 4 representing “MLB-AL”,“MLB-AL,MLB-NL”, “MLB-NL”, and “MLB-NL,MLB-AL”, respectively.

```
Pitchers$Level = factor(Pitchers$Level)

for (i in c(1:45))
    if (class(Pitchers[,i]) == "factor")
        print(levels(Pitchers[,i]))
```

```
## [1] "MLB-AL"        "MLB-AL,MLB-NL" "MLB-NL"        "MLB-NL,MLB-AL"
```

```
str(Pitchers)
```

```
## 'data.frame':    1439 obs. of  45 variables:
##  $ bbref_id: chr  "452298" "112020" "430101" "466918" ...
##  $ season  : chr  "2008" "2008" "2008" "2008" ...
##  $ Name    : chr  "Jarrod Washburn" "Joel Pineiro" "Ryan Rowland-Smith" "Miguel Batista" ...
##  $ Age     : num  33 29 25 37 24 28 35 26 25 30 ...
##  $ Level   : Factor w/ 4 levels "MLB-AL","MLB-AL,MLB-NL",..: 1 3 1 1 3 3 3 1 1 3 ...
##  $ Team    : chr  "Seattle" "St. Louis" "Seattle" "Seattle" ...
##  $ G       : num  28 26 47 44 47 72 54 49 64 71 ...
##  $ GS      : num  26 25 12 20 9 4 5 4 0 0 ...
##  $ W       : num  5 7 5 4 4 3 4 4 6 5 ...
##  $ L       : num  14 7 3 14 7 7 4 4 1 4 ...
##  $ SV      : num  1 1 2 1 1 3 2 1 3 1 ...
##  $ IP      : num  153 148 118 115 108 ...
##  $ H       : num  174 180 114 135 112 86 97 98 62 81 ...
##  $ R       : num  87 89 49 89 53 44 43 55 29 33 ...
##  $ ER      : num  80 85 45 80 49 40 36 48 22 28 ...
##  $ uBB     : num  48 35 48 73 29 41 29 27 38 28 ...
##  $ BB      : num  50 35 48 79 30 47 36 31 39 35 ...
##  $ SO      : num  87 81 77 73 93 68 79 45 92 63 ...
##  $ HR      : num  19 22 13 19 18 5 12 12 6 5 ...
##  $ HBP     : num  7 2 2 6 3 7 4 5 4 4 ...
##  $ ERA     : num  4.69 5.15 3.42 6.26 4.07 3.7 3.4 4.84 2.22 2.87 ...
##  $ AB      : num  607 598 451 457 421 354 368 350 320 319 ...
##  $ X1B     : num  109 106 73 77 75 65 67 66 43 64 ...
##  $ X2B     : num  42 47 27 37 17 15 17 15 13 11 ...
##  $ X3B     : num  4 5 1 2 2 1 1 5 0 1 ...
##  $ IBB     : num  2 0 0 6 1 6 7 4 1 7 ...
##  $ GDP     : num  15 14 11 7 6 13 11 15 7 15 ...
##  $ SF      : num  5 3 3 11 1 5 0 1 1 2 ...
##  $ SB      : num  5 5 4 19 5 7 5 1 10 3 ...
##  $ CS      : num  2 3 2 2 2 2 1 2 3 4 ...
##  $ PO      : num  3 2 0 0 0 0 0 1 1 0 ...
##  $ BF      : num  675 645 506 556 464 419 412 388 370 365 ...
##  $ Pit     : num  2632 2227 1946 2104 1788 ...
##  $ Str     : num  0.61 0.66 0.63 0.57 0.63 0.6 0.64 0.61 0.61 0.6 ...
##  $ StL     : num  0.17 0.17 0.18 0.16 0.17 0.19 0.19 0.15 0.22 0.17 ...
##  $ StS     : num  0.06 0.07 0.09 0.07 0.11 0.08 0.1 0.07 0.1 0.08 ...
##  $ GB.FB   : num  0.36 0.48 0.39 0.46 0.47 0.63 0.51 0.59 0.54 0.46 ...
##  $ LD      : num  0.23 0.23 0.22 0.24 0.2 0.13 0.21 0.15 0.19 0.18 ...
##  $ PU      : num  0.1 0.03 0.1 0.06 0.05 0.04 0.06 0.04 0.07 0.11 ...
##  $ WHIP    : num  1.46 1.45 1.37 1.86 1.31 ...
##  $ BAbip   : num  0.306 0.317 0.277 0.309 0.302 0.283 0.307 0.293 0.251 0.3 ...
##  $ SO9     : num  5.1 4.9 5.9 5.7 7.7 6.3 7.5 4.5 9.3 6.5 ...
##  $ SO.W    : num  1.74 2.31 1.6 0.92 3.1 1.45 2.19 1.45 2.36 1.8 ...
##  $ SO_perc : num  0.129 0.126 0.152 0.131 0.2 0.162 0.192 0.116 0.249 0.173 ...
##  $ uBB_perc: num  0.071 0.054 0.095 0.131 0.062 0.098 0.07 0.07 0.103 0.077 ...
##  - attr(*, "na.action")= 'omit' Named int [1:7068] 1 2 3 4 5 6 7 8 9 10 ...
##   ..- attr(*, "names")= chr [1:7068] "1" "2" "3" "4" ...
```

3.3 Contrast Encoding Introduction

As previously mentioned, there is a motivation behind transforming a variable into a factor variable. It provides further insight of qualitative information for the model at hand in the sense of becoming quantitative, or numeric. A machine straightforwardly interprets data as numbers, so this process allows us to feed in character or string information to a machine as a disguised numeric value. Following the process of factor variables, the contrast encoding procedure is one of a few methods that allows us to achieve the previously mentioned task. In a short summary, contrast encoding takes a factor variable and devices a binary procedure (0-False and 1-True) of L-1 levels,where L is the total number of levels, in order to differentiate among all levels in a matrix fashion.

For example, with the Level variable, there are 4 Levels total (refer to Looking at Factor Variables section). The statement below provides a simple demonstration as to what the matrix would look like with 4 variables where, similarly, the numbers 1 to 4 can represent the different levels of the variable Level (i.e. 1 is MLB-AL).

NOTICE: The first level for the columns is omitted (4x3 matrix).

```
contr.treatment(4)
```

```
##   2 3 4
## 1 0 0 0
## 2 1 0 0
## 3 0 1 0
## 4 0 0 1
```

Now, for a given observation when the sequence of 0's and 1's in a row is '0, 1, 0' then we refer back to the Level order and now the machine can denote if it is 0 for the 2nd level, 1 for the 3rd level, and 0 for the 4th level then the machine can interpret it as a numeric value related to the respective categorical variable. Similarly, when all level values are 0's, then its respective categorical value will be the first level that is omitted suggesting that it is none of the other L-1 options.

NOTE: Contrast encoding is a better alternative in this case because if left as factor variables in an integer coding setup the machine will decide importance if the level indicated 1>2>3>4. The idea is to set a fair ground for all levels with the value of 1 to have the same impact and then let the model interpret which level is more significant, if any.

3.4 Constrast Encoding Application

Provided here is a coded demonstration for the contrast encoding function where z is the factor variable and basename is the name of the variable of interest (Bhat,Harish. Modern Applied Statistics, University of California Merced. Merced,CA. October 2020).

```
contrastencode <- function(z,basename){
    # z = factor variable
    # basename = names of the variable
    nl = length(levels(z))
    newmat = matrix(0,nrow=length(z),ncol=(nl-1))
    con = contrasts(z)
    for (j in c(1:ncol(con))){
        thislevel = colnames(con)[j]
        theserows = which(z == thislevel)
        newmat[theserows, j] = 1
    }
    colnames(newmat) = paste(basename,".is.",colnames(con),sep='')
    return(newmat)
}
```

```
set.seed(1)
response = Pitchers$ERA
predictors = Pitchers[ ,-c(1,3,4,6)]

contrast_vars = list(NULL)
contrast_vars[[1]] = contrastencode(predictors[,2],names(predictors)[2])
newPitchers = data.frame(ERA=response,predictors[])
newPitchers = cbind(ERA = response, season=newPitchers$season, contrast_vars[[1]], predictors[,-c(1,2)])
rownames(newPitchers) <- 1:nrow(newPitchers)
```

Prior to the contrast encoding procedure, two things are set up such as isolating the response variable(ERA) from the rest of the predictors to ensure no alterations are made and secondly, a list of NULL values is established so that the 4 different levels of the Level can be readily included. The contrast encoding function is then applied to the Level variable for which they are then stored into the first index of the list, 'contrast_vars'. Once the Level variable is readily encoded, a new dataframe is constructed to bind the response variables we isolated along with the 'contrast_vars' list, and the remaining predictors. Now, we have 3 newly added variables.

```
head(newPitchers)
```

```
##    ERA season Level.is.MLB-AL,MLB-NL Level.is.MLB-NL Level.is.MLB-NL,MLB-AL  G
## 1 4.69   2008                       0               0                       0 28
## 2 5.15   2008                       0               1                       0 26
## 3 3.42   2008                       0               0                       0 47
## 4 6.26   2008                       0               0                       0 44
## 5 4.07   2008                       0               1                       0 47
## 6 3.70   2008                       0               1                       0 72
##   GS W  L SV    IP   H  R ER uBB BB SO HR HBP  ERA  AB X1B X2B X3B IBB GDP SF
## 1 26 5 14  1 153.2 174 87 80  48 50 87 19   7 4.69 607 109  42   4   2  15  5
## 2 25 7  7  1 148.2 180 89 85  35 35 81 22   2 5.15 598 106  47   5   0  14  3
## 3 12 5  3  2 118.1 114 49 45  48 48 77 13   2 3.42 451  73  27   1   0  11  3
## 4 20 4 14  1 115.0 135 89 80  73 79 73 19   6 6.26 457  77  37   2   6   7 11
## 5  9 4  7  1 108.1 112 53 49  29 30 93 18   3 4.07 421  75  17   2   1   6  1
## 6  4 3  7  3  97.1  86 44 40  41 47 68  5   7 3.70 354  65  15   1   6  13  5
##   SB CS PO  BF  Pit  Str  StL  StS GB.FB   LD   PU  WHIP BAbip SO9 SO.W SO_perc
## 1  5  2  3 675 2632 0.61 0.17 0.06  0.36 0.23 0.10 1.458 0.306 5.1 1.74   0.129
## 2  5  3  2 645 2227 0.66 0.17 0.07  0.48 0.23 0.03 1.446 0.317 4.9 2.31   0.126
## 3  4  2  0 506 1946 0.63 0.18 0.09  0.39 0.22 0.10 1.369 0.277 5.9 1.60   0.152
## 4 19  2  0 556 2104 0.57 0.16 0.07  0.46 0.24 0.06 1.861 0.309 5.7 0.92   0.131
## 5  5  2  0 464 1788 0.63 0.17 0.11  0.47 0.20 0.05 1.311 0.302 7.7 3.10   0.200
## 6  7  2  0 419 1607 0.60 0.19 0.08  0.63 0.13 0.04 1.366 0.283 6.3 1.45   0.162
##   uBB_perc
## 1    0.071
## 2    0.054
## 3    0.095
## 4    0.131
## 5    0.062
## 6    0.098
```

4 Exploratory Data Analysis

4.1 Numeric Summary of Variables

First and foremost, a summary of the dataframe is provided in reference to each individual variable. Some if not most of this information would be relevant in future explorations of the data. This is kept primarily for referencing at a later time.

```
summary(newPitchers)
```

```
##       ERA           season          Level.is.MLB-AL,MLB-NL Level.is.MLB-NL
##  Min.   : 0.54   Length:1439        Min.   :0.00000        Min.   :0.0000
##  1st Qu.: 2.67   Class :character   1st Qu.:0.00000        1st Qu.:0.0000
##  Median : 3.49   Mode  :character   Median :0.00000        Median :0.0000
##  Mean   : 3.61                      Mean   :0.04378        Mean   :0.4753
##  3rd Qu.: 4.33                      3rd Qu.:0.00000        3rd Qu.:1.0000
##  Max.   :13.50                      Max.   :1.00000        Max.   :1.0000
##  Level.is.MLB-NL,MLB-AL       G               GS              W
##  Min.   :0.00000        Min.   : 5.00   Min.   : 0.00   Min.   : 1.000
##  1st Qu.:0.00000        1st Qu.:46.50   1st Qu.: 0.00   1st Qu.: 2.000
##  Median :0.00000        Median :61.00   Median : 0.00   Median : 3.000
##  Mean   :0.02988        Mean   :56.13   Mean   : 1.54   Mean   : 3.639
##  3rd Qu.:0.00000        3rd Qu.:69.00   3rd Qu.: 0.00   3rd Qu.: 5.000
##  Max.   :1.00000        Max.   :86.00   Max.   :33.00   Max.   :16.000
##        L                SV               IP               H
##  Min.   : 1.000   Min.   : 1.000   Min.   :  8.10   Min.   :  6.00
##  1st Qu.: 2.000   1st Qu.: 1.000   1st Qu.: 51.20   1st Qu.: 41.00
##  Median : 3.000   Median : 2.000   Median : 62.10   Median : 52.00
##  Mean   : 3.659   Mean   : 9.308   Mean   : 62.75   Mean   : 55.72
##  3rd Qu.: 5.000   3rd Qu.:13.000   3rd Qu.: 71.00   3rd Qu.: 63.00
##  Max.   :15.000   Max.   :62.000   Max.   :204.20   Max.   :213.00
##        R                ER             uBB              BB
##  Min.   :  2.00   Min.   : 2.00   Min.   : 0.00   Min.   : 2.00
##  1st Qu.: 19.00   1st Qu.:17.00   1st Qu.:14.00   1st Qu.:16.00
##  Median : 25.00   Median :22.00   Median :19.00   Median :22.00
##  Mean   : 26.99   Mean   :24.79   Mean   :20.56   Mean   :22.91
##  3rd Qu.: 31.00   3rd Qu.:29.00   3rd Qu.:25.00   3rd Qu.:28.00
##  Max.   :109.00   Max.   :99.00   Max.   :78.00   Max.   :86.00
##        SO               HR              HBP              ERA
##  Min.   :  5.00   Min.   : 0.000   Min.   : 0.000   Min.   : 0.54
##  1st Qu.: 45.00   1st Qu.: 4.000   1st Qu.: 1.000   1st Qu.: 2.67
##  Median : 59.00   Median : 6.000   Median : 2.000   Median : 3.49
##  Mean   : 61.34   Mean   : 6.446   Mean   : 2.334   Mean   : 3.61
##  3rd Qu.: 75.00   3rd Qu.: 8.000   3rd Qu.: 3.000   3rd Qu.: 4.33
##  Max.   :221.00   Max.   :32.000   Max.   :15.000   Max.   :13.50
##        AB             X1B             X2B             X3B
##  Min.   : 30.0   Min.   :  2.0   Min.   : 0.00   Min.   : 0.00
##  1st Qu.:193.0   1st Qu.: 27.0   1st Qu.: 7.00   1st Qu.: 0.00
##  Median :232.0   Median : 35.0   Median :10.00   Median : 1.00
##  Mean   :235.9   Mean   : 37.6   Mean   :10.55   Mean   : 1.12
##  3rd Qu.:264.0   3rd Qu.: 44.0   3rd Qu.:13.00   3rd Qu.: 2.00
##  Max.   :777.0   Max.   :148.0   Max.   :47.00   Max.   :11.00
##       IBB              GDP              SF               SB
##  Min.   : 0.000   Min.   : 0.000   Min.   : 0.000   Min.   : 0.000
##  1st Qu.: 1.000   1st Qu.: 3.000   1st Qu.: 1.000   1st Qu.: 2.000
##  Median : 2.000   Median : 4.000   Median : 1.000   Median : 3.000
##  Mean   : 2.351   Mean   : 5.055   Mean   : 1.702   Mean   : 4.245
##  3rd Qu.: 3.000   3rd Qu.: 7.000   3rd Qu.: 2.500   3rd Qu.: 6.000
##  Max.   :12.000   Max.   :26.000   Max.   :11.000   Max.   :26.000
##        CS              PO               BF              Pit
##  Min.   :0.000   Min.   :0.0000   Min.   : 32.0   Min.   : 126
##  1st Qu.:0.000   1st Qu.:0.0000   1st Qu.:218.0   1st Qu.: 849
##  Median :1.000   Median :0.0000   Median :259.0   Median :1017
##  Mean   :1.092   Mean   :0.2752   Mean   :264.6   Mean   :1036
##  3rd Qu.:2.000   3rd Qu.:0.0000   3rd Qu.:295.0   3rd Qu.:1170
##  Max.   :7.000   Max.   :7.0000   Max.   :870.0   Max.   :3401
##       Str              StL              StS             GB.FB
##  Min.   :0.5200   Min.   :0.1000   Min.   :0.0300   Min.   :0.1500
##  1st Qu.:0.6200   1st Qu.:0.1500   1st Qu.:0.0900   1st Qu.:0.3800
```

```
##  Median :0.6400   Median :0.1700   Median :0.1100   Median :0.4400
##  Mean   :0.6373   Mean   :0.1697   Mean   :0.1115   Mean   :0.4471
##  3rd Qu.:0.6600   3rd Qu.:0.1800   3rd Qu.:0.1300   3rd Qu.:0.5000
##  Max.   :0.7400   Max.   :0.2400   Max.   :0.2300   Max.   :0.8100
##        LD               PU               WHIP           BAbip
##  Min.   :0.0800   Min.   :0.00000   Min.   :0.565   Min.   :0.1570
##  1st Qu.:0.1900   1st Qu.:0.05000   1st Qu.:1.103   1st Qu.:0.2640
##  Median :0.2200   Median :0.08000   Median :1.251   Median :0.2900
##  Mean   :0.2275   Mean   :0.07813   Mean   :1.260   Mean   :0.2895
##  3rd Qu.:0.2600   3rd Qu.:0.10000   3rd Qu.:1.403   3rd Qu.:0.3160
##  Max.   :0.4200   Max.   :0.22000   Max.   :2.400   Max.   :0.4760
##       SO9             SO.W           SO_perc          uBB_perc
##  Min.   : 2.30   Min.   : 0.690   Min.   :0.0590   Min.   :0.00000
##  1st Qu.: 7.30   1st Qu.: 2.030   1st Qu.:0.1890   1st Qu.:0.05900
##  Median : 8.70   Median : 2.680   Median :0.2300   Median :0.07600
##  Mean   : 8.85   Mean   : 3.039   Mean   :0.2349   Mean   :0.07827
##  3rd Qu.:10.30   3rd Qu.: 3.600   3rd Qu.:0.2740   3rd Qu.:0.09500
##  Max.   :16.70   Max.   :14.000   Max.   :0.5020   Max.   :0.20300
```

4.2 Plotting Response Variable

Due to ERA being the response variable, it is best to explore the variable alone and may be best visualized.

```
ERA_years = c('2008','2009', '2010', '2011', '2012', '2013', '2014', '2015', '2016', '2017', '2018', '2019')
mean_ERA = list(NULL)

for (i in 1:length(ERA_years))
  mean_ERA[i] = mean(newPitchers$ERA[newPitchers$season==ERA_years[[i]]])

plot(newPitchers$season, newPitchers$ERA,
     xlab="Season",
     ylab="ERA",
     main="Plots of ERA from 2008-2019",
     sub="Figure 1: Plotting each pitcher's ERA for each season ranging from 2008 to 2019")
```

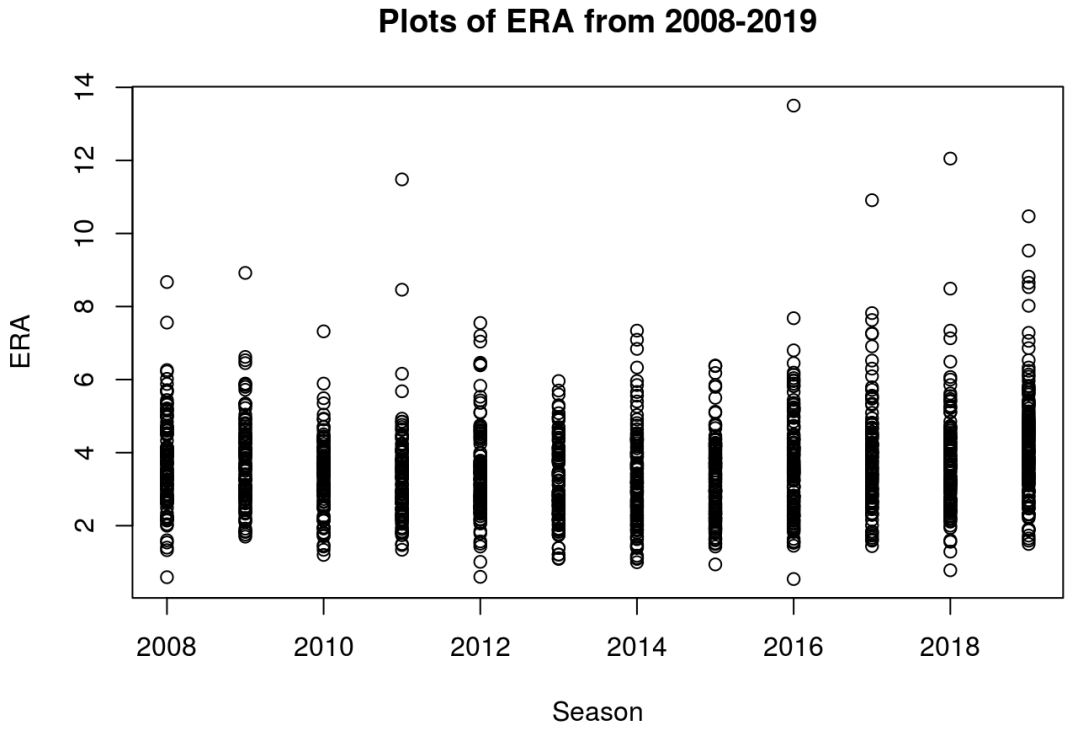


Figure 1: Plotting each pitcher's ERA for each season ranging from 2008 to 2019

For each season, there is a range for which the ERA values scale. It is noticeable that most values condense around a certain area, more specifically the mean. Over the course of 11 years, the mean value for ERA is given as 3.61 (obtained from the summary information). According to Baseball Wiki [2], an ERA score between 3.00 and 4.00 is above average. Given that the mean value lies in this range, it is fair to say that the value sets the foundation in terms of understanding why many pitchers' ERA gravitates towards or away from it.

4.3 Correlation

Correlation shows the strength of relationship between two variables and is given quantitatively through a scale of -1.0 to 1.0. For a variable to be fully correlated with another variable, their correlation will be 1 or -1 (depending on if it is a positive or negative correlation). In other words, if the first variable were to increase by a unit as well as the second variable they have a 1-to-1 relationship, or fully positive correlation. Similarly, if the first variable were to decrease by one unit but the other variable were to increase by 1 unit then it will be a 1-to-1 relationship as well, but it is considered a fully negative correlation. Lastly, when there is no determination of the trend for one variable with another, then it is said to have no correlation, or a value of 0.

NOTE: Correlation simply is a measure for association and not a way to state that X causes Y or vice-versa.

$$CorrelationFormula: C = \frac{\sum (X - \bar{X})(Y - \bar{Y})}{\sqrt{\sum (X - \bar{X})^2 \sqrt{\sum (Y - \bar{Y})^2}}}$$

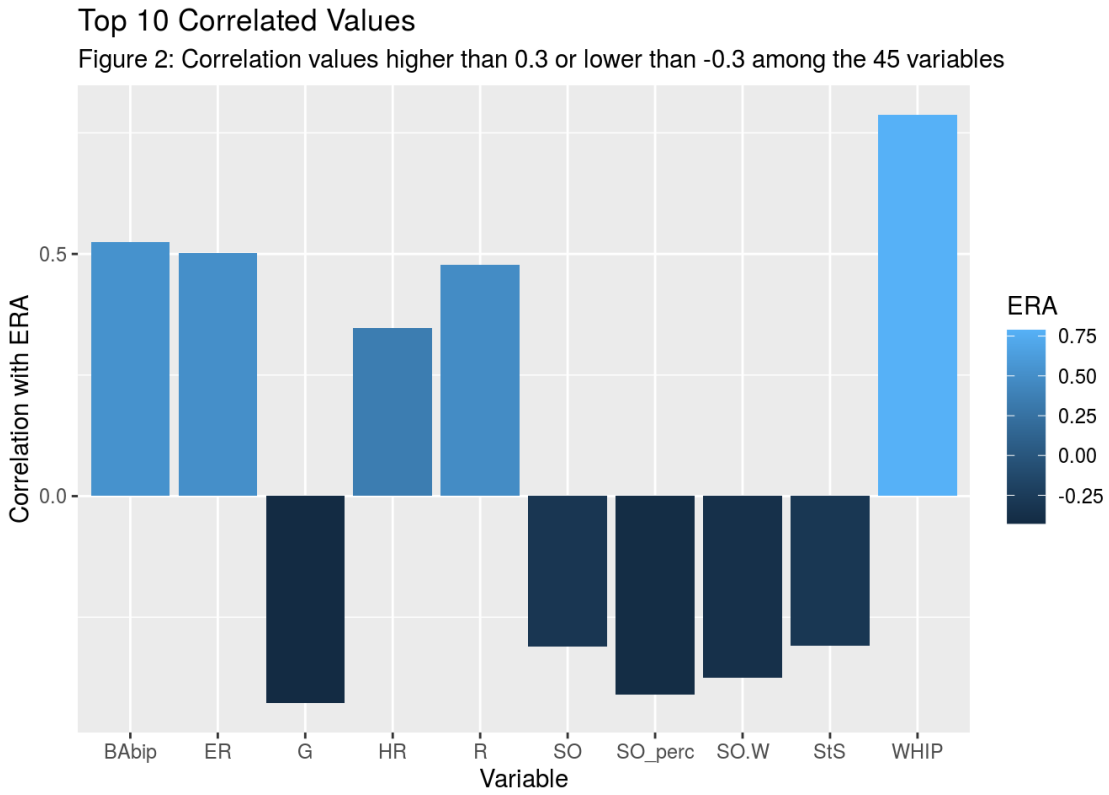
C is the correlation coefficient

$\bar{X}$ is the average value of the first variable

$\bar{Y}$ is the average value of the second variable

```
newPitchers[,-c(1,2)] %>% correlate()%>% focus(ERA)%>% filter(ERA>=0.3 | ERA<=-0.3) %>%
  ggplot(aes(x=rowname, y=ERA, fill=ERA)) +
  labs(title = "Top 10 Correlated Values", subtitle = "Figure 2: Correlation values higher than 0.3 or lower t
han -0.3 among the 45 variables ")+
  geom_bar(stat = "identity")+
  ylab("Correlation with ERA")+
  xlab("Variable")
```

```
##
## Correlation method: 'pearson'
## Missing treated using: 'pairwise.complete.obs'
```



A general assumption for correlation values is typically to be over the value of 0.7 for a notable relationship. Given the extensive amount of variables in the newPitchers dataset, the modified assumption was adjusted to consider values greater than or equal to 0.3 and lower than or equal to -0.3. Thus, it resulted in a size reduction with a total of 10 variables whose values are notably identified to be on the stronger side of positively or negatively correlated.

The variables that resulted in a positive correlation with ERA are the following: BAbip, ER, HR, R, and WHIP. On the contrary, the variables that resulted in a negative correlation with ERA are the following: G, SO, SO_perc, SO.W, and Sts.

Surprisingly, the split between the 10 variables are distributed evenly in term of of positive and negative correlation. Nonetheless, the variables that resulted in a negative correlation will be of greater interest due to a relationship of the ERA decreasing. Recall, a lower ERA score for a pitcher is most desirable, so these variables are an indicative measure to determine how a pitcher can decrease their ERA score. If a pitcher were to strikeout the batter more frequently in connection to more games they are allowed to pitch, the pitcher will have higher praise. Consequently, if a pitcher gives more HomeRuns/Runs or Hits, then the pitcher will be have lower praise as they are giving the opponent team more chances to score thus increasing their ERA score.

5 Model Preparation

5.1 Train-Test Split

An important aspect to assess a model is not only being able to provide reproducibility, but also providing a training and test split of the given data. This procedure is crucial in the sense of when there is limited data because then we would want to make use of what is provided such that a portion is allotted to training the model in a given learning environment then another portion can be given to assess the accuracy with never before seen data in a separate environment. In other words, the training data is directed to provide a supervised learning procedure to a given model such that it can interpret and map relationships numerically. Then, given a fitted model, we can provide a "set-aside" data whose purpose is to test or assess how well the trained model can deal with newly-introduced data.

To do the mentioned task, there are a number of different possibilities. The one given here ensures reproducibility with the "set.seed(1)" command. The procedure of choice does a 70% training and 30% testing split of the data. It gives a label '1' indicative for training or 2 indicative for testing. For a large dataset, the decision for sample replacement was made because if not replaced there will be a change in the probabilities too often for when a new observation is picked and will be too complex in the long run. Sampling with replacement also allows us to see which observations are more frequent.

In the end, our testing dataset consists of 1439 observations and our testing dataset consists of 431 observations.

```
set.seed(1)
names(newPitchers) <- make.names(names(newPitchers))
split = sample(2, nrow(newPitchers), replace=TRUE, prob=c(0.70,0.30))
train= newPitchers[split==1, ] #choose ERA scores whose labels are 1
rownames(train) <- 1:nrow(train)
test=newPitchers[split==2, ] # choose ERA scores whose labels are 2
rownames(test) <- 1:nrow(test)
table(split) #70% of 1439 = 1008 and 30% of 1439 = 431
```

```
## split
##    1    2
## 1008  431
```

Section 1: Regression

6 Principal Component Regression and Partial Least Squares Regression

6.1 Dimension Reduction Methods

Dimension Reduction Methods are a umbrella of methods that control variance in the data. However, the main aspect that sets them aside from other methods is that they do not use the original predictors. Instead, these methods transform the predictors and fit a least squares model.

Mathematically, the new set of transformed variables can be derived from the given formula:

$$Z_m = \sum_{j=1}^{p} \phi_{jm} X_j$$

Z_m is the m-th transformed feature

ϕ_{jm} are some arbitrary constants that need to be decided wisely

$X_j$is the j-th original predictor

Given the formulaic representation, the linear regression model can then be fitted using the following approach of least squares:

$$y_i = \theta_0 + \sum_{m=1}^{M} \theta_m z_{im} + \epsilon_i$$

$\theta_0, \theta_1, \ldots, \theta_M$ are the regression coefficients

z_{im} is the transformed feature value of the ith observation and j-th variable

ϵ_i is the irreducible error

Notice, the dimensionality reduction aspect comes from the approach of reducing the computation to estimate the original p+1 coefficients in the linear regression model by providing a simpler problem of estimating the M+1 transformed variables where M<p, otherwise there is no reduction made. In other words, these methods reduce the dimensionality of the data from p+1 predictors to M+1 features.

With the given data at hand, the two Dimension Reduction Methods that decipher the 'phi' constant are Principal Component Analysis/Regression and Partial Least Squares Regression.

6.2 Principal Component Analysis/Regression

Principal Component Analysis (PCA) is considered a dimension-reduction method since we can obtain a smaller dataset in comparison to the original while still preserving as much information as needed. When the dimension of the dataset is reduced, it is beneficial through the scope of simplicity, but it comes at the expense of accuracy. The direction of the first principal component corresponds to how the observations vary the most. In other words, it is a vector that is defined to be a line closest to the data as possible. In such task, it minimizes the sum of squares predictions perpendicular to the line. Similar to the first component vector, it is possible to have constructed up to p-distinct components, where the second component vector is a linear combination of the variables constructed from the first component vector.

Principal Component Regression is cost efficient as it proposes the ability to transform the columns of a dataset into a new set of features. This new set of features allows the model to account for the necessary amount of predictors needed to maximize the variance, but also minimize the error.

```
Pitcher_PCA = pcr(ERA~. - season, ncomp=40, data=train, scale=TRUE, validation="CV")
summary(Pitcher_PCA)
```

```
## Data:    X dimension: 1008 41
##  Y dimension: 1008 1
## Fit method: svdpc
## Number of components considered: 40
##
## VALIDATION: RMSEP
## Cross-validated using 10 random segments.
##        (Intercept)  1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
## CV           1.336      1.3     1.09     1.09   0.8275   0.7840   0.7842
## adjCV        1.336      1.3     1.09     1.09   0.8268   0.7833   0.7840
##        7 comps  8 comps  9 comps  10 comps  11 comps  12 comps  13 comps
## CV      0.7822   0.7516   0.7484    0.7446    0.7411    0.7427    0.7318
## adjCV   0.7821   0.7506   0.7479    0.7442    0.7413    0.7442    0.7245
##        14 comps  15 comps  16 comps  17 comps  18 comps  19 comps  20 comps
## CV       0.7207    0.7133    0.7137    0.7118    0.7057    0.6744    0.6674
## adjCV    0.7199    0.7119    0.7136    0.7117    0.7169    0.6727    0.6704
##        21 comps  22 comps  23 comps  24 comps  25 comps  26 comps  27 comps
## CV       0.6556    0.6134    0.6104    0.6021    0.5989    0.5973    0.5619
## adjCV    0.6550    0.6124    0.6096    0.6012    0.5981    0.5969    0.5613
##        28 comps  29 comps  30 comps  31 comps  32 comps  33 comps  34 comps
## CV       0.5623    0.4922    0.4920    0.4906    0.3967    0.3678    0.3409
## adjCV    0.5622    0.4912    0.4911    0.4895    0.3955    0.3665    0.3392
##        35 comps  36 comps  37 comps  38 comps  39 comps  40 comps
## CV       0.3119    0.2994    0.3003    0.2952    0.2954    0.2955
## adjCV    0.3105    0.2982    0.2990    0.2939    0.2941    0.2941
##
## TRAINING: % variance explained
##      1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps  8 comps
## X     30.832    43.05    50.71    56.76    61.09    64.51    67.75    70.76
## ERA    5.415    33.56    33.85    62.19    66.01    66.05    66.33    69.05
##      9 comps  10 comps  11 comps  12 comps  13 comps  14 comps  15 comps
## X      73.58     76.08     78.28     80.39     82.41     84.40     86.12
## ERA    69.28     69.63     70.03     70.12     71.73     71.77     72.57
##      16 comps  17 comps  18 comps  19 comps  20 comps  21 comps  22 comps
## X       87.81     89.35     90.80     92.22     93.51     94.79     95.85
## ERA     72.60     72.91     72.95     75.72     75.93     76.97     80.00
##      23 comps  24 comps  25 comps  26 comps  27 comps  28 comps  29 comps
## X       96.66     97.36     97.97     98.42     98.83     99.16     99.45
## ERA     80.22     80.81     80.99     81.18     83.32     83.33     87.44
##      30 comps  31 comps  32 comps  33 comps  34 comps  35 comps  36 comps
## X       99.64     99.78     99.88     99.93     99.95     99.98     99.99
## ERA     87.48     87.80     92.21     93.32     94.42     95.38     95.71
##      37 comps  38 comps  39 comps  40 comps
## X      100.00    100.00    100.00    100.00
## ERA     95.73     95.89     95.89     95.89
```

The provided data for the learning procedure of the model was scaled so that each contributes to the analysis with equal effect. The model is assumed to perform better with the standardizing procedure because in the latter it is quite sensitive to the variance based on the first component. For example, a variable ranging from 1-100 will have more affect then a variable that ranges from 0 to 1. Therefore, we adjust the scales and make each variable comparable to one another to address this problem. The mathematical interpretation of scaling is the following:

$$\text{Scale Value} = \frac{X - \bar{X}}{\sigma}$$

X is the observation's value

$\bar{X}$ is the mean value of all observations for that given variable X

σ is the standard deviation for that given variable X

With the made assumption, the model accounts for 40 out of the 44 predictors as given by the “ncomp” parameter. Referring to the Cross Validation Error portion of the summary, the trend for the values tends to decrease the more components there are. On the other hand, the Variance portion tends to increase in value the more components there are. Between the CV error and variance explained, it is best to find the sweet spot for the number of components that can perform well but also explains most of the variance with the fewest components as possible, otherwise it is not wise to use this procedure.

Note: The summary function of the model provides the Root Mean Squared Error (RMSEP). For the next portion, the plots will be using the Mean Squared Error (MSEP). To obtain the MSEP, we square the RMSEP value.

```
par(mfrow=c(1,2))
plot(Pitcher_PCA, "validation",val.type="MSEP", main="MSE Plot",xlab="# of Components", sub="Figure 3a: Measur
ing training error", col.main="navy", col.sub="navy")
plot(Pitcher_PCA, "validation", val.type="R2", main="R-Squared Plot", xlab="# of Components", sub="Figure 3b:
Measuring Variance", col.main="navy", col.sub="navy")
```

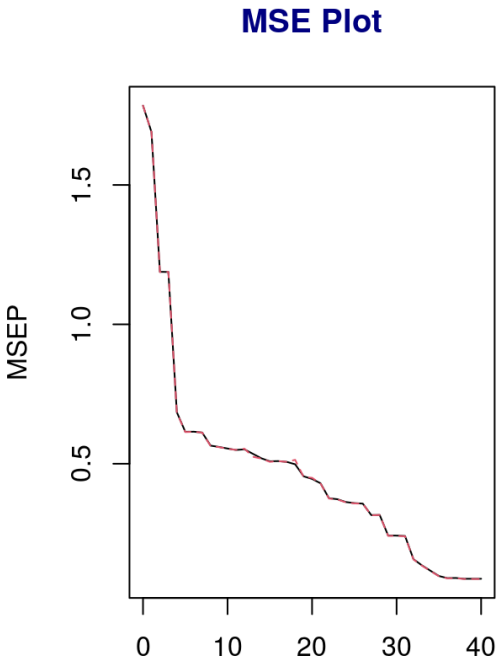


Figure 3a: Measuring training error

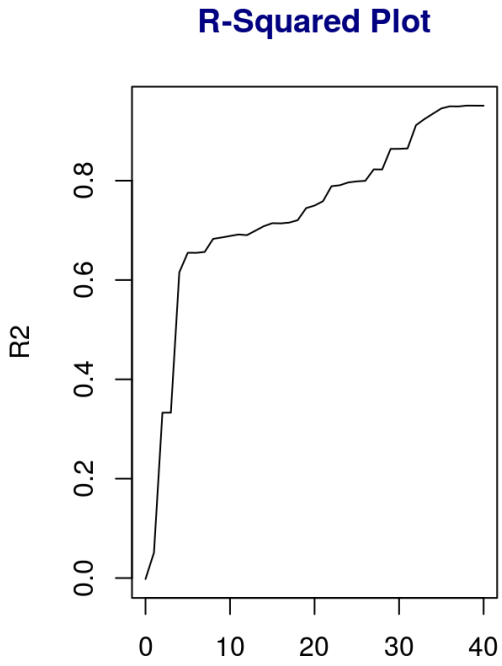


Figure 3b: Measuring Variance

```
par(mfrow=c(1,1))
```

From Figures 3a and 3b, the trend for cross validation error and variance explained based on the number of components are provided, respectively, otherwise indicative of a bias-variance trade-off. The more components we have the less of an error the model has because it is fitting to the training data more properly. In effect, the variance explained by the model is increasing as the number of components considered increases. From a graphical viewpoint, the best model that reduces the error and increases the r-squared metric uses a total of 40 components.

The use of 40 components does nothing in relation to the purpose of Principal Component Analysis and the reduction of dimensionality. Nonetheless, the use of 40 components is able to have a fairly linear trend in the actual vs predicted trend with the cost of over-fitting to the training data.

```
predplot(Pitcher_PCA, main="Predicted vs Actual ERA using 40 components", sub=c("Figure 4: Expresses PCA accur
acy with cross validated training data"), col.main="navy", col.sub="navy", )
abline(0,1, col="red")
```

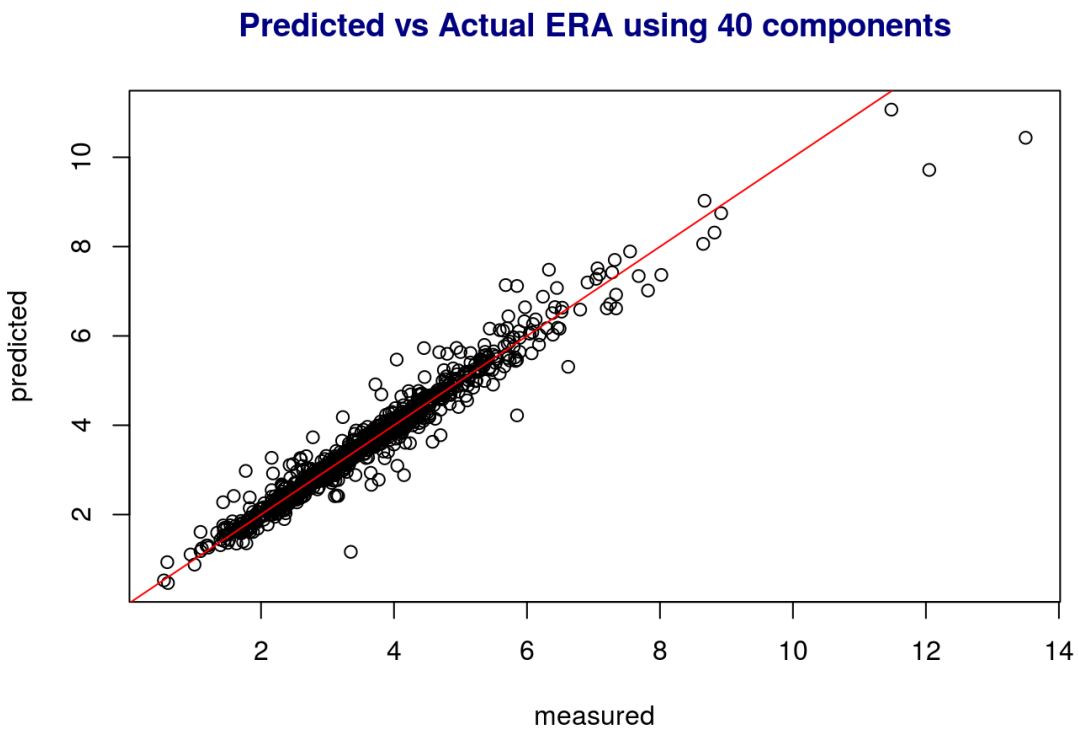


Figure 4: Expresses PCA accuracy with cross validated training data

It would be best to have a model that has a larger reduction in components in comparison to the original dataset and still has a decent level of effectiveness in explaining the variance and having a low error. The assumption of having a model where 90% of the variance is explained can be a better threshold alternative. In that case, referring back to the summary of the model data, the model with 18 components is the first number that surpasses the 90% by having 90.8% of the variance explained. The given table below explores the different possibilities of components ranging from 1 to 40 in attempt to showcase a more appropriate choice. However, these various number of component trials well be for the assessment of the trained model on the testing data to address any over-fitting/under-fitting.

```
Pitcher_PCApred = predict(Pitcher_PCA, test, ncomp=1)
PCA_err1 =  round(mean((Pitcher_PCApred-test$ERA)^2)*100,2)

Pitcher_PCApred2 = predict(Pitcher_PCA, test, ncomp=8)
PCA_err2 = round(mean((Pitcher_PCApred2-test$ERA)^2)*100,2)

Pitcher_PCApred3 = predict(Pitcher_PCA, test,  ncomp=18)
PCA_err3 = round(mean((Pitcher_PCApred3-test$ERA)^2)*100,2)

Pitcher_PCApred4 = predict(Pitcher_PCA, test,  ncomp=30)
PCA_err4 = round(mean((Pitcher_PCApred4-test$ERA)^2)*100,2)

Pitcher_PCApred5 = predict(Pitcher_PCA, test,  ncomp=40)
PCA_err5 = round(mean((Pitcher_PCApred5-test$ERA)^2)*100,2)
```

Table 1 MSE for PCA

Number of Components	MSE
1	200.33

8	61.24
18	54.38
30	31.32
40	9.93

Given the table of values, the 18 components does a decent job in size reduction, but it does not do the best with new data as more than 50% was accounted for in error. Another alternative may be presentable as well as 30 components is not on the lower spectrum of numbers but it is not quite at the higher end of the spectrum of numbers for the number of components used. In the case that 30 components are used, then we have a much smaller error in comparison to 18 components with only 31%. The best would be to use 40 components; however, that does the method no justice in fulfilling the dimension reduction of the dataset variables. For this method, the comparable argument that 18 or 30 components used would be the selling point for the method to be efficient.

6.3 Partial Least Squares Regression

Similar to PCA, Partial Least Squares Regression (PLSR) is a supervised alternative in the umbrella of dimension reduction methods. PLSR constructs a new set of features dependent of the original features then fits a linear model via least squares. The aspect of supervised learning is given through the fact that it incorporates the response variable in order to find the direction of the components that help explain both the response and predictors at the same time.

PLSR computes the first transformed feature by placing the strongest value of the variable that is informative to the response. It then identifies the next components by regressing them and taking the residuals in relation to the first feature. In a sense, the linear combinations of the new features allows the model to explain the independent and dependent variables provided. In the process of the ‘plsr’ function, scale is set to TRUE to scale the training data values so that they have equal impact on the response variable.

```
Pitcher_PLS =plsr(ERA~. - season, data=train, ncomp=40, scale=TRUE, validation="CV")
summary(Pitcher_PLS)
```

```
## Data:    X dimension: 1008 41
##  Y dimension: 1008 1
## Fit method: kernelpls
## Number of components considered: 40
##
## VALIDATION: RMSEP
## Cross-validated using 10 random segments.
##        (Intercept)  1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
## CV           1.336   0.9511   0.7508   0.6399   0.5522   0.5074   0.4751
## adjCV        1.336   0.9505   0.7504   0.6392   0.5508   0.5057   0.4760
##        7 comps  8 comps  9 comps  10 comps  11 comps  12 comps  13 comps
## CV      0.4414   0.4154   0.3981    0.3799    0.3646    0.3484    0.3409
## adjCV   0.4419   0.4139   0.3964    0.3773    0.3617    0.3462    0.3389
##        14 comps  15 comps  16 comps  17 comps  18 comps  19 comps  20 comps
## CV       0.3332    0.3296    0.3198    0.3130    0.3071    0.3019    0.3010
## adjCV    0.3315    0.3281    0.3187    0.3118    0.3063    0.3008    0.2998
##        21 comps  22 comps  23 comps  24 comps  25 comps  26 comps  27 comps
## CV       0.3009    0.3014    0.3011    0.3001    0.2997    0.2993    0.2995
## adjCV    0.2997    0.3002    0.2997    0.2988    0.2984    0.2981    0.2982
##        28 comps  29 comps  30 comps  31 comps  32 comps  33 comps  34 comps
## CV       0.2998    0.2997    0.2996    0.2996    0.2992    0.2991    0.2986
## adjCV    0.2985    0.2984    0.2983    0.2983    0.2980    0.2977    0.2973
##        35 comps  36 comps  37 comps  38 comps  39 comps  40 comps
## CV       0.2989    0.2979    0.2975    0.2966    0.2967    0.2968
## adjCV    0.2977    0.2966    0.2962    0.2952    0.2953    0.2953
##
## TRAINING: % variance explained
##      1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps  8 comps
## X      20.59    40.43    48.30    51.22     54.7    59.75    63.36    65.66
## ERA    50.67    69.24    77.96    84.12     86.8    88.39    90.18    91.60
##      9 comps  10 comps  11 comps  12 comps  13 comps  14 comps  15 comps
## X      67.39     68.82     70.70     72.44     74.23     75.84     77.92
## ERA    92.45     93.31     93.93     94.35     94.58     94.77     94.91
##      16 comps  17 comps  18 comps  19 comps  20 comps  21 comps  22 comps
## X       79.55     81.07     82.76     83.99     85.64     87.00     87.87
## ERA     95.11     95.31     95.46     95.61     95.64     95.66     95.68
##      23 comps  24 comps  25 comps  26 comps  27 comps  28 comps  29 comps
## X       88.60     89.99     91.19     92.13     92.67     94.26     95.01
## ERA     95.71     95.72     95.72     95.73     95.74     95.75     95.75
##      30 comps  31 comps  32 comps  33 comps  34 comps  35 comps  36 comps
## X       96.32     96.88     98.08     98.29     99.60     99.96     99.98
## ERA     95.75     95.75     95.76     95.77     95.77     95.78     95.84
##      37 comps  38 comps  39 comps  40 comps
## X       99.99    100.00    100.00    100.02
## ERA     95.87     95.89     95.89     95.89
```

The model accounts for 40 out of the 44 predictors as given by the “ncomp” parameter. Similar to PCA, the Cross Validation Error value decrease the more components there are. On the other hand, the Variance portion tends to increase in value the more components there are. Again, with the CV error and variance explained, it is best to find the sweet spot for the number of components that can perform well but also explains most of the variance but with the fewest components as possible, otherwise it is not wise to use this procedure.

Note: The summary function of the model provides the Root Mean Squared Error (RMSEP). For the next portion, the plots will be using the Mean Squared Error (MSEP). To obtain the MSEP, we square the RMSEP value.

```
par(mfrow=c(1,2))
plot(Pitcher_PLS, "validation",val.type="MSEP", main="MSE Plot",xlab="# of Components", sub="Figure 5a: Measur
ing training error", col.main="navy", col.sub="navy")
plot(Pitcher_PLS, "validation", val.type="R2", main="R-Squared Plot", xlab="# of Components", sub="Figure 5b:
Measuring Variance", col.main="navy", col.sub="navy")
```

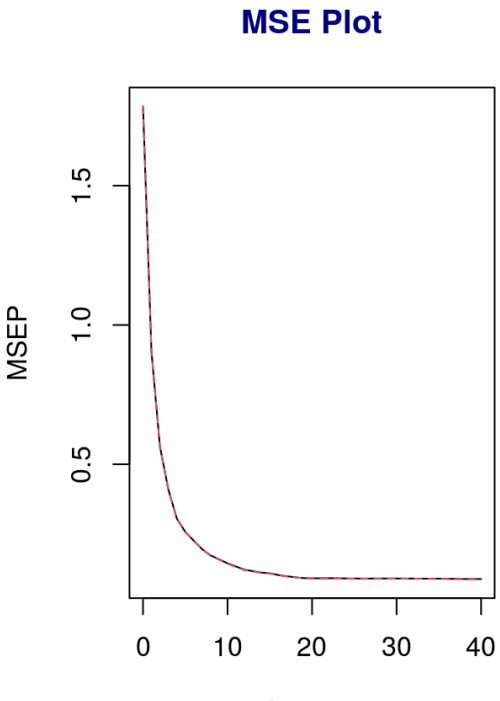


Figure 5a: Measuring training error

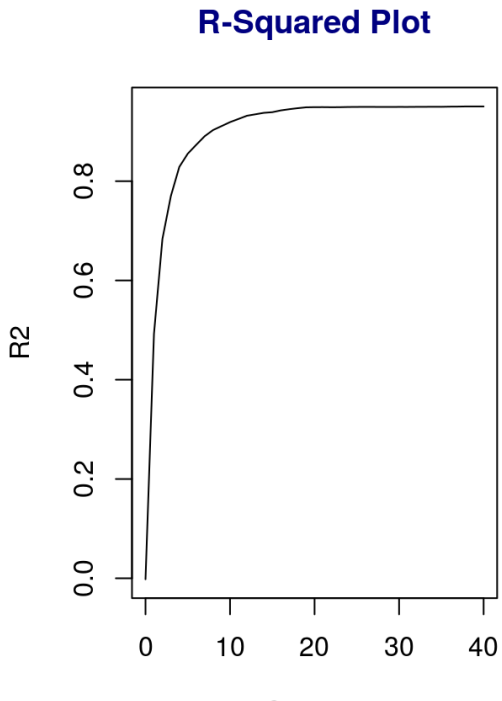


Figure 5b: Measuring Variance

```
par(mfrow=c(1,1))
```

Given from Figure 5a and Figure 5b, the two points are indicative for the following: the prediction error decreases as the number of components increases and the variance explained increases as the number of components increases. However, one notable feature of the plots is that there is no significant decrease after the use of 10 components – it begins to plateau.

```
predplot(Pitcher_PLS, main="Evaluation of Model on Validation Set", sub="Figure 6: Expresses PLSR accuracy wit
h cross validated training data", col.main="navy", col.sub="navy")
abline(0,1, col="red")
```

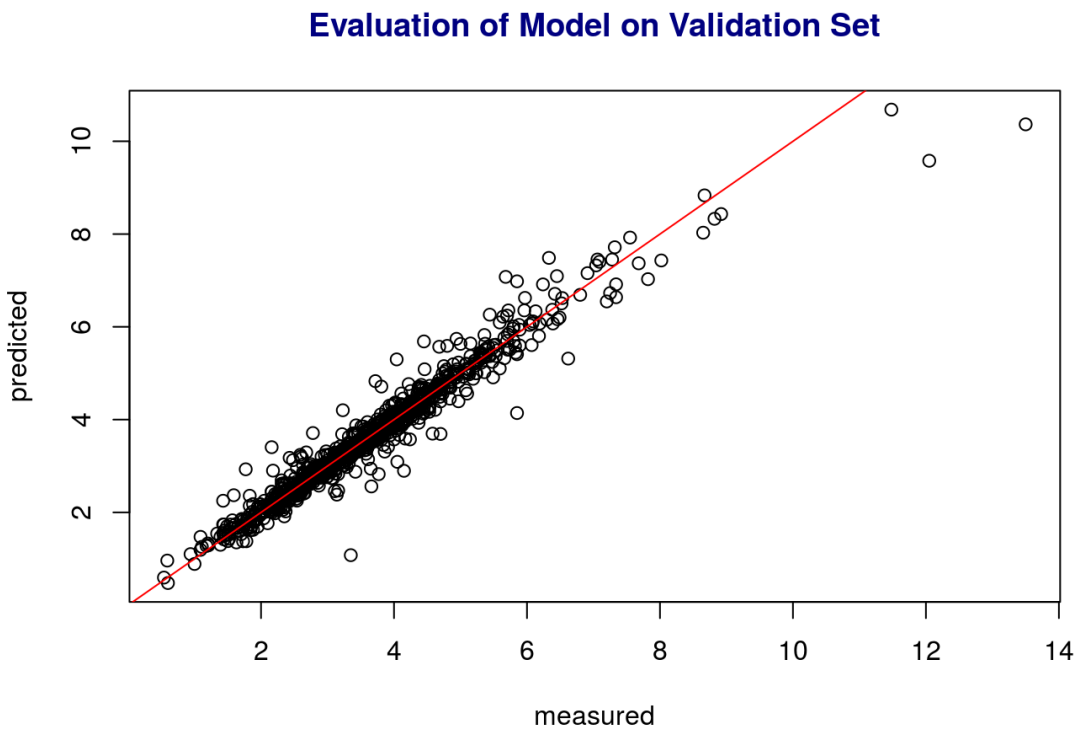


Figure 6: Expresses PLSR accuracy with cross validated training data

The scatter-plot of the fitted values is best analyzed when the points on the graph tend towards a linear trend– meaning greater accuracy. For example, when the measured valued is 2 then the predicted value would be exactly or very close to the value 2. Visually, there is a large cluster of data points centralized in the interval of measured values 2-6 that lie on or close to the linear line(given in red). However, it can also be noted that this model is not performing well with higher values as the data points stray further away from the linear line. Overall, in Figure 6, PLSR does well in having its predicted values close to the regressed linear trend, but there is worry that the model considered the 40 components is over-fitting the training data. Given from the MSEP(Figure 5a) and R2(Figure 5b) plots, there is a strong possibility the original number of 40 components can be reduced significantly while still achieving a high percentage of variance explained along with a low error value.

```
Pitcher_PLSpred = predict(Pitcher_PLS, test,scale=TRUE, ncomp=10)
PLS_err1 = round(mean((Pitcher_PLSpred-test$ERA)^2)*100,2)

Pitcher_PLSpred2 = predict(Pitcher_PLS, test, scale=TRUE, ncomp=20)
PLS_err2 = round(mean((Pitcher_PLSpred2-test$ERA)^2)*100,2)

Pitcher_PLSpred3 = predict(Pitcher_PLS, test, scale=TRUE, ncomp=30)
PLS_err3 = round(mean((Pitcher_PLSpred3-test$ERA)^2)*100,2)

Pitcher_PLSpred4 = predict(Pitcher_PLS, test, scale=TRUE, ncomp=40)
PLS_err4 = round(mean((Pitcher_PLSpred4-test$ERA)^2)*100,2)
```

Table 2 MSE for PLSR

Number of Components	MSE
10	16.66
20	10.60
30	10.20
40	9.94

From Table 2, an exploration of different alternatives for the number of components used in PLSR for the testing set is presented. Referencing figures 5a and 5b, the values in both plots indicates that 10 components are well-informative of the data and then begin to gradually decrease and plateau around 30 components. Although 40 components yields in the lowest MSE, the motivation behind Dimension Reduction Methods is thrown aside. Given the scope of the method, it is best to reduce the number of variables while still achieving better metrics of model analysis. In particular, the use of 10 variables only has an MSE of 16.66 while the use of 20 components has a MSE of 10.60. The use of 30 components is also considerable but having 10 less components would be a better trade-off than a 0.4 decrease in the MSE. Comparably, this method would perform best with the use of 10-20 components.

7 Neural Networks

Another comparable model and more modern approach in attempt to understand the underlying relationships between the predictors and response variable are Neural Networks. A Neural Network models after a human brain such that its architecture consists of many interconnected nodes produced by a multiple linear regression followed by a non-linear activation function. The simplest but yet important component of the neural networks is the “neuron” itself as it holds the mathematical procedure for the input-to-output computation. Generally, each “neuron” has its associated weight value and the model aims to adjust and control these values such that it minimizes the error.

7.1 Architecture of a Neural Network

- Input Layer
  - A block of nodes otherwise considered as the features used from your data
  - Each node of this layer represents one feature
  - No computer is done, just declaration
- Connection
  - A link between two nodes of different layers
  - Relevant to the weights that each node carries on
- Weights
  - Represents the numerical strength of a connection between two nodes in different layers
  - Each connection has its associated weight value
  - Alters the rate of increase for an activation function
- Biases
  - An attached values to the activation function
  - Allows the activation function to be shifted either left or right to better fit the data
  - Only affects output values and not input values
- Hidden Layer
  - A performative block of nodes that contains the activation function
  - Each node transforms the weights to the next given layer of nodes
  - Depth and Width of Hidden Layers are possible
    - There can be multiple hidden layers before arriving at the output layer (performative boost)
    - The amount of nodes in each hidden layer can vary (learning performance)
- Activation Function
  - Mathematical function that require an input node along with a weight and bias to construct a relative output value
  - Attached to each neuron in the layer
  - Normalizes the output value to be between 0 and 1
  - Hidden Layer Activation Function
- Output Layer
  - Corresponds to the number of predictions needed to be made
  - Activation function used to map to the output value
  - Contains an activation function
    - For a regression problem: A activation function is not needed since we want the output to map to any value

For a visual representation, please refer to this simple case of a neural network. *documentation (https://techvidvan.com/tutorials/wp-content/uploads/sites/2/2020/05/Architecture.jpg)

7.2 Mathematical Interpretation of a Nueral Network

Some Generic Notation:

- The vector(1xp) “X” represents the input nodes.
- The matrix(nxp) “W” represents the weight matrix. It holds the values of the weights per connection.
- The vector(1xn) “b” represents the biases applying to each node in the hidden layer
- The vector(nx1) “U” represents the the weights from the final hidden layer to the output layer
- The scalar(1x1) “a” represents a value similar to a bias in the scope of a regression problem where $y \epsilon R$
- The activation function ϕ in term of the inputs, weights, and biases

Note:‘p’ represents the number of predictors there are and ‘n’ represents the number of nodes per that respective layer

The activation function most proper for a regression problem is a sigmoid/ logistic function given as the following: $\phi(Z) = \frac{1}{1 + e^{-Z}}$

Forward Propagation

The first process of a neural network is the forward propagation of data and it represented mathematically in the following way:

Step 1: The first step is to establish the weight influence across all nodes. For each input node, a weight value is multiplied with it then the sum of all products will capture the combined effect.

$$\sum_{i=1}^{P} X_i W_{ij} = X_1 W_{1j} + X_2 W_{2j} + \ldots + X_P W_{Pj}$$

Step 2: Next, the biases are attached to sum of matrix-vector products. The sum is then stored in a arbitrary variable named “Z” that will be passed into the activation function.

$$Z = \vec{x}W + \vec{b}$$

Step 3: The contents of Z are then passed into a nonlinear activation function (typically sigmoid/logistic) for a single variable regression problem. The contribution the activation function has is crucial in expressing the non-linearity for the output of the neurons as they contribute to the learning rate of the model. If not applied, the neural network simply becomes as simple as a linear regression method.

$$h = \phi(Z) = \phi(\vec{x}W + \vec{b})$$

Step 4: Lastly, the forward propagation process reaches the output layer to provide a prediction.

$$\hat{y} = hU + a$$

Backward Propogation

The second process of of a neural network is the Backward Propagation procedure. This procedure is related to the learning procedure of the model and is represented mathematically by the following steps:

Step 1: Compute the Mean Squared Error (MSE) to assess how far off the model’s prediction is from the actual response value. Then, the error across the entire data set is calculated – otherwise known as the loss function denoted by the arbitrary variable ‘C’.

$$\text{For an individual instance: } MSE = (y_i - \hat{y}_i)^2$$

$$\text{Loss Function: } C = \frac{1}{P} \sum_{i=1}^{P} (y_i - \hat{y}_i)^2$$

Step 2: Next, the model uses this information to update the weights and biases such that it learns the relationship between the two. The relationship is established through how one changes with respect to the other and the model interprets that rate of change – or gradient – and wishes to optimize it. Ultimately, a gradient is performed on the loss function with respect to the weights and biases. For a given path to the prediction, the gradient requires a chain-rule partial differentiation procedure since the cost function is not directly related to the weights and biases. It first traverses from the cost function to the predictions then to the weights and biases in the Z expression formulated in the forward propagation section.

The gradient expression for the weights is given as :

$$\frac{\partial C}{\partial w_i} = \frac{\partial C}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial Z} \frac{\partial Z}{\partial w_i}$$

Breaking this down into components

$$\frac{\partial C}{\partial w_i} = \frac{2}{P} \sum_{i=1}^{P} y_i - \hat{y}_i$$

$$\frac{\partial \hat{y}}{\partial Z} = \phi(Z)(1 - \phi(Z))$$

$$\frac{\partial Z}{\partial w_i} = x_i$$

Therefore, the final expression in regards to the weight is:

$$\frac{\partial C}{\partial w_i} = (\frac{2}{P} \sum_{i=1}^{P} y_i - \hat{y}_i)(\phi(Z)(1 - \phi(Z)))(x_i)$$

The gradient expression for the biases is given as the same for the weights with the precondition that the bias value is to have a input of constant value 1:

$$\frac{\partial C}{\partial b} = (\frac{2}{P} \sum_{i=1}^{P} y_i - \hat{y}_i)(\phi(Z)(1 - \phi(Z)))$$

Note: The previously mentioned information and formulations were provided predominantly in the scope of a regression problem.

7.3 Neural Network Application

Similar to the scale parameter in the PCA section, the normalize function does a similar effect but with a procedure that is called “max-min normalization”. Essentially, the value is subtracted by the minimum value and then divided by the range. This ensures that all variables are on the same scale with transformed values between 0 and 1.

```
normalize <- function(x) {
  return ((x - min(x)) / (max(x) - min(x)))
}

set.seed(1)
maxmin_Pitcherstrain = as.data.frame(lapply(train[,-2], normalize))
maxmin_Pitcherstest = as.data.frame(lapply(test[,-2], normalize))

NN_testresponse = maxmin_Pitcherstest[,1]
NN_testpredictors = maxmin_Pitcherstest[,-1]
```

The design of a neural network is completely interpretive to the designer as the relationships among the variables are unique and it might take different means to interpret. There are a few parameters and hyper-parameters that can be designer choice such as the number of hidden layers in the neural network, the amount of neurons per hidden layer, and the activation function. When exploring the amount of layers

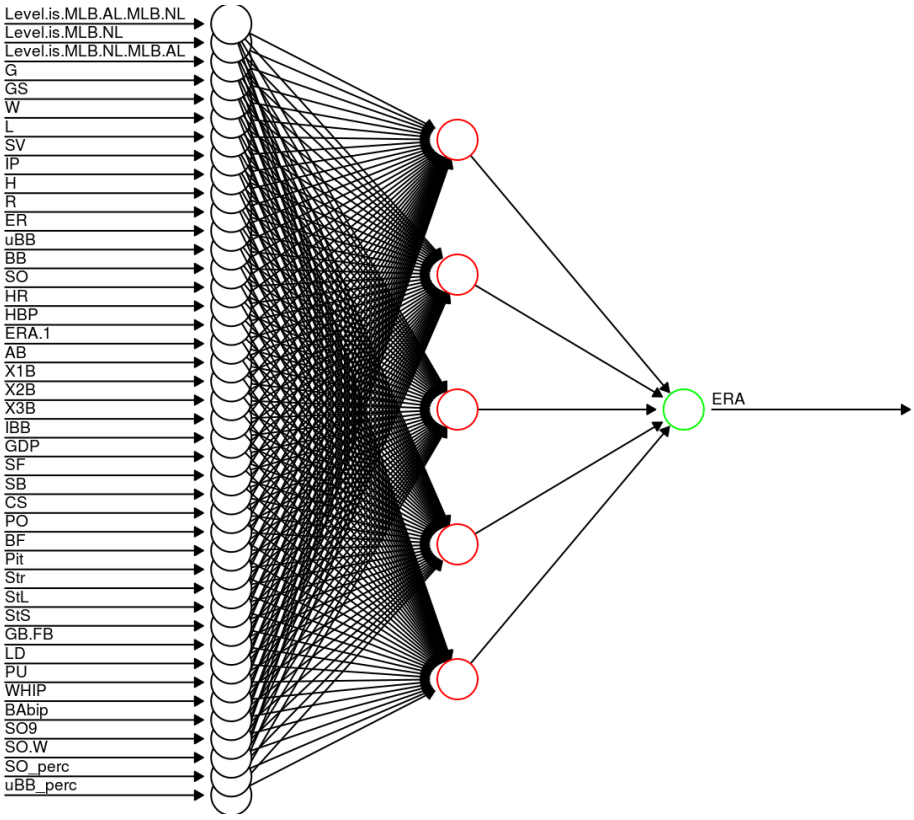
or the amount of neurons per layer, the question of Depth vs Breadth comes into play. A neural network should be fast but also generalizable and that is dependent on the layers and neurons per layer.

The nueralnet() function provides the build of the model with given parameters such as 'hidden' as a vector to determine how many neurons per layer (separated by a comma), 'rep' to denote the number of repetitions for the neural network training, 'act.fct' to note the type of activation function used, and 'linear.output' to imply an activation function on the output neurons. One thing to address, is that the neuralnet() function was kept to have the default algorithm of resilient propagation (rprop) as opposed to using the back-propagation method mathematically denoted earlier. The iterative algorithm is similar to back-propagation except for two changes that address the fallback of back-propagation. The first difference rprop addresses is not considering the magnitude of the gradient but rather the sign of it. From back-propagation, there can be the chance that the gradient misses the gradient value so in the next iteration the algorithm can cause the update to overshoot the weight value but in the opposing direction. Secondly, rprop does not use a single learning rate value for all weights an biases; it holds values unique to each weight and bias and adapts the differences during the training procedure.

```
Pitchers_NN = neuralnet(ERA~.,data=maxmin_Pitcherstrain, hidden=c(5), rep=3, act.fct = "logistic", linear.outp
ut = FALSE)
```

The plot below shows a simple structure of a neural network with the input layer on the left with 43 nodes(black), one hidden layer in the middle with 5 nodes (red), and lastly the output layer on the right with one node(green). The first and only weight matrix W in this model corresponds to the the connections from the input nodes to the nodes in the hidden layer and this will be of size 5-by-43. The matrix U needed for the prediction formula is derived from the connections between the final hidden layer and the output layer and this will be a vector because there is only one node in the outer layer. The size of this vector will be 5-by-1.

```
plot(Pitchers_NN, rep="best", show.weights=FALSE, cex=0.6, col.hidden="red", col.out="green", intercept=FALSE)
```



Given the background of neural networks and design questions, the following section will cover the performances of different Neural Networks with different amounts of hidden layer along with variations of neurons in each hidden layer.

```
# 3 Layers and small amount of neurons
set.seed(1)
Pitchers_NN1 = neuralnet(ERA~. ,data=maxmin_Pitcherstrain, hidden=c(5,7,10), rep=3, act.fct = "logistic", line
ar.output = FALSE) # Model 1
Pitchers_NN2 = neuralnet(ERA~. ,data=maxmin_Pitcherstrain, hidden=c(10,7,5), rep=3, act.fct = "logistic", line
ar.output = FALSE) # Model 2

# 3 Layers and larger amount of neurons
Pitchers_NN3 = neuralnet(ERA~. ,data=maxmin_Pitcherstrain, hidden=c(30,20,10), rep=3, act.fct = "logistic", li
near.output = FALSE) # Model 3
Pitchers_NN4 = neuralnet(ERA~. ,data=maxmin_Pitcherstrain, hidden=c(10,20,30), rep=3, act.fct = "logistic", li
near.output = FALSE) # Model 4

# 2 layers and small amount of neurons
Pitchers_NN5 = neuralnet(ERA~. ,data=maxmin_Pitcherstrain, hidden=c(3,7), rep=3, act.fct = "logistic", linear.
output = FALSE) # Model 5
Pitchers_NN6 = neuralnet(ERA~. ,data=maxmin_Pitcherstrain, hidden=c(7,3), rep=3, act.fct = "logistic", linear.
output = FALSE) # Model 6

# 2 layers and larger amount of neurons
Pitchers_NN7 = neuralnet(ERA~. ,data=maxmin_Pitcherstrain, hidden=c(25,17), rep=3, act.fct = "logistic", linea
r.output = FALSE) # Model 7
Pitchers_NN8 = neuralnet(ERA~. ,data=maxmin_Pitcherstrain, hidden=c(17,25), rep=3, act.fct = "logistic", linea
r.output = FALSE) # Model 8
```

Next, the predictions are made with the trained model and we save those accordingly.

```
NN_list = list(Pitchers_NN1, Pitchers_NN2, Pitchers_NN3, Pitchers_NN4, Pitchers_NN5, Pitchers_NN6, Pitchers_NN
7, Pitchers_NN8)
NN_pred = lapply(NN_list,compute, NN_testpredictors)
```

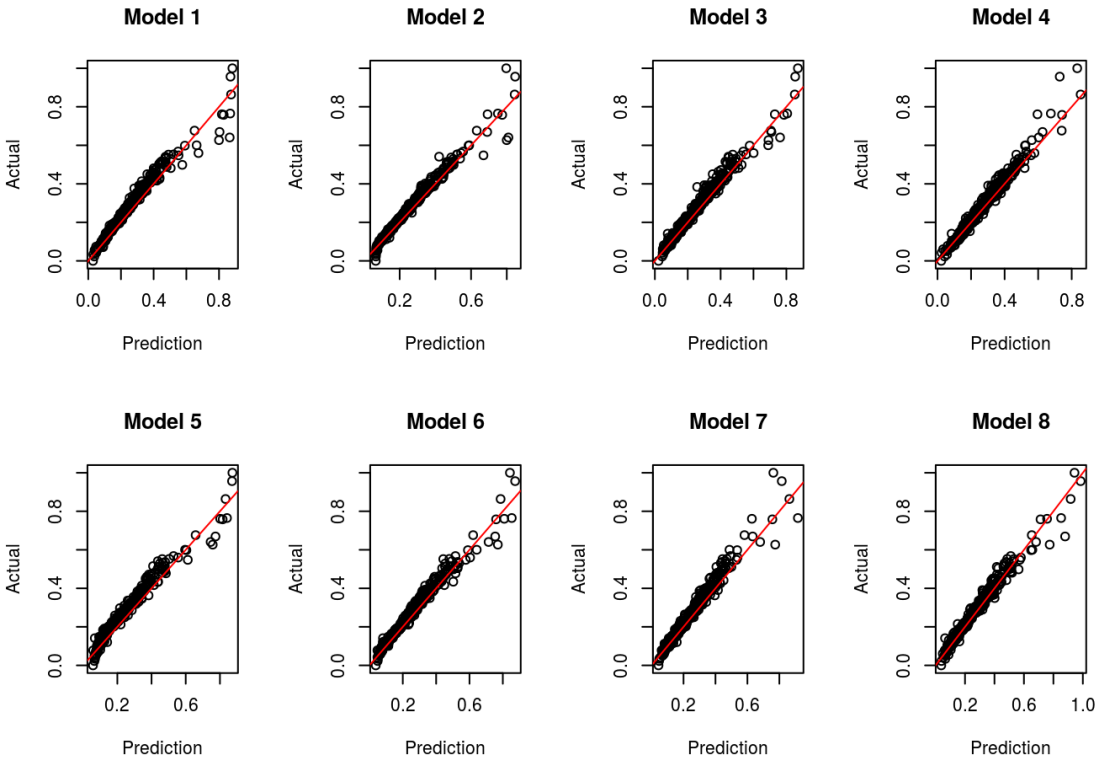
Graphically, the plots below illustrate how far off the predictions on the testing data are from the actual values. Comparing it to a linear line to denote a one-to-one accurate prediction, the model whose scatter-plot is more compact and follows the linear trend the most is the more efficient model. In the following list, the models will be evaluated based on the graphical downfall and strengths.

- Model 1
  - In the beginning, most scatterplot values lie above the linear line but are somewhat compact. (+/-)
  - When the values are higher, the model does not seem to do well. (-)
    - most of them tend below the linear line
- Model 2
  - In the beginning, more scatterplot values are very compact around the linear line (+)
  - Does not seem to capture the larger values well (+/-)
    - A few outliers at the end not captured fully and are a bit scattered below
- Model 3
  - In the beginning, scatterplot values are more central to and compact around the linear line (+)
  - Larger values are captured well but are a bit scattered away (+/-)
- Model 4
  - In the beginning, scatterplot values are more central around the linear line but are dispersed a bit more (+/-)
  - Larger values are not captured too well (+/-)
    - they tend to be above the linear line
- Model 5
  - In the beginning, the scatterplot values appear to be following a logarithmic trend and a good portions lies slight distant from the linear line(-)
  - The larger values are somewhat captured but they stray a bit far from the linear line (+/-)
- Model 6
  - In the beginning, the scatter plot values are compact around the linear line but some portion of it lies above the line (+/-)

    - Larger values are scattered more but they are equidistant to the line on both sides (+/-)
- Model 7
    - In the beginning, scatterplot values are local to the linear line, but lie more above the linear line (+/-)
    - Larger values are scattered around the linear line equidistantly (+/-)
        - Captured somewhat well as the linear line is central to those outliers
- Model 8
    - For the most part, the scatterplot values follow the same steepness of the linear trend and is compact (+)
    - Larger values are scattered but they are somewhat equidistant to the linear line (+/-)

Note: (+) is positive comment about the model and (-) is negative comment about the model

```
par(mfrow=c(2,4))
for (i in 1:8){
  plot(NN_pred[[i]]$net.result, NN_testresponse, xlab="Prediction", ylab="Actual", main = paste('Model', i, se
p=' '))
  abline(0,1, col="red")
}
```




```
par(mfrow=c(1,1))
```

Finally, the MSE's are computed accordingly per model to assess it's prediction accuracy. With the given table of errors below, the model that does the best is Model 8 with a error of 0.07%.

```
NN_testerr = list(NULL)
for (i in 1:8){
  NN_testerr[i] = round(mean((NN_pred[[i]]$net.result - NN_testresponse)^2)*100, 2)
}

col_names = c("Model 1","Model 2", "Model 3","Model 4","Model 5","Model 6","Model 7","Model 8")
rbind(Model = col_names, MSE = NN_testerr)
```

```
##       [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## Model "Model 1" "Model 2" "Model 3" "Model 4" "Model 5" "Model 6" "Model 7"
## MSE   0.15      0.09      0.09      0.08      0.18      0.1       0.13
##       [,8]
## Model "Model 8"
## MSE   0.07
```

Overall, this section explored the use of neural networks and its ability to interpret normalized data to learn and predict different cases. By exploring a few variations of neural networks with the design choices primarily in the number of hidden layers and the number of neurons in a given layer, evidently some design choices were more efficient than others. It is rather inconclusive to assess what general design choice was best as the provided models did not encompass all possibilities. However, to point something out from these cases the model that had the best performance was one that had two hidden layers and a rather large amount of neurons in each layer in an ascending fashion. The reasoning behind these numbers was to see if there was a fractional relationship between the number of input nodes and the hidden layer nodes. For example, would having 1/3 or 2/3 of nodes in the hidden layers be a good design choice? The next runner up model would be Model 3 that contained 3 hidden layers where the nodes were on the larger side and, again, in an increasing order. That being said, with the given data, it generally seems that so long as the number of layers are kept minimal and the number nodes are a fractional ratio to the number of predictors in an ascending order the more likely the model will perform well.

8 Random Forest Regression

8.1 Introduction

The Random Forest model is a technique that uses bagging alongside bootstrapping in attempt to lower the variance explained. In particular, bagging does well as its focus is to pick a sample of observations with replacement and creates “B-amount” of training data sets. This allows the bagging procedure to take the average of multiple, relatively-unbiased models. Random Forests are dependent on uncorrelated trees that run in parallel, so that the compilation of these trees outputs an average value for the predictions provided. Individually, these trees have high variance as they can become extensive in terms of depth, but computing the average of predictions reduces that variance.

For each tree:

- About two-thirds of the total data is given for training
- The selection of predictor variables at a given node is chosen at random
    - For regression, the ideal amount is one-third of the predictors
    - It ensures that a given model is not dependent on a given predictor

8.2 Mathematical Interpretation

As mentioned above, in the scope of regression, the bagging and bootstrapping technique is helpful in generating ‘B-amount’ of training data sets so that then the model is trained on the ‘b-th’ training set and we obtain the average of averaged predictions:

$$\hat{f}_{bag}(x) = \frac{1}{B}\sum_{b=1}^{B}\hat{f}^{*b}(x)$$

8.3 Random Forest Application

$$\boldsymbol{Model - 1}$$

The first variation of the Random Forest model attributes to the one-third of total predictors given for each split with the parameter ‘mtry’. Given 44 predictors, a third of that amount was a little under 15, so in an attempt to explore different possibilities an additional 5 predictors were added to create the upper bound of 20 predictors considered at each split. Then, by default, the randomForest function constructs a model with 500 trees if not specified otherwise. The exploration for this model was to see if a higher number of predictors at each split in connection to a larger Random Forest would explain more of the variance.

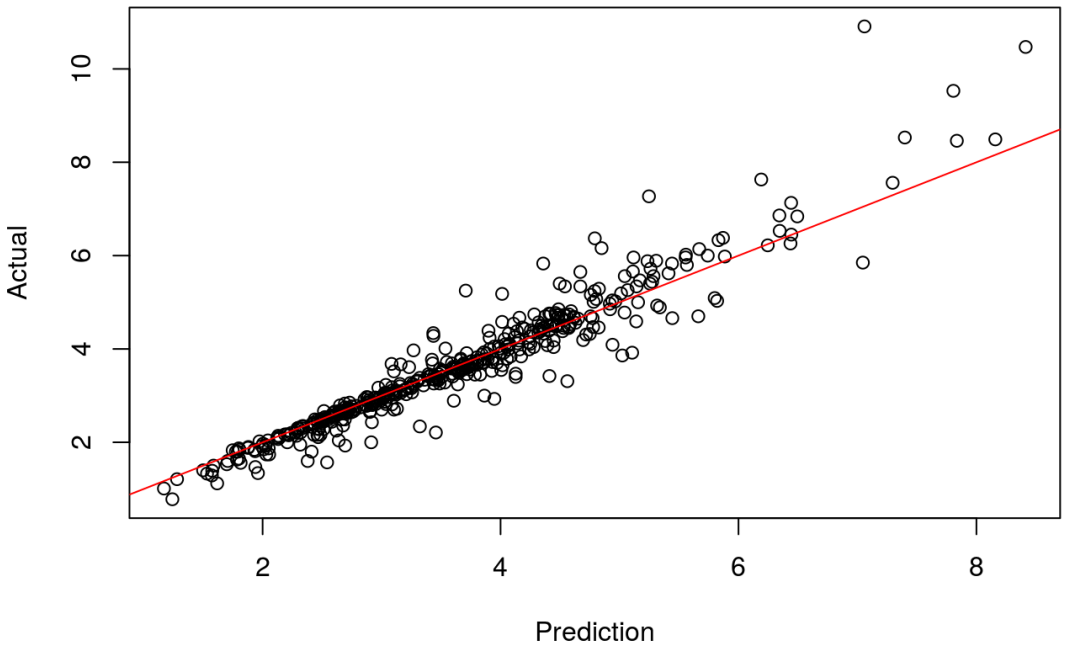
Given the model summary, it can be seen that a total of 89.37% of the variance was explained. Typically, a 90% of the variance explained can be deemed a good model despite the cutoff percentage. It would be safe to say that this model does seemingly well. The main concern of this model as denoted through Figure 8 is that the model doesn't seem to do well with large values as the outliers can attest for that.

```
set.seed(1)
Pitchers_RFM = randomForest(ERA~. - season, data=train, mtry=20, importance=TRUE)
Pitchers_RFM
```

```
##
## Call:
##  randomForest(formula = ERA ~ . - season, data = train, mtry = 20,      importance = TRUE)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 20
##
##           Mean of squared residuals: 0.1892927
##                     % Var explained: 89.37
```

```
ERA_pred = predict(Pitchers_RFM, test)
plot(ERA_pred,test$ERA, xlab='Prediction', ylab='Actual', main = 'Figure 8: Random Forest with 20 predictors a
t each split and 500 trees')
abline(0,1, col='red')
```

Figure 8: Random Forest with 20 predictors at each split and 500 trees



In opposition to a graphical representation of the model's accuracy, a side-by-side table of the actual ERA scores on the left and the model's prediction on the right is provided. The first 10 comparisons to the test set is provided and they are not outstandingly far off from one another. The total average test error over the course of all averaged predictions is 19.2% where the training error was given as 18.9%. The similarity in both these percentage indicates that the procedure did well and it addressed the over-fitting/under-fitting concern appropriately as the model did not perform drastically with the testing set.

```
cbind(Actual = test$ERA, Pred. = ERA_pred)[c(1:10),]
```

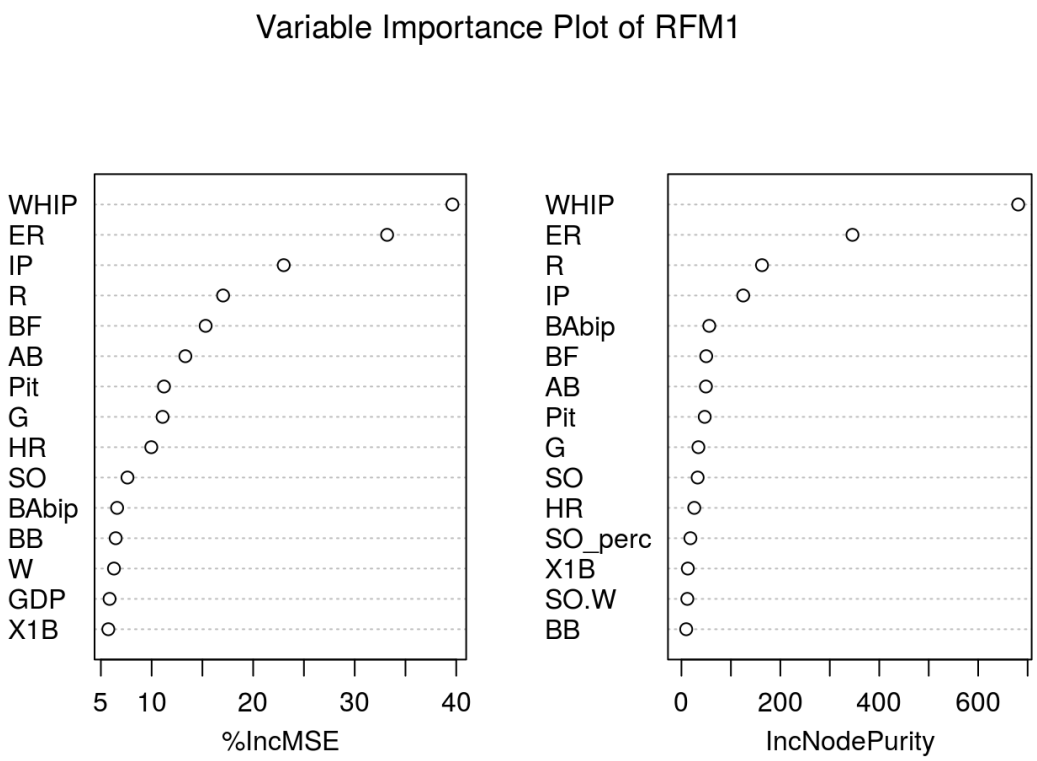
```
##    Actual    Pred.
## 1    6.26 6.437028
## 2    3.70 3.970300
## 3    3.40 4.126969
## 4    2.14 2.285803
## 5    4.67 4.617836
## 6    3.55 4.005935
## 7    3.52 3.513629
## 8    5.19 5.012825
## 9    4.66 4.517504
## 10   2.64 2.717965
```

```
mean((test$ERA-ERA_pred)^2)*100
```

```
## [1] 19.23366
```

Given the constructed model, it is just to interpret which variables were thought to be of greater importance. For the sake of the question, it is by choice that the '%IncMSE' plot is most meaningful for the results towards the posed question since we are dealing with a bagging scenario. This given metric interprets the increase in the MSE given that a given variable is extracted from the model. When one variable is removed then it analyzes its effects towards the MSE and thus we are given the order of importance. For the purposes of concentrating the amount of variables, the ones deemed most important are the following: WHIP, ER, IP, R, BF, and AB.

```
varImpPlot(Pitchers_RFM, n.var = 15, main= 'Variable Importance Plot of RFM1' )
```



Model – 2

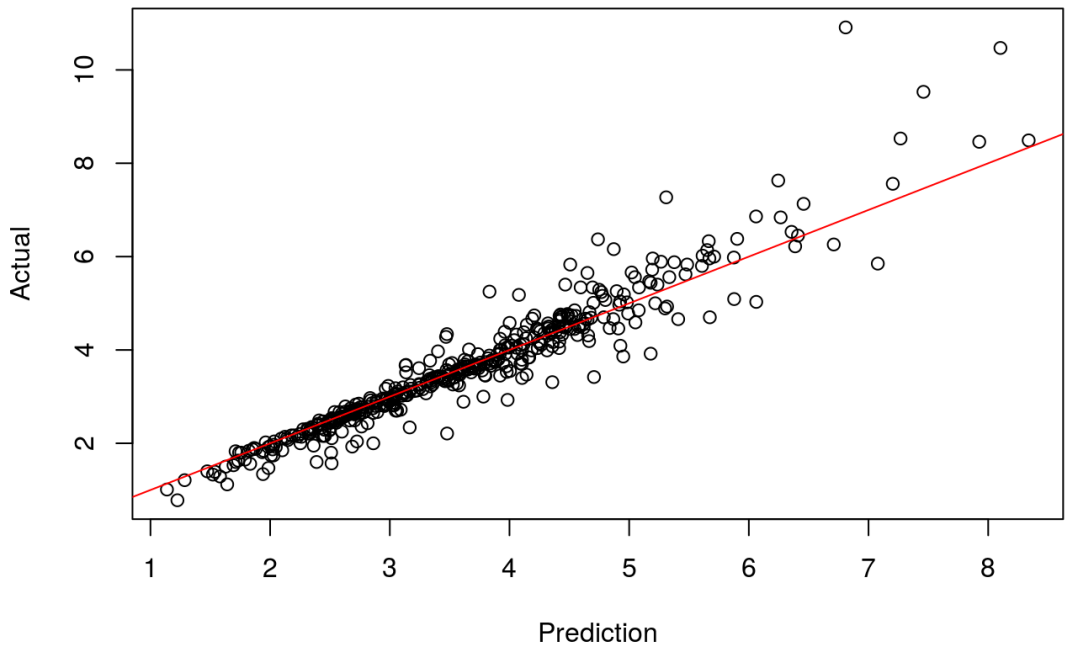
This second variation of a Random Forest model still accounts for the increase of 5 parameters for each node from the ideal one-third values to be a total of 20 variables considered at each split. However, instead of the default value for 500 trees the model now consists of 100 trees. The motivation for this model was to similarly keep the same amount of predictor randomness at each node, but now decrease the number of trees it can be done with. As seen through the summary of the model, the variance explained from this model was still close to 90% but slightly under Model 1 with a percentage of 88.79%. Similarly, this model does not seem to capture the larger values well as those values are scattered more around the linear line provided. Nonetheless, the model is still a plausible model to consider to make predictions.

```
set.seed(1)
Pitchers_RFM2 = randomForest(ERA~.-season, data=train, mtry=20,ntree=100, importance=TRUE)
Pitchers_RFM2
```

```
##
## Call:
##  randomForest(formula = ERA ~ . - season, data = train, mtry = 20,      ntree = 100, importance = TRUE)
##                Type of random forest: regression
##                      Number of trees: 100
## No. of variables tried at each split: 20
##
##           Mean of squared residuals: 0.1995805
##                     % Var explained: 88.79
```

```
ERA_pred2 = predict(Pitchers_RFM2, test)
plot(ERA_pred2,test$ERA, xlab='Prediction', ylab='Actual', main = 'Figure 9: Random Forest with 20 predictors
at each split and 100 trees')
abline(0,1, col='red')
```

Figure 9: Random Forest with 20 predictors at each split and 100 trees



The following shows the side-by-side comparison of the actual and predicted values from the second Random Forest model. In comparison to the first, this one has more noticeable differences in the values as they stray away from each other a bit more. That being said, the error for the testing set is 20.4%, which was slightly greater than the previous one. Again, it can be seen that over-fitting is not an issue as the testing error is not drastic as well as relatively close to the training error (~20%).

```
cbind(Actual = test$ERA, Pred. = ERA_pred2)[c(1:10),]
```

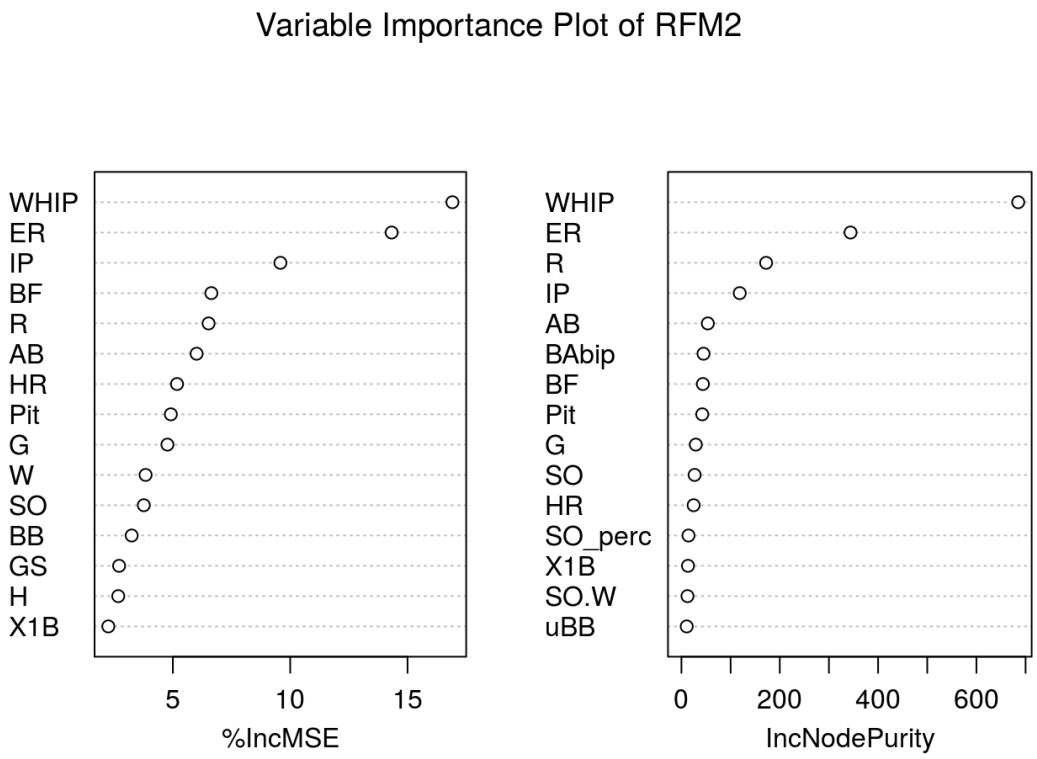
```
##    Actual    Pred.
## 1    6.26 6.709535
## 2    3.70 4.104445
## 3    3.40 4.105158
## 4    2.14 2.261488
## 5    4.67 4.634150
## 6    3.55 4.004365
## 7    3.52 3.553823
## 8    5.19 4.955695
## 9    4.66 4.430932
## 10   2.64 2.693483
```

```
mean((test$ERA-ERA_pred2)^2)*100
```

```
## [1] 20.41655
```

Similarly, this model has WHIP, ER, IP, BF, R, and AB as the top 6 predictors (referring to the '%IncMSE' plot). However, one small difference was the order for which the predictors BF and R were in. This model suggests that BF is of greater importance than R, but by a very slim margin. The level of importance for both variables practically appear as the same

```
varImpPlot(Pitchers_RFM2, n.var = 15, main= 'Variable Importance Plot of RFM2' )
```



Model – 3

The third variation of a Random Forest model is to now focus on the lower-bound of -5 from the ideal predictor split. Given that 15 was the ideal number, this model will focus on how well it will perform when a given split is given 10 predictors to look at. Like the first model, this variation will consist of the default number of 500 trees. Given the model summary, it performed worse than the previous two by only

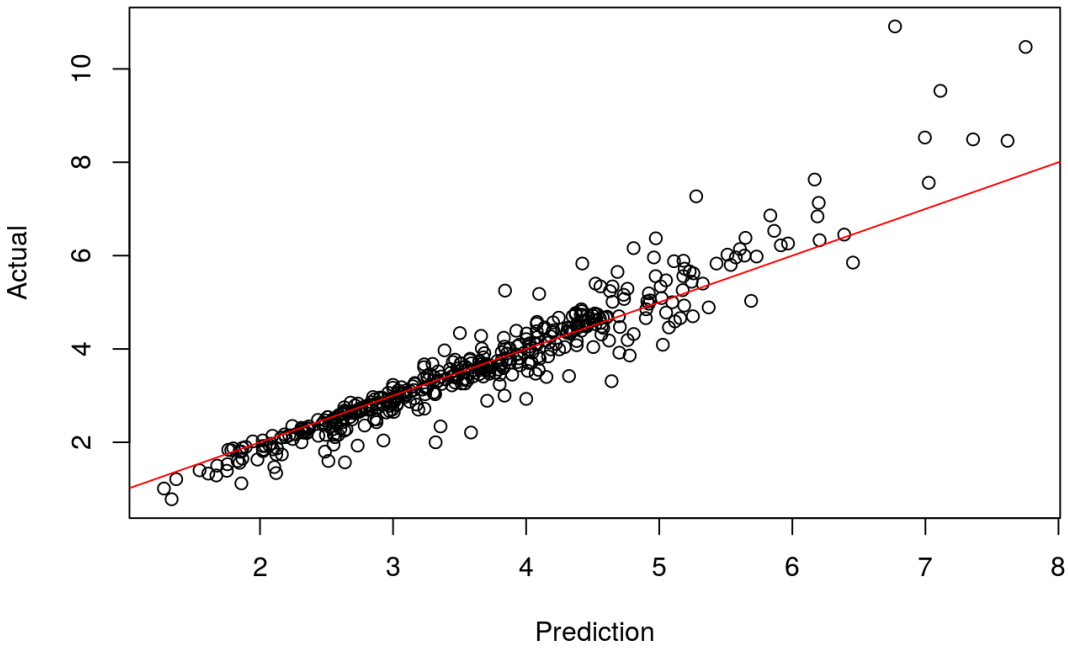
explaining 87.8% of the variance. Visually, it also appears to perform worse as the larger values are not captured well and just having most of those values like above the linear line. The model for the training data lacks a strong prediction accuracy as the points trend are not following the linear line the best.

```
set.seed(1)
Pitchers_RFM3 = randomForest(ERA~.-season, data=train, mtry=10,importance=TRUE)
Pitchers_RFM3
```

```
##
## Call:
##  randomForest(formula = ERA ~ . - season, data = train, mtry = 10,      importance = TRUE)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 10
##
##           Mean of squared residuals: 0.2172139
##                     % Var explained: 87.8
```

```
ERA_pred3 = predict(Pitchers_RFM3, test)
plot(ERA_pred3,test$ERA, xlab='Prediction', ylab='Actual', main = 'Figure 10: Random Forest with 10 predictors
at each split and 500 trees')
abline(0,1, col='red')
```

Figure 10: Random Forest with 10 predictors at each split and 500 trees



Numerically, the side-by-side comparison of the first 10 values are notably different. Despite being near the cutoff, one thing that can be seen is that more values are different according to the values on the left of the decimal. In that sense, most of the cases are around a > 0.3 difference, which in the long run already does not seem pleasing. With the averages of all these differences, the testing error rounds up to be 23.1%.

```
cbind(Actual = test$ERA, Pred. = ERA_pred3)[c(1:10),]
```

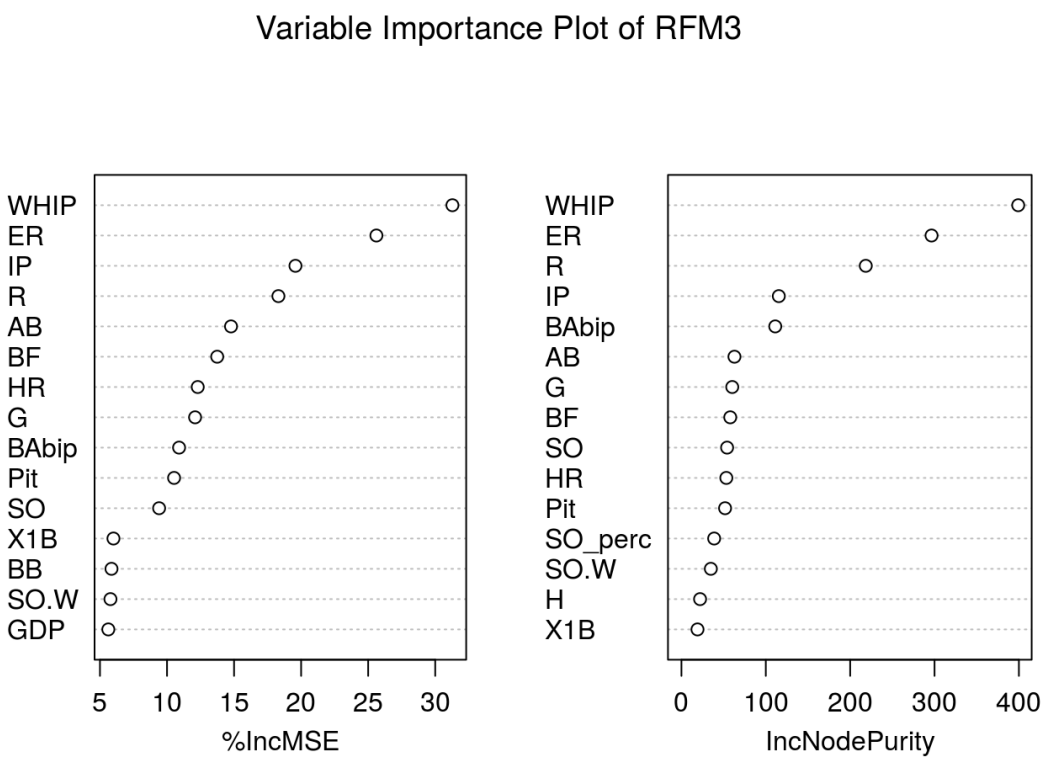
```
##    Actual    Pred.
## 1    6.26 5.967676
## 2    3.70 4.016728
## 3    3.40 4.152543
## 4    2.14 2.438757
## 5    4.67 4.576774
## 6    3.55 4.094396
## 7    3.52 3.535148
## 8    5.19 4.921595
## 9    4.66 4.469386
## 10   2.64 2.717084
```

```
mean((test$ERA-ERA_pred3)^2)*100
```

```
## [1] 23.13223
```

In terms of the variable importance plot, this model suggests the order of WHIP, ER, IP, R, AB, and BF being the top 6. Notice, how the order of the last three variables shifted in comparison to the previous two models. From this plot, the variable BF is now the 6th variable where in the previous two it was the 5th and 4th, respectively and AB shifted upwards.

```
varImpPlot(Pitchers_RFM3, n.var = 15, main= 'Variable Importance Plot of RFM3' )
```



Model – 4

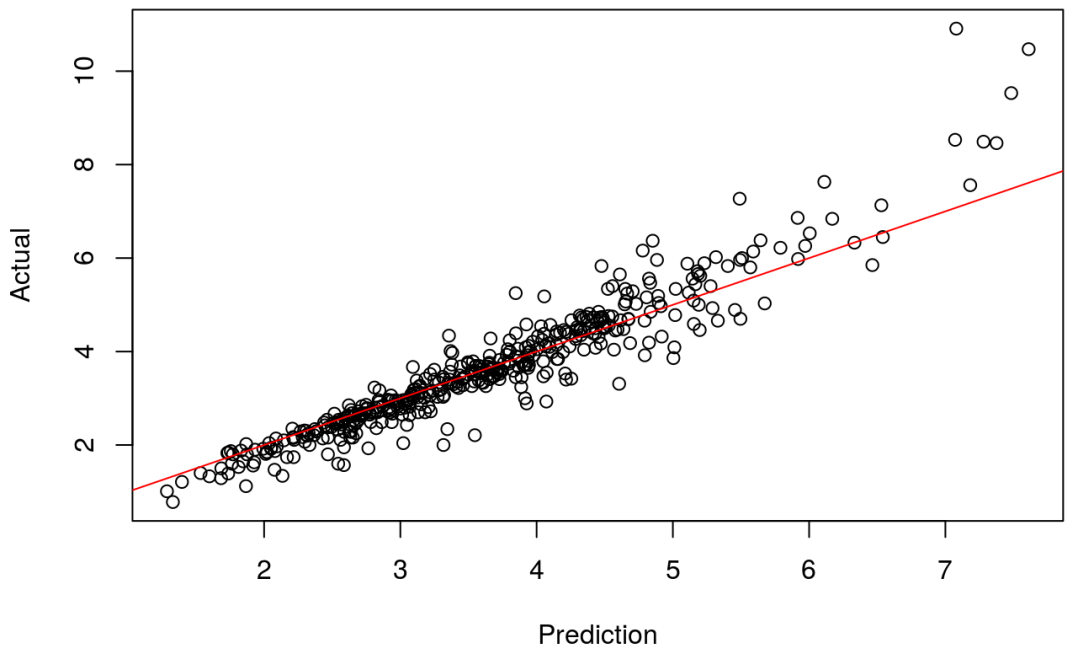
With a similar trend, this variation of a Random Forest model consists of the -5 variables at each split; however,less trees will be used now. From the summary, it appears that this variation performs the worst out of all 4. This model appears to explain 86.98% of the variance while having a training error of 23.2%. Visually, the trend at which the predicted and actual values follow does appear on the linear side more; however, scattered behavior of those points are concerning as it is evident a larger MSE will occur.

```
set.seed(1)
Pitchers_RFM4 = randomForest(ERA~.-season, data=train, mtry=10, ntree=100, importance=TRUE)
Pitchers_RFM4
```

```
##
## Call:
##  randomForest(formula = ERA ~ . - season, data = train, mtry = 10,      ntree = 100, importance = TRUE)
##                Type of random forest: regression
##                      Number of trees: 100
## No. of variables tried at each split: 10
##
##           Mean of squared residuals: 0.2319436
##                     % Var explained: 86.98
```

```
ERA_pred4 = predict(Pitchers_RFM4, test)
plot(ERA_pred4,test$ERA, xlab='Prediction', ylab='Actual', main = 'Figure 11: Random Forest with 10 predictors
at each split and 100 trees')
abline(0,1, col='red')
```

Figure 11: Random Forest with 10 predictors at each split and 100 trees



The side-by-side numerical values of the actual and predicted values are presented. Notice how the majority of difference surpasses a 0.3 threshold yielding to a testing error of 23.8%.

```
cbind(Actual = test$ERA, Pred. = ERA_pred4)[c(1:10),]
```

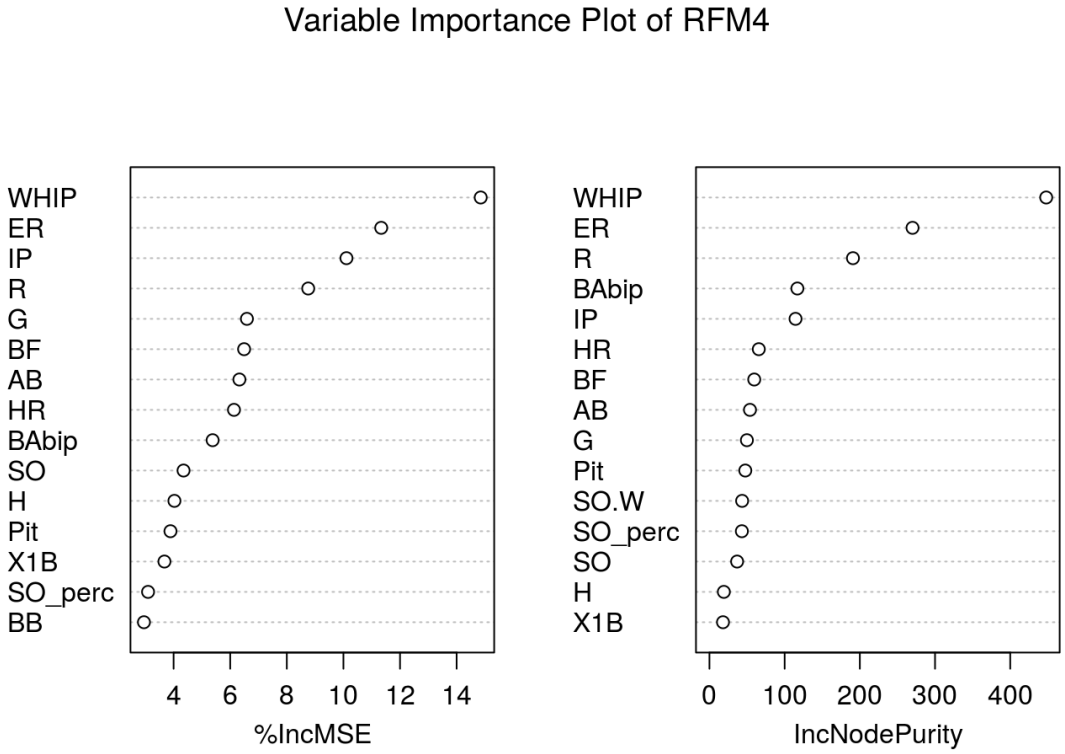
```
##    Actual    Pred.
## 1    6.26 5.971257
## 2    3.70 3.904793
## 3    3.40 4.216263
## 4    2.14 2.430597
## 5    4.67 4.374882
## 6    3.55 4.074122
## 7    3.52 3.547278
## 8    5.19 4.891978
## 9    4.66 4.468382
## 10   2.64 2.710940
```

```
mean((test$ERA-ERA_pred4)^2)*100
```

```
## [1] 23.80934
```

What is striking at this model is that the Variable Importance Plot introduces new variables in the top 6. For this model, the indicated important variables are WHIP, ER, IP, R, G, and BF. The first 3 remain the same as always; however, now we have a new variable 'G' that surpasses BF and AB. However, the fact that this model performed the worst from the previous ones, it is not as persuasive to believe that these variables model the question at hand the most.

```
varImpPlot(Pitchers_RFM4, n.var = 15, main= 'Variable Importance Plot of RFM4' )
```



Model – 5

Finally, the last variation consists of a fusion of the two aspects that presented itself in the past two models: 1) having higher amount of predictors at each node and 2) having a forest with a large amount of trees. With that, this model constructs a random forest with a compromise of both being a total of 17 variables being considered at each node. This values is +2 than the ideal 15 number mentioned earlier, but less than the +5 upper-bound that was experimented with in the first and second model. Then the amount of trees was kept the

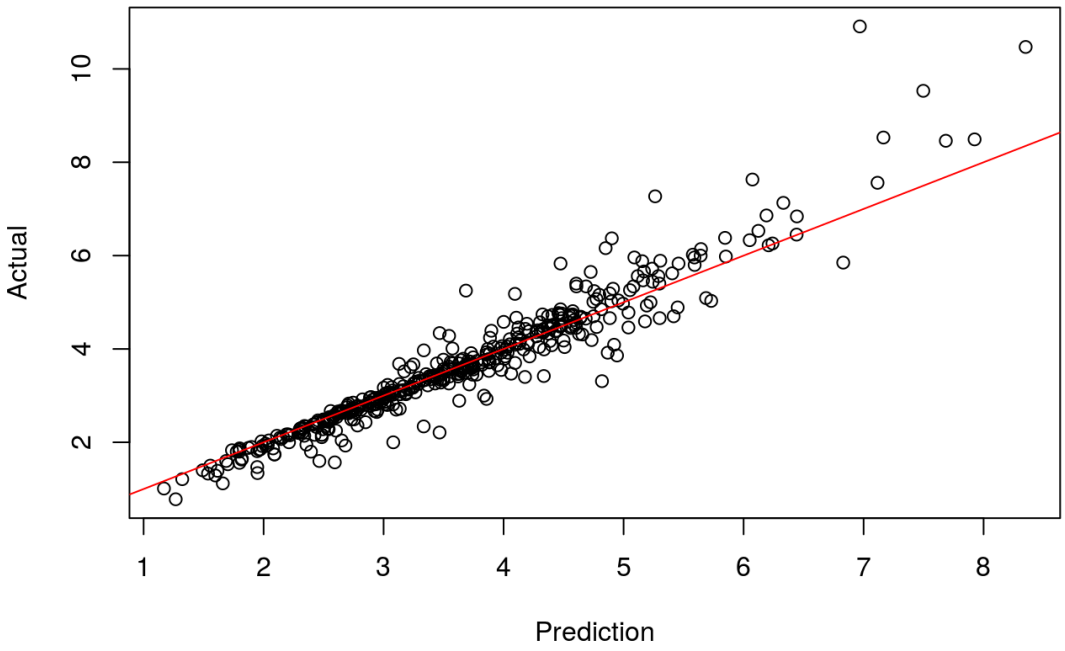
same as the default amount of 500. With these parameters, the summary of the model indicates 89.51% of the variance was explained with a training error of 18.7%. Out of these 5 constructed models, this one performs the best and visually the points are more compact and equidistant to the red-linear line. Although, there is sparsity in the plotted points, this model has the highest potential to model the question at hand.

```
set.seed(1)
Pitchers_RFM5 = randomForest(ERA~.-season, data=train, mtry=17, ntree=500, importance=TRUE)
Pitchers_RFM5
```

```
##
## Call:
##  randomForest(formula = ERA ~ . - season, data = train, mtry = 17,      ntree = 500, importance = TRUE)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 17
##
##           Mean of squared residuals: 0.186769
##                     % Var explained: 89.51
```

```
ERA_pred5 = predict(Pitchers_RFM5, test)
plot(ERA_pred5,test$ERA, xlab='Prediction', ylab='Actual', main = 'Figure 12: Random Forest with 17 predictors
at each split and 500 trees')
abline(0,1, col='red')
```

Figure 12: Random Forest with 17 predictors at each split and 500 trees



Many of these values are relatively close to one another and thus the average error over all predictions results in a testing error of 19.7%. So far, this model performs the best in both training and testing error and has no concerning factors to believe that it is over-fitting or under-fitting the data.

```
cbind(Actual = test$ERA, Pred. = ERA_pred5)[c(1:10),]
```

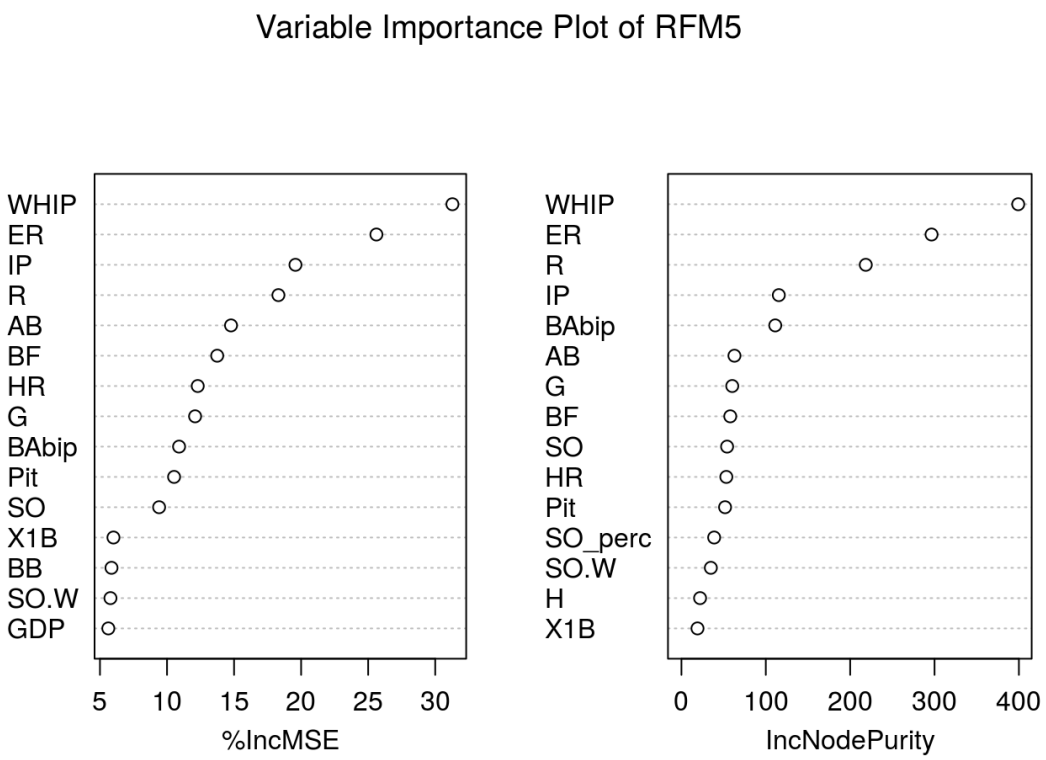
```
##    Actual    Pred.
## 1    6.26 6.240841
## 2    3.70 4.095730
## 3    3.40 4.177931
## 4    2.14 2.338696
## 5    4.67 4.640305
## 6    3.55 3.980461
## 7    3.52 3.516970
## 8    5.19 4.884166
## 9    4.66 4.485393
## 10   2.64 2.719587
```

```
mean((test$ERA-ERA_pred5)^2)*100
```

```
## [1] 19.66083
```

Given the best performing model out of the 5, the validity of the importance plot has more credibility in its choice. With the first three being staple across all models WHIP, ER, and IP remain in the same order and R, AB, and BF following in the latter.

```
varImpPlot(Pitchers_RFM3, n.var = 15, main= 'Variable Importance Plot of RFM5' )
```



Overall, this section covered 5 variations of a Random Forest model focusing in on the ‘random’ aspects: bagging, bootstrapping, and the number of trees. The motive for the variations of Random Forest models was to get a better understanding of which design factors lead to a model that explained a higher percentage of the variance for a given training set. In a given bootstrapping and bagging environment, the model was analyzed around the ideal number of predictors for a given split. For this regression question, the ideal number of predictors for a split was 15 given by the expression: total # of predictors/3. Given this number, the models explored a -5/+5 bound for predictors for analytic purposes. Then, a final model the knit-picked the effective design aspects from each model into one where 17 predictors and 500 trees were given. Unfortunately, these model designs encompassed the same assumption that it would perform its best around the ideal number of

predictors for each split, so it goes without saying other design aspects could be more efficient. Nonetheless, comparing the last model to the others, the design principals are justified as it performed the best. That being said, our best performing Random Forest model indicates the following variables (WHIP, ER, IP, R, AB, BF) as the most important in the scope of the regression question.

Section 2: Classification

#Preparation of Data for Classification Approach To set up the classification problems from a numeric data-set, the below lines of code constructs four even intervals across the range of values and gives those interval of values a label as seen.

```
maxmin_Pitcherstrain$ERA = cut(maxmin_Pitcherstrain$ERA, 4, labels=c('Strong Pitcher', 'Fairly Good Pitcher',
'Weak Pitcher', 'Pitcher in Trouble'))
table(maxmin_Pitcherstrain$ERA)
```

```
##
##      Strong Pitcher Fairly Good Pitcher        Weak Pitcher  Pitcher in Trouble
##                 602                 386                  17                   3
```

```
maxmin_Pitcherstest$ERA = cut(maxmin_Pitcherstest$ERA, 4, labels=c('Strong Pitcher', 'Fairly Good Pitcher', 'W
eak Pitcher', 'Pitcher in Trouble'))
table(maxmin_Pitcherstest$ERA)
```

```
##
##      Strong Pitcher Fairly Good Pitcher        Weak Pitcher  Pitcher in Trouble
##                 190                 212                  23                   6
```

9 Naive Bayes

9.1 Introduction

The Naive Bayes classifier is a method that constructs a prediction based on both prior knowledge and current information. From a statistical standpoint, the prior knowledge is referred to as the prior probability and the prediction is referred to as the posterior probability. The posterior probability is of more interest since it reflects what option would be the most probable. The following expression, constructs the computation need for the posterior probability:

$$P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)}$$

where P(Y) is the probability of the event Y occurring, P(X) is the probability of the event X occurring, P(Y|X) is the conditional probability of Y given X, and P(X|Y) is the conditional probability of X given Y.

It is assumed that all the predictors are continuous and thus independent of each other given the same outcome.

9.2 Application

The Naive Bayes model consists of the same function as before such that we want to model ERA through the rest of the predictors. However, for the classification problem the Response variable ‘ERA’ now has 4 evenly divided layers with a given label. Rather than trying to fit the data to a single numeral value, it is now fitting it to an interval or label. The provided information of the model details the prior and conditional probabilities as mentioned mathematically. For a given predictor, the the Y-variables denote the conditional probability mean and standard deviations for every class, respectively. Similarly to the formula given above, the model uses the given prior and conditional probabilities to compute the posterior probability. It then

```
set.seed(1)
Pitchers_nBayes = naiveBayes(ERA~., data=maxmin_Pitcherstrain)
```

The naive Bayes classification procedure is a fast procedure, but at times its efficiency may be questionable. Even with a fitted model on the training data, the predictions once again made on the training data yielded in an 81% accuracy. The model predictions on the training data seemed to do well with the 'Pitcher in Trouble" level because there was only a few and most likely over-fitted to those outliers, but for the level with more data points it yielded in more misclassifications. In particular, for both the training and testing data predictions, the model does not seem to do well with the 'Strong Pitcher' level. The model predicts the level "Fairly Good Pitcher" a number of times for the true value of "Strong Pitcher". The mis-classification rate for this particular label is approximately 30% for both the testing and training predictions. Other than that, the testing accuracy was 76%, suggesting that the model overfitted the training data and was not able to accommodate for new, unseen cases in the the testing data.

```
nBayes_predtrain = predict(Pitchers_nBayes, maxmin_Pitcherstrain)
(nBayes_traintable = table(nBayes_predtrain, maxmin_Pitcherstrain$ERA))
```

```
##
## nBayes_predtrain      Strong Pitcher Fairly Good Pitcher Weak Pitcher
##   Strong Pitcher                554                 119            0
##   Fairly Good Pitcher            48                 245            0
##   Weak Pitcher                    0                  21           17
##   Pitcher in Trouble              0                   1            0
##
## nBayes_predtrain      Pitcher in Trouble
##   Strong Pitcher                      0
##   Fairly Good Pitcher                 0
##   Weak Pitcher                        0
##   Pitcher in Trouble                  3
```

```
trainAcc=sum(diag(nBayes_traintable))/sum(nBayes_traintable)

nBayes_predtest = predict(Pitchers_nBayes, maxmin_Pitcherstest)
(nBayes_testtable = table(nBayes_predtest, maxmin_Pitcherstest$ERA))
```

```
##
## nBayes_predtest       Strong Pitcher Fairly Good Pitcher Weak Pitcher
##   Strong Pitcher                174                  68            1
##   Fairly Good Pitcher            16                 133            4
##   Weak Pitcher                    0                  11           16
##   Pitcher in Trouble              0                   0            2
##
## nBayes_predtest       Pitcher in Trouble
##   Strong Pitcher                      0
##   Fairly Good Pitcher                 0
##   Weak Pitcher                        3
##   Pitcher in Trouble                  3
```

```
testAcc=sum(diag(nBayes_testtable))/sum(nBayes_testtable)

(round(cbind(Train_Accuracy=trainAcc, Test_Accuracy=testAcc),2))
```

```
##      Train_Accuracy Test_Accuracy
## [1,]           0.81          0.76
```

Overall, the naive Bayes classification procedure is a simple yet fast-performing method. The foundation that it incorporates the prior probabilities for each variable make it a unique approach as it narrows down the options of understanding the occurrence of one event and then calculating the impact it has on the following event. The provided performance of the naive Bayes method is a fairly decent approach as the accuracy for both the training and testing data predictions are not under 70%. A potential fix to increase the accuracy is to potentially create intervals such that each interval encompasses an equal amount of data points.

10 K-Nearest Neighbors

10.1 Introduction

Another candidate method to consider the classification problem would be a non-parametric one – otherwise known as k-Nearest Neighbors (kNN). Rather than having the 'beta' parameters in a logistic regression setting, kNN has a tuning parameter 'k'. Similar to how the 'Beta' parameter is learned through training, the tuning parameter is what determines how the model is trained.

Mathematically, kNN constructs conditional probabilities using the 'k' neighbors expressed as the following:

$$\hat{p}_{kg}(x) = \hat{P}_k(Y = g|X = x) = \frac{1}{k}\sum_{i\epsilon N_0} I(y_i = g)$$

where Y is the response, X is the matrix of predictors, 'g' denotes an individual class, 'k' is the number of neighbors used relevant to the point 'x', N is the set of the k nearest neighbors, and I is an indicator function that evaluates to 1 if the observation in the neighborhood is a member of class 'g' and 0 for when it is not.

Then, the classifier takes the highest estimated probability and assigns the given point to that respective class.

$$\hat{C}_k(x) = \underset{g}{\text{argmax}}\, \hat{p}_{kg}(x)$$

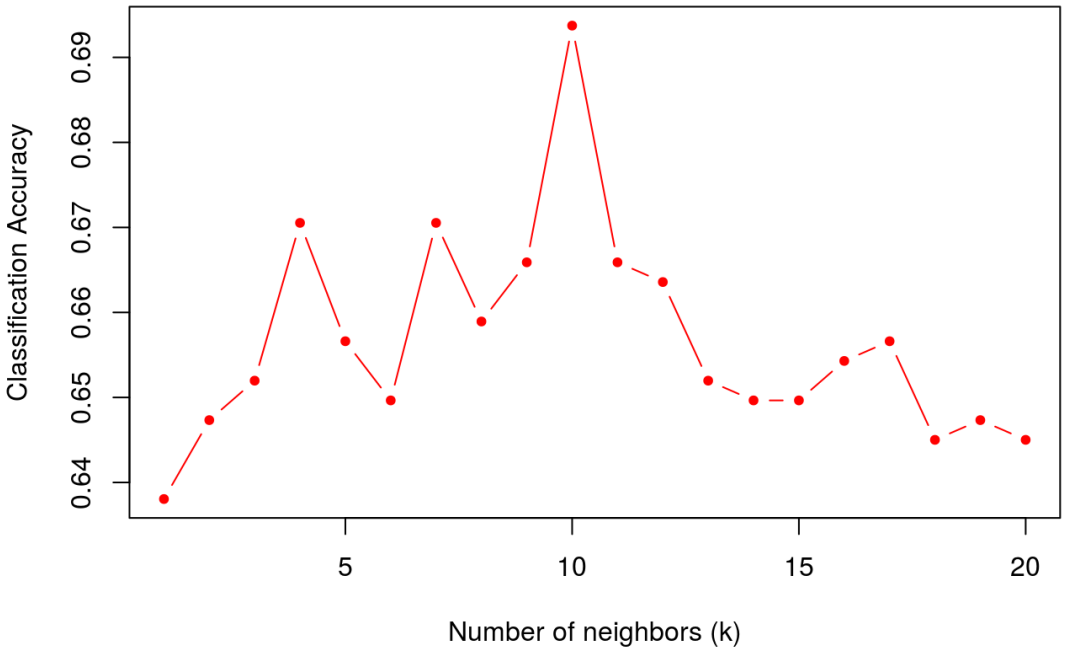
10.2 Application

```
set.seed(1)
num_K = 1:20
knn_err = rep(0, times=length(num_K))

for (i in seq_along(num_K)){
  Pitcher_kNNpred = knn(maxmin_Pitcherstrain[,-1],
                        maxmin_Pitcherstest[,-1],
                        maxmin_Pitcherstrain$ERA,
                        k=num_K[i])
  (knn_predtable = table(Pitcher_kNNpred, maxmin_Pitcherstest$ERA))
   knn_err[i]=sum(diag(knn_predtable))/sum(knn_predtable)
  }

plot(knn_err, type='b', col='red',pch=20,
     xlab="Number of neighbors (k)",
     ylab="Classification Accuracy",
     main= "Figure 13: Test Accuracy over various k-choices")
```

Figure 13: Test Accuracy over various k-choices



Given the different 20 cases of k values, Figure 13 visually provides the MSE and it can be seen that it reaches its peak with 10 neighbors. The use of 10 values leads us to the model having a higher accuracy rate in classifying the 4 classes. Below, the plot can be interpreted numerically as the accuracy is given side-by-side with the number of k-values used in the model.

```
cbind(Number_of_K = 1:20, Accuracy = knn_err[1:20])
```

```
##       Number_of_K  Accuracy
##  [1,]           1 0.6380510
##  [2,]           2 0.6473318
##  [3,]           3 0.6519722
##  [4,]           4 0.6705336
##  [5,]           5 0.6566125
##  [6,]           6 0.6496520
##  [7,]           7 0.6705336
##  [8,]           8 0.6589327
##  [9,]           9 0.6658933
## [10,]          10 0.6937355
## [11,]          11 0.6658933
## [12,]          12 0.6635731
## [13,]          13 0.6519722
## [14,]          14 0.6496520
## [15,]          15 0.6496520
## [16,]          16 0.6542923
## [17,]          17 0.6566125
## [18,]          18 0.6450116
## [19,]          19 0.6473318
## [20,]          20 0.6450116
```

```
## Value of K with the Highest Accuracy
which.max(knn_err)
```

```
## [1] 10
```

Overall, this model explores a variety of options to play around with the k-value. It looked at values from 1 to 20 neighbors and assessed its prediction accuracy for each case. Given the peak of 10 neighbors, the model resulted in a 69% classification accuracy. Although it is a moderately high number and close to my passing criteria off 70% accuracy, this model may have complications with the complexity of the data set being large as well as containing potential underlying relationships. Given a theoretical plot of the data points respective to their labels, the ERA values are bound to overlap especially because there are a lot of occurrences for the first two classes. That being said, the overlap and fusion of the class data points having a boundary for which the model picks 'k' amount of neighboring points will prove to be difficult. It will be inevitable that the influence the neighbors have will be in the same effect that another point of a different class has. In addition, it is probable that dealing with the 4th class or a class where there are less observations it will construct a very large boundary since typically these are the outliers and this has the chance of increasing the mis-classification error. Despite these personal claims, the model does an average job in predicting the 4 classes.

11 Support Vector Machines

11.1 Introduction

The following section will cover Support Vector Machine functionality in the scope of non-linear data as real-life data is never as simple to be divided by a linear trend. The idea behind support vector machines for classification is to provide a decision boundary for the different labels and this decision boundary lives in an n-dimensional space where 'n' is the amount of features in a given dataset. Given the pitcher dataset and its 43 features, each value or observation is assigned their corresponding coordinate in a 43-dimensional plot. Each point is then given their corresponding label. In this case there are 4 classes, so our hyper-plane will be constructed to differentiate between the 4 labels.

In a non-linear setting with a non-linear dimension, it is possible to just map the space into a higher dimension; however, as the dimension increases, this strategy can be costly. With this strategy, a lot of transformations will need to be calculated and each calculation can potentially be complicated. Applying the transformation to every vector is not the best idea, so instead a dot product is computed between the vectors. This dot product motivation is also known as the kernel. It essentially makes the feature space larger to provide that non-linear boundary needed to differentiate the classes.

Given the brief introduction to Support Vector Machines, it is suggestive that this model is effective in dealing with high-dimensional spaces. From the kernel function, the method is being memory conscious as it is being efficient through the dot product procedure. Lastly, the decision boundary can be constructed flexibly such that the separation of classes are more distinctive and leads to better classification performance.

With the given dataset, there is no worry about poor performance through the parameter selection for the kernels because the feature space does not exceed the number of observations. With a feature space larger than the number of observations, it runs into the issue of finding effective support vectors for a better classifying hyper-plane. However, like the previous two methods, this classification method in non-probabilistic as each observation is placed in a given location relative to the decision boundaries. Then, the boundaries are updated to be optimal as the distances to the boundary needs to be minimal.

11.2 Mathematical Interpretation

To interpret the procedure of Support Vector Machines, there are a few smaller components that must set the foundation first. 1. The Length of a Vector * Given a n-dimensional vector "x", we can find the length of a vector ("||x||") with the given formula:

$$||\text{x}|| = \sqrt{x_1^2 + x_2^2 + \ldots + x_n^2}$$

2. The Direction of a Vector

- Given a simple 2D-vector with components x1 and x2, the direction (z) of the vector can be found using the following:

$$z = (\frac{x_1}{||x||}, \frac{x_2}{||x||})$$

3. The Dot Product of two vectors

- The dot product of two vectors x and y is done component wise as denoted by the following:

$$x \cdot y = \sum_{i=1}^{N} (x_i y_i)$$

4. Idea of Linearly Separable

- Given a two dimensional data set, the idea of it being linearly separable is if the two classes can be distinguished as easily with a linear line. If persay the data points are mix withing each other too much then it will be nearly impossible to construct a linear line to differentiate the two classes - indicated linearly inseparable.

5. Construction of Hyperplane

- For reasons of simplicity, the following will deal with a two dimensional case. If a data set is linearly, separable then it is possible to construct a hyperplane, or line, between the two given by

$$0 = ax_1 - x_2 + b$$

- Given the steps above, we can then transform the hyperplane equation as

$$U \cdot X + b = 0$$

where U is a vector containing the coefficients a and -1 and X is a vector containing the components x1 and x2.

6. Applying Hyper-plane to a Classifier

- This step is rather simple and intuitive. Given out hyperplane equation (

$$U \cdot X + b = 0$$

), it is possible to set up a piece-wise expressions where when evaluated over or equal to 0 then it will be classified as 1 where as if it resulted in a value lower than zero then the classifier will identify it as -1.

$$\begin{cases} +1 \text{ if } U \cdot X + b \geq 0 \\ -1 \text{ if } U \cdot X + b < 0 \end{cases}$$

Given this set of information, we want to find out how to construct the decision boundaries. With a given point (x,y) that we know if not on the hyperplane for a 2D case, the equation can either result in a positive or negative number, but the main concern during the training procedure is to find the point that is closest to this hyperplane. Given the dataset of x and y points, where in terms of our previous proposed classifier y is either -1 or 1, the model will try to update its weights in attempt to lowest the cost function. Mathematically this can be interpreted as the following:

Defining f to be the classifier label that can either be positive or negative depending on $f = y(U \cdot X + b)$.

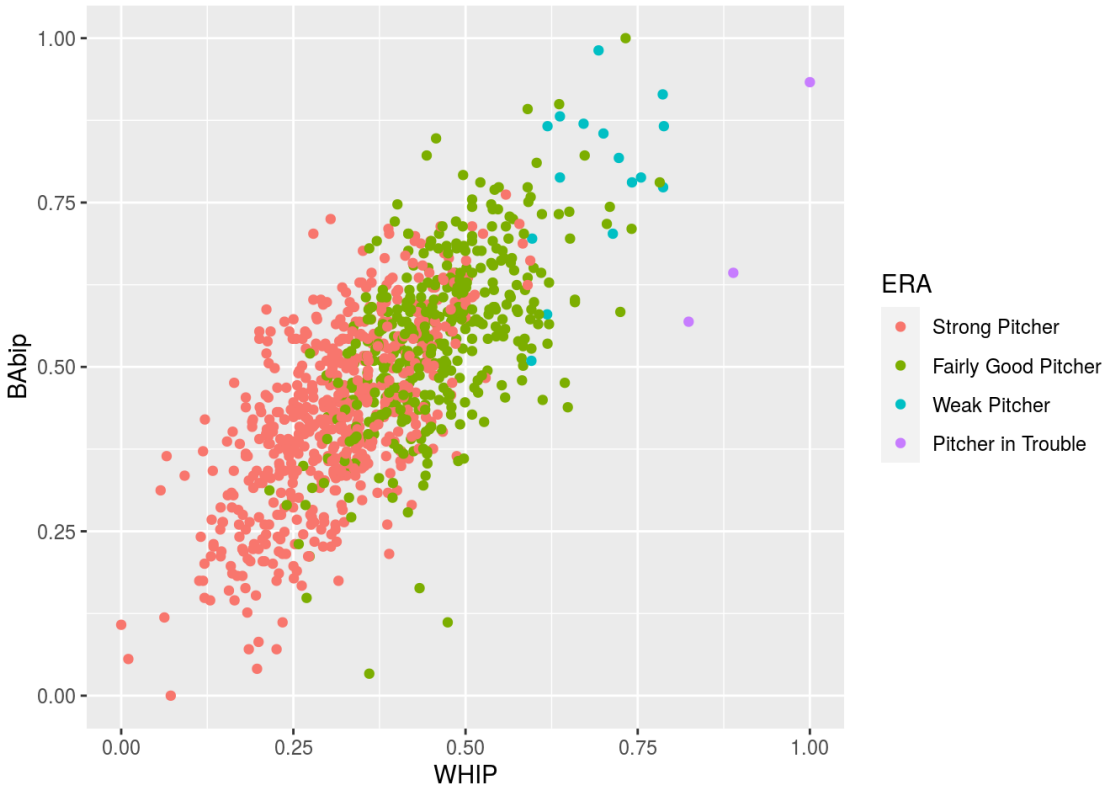
Then, it is possible to find the functional margin(F) through $F = \underset{i=1...m}{\operatorname{argmin}}\, y(U \cdot X + b)$.

As the hyperplanes are compared, the model will choose the one with the largest F value.

11.3 Application

Visually, we have a plot of a WHIP vs BAbip data points where a color coordinating system is in place for the factor variable ERA as given in the plot legend. From first impressions, the level of “Pitcher in Trouble” appears to be a simple matter where as constructing an effective boundary between “Strong Pitcher” and “Fairly Good Pitcher”. The goal of SVM is to create the hyperplane that divides the classes with the proper decision boundaries based on the supporting vectors. Further on, there will be some representations on the variables WHIP, BAbip, ER, HR,G, R, and SO.W. These were personally chosen as it refers to being the top 7 of the 10 highly correlated variables. For the sake of a simpler exploration, these features will focus in on “WHIP” as this was another personal choice for having the highest correlation with ERA.

```
qplot(WHIP, BAbip, data=maxmin_Pitcherstrain, color=ERA)
```



For such a high-dimensional procedure, a few combinations were tried with respect to the cost and gamma value for a SVM model. Again, the provided intervals are established to be a more efficient one for proper exploration purposes. From exploring different intervals, cost was explored from 1 to 8 in powers of 2 and gamma was explored from 0 to 0.01 with a small step size of 0.001. Once the different SVM methods were constructed, the summary indicated the best model was the one having a cost of 8 and 0.004. With these parameters, the training error returned was 10%. With a 10% error, it appears that this model does a outstanding job in classifying the different classes.

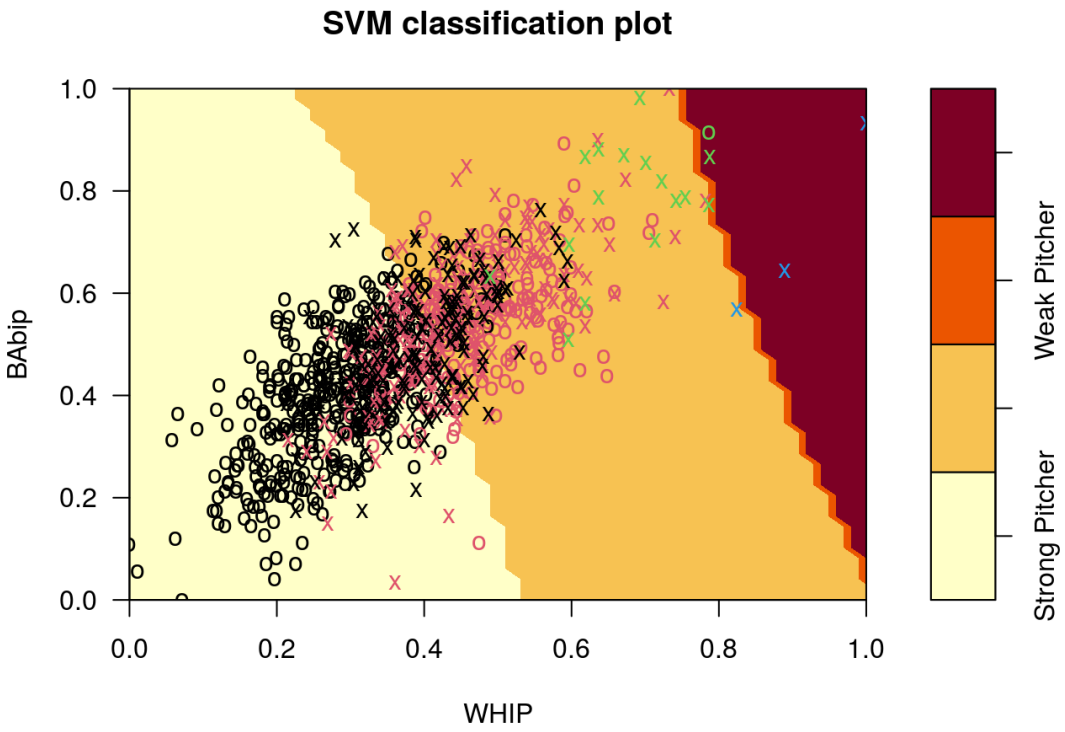
```
set.seed(1)

tuned_SVM = tune(svm, ERA~., data=maxmin_Pitcherstrain[,-19],
                 ranges = list(gamma=seq(0,0.01,0.001), cost=2^(0:3)))
summary(tuned_SVM)
```

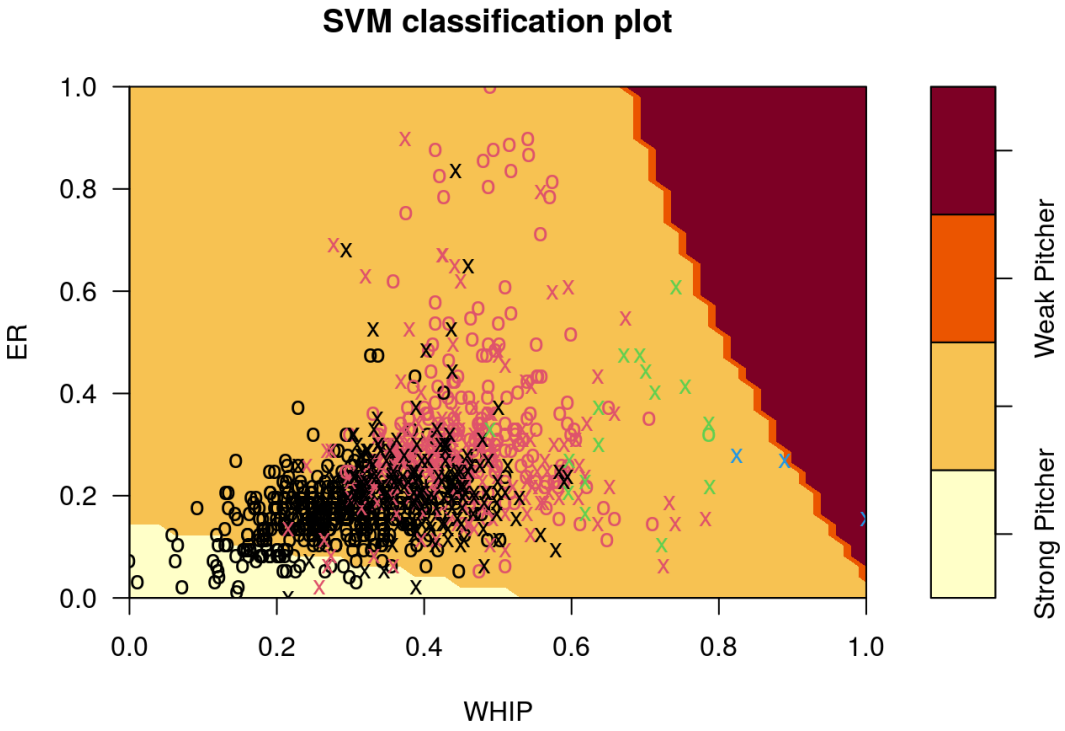
```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  gamma cost
##  0.004    8
##
## - best performance: 0.1001782
##
## - Detailed performance results:
##    gamma cost     error dispersion
## 1  0.000    1 0.4027525 0.05698373
## 2  0.001    1 0.1726139 0.05050413
## 3  0.002    1 0.1558020 0.04351030
## 4  0.003    1 0.1439010 0.04331945
## 5  0.004    1 0.1448713 0.04567086
## 6  0.005    1 0.1409109 0.03505708
## 7  0.006    1 0.1389505 0.03735741
## 8  0.007    1 0.1369406 0.03473711
## 9  0.008    1 0.1438911 0.03490770
## 10 0.009    1 0.1438812 0.03325664
## 11 0.010    1 0.1399010 0.02981950
## 12 0.000    2 0.4027525 0.05698373
## 13 0.001    2 0.1508416 0.04492197
## 14 0.002    2 0.1429010 0.03702253
## 15 0.003    2 0.1329604 0.02747867
## 16 0.004    2 0.1319505 0.02767708
## 17 0.005    2 0.1309505 0.03016928
## 18 0.006    2 0.1259802 0.03232771
## 19 0.007    2 0.1190495 0.02684932
## 20 0.008    2 0.1190495 0.03028111
## 21 0.009    2 0.1220396 0.03502890
## 22 0.010    2 0.1210396 0.03554966
## 23 0.000    4 0.4027525 0.05698373
## 24 0.001    4 0.1379505 0.03142594
## 25 0.002    4 0.1200297 0.02935951
## 26 0.003    4 0.1150693 0.02880860
## 27 0.004    4 0.1081485 0.02828450
## 28 0.005    4 0.1091485 0.02930887
## 29 0.006    4 0.1111188 0.02513682
## 30 0.007    4 0.1121188 0.02253423
## 31 0.008    4 0.1140990 0.02308435
## 32 0.009    4 0.1130891 0.02565020
## 33 0.010    4 0.1150891 0.02929233
## 34 0.000    8 0.4027525 0.05698373
## 35 0.001    8 0.1130792 0.02604772
## 36 0.002    8 0.1031584 0.02559892
## 37 0.003    8 0.1021485 0.03192803
## 38 0.004    8 0.1001782 0.02612848
## 39 0.005    8 0.1011782 0.02673057
## 40 0.006    8 0.1011683 0.02270804
## 41 0.007    8 0.1011782 0.02504768
## 42 0.008    8 0.1041584 0.02852024
## 43 0.009    8 0.1071485 0.02874664
## 44 0.010    8 0.1121188 0.03106868
```

Given that our best model was one of cost=8 and gamma=0.004, the plots below take that same model and illustrate the decision boundaries that separate the classes. Due to there being very few observations of the last class, the predominant boundaries are in accordance to “Strong Pitcher” and “Fairly Good Pitcher” mostly. One thing to note is how small of a sliver the model constructs a boundary for the “Weak Pitcher” label.

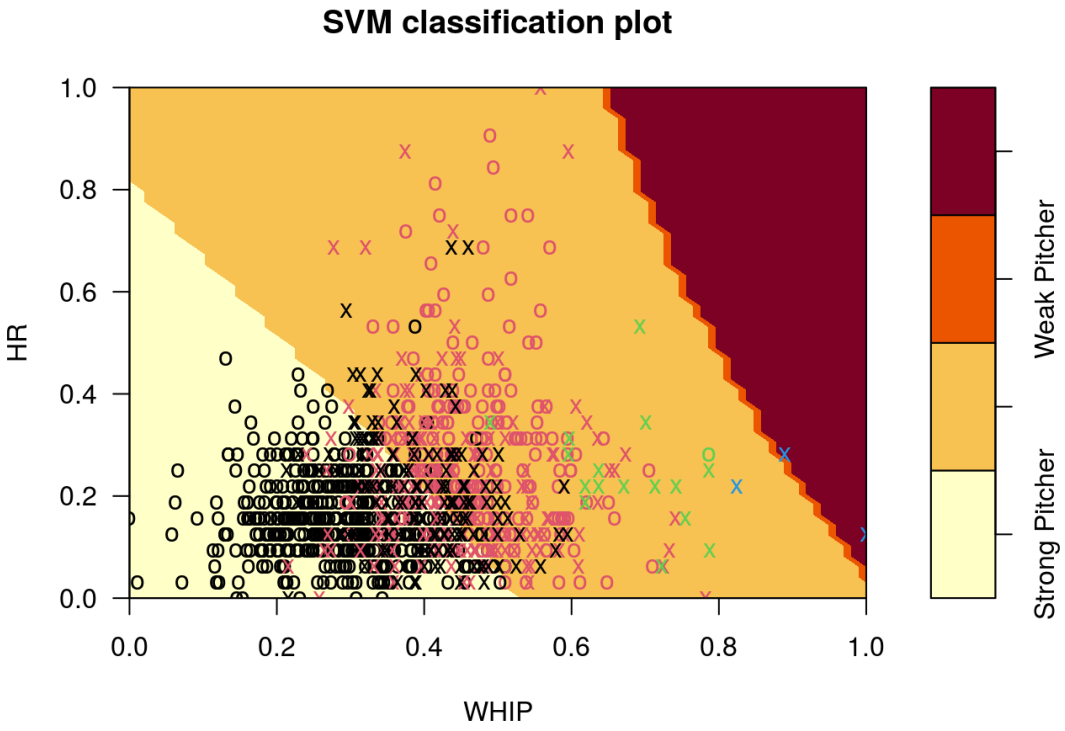
```
plot(tuned_SVM$best.model, data=maxmin_Pitcherstrain, BAbip ~ WHIP)
```



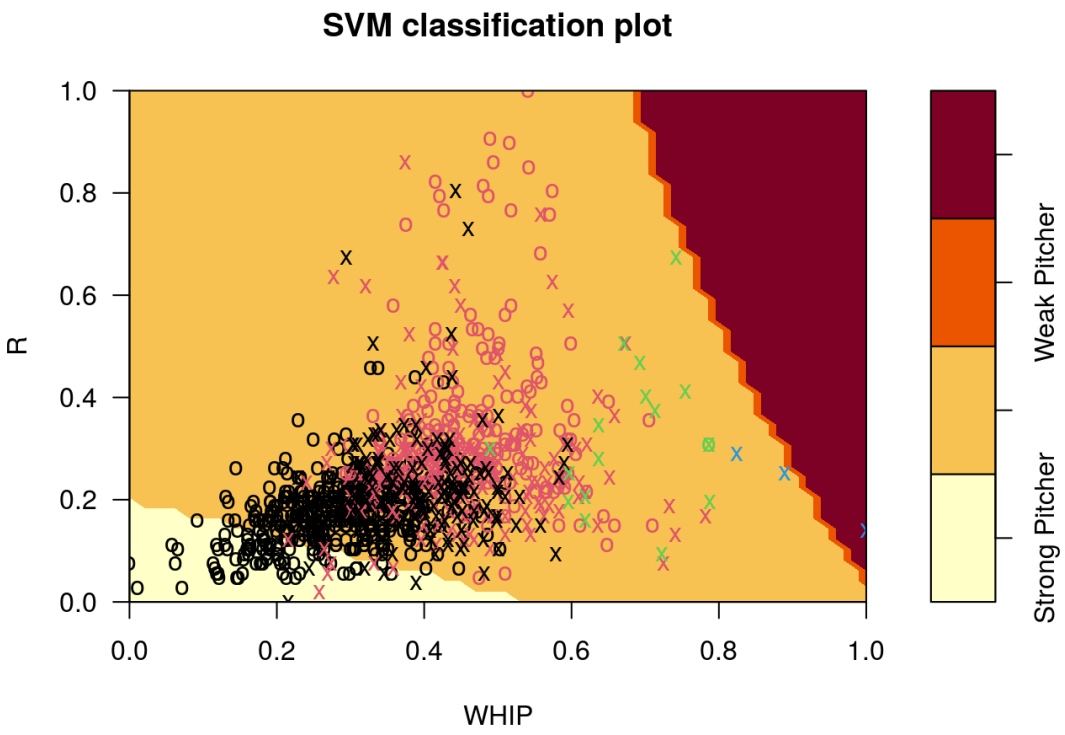

```
plot(tuned_SVM$best.model, data=maxmin_Pitcherstrain, ER ~ WHIP)
```



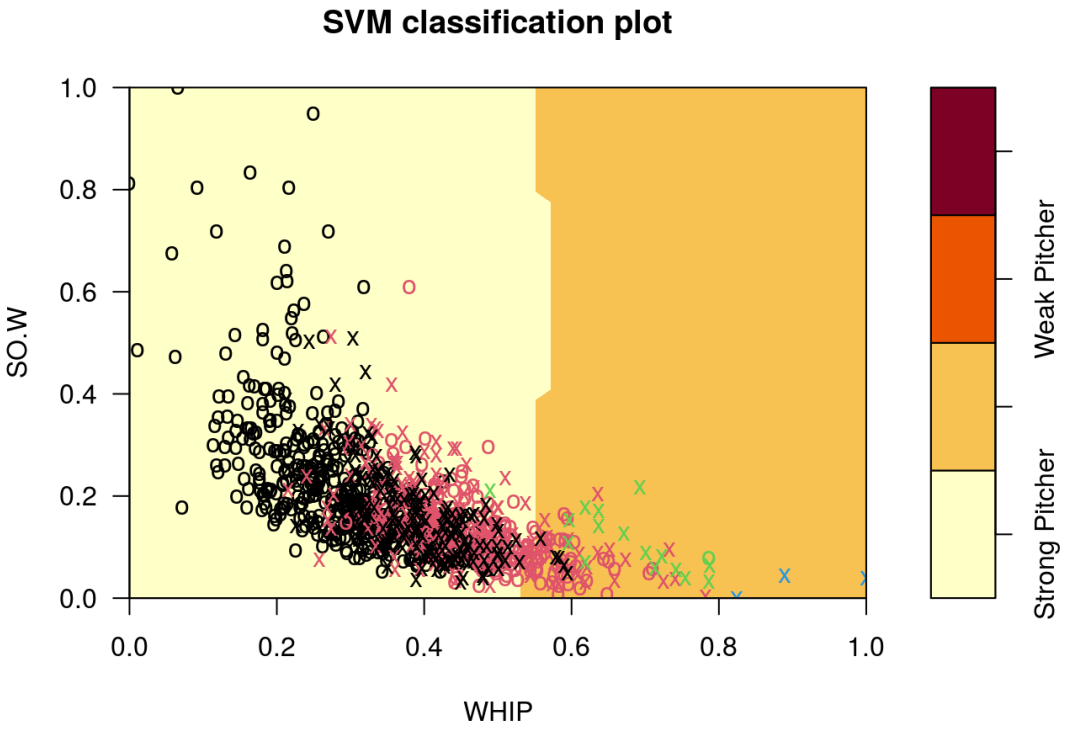

```
plot(tuned_SVM$best.model, data=maxmin_Pitcherstrain, HR ~ WHIP)
```



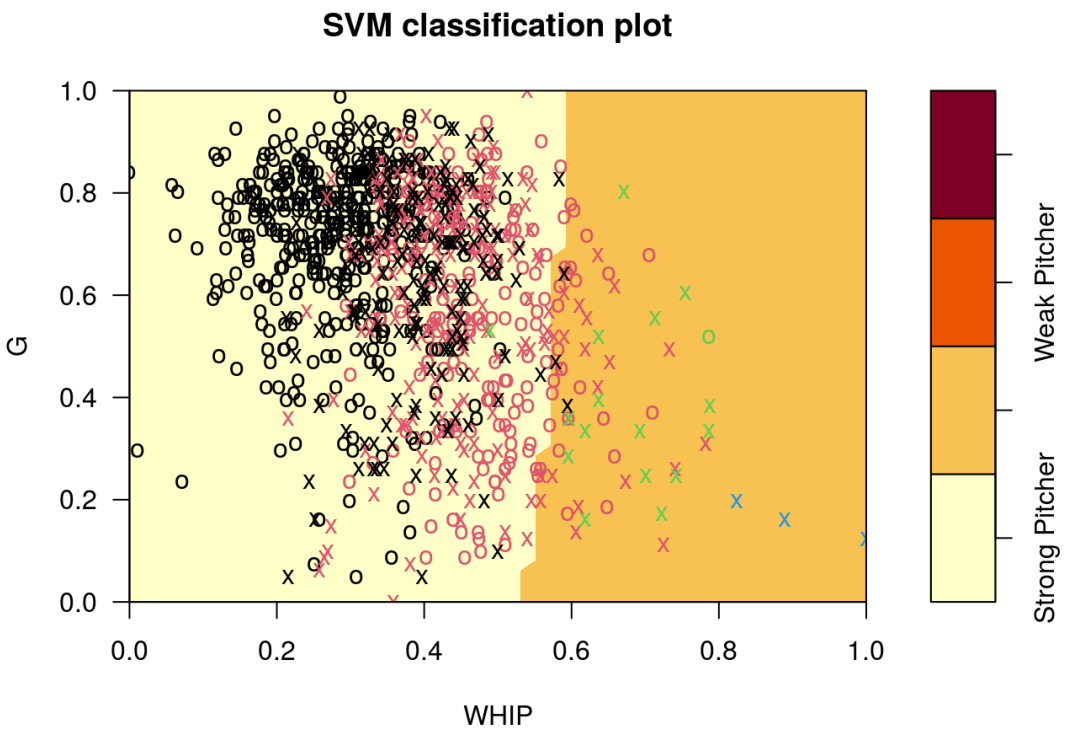

```
plot(tuned_SVM$best.model, data=maxmin_Pitcherstrain, R ~ WHIP)
```




```
plot(tuned_SVM$best.model, data=maxmin_Pitcherstrain, SO.W ~ WHIP)
```




```
plot(tuned_SVM$best.model, data=maxmin_Pitcherstrain, G ~ WHIP)
```



Lastly, to assess the performance of the model to a new set of data, or the test set, the same best model where cost=8 and gamma=0.004 is applied to the predict function. In the confusions matrix, it seems to predict the first two classes (Strong Pitcher and Fairly Good Pitcher) well, but misses all observations for the last label (Pitcher in Trouble). The “Weak Pitcher” label is also largely mis-classified having one 1 being correct. The model seems to perform better with observations whose count for classes is a larger number. The issue that presents itself here is that from the training procedure, the model seems to over-fit the data and this the accuracy for the testing process suffers. Given the best model found, the model obtains a 73% accuracy and a 27% error.

```
SVM_pred = predict(tuned_SVM$best.model, maxmin_Pitcherstest[,-19])
(SVM_testtable = table(Actual= maxmin_Pitcherstest$ERA, Prediction = SVM_pred))
```

```
##                       Prediction
## Actual                 Strong Pitcher Fairly Good Pitcher Weak Pitcher
##   Strong Pitcher                  188                   2            0
##   Fairly Good Pitcher              87                 125            0
##   Weak Pitcher                      0                  22            1
##   Pitcher in Trouble                0                   0            6
##                       Prediction
## Actual                 Pitcher in Trouble
##   Strong Pitcher                       0
##   Fairly Good Pitcher                  0
##   Weak Pitcher                         0
##   Pitcher in Trouble                   0
```

```
error = 1-sum(diag(SVM_testtable))/sum(SVM_testtable)
accuracy = sum(diag(SVM_testtable))/sum(SVM_testtable)
print(round(cbind(Testing_Accuracy = accuracy, Testing_Error = error),2))
```

```
##      Testing_Accuracy Testing_Error
## [1,]             0.73          0.27
```

Overall, the Support Vector Machine models handles larger data-sets very well despite containing a larger amount features. Although, the Support Vector Machines were able to handle the high-dimensionality of the dataset, it is not to say that it was not prone to over-fitting. That is exactly what presented itself as the model did a superb job with the training data but because it adjusted too much to the training data the testing set accuracy suffered slightly more than one would expect. Through my exploration of the model parameters, I made an indication from the trends I saw that the model would perform best with a moderate number for cost but it must be attached with a low gamma value to counteract the bias-variance takeoff effect. With a large cost value, the model is prone to low bias but high variance where as a small cost values gives a high bias but low variance. The tuning procedure allowed for the best hyper-parameters to emerge; however, it came with the assumption that the intervals given for cost and gamma were optimal. Due to computational power, the range was defined after exploring different combinations because it would be a extensive procedure to have larger ranges for cost and gamma in addition to applying to the number of provided features. Given the intervals for cost and gamma, there were a total of 70 different combinations and then have it applied even further with the features is strenuous on local memory. Nonetheless, the optimal numbers for cost and gamma were not near the end values indicating that the intervals were a proper choice leading to the best if not close to the best possible SVM model for the given dataset.

12 Reflection

Over the course of the report, the main question at hand was to figure out which of the many pitcher statistics were best to describe an Estimated Run Average (ERA) score. The lower the ERA score a pitcher has the better as it pertains to how well they pitch to not allow the opposing team to obtain any runs. In the given report, the question was answered in the scope of a regression case as well as a classification case. The regression case explored Principal Component Analysis with Partial Least Squares Regression, Neural Networks, and Random Forests where it assessed how far off the predicted value from the model was in comparison to the actual values through Mean Squared Error. Comparing the regression models, the one that performed the best was a Neural Network. The design choices of the neural network with the given functional parameters lead to a well higher performance in prediction accuracy given that the update of weights was though a resilient propagation rather than a traditional back-propagation. Before applying the neural network function, I had only known of forward and backward propogation so being introduced to the resilient propagation technique for the first time was rather informative ahd helpful to understand the different approaches made and seeing the benefit they can provide for the model. With the regression portion, I was able to explore a bit out of my comfort zone by implementing Random Forests and exactly what was happening in terms of the procedure. Another thing that I was able to get out from this project was to understand the PCS and PLSR procedure and interpret what it was doing and its motivation for doing so. During the time I was analyzing and reading articles about it, it began to all make sense and the dots were connecting as I started to then implement it.

For the classification approach, I was able to transform the regression question into a classification one by creating intervals of values and constructing them as labels. I would say through this project I was able to familiarize myself more with factor variables and not only its application but more so the importance of why they are constructed/included. That goes with saying, learning about contrast-encoding more and the proedure it does was also insightful. That being said the classification model that performed the best was the Naive Bayes Classification with a 76% accuracy rate. Nonetheless, I would consider the SVM method not as a second runner up but as a co-winner in terms of accuracy performance. The model with its complexity and strong ability to deal with the large and non-linear data was outstanding as it yielded in a 73% testing accuracy. I consider SVM to be powerful as given a better setup, I believe the method would predict better and yield in a better testing accuracy.

Something I would do differently in the classification approach would be to create classes with a balance number of observations. For some classes there were only about 3 observations and the model would have a hard time learning off of only 3 example in comparison to a dataset of 1000 observations. To provide better performance for the classification approach, I would next try to compile the observations in a fair manner or split them up evenly such that each class has a sufficient amount of observations the model can learn from and increasing its prediction power. Lastly, something else I would do would be to incorporate more visuals in the classification problem. The next step for me post this report would be to provide the classification boundaries for kNN and Naive Bayes to format how certain observations would be classified, but I would need to get more familarized with more efficient plotting techniques.

In conclusion, it is safe to assume that the predictors or statistics that are more significant to the fluctuation of a pitcher's ERA are WHIP, AB, ER, R, G, IP, SO.W, and BAbip. The number of games pitched in additional to the number of strikeouts the pitcher has obtained in a game that resulted in a win helps decrease a pitchers ERA. On the other hand, the remaining variables impact the pitchers ERA negatively as it increases their ERA score. Straightforwardly, this makes sense because the qualities of a good pitcher relate to their ability in a large number of strikeouts per game won whereas giving the opposing team hits or runs is not favorable.

13 References

Petti, Bill. “Baseballr.” Root Mean Squared Musings, 2020, billpetti.github.io/baseballr/.

“Earned Run Average.” Baseball Wiki, baseball.fandom.com/wiki/Earned_run_average.

Dasaradh, S K. “A Gentle Introduction To Math Behind Neural Networks.” Medium, Towards Data Science, 30 Oct. 2020, towardsdatascience.com/introduction-to-math-behind-neural-networks-e8b60dbbdeba.

Alice, Michy. “Fitting a Neural Network in R; Neuralnet Package.” R, 23 Sept. 2015, www.r-bloggers.com/2015/09/fitting-a-neural-network-in-r-neuralnet-package/.

Alto, Valentina. “Neural Networks: Parameters, Hyperparameters and Optimization Strategies.” Medium, Towards Data Science, 6 July 2019, towardsdatascience.com/neural-networks-parameters-hyperparameters-and-optimization-strategies-3f0842fac0a5.

Zhang, Zhongheng. “Naïve Bayes Classification in R.” Annals of Translational Medicine, AME Publishing Company, June 2016, www.ncbi.nlm.nih.gov/pmc/articles/PMC4930525/.

Analytics, Perceptive. “Understanding Naïve Bayes Classifier Using R.” R, 22 Jan. 2018, www.r-bloggers.com/2018/01/understanding-naive-bayes-classifier-using-r/.

Garg, Rohit. “7 Types of Classification Algorithms.” Analytics India Magazine, 17 Oct. 2020, analyticsindiamag.com/7-types-classification-algorithms/.

Dasaradh, S K. “A Gentle Introduction To Math Behind Neural Networks.” Medium, Towards Data Science, 30 Oct. 2020, towardsdatascience.com/introduction-to-math-behind-neural-networks-e8b60dbbdeba.

Miller, Max. “The Basics: KNN for Classification and Regression.” Medium, Towards Data Science, 18 Oct. 2019, towardsdatascience.com/the-basics-knn-for-classification-and-regression-c1e8a6c955.

Libretexts. “1: Support Vector Machine (SVM).” Statistics LibreTexts, Libretexts, 17 Aug. 2020, stats.libretexts.org/Bookshelves/Computing_and_Modeling/RTG:_Classification_Methods/1:*Support_Vector_Machine*(SVM).

Libretexts. “3: K-Nearest Neighbors (KNN).” Statistics LibreTexts, Libretexts, 17 Aug. 2020, stats.libretexts.org/Bookshelves/Computing_and_Modeling/RTG:_Classification_Methods/3:*K-Nearest_Neighbors*(KNN).

Lenten, Scott Van. “Predicting 2015 Starting Pitcher Performance Using Regression Trees.” Community Blog, community.fangraphs.com/predicting-2015-starting-pitcher-performance-using-regression-trees/.

“Dodgers Stats.” MLB.com, www.mlb.com/dodgers/stats/.

Hagen, Alex. “Using Linear Regression to Predict a Pitcher’s Performance.” CCG Blog, blog.ccganalytics.com/using-linear-regression-pitchers-performance.

Lenten, Scott Van. “Predicting 2015 Starting Pitcher Performance Using Regression Trees.” Community Blog, community.fangraphs.com/predicting-2015-starting-pitcher-performance-using-regression-trees/.

“What Does It Take to Predict MLB Starting Pitcher Performance Accurately?” NYC Data Science Academy, nycdatascience.com/blog/student-works/take-predict-mlb-starting-pitcher-performance-accurately/.

Jackson, Matt. “What Drives the Difference between ERA and FIP?” Beyond the Box Score, Beyond the Box Score, 28 July 2015, www.beyondtheboxscore.com/2015/7/28/9040603/era-minus-fip-regression-starting-pitcher.

“(Tutorial) NEURAL NETWORK Models in R.” DataCamp Community, www.datacamp.com/community/tutorials/neural-network-models-r.

“Support Vector Machines in R.” DataCamp Community, www.datacamp.com/community/tutorials/support-vector-machines-r.

Dalpiaz, David. “R For Statistical Learning.” Chapter 7 k-Nearest Neighbors, 28 Oct. 2020, daviddalpiaz.github.io/r4sl/knn-reg.html.

James, Gareth, et al. An Introduction to Statistical Learning with Applications in R. Springer, 2017.

“Earned Run Average.” Baseball Wiki, baseball.fandom.com/wiki/Earned_run_average.