Final Project: Preditcing MLB Pitcher ERA Using Three Different Learning Methods

Zach Bishof

May 13, 2020

Goal

The intent of this study was to determine whch of three different learning methods is best at predicting the ERA of MLB pitchers after the Steroid Era, and then to build a model using that learning method. It should be noted that the goal of a pitcher is to have the lowest ERA possible. The three different methods used were Multiple Linear Regression (MLR), Principal Component Regression (PCR), and Neural Network (NN).

The three methods are compared, using 10-fold cross validation (CV), and the best one is fitted to the data to create a model.

Data

The data used comes from the Sean Lahman database at http://www.seanlahman.com/baseball-archive/statistics/. It consists of every single MLB pitcher since the beginning of records dating back to 1871. The info about each pitcher is the typical statistics one would find on the back of a baseball card. This info we interpret as covariates, and their description is found on Lahman's website. The descriptions are: playerID - Player ID code, yearID - Year, teamID - Team, lgID - League, W - Wins, L - Losses, G - Games, GS - Games Started, CG - Complete Games, SHO - Shutouts, SV - Saves, IPouts - Outs Pitched (innings pitched x 3), H - Hits, ER - Earned Runs, HR - Homeruns, BB - Walks, SO - Strikeouts, BAOpp - Opponents' batting average, ERA -Earned Run Average, IBB - Intentional Walks, WP - Wild Pitches, HBP - Batters Hit By Pitch, BK - Balks, BFP - Batters faced by Pitcher, GF - Games Finished, R - Runs Allowed, SH - Sacrifice Hits allowed, SF - Sacrifice Flies allowed, GIDP - Grounded into Double Plays

```
library(dplyr)
## Read Pitching Data from Lahman Database
library(readxl)
Pitchingraw <- read_excel("C:/Users/user/Documents/R/Advanced Stat Learning/Final Project/Pitching.xlsx
```

Cleaning the Data

Although this dataset is thourough, it is a bit too comprehensive for our purposes. For one, MLB baseball has changed a lot over the years. In particular, we will be concerned with the game after the Steroid Era, which means that we only retain stats on players from 2010 onward.

Also, it was necessary to remove the variable ER, because the calculation for ERA is merely ER divided by innings pitched. This means that ER would cause a trivial model when used in conjunction with IPouts.

It became necessary to remove data points for pitchers who have rarely appeared. If a pitcher appears in fewer than ten games or fewer than 27 outs (the number of outs in an entire game), then that pitcher might not really be the best example of a major league pitcher. Often times teams bring up pitchers from the minor leagues, for a single outing, due to a vacancy in the roster caused by an injury. These pitchers tend to have

bizarre stats. For example, one pitcher had an ERA of over 100, but only pitched a single game. This only happens in rare cases, and is not of concern for this report.

The last thing that was necessary for data cleaning was to remove observations with NaN or NA. This was done for similar reasons as removing pitchers with few appearances, as it gives observations with too little information. Thankfully, there were very few observations with this problem.

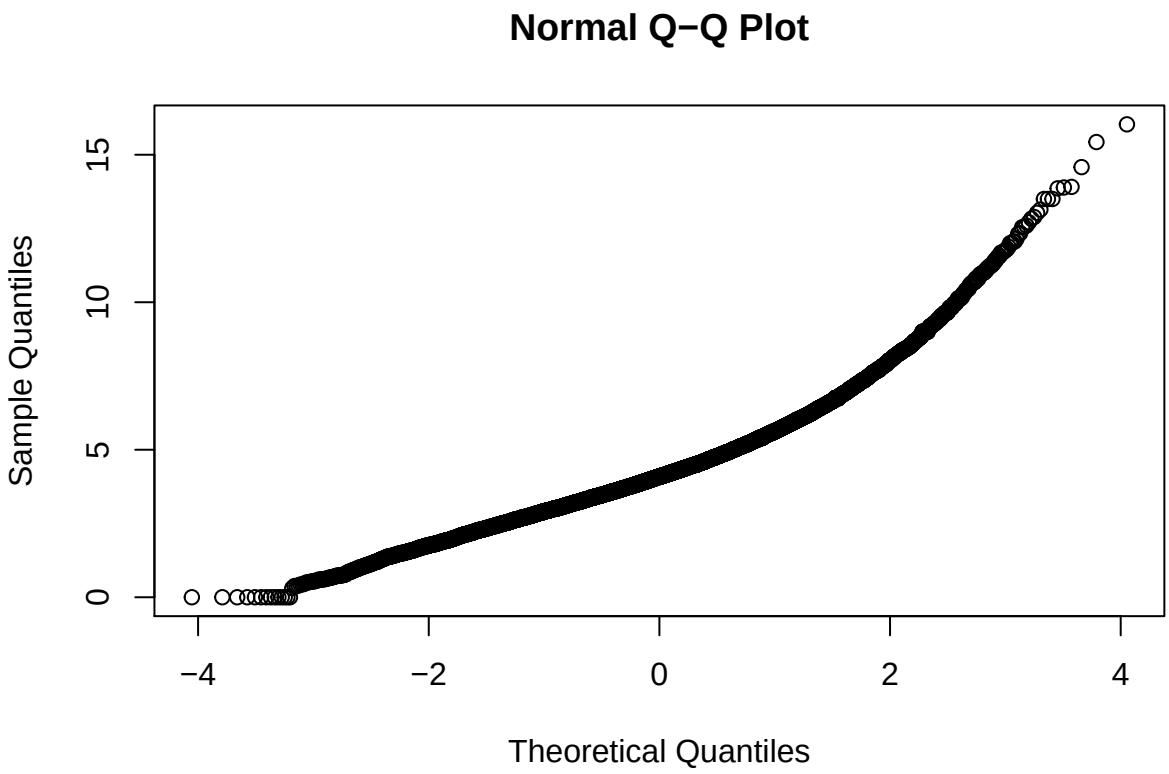
```
## Removing Categorical Variables, ER (or else
## model becomes trivial), and only including data
## from after Steroid Era (2010 onward)
Pitching <- filter(Pitchingraw, yearID >=2010)
Pitching <- Pitchingraw[-c(1:5, 15)]


## Omit NA
Pitching <-na.omit(Pitching)

## Remove observations with Pitchers pitching
## fewer than 10 games or fewer than 27 outs.
Pitching <- filter(Pitching, G>=10)
Pitching <- filter(Pitching, IPouts >=27)
```

Normality

Before we can try any of the learning methods, we check if the response ERA is normally distributed. This is a necessary assumption in a wide variety of learning methods, including those within this report.

We construct a qq-plot of the response ERA, and we can see that the plots do not form quite as straight of a line as one would like, however good enough to continue. We will persist with the assumption of normality in response ERA.

```
qqnorm(Pitching$ERA)
```



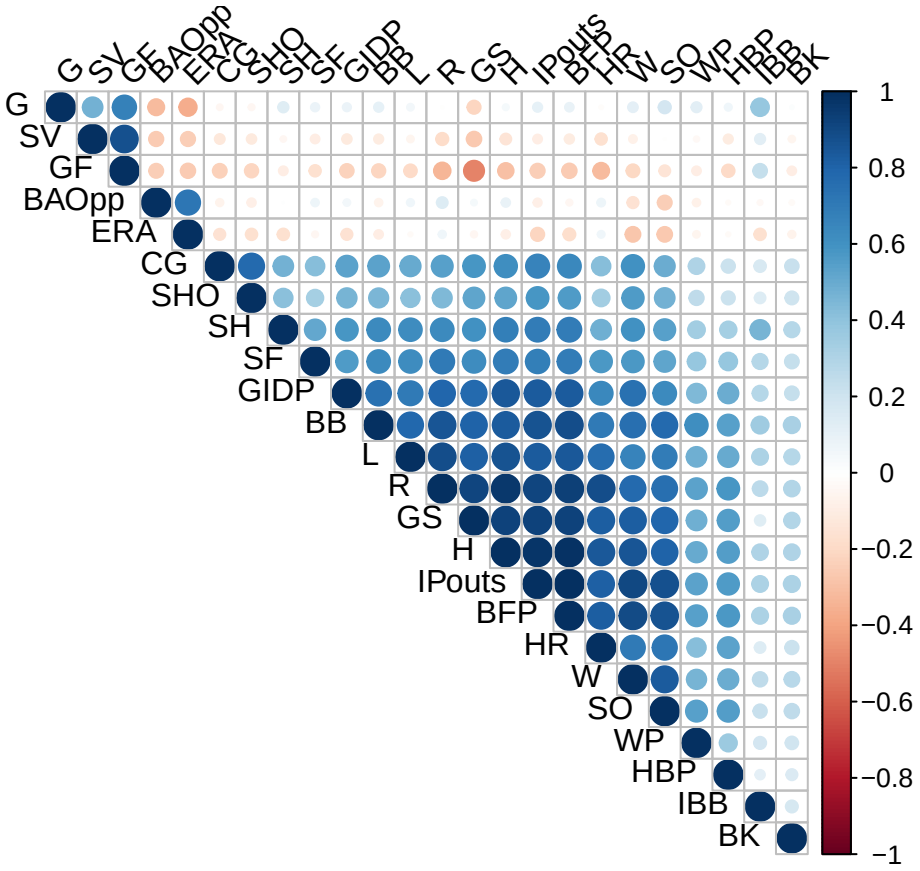
Correlation Matrix

We can take a quick look at the data by examining the correlation matrix. The chart below shows the correlation between covariates. Blue or red corresponds to correlation positive or negative respectively, with the darker the color the stronger the correlation.

We can see that there are a good number of covariates that are correlated. For any baseball fan, this should be entirely obvious for some of these stats. For example, it should be clear that the number of hits a pitcher allows is correlated with the number of runs they allow.

We will address these correlations at the end of the report when we discuss multicolinearity.

```
library(Hmisc)
library(corrplot)
res <- cor(Pitching)
corrplot(res, type = "upper", order = "hclust",
         tl.col = "black", tl.srt = 45)
```



Normalizing the Data

For the purpose of the neural network, the data was normalized. This meant that the normalized data would be used with MLR and PCR models as well, but only for the purpose of comparing the three models doing CV.

As we will see later in the report, MLR will be the learning technique for which we will build a model. When this happens, we will use the regular non-normalized data.

```
## Normalize Data called scaled
mean <- apply(Pitching, 2, mean)
std <- apply(Pitching, 2, sd)

scaled <- as.data.frame(scale(Pitching, center = mean, scale = std))
```

Methods

Model 1: Multiple Linear Regression (MLR)

MLR is the most commonly taught and used learning method for data like this. Models built with MLR have the advantage of being highly interpretable, and don't use as much computing power as something like NN.

The four main assumption of MLR are:

1) Linearity - relationship between predictors and response is linear
2) Homoscedasticity - constant variance
3) Independence - observations are independent

4) Normality - Y is normally distributed

We will discuss these assumptions in greater detail in the Results section of this report.

MLR Briefly Explained

The concept of MLR is to fit a line to our data. If our data has n observations and p predictors, then we try to fit a line with p terms. This is called the full model. We attempt to estimate the coefficients on each term (called parameter estimates) by choosing those that minimize a mean squared error calculation. This is known as the method of least squares. Essentially, we are looking for the line that is closest to all the data points.

The full model will look something like $\hat{y} = \hat{\beta}_0 + \sum_{i=1}^{p} \hat{\beta}_i X_i$

And parameter estimates $\hat{\beta}_i$ are those which minimize the function

$MSE(X) = \frac{1}{n} \sum_{i=1}^{n} (\hat{y}_i - y_i)^2$

10-fold Cross Validation (CV)

The concept of 10-fold cross validation (CV) is a very useful one when comparing learning methods. Essentially, the data is broken up into 10 folds of approximately equal size. We take the first fold and set it aside, and train the data on the remaining nine folds. The first fold is then used to validate the model, meaning it is used as a test set to the other nine folds as a training set. We repeat this process ten times so that each fold gets a chance to be the validation set.

In the following r chunk, we can see that we have done 10-fold CV using MLR as our model for which we build upon. The nine folds are trained using MLR, and then validated using the remaing fold. The RMSE is reported by R, and all we need to do the get MSE is to square RMSE.

```
library(caret)
set.seed(666)

## Divide data into folds
train.control <- trainControl(method = "cv", number = 10)
# Train the model
model.lm <- train(ERA~., data = scaled, method = "lm", trControl = train.control)
# Summarize the results
print(model.lm)
```

```
## Linear Regression
##
## 19877 samples
##    23 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 17890, 17889, 17890, 17889, 17890, 17888, ...
## Resampling results:
##
##   RMSE       Rsquared   MAE
##   0.4912571  0.7587396  0.3311496
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

Model 2: Principal Component Analysis (PCA) Regression

PCA is a very interesting concept with machine learning. It was the opinion of the author of this report that PCA might work best for the Pitching data. The reason why this might have been the case is that PCA does

a good job in dealing with correlated data. This guess did not turn out to be true, which will be shown later in the report.

The assumptions of PCA are:

1) Variables are continuous
2) Variables have linear relationship
3) Large sample
4) Adequate correlations between variables
5) No outliers

Going into the report, it was the opinion of the author that these assumptions were sufficiently valid. After completing the analysis, it is likely that the fourth assumption was not entirely true. Several of the variables are correlated, but perhaps not enough.

PCA Regression Explained

PCA is a technique where the data is broken into principal components that represent the highest variability in the data. With MLR, we think of each predictor as an axis in p-dimensional space. We then fit the data by drawing a line as described by a linear combination of the predictor variables. In PCA, we are not constricted to these axes. In PCA, we find the direction that the data has the most variability, and then call that our new axis, or first principal component (PC1). We then find the orthogonal direction from PC1 with the most variability and that is PC2. This process is repeated over and over again.

In PCR, we can now build our model using MLR, except now we look at our model in terms of the principal components rather than the predictor variables. It should be noted that each principal component can be understood as a vector that can be represented in terms of the predictors.

This process often times has the desirable effect of reducing the complexity of the model. Typically, a model can be reduced to only a few principal components, which means that a highly flexible model can be turned into a less flexible model. This has the added effect of finding the sweet spot on the bias-variance tradeoff.

10-fold Cross Validation

CV for PCA regression is very much similar to that of CV for MLR. We split the data into folds, train on nine folds, validate on one, repeat ten times. The only difference is that now we are using PCA to train the data.

```
set.seed(667)
train.control = trainControl(method = "cv", number=10)
PCA.lm <- train(ERA~., scaled, method = "lm", preProcess=c("pca"), trControl = train.control)

print(PCA.lm)
```

```
## Linear Regression
##
## 19877 samples
##    23 predictor
##
## Pre-processing: principal component signal extraction (23), centered
##  (23), scaled (23)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 17889, 17889, 17890, 17889, 17889, 17889, ...
## Resampling results:
##
##   RMSE       Rsquared   MAE
##   0.6054527  0.6333757  0.4344173
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

Model 3: Neural Network (NN)

The nice thing about neural networks is that we do not need any assumptions about the data. This makes neural networks particularly advantageous with data that is neither normal, linear, nor of constant variance. A neural network is a learning technique that can be understood as somewhat analgous to how our own brains work.

Neural Network Briefly Explained

A neural network of one layer for a regression problem is very similar to MLR. When we start dealing with multiple layers is where it gets a bit more complicated.

We have p+1 nodes in the hidden layer, where there are the p predictors and a bias term (similar to the intercept in the context of MLR), and each node is given K weights. K represents the number of nodes in the first layer. The weights can be thought of as parameter estimates, and are calculated using the same cost function as MLR (i.e. least squares). These weights are then inputed into an activation function, which in the context of regression is just the identity function. This gives us a representation of each node, excluding the bias node, in the first hidden layer.

This process is repeated for however many layers is specified by the NN model in question. The second layer is computed by assigning weights to each node in the first layer equal to the number of nodes in the second layer. These weights are computed much like parameter estimates, and then they are sent through an activation function.

This process repeats until we go through all layers and we are left with the final layer, which gives us predictions on the response.

10-Fold CV

CV for NN is similar as to what has been discussed. The main difference now is that we train the data using NN, and validate using the model that arises from training the nine folds using NN.

```
## This chunk and the next chunk were kept seperate for ease of use.

set.seed(668)

train.control <- trainControl(method = "cv", number = 10)
model.nn <- train(ERA ~., data = scaled, method = "nnet", trControl = train.control)
```

```
## Print Model
print(model.nn)
```

```
## Neural Network
##
## 19877 samples
##    23 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 17889, 17889, 17890, 17890, 17889, 17890, ...
## Resampling results across tuning parameters:
##
##   size  decay  RMSE       Rsquared   MAE
##   1     0e+00  0.9996454        NaN  0.7509887
##   1     1e-04  0.8983566  0.2975109  0.6718973
##   1     1e-01  0.7694090  0.6291402  0.5567747
##   3     0e+00  0.9996454        NaN  0.7509887
##   3     1e-04  0.9591739  0.1357131  0.7179453
```

```
##   3     1e-01  0.7647344  0.6766002  0.5438061
##   5     0e+00  0.9996454        NaN  0.7509887
##   5     1e-04  0.9053555  0.3111945  0.6693512
##   5     1e-01  0.7635641  0.6872216  0.5408928
##
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were size = 5 and decay = 0.1.
```

Results

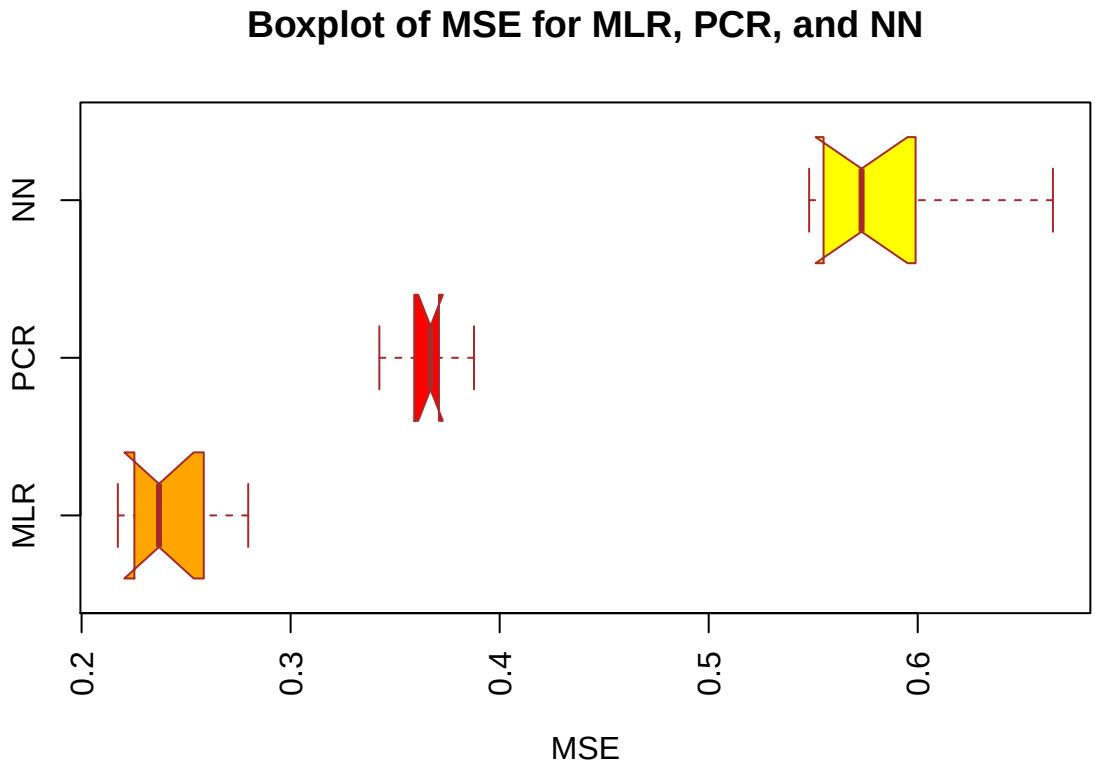
Boxplots and t-Test

Each RMSE value was stored for each fold of 10-fold CV for each learning method. This will allow us to compare the strength of the methods. We can square RMSE and obtain MSE, then plot the MSE for each fold in a boxplot. This is displayed below.

We can see that MLR seems to have lower values for MSE across each fold. In fact, the largest MSE for MLR was still lower than all of the values for MSE for PCA and NN. This implies that MLR is the better method for this data set.

However, we cannot simply look at the boxplots. It is important to go ahead and do a t-test to see if the means of each set of MSE are the same. Upon doing so we can see that the p-values for MLR compared with NN and MLR compared wtih PCA are very low. This implies that it is safe to assume that the MSE found from MLR will be lower than that of the other two methods.

```
## Boxplots
boxplot((model.lm$resample$RMSE)^2, (PCA.lm$resample$RMSE)^2, (model.nn$resample$RMSE)^2,
main = "Boxplot of MSE for MLR, PCR, and NN",
xlab= "MSE",
at = c(1,2,3),
names = c("MLR", "PCR", "NN"),
las = 3,
col = c("orange","red", "yellow"),
border = "brown",
horizontal = TRUE,
notch = TRUE
)
```




```
## Check if mean values for MSE are different using t-test
t.test((model.lm$resample$RMSE)^2, model.nn$resample$RMSE^2)
```

```
##
##  Welch Two Sample t-test
##
## data:  (model.lm$resample$RMSE)^2 and model.nn$resample$RMSE^2
## t = -26.118, df = 14.621, p-value = 1.117e-13
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.3697201 -0.3138123
## sample estimates:
## mean of x mean of y
## 0.2417374 0.5835036
```

```
t.test(model.lm$resample$RMSE^2, PCA.lm$resample$RMSE^2)
```

```
##
##  Welch Two Sample t-test
##
## data:  model.lm$resample$RMSE^2 and PCA.lm$resample$RMSE^2
## t = -15.596, df = 15.676, p-value = 5.8e-11
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.1419706 -0.1079430
## sample estimates:
## mean of x mean of y
```

```
## 0.2417374 0.3666943
```

```
t.test(PCA.lm$resample$RMSE^2, model.nn$resample$RMSE^2)
```

```
##
##  Welch Two Sample t-test
##
## data:  PCA.lm$resample$RMSE^2 and model.nn$resample$RMSE^2
## t = -17.912, df = 11.736, p-value = 6.948e-10
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.2432479 -0.1903709
## sample estimates:
## mean of x mean of y
## 0.3666943 0.5835036
```

Build Model using MLR on Full Pitching Data Set

It is now time to build a model using the entire Pitching data set using MLR. To do this we can use a variety of variable selection techniques.

In this report we will use backward selection, then we will eliminate variables that experience colinearity by checking VIF scores, then finally eliminate variables with insignificant parameter estimates by checking p-values.

Backward Selection

Backward selection is a useful technique for turning an MLR model with a lot of predictors into something less complex.

The concept here is to start with the full model, and then remove one predictor and run the model again, then repeat this for each predictor until we have ran the full model, and p other models with p-1 predictors. We use each of these models and calculate their AIC value, which essentially measure their quality of the model compared to the other models. The model with the highest AIC tells us which variable to remove. This process is repeated until the model without any variables removed has the highest AIC.

We did backward regression on the full model and obtained a model that looks like: ERA ~ W + L + G + GS + CG + SHO + SV + IPouts + H + HR + BB + SO + BAOpp + IBB + BK + BFP + GF + R + SH + SF + GIDP

Note: The output of the code below was left out of the report for simplicity.

```
# Backward Regression
library(MASS)
fit <- lm(ERA~. , data=Pitching, family = "binomial")
backward <- stepAIC(fit, direction="backward")
backward$anova # display results
```

Multicolinearity Check

Now we check for multicolinearity. To do this, we take the model that was selected from backward regression, and then check the VIF scores. A VIF score of above 10 is considered problematic. Each time a model was checked, the covariate with the highest VIF score was removed. Then the VIF scores were checked on the new model. The process was repeated until not a single covariate had a VIF score above 10.

After doing this process, we are left with the following model: ERA ~ W + L + G + CG + SHO + SV+ HR + BB + SO + BAOpp + IBB + BK + SH + SF + GIDP

Note: As before, the output of the r chunk below was left out of the report for simplicity.

```
library(car)
vif(lm(ERA ~ W + L + G + GS + CG + SHO + SV + IPouts + H + HR + BB +
    SO + BAOpp + IBB + BK + BFP + GF + R + SH + SF + GIDP, Pitching))
## Remove BFP
vif(lm(ERA ~ W + L + G + GS + CG + SHO + SV + IPouts + H + HR + BB +
    SO + BAOpp + IBB + BK + GF + R + SH + SF + GIDP, Pitching))
## Remove IPouts
vif(lm(ERA ~ W + L + G + GS + CG + SHO + SV + H + HR + BB +
    SO + BAOpp + IBB + BK + GF + R + SH + SF + GIDP, Pitching))
## Remove H
vif(lm(ERA ~ W + L + G + GS + CG + SHO + SV+ HR + BB +
    SO + BAOpp + IBB + BK + GF + R + SH + SF + GIDP, Pitching))
## Remove GS
vif(lm(ERA ~ W + L + G + CG + SHO + SV+ HR + BB +
    SO + BAOpp + IBB + BK + GF + R + SH + SF + GIDP, Pitching))
## Remove R
vif(lm(ERA ~ W + L + G + CG + SHO + SV+ HR + BB +
    SO + BAOpp + IBB + BK + GF + SH + SF + GIDP, Pitching))
## Remove GF
vif(lm(ERA ~ W + L + G + CG + SHO + SV+ HR + BB +
    SO + BAOpp + IBB + BK + SH + SF + GIDP, Pitching))
```

Cutting out Insignificant Covariates

Now we check the remaining model for insiginificant covariates. A covariate is considered insiginificant if it has parameter estimate with a p-value of higher than .05. This p-value tests the null hypothesis that this parameter is insignificant in the model. This means that high p-values imply that the covariate in question is not needed in the model.

So the process that follows is we check the model given from the multicolinearity section, check the p-values, and remove the covariate with the highest p-value above .05. This process is repeated until not a single covariate has a parameter estimate with a p-value above .05.

This gives us our final model.

Note: The output of this R chunk was left out of the report for simplicity.

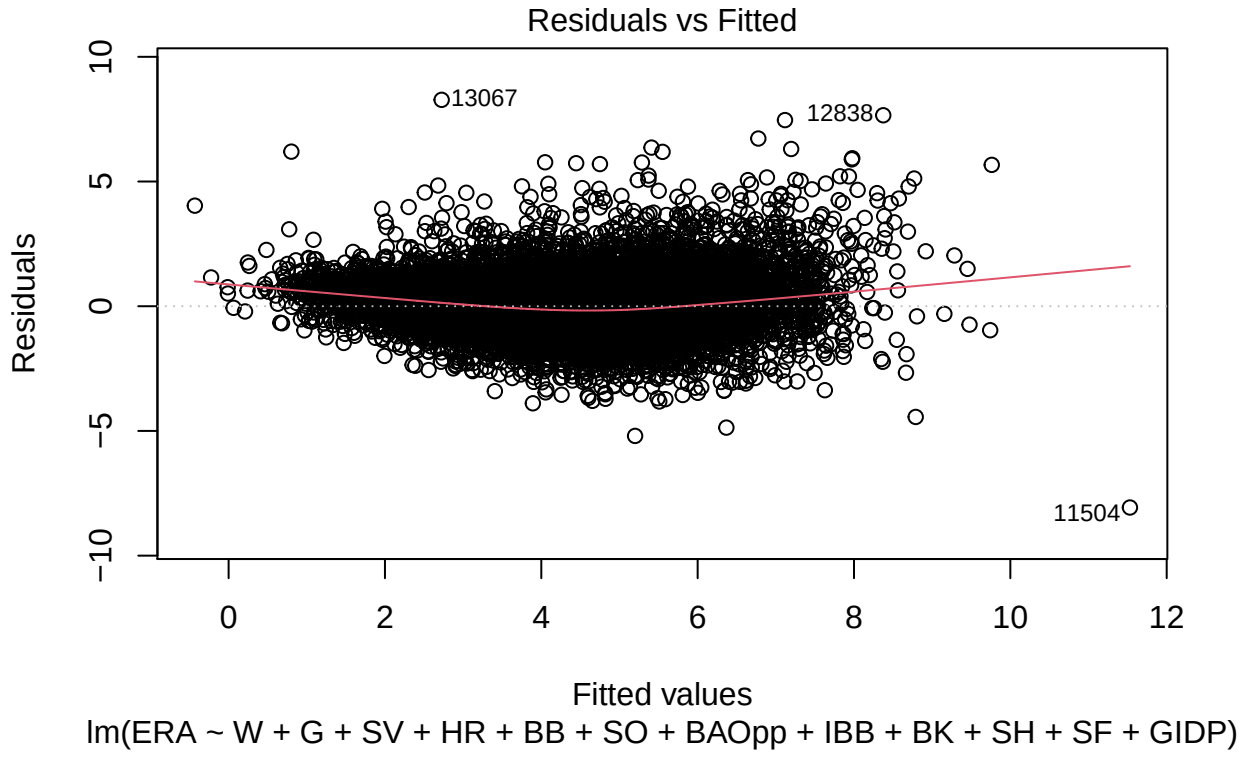
```
## P-value Check

summary(lm(ERA ~ W + L + G + CG + SHO + SV+ HR + BB +
    SO + BAOpp + IBB + BK + SH + SF + GIDP, Pitching))

## Remove L due to high p-value
summary(lm(ERA ~ W + G + CG + SHO + SV+ HR + BB +
    SO + BAOpp + IBB + BK + SH + SF + GIDP, Pitching))
## Remove CG due to high p-value
summary(lm(ERA ~ W + G + SHO + SV+ HR + BB +
    SO + BAOpp + IBB + BK + SH + SF + GIDP, Pitching))

## Remove SHO due to high p-value

summary(lm(ERA ~ W + G + SV+ HR + BB +
    SO + BAOpp + IBB + BK + SH + SF + GIDP, Pitching))

## No more insignificant covariates, as evidence
```

```
## from p-values associated with parameter
## estimates.
## We keep this as our final model
```

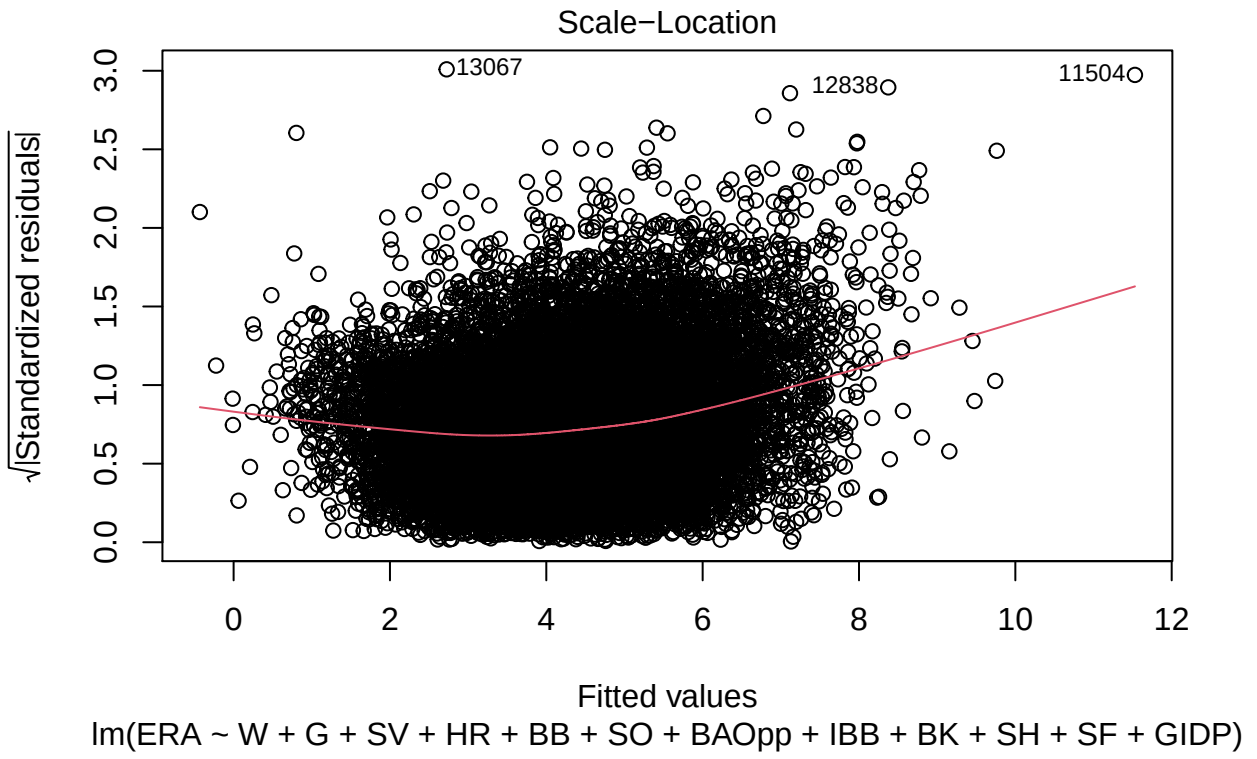
Residual Plots

Lastly, we check residual plots. This is important for the purpose of validating model assumptions. First we check the Residual vs. Fitted graph and see that there is no clear pattern, which is ideal. Then we check the Scale-Location graph, which helps us determine if our residuals have constant variance. If they are evenly spread out, then we can say that this assumption holds true. In our case, the Scale-Location graph is not perfect, but good enough for our purposes. Then we check the QQ-Plot of the residuals, which helps us determine if the residuals are normally dsitributed. We want this to look like a straight line, which it does not do perfectly, but well enough that we can move on with this model. Lastly, we look at the Residuals vs. Leverage graph, which can help us determine if there are any outliers. Because there are no plots outside of Cook's distance, we can safely say there are no outliers.

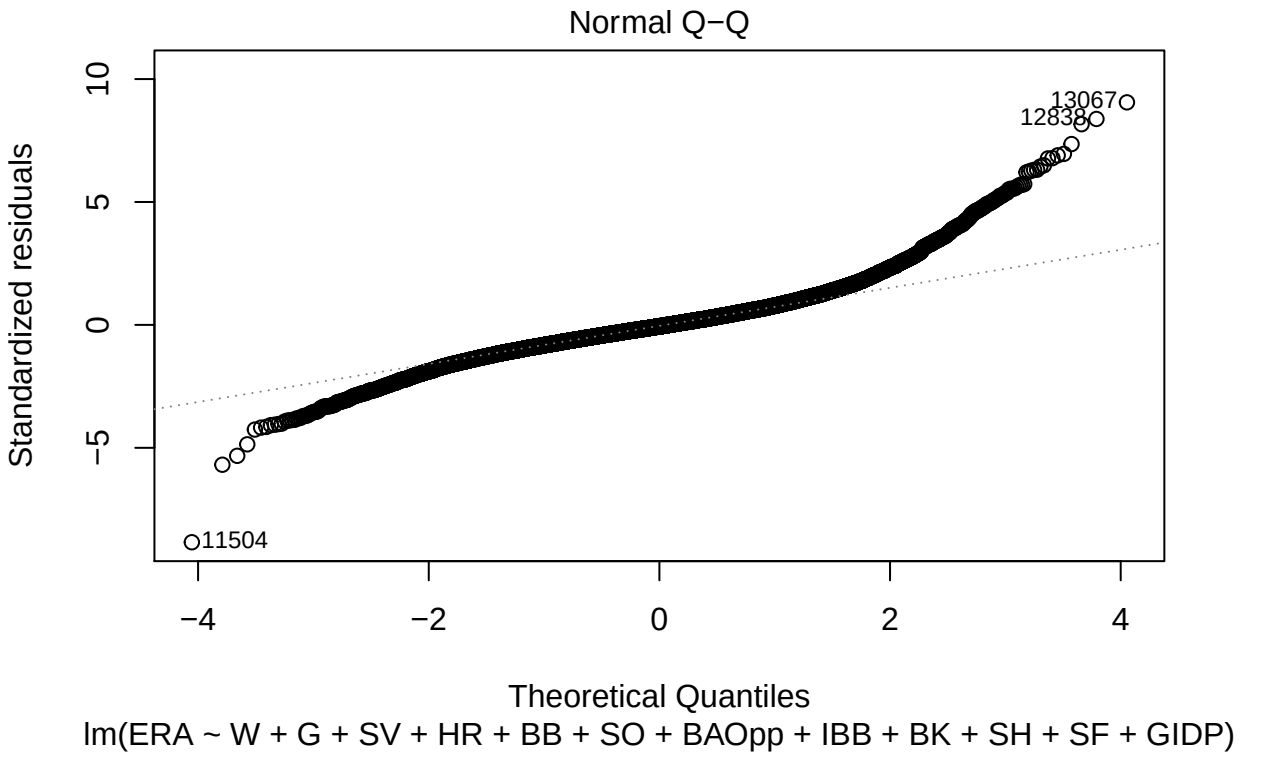
```
finally<-lm(ERA ~ W + G + SV+ HR + BB +
    SO + BAOpp + IBB + BK + SH + SF + GIDP, Pitching)
plot(finally, which =1, data=Pitching)
```



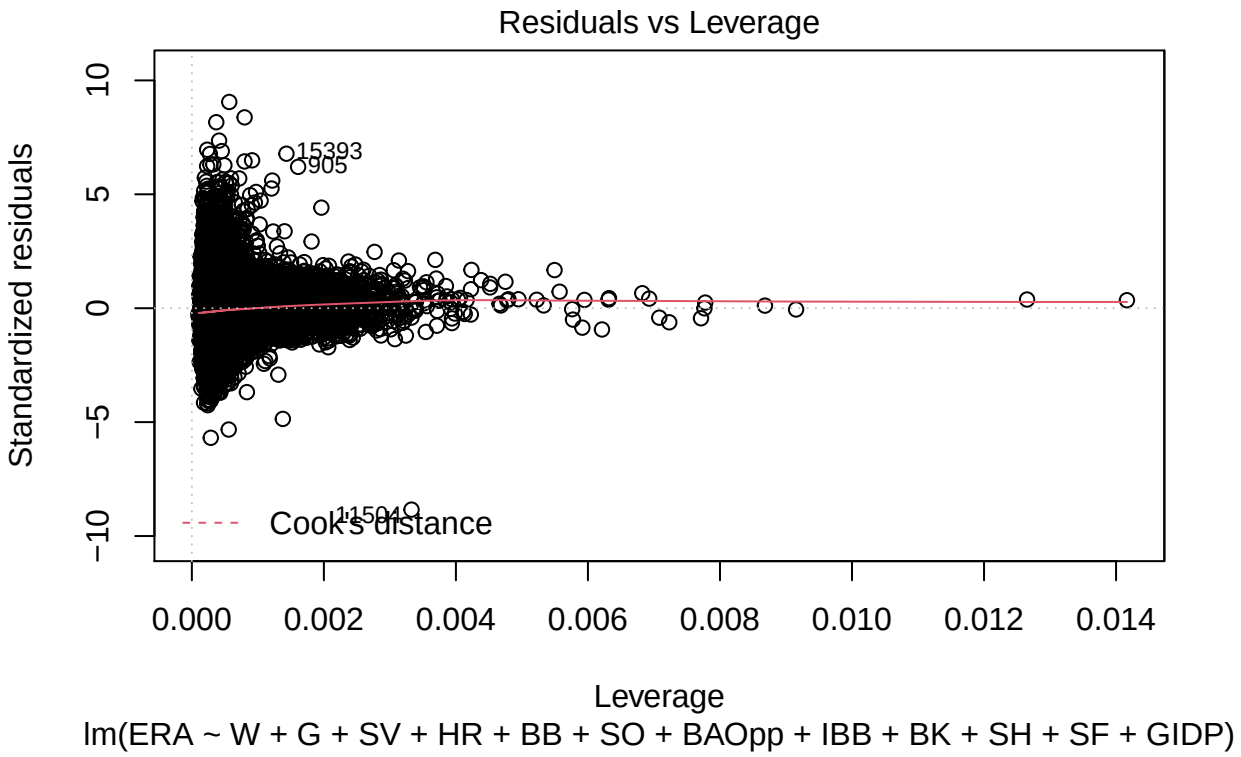

```
plot(finally, which =3, data=Pitching)
```




```
plot(finally, which =2)
```




```
plot(finally, which =5, data=Pitching)
```



Conclulsion

It turns out that according to the MSE values obtained from 10-fold cross validation, the best learning method at fitting the Pitching data was in fact our old trusted MLR. This is clear based on the boxplots and t-tests in the Results section.

The model that is eventually built using MLR, and variable selection is below. The way one could interpret this is fairly simple. Each coefficient in front of a variable tells you how much we expect the ERA of a pitcher to change with an increase in that variable by one. For example, if a pitcher has one more strikeout (SO), then we should expect their ERA to decrease by 0.002869.

The natural question is to ask how good is this model. Well, unfortunately it is not as good as we would like. We obtained an R-Squared value of 0.64, and approximately the same adjusted R-Squared. This means that our model only accounts for about 64 percent of variability in the data. Our model is decent at predicting ERA, but it is the opinion of the author of this report that there are more factors at play not included in this model.

$E\hat{R}A = -1.899 + (-0.05783) * W + (-0.006827) * G + (-0.004637) * SV + (0.04773) * HR + (0.02241) * BB + (-0.002869) * SO + (25.29) * BAOpp + (-0.03037) * IBB + (-0.02471) * BK + (-0.04622) * SH + (-0.03003) * SF + (-0.06755) * GIDP$

```
summary(finally)
```

```
##
## Call:
## lm(formula = ERA ~ W + G + SV + HR + BB + SO + BAOpp + IBB +
##     BK + SH + SF + GIDP, data = Pitching)
```

```
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -8.0699 -0.5134 -0.0542  0.4412  8.2750
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -1.8994521  0.0558569 -34.006  < 2e-16 ***
## W           -0.0578254  0.0029281 -19.749  < 2e-16 ***
## G           -0.0068265  0.0004687 -14.565  < 2e-16 ***
## SV          -0.0046369  0.0010655  -4.352 1.36e-05 ***
## HR           0.0477274  0.0014632  32.619  < 2e-16 ***
## BB           0.0224105  0.0005925  37.822  < 2e-16 ***
## SO          -0.0028685  0.0002915  -9.841  < 2e-16 ***
## BAOpp       25.2857432  0.1924418 131.394  < 2e-16 ***
## IBB         -0.0303671  0.0030609  -9.921  < 2e-16 ***
## BK          -0.0247112  0.0070890  -3.486 0.000492 ***
## SH          -0.0462293  0.0028108 -16.447  < 2e-16 ***
## SF          -0.0300301  0.0036761  -8.169 3.29e-16 ***
## GIDP        -0.0675476  0.0017248 -39.163  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9143 on 19864 degrees of freedom
## Multiple R-squared:  0.6435, Adjusted R-squared:  0.6433
## F-statistic:  2989 on 12 and 19864 DF,  p-value: < 2.2e-16
```

Additions To Final Model If Given More Time

It would be interesting to look into some of the normality issues. For example, the residuals were not as normal as one would like, nor was the response ERA. It likely has to do with the difference in the roles of pitchers. Some relief pitchers have an extremely low ERA, mostly because they are only asked to get a single out per game, while often starting pitchers have to pitch the majority of the game, and thus allow more runs when they begin to get tired. Perhaps a solution would be to add a binary covariate for whether the pitcher was a starter or a relief pitcher. Further, it would be advantageous to add interaction terms with such a binary covariate.

Another thing worth investigating is how well this model would perform on post-season MLB pitchers. Does the post-season change the importance of BK? Does the model built by MLR on post-season data look similar to our final model?

There are certainly many more questions to ask, but this report may serve as a simple introduction into predicting the performance of MLB pitchers.