Project Report - Optimizing Likelihood of MLB Playoff Apperance

Shannon Leiss

Introduction

Baseball is a game with many different variables and factors that contribute to a teams success that can be broadly grouped into offensive ability and defensive ability. Since the introduction of Sabermetrics and famous prediction formulas - made popular by Moneyball - many different attempts have been made to optimize a teams performance using statistical modeling. With the abundance of data available, dating back to the 1871 season, many different aspects of the game are able to be explore in depth. The `Lahman` package in R provides many different historical Baseball data sets, including the `Teams` data set. The `Teams` data set has each teams overall performance in a wide variety of variables per season, along with information on playoff appearances, ball park factors, and attendance totals.

One main area that analysts have tried to predict is a teams likelihood of making the playoffs. With the amount of data available, models can easily get over complicated and over fitted. Using the `Teams` data set for the seasons of 1998 - 2019, since 1998 was the last season additional teams were added to the major leagues and 2020 was a shortened season, offensive statistics were used to model the likelihood of a team making the playoffs. From in depth comparisons of logistic regression models - which modeled the log odds of a team making the playoffs - it was found that the simplest model that best predicted a teams likelihood of making the playoffs was using a teams run average(Runs scored divided by At Bats per season) and a teams batting average(Hits divided by At Bats) as predictor variables without any interaction between the two.

Methods

The model that will be optimized is the following:

$$log(\frac{p_i}{1-p_i}) = \alpha + \beta_1(R_i) + \beta_2(H_i)$$

Where p_i is the probability that a given team will make the playoffs, R_i is a team's run average for a given season, and H_i is a team's batting average for a given season. The unknown values that will be optimized are $\theta = \begin{pmatrix} \alpha \\ \beta_1 \\ \beta_2 \end{pmatrix}$ From the `Teams` data set, the values of y_i - whether a team made the playoffs or not,R_i, and H_i are known. The probability of a team making the playoffs can be expressed as follows:

$$p_i = \frac{e^{\alpha+\beta_1(R_i)+\beta_2(H_i)}}{1+e^{\alpha+\beta_1(R_i)+\beta_2(H_i)}}$$

To optimize these unknown variables, three different optimizer algorithms will be used - Newton-Raphson, Quasi-Newton - along with using bootstrapping techniques to find the 95% confidence intervals for each element of θ, using the Quasi-Newton optimizer.

Newton-Raphson

The Newton-Raphson optimizer has the following updating formula for finding the Maximum Likelihood Estimator(MLE) of θ:

$$\theta^{(t+1)} = \theta^{(t)} - (\ell''(\theta^{(t)}))^{-1}(\ell'(\theta^{(t)}))$$

Where $\ell''(\theta^{(t)})$ is the double derivative of the log-likelihood of $\theta^{(t)}$ and $\ell'(\theta^{(t)})$ is the derivative of the log-likelihood of $\theta^{(t)}$.

Since whether or not a team made the playoffs(y_i) can only take on the value of 1 or 0, y_i will follow a Bernoulli distribution with p_i equal to the definition provided in the previous section. It follows that:

$$\begin{aligned}
f(y_i) &= p_i^{y_i}(1-p_i)^{1-y_i} \\
f(y_i) &= (\frac{e^{\alpha+\beta_1(R_i)+\beta_2(H_i)}}{1+e^{\alpha+\beta_1(R_i)+\beta_2(H_i)}})^{y_i}(1-(\frac{e^{\alpha+\beta_1(R_i)+\beta_2(H_i)}}{1+e^{\alpha+\beta_1(R_i)+\beta_2(H_i)}}))^{1-y_i} \\
f(y_i) &= (\frac{e^{\alpha+\beta_1(R_i)+\beta_2(H_i)}}{1+e^{\alpha+\beta_1(R_i)+\beta_2(H_i)}})^{y_i}(\frac{1}{1+e^{\alpha+\beta_1(R_i)+\beta_2(H_i)}})^{1-y_i}
\end{aligned}$$

From the above definition, it can be seen that the log-likelihood and its derivatives are:

$$\begin{aligned}
\ell(\theta) &= \sum_{i=1}^{n=660} y_i log(\frac{e^{\alpha+\beta_1(R_i)+\beta_2(H_i)}}{1+e^{\alpha+\beta_1(R_i)+\beta_2(H_i)}}) + (1-y_i)log(\frac{1}{1+e^{\alpha+\beta_1(R_i)+\beta_2(H_i)}}) \\
&= \sum_{i=1}^{n=660} y_i log(e^{\alpha+\beta_1(R_i)+\beta_2(H_i)}) - y_i log(1+e^{\alpha+\beta_1(R_i)+\beta_2(H_i)}) + -log(1+e^{\alpha+\beta_1(R_i)+\beta_2(H_i)}) + y_i log(1+e^{\alpha+\beta_1(R_i)+\beta_2(H_i)}) \\
\ell(\theta) &= \sum_{i=1}^{n=660} y_i(\alpha+\beta_1(R_i)+\beta_2(H_i)) - log(1+e^{\alpha+\beta_1(R_i)+\beta_2(H_i)}) \\
\ell'(\theta) &= \begin{bmatrix} \sum_{i=1}^{n=660} y_i - \frac{e^{\alpha+\beta_1(R_i)+\beta_2(H_i)}}{1+e^{\alpha+\beta_1(R_i)+\beta_2(H_i)}} \\ \sum_{i=1}^{n=660} R_i(y_i) - \frac{(R_i)e^{\alpha+\beta_1(R_i)+\beta_2(H_i)}}{1+e^{\alpha+\beta_1(R_i)+\beta_2(H_i)}} \\ \sum_{i=1}^{n=660} H_i(y_i) - \frac{(H_i)e^{\alpha+\beta_1(R_i)+\beta_2(H_i)}}{1+e^{\alpha+\beta_1(R_i)+\beta_2(H_i)}} \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{n=660} y_i - p_i \\ \sum_{i=1}^{n=660} R_i(y_i) - R_i(p_i) \\ \sum_{i=1}^{n=660} H_i(y_i) - H_i(p_i) \end{bmatrix} \\
\ell''(\theta) &= \begin{bmatrix} \sum_{i=1}^{n=660} -\frac{e^{\alpha+\beta_1(R_i)+\beta_2(H_i)}}{(1+e^{\alpha+\beta_1(R_i)+\beta_2(H_i)})^2} & \sum_{i=1}^{n=660} -R_i\frac{e^{\alpha+\beta_1(R_i)+\beta_2(H_i)}}{(1+e^{\alpha+\beta_1(R_i)+\beta_2(H_i)})^2} & \sum_{i=1}^{n=660} -H_i\frac{e^{\alpha+\beta_1(R_i)+\beta_2(H_i)}}{(1+e^{\alpha+\beta_1(R_i)+\beta_2(H_i)})^2} \\ \sum_{i=1}^{n=660} -R_i\frac{e^{\alpha+\beta_1(R_i)+\beta_2(H_i)}}{(1+e^{\alpha+\beta_1(R_i)+\beta_2(H_i)})^2} & \sum_{i=1}^{n=660} -(R_i^2)\frac{e^{\alpha+\beta_1(R_i)+\beta_2(H_i)}}{(1+e^{\alpha+\beta_1(R_i)+\beta_2(H_i)})^2} & \sum_{i=1}^{n=660} -(R_iH_i)\frac{e^{\alpha+\beta_1(R_i)+\beta_2(H_i)}}{(1+e^{\alpha+\beta_1(R_i)+\beta_2(H_i)})^2} \\ \sum_{i=1}^{n=660} -H_i\frac{e^{\alpha+\beta_1(R_i)+\beta_2(H_i)}}{(1+e^{\alpha+\beta_1(R_i)+\beta_2(H_i)})^2} & \sum_{i=1}^{n=660} -(R_iH_i)\frac{e^{\alpha+\beta_1(R_i)+\beta_2(H_i)}}{(1+e^{\alpha+\beta_1(R_i)+\beta_2(H_i)})^2} & \sum_{i=1}^{n=660} -(H_i^2)\frac{e^{\alpha+\beta_1(R_i)+\beta_2(H_i)}}{(1+e^{\alpha+\beta_1(R_i)+\beta_2(H_i)})^2} \end{bmatrix}
\end{aligned}$$

The values of $\ell'(\theta)$ were found by taking the derivative of $\ell(\theta)$ with respects to α, β_1 and β_2. The values of $\ell''(\theta)$ were found by taking the second derivatives as follows: $\frac{\delta^2\ell(\theta)}{\delta\alpha^2}, \frac{\delta^2\ell(\theta)}{\delta\beta_1^2}, \frac{\delta^2\ell(\theta)}{\delta\beta_2^2}, \frac{\delta^2\ell(\theta)}{\delta\alpha\delta\beta_1}, \frac{\delta^2\ell(\theta)}{\delta\alpha\delta\beta_2}$, and $\frac{\delta^2\ell(\theta)}{\delta\beta_1\delta\beta_2}$. Plugging in these values to the updating formula for the Newton-Raphson method, the following updating method was found for this model:

$$\begin{pmatrix} \alpha^{(t+1)} \\ \beta_1^{(t+1)} \\ \beta_2^{(t+1)} \end{pmatrix} = \begin{pmatrix} \alpha^{(t)} \\ \beta_1^{(t)} \\ \beta_2^{(t)} \end{pmatrix} - (\ell''(\theta^{(t)}))^{-1}(\ell'(\theta^{(t)}))$$

This optimization formula was allowed to update until the difference between $\theta^{(t+1)}$ and $\theta^{(t)}$ was less than 0.00001.

Quasi-Newton

It can be noted that the second derivation of the log-likelihood for θ is complicated and could cause computation time to be very large. An alternative method to the Newton-Raphson method is using

the Newton-Like method of Quasi-Newton optimization. This method is a generalization of the secant condition($\ell'(\theta^{(t+1)}) - \ell'(\theta^{(t)}) = M^{(t+1)}(\theta^{(t+1)} - \theta^{(t)})$), and updates the values of θ through updates for M. Rewriting the secant condition, the updating method can be found to be:

$$M^{(t+1)} = \frac{\ell'(\theta^{(t+1)}) - \ell'(\theta^{(t)})}{\theta^{(t+1)} - \theta^{(t)}}$$

For this optimization, the Hessian approximation will be done using the BFGS method, which has the following updating rule:

$$M^{(t+1)} = M^{(t)} - \frac{(M^{(t)}(\theta^{(t+1)} - \theta^{(t)}))(M^{(t)}(\theta^{(t+1)} - \theta^{(t)}))^T}{(\theta^{(t+1)} - \theta^{(t)})^T M^{(t)}(\theta^{(t+1)} - \theta^{(t)})} + \frac{(\ell'(\theta^{(t+1)}) - \ell'(\theta^{(t)}))(\ell'(\theta^{(t+1)}) - \ell'(\theta^{(t)}))^T}{(\theta^{(t+1)} - \theta^{(t)})^T(\ell'(\theta^{(t+1)}) - \ell'(\theta^{(t)}))}$$

This method was implemented using the `optim()` function.

Bootstraping for Confidence Intervals

Out of the three above optimization methods, the Quasi-Newton method has the fastest computation time - even though the Newton-Raphson method is able to find the MLE in the fewest iterations - and thus was used to create confidence intervals for each parameter(α, β_1, β_2) using bootstrapping methods.

For the season of 1998-2019 in the `Teams` dataset, there are 660 individual observations. To create 95% confidence intervals for the parameters, a new set of 660 observations was created by sampling the `Teams` dataset - allowing for observations to be repeated - then this new dataset was optimized using the Quasi-Newton optimization that is outlined above. This process was repeated 10,000 times to create 10,000 variations of the optimized parameters. The confidence intervals for the optimized parameters were then found by using the quantile method.

Results

All of the optimization methods outlined above were implemented using R with three different starting values vectors for $\theta = \begin{pmatrix}0\\0\\0\end{pmatrix}, \begin{pmatrix}-1\\1\\-1\end{pmatrix}, \begin{pmatrix}-8\\80\\-20\end{pmatrix}$ With the third starting value being based on the parameter estimates that are given when the logistic model is fitted. Below is a table that shows the output of all three optimizers at all three starting points and the number of iterations each one took:

	Alpha	Beta 1	Beta 2	Iterations
Newton:(0,0,0)	-7.8077	87.7043	-19.5363	5
Newton:(-1,1,-1)	-7.8077	87.7043	-19.5363	4
Newton:(-8,80,-20)	-7.8077	87.7043	-19.5363	5
Quasi-Newton:(0,0,0)	-7.8015	87.6818	-19.5476	28
Quasi-Newton:(-1,1,-1)	-7.8219	87.6086	-19.4316	25
Quasi-Newton:(-8,80,-20)	-7.8214	87.6017	-19.4302	18
Fisher Scoring(through glm()):	-7.8080	87.7040	-19.5360	4

The following table shows the 95% confidence intervals for each parameter, along with their point estimates and standard deviations that were found using bootstrap resampling techniques.

	Lower	Upper	Estimate	Standard Deviation
Alpha	-12.1018	-3.7367	-7.8636	2.1280

	Lower	Upper	Estimate	Standard Deviation
Beta 1	67.6702	109.1632	88.0728	10.5754
Beta 2	-41.5621	2.2570	-19.5275	11.0820

Conclusion

From the table in the results section it can be seen that the Newton-Raphson method is the most consistent method and has the closest optimized values to the logistic regression model fit. It can also be noted that the number of iterations needed for the Newton were much lower than the number of iterations needed for the Quasi-Newton method. Although the Quasi-Newton method consistently had the highest number of iterations needed, it was the method that had the fastest computation time. It can also be seen that the starting point does not seem to effect the amount of iterations needed for the Newton-Raphson method to reach $\hat{\theta}$, while the starting value does seem to impact the number of iterations needed for the Quasi-Newton method, where the closer to the MLE value the starting value is, the quicker that the Quasi-Newton method finds $\hat{\theta}$.

Taking the confidence intervals and parameter estimations created by using bootstrapping methods on the Quasi-Newton optimizer, it can be seen that the MLE values for θ are:

$$\begin{pmatrix} \hat{\alpha} \\ \hat{\beta}_1 \\ \hat{\beta}_2 \end{pmatrix} = \begin{pmatrix} -7.8636 \\ 88.0728 \\ -19.5275 \end{pmatrix}$$

Using Newton-Raphson optimization, the optimal equation that predicts a teams likelihood of going to playoffs based off of the teams run average and batting average is as follows:

$$log(\frac{p_i}{1-p_i}) = -7.808 + 87.704(R_i) - 19.536(H_i)$$

And the optimal equation from the bootstrapping methods using the Quasi-Newton optimizer is as follows:

$$log(\frac{p_i}{1-p_i}) = -7.864 + 88.073(R_i) - 19.528(H_i)$$

Using the 2019 season, the probability of a team going to the playoffs - based on the teams run average and batting average - ar as follows(using both variations of the formula):

Team	ARI	ATL	BAL	BOS	CHW	CHC	CIN	CLE	COL	DET
Newton.Probability	0.482	0.656	0.232	0.652	0.158	0.583	0.216	0.437	0.487	0.035
Bootstrap.Quasi.Newton	0.482	0.657	0.231	0.652	0.157	0.583	0.215	0.437	0.487	0.035

Team	HOU	KCR	ANA	LAD	FLA	MIL	MIN	NYY	NYM	OAK
Newton.Probability	0.771	0.168	0.387	0.787	0.062	0.389	0.784	0.856	0.379	0.659
Bootstrap.Quasi.Newton	0.772	0.167	0.386	0.788	0.061	0.388	0.785	0.857	0.379	0.659

Team	PHI	PIT	SDP	SEA	SFG	STL	TBD	TEX	TOR	WSN
Newton.Probability	0.396	0.227	0.205	0.412	0.140	0.425	0.315	0.542	0.302	0.713
Bootstrap.Quasi.Newton	0.395	0.226	0.204	0.411	0.139	0.425	0.314	0.542	0.301	0.714

From the above table, it can be seen that these two methods for optimizing produce almost identical results - with the Newton-Raphson Method producing identical results to fitting the logistic regression model. The

main difference between the two comes down to iterations needed and computation time, thus to minimize computing time, the Quasi-Newton method will be preferred over the Newton-Raphson method. The Quasi-Newton method also requires less computations, as only the log likelihood function and the data are needed for the function, while the Newton-Raphson method needs the log likelihood, the derivative of the log-likelihood, and the second derivative of the log-likelihood.

Appendix

Code

```
library(tidyverse)
library(Lahman) ##Data within this package
setwd("~/OSU/F2021/ST 541/Project/")
set.seed(101214)
Season_by_Team <- Teams
Season_by_Team <- Season_by_Team%>%subset(yearID >= 1998) ## Choosing all seasons from 1998-2020
Season_by_Team <- Season_by_Team %>%
  mutate(Playoffs = ifelse(DivWin == "Y" | WCWin == "Y",1,0),
         WS_Appearance = ifelse(LgWin == "Y",1,0),
         WS_Champ = ifelse(WSWin == "Y",1,0) ) ##Turning Playoff Appearance in Binary
## Getting all offensive variables and removing 2020
Season_Offensive <- Season_by_Team[Season_by_Team$yearID != 2020,c(15:21,26,44,49:51)]
Season_Offensive <- Season_Offensive%>% mutate(RperAB = R/AB, HperAB = H/AB)
##Graphical Check of responses to make sure no problems
hist(Season_by_Team$HR)
hist(Season_Offensive$H)
hist(Season_Offensive$R)
hist(Season_Offensive$AB)
hist(Season_Offensive$RperAB)
hist(Season_Offensive$HperAB)
##It can be seen that the number of HR a team hits in a season is approximately normal
##It can also be seen that hits and runs and At bats are also approximately normal
##Model that were found to be "best" from glm project
mod2 <- glm(Playoffs ~RperAB + HperAB,data=Season_Offensive,family=binomial)
hist(mod2$fitted.values)
log_odds_1 <- mod2$fitted.values
odds_1 <- exp(log_odds_1)
probs_1 <- odds_1/(1+odds_1)
hist(probs_1)
##Since this is binomial regression, we can assume that the
##log odds of a team making the playoffs has a Bernoulli distribution
##with  pi=exp(RperAb + HperAB)/1+(exp(RperAb + HperAB))
##since logistic regression. It can also be noted that all of those predictor
##variables are approximately normally distributed

##Newton - Raphson
##Getting subset of just Run average, Batting average, and Playoff Appearance
New_Offensive <- Season_Offensive[,c(13,14,10)]
##Creating Bernoulli PMF
bernou <- function(x,theta){
  y <- x[,3]
  r <- x[,1]
  h <- x[,2]
  a <- theta[1]
  b1 <- theta[2]
  b2 <- theta[3]
  p <- (exp(a+r*b1+h*b2))/(1+exp(a+r*b1+h*b2))
  dbinom(y,1,p)##Since Bernoulli(p) is equal to Binom(1,p)
}
```

```
##Creating Log-Likelihood of Distribution
log.like.new <- function(x,theta){
  d <- bernou(x,theta)
  -1*sum(log(d))
}
##Attaching Observed values to variables used in the likelihoods
y <- New_Offensive$Playoffs
r <- New_Offensive$RperAB
h <- New_Offensive$HperAB
##Creating the derivative of the log likelihood
log.like.prime <- function(theta1,theta2,theta3){
  a <- vector(length = nrow(New_Offensive)) ##alpha term
  for(i in 1:nrow(New_Offensive)){
    a[i] <- y[i] - (exp(theta1+theta2*r[i]+theta3*h[i])/
                  (exp(theta1+theta2*r[i]+theta3*h[i])+1))
  }
  b1 <- vector(length = nrow(New_Offensive)) ##beta_1 term
  for(i in 1:nrow(New_Offensive)){
    b1[i] <- r[i]*y[i] - ((r[i]*exp(theta1+theta2*r[i]+theta3*h[i]))/
                          (exp(theta1+theta2*r[i]+theta3*h[i])+1))
  }
  b2 <- vector(length = nrow(New_Offensive)) ##beta_2 term
  for(i in 1:nrow(New_Offensive)){
    b2[i] <- h[i]*y[i] - ((h[i]*exp(theta1+theta2*r[i]+theta3*h[i]))/
                            (exp(theta1+theta2*r[i]+theta3*h[i])+1))
  }
  out <- matrix(c(sum(a),sum(b1),sum(b2)))
  return(out) ##Return a matrix of the derivative log-likelihood values
}
##Creating the second derivative of the log likelihood
log.lik.prime.2 <- function(theta1,theta2,theta3){
  a <- vector(length = nrow(New_Offensive))##Double of alpha
  for(i in 1:nrow(New_Offensive)){
    a[i] <- - (exp(theta1+theta2*r[i]+theta3*h[i])/
                 ((exp(theta1+theta2*r[i]+theta3*h[i])+1)^2))
  }
  b1 <- vector(length = nrow(New_Offensive))##Double derivative of beta 1
  for(i in 1:nrow(New_Offensive)){
    b1[i] <- - (((r[i]^2)*exp(theta1+theta2*r[i]+theta3*h[i]))/
                  ((exp(theta1+theta2*r[i]+theta3*h[i])+1)^2))
  }
  b2 <- vector(length = nrow(New_Offensive)) ##Double Derivative of beta 2
  for(i in 1:nrow(New_Offensive)){
    b2[i] <- - (((h[i]^2)*exp(theta1+theta2*r[i]+theta3*h[i]))/
                  ((exp(theta1+theta2*r[i]+theta3*h[i])+1)^2))
  }
  ab1 <- vector(length = nrow(New_Offensive)) ##alpha and beta 1
  for(i in 1:nrow(New_Offensive)){
    ab1[i] <- - (((r[i])*exp(theta1+theta2*r[i]+theta3*h[i]))/
                   ((exp(theta1+theta2*r[i]+theta3*h[i])+1)^2))
  }
  ab2<- vector(length = nrow(New_Offensive)) ##Alpha and beta 2
  for(i in 1:nrow(New_Offensive)){
```

```
    ab2[i] <- - (((h[i])*exp(theta1+theta2*r[i]+theta3*h[i]))/
                   ((exp(theta1+theta2*r[i]+theta3*h[i])+1)^2))
  }
  b1b2 <- vector(length = nrow(New_Offensive)) ##beta 1 and beta 2 term
  for(i in 1:nrow(New_Offensive)){
    b1b2[i] <- - (((r[i]*h[i])*exp(theta1+theta2*r[i]+theta3*h[i]))/
                    ((exp(theta1+theta2*r[i]+theta3*h[i])+1)^2))
  }
  ##Creating the matrix output for the second derivative of the log-likelihood
  out <- matrix(c(sum(a), sum(ab1), sum(ab2),
                  sum(ab1), sum(b1), sum(b1b2),
                  sum(ab2),sum(b1b2),sum(b2)), nrow=3,byrow = TRUE)
  return(out)
}
##Function for Newton-Raphson Updating
newton_MLE <- function(theta,m=10000, tol=0.00001){
  iter <- 0
  theta_t <- theta
  ##Updating rule
  theta <- theta_t -
    solve(log.lik.prime.2(theta1=theta_t[1,],theta2=theta_t[2,],theta3=theta_t[3,])) %*%
    log.like.prime(theta1=theta_t[1,],theta2=theta_t[2,],theta3=theta_t[3,])
  while(abs(theta[1,] - theta_t[1,])>tol){
    ##Keep running while difference of updated and new is less than the tolerance level
    while(abs(theta[2,] - theta_t[2,])>tol){
      while(abs(theta[3,] - theta_t[3,])>tol){
        iter <- iter+1
        if(iter > m)
          stop("No solution found")
          theta_t <- theta
          theta <- theta_t -
            solve(log.lik.prime.2(theta1=theta_t[1,],theta2=theta_t[2,],theta3=theta_t[3,])) %*%
            log.like.prime(theta1=theta_t[1,],theta2=theta_t[2,],theta3=theta_t[3,])

      }
    }
  }
  info <<- list(theta,iter) ##Return MLE and number of iterations needed
  return(info)
}
newton_MLE(theta=rbind(-8,80,-20)) ##Matches output from GLM
newton_MLE(theta=rbind(0,0,0))
newton_MLE(theta=rbind(-1,1,-1))

##Trying the quasinewton method
optim(par=c(0,0,0),fn=log.like.new,x=New_Offensive,method="BFGS") ##Implementation
optim(par=c(-1,1,-1),fn=log.like.new,x=New_Offensive,method="BFGS")
optim(par=c(-8,80,-20),fn=log.like.new,x=New_Offensive,method="BFGS")

##Quasi-Newton takes longer than regular Newton

##Bootstrapping for confidence intervals of estimates,
##starting at 0,0,0, using resampling of the data and allowing for replicates:
```

```r
set.seed(101214)
iter<- 10000 ##Size of Bootstrap Sample
thetas <- matrix(nrow=3,ncol = iter) ##Creating matrix to store values
for(i in 1:iter){
  ##Creating new sample from observed sample,allowing repeated observations
  Updated_Offsive <- New_Offensive[sample(1:nrow(New_Offensive),replace=T),]
  y <- Updated_Offsive$Playoffs ##Assigning terms to new observations
  r <- Updated_Offsive$RperAB
  h <- Updated_Offsive$HperAB
  ##Quasi-Newton Method to save computation time
  x <- optim(par=c(0,0,0),fn=log.like.new,x=Updated_Offsive,method="BFGS")
  thetas[,i] <- x$par ##Store MLE values into Theta matrix
}
ci.95.a <- quantile(thetas[1,],c(0.025,0.975),na.rm=T) ##95% confidence interval for alpha
ci.95.b1 <- quantile(thetas[2,],c(0.025,0.975),na.rm=T)##95% confidence interval for beta 1
ci.95.b2 <- quantile(thetas[3,],c(0.025,0.975),na.rm=T)##95% confidence interval for beta 2
##Creating table to display results
Intervals <- data.frame(Lower = c(ci.95.a[1], ci.95.b1[1], ci.95.b2[1]), ##lower bounds
                        Upper = c(ci.95.a[2], ci.95.b1[2], ci.95.b2[2]), ##Upper bounds
                        Estimate = c(mean(thetas[1,]),mean(thetas[2,]),mean(thetas[3,])), ##Theta hat
                        "Standard Deviation" = c(sd(thetas[1,]),sd(thetas[2,]),sd(thetas[3,])))
rownames(Intervals) <- c("a", "b1", "b2") ##Naming rows with which variable they are
Intervals
## optim() used since the computing time is faster,
##even if the number of iterations needed may be more than the Newton-Method
```