STAT 8670 Computational Methods in Statistics

Final project report

Nikhil Gupta
ngupta9@student.gsu.edu
College of Arts & Sciences
Georgia State University, Atlanta, Georgia

English Premier League winner prediction using Hidden Markov Model

Introduction

The English Premier League is one of the biggest football competitions on the planet. It was founded in 1992 and has only grown since. Today, it holds the crown as the most-watched sports league in the world and brings in staggering numbers as it is broadcasted in 212 territories with a potential audience of 4.7 billion people. The league employees a point system for every match. A team receives 3 points for a win and 1 point for a draw. This project aims to predict the winner of the league. Forecasting the future winner is exceptionally difficult and numerous professionals build lucrative careers attempting to do so. A Hidden Markov Model is a tool that allows us to represent the probability distribution over a sequence of observations. The HMM assumed that an observation at time *'t'* was generated by a process whose state is hidden from an observer.

In this paper, the Hidden Markov Model (HMM) will be utilized to forecast the final points for teams, Manchester City Football Club, Manchester United Football Club and Liverpool Football Club. The hidden sequence of states, along with their corresponding probabilities, will be calculated for a given observation sequence.

Methodology

An HMM, defined as $\lambda=(P, E, \pi)$, consists of following elements→
S: set of hidden states
O: set of hidden observations
P: set of transition probability matrix
E: observation emission probability matrix
π: prior probability matrix

Consider a match, if a team is consistently performing well in a season, there is higher chance that the team will perform well in the next match as well. The hidden states and observations are the following for this project.

→ *hidden states = Win, if the team wins the match*
= Lose, otherwise.
→ *observation = High, if the number of goals scored > 2*
= Medium, if goals scored = 2
= Low, if goals scored < 2

Then, employing the Baum-Welch algorithm allows for the training of the HMM. Thus, given the observations and general structure of the HMM, determining the HMM parameters λ that best fit the observations is possible. Next, the forward-backward algorithm will be applied to compute the P(O|)and the Viterbi algorithm generates the most likely sequence of hidden states that produce this observation sequence O. The data set for this project is available at Kaggle[1]. The data-set contains all match data from season 2006/2007 to 2017/2018. Finally, Viterbi algorithm will be utilized to calculate the overall points of the last season(2018-19) top 3 teams and the accuracy will be measured.

Probability calculation

The data acquired from the source is loaded into R programming environment and cleaned for processing. A sequence of *'Win'* and *'Lose'* is generated for the respective team. The label 'Lose' here also considers the matches which resulted into a *'Draw'*. These states will be considered as the hidden states and their calculation is as follows.

Transition matrix:

$$P(Win/Win) = \frac{Number\ of\ instances\ with\ consecutive\ wins}{Total\ number\ of\ instances}$$

$$P(Win/Lose) = \frac{Number\ of\ instances\ with\ a\ Lose\ followed\ by\ a\ Win}{Total\ number\ of\ instances}$$

$$P(Lose/Lose) = \frac{Number\ of\ instances\ with\ consecutive\ Lose}{Total\ number\ of\ instances}$$

$$P(Lose/Win) = \frac{Number\ of\ instances\ with\ a\ Win\ followed\ by\ a\ Lose}{Total\ number\ of\ instances}$$

Note: *'Win' and 'Lose' are encoded as 'A' and 'B' in the code for simplification.*

Prior probability:

The calculation of the initial distribution or the prior probability is calculated using the data total data from season 2006-07 to 2017-18. The last season data which is 2018-19 is used to predict the winner of the league using Viterbi algorithm.

$$\pi = [Probability\ of\ a\ Wins, Probability\ of\ a\ Lose]$$

Note: This is calculated for every individual team as per their past performance.

Emission probability matrix:
The emission matrix is calculated using the past data and the hidden states in the chain. Every combination of hidden states [Win, Lose] and observations [High, Low, Medium] is calculated along with their instance and then divided by the total number of instances in a row to generate the probability matrix.

$$S = \begin{bmatrix} Win.High & Win.Low & Win.Medium \\ Lose.High & Lose.Low & Lose.Medium \end{bmatrix}$$

Note: For simplicity the observations, High, Low and Medium are encoded as 1, 2, 3 respectively in the R code.

High performance = 1;
Low performance = 2;
Medium performance = 3;

The given figure shows the Hidden Markov Chain for this task. The set of hidden states are Win/Lose and the set of observations are High/Medium/Low

Results

The following figure shows a basic flow of the Hidden Markov Model. We have, the hidden conditions which is either Win or Lose. And then the observations which have three states, High, Medium and Low. Figure[2] Shows the number of goals scored by the respective them in their last 418 matches. We can see that the data is evenly distributed with highs and lows. Last season's data which is 2017-18 is not included while calculating the hidden parameters of the Hidden Markov model using the Baum-Welch algorithm. The hold out set will be used to measure the accuracy of the algorithm and then predicting the winner among the three teams.
The Baum–Welch algorithm is a special case of the EM algorithm used to find the unknown parameters of a hidden Markov model (HMM). It makes use of the forward-backward algorithm to compute the statistics for the expectation step.

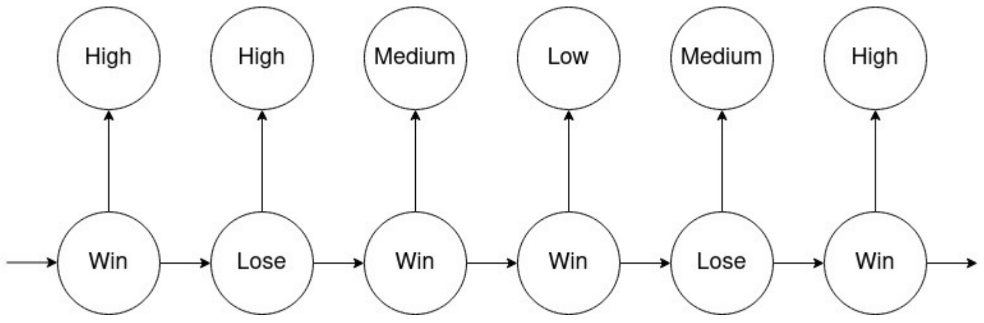


Figure 1: Hidden Markov Chain for English Premier League

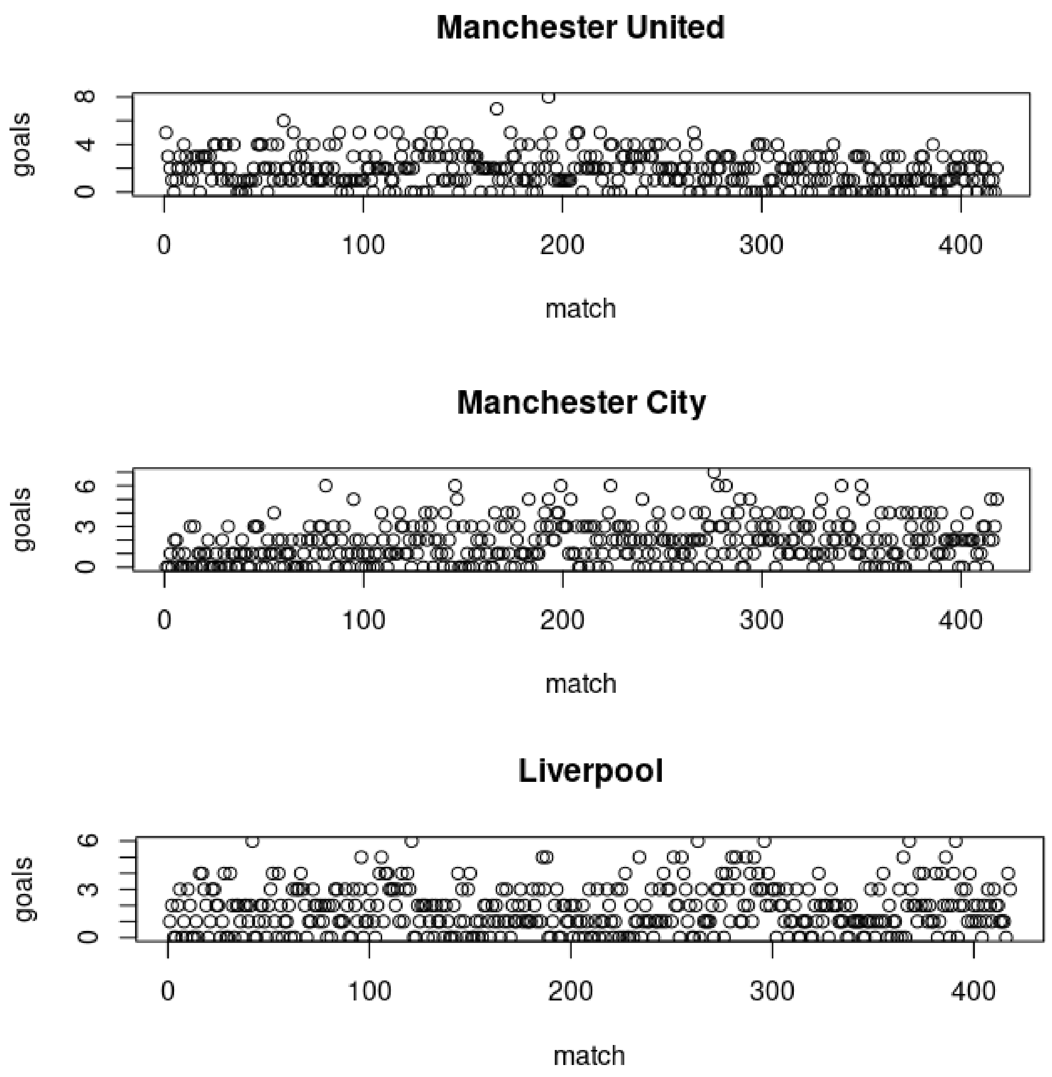


Figure 2: Goals scored by each team in their last 418 matches

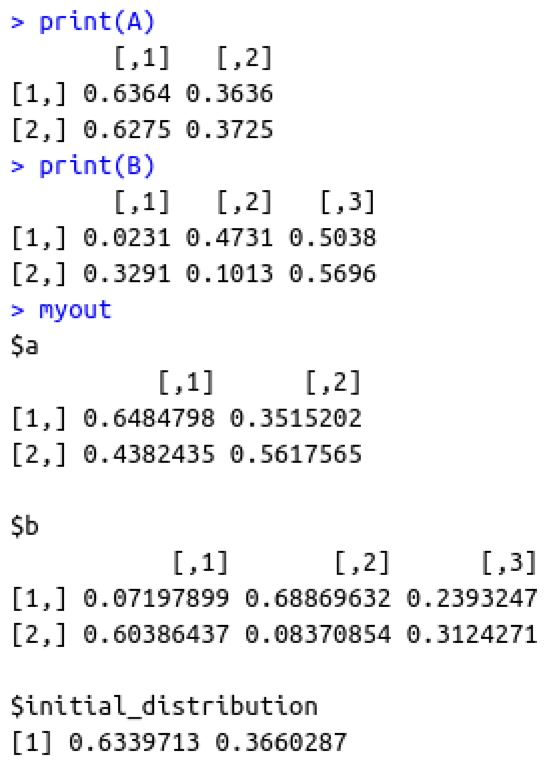
The initial distribution of following teams are:

Manchester United: *π = [0.0634   0.366]*
Manchester City: *π =[0.536   0.464]*
Liverpool: *π = [0.51   0.49]*

From the output below, we can see that the original estimations for transition matrices and emission matrices are altered substantially. We see that the transition matrices all seem lopsided and vary greatly, even more sore when you consider how they changed from the original estimates of the transition matrices.

The number of iterations for a team are run until the transition and emission matrices converges. And the changes in the values are stated below

Manchester United

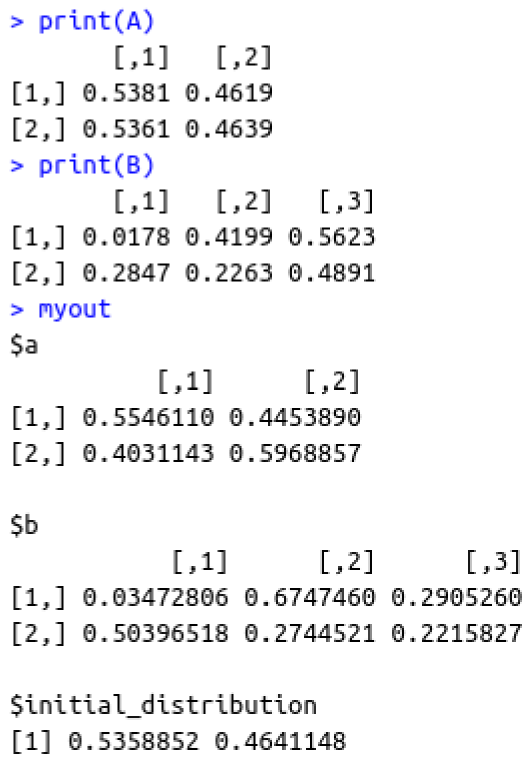


```
> print(A)
       [,1]   [,2]
[1,] 0.6364 0.3636
[2,] 0.6275 0.3725
> print(B)
       [,1]   [,2]   [,3]
[1,] 0.0231 0.4731 0.5038
[2,] 0.3291 0.1013 0.5696
> myout
$a
          [,1]      [,2]
[1,] 0.6484798 0.3515202
[2,] 0.4382435 0.5617565

$b
           [,1]       [,2]      [,3]
[1,] 0.07197899 0.68869632 0.2393247
[2,] 0.60386437 0.08370854 0.3124271

$initial_distribution
[1] 0.6339713 0.3660287
```

Manchester City

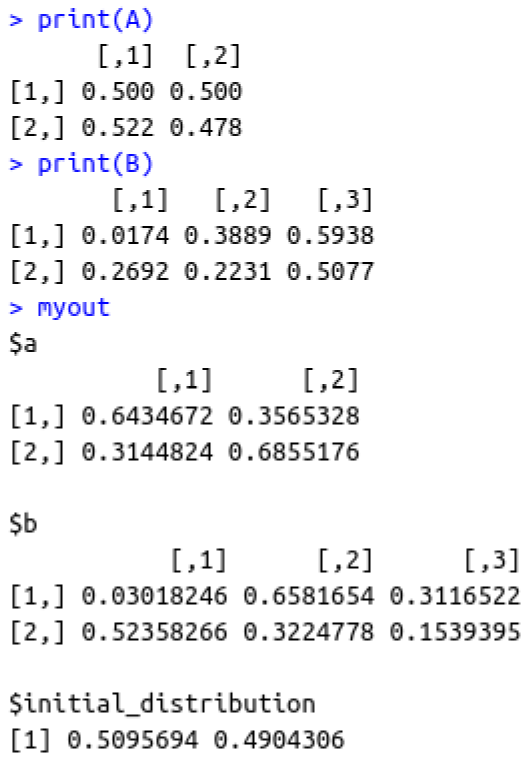


```
> print(A)
       [,1]   [,2]
[1,] 0.5381 0.4619
[2,] 0.5361 0.4639
> print(B)
       [,1]   [,2]   [,3]
[1,] 0.0178 0.4199 0.5623
[2,] 0.2847 0.2263 0.4891
> myout
$a
          [,1]      [,2]
[1,] 0.5546110 0.4453890
[2,] 0.4031143 0.5968857

$b
           [,1]      [,2]      [,3]
[1,] 0.03472806 0.6747460 0.2905260
[2,] 0.50396518 0.2744521 0.2215827

$initial_distribution
[1] 0.5358852 0.4641148
```

Liverpool



```
> print(A)
      [,1]  [,2]
[1,] 0.500 0.500
[2,] 0.522 0.478
> print(B)
       [,1]   [,2]   [,3]
[1,] 0.0174 0.3889 0.5938
[2,] 0.2692 0.2231 0.5077
> myout
$a
          [,1]      [,2]
[1,] 0.6434672 0.3565328
[2,] 0.3144824 0.6855176

$b
           [,1]      [,2]      [,3]
[1,] 0.03018246 0.6581654 0.3116522
[2,] 0.52358266 0.3224778 0.1539395

$initial_distribution
[1] 0.5095694 0.4904306
```

Figure 3: Parameters for each team after Baum-Welch algorithm

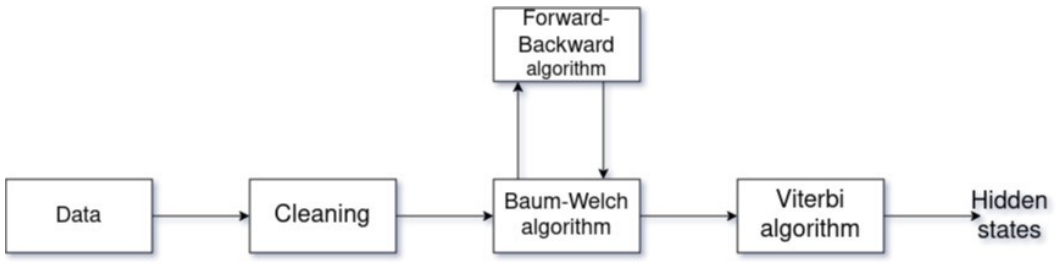


Figure 4: Overall flow of the project from Data to Hidden states as output

The hidden states and points for last 38 matches are:

Manchester United:

```
>   tail(myout.hidden,38)
 [1] "A" "A" "B" "A" "B" "B" "B" "B" "B" "B" "A" "A" "A" "A" "A" "A" "B" "A" "A"
"A" "A" "B" "B" "A" "A" "B" "B" "B" "B" "A"
[31] "B" "B" "A" "B" "B" "A" "A" "A"
>   print(paste("Total points for",p,sep = " " ))
[1] "Total points for Manchester United are: 78"
```

Manchester City:

```
>   tail(myout.hidden,38)
 [1] "A" "A" "B" "B" "A" "A" "B" "A" "B" "A" "A" "A" "A" "A" "B" "A" "A" "B" "A"
"A" "B" "A" "A" "A" "A" "A" "A" "A" "B" "B"
[31] "A" "A" "A" "B" "B" "B" "B" "A"
>   print(paste("Total points for",p,sep = " " ))
[1] "Total points for Manchester City are: 88"
```

Liverpool:

```
>   tail(myout.hidden,38)
 [1] "B" "B" "A" "A" "A" "B" "B" "B" "B" "A" "A" "B" "B" "A" "A" "B" "B" "B" "A"
"A" "B" "B" "B" "A" "A" "B" "B" "B" "B" "B"
[31] "B" "A" "A" "A" "B" "B" "B" "A"
>   print(paste("Total points for",p,sep = " " ))
[1] "Total points for Liverpool are: 68"
```

Conclusion

As per the above simulation, Manchester City would be the winner among the three teams who competed with most of the points. The results can be verified by observing the original data for the season 2018-19 when the actual winner was also Manchester City. Though, there are many assumptions which this algorithm takes. But, this is just a simulation of the events which could happen and can lead to Manchester City winning the tournament. The same approach could be applied to all the teams and points can be calculated.

Works Cited

[1] Data retrieved from https://www.kaggle.com/zaeemnalla/premier-league
[2] https://en.wikipedia.org/wiki/Baum%E2%80%93Welch_algorithm
[3] www.adeveloperdiary.com

R-code

```
#CMS-Final project
#Reading raw data downloaded from kaggle into a dataframe
setwd("~/Documents/cms/project")
df = read.csv('results.csv')

final <- function(df, team, n.iter){

#Change this team to get the points for that team
#team = "Manchester United"
#team = "Manchester City"
#team = "Liverpool"
#Extracting one team's data from the dataframe
#mufc <- subset(df, df$home_team==team | df$away_team==team)
#df <- subset(mufc, mufc$season != "2017-2018")
df <- subset(df, df$home_team==team | df$away_team==team)


#Data preparation for project
goals <- c()
result <- c()
for(i in 1:nrow(df)){
  if(df$home_team[i]==team){
    goals[i] = df$home_goals[i]
    if(df$result[i]=="H"){
      result[i] = "Win"
    } else if(df$result[i]=="D"){
      result[i] = "Draw"
    } else{
      result[i] = "Lose"
    }
  } else {
    goals[i] = df$away_goals[i]
    if(df$result[i]=="A"){
      result[i] = "Win"
    } else if((df$result[i]=="D")){
      result[i] = "Draw"
    } else{
      result[i] = "Lose"
    }
  }
}

result[result=="Draw"] <- "Lose"

performance <- c()
for (i in 1: length(goals)) {
  if(goals[i] > 2){
    performance[i] = "H"
  } else if(goals[i] == 2){
    performance[i] = "M"
  } else{
    performance[i] = "L"
  }
}

#Calculation of Transition Matrix Probabilities
```

```r
ww <- 0
wl <- 0
lw <- 0
ll <- 0

loop <- length(result) - 1
for (i in 1:loop) {
  temp = paste(result[i], result[i+1], sep = "")
  if(temp=="WinWin"){
    ww <- ww + 1
  } else if(temp=="WinLose"){
    wl <- wl + 1
  } else if(temp=="LoseWin"){
    lw <- lw + 1
  } else{
    ll <- ll + 1
  }
}

w2w <- round(ww/(ww+wl),4)
w2l <- 1 - w2w
l2l <- round(ll/(ll+lw),4)
l2w <- 1 - l2l

merge <- c()
for (i in 1:length(performance)) {
  temp = paste(performance[i], result[i], sep = "")
  merge[i] = temp
}
x <- table(merge)
x
A = matrix(c(w2w,w2l,l2w,l2l),2,2,byrow = T)
B = matrix(x, 2,3, byrow = T)
B = B/rowSums(B)
B <- round(B, 4)
B

result_encoded <- result
result_encoded[result_encoded=="Win"] <- "A"
result_encoded[result_encoded=="Lose"] <- "B"

performance_encoded <- performance
performance_encoded[performance_encoded=="H"] <- 1
performance_encoded[performance_encoded=="L"] <- 2
performance_encoded[performance_encoded=="M"] <- 3
performance_encoded <- as.numeric(performance_encoded)

newdf <- data.frame(Hidden=result_encoded, Visible=performance_encoded)

forward = function(v, a, b, initial_distribution){

  T = length(v)
  M = nrow(a)
  alpha = matrix(0, T, M)

  alpha[1, ] = initial_distribution*b[, v[1]]

  for(t in 2:T){
    tmp = alpha[t-1, ] %*% a
    alpha[t, ] = tmp * b[, v[t]]
  }
  return(alpha)
}

backward = function(v, a, b){
  T = length(v)
  M = nrow(a)
  beta = matrix(1, T, M)

  for(t in (T-1):1){
    tmp = as.matrix(beta[t+1, ] * b[, v[t+1]])
    beta[t, ] = t(a %*% tmp)
  }
  return(beta)
}


BaumWelch = function(v, a, b, initial_distribution, n.iter){

  for(i in 1:n.iter){
    T = length(v)
    M = nrow(a)
    K=ncol(b)
    alpha = forward(v, a, b, initial_distribution)
    beta = backward(v, a, b)
    xi = array(0, dim=c(M, M, T-1))

    for(t in 1:T-1){
      denominator = ((alpha[t,] %*% a) * b[,v[t+1]]) %*% matrix(beta[t+1,])
      for(s in 1:M){
        numerator = alpha[t,s] * a[s,] * b[,v[t+1]] * beta[t+1,]
        xi[s,,t]=numerator/as.vector(denominator)
      }
    }


    xi.all.t = rowSums(xi, dims = 2)
    a = xi.all.t/rowSums(xi.all.t)

    gamma = apply(xi, c(1, 3), sum)
    gamma = cbind(gamma, colSums(xi[, , T-1]))
    for(l in 1:K){
      b[, l] = rowSums(gamma[, which(v==l)])
    }
    b = b/rowSums(b)

  }
  return(list(a = a, b = b, initial_distribution = initial_distribution))
}


Viterbi=function(v,a,b,initial_distribution) {

  T = length(v)
  M = nrow(a)
  prev = matrix(0, T-1, M)
  omega = matrix(0, M, T)

  omega[, 1] = log(initial_distribution * b[, v[1]])
  for(t in 2:T){
    for(s in 1:M) {
      probs = omega[, t - 1] + log(a[, s]) + log(b[s, v[t]])
      prev[t - 1, s] = which.max(probs)
      omega[s, t] = max(probs)
    }
  }

  S = rep(0, T)
  last_state=which.max(omega[,ncol(omega)])
  S[1]=last_state

  j=2
  for(i in (T-1):1){
    S[j]=prev[i,last_state]
```

```
      last_state=prev[i,last_state]
      j=j+1
    }

    S[which(S==1)]='A'
    S[which(S==2)]='B'

    S=rev(S)

    return(S)

  }
  A_dist = sum(result_encoded=="A")
  B_dist = sum(result_encoded=="B")

  initial_distribution = c(A_dist/length(result_encoded), B_dist/length(result_encoded))

  x <- sample(1:3, 38, replace=TRUE)

  p <- c(performance_encoded, x)
  myout = BaumWelch(newdf$Visible, A, B, initial_distribution, n.iter)
  myout.hidden=Viterbi(p,myout$a,myout$b,initial_distribution)
  points <- tail(myout.hidden, 38)
  points[points=="B"] = 1
  points[points=="A"] = 3
  points <- as.numeric(points)
  p = paste(team, sum(points), sep = " are: ")
  tail(myout.hidden,38)
  return (paste("Total points for",p,sep = " " ))
}
  final(df, "Manchester United", 100)
  final(df, "Manchester City", 10)
  final(df, "Liverpool", 50)

###################################################################################################
```