Estimating the shooting efficiency of top NBA point guard

Xiang Gao, Jerome Finn, Nilesh Malpekar

2017/12/03

Purpose

Background

As we all know,a point guard controll the court by passing the ball to the player who is wide open. He is a decision maker to deliver assists or finish the attack by himself. The question arises, who has the highest shooting percentage among the top NBA point guards. Is this related to the professional years of experience?

We are using logistic regression to assess the probablity of shooting, also use a binomial model to estimate field goals made by the NBA players. In order to build a hierachical model, each player is treated as a individual group by introducing a random effect called player effect. What's more, recent three years data will be used to check the continuous improvement.

Data

Original data

Original data is retrieved from Kaggle competition site NBA Dataset

Using our homework datasets as a guide, Xiang got our data set to manageable level for our questions. We concentrate on players and years with players representing groups similar to how rats where used as groups in our previous lectures.

The zip file contains two separate CSV files:

- Seasons_Stats.csv - season specific data since 1950
- Players.csv - player specific data

```
season_data <- read.table("Seasons_Stats.csv", header=TRUE, sep = ",", quote = '"')
player_data <- read.table("Players.csv", header=TRUE, sep = ",", quote = '"')
```

For this project, we are focusing on specific fields within this dataset which are described beow:

Datasource	Field_Name	Description
Seasons_Stats	Year	NBA year
Season_Stats, Players	Player	Player name
Players	Height	Height in cm
Seasons_Stats	Pos	Player position
Seasons_Stats	Tm	NBA team name

Datasource	Field_Name	Description
Seasons_Stats	Age	Player age
Seasons_Stats	MP	Minutes played
Seasons_Stats	PER	Player Efficiency Rating
Seasons_Stats	FG	Field Goals
Seasons_Stats	FGA	Field Goals Attempted
Seasons_Stats	BLK	Blocks

The dataset contains duplicate rows for multiple players for the same year. As part of the data preparation, we have removed duplicate rows based on **Year** and **Player**.

```
# Keep only single record for a player and year; rows with highest FG is retained
season_data <- season_data[with(season_data, order(Year, Player, -FG)), ]
season_data <- distinct(season_data, Year, Player, .keep_all = TRUE)
```

Feature creation

For this project, we need to extract following two features from the original dataset

- **Experience** as number of years of NBA experience.
- **FG%** as $FieldGoals/FieldGoalsAttempted$

```
season_data <- season_data %>% group_by(Player) %>% mutate(EXP = 1:n())
season_data[, "FG%"] <- season_data$FG / season_data$FGA
```

Merge Seasons_Stat with Players to get height of the player

```
player_data <- subset(player_data, select = c("Player", "height"))
colnames(player_data)[2] <- "Height"
season_data <- merge(season_data, player_data, by = c("Player"))
```

Preparing modeling data

For our project, we decided to use the data for the last 3 NBA seasons, i.e. 2017, 2016 and 2015.

The data so far is in long format, i.e. for each player there is one row per year. However, we need daa in wide format so that there is a single row per player and year specific attributes should be columns in the dataset. As an example for **FG** (Field Goals) columns should be as follows:

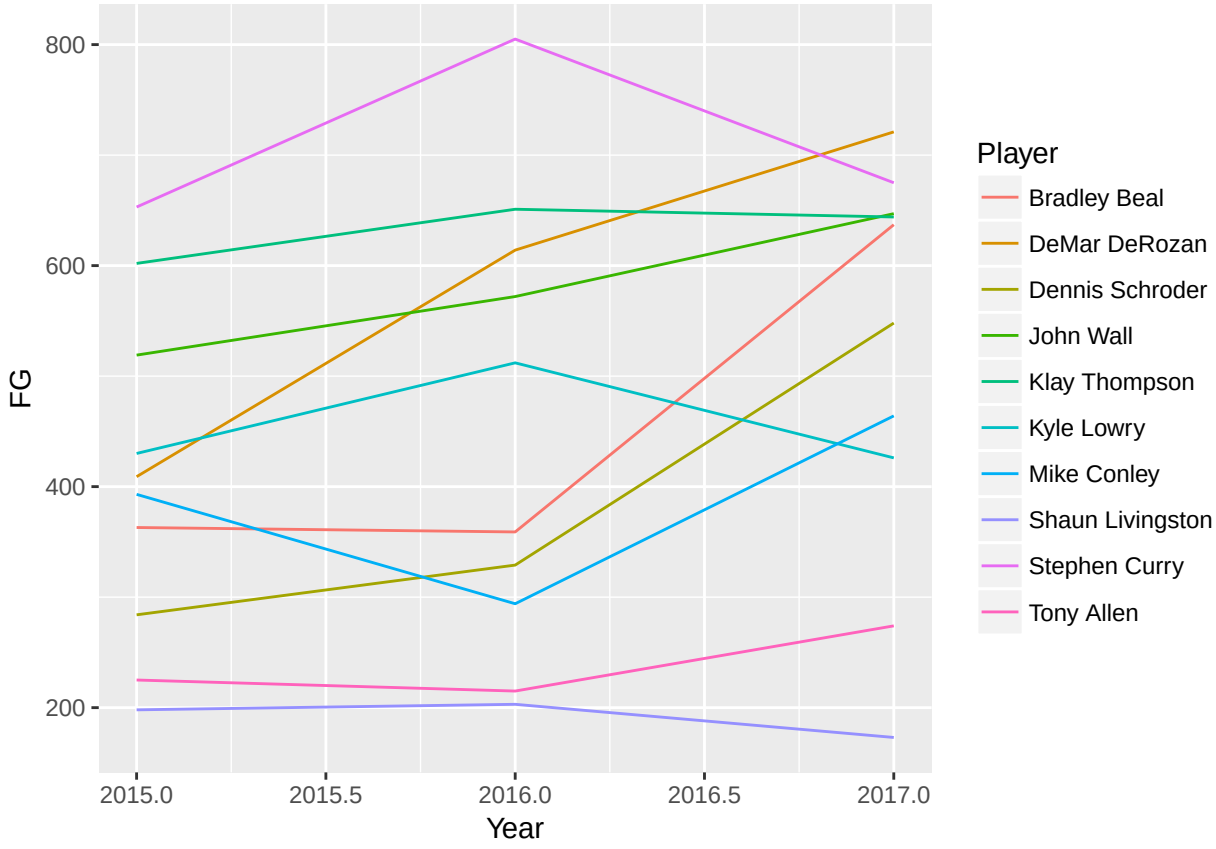
- latest year: FG_0
- 1st prior year: FG_PRIOR_1
- Nth prior year: FG_PRIOR_N

```
# get modeling data
model_data <- get_model_data_wide(season_data, INTERESTED_YEARS, INTERESTED_POSITIONS)
# filter by teams
model_data <- subset(model_data, Tm %in% INTERESTED_TEAMS)
```

```
model_data$Tm <- factor(model_data$Tm)
model_data_row_count <- nrow(model_data)
```

Modeling data visualization

The chat below shows field goals for each player per year. For most player there is growth in field goals from previous year to next year.



Model

Notation

$$y_{ij} = \text{Shooting rate of player *i* at year *j*}\ x_1 = 2017,\ x_2 = 2016,\ x_3 = 2015$$

$$y_i \mid \beta, X_i \quad \text{indep. Bin}(n_i, p_i)\ \text{logit}(p_i) = X_i\beta + \epsilon_i,\ \epsilon_i \quad \text{iid N}(0, \sigma_\epsilon^2),\ \text{Inv-}\chi^2(\nu_i, s_j^2)$$

DAG Model

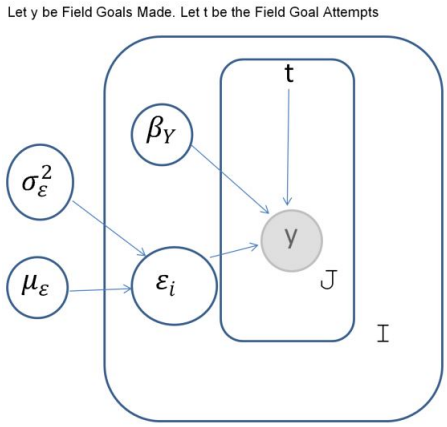


Figure 1: I represents the players(groups) and J the years

Code

JAGS model. I will use scaled-t1 on coefficients in beta. and a flat uniform distribution for sigma of player effect

```
data {
  dimY <- dim(FGM)
}
model {
  for (i in 1:dimY[1]) { ## row per player; total 8 players
    for (j in 1:dimY[2]) { ## column per year; total 3 years i.e. 2017, 2016, 2015
      FGM[i,j] ~ dbin(prob[i,j], FGA[i,j])
      logit(prob[i,j]) <- beta.Year[i]*Yr.Exper[i,j]+Player.Effect[i]
      FGMrep[i,j] ~ dbin(prob[i,j],FGA[i,j])
    }
    beta.Year[i] ~ dt(0,0.16,1)
    Player.Effect[i] ~ dnorm(mu, 1/sigmaPE^2)
  }
  mu ~ dt(0,0.01,1)
  sigmaPE ~ dunif(0,100)
}
```

Computation

Prepare data binding

Subset out the FGM(field goal made),FGA(field goal attempt),Yr.Exper(Years of professional experience)

```
d1 <- list(FGM = model_data[, get_column_names("FG", YEAR_COUNT)],
           FGA = model_data[,get_column_names("FGA", YEAR_COUNT)],
           Yr.Exper = model_data[,get_column_names("EXP", YEAR_COUNT)])

inits1 <- list(list(beta.Year=rep(-1, model_data_row_count), mu=10, sigmaPE=0.001,
                    .RNG.name = "base::Marsaglia-Multicarry", .RNG.seed = 123),
               list(beta.Year=rep(-1, model_data_row_count), mu=-10, sigmaPE=99,
                    .RNG.name = "base::Marsaglia-Multicarry", .RNG.seed = 234),
               list(beta.Year=rep(-1, model_data_row_count), mu=10, sigmaPE=99,
                    .RNG.name = "base::Marsaglia-Multicarry", .RNG.seed = 345),
               list(beta.Year=rep(-1, model_data_row_count), mu=-10, sigmaPE=0.01,
                    .RNG.name = "base::Marsaglia-Multicarry", .RNG.seed = 456))
```

Build model

```
m1 <- jags.model('model-logistic.bug', d1, inits = inits1, n.chains = 4, n.adapt = 1000)
```

```
## Compiling data graph
##    Resolving undeclared variables
##    Allocating nodes
##    Initializing
##    Reading data back into data table
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 30
##    Unobserved stochastic nodes: 52
##    Total graph size: 242
##
## Initializing model
```

Burn-ins and check for convergence

We tried various values for to speed convergence. None lead to very fast convergence but the above values, after much trial and error, were finally acceptable. Still it took a burn-in of over 1 million iterations to get convergence.

```
update(m1, 1024000)
```

Posterior samples and Gelman Statistic

```
x1 <- coda.samples(m1,c('beta.Year','Player.Effect'), n.iter = 40000)
```

```
g.d <- gelman.diag(x1, autoburnin = F)
```

Gelman-Rubin statistic value is 1.0082689.

For details on individual parameters and Gelman plots, please refer to the appendix.

Effective samples sizes are adequate.

```
e.s <- effectiveSize(x1)
all(e.s > 400)
```

```
## [1] TRUE
```

For details on individual sample sizes, please refer to the appendix.

Retrieve replicate dataset and probabilities

```
x2 <- coda.samples(m1, c('beta.Year','Player.Effect','prob','FGMrep'),
                   n.iter = 40000, thin=40)
```

Model Assessment

Coda summary

```
s.x2 <- summary(x2)
```

Beta statistics

	Mean	SD	Naive SE	Time-series SE
Player.Effect[1]	-0.5691785	0.1681858	0.0026593	0.0043554
Player.Effect[2]	-0.7283538	0.2651444	0.0041923	0.0104451
Player.Effect[3]	-0.4280353	0.1397761	0.0022101	0.0025566
Player.Effect[4]	-0.3707747	0.2056030	0.0032509	0.0060162
Player.Effect[5]	-0.2551783	0.1767648	0.0027949	0.0050448
Player.Effect[6]	-0.7411560	0.3491376	0.0055204	0.0165287
Player.Effect[7]	-0.4450910	0.2824370	0.0044657	0.0097402
Player.Effect[8]	-0.5150361	0.3681981	0.0058217	0.0114694
Player.Effect[9]	-0.0654531	0.2544315	0.0040229	0.0107091
Player.Effect[10]	-0.2124962	0.3912786	0.0061867	0.0153389
beta.Year[1]	0.0959709	0.0399059	0.0006310	0.0009681
beta.Year[2]	0.0719957	0.0368259	0.0005823	0.0014483
beta.Year[3]	0.0538990	0.0424348	0.0006710	0.0007836
beta.Year[4]	0.0212636	0.0338041	0.0005345	0.0009818
beta.Year[5]	0.0241994	0.0347227	0.0005490	0.0009764
beta.Year[6]	0.0476580	0.0349015	0.0005518	0.0016575
beta.Year[7]	0.0247575	0.0312569	0.0004942	0.0011026
beta.Year[8]	0.0569522	0.0342255	0.0005412	0.0010597
beta.Year[9]	0.0012486	0.0360023	0.0005692	0.0014777
beta.Year[10]	0.0074255	0.0325377	0.0005145	0.0012832

Beta Quantiles

	2.5%	25%	50%	75%	97.5%
Player.Effect[1]	-0.9210372	-0.6798994	-0.5652727	-0.4546087	-0.2580651
Player.Effect[2]	-1.3018897	-0.9013637	-0.7044385	-0.5263725	-0.2901615
Player.Effect[3]	-0.7063250	-0.5212592	-0.4275187	-0.3325165	-0.1573735
Player.Effect[4]	-0.7824667	-0.5033844	-0.3723625	-0.2416155	0.0419512
Player.Effect[5]	-0.5850971	-0.3792279	-0.2576602	-0.1384623	0.1041661
Player.Effect[6]	-1.5335090	-0.9432669	-0.6899535	-0.4892113	-0.1848889
Player.Effect[7]	-1.0063285	-0.6123765	-0.4442214	-0.2788951	0.1297993
Player.Effect[8]	-1.3407195	-0.7078874	-0.4868642	-0.2985312	0.2111082
Player.Effect[9]	-0.5043808	-0.2553142	-0.0790411	0.1064030	0.4524978
Player.Effect[10]	-0.8585803	-0.4647988	-0.2712098	-0.0182398	0.7492847
beta.Year[1]	0.0228705	0.0687827	0.0947677	0.1223815	0.1782206

	2.5%	25%	50%	75%	97.5%
beta.Year[2]	0.0101213	0.0439879	0.0689149	0.0960583	0.1518915
beta.Year[3]	-0.0290192	0.0260702	0.0528923	0.0822528	0.1375670
beta.Year[4]	-0.0464527	-0.0001511	0.0212426	0.0430840	0.0895525
beta.Year[5]	-0.0462552	0.0012886	0.0250437	0.0488005	0.0900550
beta.Year[6]	-0.0075821	0.0227155	0.0429020	0.0686833	0.1282320
beta.Year[7]	-0.0408222	0.0066421	0.0246440	0.0431402	0.0874128
beta.Year[8]	-0.0101354	0.0367730	0.0544384	0.0756407	0.1325270
beta.Year[9]	-0.0725095	-0.0232172	0.0033091	0.0281596	0.0642608
beta.Year[10]	-0.0728610	-0.0090822	0.0119274	0.0283390	0.0608776

For details on the coda summary, please refer to the appendix.

Check overdispersion, chi-square discrepancy

```
Tchi <- matrix(NA, nrow(FGMrep), model_data_row_count * YEAR_COUNT)
Tchirep <- matrix(NA, nrow(FGMrep), model_data_row_count * YEAR_COUNT)
for (s in 1:nrow(FGMrep)){
  Tchi[s,] <- sum((FGM.v - FGA.v * probs[s,])^2 / (FGA.v * probs[s,] * (1-probs[s,])))
  Tchirep[s,] <- sum((FGMrep[s,] - FGA.v * probs[s,])^2 / (FGA.v * probs[s,] * (1-probs[s,])))
}
```

No over dispersion problem as 0.473.

Results

Density of Various Player through the Years

If we look as Steven Curry's density plots we see no improvement in his performance over the 3 years examined. This was the case with most of our players

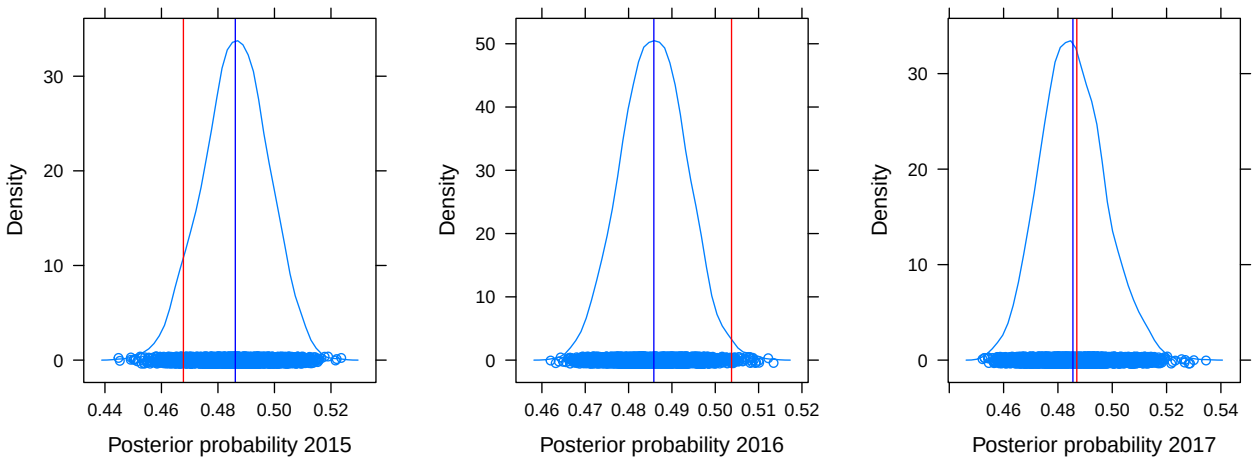
```
player_row_id <- which(model_data$Player == "Stephen Curry")
# posterior prob
posterior <- get_player_posterior_probs(df, model_data, player_row_id)
df_posterior_observed <- get_player_posterior_vs_observed(model_data, player_row_id, posterior)
kable(df_posterior_observed)
```

	posterior	observed
YEAR_0	0.4861452	0.4677755
YEAR_PRIOR_1	0.4858287	0.5037547
YEAR_PRIOR_2	0.4855212	0.4869500

The posterior density does not show Stephen's improvement of making a field goal, let's also check

Russell Westbrook.

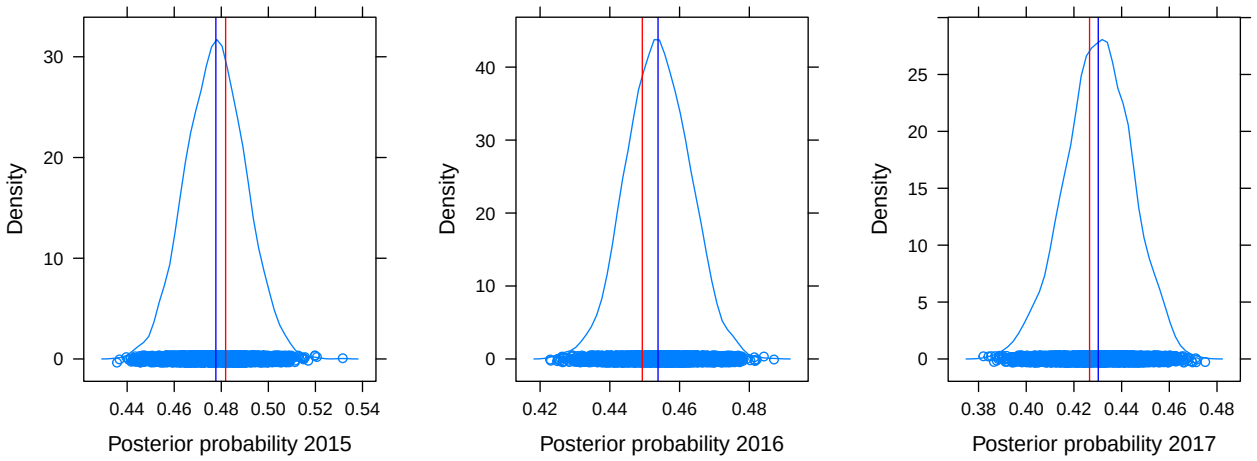
```
plot_player_posterior_probs(model_data, player_row_id, posterior)
```



Check Bradley Beal successfully makes an attempted field goal for the past three years.

	posterior	observed
YEAR_0	0.4776979	0.4818457
YEAR_PRIOR_1	0.4538236	0.4493116
YEAR_PRIOR_2	0.4302003	0.4265570

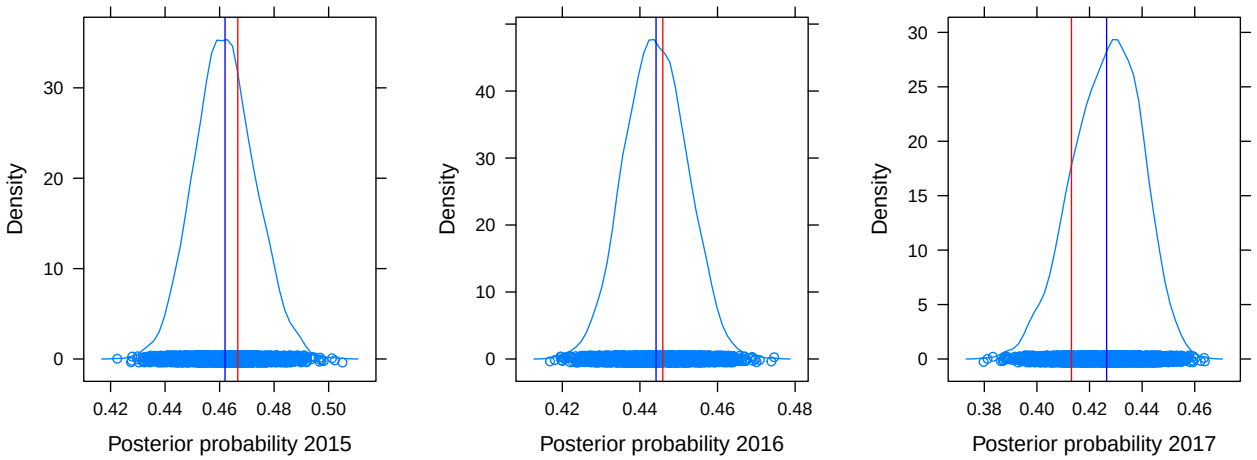
```
plot_player_posterior_probs(model_data, player_row_id, posterior)
```



Check DeMar DeRozan successfully makes an attempted field goal for the past three years.

	posterior	observed
YEAR_0	0.4619954	0.4666667
YEAR_PRIOR_1	0.4441528	0.4458969
YEAR_PRIOR_2	0.4264941	0.4131313

```
plot_player_posterior_probs(model_data, player_row_id, posterior)
```



Check Dennis Schroder successfully makes an attempted field goal for the past three years.

	posterior	observed
YEAR_0	0.4471253	0.4510288
YEAR_PRIOR_1	0.4338317	0.4212548
YEAR_PRIOR_2	0.4206945	0.4270677

Posterior Odds

Here we show the posterior odds of each player improving from one year to the next. We take our poster sample for each player, and take the mean of comparing one year's vector being greater than the previous. As we can see the odds are not extreme that a player may improve from one year to the next, nor do we have a clear pattern. The model does not support the proposition that players improve from one year to another.

Player	2016-2017	2015-2016
Bradley Beal	0.99350	0.00650
DeMar DeRozan	0.99225	0.00775
Dennis Schroder	0.90325	0.09675
John Wall	0.74900	0.25100
Klay Thompson	0.76175	0.23825
Kyle Lowry	0.95175	0.04825
Mike Conley	0.81275	0.18725
Shaun Livingston	0.95825	0.04175
Stephen Curry	0.53550	0.46450
Tony Allen	0.65225	0.34775

Contributions

Xiang deserves a bulk of the credit as the idea was his and did the data gathering and model design, as well a first pass at much of the analysis. We all contributed to the final analysis, although Nilesh did much to improve the R coding. Jerry also contributed to the analysis and lead on much of the early work with regards to putting together the proposal and video presenation with the team's input.

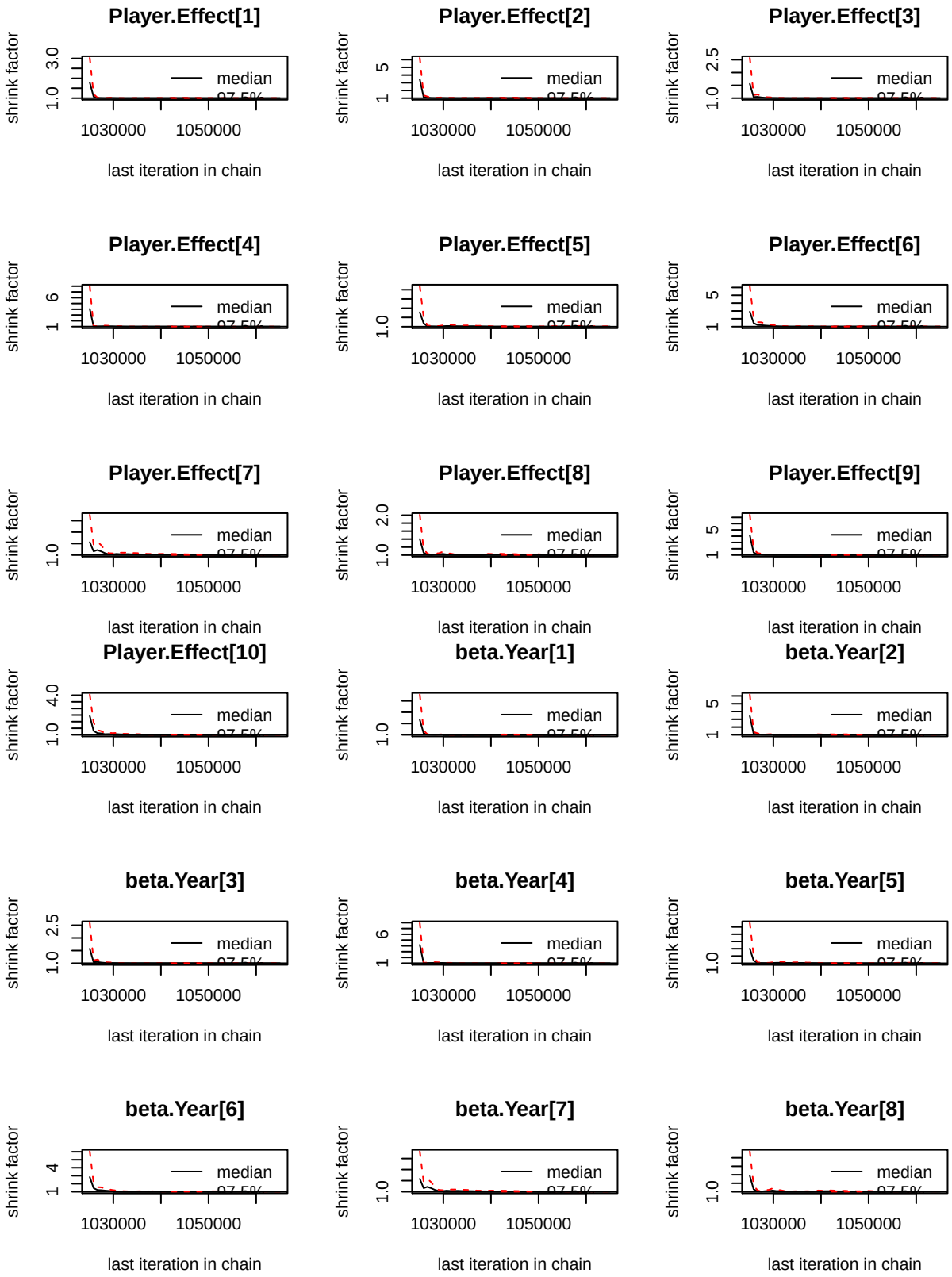
References

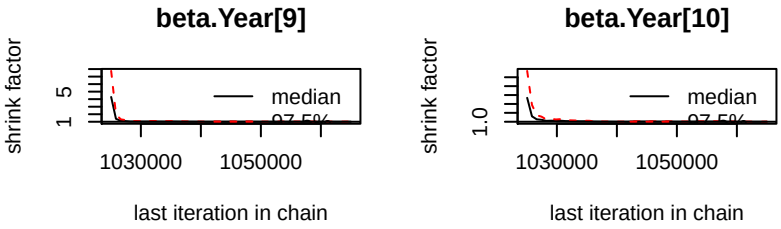
- NBA players stats since 1950
- NBA Dataset
- Basketball reference glossary

Appendix

Gelman-Rubin statistic details

```
##                    Point est. Upper C.I.
## Player.Effect[1]     1.002009   1.005603
## Player.Effect[2]     1.005144   1.012650
## Player.Effect[3]     1.002106   1.006047
## Player.Effect[4]     1.001169   1.001690
## Player.Effect[5]     1.003087   1.008516
## Player.Effect[6]     1.005354   1.014480
## Player.Effect[7]     1.002065   1.005442
## Player.Effect[8]     1.003212   1.009160
## Player.Effect[9]     1.004614   1.012355
## Player.Effect[10]    1.002751   1.005607
## beta.Year[1]         1.002005   1.005605
## beta.Year[2]         1.005080   1.012436
## beta.Year[3]         1.002054   1.005957
## beta.Year[4]         1.001194   1.001737
## beta.Year[5]         1.003018   1.008317
## beta.Year[6]         1.005400   1.014612
## beta.Year[7]         1.002068   1.005465
## beta.Year[8]         1.003173   1.009125
## beta.Year[9]         1.004622   1.012328
## beta.Year[10]        1.002688   1.005543
```


Player.Effect[1]
Player.Effect[2]
Player.Effect[3]
Player.Effect[4]
Player.Effect[5]
Player.Effect[6]
Player.Effect[7]
Player.Effect[8]
Player.Effect[9]
Player.Effect[10]
beta.Year[1]
beta.Year[2]
beta.Year[3]
beta.Year[4]
beta.Year[5]
beta.Year[6]
beta.Year[7]
beta.Year[8]
shrink factor
median
97.5%
1030000
1050000
last iteration in chain



Effective sample size details

```
##  Player.Effect[1]  Player.Effect[2]  Player.Effect[3]  Player.Effect[4]
##         2375.1384          763.3928         3747.3561         1292.1944
##  Player.Effect[5]  Player.Effect[6]  Player.Effect[7]  Player.Effect[8]
##         1508.3783          540.4367         1066.7385         1227.6973
##  Player.Effect[9] Player.Effect[10]      beta.Year[1]      beta.Year[2]
##          616.6346          864.5285         2349.0174          760.0363
##      beta.Year[3]      beta.Year[4]      beta.Year[5]      beta.Year[6]
##         3872.6954         1310.9243         1535.6449          535.9006
##      beta.Year[7]      beta.Year[8]      beta.Year[9]     beta.Year[10]
##         1082.9762         1215.1312          621.2034          869.3080
```

Coda summary details

```
##
## Iterations = 1065040:1105000
## Thinning interval = 40
## Number of chains = 4
## Sample size per chain = 1000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##                         Mean        SD  Naive SE Time-series SE
## FGMrep[1,1]       631.502250 24.591577 0.3888270      0.4075162
## FGMrep[2,1]       714.017750 25.930928 0.4100040      0.5811230
## FGMrep[3,1]       543.672000 22.897303 0.3620381      0.3665709
## FGMrep[4,1]       638.452000 25.289753 0.3998661      0.4841101
## FGMrep[5,1]       650.394000 23.941169 0.3785431      0.4406070
## FGMrep[6,1]       409.506750 19.133739 0.3025310      0.4365414
## FGMrep[7,1]       454.990000 20.343079 0.3216523      0.3640094
## FGMrep[8,1]       171.211500 10.631723 0.1681023      0.1814144
## FGMrep[9,1]       701.395250 25.073925 0.3964536      0.5562042
## FGMrep[10,1]      280.078500 15.132947 0.2392729      0.2964301
## FGMrep[1,2]       362.690500 15.823663 0.2501941      0.2539261
## FGMrep[2,2]       611.688000 21.758884 0.3440382      0.3443898
## FGMrep[3,2]       338.852000 16.030435 0.2534634      0.2572985
## FGMrep[4,2]       592.919500 20.394276 0.3224618      0.3225629
```

```
## FGMrep[5,2]       646.649500 21.379217 0.3380351      0.3409987
## FGMrep[6,2]       520.248250 19.872314 0.3142089      0.3142610
## FGMrep[7,2]       309.551750 14.877757 0.2352380      0.2352515
## FGMrep[8,2]       200.205250 11.373542 0.1798315      0.1804913
## FGMrep[9,2]       776.527750 23.885599 0.3776645      0.3871616
## FGMrep[10,2]      220.012500 12.187554 0.1927022      0.1926446
## FGMrep[1,3]       366.015500 18.894404 0.2987468      0.3529819
## FGMrep[2,3]       422.218500 20.129917 0.3182819      0.4827020
## FGMrep[3,3]       279.846750 16.628466 0.2629191      0.2700686
## FGMrep[4,3]       506.717500 21.340374 0.3374209      0.4192821
## FGMrep[5,3]       598.049000 23.269346 0.3679207      0.4344496
## FGMrep[6,3]       440.760000 20.172993 0.3189630      0.4929920
## FGMrep[7,3]       386.786000 18.209739 0.2879213      0.3248282
## FGMrep[8,3]       203.165750 11.717022 0.1852624      0.2012808
## FGMrep[9,3]       650.712500 23.912001 0.3780819      0.6285229
## FGMrep[10,3]      212.819000 12.570571 0.1987582      0.2148749
## Player.Effect[1]   -0.569179  0.168186 0.0026593      0.0043554
## Player.Effect[2]   -0.728354  0.265144 0.0041923      0.0104451
## Player.Effect[3]   -0.428035  0.139776 0.0022101      0.0025566
## Player.Effect[4]   -0.370775  0.205603 0.0032509      0.0060162
## Player.Effect[5]   -0.255178  0.176765 0.0027949      0.0050448
## Player.Effect[6]   -0.741156  0.349138 0.0055204      0.0165287
## Player.Effect[7]   -0.445091  0.282437 0.0044657      0.0097402
## Player.Effect[8]   -0.515036  0.368198 0.0058217      0.0114694
## Player.Effect[9]   -0.065453  0.254431 0.0040229      0.0107091
## Player.Effect[10]  -0.212496  0.391279 0.0061867      0.0153389
## beta.Year[1]        0.095971  0.039906 0.0006310      0.0009681
## beta.Year[2]        0.071996  0.036826 0.0005823      0.0014483
## beta.Year[3]        0.053899  0.042435 0.0006710      0.0007836
## beta.Year[4]        0.021264  0.033804 0.0005345      0.0009818
## beta.Year[5]        0.024199  0.034723 0.0005490      0.0009764
## beta.Year[6]        0.047658  0.034902 0.0005518      0.0016575
## beta.Year[7]        0.024757  0.031257 0.0004942      0.0011026
## beta.Year[8]        0.056952  0.034225 0.0005412      0.0010597
## beta.Year[9]        0.001249  0.036002 0.0005692      0.0014777
## beta.Year[10]       0.007426  0.032538 0.0005145      0.0012832
## prob[1,1]           0.477698  0.012608 0.0001994      0.0002346
## prob[2,1]           0.461995  0.011167 0.0001766      0.0003418
## prob[3,1]           0.447125  0.013022 0.0002059      0.0002090
## prob[4,1]           0.444773  0.011309 0.0001788      0.0002558
## prob[5,1]           0.472547  0.011547 0.0001826      0.0002578
## prob[6,1]           0.446017  0.012684 0.0002006      0.0003957
## prob[7,1]           0.450813  0.012695 0.0002007      0.0003064
## prob[8,1]           0.541938  0.018932 0.0002993      0.0004172
## prob[9,1]           0.486145  0.011567 0.0001829      0.0003704
## prob[10,1]          0.471069  0.015355 0.0002428      0.0003525
## prob[1,2]           0.453824  0.009046 0.0001430      0.0001450
## prob[2,2]           0.444153  0.008044 0.0001272      0.0001428
```

```
## prob[3,2]           0.433832  0.009716 0.0001536      0.0001553
## prob[4,2]           0.439514  0.007717 0.0001220      0.0001263
## prob[5,2]           0.466513  0.007748 0.0001225      0.0001225
## prob[6,2]           0.434260  0.008843 0.0001398      0.0001380
## prob[7,2]           0.444680  0.009729 0.0001538      0.0001499
## prob[8,2]           0.527805  0.015267 0.0002414      0.0002414
## prob[9,2]           0.485829  0.007481 0.0001183      0.0001175
## prob[10,2]          0.469211  0.012657 0.0002001      0.0002021
## prob[1,3]           0.430200  0.014049 0.0002221      0.0003147
## prob[2,3]           0.426494  0.012959 0.0002049      0.0004485
## prob[3,3]           0.420695  0.015308 0.0002420      0.0002708
## prob[4,3]           0.434303  0.011387 0.0001800      0.0002764
## prob[5,3]           0.460509  0.011656 0.0001843      0.0002900
## prob[6,3]           0.422615  0.011916 0.0001884      0.0004102
## prob[7,3]           0.438591  0.012126 0.0001917      0.0003086
## prob[8,3]           0.513605  0.015840 0.0002504      0.0002780
## prob[9,3]           0.485521  0.011817 0.0001869      0.0004494
## prob[10,3]          0.467370  0.014682 0.0002321      0.0003622
##
## 2. Quantiles for each variable:
##
##                        2.5%        25%        50%       75%     97.5%
## FGMrep[1,1]       582.000000  6.160e+02 631.000000 647.00000 680.00000
## FGMrep[2,1]       663.000000  6.960e+02 714.000000 731.00000 765.00000
## FGMrep[3,1]       499.000000  5.280e+02 544.000000 559.00000 589.00000
## FGMrep[4,1]       589.000000  6.210e+02 638.000000 656.00000 688.00000
## FGMrep[5,1]       603.000000  6.340e+02 650.000000 666.00000 697.00000
## FGMrep[6,1]       372.000000  3.960e+02 409.000000 422.00000 447.00000
## FGMrep[7,1]       415.000000  4.420e+02 455.000000 468.00000 495.00000
## FGMrep[8,1]       150.000000  1.640e+02 171.000000 179.00000 192.00000
## FGMrep[9,1]       651.000000  6.840e+02 702.000000 718.00000 750.00000
## FGMrep[10,1]      250.000000  2.700e+02 280.000000 290.00000 310.00000
## FGMrep[1,2]       332.000000  3.520e+02 363.000000 373.00000 394.00000
## FGMrep[2,2]       569.000000  5.970e+02 611.000000 626.00000 655.00000
## FGMrep[3,2]       307.000000  3.280e+02 339.000000 350.00000 370.00000
## FGMrep[4,2]       553.000000  5.790e+02 593.000000 607.00000 633.00000
## FGMrep[5,2]       605.000000  6.320e+02 647.000000 661.00000 688.00000
## FGMrep[6,2]       481.000000  5.070e+02 520.000000 534.00000 559.02500
## FGMrep[7,2]       280.000000  3.000e+02 310.000000 319.00000 339.00000
## FGMrep[8,2]       178.000000  1.930e+02 200.000000 208.00000 222.00000
## FGMrep[9,2]       729.000000  7.610e+02 776.500000 792.00000 823.00000
## FGMrep[10,2]      197.000000  2.120e+02 220.000000 228.00000 244.00000
## FGMrep[1,3]       328.975000  3.540e+02 366.000000 379.00000 403.00000
## FGMrep[2,3]       381.000000  4.080e+02 423.000000 436.00000 461.00000
## FGMrep[3,3]       247.975000  2.690e+02 280.000000 291.00000 312.00000
## FGMrep[4,3]       465.000000  4.920e+02 507.000000 521.00000 548.00000
## FGMrep[5,3]       554.000000  5.820e+02 597.000000 614.00000 645.02500
## FGMrep[6,3]       400.000000  4.270e+02 441.000000 455.00000 479.00000
```

```
## FGMrep[7,3]       351.975000  3.750e+02 387.000000 399.00000 422.00000
## FGMrep[8,3]       180.000000  1.950e+02 203.000000 211.00000 226.00000
## FGMrep[9,3]       603.975000  6.350e+02 651.000000 667.00000 699.00000
## FGMrep[10,3]      189.000000  2.040e+02 213.000000 221.00000 238.02500
## Player.Effect[1]   -0.921037 -6.799e-01  -0.565273  -0.45461  -0.25807
## Player.Effect[2]   -1.301890 -9.014e-01  -0.704438  -0.52637  -0.29016
## Player.Effect[3]   -0.706325 -5.213e-01  -0.427519  -0.33252  -0.15737
## Player.Effect[4]   -0.782467 -5.034e-01  -0.372363  -0.24162   0.04195
## Player.Effect[5]   -0.585097 -3.792e-01  -0.257660  -0.13846   0.10417
## Player.Effect[6]   -1.533509 -9.433e-01  -0.689953  -0.48921  -0.18489
## Player.Effect[7]   -1.006328 -6.124e-01  -0.444221  -0.27890   0.12980
## Player.Effect[8]   -1.340720 -7.079e-01  -0.486864  -0.29853   0.21111
## Player.Effect[9]   -0.504381 -2.553e-01  -0.079041   0.10640   0.45250
## Player.Effect[10]  -0.858580 -4.648e-01  -0.271210  -0.01824   0.74928
## beta.Year[1]        0.022871  6.878e-02   0.094768   0.12238   0.17822
## beta.Year[2]        0.010121  4.399e-02   0.068915   0.09606   0.15189
## beta.Year[3]       -0.029019  2.607e-02   0.052892   0.08225   0.13757
## beta.Year[4]       -0.046453 -1.511e-04   0.021243   0.04308   0.08955
## beta.Year[5]       -0.046255  1.289e-03   0.025044   0.04880   0.09005
## beta.Year[6]       -0.007582  2.272e-02   0.042902   0.06868   0.12823
## beta.Year[7]       -0.040822  6.642e-03   0.024644   0.04314   0.08741
## beta.Year[8]       -0.010135  3.677e-02   0.054438   0.07564   0.13253
## beta.Year[9]       -0.072510 -2.322e-02   0.003309   0.02816   0.06426
## beta.Year[10]      -0.072861 -9.082e-03   0.011927   0.02834   0.06088
## prob[1,1]           0.453357  4.690e-01   0.477684   0.48625   0.50244
## prob[2,1]           0.440953  4.545e-01   0.461762   0.46932   0.48445
## prob[3,1]           0.421999  4.384e-01   0.446846   0.45597   0.47270
## prob[4,1]           0.423308  4.369e-01   0.445033   0.45248   0.46700
## prob[5,1]           0.449976  4.647e-01   0.472333   0.48029   0.49540
## prob[6,1]           0.422822  4.373e-01   0.445205   0.45417   0.47248
## prob[7,1]           0.426273  4.422e-01   0.450739   0.45944   0.47496
## prob[8,1]           0.504420  5.291e-01   0.541884   0.55431   0.57951
## prob[9,1]           0.463271  4.786e-01   0.486405   0.49409   0.50816
## prob[10,1]          0.440954  4.608e-01   0.471214   0.48120   0.50073
## prob[1,2]           0.436187  4.477e-01   0.453692   0.45997   0.47120
## prob[2,2]           0.428633  4.387e-01   0.444066   0.44965   0.45970
## prob[3,2]           0.414602  4.274e-01   0.433801   0.44034   0.45263
## prob[4,2]           0.424341  4.343e-01   0.439494   0.44485   0.45477
## prob[5,2]           0.451617  4.613e-01   0.466299   0.47166   0.48169
## prob[6,2]           0.416785  4.283e-01   0.434092   0.44013   0.45200
## prob[7,2]           0.425539  4.380e-01   0.444680   0.45126   0.46386
## prob[8,2]           0.497404  5.177e-01   0.527626   0.53820   0.55787
## prob[9,2]           0.471454  4.806e-01   0.485835   0.49095   0.50049
## prob[10,2]          0.444417  4.609e-01   0.468974   0.47760   0.49468
## prob[1,3]           0.401296  4.212e-01   0.430488   0.44002   0.45719
## prob[2,3]           0.399335  4.177e-01   0.427329   0.43597   0.44988
## prob[3,3]           0.390399  4.104e-01   0.420632   0.43127   0.45011
## prob[4,3]           0.411877  4.268e-01   0.434102   0.44158   0.45783
```

```
## prob[5,3]          0.438433  4.523e-01   0.460286   0.46811   0.48434
## prob[6,3]          0.397857  4.152e-01   0.423310   0.43068   0.44436
## prob[7,3]          0.415384  4.304e-01   0.438695   0.44672   0.46243
## prob[8,3]          0.482670  5.031e-01   0.513834   0.52430   0.54456
## prob[9,3]          0.463901  4.775e-01   0.485055   0.49321   0.51066
## prob[10,3]         0.439287  4.578e-01   0.466960   0.47657   0.49806
```