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
Seasons_Stats	Player	Player name
Seasons_Stats	Pos	Player position
Seasons_Stats	FG	Field Goals
Seasons_Stats	FGA	Field Goals Attempted

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

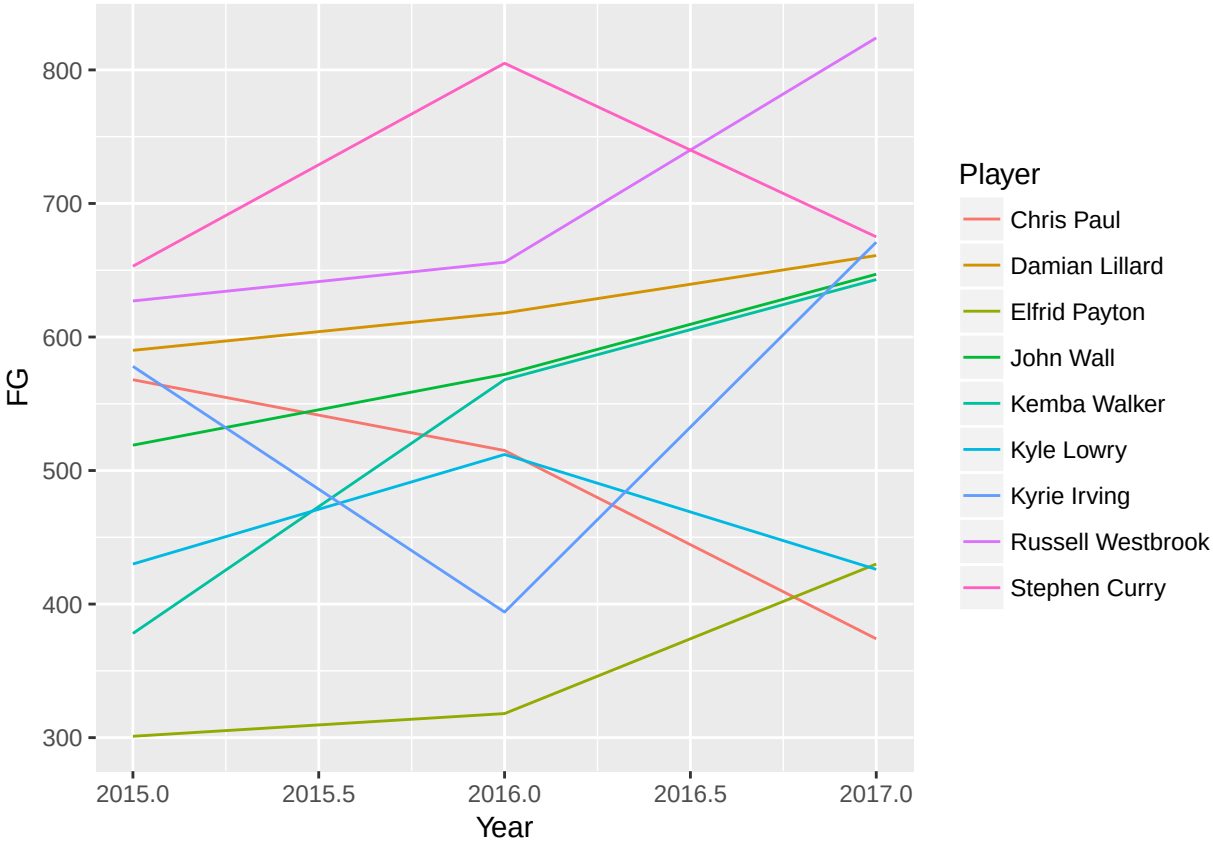
```
# get modeling data
model_data <- get_model_data_wide(season_data, INTERESTED_YEARS, INTERESTED_POSITIONS, 300)
# filter by teams
#model_data <- subset(model_data, Tm %in% INTERESTED_TEAMS)
#model_data$Tm <- factor(model_data$Tm)
model_data_row_count <- nrow(model_data)
display_column_names <- c("Player", get_column_names("FG", YEAR_COUNT), get_column_names("FGA"
head(model_data[, display_column_names])
```

```
##           Player FG_0 FG_PRIOR_1 FG_PRIOR_2 FGA_0 FGA_PRIOR_1 FGA_PRIOR_2
## 1     Chris Paul  374        515        568   785        1114        1170
## 2 Damian Lillard  661        618        590  1488        1474        1360
## 3  Elfrid Payton  430        318        301   912         730         708
## 4      John Wall  647        572        519  1435        1349        1166
## 5   Kemba Walker  643        568        378  1449        1331         981
```

```
## 6     Kyle Lowry  426        512        430    918        1198        1043
```

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

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

$$y_i \mid \beta,\ X_i \quad \text{indep. } \text{Bin}(n_i,\ p_i)\ \text{logit}(p_i) = X_i\beta + \epsilon_i,$$

$$\epsilon_i \quad \text{iid } \text{N}(0,\ \sigma_\epsilon^2)$$

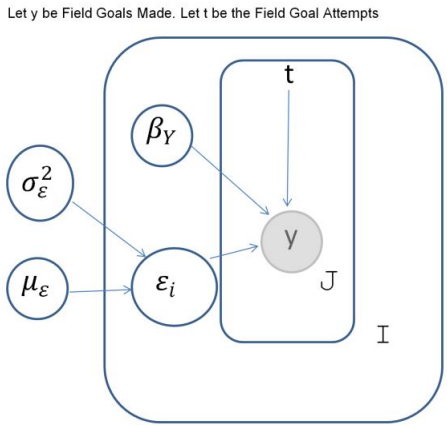


Figure 1: I represents the players(groups) and J the years

DAG Model

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: 27
##    Unobserved stochastic nodes: 47
##    Total graph size: 219
##
## 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 = 70000)
```

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

Gelman-Rubin statistic value is 1.0093691.

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 = 70000, thin=40)
```

Model Assessment

Coda summary

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

Beta statistics

	Mean	SD	Naive SE	Time-series SE
Player.Effect[1]	-0.2260386	0.2937355	0.0035108	0.0095979
Player.Effect[2]	-0.3624165	0.1334658	0.0015952	0.0022049
Player.Effect[3]	-0.4058715	0.1041878	0.0012453	0.0013099
Player.Effect[4]	-0.3457453	0.1865878	0.0022302	0.0040446
Player.Effect[5]	-0.6801438	0.2194273	0.0026227	0.0065100
Player.Effect[6]	-0.6575557	0.3405588	0.0040705	0.0127991
Player.Effect[7]	-0.2634968	0.1653197	0.0019759	0.0032058
Player.Effect[8]	-0.2842511	0.2101956	0.0025123	0.0053309
Player.Effect[9]	-0.0721024	0.2506962	0.0029964	0.0090107
beta.Year[1]	0.0113395	0.0269881	0.0003226	0.0008586
beta.Year[2]	0.0224703	0.0328320	0.0003924	0.0005397
beta.Year[3]	0.0918570	0.0467675	0.0005590	0.0005834
beta.Year[4]	0.0172570	0.0306301	0.0003661	0.0006560
beta.Year[5]	0.0726680	0.0422433	0.0005049	0.0012855

	Mean	SD	Naive SE	Time-series SE
beta.Year[6]	0.0393491	0.0341718	0.0004084	0.0012897
beta.Year[7]	0.0237499	0.0325188	0.0003887	0.0006269
beta.Year[8]	0.0020888	0.0259743	0.0003105	0.0006594
beta.Year[9]	0.0022436	0.0354884	0.0004242	0.0012680

Beta Quantiles

	2.5%	25%	50%	75%	97.5%
Player.Effect[1]	-0.7430554	-0.4143026	-0.2648106	-0.0638835	0.4399994
Player.Effect[2]	-0.6258754	-0.4498736	-0.3613949	-0.2777075	-0.0947695
Player.Effect[3]	-0.6122213	-0.4749100	-0.4049093	-0.3360550	-0.2033266
Player.Effect[4]	-0.7193049	-0.4616980	-0.3489906	-0.2290709	0.0321419
Player.Effect[5]	-1.1318120	-0.8333985	-0.6707660	-0.5108565	-0.3055766
Player.Effect[6]	-1.4876568	-0.8473087	-0.5946969	-0.4155923	-0.1331469
Player.Effect[7]	-0.5671283	-0.3770446	-0.2737184	-0.1565175	0.0801953
Player.Effect[8]	-0.6877947	-0.4189982	-0.2978736	-0.1512430	0.1488200
Player.Effect[9]	-0.4859876	-0.2655888	-0.0938663	0.0940382	0.4539535
beta.Year[1]	-0.0496820	-0.0035860	0.0148405	0.0285210	0.0580974
beta.Year[2]	-0.0432462	0.0013023	0.0225515	0.0441000	0.0873803
beta.Year[3]	0.0005024	0.0605036	0.0914311	0.1228105	0.1855443
beta.Year[4]	-0.0445067	-0.0017889	0.0178579	0.0360977	0.0783745
beta.Year[5]	-0.0007675	0.0409157	0.0706190	0.1021733	0.1605875
beta.Year[6]	-0.0135931	0.0151473	0.0331681	0.0587785	0.1213923
beta.Year[7]	-0.0437958	0.0028589	0.0253941	0.0460258	0.0840493
beta.Year[8]	-0.0508974	-0.0144524	0.0038538	0.0187537	0.0514529
beta.Year[9]	-0.0718489	-0.0213356	0.0050399	0.0294501	0.0606999

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 0.2272857.

Marginal posterior p-value

Here we are checking the marginal posterior predictive p-value of (FGMrep[i] > y[i]). Effectively, comparing replicated data to our model's actual data. The closer we get to 1 or 0, the more the model is off.

```
FGM.p2017 <- numeric(model_data_row_count)
FGM.p2016 <- numeric(model_data_row_count)
FGM.p2015 <- numeric(model_data_row_count)

for (s in 1:model_data_row_count) {
  col_index <- which(colnames(FGMrep) == paste("FGMrep[", s, ",1]", sep = ""))
  FGM.p2017[s] <- mean(FGMrep[,  col_index] >= model_data[s, "FG_0"])

  col_index <- which(colnames(FGMrep) == paste("FGMrep[", s, ",2]", sep = ""))
  FGM.p2016[s] <- mean(FGMrep[,  col_index] >= model_data[s, "FG_PRIOR_1"])

  col_index <- which(colnames(FGMrep) == paste("FGMrep[", s, ",3]", sep = ""))
  FGM.p2015[s] <- mean(FGMrep[,  col_index] >= model_data[s, "FG_PRIOR_2"])
}
posterior_p_df <- data.frame( Player = model_data$Player,
    pValue.2017 = FGM.p2017,
    pValue.2016 = FGM.p2016,
    pValue.2015 = FGM.p2015
    )
kable(posterior_p_df)
```

Player	pValue.2017	pValue.2016	pValue.2015
Chris Paul	0.5332857	0.7611429	0.2222857
Damian Lillard	0.3574286	0.8158571	0.3517143
Elfrid Payton	0.4420000	0.6785714	0.4624286
John Wall	0.3430000	0.8477143	0.3037143
Kemba Walker	0.4125714	0.3674286	0.8150000
Kyle Lowry	0.1730000	0.6662857	0.7547143
Kyrie Irving	0.4352857	0.8052857	0.2882857
Russell Westbrook	0.7522857	0.0787143	0.6761429
Stephen Curry	0.8590000	0.1162857	0.4697143

We can see that the model looks good for Elfrid Payton, but for the other players we are not too precise. The evidence is not strong that years of experience effect a higher rate of field goals made per field goals attempted.

Results

Density of Various Player through the Years

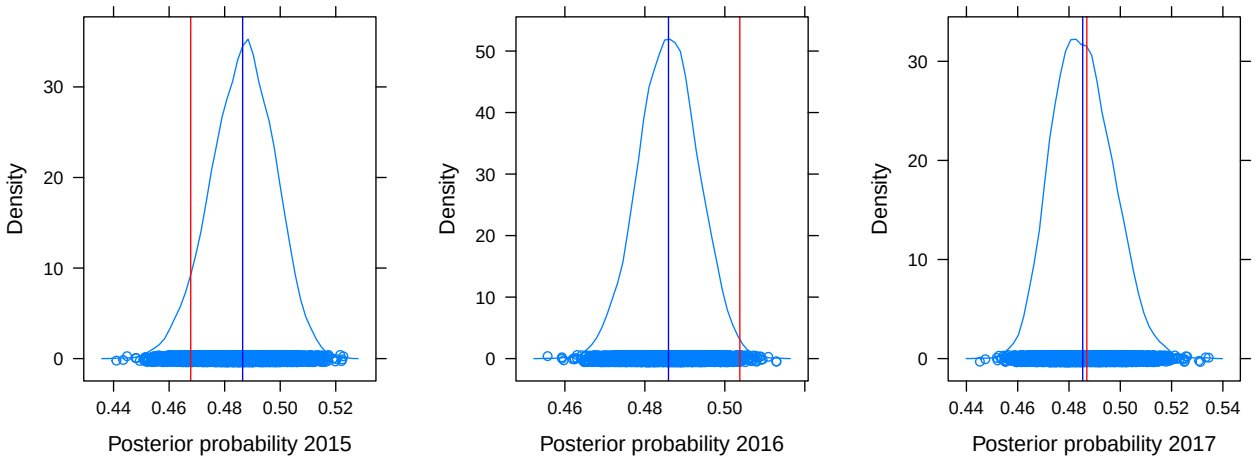
If we look as Stephen Curry's density plots we see no improvement in his performance over the 3 years examined. This was the case with most of our players

```
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	0.4864725	0.4677755
YEAR_PRIOR_1	0.4859077	0.5037547
YEAR_PRIOR_2	0.4853515	0.4869500

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

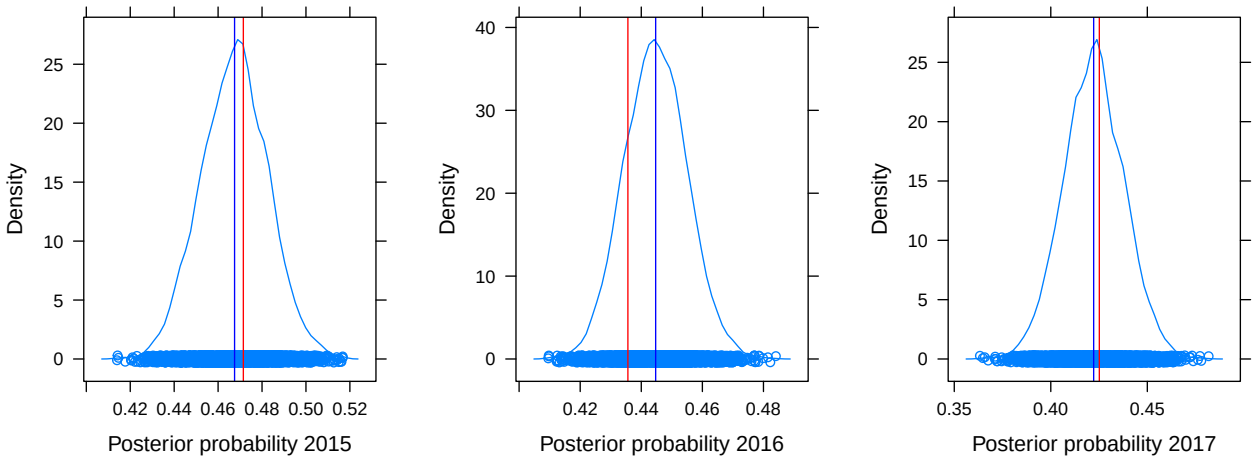
```
plot_player_posterior_probs(model_data, player_row_id, posterior)
```



Check Elfrid Payton successfully makes an attempted field goal for the past three years.

	posterior	observed
YEAR_0	0.4675009	0.4714912
YEAR_PRIOR_1	0.4447113	0.4356164
YEAR_PRIOR_2	0.4222130	0.4251412

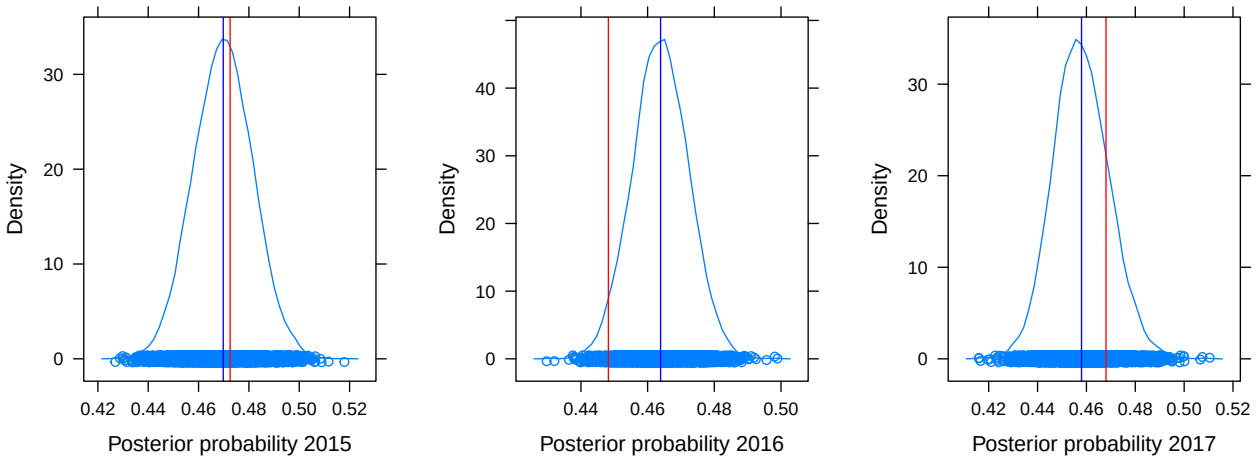
```
plot_player_posterior_probs(model_data, player_row_id, posterior)
```



Check Kyrie Irving successfully makes an attempted field goal for the past three years.

	posterior	observed
YEAR_0	0.4698042	0.4725352
YEAR_PRIOR_1	0.4638863	0.4482366
YEAR_PRIOR_2	0.4579970	0.4680162

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



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
Chris Paul	0.7091429	0.2908571
Damian Lillard	0.7621429	0.2378571
Elfrid Payton	0.9757143	0.0242857
John Wall	0.7341429	0.2658571
Kemba Walker	0.9725714	0.0274286
Kyle Lowry	0.9204286	0.0795714
Kyrie Irving	0.7765714	0.2234286
Russell Westbrook	0.5625714	0.4374286
Stephen Curry	0.5547143	0.4452857

Conclusion

This project shows that the shooting accuracy doesn't have a statistically significant relationship with years of professional experience. The player's personal effect is still the major impact. This is related to the individual player to make a better decision not to pass but taking over and make an attempted basket.

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.

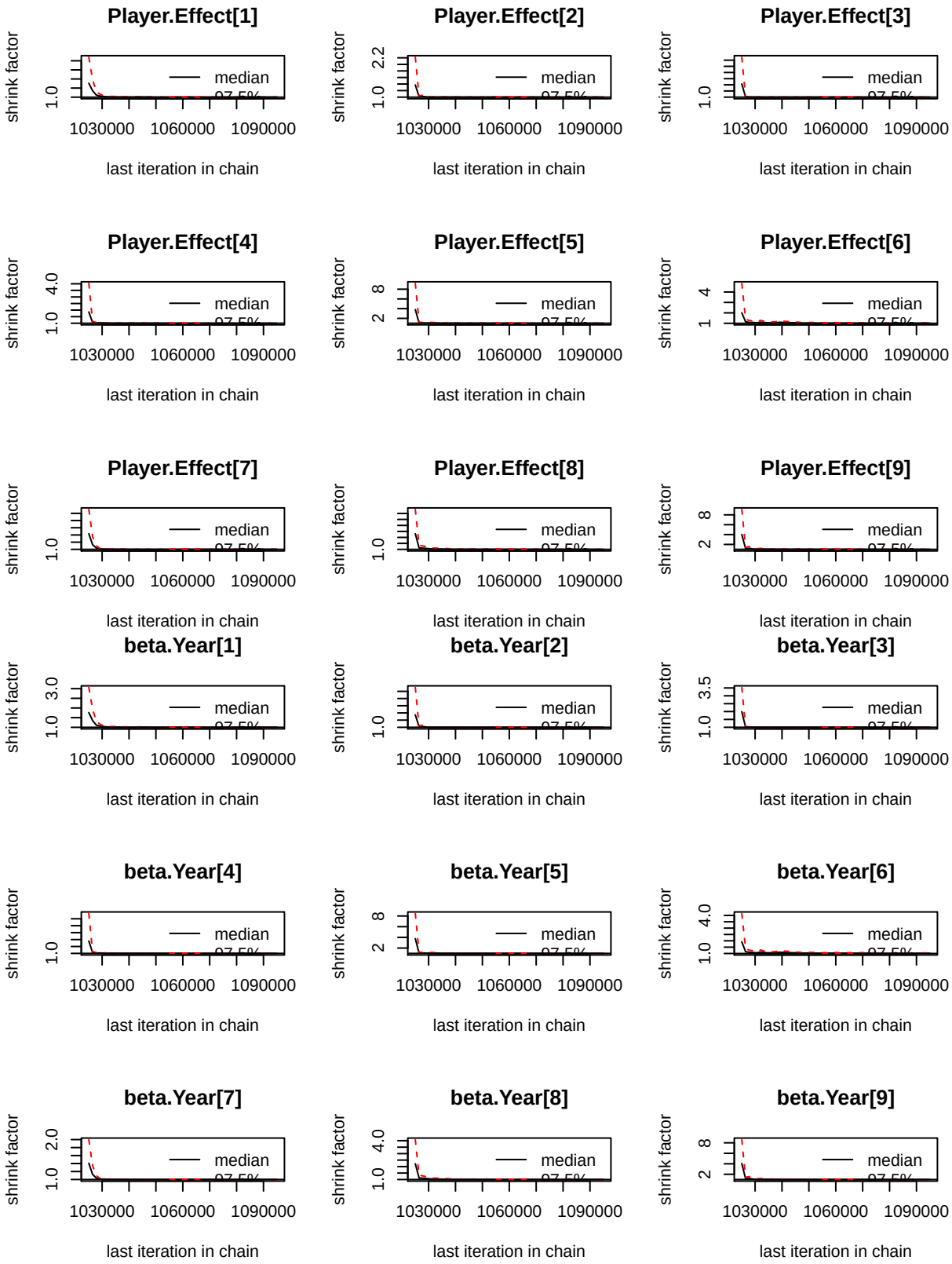
References

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

Appendix

Gelman-Rubin statistic details

```
##                   Point est. Upper C.I.
## Player.Effect[1]   1.003680   1.006958
## Player.Effect[2]   1.000718   1.002027
## Player.Effect[3]   1.000545   1.001573
## Player.Effect[4]   1.000990   1.002863
## Player.Effect[5]   1.001095   1.003374
## Player.Effect[6]   1.018470   1.038213
## Player.Effect[7]   1.001505   1.004540
## Player.Effect[8]   1.002727   1.004686
## Player.Effect[9]   1.003210   1.009200
## beta.Year[1]       1.003601   1.006797
## beta.Year[2]       1.000704   1.001969
## beta.Year[3]       1.000432   1.001241
## beta.Year[4]       1.000985   1.002854
## beta.Year[5]       1.001090   1.003361
## beta.Year[6]       1.018326   1.037958
## beta.Year[7]       1.001508   1.004549
## beta.Year[8]       1.002686   1.004597
## beta.Year[9]       1.003233   1.009264
```


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]
beta.Year[1]
beta.Year[2]
beta.Year[3]
beta.Year[4]
beta.Year[5]
beta.Year[6]
beta.Year[7]
beta.Year[8]
beta.Year[9]
shrink factor
median
97.5%
1030000
1060000
1090000
last iteration in chain

Effective sample size details

```
## Player.Effect[1] Player.Effect[2] Player.Effect[3] Player.Effect[4]
##        1145.1389        4473.8085       13224.5719        2491.6488
## Player.Effect[5] Player.Effect[6] Player.Effect[7] Player.Effect[8]
##        1683.0668         903.5637        3239.0955        1499.7955
## Player.Effect[9]     beta.Year[1]     beta.Year[2]     beta.Year[3]
##        1167.0559        1158.6536        4412.4143       13638.2564
##     beta.Year[4]     beta.Year[5]     beta.Year[6]     beta.Year[7]
##        2524.3486        1617.1695         900.8803        3229.8443
##     beta.Year[8]     beta.Year[9]
##        1533.3348        1173.8779
```

Coda summary details

```
##
## Iterations = 1095040:1165000
## Thinning interval = 40
## Number of chains = 4
## Sample size per chain = 1750
##
## 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]      374.967143 16.861311 2.015e-01      2.567e-01
## FGMrep[2,1]      651.490429 25.484535 3.046e-01      3.180e-01
## FGMrep[3,1]      426.564857 20.195347 2.414e-01      2.525e-01
## FGMrep[4,1]      636.894571 24.391126 2.915e-01      3.399e-01
## FGMrep[5,1]      636.642571 25.805862 3.084e-01      4.096e-01
## FGMrep[6,1]      407.639429 18.805499 2.248e-01      3.590e-01
## FGMrep[7,1]      666.610000 25.233747 3.016e-01      3.114e-01
## FGMrep[8,1]      842.541571 28.534736 3.411e-01      4.174e-01
## FGMrep[9,1]      701.637571 25.329258 3.027e-01      4.888e-01
## FGMrep[1,2]      528.383857 19.462992 2.326e-01      2.555e-01
## FGMrep[2,2]      637.500857 22.062116 2.637e-01      2.755e-01
## FGMrep[3,2]      324.519000 15.184362 1.815e-01      1.904e-01
## FGMrep[4,2]      593.040714 20.940855 2.503e-01      2.545e-01
## FGMrep[5,2]      560.785429 20.714084 2.476e-01      2.423e-01
## FGMrep[6,2]      520.365000 20.367654 2.434e-01      2.428e-01
## FGMrep[7,2]      407.509571 16.275214 1.945e-01      1.938e-01
## FGMrep[8,2]      626.145857 20.677136 2.471e-01      2.538e-01
## FGMrep[9,2]      776.553429 23.354594 2.791e-01      2.809e-01
## FGMrep[1,3]      551.609000 20.685512 2.472e-01      3.348e-01
## FGMrep[2,3]      580.462143 23.124881 2.764e-01      2.997e-01
## FGMrep[3,3]      299.057143 17.031296 2.036e-01      2.101e-01
## FGMrep[4,3]      508.013857 20.989529 2.509e-01      2.848e-01
```

```
## FGMrep[5,3]      396.344429 20.767770 2.482e-01      4.062e-01
## FGMrep[6,3]      442.965714 19.921885 2.381e-01      3.824e-01
## FGMrep[7,3]      565.482286 22.655183 2.708e-01      3.077e-01
## FGMrep[8,3]      637.263143 23.807742 2.846e-01      3.773e-01
## FGMrep[9,3]      651.053143 24.358019 2.911e-01      4.814e-01
## Player.Effect[1]  -0.226039  0.293736 3.511e-03      9.598e-03
## Player.Effect[2]  -0.362417  0.133466 1.595e-03      2.205e-03
## Player.Effect[3]  -0.405872  0.104188 1.245e-03      1.310e-03
## Player.Effect[4]  -0.345745  0.186588 2.230e-03      4.045e-03
## Player.Effect[5]  -0.680144  0.219427 2.623e-03      6.510e-03
## Player.Effect[6]  -0.657556  0.340559 4.070e-03      1.280e-02
## Player.Effect[7]  -0.263497  0.165320 1.976e-03      3.206e-03
## Player.Effect[8]  -0.284251  0.210196 2.512e-03      5.331e-03
## Player.Effect[9]  -0.072102  0.250696 2.996e-03      9.011e-03
## beta.Year[1]       0.011340  0.026988 3.226e-04      8.586e-04
## beta.Year[2]       0.022470  0.032832 3.924e-04      5.397e-04
## beta.Year[3]       0.091857  0.046767 5.590e-04      5.834e-04
## beta.Year[4]       0.017257  0.030630 3.661e-04      6.560e-04
## beta.Year[5]       0.072668  0.042243 5.049e-04      1.286e-03
## beta.Year[6]       0.039349  0.034172 4.084e-04      1.290e-03
## beta.Year[7]       0.023750  0.032519 3.887e-04      6.269e-04
## beta.Year[8]       0.002089  0.025974 3.105e-04      6.594e-04
## beta.Year[9]       0.002244  0.035488 4.242e-04      1.268e-03
## prob[1,1]          0.477538  0.012048 1.440e-04      2.541e-04
## prob[2,1]          0.437839  0.011257 1.345e-04      1.576e-04
## prob[3,1]          0.467501  0.015158 1.812e-04      1.876e-04
## prob[4,1]          0.444026  0.010892 1.302e-04      1.739e-04
## prob[5,1]          0.439303  0.012042 1.439e-04      2.714e-04
## prob[6,1]          0.444093  0.012773 1.527e-04      3.492e-04
## prob[7,1]          0.469804  0.011746 1.404e-04      1.820e-04
## prob[8,1]          0.434048  0.009428 1.127e-04      1.632e-04
## prob[9,1]          0.486473  0.011487 1.373e-04      2.831e-04
## prob[1,2]          0.474704  0.009053 1.082e-04      1.116e-04
## prob[2,2]          0.432300  0.007466 8.923e-05      8.925e-05
## prob[3,2]          0.444711  0.010238 1.224e-04      1.290e-04
## prob[4,2]          0.439759  0.007844 9.376e-05      9.278e-05
## prob[5,2]          0.421476  0.008068 9.643e-05      1.064e-04
## prob[6,2]          0.434386  0.008859 1.059e-04      1.070e-04
## prob[7,2]          0.463886  0.008332 9.959e-05      1.045e-04
## prob[8,2]          0.433524  0.007040 8.415e-05      8.414e-05
## prob[9,2]          0.485908  0.007461 8.917e-05      9.165e-05
## prob[1,3]          0.471881  0.010449 1.249e-04      2.288e-04
## prob[2,3]          0.426813  0.010696 1.278e-04      1.579e-04
## prob[3,3]          0.422213  0.015604 1.865e-04      1.921e-04
## prob[4,3]          0.435527  0.010869 1.299e-04      1.918e-04
## prob[5,3]          0.403923  0.013947 1.667e-04      3.814e-04
## prob[6,3]          0.424765  0.011593 1.386e-04      3.178e-04
## prob[7,3]          0.457997  0.011467 1.371e-04      1.919e-04
```

```
## prob[8,3]          0.433023  0.009570 1.144e-04      1.973e-04
## prob[9,3]          0.485352  0.011676 1.396e-04      3.350e-04
##
## 2. Quantiles for each variable:
##
##                         2.5%        25%        50%       75%     97.5%
## FGMrep[1,1]       3.410e+02 364.000000 375.000000 386.00000 408.00000
## FGMrep[2,1]       6.020e+02 634.000000 651.000000 669.00000 702.00000
## FGMrep[3,1]       3.870e+02 413.000000 427.000000 440.00000 466.00000
## FGMrep[4,1]       5.900e+02 620.000000 637.000000 653.00000 685.00000
## FGMrep[5,1]       5.870e+02 619.000000 637.000000 654.00000 687.00000
## FGMrep[6,1]       3.710e+02 395.000000 408.000000 420.00000 444.02500
## FGMrep[7,1]       6.170e+02 650.000000 667.000000 683.00000 715.00000
## FGMrep[8,1]       7.870e+02 824.000000 843.000000 862.00000 899.00000
## FGMrep[9,1]       6.530e+02 685.000000 702.000000 719.00000 752.00000
## FGMrep[1,2]       4.900e+02 515.000000 528.000000 542.00000 566.00000
## FGMrep[2,2]       5.940e+02 622.000000 638.000000 653.00000 681.00000
## FGMrep[3,2]       2.940e+02 314.000000 325.000000 335.00000 354.00000
## FGMrep[4,2]       5.510e+02 579.000000 593.000000 607.00000 634.00000
## FGMrep[5,2]       5.200e+02 547.000000 561.000000 574.00000 602.00000
## FGMrep[6,2]       4.810e+02 506.000000 520.000000 534.00000 561.00000
## FGMrep[7,2]       3.750e+02 396.000000 408.000000 419.00000 439.00000
## FGMrep[8,2]       5.860e+02 612.000000 626.000000 640.00000 666.00000
## FGMrep[9,2]       7.310e+02 761.000000 776.000000 793.00000 822.00000
## FGMrep[1,3]       5.120e+02 537.000000 552.000000 566.00000 592.00000
## FGMrep[2,3]       5.350e+02 565.000000 581.000000 596.00000 625.00000
## FGMrep[3,3]       2.660e+02 287.000000 299.000000 311.00000 333.00000
## FGMrep[4,3]       4.660e+02 494.000000 508.000000 522.00000 549.00000
## FGMrep[5,3]       3.560e+02 382.000000 397.000000 410.00000 437.00000
## FGMrep[6,3]       4.030e+02 430.000000 443.000000 457.00000 482.00000
## FGMrep[7,3]       5.210e+02 550.000000 565.000000 580.00000 611.00000
## FGMrep[8,3]       5.910e+02 621.000000 637.000000 653.00000 684.00000
## FGMrep[9,3]       6.040e+02 635.000000 650.000000 667.00000 700.00000
## Player.Effect[1] -7.431e-01  -0.414303  -0.264811  -0.06388   0.44000
## Player.Effect[2] -6.259e-01  -0.449874  -0.361395  -0.27771  -0.09477
## Player.Effect[3] -6.122e-01  -0.474910  -0.404909  -0.33606  -0.20333
## Player.Effect[4] -7.193e-01  -0.461698  -0.348991  -0.22907   0.03214
## Player.Effect[5] -1.132e+00  -0.833398  -0.670766  -0.51086  -0.30558
## Player.Effect[6] -1.488e+00  -0.847309  -0.594697  -0.41559  -0.13315
## Player.Effect[7] -5.671e-01  -0.377045  -0.273718  -0.15652   0.08020
## Player.Effect[8] -6.878e-01  -0.418998  -0.297874  -0.15124   0.14882
## Player.Effect[9] -4.860e-01  -0.265589  -0.093866   0.09404   0.45395
## beta.Year[1]     -4.968e-02  -0.003586   0.014841   0.02852   0.05810
## beta.Year[2]     -4.325e-02   0.001302   0.022552   0.04410   0.08738
## beta.Year[3]      5.024e-04   0.060504   0.091431   0.12281   0.18554
## beta.Year[4]     -4.451e-02  -0.001789   0.017858   0.03610   0.07837
## beta.Year[5]     -7.675e-04   0.040916   0.070619   0.10217   0.16059
## beta.Year[6]     -1.359e-02   0.015147   0.033168   0.05878   0.12139
```

```
## beta.Year[7]     -4.380e-02   0.002859   0.025394   0.04603   0.08405
## beta.Year[8]     -5.090e-02  -0.014452   0.003854   0.01875   0.05145
## beta.Year[9]     -7.185e-02  -0.021336   0.005040   0.02945   0.06070
## prob[1,1]         4.526e-01   0.469924   0.477811   0.48554   0.50045
## prob[2,1]         4.157e-01   0.430235   0.437985   0.44524   0.46022
## prob[3,1]         4.382e-01   0.457180   0.467718   0.47774   0.49744
## prob[4,1]         4.228e-01   0.436779   0.443886   0.45128   0.46569
## prob[5,1]         4.163e-01   0.430972   0.439120   0.44743   0.46314
## prob[6,1]         4.202e-01   0.435275   0.443528   0.45240   0.47048
## prob[7,1]         4.470e-01   0.461819   0.469860   0.47779   0.49292
## prob[8,1]         4.159e-01   0.427573   0.434119   0.44051   0.45252
## prob[9,1]         4.626e-01   0.478889   0.486961   0.49453   0.50783
## prob[1,2]         4.567e-01   0.468630   0.474738   0.48092   0.49241
## prob[2,2]         4.177e-01   0.427206   0.432321   0.43740   0.44680
## prob[3,2]         4.249e-01   0.437723   0.444666   0.45160   0.46478
## prob[4,2]         4.244e-01   0.434374   0.439725   0.44502   0.45517
## prob[5,2]         4.061e-01   0.415868   0.421423   0.42706   0.43715
## prob[6,2]         4.168e-01   0.428492   0.434282   0.44035   0.45162
## prob[7,2]         4.476e-01   0.458332   0.463893   0.46955   0.48028
## prob[8,2]         4.198e-01   0.428748   0.433507   0.43835   0.44733
## prob[9,2]         4.711e-01   0.480861   0.485890   0.49093   0.50036
## prob[1,3]         4.517e-01   0.464741   0.471819   0.47890   0.49229
## prob[2,3]         4.057e-01   0.419941   0.426701   0.43376   0.44823
## prob[3,3]         3.917e-01   0.411850   0.422323   0.43270   0.45293
## prob[4,3]         4.138e-01   0.428398   0.435474   0.44274   0.45705
## prob[5,3]         3.757e-01   0.394442   0.404212   0.41413   0.42921
## prob[6,3]         3.998e-01   0.417526   0.425548   0.43262   0.44578
## prob[7,3]         4.368e-01   0.450201   0.457548   0.46543   0.48139
## prob[8,3]         4.144e-01   0.426681   0.432820   0.43918   0.45242
## prob[9,3]         4.645e-01   0.476938   0.484763   0.49314   0.50903
```