An Improved Estimate of Plus-Minus for NBA Players Using Bayesian Regression with Contract and Team Rating Priors

Willis Lu, Andrew Liu, Reed Peterson

Faculty Advisor: Brian Macdonald

Client: Kostas Pelechrinis

Carnegie Mellon University MSP Program

Abstract

In this paper we seek an improved estimate of the Plus-Minus statistic for NBA players using Bayesian regression. Using a Bayesian approach to model this statistic will allow us to generate a distribution rather than a point estimate for each player's true Plus-Minus, providing improved interpretability over non-Bayesian methods. We work with data from the 2010/11 NBA season up to and including the 2018/19 season, and our methods should be able to be easily applied to any future or past NBA seasons. We use Bayesian regression to model the Plus-Minus statistic for players, and we use a nested regression framework to derive logical prior distributions for each player based on their contract value. The model we arrive at corrects for teammate performance, and we believe the model offers improvements on conventional methods for evaluating individual player performance by using additional data such as contract value and offering a measure of uncertainty about a player's true abilities. We call our model the Bayesian Contract Plus Minus, or BCPM. However, the model does somewhat struggle to accurately assess players on rookie contracts, which is an area to explore in future work.

Introduction

There are currently a number of different statistics that are used to measure the performance of NBA players as this is one of the most prevalent tasks for NBA teams. Every front office would like to be able to confidently answer questions such as "is player A better than player B?", allowing them to make better decisions when building a roster. The plus-minus statistic (Basketball-Reference, 2020) is a very natural metric to use when measuring players' contributions to their teams. By definition, plus-minus measures the net point differential for a player's team while that player is on the court. For example, suppose player A entered the game with his team down by 2 points and got substituted out 5 minutes later with his team up by 3. Player A had a plus-minus of 5 points in that stint of play, and the overall statistic is then normalized as plus-minus per 100 possessions, with a possession being any time the team who has the ball changes.

Many estimates of players' plus-minus statistics are biased due to the fact that there are 10 players on the court at any given time, so players who frequently play with an elite player like LeBron James will have an artificially inflated plus-minus compared to players who might be equally productive but play alongside sub-par players. Additionally, point estimates of plus-minus are less informative than distributions of plus-minus since distributions would allow us to examine the uncertainty associated with a certain player's performance. These are some of the core issues that we seek to address in this paper.

Our main tasks are summarized below:

- Can we come up with reasonably informed, logical prior distributions for the players using contract value and potentially team ratings to help improve our main Bayesian regression model?
- Can we construct an informative Bayesian regression model that conditions on all players on the court to eliminate collinearity while also outputting reasonable distributions for plus-minus statistics?
- Can we create an intuitive interactive visualization to display the results of our Bayesian model and allow for comparisons between players?

Data

Contract Data

The first dataset contains information about player contracts. This data was obtained from web-scraping data found on spotrak.com (2017-2019 seasons) and downloading a data set from Kaggle (1990 - 2017 seasons). These two data sets were joined on player names, and the final data frame resulted in 12,724 total contracts, given to 2406 unique players across 32 teams. The joined data frame consisted of the following variables:

- Player Name
- Contract Value
- Year of Contract
- Team
- Type (Rookie vs Non-rookie)

When joining the data frames, we did run into some difficulty with inconsistencies in player names; for example "PJ Tucker" and "P.J. Tucker" are the same player but listed differently. Since there was not a player id common to both data sets and there was not a solution that worked for every player, this issue was fixed manually.

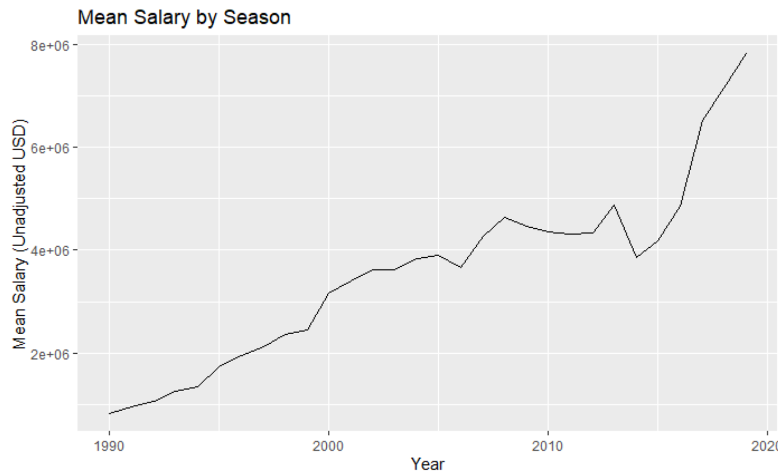


Figure 1: Mean Salary by Season in the NBA

As seen in Figure 1 above, the average salary from 1990 to 2019 has been increasing. After adjusting for this, we decided to use data from during and after the 2010-2011 season, as data before this season had holes such as entire teams missing or inconsistent number of players per team. As will be discussed in the methods section, 5 seasons are used to construct priors, so our model produces results from the 2015-2016 season to the 2018 - 2019 seasons.

Games Data

Our NBA games data comes from fivethirtyeight (fivethirtyeight, 2019). This data is actually used by fivethirtyeight to create ELO rankings for each team updated after each game (though the ELO scores themselves are not relevant to our work). The data goes back to 1946 and includes variables such as the final game scores and who had home court advantage. We will use this data to create a team rating system that may be used to create priors for our bayesian regression.

To actually use this dataset, we end up only using a few variables:

- Team 1 and Team 2
- Final scores for both teams

We use the final scores to calculate a point differential which tells us how much a team won by. Additionally, we also want an indicator for which team is home. Our dataset contains games data in which team 1 is always the home team. In order to create this home/away indicator we duplicate each game so that each game shows twice. So for a game between Team 1 and Team 2, there will be two entries for this game: one entry will show team 1 as the home team and the other entry will show team 2 as the away team. This allows us to even out our dataset and factor in home court advantage into our team rankings. Including home court advantage in our analysis also gives us some convenient information about how much home court advantage is worth. Figure 2 below shows us the average point differential associated with home court advantage. This comes out to 2.793 points in the 2018-2019 season and is around 2.5 for all seasons.

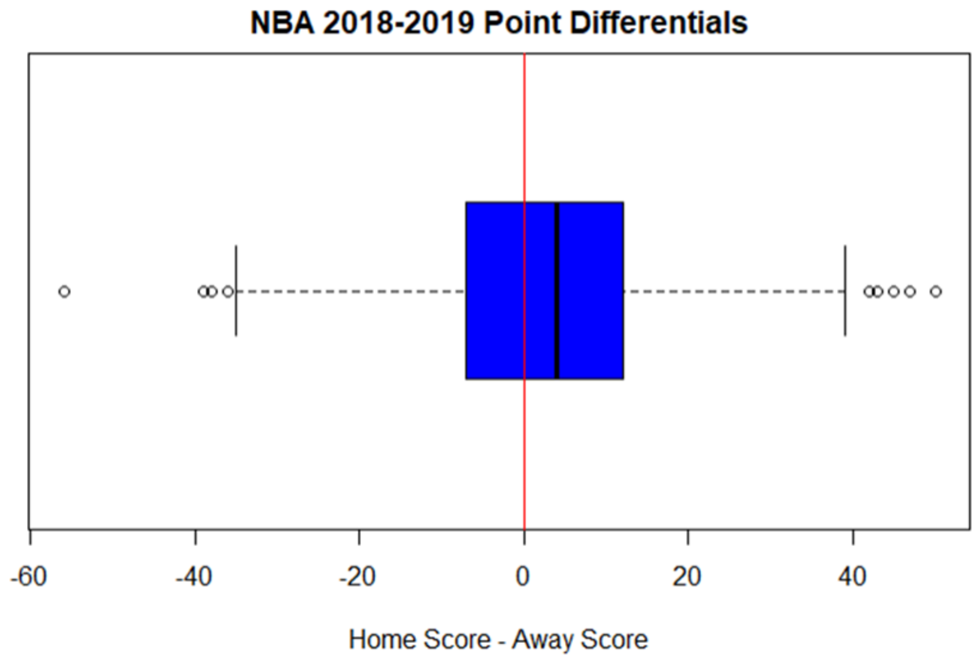


Figure 2: Distribution of NBA Point Differentials for 2018/19 season

Shifts Data

The last main dataset that we used was derived from NBA play-by-play data from eighthirtyfour (Eight Thirty Four, 2019). The play-by-play dataset includes a number of different variables with only a few of interest to us:

- score margin
- IDs of the players on the court for the home and away teams
- ID of a player being substituted on
- Home team
- Away team
- Field goal attempts
- Offensive rebounds
- Free throw attempts
- Turnovers

We perform some data wrangling to reformat this play-by-play data into shifts, where a shift is defined as a period of time where the same 10 players are on the court without any substitutions. For each shift we record the home and away teams, the difference between home points and away points, the IDs of the players on the court for that shift, and the approximate number of possessions that took place during the shift. The number of possessions in a shift was computed using the formula (NBAstuffer, 2021)

$$P = 0.96 \cdot (FGA + TO + 0.44(FTA) - OREB),$$

where P = approximate number of possessions, FGA = field goal attempts, TO = turnovers, FTA = free throw attempts, and $OREB$ = offensive rebounds.

Shifts were then normalized to record point differential per 100 possessions. The final shifts dataset is structured such that each row corresponds to a unique shift, the first column stores the normalized point differential, and the remaining columns (around 500 columns, varies by season) each correspond to an NBA player for the given season. These columns are all filled with zeros *except* for the five players from the home team and the five players on the away team who were on the court during the given shift. The five home players in a shift are denoted with +1, while the five away players in a shift are denoted with -1.

This dataset is used to train our Bayesian regression model, where the set of columns corresponding to the players forms our design matrix $\boldsymbol{X}$ (a sparse matrix of mostly zeros, with five +1s and five -1s per row), while the first column (corresponding to the normalized point differential) is our response variable. This allows for the convenient interpretation that the i^{th}coefficient mean is our estimate of the i^{th}player's plus-minus.

Methods

Deriving Meaningful Priors

When creating our final priors to be used in Bayesian regression, we split the data into rookies and veterans due to the discrepancies in their contract values. For instance, a player on a rookie contract who is performing on a superstar level, such as Luka Doncic, could be extremely underrated if his contract prior was not adjusted upwards, as those on rookie contracts tend to make less than veterans. Our model has two potential variables that we will consider to try to predict player performance: contract value and team rating. Contract value is taken as is, after adjusting for contract value inflation, while team rating is created through linear regression.

To develop team ratings, we use the games data mentioned in the data section above. Our linear regression takes point differentials of each game in a season as its dependent variable and uses team, opponent, and location (home or away) as its independent variables. In this manner, by regressing over all games in a season, we get a coefficient for each team that we then use as team ratings.

Ridge Regression

Ridge regression was used to predict coefficients for each player using a per 100 possession point differential variable. Specifically, we utilize the sparse matrix of shifts data discussed in the data section above. We use the point differential per 100 possessions as the dependent variable and our sparse matrix Xas the design matrix. This does not consider any prior information about the players, their teams, or any other factors - it simply computes a coefficient for each player based on the shifts data. These coefficients act as a proxy for how well the player actually performed.

Random Forest Regression and Gradient Boosting Regression

Now that we have our player coefficients from the ridge regression as well as the team rating and contract values, we can build a model to predict a mean for each player that will serve as the prior mean in the eventual Bayesian regression.

To select our model we first tested linear and ridge regression but both of these yielded undesirable results. We ultimately compared two main methods: the random forest regressor and the gradient boosting regressor. Each model used the following methodology to train and validate in order to select the best model.

- Each of the models utilize five seasons of past data. During training, each model only uses the first four seasons of data. The last season is used as the validation set.
- We considered four models: two random forest regressors and two gradient boosting regressors, each with and without team rating as a prior. All four models included the contract prior.
- The models were compared using MSE (mean squared error) as the main metric for comparison. We also examined the actual results manually to confirm that the results are reasonable and informative as priors.

Ultimately, we ended up choosing the random forest regressor with only contract rating as a predictor as our final model. This model gave us the best results in terms of MSE and manual inspection. The result is a prior mean that is produced by the random forest regressor and a prior standard deviation that is derived from the RMSE (root mean squared error) of the model from the validation set. These priors and standard deviations for each player will then be passed on to our ultimate Bayesian Regression model. An example of this prior model selection process can be found in the technical appendix under "Prior Model Selection 2015/16".

Bayesian Regression

Once we have a sound methodology for deriving meaningful prior distributions for NBA players' plus-minus statistics, we can turn our attention to the second research task of building an informative Bayesian regression model to estimate plus-minus. With inspiration from Deshpande and Jensen (Deshpande and Jensen 2016), we seek a model of the form

$$y = \beta_0 + X\beta + \epsilon,$$

where y is a vector containing the point differential in each shift, β_0 is a constant representing home-court advantage, X is our sparse design matrix described above in the Data section, and β is our vector of coefficients for each player. Note that β is p dimensional, X is $n \times p$ and y is n dimensional where p is the number of NBA players who participated in a given season and n is the number of shifts in a season.

Essentially what this becomes is a regression of point differential on only the ten players on the court for each shift, since all other players take value 0. Also, recall that we have chosen to denote home players with +1 and away players with -1 in order to stay consistent with our choice of representing point differential as $points_{home} - points_{away}$. By regressing the point differential on all players on the court, we should theoretically be able to accomplish the task of obtaining a conditional estimate of players' plus-minuses given the other players on the court.

We implement this model in Python using the pymc3 package (Salvatier, Wiecki, and Fonnesbeck 2016). Pymc3 provides a convenient framework for specifying prior distributions, allowing us to input our priors derived from the random forest model discussed above. Since this is a Bayesian model, the final output is a distribution for each player's plus-minus, along with a distribution for the home court advantage parameter β_0 and a distribution for the error term ϵ. The code used to build this Bayesian regression model can be found in the technical appendix under "Bayesian Reg 2015/16".

Visualizing Results

The last goal of the project given to us by the client was to create an interactive visualization of the results. The application is built entirely in R, using the Shiny, Ggplot2, and Plotly packages. The final application has 4 key visualizations. The first of these visualizations is a plot that displays the selected players' BCPM distributions from the appropriate season. The BCPM estimates are displayed below the plot with the player name, team, mean and standard deviation, as the BCPM is assumed to be normal. The second visualization is a time series of player mean BCPM over the course of 4 seasons. The third visualization is a scatter plot of the mean BCPM vs prior value by season of all players; the plot also allows the user to filter by team. The final plot is a heat-map like matrix, where the user selects a list of players from any of four seasons, and the plot displays the probability that the true BCPM of Player 1 on the x-axis is greater than that of Player 2 on the y-axis. This probability is obtained by directly comparing 2000 samples drawn from each player's BCPM distribution. Further details of implementation can be found in the Application Code portion of the Appendix. The application itself can be found at https://coly1119.shinyapps.io/NBA_Project/ .

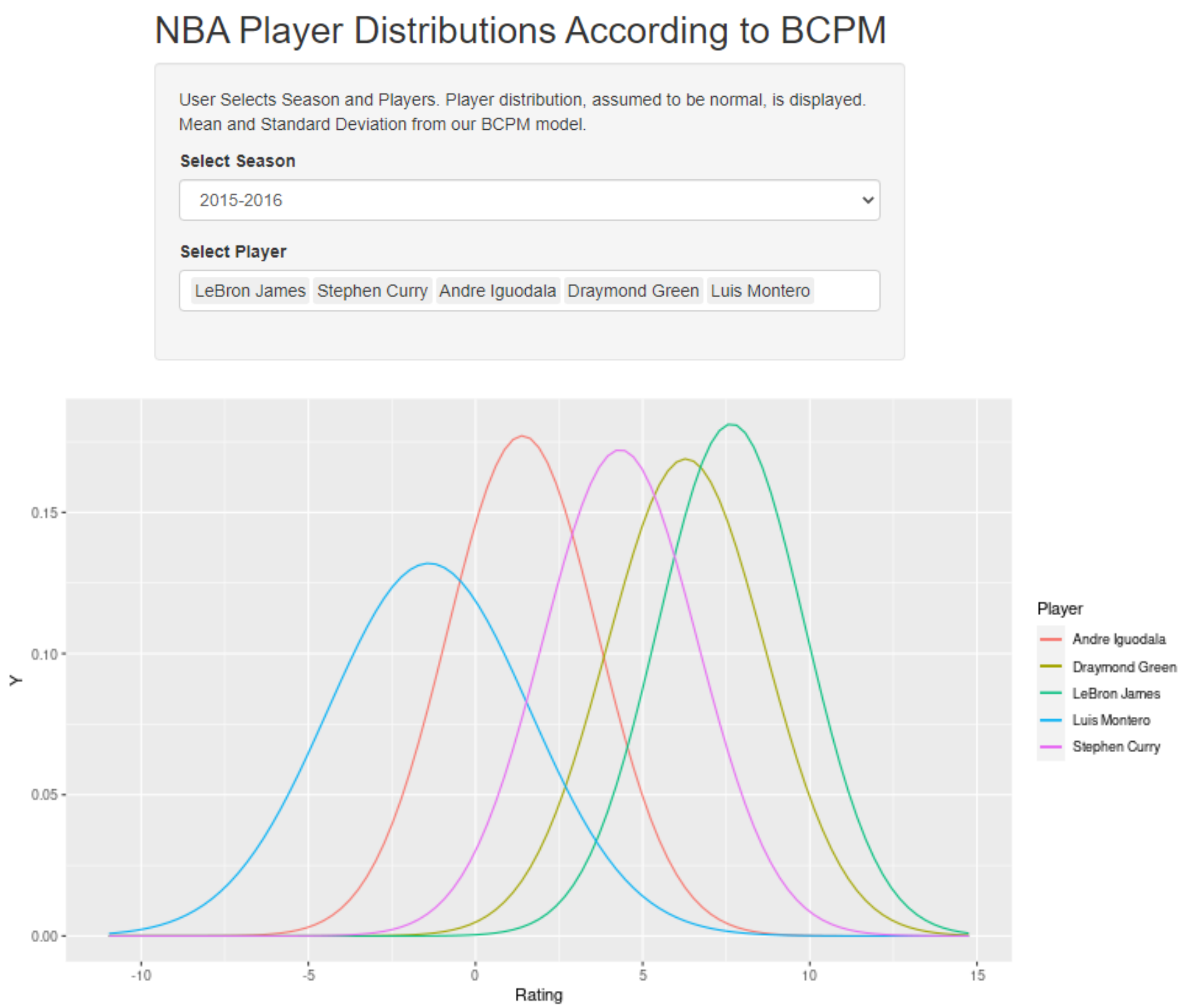
Results

Plus-Minus Posterior Distributions with Bayesian Regression

Our Bayesian regression model yields BCPM estimates that appear to be quite reasonable. One result we can examine is the top ten players based on our BCPM metric in a given season. The top ten players from the 2018/19 NBA season according to BCPM were Jrue Holiday, Steph Curry, James Harden, Paul George, Damian Lillard, Giannis Antetokounmpo, Al Horford, Gordon Hayward, LeBron James, and Mike Conley. This is a reasonable list of mostly elite superstar players who we would expect to be on this list. Some players like Al Horford and Gordon Hayward are slightly overvalued due to extremely high contract values in this season, but overall these results are encouraging.

Examples of the distributions that are fit from the Bayesian model can be found in the next section where we show a prototype of an interactive visualization to display plus-minus.

Visualizing Results



	Name	Team	Rating	SD
20	Andre Iguodala	Golden State Warriors	1.402	2.252
125	Draymond Green	Golden State Warriors	6.289	2.361
284	LeBron James	Cleveland Cavaliers	7.654	2.200
290	Luis Montero	Portland Trailblazers	-1.394	3.023
409	Stephen Curry	Golden State Warriors	4.325	2.317

Figure 3: BCPM Distributions of Select Players from 2015-2016 NBA Season

Figure 3 shows an example output of our first key visualization. As we would expect, the best players have the largest mean BCPM. In our example, Lebron James has the highest mean BCPM, while two other superstars, Draymond Green and Steph Curry are not far behind. An excellent role player in Andre Igoudala is considered above average, i.e. rated greater than 0, while a lesser known role player in Luis Montero is labeled as below average. We also witness the behavior that Luis Montero appears to have a larger standard deviation in BCPM, which can be attributed to less playing time, as having less data on Montero would contribute to greater uncertainty in his BCPM estimate.

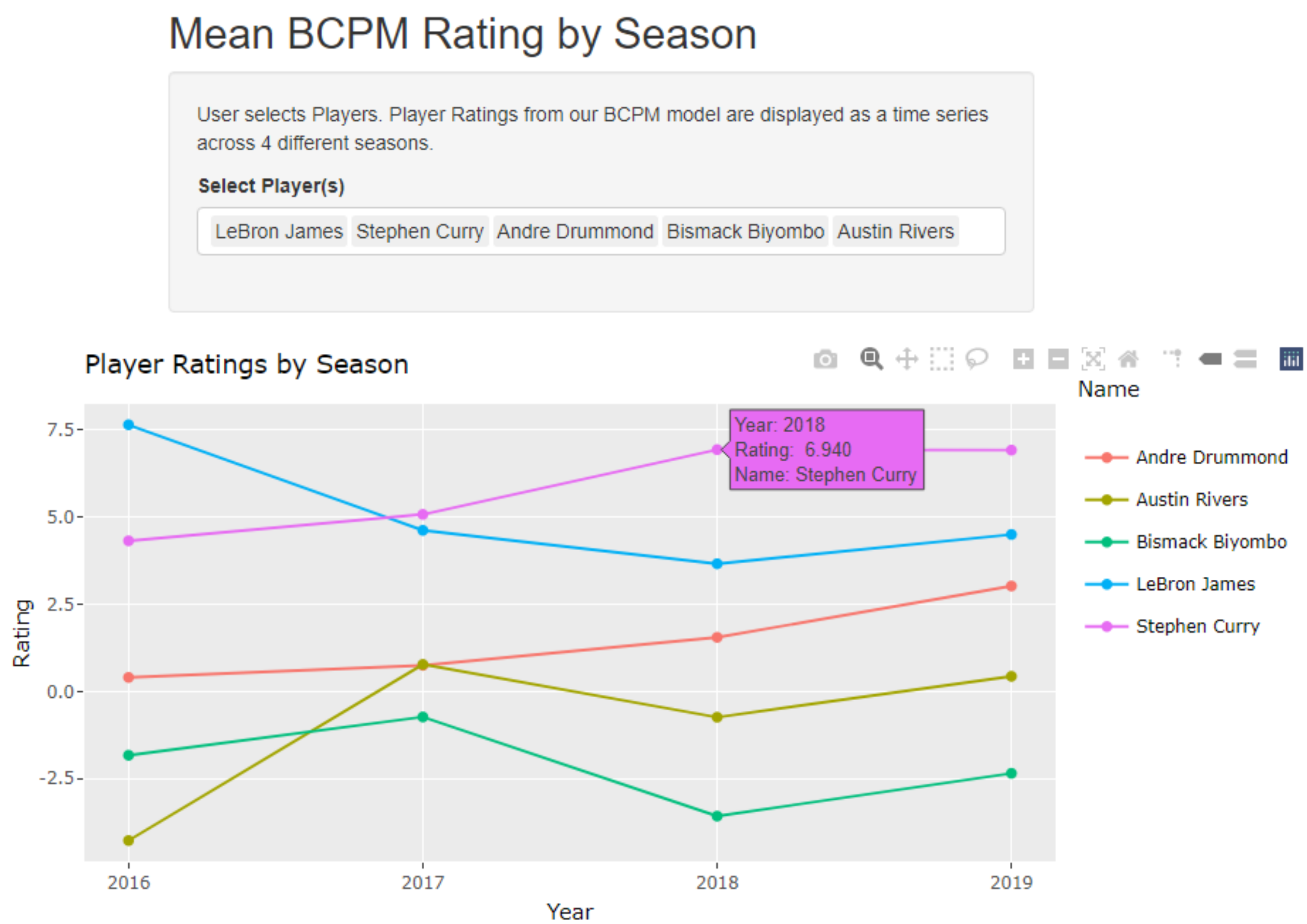


Figure 4: Mean BCPM by Season for Select Players

In Figure 4, we have an example output of the time series visualization. We see that the superstars of the league, Lebron James and Stephen Curry are towards the top. Andre Drummon, known for producing great stats on poor teams, is listed as slightly above average, save for the 2018-2019 season when his team, the Detroit Pistons made the playoffs. Austin Rivers, a quintessential average player from since the 2016-2017 season, has an estimated average that hovers around 0. Furthermore, Bismack Biyombo, a below average NBA center, is consistently below the average value of 0.

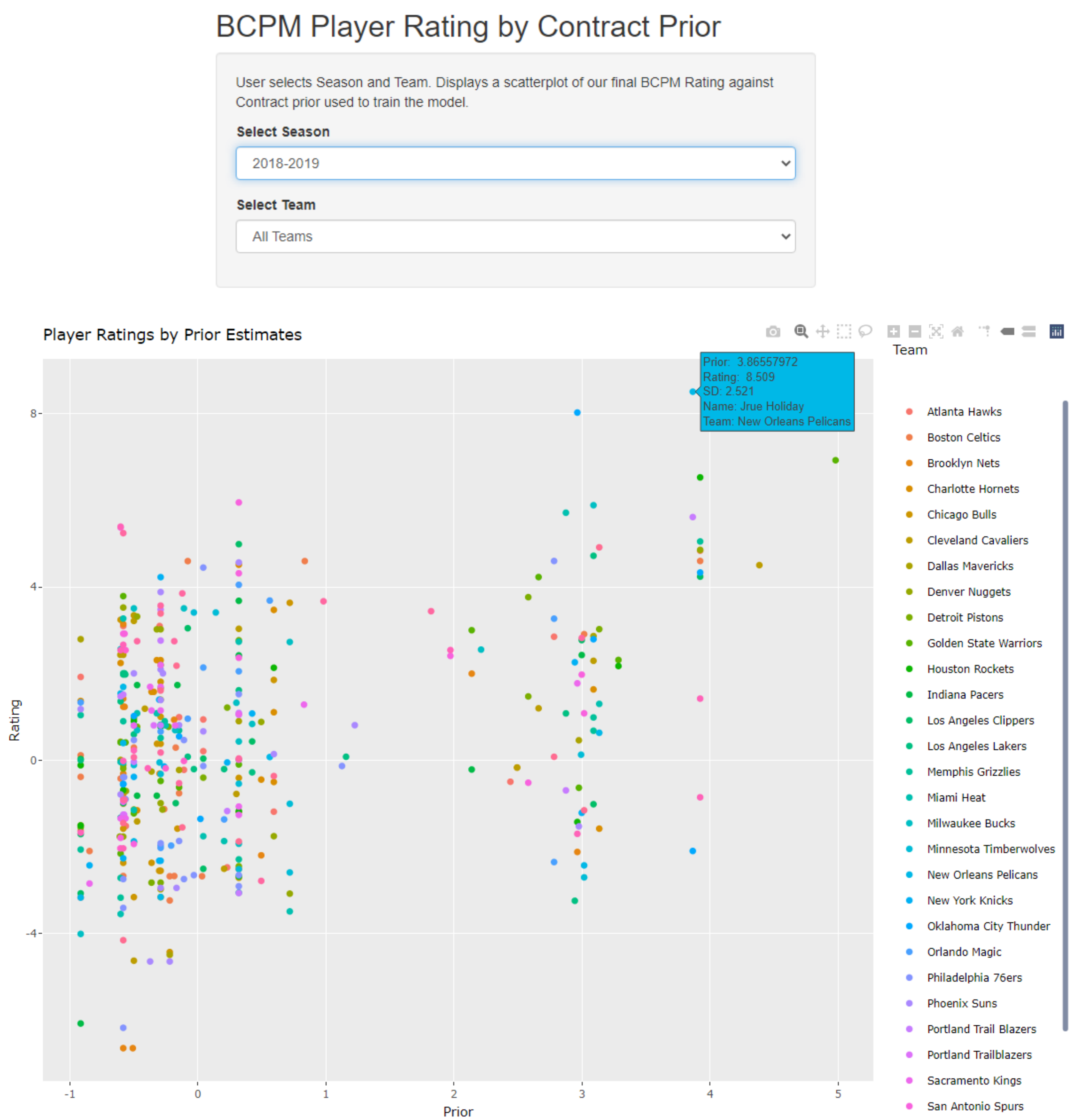


Figure 5: Player Ratings by Prior Estimates, 2018-2019 Season

Figure 5 displays a scatter plot of our final mean BCPM ratings against our model priors. One thing we notice is that there tends to be groups of players with very similar priors. This is to be expected as players tend to be paid similarly in factors of 1 million dollars. For example, a solid role player tends to be paid around 10 to 15 million, while a superstar player will be paid around 30 to 35 million dollars per season. Another promising factor of our model is that for each vertical cluster by prior, there is a wide range of final ratings, meaning that our model does a good job of separating similarly paid players based on performance. The better players tend to be towards the top, while the lesser players are towards the bottom.

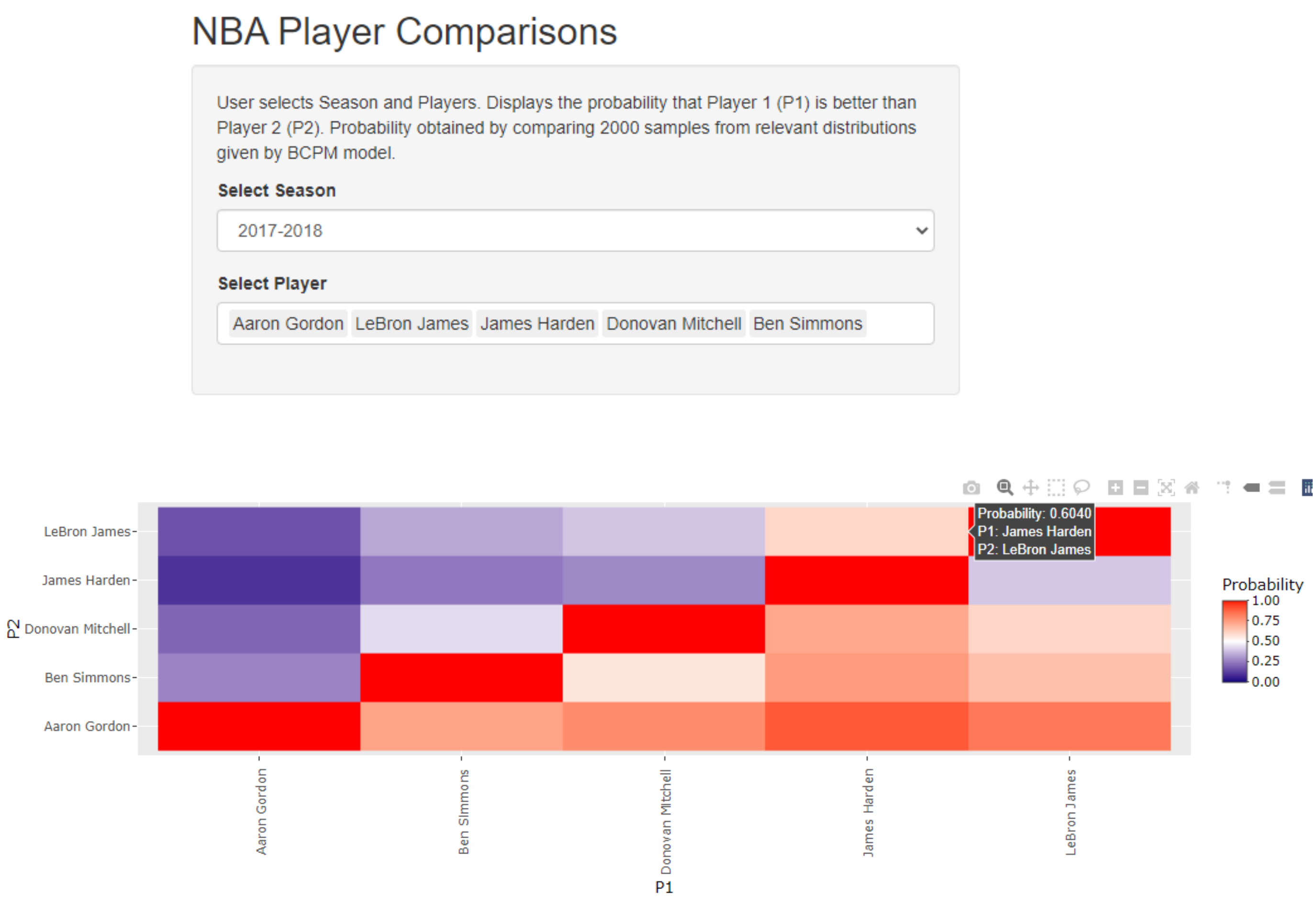


Figure 6: Player 1 vs Player 2 Probabilities, 2017-2018 Season

In Figure 6, we have a visualization showing the probability that the BCPM of Player 1 (P1) is greater than or equal to that of Player 2 (P2). Some interesting headlines of the 2017-2018 season were the race of Most Valuable Player and Rookie of the Year. These races tend to be pretty clear in most seasons, but for this season both races were heavily debated. The race for Most Valuable Player was between James Harden and Lebron James. By our metric, the winner, James Harden, slightly edges Lebron James, with a probability of 0.6040. The Rookie of the Year Race was between Donovan Mitchell and Ben Simmons; by our metric, the winner Ben Simmons had a lower probability of 0.4335 of beating out Donovan Mitchell. It was promising to see that the two candidates for each award were very close in BCPM, while an average player used as a sanity check, Aaron Gordon, was consistently below the other four players.

Discussion

As mentioned previously, one advantage of our method is that it provides a range of plus-minus values for each player, which allows us to account for a range of player performance, as opposed to the point estimates used in conventional methods such as box score plus-minus. The main component of our prior, the contract value, allowed us to adjust expectations for players based on how a team views the player's value. Additionally, we were able to mostly adjust for a teammate's impact on a player's plus-minus by structuring the Bayesian regression such that we regress on all players on the court in a given shift. For example, players who always played with LeBron James, one of the greatest basketball players of all time, but did not produce as much on an individual level had a more muted posterior distribution compared to the box plus-minus. Another factor that negatively impacts existing metrics like box score plus-minus is team rating, as superstar players on bad teams suffer from conventional methods. Despite excluding a team rating variable from any part of the Bayesian or prior models, our resulting BCPM metric seems to do a much better job of accurately assessing players on below-average teams. One such example is Bradley Beal on the Washington Wizards - a terrible team during the 2018-2019 season, who has a box score plus-minus consistent with a decent starter, but has the 13th highest mean by our BCPM metric - a position more consistent with his superstar status. For future work, we would like to incorporate team rating into our prior in a way that produces reasonable results.

One major challenge our team faced was determining how accurate our final posteriors were. Since ranking players is a largely subjective task, there is no ground truth ranking for us to compare our results. When ranking players by the posterior distribution mean, we did find that the league's best players were towards the top, while important role players filled out the top half, with the bottom half being mostly benchwarmers. Comparing our results with ESPN's Real Plus Minus, we see a lot of similarities in the best players; there are anomalies in both metrics, but we believe that by making our standard deviations publicly available, we are able to provide a more flexible metric.

When manually inspecting our results we did notice that while the vast majority of BCPM ratings seemed reasonable, our model does tend to struggle to accurately assess players on their rookie contracts. This result is slightly surprising considering that we did attempt to address this concern by fitting separate prior models for rookies and non-rookies, but evidently this subject needs to be examined further to improve our model's performance on rookie players.

References

1. Deshpande, S. & Jensen, S. 2016. *Estimating an NBA player's impact on his team's chances of winning*. Journal of Quantitative Analytics Sports.
2. Salvatier J., Wiecki T.V., Fonnesbeck C. 2016. *Probabilistic programming in Python using PyMC3*. PeerJ Computer Science 2:e55 DOI: 10.7717/peerj-cs.55.
3. Eightthirtyfour.com. 2019. [online] Available at: <https://eightthirtyfour.com/data>.
4. NBAstuffer. 2021. *Possession in Basketball Explained*. [online] Available at: <https://www.nbastuffer.com/analytics101/possession/>.
5. Spotrac.com. 2019. *NBA*. [online] Available at: <https://www.spotrac.com/nba/>.
6. Kaggle.com. 2017. *NBA - Player Salary (1990-2017)*. [online] Available at: <https://www.kaggle.com/whitefero/nba-player-salary-19902017>.
7. Basketball-Reference.com. 2020. *About Box Plus/Minus (BPM) | Basketball-Reference.com*. [online] Available at: <https://www.basketball-reference.com/about/bpm2.html>.
8. GitHub. 2019. *fivethirtyeight/data*. [online] Available at: <https://github.com/fivethirtyeight/data/tree/master/nba-forecasts>.
9. ESPN.com. 2021. *NBA Real Plus-Minus - National Basketball Association - ESPN*. [online] Available at: <http://www.espn.com/nba/statistics/rpm/_/year/2019>.

Appendix: Application Code

Below is the entirety of our application code. It involves three major parts: reading and formatting the data, formatting the User Interface, and implementing the interactive displays with ggplot and plotly.

```
library(shiny)
library(plotly)
library(ggplot2)

# results <- read.csv("data/bayesian_results_df.csv")
# results <- results[3:nrow(results)-1,]
# player_names <- read.csv("data/player_index_map.csv")
# results$names <- player_names$player_name
# priors_vet <- read.csv("data/Ridge_Priors+SE_2017_nonrookie.csv")
# priors_rookie <- read.csv("data/Ridge_Priors+SE_2017_rookie.csv")
# priors_all <- rbind(priors_vet, priors_rookie)
# results_merged <- merge(results, priors_all, by.x = "names", by.y = "name")
# results <- results_merged[,c("names", "Team", "mean", "sd.x", "finalpriors")]
# names(results) <- c("Name", "Team", "Rating", "SD", "Prior")

#Read and format results
results_16 <- read.csv("data/bayesian_results_df_2015_16.csv")
results_17 <- read.csv("data/bayesian_results_df_2016_17.csv")
results_18 <- read.csv("data/bayesian_results_df_2017_18.csv")
results_19 <- read.csv("data/bayesian_results_df_2018_19.csv")
results_16 <- results_16[3:nrow(results_16)-1,]
results_17 <- results_17[3:nrow(results_17)-1,]
results_18 <- results_18[3:nrow(results_18)-1,]
results_19 <- results_19[3:nrow(results_19)-1,]

#Read and format indices, add to results
player_names_16 <- read.csv("data/player_index_map_2015-16.csv")
player_names_17 <- read.csv("data/player_index_map_2016-17.csv")
player_names_18 <- read.csv("data/player_index_map_2017-18.csv")
player_names_19 <- read.csv("data/player_index_map.csv")
results_16$names <- player_names_16$player_name
results_17$names <- player_names_17$player_name
results_18$names <- player_names_18$player_name
results_19$names <- player_names_19$player_name

#read and format priors 2015-2016
priors_vet_16 <- read.csv("data/final_priors_vets_2015_16.csv")
priors_rookie_16 <- read.csv("data/final_priors_rookies_2015_16.csv")
priors_all_16 <- rbind(priors_vet_16, priors_rookie_16)
results_merged_16 <- merge(results_16, priors_all_16,
                           by.x = "names", by.y = "name")
results_16 <-
  results_merged_16[,c("names", "Team", "mean", "sd.x", "finalpriors")]
```

```
names(results_16) <- c("Name", "Team", "Rating", "SD", "Prior")

#read and format priors 2016-2017
priors_vet_17 <- read.csv("data/final_priors_vets_2016_17.csv")
priors_rookie_17 <- read.csv("data/final_priors_rookies_2016_17.csv")
priors_all_17 <- rbind(priors_vet_17, priors_rookie_17)
results_merged_17 <- merge(results_17, priors_all_17,
                           by.x = "names", by.y = "name")
results_17 <-
  results_merged_17[,c("names", "Team", "mean", "sd.x", "finalpriors")]
names(results_17) <- c("Name", "Team", "Rating", "SD", "Prior")

#read and format priors 2017-2018
priors_vet_18 <- read.csv("data/final_priors_vets_2017_18.csv")
priors_rookie_18 <- read.csv("data/final_priors_rookies_2017_18.csv")
priors_all_18 <- rbind(priors_vet_18, priors_rookie_18)
results_merged_18 <- merge(results_18, priors_all_18,
                           by.x = "names", by.y = "name")
results_18 <-
  results_merged_18[,c("names", "Team", "mean", "sd.x", "finalpriors")]
names(results_18) <- c("Name", "Team", "Rating", "SD", "Prior")

#read and format priors 2018-2019
priors_vet_19 <- read.csv("data/final_priors_vets_2018_19.csv")
priors_rookie_19 <- read.csv("data/final_priors_rookies_2018_19.csv")
priors_rookie_19 <- priors_rookie_19[, c("name", "finalpriors", "Team")]
priors_vet_19 <- priors_vet_19[,c("name", "finalpriors", "Team")]
priors_all_19 <- rbind(priors_vet_19, priors_rookie_19)
results_merged_19 <- merge(results_19, priors_all_19,
                           by.x = "names", by.y = "name")
results_19 <-
  results_merged_19[,c("names", "Team", "mean", "sd", "finalpriors")]
names(results_19) <- c("Name", "Team", "Rating", "SD", "Prior")

#format data for times series
results_16$Year <- rep(2016, nrow(results_16))
results_17$Year <- rep(2017, nrow(results_17))
results_18$Year <- rep(2018, nrow(results_18))
results_19$Year <- rep(2019, nrow(results_19))
all_players <- rbind(results_16, results_17, results_18, results_19)
all_players <- all_players[,c("Name", "Year", "Rating")]
teams <- sort(unique(results_16$Team))
teams <- c("All Teams", teams)

#samples data for matrix
samples_16 <- read.csv("data/bayesian_posterior_samples_2015_16.csv")
samples_16 <- samples_16[, -1]
names(samples_16) <- player_names_16$player_name

samples_17 <- read.csv("data/bayesian_posterior_samples_2016_17.csv")
samples_17 <- samples_17[, -1]
names(samples_17) <- player_names_17$player_name
```

```r
samples_18 <- read.csv("data/bayesian_posterior_samples_2017_18.csv")
samples_18 <- samples_18[, -1]
names(samples_18) <- player_names_18$player_name

samples_19 <- read.csv("data/bayesian_posterior_samples_2018_19.csv")
samples_19 <- samples_19[, -1]
names(samples_19) <- player_names_19$player_name


ui <- navbarPage("NBA Project Visualizations with BCPM Rating",
        tabPanel("Player Distributions",
          titlePanel("NBA Player Distributions According to BCPM"),
          sidebarLayout(
            # Sidebar panel for inputs ----
            sidebarPanel(
              p("User Selects Season and Players. Player distribution, assumed
                to be normal, is displayed. Mean and Standard Deviation
                from our BCPM model."),
              selectInput(
                inputId = "select_season",
                label = "Select Season",
                choices = c('2015-2016', '2016-2017', '2017-2018', '2018-2019'),
                selected = NULL,
                multiple = FALSE,
                selectize = FALSE,
                width = NULL,
                size = NULL
              ),
              conditionalPanel(
                condition = "input.select_season == '2015-2016'",
                selectInput(
                  inputId = "select_players_16",
                  label = "Select Player",
                  choices = results_16$Name,
                  selected = NULL,
                  multiple = TRUE,
                  selectize = TRUE,
                  width = NULL,
                  size = NULL
                )
              ),
              conditionalPanel(
                condition = "input.select_season == '2016-2017'",
                selectInput(
                  inputId = "select_players_17",
                  label = "Select Player",
                  choices = results_17$Name,
                  selected = NULL,
                  multiple = TRUE,
                  selectize = TRUE,
                  width = NULL,
                  size = NULL
                )
```

```
          ),
          conditionalPanel(
            condition = "input.select_season == '2017-2018'",
            selectInput(
              inputId = "select_players_18",
              label = "Select Player",
              choices = results_18$Name,
              selected = NULL,
              multiple = TRUE,
              selectize = TRUE,
              width = NULL,
              size = NULL
            )
          ),
        conditionalPanel(
          condition = "input.select_season == '2018-2019'",
          selectInput(
            inputId = "select_players_19",
            label = "Select Player",
            choices = results_19$Name,
            selected = NULL,
            multiple = TRUE,
            selectize = TRUE,
            width = NULL,
            size = NULL
          )
        )
      ),
      mainPanel(
        plotOutput(outputId = "plot"),
        verbatimTextOutput("player_info")
      )
    )
  ),
  tabPanel("Mean Rating by Season",
           titlePanel("Mean BCPM Rating by Season"),
           sidebarLayout(

             # Sidebar panel for inputs ----
             sidebarPanel(
               p("User selects Players. Player Ratings from our BCPM model
                 are displayed as a time series across 4
                 different seasons."),
               selectInput(
                 inputId = "select_players_timeline",
                 label = "Select Player(s)",
                 choices = unique(all_players$Name),
                 selected = NULL,
                 multiple = TRUE,
                 selectize = TRUE,
                 width = NULL,
                 size = NULL
               )
```

```
            ),
            mainPanel(
              plotlyOutput(outputId = "player_timeline")
            )
          )
  ),
  tabPanel("Ratings by Prior",
          titlePanel("BCPM Player Rating by Contract Prior"),
          sidebarLayout(

            # Sidebar panel for inputs ----
            sidebarPanel(
              p("User selects Season and Team. Displays a scatterplot
                of our final BCPM Rating against Contract prior
                used to train the model."),
              selectInput(
                inputId = "select_season_s",
                label = "Select Season",
                choices = c('2015-2016', '2016-2017', '2017-2018', '2018-2019'),
                selected = NULL,
                multiple = FALSE,
                selectize = FALSE,
                width = NULL,
                size = NULL
              ),
              selectInput(
                inputId = "select_team",
                label = "Select Team",
                choices = teams,
                selected = NULL,
                multiple = FALSE,
                selectize = FALSE,
                width = NULL,
                size = NULL
              )
            ),
            mainPanel(
              plotlyOutput("plot_scatter", height = 900, width = 1200)
            )
          )
  ),
  tabPanel("Player Matrix",
          titlePanel("NBA Player Comparisons"),
          sidebarLayout(

            # Sidebar panel for inputs ----
            sidebarPanel(
              p("User selects Season and Players.
                Displays the probability that Player 1 (P1) is better than
                Player 2 (P2). Probability obtained by comparing 2000
                samples from relevant distributions given by BCPM model."),
              selectInput(
                inputId = "select_season_m",
```

```
                    label = "Select Season",
                    choices = c('2015-2016', '2016-2017', '2017-2018', '2018-2019'),
                    selected = NULL,
                    multiple = FALSE,
                    selectize = FALSE,
                    width = NULL,
                    size = NULL
                  ),
                  conditionalPanel(
                    condition = "input.select_season_m == '2015-2016'",
                    selectInput(
                      inputId = "select_players_16_m",
                      label = "Select Player",
                      choices = results_16$Name,
                      selected = NULL,
                      multiple = TRUE,
                      selectize = TRUE,
                      width = NULL,
                      size = NULL
                    )
                  ),
                  conditionalPanel(
                    condition = "input.select_season_m == '2016-2017'",
                    selectInput(
                      inputId = "select_players_17_m",
                      label = "Select Player",
                      choices = results_17$Name,
                      selected = NULL,
                      multiple = TRUE,
                      selectize = TRUE,
                      width = NULL,
                      size = NULL
                    )
                  ),
                  conditionalPanel(
                    condition = "input.select_season_m == '2017-2018'",
                    selectInput(
                      inputId = "select_players_18_m",
                      label = "Select Player",
                      choices = results_18$Name,
                      selected = NULL,
                      multiple = TRUE,
                      selectize = TRUE,
                      width = NULL,
                      size = NULL
                    )
                  ),
                  conditionalPanel(
                    condition = "input.select_season_m == '2018-2019'",
                    selectInput(
                      inputId = "select_players_19_m",
                      label = "Select Player",
                      choices = results_19$Name,
```

```
                        selected = NULL,
                        multiple = TRUE,
                        selectize = TRUE,
                        width = NULL,
                        size = NULL
                      )
                    )
                  ),
                  mainPanel(
                    plotlyOutput(outputId = "matrix")
                  )
                )
       )

)

server <- function(input, output) {
  #normal dist output
  output$plot <- renderPlot({
    if(input$select_season == '2015-2016') {
      results = results_16
      filtered_res <- results[results$Name %in% input$select_players_16,]
      xlow = min(filtered_res$Rating - 3*filtered_res$SD) - 0.5
      xhigh = max(filtered_res$Rating + 3*filtered_res$SD) + 0.5
      g <- ggplot(filtered_res) +
        xlim(xlow,xhigh)
    }
    else if (input$select_season == '2016-2017'){
      results = results_17
      filtered_res <- results[results$Name %in% input$select_players_17,]
      xlow = min(filtered_res$Rating - 3*filtered_res$SD) - 0.5
      xhigh = max(filtered_res$Rating + 3*filtered_res$SD) + 0.5
      g <- ggplot(filtered_res) +
        xlim(xlow,xhigh)
    }
    else if (input$select_season == '2018-2019'){
      results = results_19
      filtered_res <- results[results$Name %in% input$select_players_19,]
      xlow = min(filtered_res$Rating - 3*filtered_res$SD) - 0.5
      xhigh = max(filtered_res$Rating + 3*filtered_res$SD) + 0.5
      g <- ggplot(filtered_res) +
        xlim(xlow,xhigh)
    }
    else {
      results = results_18
      filtered_res <- results[results$Name %in% input$select_players_18,]
      xlow = min(filtered_res$Rating - 3*filtered_res$SD) - 0.5
      xhigh = max(filtered_res$Rating + 3*filtered_res$SD) + 0.5
      g <- ggplot(filtered_res) +
        xlim(xlow,xhigh)
    }
```

```
  if(nrow(filtered_res > 0))
  {

    for(i in 1:nrow(filtered_res))
    {
      g <- g + stat_function(fun = dnorm,
                             args = list(mean = filtered_res$Rating[i],
                                         sd = filtered_res$SD[i]),
                             aes(color = !!filtered_res$Name[i]))
    }
  }
  g <- g + labs(color='Player') + xlab("Rating") + ylab("Y")

  g
  })

#timeline plot
output$player_timeline <- renderPlotly({
  filtered_timeline <-
    all_players[all_players$Name %in% input$select_players_timeline,]
  if(nrow(filtered_timeline) == 0){
    p <- ggplot(filtered_timeline, aes(Year, Rating, color = Name))
  }
  else{
  p <- ggplot(filtered_timeline, aes(Year, Rating, color = Name)) +
    geom_line() +
    geom_point() +
    ggtitle("Player Ratings by Season") +
    scale_x_continuous(breaks = c(2016,2017,2018,2019),
                       labels = c("2016","2017","2018","2019"))
  }
  ggplotly(p)
})

#scatter plot of rating vs prior
output$plot_scatter <- renderPlotly({
  if(input$select_season_s == '2015-2016') results = results_16
  else if (input$select_season_s == '2016-2017') results = results_17
  else if (input$select_season_s == '2018-2019') results = results_19
  else results = results_18
  if(input$select_team == "All Teams") results = results
  else{
    results = results[results$Team == input$select_team,]
  }
  p_scatter <-
    ggplot(results, aes(Prior, Rating, label = SD, label2 = Name,
                        color = Team)) + geom_point() +
    ggtitle("Player Ratings by Prior Estimates")
  ggplotly(p_scatter)
})

#probability matrix
output$matrix <- renderPlotly({
```

```
  getPlayerProb <- function(player1, player2, samples){
    sum(samples[,player1] >= samples[,player2])/nrow(samples)
  }
  if(input$select_season_m == '2015-2016'){
    samples = samples_16
    selected_names = input$select_players_16_m
  }
  else if (input$select_season_m == '2016-2017'){
    samples = samples_17
    selected_names = input$select_players_17_m
  }
  else if (input$select_season_m == '2018-2019'){
    samples = samples_19
    selected_names = input$select_players_19_m
  }
  else {
    samples = samples_18
    selected_names = input$select_players_18_m
  }
  test <- data.frame(matrix(NA, nrow = length(selected_names)^2, ncol = 3))
  names(test) <- c("P1", "P2", "Probability")

  curr_row <- 1
  for(p1 in selected_names){
    for(p2 in selected_names){
      test[curr_row, "P1"] = p1
      test[curr_row, "P2"] = p2
      test[curr_row, "Probability"] = getPlayerProb(p1, p2, samples)
      curr_row = curr_row+1
    }
  }

  #test <- test[order(test$P2, test$P1, decreasing = TRUE), ]
  if(nrow(test) == 0) g <- ggplot(test, aes(x = P1, y = P2))
  else{
    g <-  ggplot(test, aes(x = P1, y = P2)) +
      geom_tile(aes(fill = Probability)) +
      theme(axis.text.x = element_text(angle = 90)) +
      scale_fill_gradient2(low="navy", mid="white", high="red",
                           midpoint=0.5, limits=c(0,1))
  }
  ggplotly(g)
})

output$player_info <- renderPrint({
  if(input$select_season == '2015-2016') {
    results = results_16
    filtered_res <- results[results$Name %in% input$select_players_16,]
  }
  else if (input$select_season == '2016-2017'){
    results = results_17
    filtered_res <- results[results$Name %in% input$select_players_17,]
  }
```

```
    else if (input$select_season == '2018-2019'){
      results = results_19
      filtered_res <- results[results$Name %in% input$select_players_19,]
    }
    else {
      results = results_18
      filtered_res <- results[results$Name %in% input$select_players_18,]
    }
    filtered_res[,c("Name","Team","Rating","SD")]
  })

}


shinyApp(ui = ui, server = server)
```

prior_model_selection_2015_16

May 17, 2021

1 Prior Model Selection

This notebook will perform model selection via cross validation for our prior distributions. Models to be considered are: * Random Forest Regression (covariates: team rating and contract value) * Random Forest Regression (covariates: contract value only) * Gradient Boosting Regressor (covariates: team rating and contract value) * Gradient Boosting Regressor (covariates: contract value only)

[126]:
```python
import pandas as pd
import numpy as np

# read in all our training data

# MAIN training set for after we've validated
main_train_rookies = pd.read_csv("../data/pre_2015_16/main_train_rookies.csv")
main_train_rookies.drop(main_train_rookies.columns[0], axis = 1, inplace = True)

main_train_vets = pd.read_csv("../data/pre_2015_16/main_train_vets.csv")
main_train_vets.drop(main_train_vets.columns[0], axis = 1, inplace = True)

# training set before validation
train_rookies = pd.read_csv("../data/pre_2015_16/train_rookies.csv")
train_rookies.drop(train_rookies.columns[0], axis = 1, inplace = True)

train_vets = pd.read_csv("../data/pre_2015_16/train_vets.csv")
train_vets.drop(train_vets.columns[0], axis = 1, inplace = True)

# validation dataset
validate_rookies = pd.read_csv("../data/pre_2015_16/validate_rookies.csv")
validate_rookies.drop(validate_rookies.columns[0], axis = 1, inplace = True)

validate_vets = pd.read_csv("../data/pre_2015_16/validate_vets.csv")
validate_vets.drop(validate_vets.columns[0], axis = 1, inplace = True)
```

Define x and y variables for model fitting.

NOTE - the 1 in the variable name indicates that team rating is included as a covariate. When team rating is not included as a covariate, the variable names will have a 2 at the end.

[277]:
```
# FIRST - with team rating included as a covariate

# x and y for training
x_rookies1 = np.array(train_rookies[['rating', 'mu']])
y_rookies = np.array(train_rookies['coefs'])

x_vets1 = np.array(train_vets[['rating', 'mu']])
y_vets = np.array(train_vets['coefs'])

# x and y for validation
x_rookies_validate1 = np.array(validate_rookies[['rating', 'mu']])
y_rookies_validate = np.array(validate_rookies['coefs'])

x_vets_validate1 = np.array(validate_vets[['rating', 'mu']])
y_vets_validate = np.array(validate_vets['coefs'])

# SECOND - without team rating as a covariate
# Note that we don't need to change the y variables since they stay the same␣
↪regardless of the covariates
x_rookies2 = np.array(train_rookies['mu']).reshape(-1, 1)
x_vets2 = np.array(train_vets['mu']).reshape(-1, 1)
x_rookies_validate2 = np.array(validate_rookies['mu']).reshape(-1, 1)
x_vets_validate2 = np.array(validate_vets['mu']).reshape(-1, 1)

# Now create dataset for main training sets
x_main_rookies = np.array(main_train_rookies['mu']).reshape(-1, 1)
y_main_rookies = np.array(main_train_rookies['coefs'])
x_main_vets = np.array(main_train_vets['mu']).reshape(-1, 1)
y_main_vets = np.array(main_train_vets['coefs']).reshape(-1, 1)
```

1.1 Now Model Training

We will train and validate 4 models for rookies and vets (so 8 models total) - random forest with and without team rating as a covariate (2 models), and gradient boosting regressor with and without team rating as a covariate (2 models). We will select the model for rookies and vets that performs best on our validation data, then we will retrain that chosen model on ALL the data to get priors for the 2015/16 NBA season.

1.1.1 First Random Forest Models

A note on whether or not team rating boosts model performance - initially, based on only the random forest models, it appears that the models perform very slightly better on validation data WITHOUT team rating as a covariate. We will investigate this in gradient boosting as well, but if we see similar results there we will officially drop team rating as a covariate since it doesn't seem to be helping at all and it needlessly increases model complexity.

1.1.2 Best RF Model for Rookies: random forest with optimized hyperparameters without team rating (MSE 15.21)

This seems to give the most intuitively reasonable results with Kyrie Irving as the top rookie.

1.1.3 For Veterans: both models look good - we chose optimized params without team rating (MSE 13.6)

Since we prefer the model without team rating for rookies, we will be consistent and choose the model without team rating for veterans as well since both perform similarly anyways.

[206]:
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV

# first for rookies with team rating
rf_rookie1 = RandomForestRegressor()
params = {'max_depth': [2,5,10], 'n_estimators': [50, 100, 200]} # optimize␣
↪over max_depth and number of estimators

rf_rookie1 = GridSearchCV(rf_rookie1, params)

rf_rookie1 = rf_rookie1.fit(x_rookies1, y_rookies)

print(rf_rookie1.best_params_) # print the best parameters so we know what␣
↪we're working with

rf_rookie1 = rf_rookie1.best_estimator_ # set the model to be the best estimator

# Now get predictions on validation set and record MSE
preds_rookie_rf1 = rf_rookie1.predict(x_rookies_validate1)
mse_rf_rookie1 = np.mean((y_rookies_validate - preds_rookie_rf1)**2)
print("MSE Random Forest Rookies Team Rating: ", mse_rf_rookie1)


# Quickly check if random forest with default hyperparameters gives more␣
↪reasonable predictions - answer - not really

# tmp_rf = RandomForestRegressor(max_depth = 2).fit(x_rookies1, y_rookies)
# tmp_preds_rf = tmp_rf.predict(x_rookies_validate1)
# idx = (-tmp_preds_rf).argsort()[:10]
# tmp_preds_rf[idx]
```

[206]:
```
array([1.51133654, 1.51133654, 1.38079844, 1.25516856, 1.24537634,
       1.22266694, 1.2224639 , 1.20892296, 1.16704222, 1.12812231])
```

[171]:
```
idx = (-preds_rookie_rf1).argsort()[:10]
preds_rookie_rf1[idx]
```

[171]: array([1.28946149, 1.21971329, 1.21555909, 1.16758456, 1.08219275,
       1.06770593, 0.95351087, 0.95351087, 0.93633871, 0.80106413])

[172]:
```
validate_rookies.iloc[idx]
```

[172]:
```
       rating                   Team    Type        mu  sd  \
9    6.984504  Oklahoma City Thunder  Rookie  0.734000   5
7    6.984504  Oklahoma City Thunder  Rookie  0.728320   5
13   6.984504  Oklahoma City Thunder  Rookie  1.354000   5
8    6.984504  Oklahoma City Thunder  Rookie  1.898225   5
6    7.952643   Los Angeles Clippers  Rookie  0.813280   5
3    9.804995      San Antonio Spurs  Rookie  0.964686   5
27   5.678120  Golden State Warriors  Rookie  1.025293   5
21   5.678120  Golden State Warriors  Rookie  1.016640   5
19   5.685673        Houston Rockets  Rookie  1.600000   5
144 -3.220032    Cleveland Cavaliers  Rookie  2.356910   5

                  name  player_id  index         player_name     coefs
9           Jeremy Lamb     203087    186         Jeremy Lamb -0.460762
7         Steven Adams     203500    152        Steven Adams -0.639499
13         Dion Waiters     203079     65        Dion Waiters -4.674635
8           Enes Kanter     202683    279         Enes Kanter  0.640074
6        Austin Rivers     203085    174       Austin Rivers -7.834556
3        Kawhi Leonard     202695    144       Kawhi Leonard  4.714348
27       Klay Thompson     202691    115       Klay Thompson  3.562053
21     Harrison Barnes     203084    178     Harrison Barnes  1.313442
19  Kostas Papanikolaou     203123    249 Kostas Papanikolaou -3.362832
144        Kyrie Irving     202681    367        Kyrie Irving  3.501904
```

[189]:
```
# Now rookies without team rating

rf_rookie2 = RandomForestRegressor()
params = {'max_depth': [2,5,10], 'n_estimators': [50, 100, 200]} # optimize␣
↪over max_depth and number of estimators

rf_rookie2 = GridSearchCV(rf_rookie2, params)

rf_rookie2 = rf_rookie2.fit(x_rookies2, y_rookies)

print(rf_rookie2.best_params_) # print the best parameters so we know what␣
↪we're working with

rf_rookie2 = rf_rookie2.best_estimator_ # set the model to be the best estimator

# Now get predictions on validation set and record MSE
preds_rookie_rf2 = rf_rookie2.predict(x_rookies_validate2)
mse_rf_rookie2 = np.mean((y_rookies_validate - preds_rookie_rf2)**2)
```

```python
print("MSE Random Forest Rookies NO Team Rating: ", mse_rf_rookie2)
```

```
{'max_depth': 2, 'n_estimators': 200}
MSE Random Forest Rookies NO Team Rating:  15.213135983382914
```

[273]:
```python
idx = (-preds_rookie_rf2).argsort()[:10]
print(preds_rookie_rf2[idx])
print(min(preds_rookie_rf2))
print(max(preds_rookie_rf2))
```

```
[1.7361497  0.57008504 0.45770079 0.45770079 0.45770079 0.33339012
 0.33339012 0.31222487 0.27037224 0.26797101]
-1.2760203496382783
1.7361497042574094
```

[191]:
```python
validate_rookies.iloc[idx]
```

[191]:
```
        rating                     Team    Type        mu  sd            name  \
144 -3.220032      Cleveland Cavaliers  Rookie  2.356910   5    Kyrie Irving
1    9.804995        San Antonio Spurs  Rookie  0.692333   5     Aron Baynes
9    6.984504    Oklahoma City Thunder  Rookie  0.734000   5     Jeremy Lamb
150 -3.488153          Detroit Pistons  Rookie  0.734790   5  Reggie Jackson
7    6.984504    Oklahoma City Thunder  Rookie  0.728320   5    Steven Adams
36   4.697486    Portland Trailblazers  Rookie  0.807000   5     CJ McCollum
177 -5.228922             Orlando Magic  Rookie  0.799280   5   Elfrid Payton
172 -5.228922             Orlando Magic  Rookie  0.793531   5   Tobias Harris
114 -0.738064           Denver Nuggets  Rookie  0.749923   5  Kenneth Faried
91   0.000000            Atlanta Hawks  Rookie  0.811111   5    Shelvin Mack

     player_id  index     player_name     coefs
144     202681    367    Kyrie Irving  3.501904
1       203382    391     Aron Baynes  4.288611
9       203087    186     Jeremy Lamb -0.460762
150     202704    188  Reggie Jackson -1.362558
7       203500    152    Steven Adams -0.639499
36      203468    247     CJ McCollum -0.638371
177     203901    380   Elfrid Payton -1.442750
172     202699     59   Tobias Harris  1.912663
114     202702    325  Kenneth Faried  1.791871
91      202714    173     Shelvin Mack  2.658083
```

[192]:
```python
# Now vets with team rating

rf_vet1 = RandomForestRegressor()
params = {'max_depth': [2,5,10], 'n_estimators': [50, 100, 200]} # optimize␣
↪over max_depth and number of estimators
```

```
rf_vet1 = GridSearchCV(rf_vet1, params)

rf_vet1 = rf_vet1.fit(x_vets1, y_vets)

print(rf_vet1.best_params_) # print the best parameters so we know what we're␣
↪working with

rf_vet1 = rf_vet1.best_estimator_ # set the model to be the best estimator

# Now get predictions on validation set and record MSE
preds_vet_rf1 = rf_vet1.predict(x_vets_validate1)
mse_rf_vet1 = np.mean((y_vets_validate - preds_vet_rf1)**2)
print("MSE Random Forest Veterans Team Rating: ", mse_rf_vet1)
```

```
{'max_depth': 2, 'n_estimators': 50}
MSE Random Forest Veterans Team Rating:  13.598488981066104
```

[210]:
```
idx = (-preds_vet_rf1).argsort()[:20]
preds_vet_rf1[idx]
```

[210]:
```
array([3.457183  , 3.41072155, 3.34612116, 3.31499142, 3.31499142,
       3.31317354, 3.31317354, 3.31317354, 3.29430166, 3.26040505,
       3.11850324, 3.0928423 , 3.05034568, 2.79658703, 2.79272861,
       2.65164843, 2.61761138, 2.61761138, 2.61761138, 2.59595066])
```

[211]:
```
validate_vets.iloc[idx] # this seems reasonable
```

[211]:

	rating	Team	Type	mu	sd	\
20	7.952643	Los Angeles Clippers	Non-rookie	6.689521	5	
25	6.984504	Oklahoma City Thunder	Non-rookie	6.331875	5	
15	7.952643	Los Angeles Clippers	Non-rookie	5.891537	5	
49	4.843455	Miami Heat	Non-rookie	6.881467	5	
37	5.685673	Houston Rockets	Non-rookie	7.145424	5	
176	-0.755792	New York Knicks	Non-rookie	7.803663	5	
171	-0.755792	New York Knicks	Non-rookie	7.486000	5	
158	-0.489994	Brooklyn Nets	Non-rookie	7.726930	5	
192	-1.422322	Sacramento Kings	Non-rookie	6.439108	5	
160	-0.489994	Brooklyn Nets	Non-rookie	6.584822	5	
127	1.522379	Chicago Bulls	Non-rookie	6.287625	5	
222	-4.658751	Los Angeles Lakers	Non-rookie	7.833333	5	
197	-3.220032	Cleveland Cavaliers	Non-rookie	6.881467	5	
107	2.782586	Memphis Grizzlies	Non-rookie	5.500000	5	
30	6.984504	Oklahoma City Thunder	Non-rookie	5.239687	5	
60	4.697486	Portland Trailblazers	Non-rookie	5.335333	5	
36	5.685673	Houston Rockets	Non-rookie	4.909615	5	
56	4.843455	Miami Heat	Non-rookie	5.000000	5	
44	5.678120	Golden State Warriors	Non-rookie	5.004000	5	

```
159 -0.489994         Brooklyn Nets  Non-rookie  5.239688   5

                  name  player_id  index         player_name      coefs
20          Chris Paul     101108    285          Chris Paul   4.353985
25        Kevin Durant     201142    284        Kevin Durant   7.042239
15       Blake Griffin     201933     76       Blake Griffin   1.336778
49          Chris Bosh       2547     24          Chris Bosh   1.858730
37       Dwight Howard       2730    105       Dwight Howard   6.055062
176   Amar'e Stoudemire      2405     32   Amar'e Stoudemire   4.285844
171     Carmelo Anthony      2546    394     Carmelo Anthony   8.285201
158         Joe Johnson      2207    478         Joe Johnson   4.187927
192            Rudy Gay    200752    349            Rudy Gay   1.973265
160      Deron Williams    101114    316      Deron Williams   1.616904
127        Derrick Rose    201565     79        Derrick Rose   3.908301
222         Kobe Bryant       977      6         Kobe Bryant   2.311105
197        LeBron James      2544    165        LeBron James   3.792951
107       Zach Randolph      2216    131       Zach Randolph   6.285065
30    Russell Westbrook    201566    451   Russell Westbrook   2.895779
60    LaMarcus Aldridge    200746     55   LaMarcus Aldridge   6.291513
36         James Harden    201935    107        James Harden  12.197839
56          Dwyane Wade      2548    287         Dwyane Wade   2.339740
44            David Lee    101135    339           David Lee   2.191165
159         Brook Lopez    201572    129         Brook Lopez   2.431535
```

[195]:

```
# Now vets without team rating
rf_vet2 = RandomForestRegressor()
params = {'max_depth': [2,5,10], 'n_estimators': [50, 100, 200]} # optimize␣
↪over max_depth and number of estimators

rf_vet2 = GridSearchCV(rf_vet2, params)

rf_vet2 = rf_vet2.fit(x_vets2, y_vets)

print(rf_vet2.best_params_) # print the best parameters so we know what we're␣
↪working with

rf_vet2 = rf_vet2.best_estimator_ # set the model to be the best estimator

# Now get predictions on validation set and record MSE
preds_vet_rf2 = rf_vet2.predict(x_vets_validate2)
mse_rf_vet2 = np.mean((y_vets_validate - preds_vet_rf2)**2)
print("MSE Random Forest Veterans NO Team Rating: ", mse_rf_vet2)
```

```
{'max_depth': 2, 'n_estimators': 50}
MSE Random Forest Veterans NO Team Rating:  13.604154551267278
```

[212]:
```
idx = (-preds_vet_rf2).argsort()[:20]
preds_vet_rf2[idx]
```

[212]:
```
array([3.91736504, 3.91736504, 3.91736504, 3.91736504, 3.77602607,
       3.77602607, 3.77602607, 3.77602607, 3.77602607, 3.77602607,
       3.75199967, 3.75199967, 3.59984226, 3.52905624, 2.28116092,
       2.28116092, 2.28116092, 2.28116092, 2.28116092, 2.28116092])
```

[213]:
```
validate_vets.iloc[idx] # also reasonable
```

[213]:
```
       rating                   Team        Type        mu  sd  \
171 -0.755792        New York Knicks  Non-rookie  7.486000   5
176 -0.755792        New York Knicks  Non-rookie  7.803663   5
222 -4.658751     Los Angeles Lakers  Non-rookie  7.833333   5
158 -0.489994          Brooklyn Nets  Non-rookie  7.726930   5
192 -1.422322       Sacramento Kings  Non-rookie  6.439108   5
37   5.685673        Houston Rockets  Non-rookie  7.145424   5
197 -3.220032    Cleveland Cavaliers  Non-rookie  6.881467   5
49   4.843455             Miami Heat  Non-rookie  6.881467   5
160 -0.489994          Brooklyn Nets  Non-rookie  6.584822   5
20   7.952643   Los Angeles Clippers  Non-rookie  6.689521   5
25   6.984504  Oklahoma City Thunder  Non-rookie  6.331875   5
127  1.522379          Chicago Bulls  Non-rookie  6.287625   5
15   7.952643   Los Angeles Clippers  Non-rookie  5.891537   5
107  2.782586      Memphis Grizzlies  Non-rookie  5.500000   5
93   3.649442         Indiana Pacers  Non-rookie  5.308560   5
103  2.782586      Memphis Grizzlies  Non-rookie  5.276563   5
159 -0.489994          Brooklyn Nets  Non-rookie  5.239688   5
30   6.984504  Oklahoma City Thunder  Non-rookie  5.239687   5
60   4.697486  Portland Trailblazers  Non-rookie  5.335333   5
199 -3.220032    Cleveland Cavaliers  Non-rookie  5.239687   5

                  name  player_id  index        player_name     coefs
171    Carmelo Anthony       2546    394    Carmelo Anthony  8.285201
176  Amar'e Stoudemire       2405     32  Amar'e Stoudemire  4.285844
222        Kobe Bryant        977      6        Kobe Bryant  2.311105
158        Joe Johnson       2207    478        Joe Johnson  4.187927
192           Rudy Gay     200752    349           Rudy Gay  1.973265
37       Dwight Howard       2730    105      Dwight Howard  6.055062
197       LeBron James       2544    165       LeBron James  3.792951
49          Chris Bosh       2547     24         Chris Bosh  1.858730
160     Deron Williams     101114    316     Deron Williams  1.616904
20          Chris Paul     101108    285         Chris Paul  4.353985
25        Kevin Durant     201142    284       Kevin Durant  7.042239
127       Derrick Rose     201565     79       Derrick Rose  3.908301
15        Blake Griffin    201933     76      Blake Griffin  1.336778
107       Zach Randolph      2216    131      Zach Randolph  6.285065
```

```
93          Paul George     202331    487         Paul George   2.510271
103          Marc Gasol     201188     31          Marc Gasol   6.091420
159          Brook Lopez    201572    129         Brook Lopez   2.431535
30    Russell Westbrook     201566    451   Russell Westbrook   2.895779
60    LaMarcus Aldridge     200746     55   LaMarcus Aldridge   6.291513
199          Kevin Love     201567    472          Kevin Love   7.897965
```

1.2 Now Gradient Boosting Regressor

NOTE - it appears that the model gives MUCH more reasonable estimates when we do not optimize over some of the hyperparameters bur rather stick with the defaults.

- When we optimize hyperparameters for rookies with team ratings, the relative ordering of rookies seems somewhat ok but the magnitudes of the estimates are far too low.
- When we optimize hyperparameters WITHOUT team rating as a covariate, the relative ordering of rookies seems actually better than when we do not optimize; however, we see very small magnitude of estimates again which is a problem.
- For VETERANS - the best model was when we optimized hyperparameters and excluded team ratings as a covariate. This gave the most reasonable intuitive ordering of top 20 players, but the magnitudes were a bit small again. Perhaps we could just settle for this and then scale up the magnitudes according to the magnitudes of coefs. Or just leave it as is and let the Bayesian model do the rest

1.2.1 For Veterans best GBR model - optimized without team ratings (MSE 13.75)

1.2.2 For rookies best GBR model - simple model without team ratings (MSE 15.86)

[265]:
```
from sklearn.ensemble import GradientBoostingRegressor

# first for rookies with team rating

# magnitudes seem good (fairly large), but ordering seems a bit suspect

gbr_rookie1 = GradientBoostingRegressor().fit(x_rookies1, y_rookies)
preds_rookie_gbr1 = gbr_rookie1.predict(x_rookies_validate1)

mse_gbr_rookie1 = np.mean((y_rookies_validate - preds_rookie_gbr1)**2)
print("MSE Gradient Boosting Rookies Team Rating: ", mse_gbr_rookie1)

idx = (-preds_rookie_gbr1).argsort()[:20]
preds_rookie_gbr1[idx]
```

```
# attempting to optimize hyperparameters:

# magnitudes are now very small. Only one positive player, the rest negative.␣
↪This seems bad.
# ordering seems somewhat acceptable but the magnitudes are just way too␣
↪problematic.

# gbr_rookie1 = GradientBoostingRegressor()

# params = {'learning_rate': [0.001, 0.01, 0.1],
#           'subsample': [1, 0.9],
#           'max_depth': [2,5,10],
#           'n_estimators': [50, 100, 200]}

# gbr_rookie1 = GridSearchCV(gbr_rookie1, params)

# gbr_rookie1 = gbr_rookie1.fit(x_rookies1, y_rookies)

# print(gbr_rookie1.best_params_) # print the best parameters so we know what␣
↪we're working with

# gbr_rookie1 = gbr_rookie1.best_estimator_ # set the model to be the best␣
↪estimator

# # Now get predictions on validation set and record MSE
# preds_rookie_gbr1 = gbr_rookie1.predict(x_rookies_validate1)
# mse_gbr_rookie1 = np.mean((y_rookies_validate - preds_rookie_gbr1)**2)
# print("MSE Gradient Boosting Rookies Team Rating: ", mse_gbr_rookie1)
```

```
MSE Gradient Boosting Rookies Team Rating:  16.699630349352667
```

[265]:

```
array([5.47454242, 4.809936  , 4.809936  , 3.8415315 , 3.59333701,
       3.59333701, 3.38100247, 3.17798228, 3.14560827, 3.14560827,
       2.44505093, 2.44505093, 2.07707618, 2.01697625, 1.76841536,
       1.72551666, 1.53484181, 1.45777023, 1.25750733, 1.14763982])
```

[266]:

```
idx = (-preds_rookie_gbr1).argsort()[:20]
print(preds_rookie_gbr1[idx])
print(min(preds_rookie_gbr1))
print(max(preds_rookie_gbr1)) # these are reasonable
```

```
[5.47454242 4.809936   4.809936   3.8415315  3.59333701 3.59333701
 3.38100247 3.17798228 3.14560827 3.14560827 2.44505093 2.44505093
 2.07707618 2.01697625 1.76841536 1.72551666 1.53484181 1.45777023
 1.25750733 1.14763982]
-4.501009855233326
```

```
5.474542416647444
```

[267]:
```
validate_rookies.iloc[idx]
```

[267]:
```
       rating                    Team    Type        mu  sd  \
8    6.984504   Oklahoma City Thunder  Rookie  1.898225   5
21   5.678120   Golden State Warriors  Rookie  1.016640   5
27   5.678120   Golden State Warriors  Rookie  1.025293   5
19   5.685673         Houston Rockets  Rookie  1.600000   5
50   3.756517  Minnesota Timberwolves  Rookie  1.836880   5
42   3.756517  Minnesota Timberwolves  Rookie  1.854640   5
6    7.952643    Los Angeles Clippers  Rookie  0.813280   5
13   6.984504   Oklahoma City Thunder  Rookie  1.354000   5
60   3.692790            Phoenix Suns  Rookie  0.981074   5
59   3.692790            Phoenix Suns  Rookie  0.996413   5
7    6.984504   Oklahoma City Thunder  Rookie  0.728320   5
9    6.984504   Oklahoma City Thunder  Rookie  0.734000   5
56   3.692790            Phoenix Suns  Rookie  1.216640   5
49   3.756517  Minnesota Timberwolves  Rookie  1.690229   5
105 -0.489994           Brooklyn Nets  Rookie  0.511280   5
33   4.697486   Portland Trailblazers  Rookie  1.004504   5
116 -0.738064          Denver Nuggets  Rookie  0.506400   5
3    9.804995       San Antonio Spurs  Rookie  0.964686   5
130 -1.300283    New Orleans Pelicans  Rookie  1.869080   5
144 -3.220032     Cleveland Cavaliers  Rookie  2.356910   5

                   name  player_id  index         player_name      coefs
8           Enes Kanter     202683    279         Enes Kanter   0.640074
21      Harrison Barnes     203084    178     Harrison Barnes   1.313442
27        Klay Thompson     202691    115       Klay Thompson   3.562053
19   Kostas Papanikolaou    203123    249 Kostas Papanikolaou  -3.362832
50       Andrew Wiggins     203952    175      Andrew Wiggins   1.459406
42      Anthony Bennett     203461     63     Anthony Bennett  -5.261511
6          Austin Rivers     203085    174        Austin Rivers  -7.834556
13          Dion Waiters     203079     65        Dion Waiters  -4.674635
60         Marcus Morris     202694    154       Marcus Morris  -2.672349
59       Markieff Morris     202693    153     Markieff Morris   5.578282
7          Steven Adams     203500    152        Steven Adams  -0.639499
9           Jeremy Lamb     203087    186         Jeremy Lamb  -0.460762
56             Alex Len     203458     44            Alex Len  -5.986509
49          Ricky Rubio     201937    296         Ricky Rubio   4.114858
105     Sergey Karasev     203508    229      Sergey Karasev   3.615371
33        Joel Freeland     200777     53       Joel Freeland  -2.837147
116         Gary Harris     203914    211         Gary Harris  -6.113417
3         Kawhi Leonard     202695    144       Kawhi Leonard   4.714348
130        Anthony Davis     203076     87       Anthony Davis  10.546145
144         Kyrie Irving     202681    367        Kyrie Irving   3.501904
```

[271]:

```python
# Now rookies no team rating

# magnitudes seem far more reasonable. Ordering seems decent. This seems like␣
↪the best option

gbr_rookie2 = GradientBoostingRegressor().fit(x_rookies2, y_rookies)
preds_rookie_gbr2 = gbr_rookie2.predict(x_rookies_validate2)

mse_gbr_rookie2 = np.mean((y_rookies_validate - preds_rookie_gbr2)**2)
print("MSE Gradient Boosting Rookies NO Team Rating: ", mse_gbr_rookie2)

idx = (-preds_rookie_gbr2).argsort()[:20]
preds_rookie_gbr2[idx]


# Attempting to optimize hyperparameters:

# Note - when we optimize hyperparameters, the ordering seems good actually but␣
↪the magnitudes are way too small.
# also - only two players get positive coefficients and the rest have negative.␣
↪This is clearly not ideal.

# gbr_rookie2 = GradientBoostingRegressor()

# params = {'learning_rate': [0.001, 0.01, 0.1],
#           'subsample': [1, 0.9],
#           'max_depth': [2,5,10],
#           'n_estimators': [50, 100, 200]}

# gbr_rookie2 = GridSearchCV(gbr_rookie2, params)

# gbr_rookie2 = gbr_rookie2.fit(x_rookies2, y_rookies)

# print(gbr_rookie2.best_params_) # print the best parameters so we know what␣
↪we're working with

# gbr_rookie2 = gbr_rookie2.best_estimator_ # set the model to be the best␣
↪estimator

# # Now get predictions on validation set and record MSE
# preds_rookie_gbr2 = gbr_rookie2.predict(x_rookies_validate2)
# mse_gbr_rookie2 = np.mean((y_rookies_validate - preds_rookie_gbr2)**2)
# print("MSE Gradient Boosting Rookies NO Team Rating: ", mse_gbr_rookie2)
```

```
MSE Gradient Boosting Rookies NO Team Rating:  15.859333950289404
```

[271]:
```
array([2.61780468, 2.44981099, 2.39852198, 2.27997957, 2.27997957,
       2.24406341, 1.98706712, 1.90017838, 1.90017838, 1.84087184,
       1.61235094, 1.49840397, 1.34810973, 1.34257192, 1.28329678,
       0.98080736, 0.98080736, 0.98080736, 0.88525148, 0.88525148])
```

[269]:
```python
idx = (-preds_rookie_gbr2).argsort()[:20]
print(preds_rookie_gbr2[idx])
print(min(preds_rookie_gbr2))
print(max(preds_rookie_gbr2))
```

```
[2.61780468 2.44981099 2.39852198 2.27997957 2.27997957 2.24406341
 1.98706712 1.90017838 1.90017838 1.84087184 1.61235094 1.49840397
 1.34810973 1.34257192 1.28329678 0.98080736 0.98080736 0.98080736
 0.88525148 0.88525148]
-3.2529270338487772
2.6178046752610786
```

[270]:
```python
validate_rookies.iloc[idx] # this is fairly reasonable, not ideal but not bad
```

[270]:
```
         rating                    Team    Type        mu  sd  \
1      9.804995       San Antonio Spurs  Rookie  0.692333   5
116   -0.738064          Denver Nuggets  Rookie  0.506400   5
82     1.522379           Chicago Bulls  Rookie  0.669583   5
36     4.697486   Portland Trailblazers  Rookie  0.807000   5
177   -5.228922            Orlando Magic  Rookie  0.799280   5
144   -3.220032     Cleveland Cavaliers  Rookie  2.356910   5
56     3.692790            Phoenix Suns  Rookie  1.216640   5
208 -10.015718      Philadelphia 76ers  Rookie  1.226120   5
80     2.771242         Toronto Raptors  Rookie  1.226120   5
8      6.984504   Oklahoma City Thunder  Rookie  1.898225   5
189   -5.593750               Utah Jazz  Rookie  0.062452   5
105   -0.489994           Brooklyn Nets  Rookie  0.511280   5
149   -3.488153         Detroit Pistons  Rookie  0.856120   5
212 -10.015718      Philadelphia 76ers  Rookie  0.403360   5
206 -10.015718      Philadelphia 76ers  Rookie  1.105040   5
9      6.984504   Oklahoma City Thunder  Rookie  0.734000   5
7      6.984504   Oklahoma City Thunder  Rookie  0.728320   5
150   -3.488153         Detroit Pistons  Rookie  0.734790   5
130   -1.300283    New Orleans Pelicans  Rookie  1.869080   5
50     3.756517  Minnesota Timberwolves  Rookie  1.836880   5

                name  player_id  index        player_name     coefs
1        Aron Baynes     203382    391        Aron Baynes  4.288611
116     Gary Harris     203914    211       Gary Harris -6.113417
82     Jimmy Butler     202710    159      Jimmy Butler  4.519813
36       CJ McCollum     203468    247       CJ McCollum -0.638371
177    Elfrid Payton     203901    380     Elfrid Payton -1.442750
```

```
144        Kyrie Irving     202681    367       Kyrie Irving   3.501904
56              Alex Len     203458     44           Alex Len  -5.986509
208     Thomas Robinson     203080    123    Thomas Robinson  -1.546704
80    Jonas Valanciunas     202685    313  Jonas Valanciunas  -2.353275
8           Enes Kanter     202683    279        Enes Kanter   0.640074
189         Jack Cooley     204022    428        Jack Cooley  -3.126129
105      Sergey Karasev     203508    229     Sergey Karasev   3.615371
149      Andre Drummond     203083    101     Andre Drummond  -0.326742
212         Tony Wroten     203100    418        Tony Wroten   3.768908
206        Nerlens Noel     203457     89       Nerlens Noel  -0.503336
9           Jeremy Lamb     203087    186        Jeremy Lamb  -0.460762
7          Steven Adams     203500    152       Steven Adams  -0.639499
150      Reggie Jackson     202704    188     Reggie Jackson  -1.362558
130       Anthony Davis     203076     87      Anthony Davis  10.546145
50       Andrew Wiggins     203952    175     Andrew Wiggins   1.459406
```

[248]:
```
# Now veterans team ratings

# here we get good magnitudes for estimates - relative ordering not great.

gbr_vet1 = GradientBoostingRegressor().fit(x_vets1, y_vets)
preds_vet_gbr1 = gbr_vet1.predict(x_vets_validate1)

mse_gbr_vet1 = np.mean((y_vets_validate - preds_vet_gbr1)**2)
print("MSE Gradient Boosting Veterans Team Rating: ", mse_gbr_vet1)


# attempting to optimize hyperparameters:

# when we optimize parameters here the relative ordering seems pretty good␣
 ↪again, magnitude is ok but still a bit too small

# gbr_vet1 = GradientBoostingRegressor()

# params = {'learning_rate': [0.001, 0.01, 0.1],
#           'subsample': [1, 0.9],
#           'max_depth': [2,5,10],
#           'n_estimators': [50, 100, 200]}

# gbr_vet1 = GridSearchCV(gbr_vet1, params)

# gbr_vet1 = gbr_vet1.fit(x_vets1, y_vets)

# print(gbr_vet1.best_params_) # print the best parameters so we know what␣
 ↪we're working with

# gbr_vet1 = gbr_vet1.best_estimator_ # set the model to be the best estimator
```

```
# # Now get predictions on validation set and record MSE
# preds_vet_gbr1 = gbr_vet1.predict(x_vets_validate1)
# mse_gbr_vet1 = np.mean((y_vets_validate - preds_vet_gbr1)**2)
# print("MSE Gradient Boosting Veterans Team Rating: ", mse_gbr_vet1)
```

```
MSE Gradient Boosting Veterans Team Rating:  15.097709396214167
```

[249]:
```
idx = (-preds_vet_gbr1).argsort()[:20]
print(preds_vet_gbr1[idx])
print(max(preds_vet_gbr1))
print(min(preds_vet_gbr1))
```

```
[8.42984127 8.16005703 7.66943092 7.08430715 7.06589504 6.15097835
 5.46029541 5.19218033 5.07944451 4.86884289 4.76152705 4.31828591
 4.2671245  4.09505441 3.97813703 3.55643397 3.54040484 3.27569912
 3.24408631 3.09649349]
8.429841267709886
-4.140874964168131
```

[250]:
```
validate_vets.iloc[idx] # seems ok (amare stoudemire had a huge contract -␣
↪outlier) - except Lebron isn't top 20, so probably not totally correct
```

[250]:
```
       rating                   Team        Type        mu  sd  \
176 -0.755792        New York Knicks  Non-rookie  7.803663   5
171 -0.755792        New York Knicks  Non-rookie  7.486000   5
158 -0.489994          Brooklyn Nets  Non-rookie  7.726930   5
222 -4.658751     Los Angeles Lakers  Non-rookie  7.833333   5
17   7.952643   Los Angeles Clippers  Non-rookie  0.018591   5
18   7.952643   Los Angeles Clippers  Non-rookie  0.129211   5
25   6.984504  Oklahoma City Thunder  Non-rookie  6.331875   5
11   9.804995      San Antonio Spurs  Non-rookie  0.041701   5
107  2.782586      Memphis Grizzlies  Non-rookie  5.500000   5
192 -1.422322       Sacramento Kings  Non-rookie  6.439108   5
49   4.843455             Miami Heat  Non-rookie  6.881467   5
30   6.984504  Oklahoma City Thunder  Non-rookie  5.239687   5
160 -0.489994          Brooklyn Nets  Non-rookie  6.584822   5
184 -1.300283   New Orleans Pelicans  Non-rookie  4.966313   5
26   6.984504  Oklahoma City Thunder  Non-rookie  4.116667   5
20   7.952643   Los Angeles Clippers  Non-rookie  6.689521   5
15   7.952643   Los Angeles Clippers  Non-rookie  5.891537   5
8    9.804995      San Antonio Spurs  Non-rookie  4.166667   5
127  1.522379          Chicago Bulls  Non-rookie  6.287625   5
19   7.952643   Los Angeles Clippers  Non-rookie  3.813375   5

                  name  player_id  index         player_name     coefs
176  Amar'e Stoudemire       2405     32   Amar'e Stoudemire  4.285844
```

```
171    Carmelo Anthony       2546    394    Carmelo Anthony  8.285201
158        Joe Johnson       2207    478        Joe Johnson  4.187927
222        Kobe Bryant        977      6        Kobe Bryant  2.311105
17       Lester Hudson     201991    320      Lester Hudson  3.148040
18       Dahntay Jones       2563    263      Dahntay Jones -8.197401
25        Kevin Durant     201142    284       Kevin Durant  7.042239
11     Reggie Williams     202130    273    Reggie Williams -2.220307
107      Zach Randolph       2216    131      Zach Randolph  6.285065
192           Rudy Gay     200752    349           Rudy Gay  1.973265
49          Chris Bosh       2547     24         Chris Bosh  1.858730
30   Russell Westbrook     201566    451  Russell Westbrook  2.895779
160     Deron Williams     101114    316     Deron Williams  1.616904
184        Eric Gordon     201569      1        Eric Gordon  0.301299
26         Serge Ibaka     201586    248        Serge Ibaka  3.379049
20          Chris Paul     101108    285         Chris Paul  4.353985
15       Blake Griffin     201933     76      Blake Griffin  1.336778
8          Tony Parker       2225    162        Tony Parker -2.466196
127       Derrick Rose     201565     79       Derrick Rose  3.908301
19      DeAndre Jordan     201599    474     DeAndre Jordan -0.893175
```

[255]:

```python
# Now veterans no team rating

# magnitudes seem good, ordering seems ok but missing lebron in top 20 seems bad

# gbr_vet2 = GradientBoostingRegressor().fit(x_vets2, y_vets)
# preds_vet_gbr2 = gbr_vet2.predict(x_vets_validate2)

# mse_gbr_vet2 = np.mean((y_vets_validate - preds_vet_gbr2)**2)
# print("MSE Gradient Boosting Veterans NO Team Rating: ", mse_gbr_vet2)




# attempting to optimize hyperparameters:

# magnitudes a bit small again, relative ordering seems solid.

gbr_vet2 = GradientBoostingRegressor()

params = {'learning_rate': [0.001, 0.01, 0.1],
          'subsample': [1, 0.9],
          'max_depth': [2,5,10],
          'n_estimators': [50, 100, 200]}

gbr_vet2 = GridSearchCV(gbr_vet2, params)

gbr_vet2 = gbr_vet2.fit(x_vets2, y_vets)
```

```
print(gbr_vet2.best_params_) # print the best parameters so we know what we're␣
↪working with

gbr_vet2 = gbr_vet2.best_estimator_ # set the model to be the best estimator

# Now get predictions on validation set and record MSE
preds_vet_gbr2 = gbr_vet2.predict(x_vets_validate2)
mse_gbr_vet2 = np.mean((y_vets_validate - preds_vet_gbr2)**2)
print("MSE Gradient Boosting Veterans NO Team Rating: ", mse_gbr_vet2)
```

```
{'learning_rate': 0.01, 'max_depth': 2, 'n_estimators': 200, 'subsample': 0.9}
MSE Gradient Boosting Veterans NO Team Rating:  13.74526264802979
```

[256]:
```
idx = (-preds_vet_gbr2).argsort()[:20]
print(preds_vet_gbr2[idx])
print(min(preds_vet_gbr2))
print(max(preds_vet_gbr2))
```

```
[3.3983553  3.02276823 3.02276823 3.02276823 3.02276823 2.93608538
 2.93608538 2.93608538 2.93608538 2.92441884 2.91714446 2.91714446
 2.91714446 2.91714446 2.07440845 2.07440845 2.07440845 2.07440845
 2.07440845 2.07440845]
-1.0444379307401035
3.398355296125337
```

[257]:
```
validate_vets.iloc[idx]
```

[257]:

	rating	Team	Type	mu	sd	\
107	2.782586	Memphis Grizzlies	Non-rookie	5.500000	5	
222	-4.658751	Los Angeles Lakers	Non-rookie	7.833333	5	
158	-0.489994	Brooklyn Nets	Non-rookie	7.726930	5	
176	-0.755792	New York Knicks	Non-rookie	7.803663	5	
171	-0.755792	New York Knicks	Non-rookie	7.486000	5	
160	-0.489994	Brooklyn Nets	Non-rookie	6.584822	5	
192	-1.422322	Sacramento Kings	Non-rookie	6.439108	5	
127	1.522379	Chicago Bulls	Non-rookie	6.287625	5	
25	6.984504	Oklahoma City Thunder	Non-rookie	6.331875	5	
15	7.952643	Los Angeles Clippers	Non-rookie	5.891537	5	
20	7.952643	Los Angeles Clippers	Non-rookie	6.689521	5	
37	5.685673	Houston Rockets	Non-rookie	7.145424	5	
197	-3.220032	Cleveland Cavaliers	Non-rookie	6.881467	5	
49	4.843455	Miami Heat	Non-rookie	6.881467	5	
137	1.428164	Washington Wizards	Non-rookie	4.915333	5	
94	3.649442	Indiana Pacers	Non-rookie	4.966313	5	
239	-5.593750	Utah Jazz	Non-rookie	4.915333	5	
36	5.685673	Houston Rockets	Non-rookie	4.909615	5	

```
184 -1.300283   New Orleans Pelicans  Non-rookie  4.966313   5
77   3.929484       Dallas Mavericks  Non-rookie  4.900000   5

                  name  player_id  index         player_name       coefs
107      Zach Randolph       2216    131       Zach Randolph    6.285065
222        Kobe Bryant        977      6         Kobe Bryant    2.311105
158        Joe Johnson       2207    478         Joe Johnson    4.187927
176  Amar'e Stoudemire       2405     32   Amar'e Stoudemire    4.285844
171     Carmelo Anthony      2546    394     Carmelo Anthony    8.285201
160      Deron Williams    101114    316      Deron Williams    1.616904
192           Rudy Gay     200752    349            Rudy Gay    1.973265
127       Derrick Rose     201565     79        Derrick Rose    3.908301
25        Kevin Durant     201142    284        Kevin Durant    7.042239
15       Blake Griffin     201933     76       Blake Griffin    1.336778
20          Chris Paul     101108    285          Chris Paul    4.353985
37       Dwight Howard       2730    105       Dwight Howard    6.055062
197       LeBron James       2544    165        LeBron James    3.792951
49          Chris Bosh       2547     24          Chris Bosh    1.858730
137          John Wall     202322    338           John Wall    5.769088
94         Roy Hibbert     201579    134         Roy Hibbert   -2.399553
239     Gordon Hayward     202330    399      Gordon Hayward    3.804865
36        James Harden     201935    107        James Harden   12.197839
184        Eric Gordon     201569      1         Eric Gordon    0.301299
77    Chandler Parsons     202718    473    Chandler Parsons    2.804578
```

2 Summary

Overall - our best random forest models seem to outperform our best gradient boosting models based on MSE for both rookies and non rookies.

2.1 Final Model Selection:

- **Rookies** - Random Forest Regression with optimized hyperparameters without team rating as a covariate
- **Veterans** - Random Forest Regression with optimized hyperparameters without team rating as a covariate

3 Now actually calculate priors and store them

[287]:
```
# read in contract data for 2015/16 season which will be used as the new data␣
↪in our model to get priors

newdata_vets = pd.read_csv("../data/Contract+team2015_NonRookie.csv")
```

```
newdata_rookies = pd.read_csv("../data/Contract+team2015_Rookie.csv")

newdata_vets.drop(newdata_vets.columns[0], axis = 1, inplace = True)
newdata_rookies.drop(newdata_rookies.columns[0], axis = 1, inplace = True)
```

[288]:
```
x_final_rookies = np.array(newdata_rookies['mu']).reshape(-1, 1)
x_final_vets = np.array(newdata_vets['mu']).reshape(-1, 1)
```

[289]:
```
# train rookie model and veteran model on all of our main data

rf_rookie2 = RandomForestRegressor(max_depth = 2, n_estimators = 200).
 ↪fit(x_main_rookies, y_main_rookies)

rf_vet2 = RandomForestRegressor(max_depth = 2, n_estimators = 50).
 ↪fit(x_main_vets, y_main_vets)

# NOTE - keep the MSE's from validation set and this will be used as our␣
 ↪standard error in the priors
mse_vets = mse_rf_vet2
mse_rookies = mse_rf_rookie2

priors_rookies_means = rf_rookie2.predict(x_final_rookies)
priors_vets_means = rf_vet2.predict(x_final_vets)

sigma_rookies = np.sqrt(mse_rookies)
sigma_vets = np.sqrt(mse_vets)

newdata_vets['finalpriors'] = priors_vets_means
newdata_rookies['finalpriors'] = priors_rookies_means

newdata_vets['finalse'] = sigma_vets
newdata_rookies['finalse'] = sigma_rookies
```

```
/Users/reedpeterson/opt/anaconda3/lib/python3.7/site-
packages/ipykernel_launcher.py:5: DataConversionWarning: A column-vector y was
passed when a 1d array was expected. Please change the shape of y to
(n_samples,), for example using ravel().
  """
```

[290]:
```
# Now add player id and index columns by merging with the player index map for␣
 ↪2015/16

player_index_map_2015 = pd.read_csv("../data/player_index_map_2015-16.csv")
player_index_map_2015.drop(player_index_map_2015.columns[0], axis = 1, inplace␣
 ↪= True)

player_index_map_2015.head()
```

```
[290]:    player_id  index      player_name
       0     201952      0      Jeff Teague
       1     203471      1  Dennis Schroder
       2     203488      2      Mike Muscala
       3     203145      3    Kent Bazemore
       4     203503      4       Tony Snell
```

```
[291]: newdata_vets = newdata_vets.merge(player_index_map_2015, how = "inner", left_on␣
        ↪= "name", right_on = "player_name")
       newdata_rookies = newdata_rookies.merge(player_index_map_2015, how = "inner",␣
        ↪left_on = "name", right_on = "player_name")

       newdata_vets
```

```
[291]:        rating                   Team        Type        mu  sd  \
       0     6.239155  Golden State Warriors  Non-rookie  0.833333   5
       1     6.239155  Golden State Warriors  Non-rookie  4.000000   5
       2     6.239155  Golden State Warriors  Non-rookie  3.790262   5
       3     6.239155  Golden State Warriors  Non-rookie  4.766667   5
       4     6.239155  Golden State Warriors  Non-rookie  3.903485   5
       ..         …                    …          …       …  ..
       245 -13.598845        New York Knicks  Non-rookie  4.333333   5
       246 -13.598845        New York Knicks  Non-rookie  0.933333   5
       247 -13.598845        New York Knicks  Non-rookie  0.550000   5
       248 -13.598845        New York Knicks  Non-rookie  0.452049   5
       249 -13.598845        New York Knicks  Non-rookie  1.633333   5

                         name  finalpriors   finalse  player_id  index  \
       0     Leandro Barbosa    -0.782111  3.688381       2571     12
       1        Andrew Bogut     1.213735  3.688381     101106    365
       2       Stephen Curry     1.178937  3.688381     201939    405
       3      Draymond Green     2.835642  3.688381     203110      9
       4      Andre Iguodala     1.178937  3.688381       2738    339
       ..                 …            …         …        …      …
       245       Robin Lopez     1.386358  3.688381     201577    127
       246    Kevin Seraphin     0.134907  3.688381     202338    128
       247     Lance Thomas    -0.863502  3.688381     202498     77
       248     Sasha Vujacic    -0.863502  3.688381       2756     75
       249  Derrick Williams     0.452774  3.688381     202682    199

                  player_name
       0     Leandro Barbosa
       1        Andrew Bogut
       2       Stephen Curry
       3      Draymond Green
       4      Andre Iguodala
       ..                 …
```

```
245        Robin Lopez
246     Kevin Seraphin
247       Lance Thomas
248      Sasha Vujacic
249   Derrick Williams

[250 rows x 11 columns]
```

[296]:
```
newdata_vets.to_csv("../data/final_priors_vets_2015_16.csv")
newdata_rookies.to_csv("../data/final_priors_rookies_2015_16.csv")
```

4 Notes -

To replicate this process for another year (2016/17 for example) using the final models selected here, we would do the following: * First two code cells are the same as in this file, just switch the years of the data that we read in. * Fit the two random forest models (rookies and vets) on the small train data for that year, then get mse on the validation data for that year and save this as it will be used as the prior standard error. * Then just use the 6 code cells above this one and make sure to put the correct year. That's all.

bayesian_reg_2015_16

May 17, 2021

1 Bayesian Regression Model 2015/16 using priors from optimized random forest model

[1]:
```
import pymc3 as pm
import pandas as pd
import numpy as np
import arviz as az

data = pd.read_csv("../data/shifts_data_final_2015_16.csv")
data.drop(data.columns[0], axis = 1, inplace = True)
data.head()
```

[1]:
```
   point_diff_per_100 home_team away_team    0    1    2    3    4    5    6  \
0          -26.939655     Hawks   Pistons  1.0  0.0  0.0  1.0  0.0  0.0  0.0
1          -32.349896     Hawks   Pistons  1.0  0.0  0.0  0.0  0.0  0.0  0.0
2            0.000000     Hawks   Pistons  1.0  0.0  0.0  0.0  0.0  0.0  0.0
3            8.373526     Hawks   Pistons  0.0  1.0  0.0  0.0  0.0  0.0  0.0
4          104.166667     Hawks   Pistons  0.0  1.0  0.0  0.0  0.0  0.0  0.0

   …  466  467  468  469  470  471  472  473  474  475
0  …  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
1  …  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
2  …  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
3  …  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
4  …  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0

[5 rows x 479 columns]
```

[2]:
```
priors_df_vets = pd.read_csv("../data/final_priors_vets_2015_16.csv")
priors_df_vets.drop(priors_df_vets.columns[0], axis = 1, inplace = True)
# need to rename the index column to idx
priors_df_vets.columns = ['rating', 'team', 'type', 'mu' ,'sd', 'name',␣
 ↪'finalpriors', 'finalse', 'player_id', 'idx', 'player_name']

priors_df_rookies = pd.read_csv("../data/final_priors_rookies_2015_16.csv")
priors_df_rookies.drop(priors_df_rookies.columns[0], axis = 1, inplace = True)
# need to rename the index column to idx
```

```
priors_df_rookies.columns = ['rating', 'team', 'type', 'mu' ,'sd', 'name',␣
 ↪'finalpriors', 'finalse', 'player_id', 'idx', 'player_name']

priors_df_vets.sort_values(by = ['idx'], inplace = True)
priors_df_rookies.sort_values(by = ['idx'], inplace = True)
```

[10]:
```
priors_df_vets.loc[priors_df_vets['idx'] == 405]
```

[10]:
```
     rating                   team        type        mu  sd           name  \
2  6.239155  Golden State Warriors  Non-rookie  3.790262   5  Stephen Curry

   finalpriors   finalse  player_id  idx    player_name
2     1.178937  3.688381     201939  405  Stephen Curry
```

[3]:
```
prior_means = np.zeros(476)
prior_sigmas = np.full(476, 4)

for i in range(len(prior_means)):
    if i in np.array(priors_df_vets['idx']):
        prior_means[i] = priors_df_vets.loc[priors_df_vets['idx'] ==␣
 ↪i]['finalpriors'].iloc[0]
        prior_sigmas[i] = priors_df_vets.loc[priors_df_vets['idx'] ==␣
 ↪i]['finalse'].iloc[0]
    elif i in np.array(priors_df_rookies['idx']):
        prior_means[i] = priors_df_rookies.loc[priors_df_rookies['idx'] ==␣
 ↪i]['finalpriors'].iloc[0]
        prior_sigmas[i] = priors_df_rookies.loc[priors_df_rookies['idx'] ==␣
 ↪i]['finalse'].iloc[0]
```

[4]:
```
home_teams = data['home_team']
away_teams = data['away_team']
# now drop these columns from the main training dataframe
data.drop(['home_team', 'away_team'], axis = 1, inplace = True)
data.head()
```

[4]:
```
   point_diff_per_100    0    1    2    3    4    5    6    7    8  …  466  \
0          -26.939655  1.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  …  0.0
1          -32.349896  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  …  0.0
2            0.000000  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  …  0.0
3            8.373526  0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  …  0.0
4          104.166667  0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  …  0.0

   467  468  469  470  471  472  473  474  475
0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
1  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
2  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
```

```
3  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
4  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0

[5 rows x 477 columns]
```

[5]:
```python
# need to rename columns now since numbers confuse pymc3
new_cols = []
for i in range(np.shape(data)[1]):
    if i == 0:
        new_cols.append("point_diff")
    else:
        new_cols.append("p" + str(i-1))

# x_df = data.iloc[:20000,]
x_df = data
x_df.columns = new_cols
x_df
```

[5]:
```
       point_diff   p0   p1   p2   p3   p4   p5   p6   p7   p8  …  p466  \
0      -26.939655  1.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  …   0.0
1      -32.349896  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  …   0.0
2        0.000000  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  …   0.0
3        8.373526  0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  …   0.0
4      104.166667  0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  …   0.0
…             …  …  …  …  …  …  …  …  …  …  …  …  …
33884    0.000000  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  …   0.0
33885   -8.768238  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  …   0.0
33886    0.000000  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  …   0.0
33887   72.337963  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  …   0.0
33888  236.742424  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  …   0.0

       p467  p468  p469  p470  p471  p472  p473  p474  p475
0       0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0
1       0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0
2       0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0
3       0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0
4       0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0
…     …    …    …    …    …    …    …    …    …
33884   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0
33885   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0
33886   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0
33887   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0
33888   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0

[33889 rows x 477 columns]
```

[6]:
```
x = np.array(x_df.iloc[:,1:])
y = np.array(x_df.iloc[:,0])

x_shape = np.shape(x)[1]

with pm.Model() as model:
    # priors
    sigma = pm.HalfCauchy("sigma", beta=10) # arbitrarily defined
    intercept = pm.Normal("Intercept", 0, sigma=20) # arbitrarily defined
    x_prior_means = prior_means # defined above
    x_prior_sigmas = prior_sigmas # defined above
#     x_prior_means = np.zeros(x_shape) # just testing with mean zero to␣
 ↪compare to ridge
    x_coeff = pm.Normal("x", mu = x_prior_means, sigma=x_prior_sigmas, shape =␣
 ↪x_shape) # original method - no list comprehension

    likelihood = pm.Normal("y", mu=intercept + x_coeff.dot(x.T), sigma=sigma,␣
 ↪observed=y) # original method - no list comprehension

    trace = pm.sample(1000, tune = 1000, cores = 1)
```

```
/Users/reedpeterson/opt/anaconda3/lib/python3.7/site-
packages/pymc3/sampling.py:468: FutureWarning: In an upcoming release, pm.sample
will return an `arviz.InferenceData` object instead of a `MultiTrace` by
default. You can pass return_inferencedata=True or return_inferencedata=False to
be safe and silence this warning.
  FutureWarning,
Auto-assigning NUTS sampler…
Initializing NUTS using jitter+adapt_diag…
Sequential sampling (2 chains in 1 job)
NUTS: [x, Intercept, sigma]
```

```
<IPython.core.display.HTML object>
```

```
/Users/reedpeterson/opt/anaconda3/lib/python3.7/site-packages/pymc3/math.py:246:
RuntimeWarning: divide by zero encountered in log1p
  return np.where(x < 0.6931471805599453, np.log(-np.expm1(-x)),
np.log1p(-np.exp(-x)))
```

```
<IPython.core.display.HTML object>
```

```
Sampling 2 chains for 1_000 tune and 1_000 draw iterations (2_000 + 2_000 draws
total) took 1641 seconds.
```

1.1 Save the trace:

[33]:
```
with model:
    path = pm.save_trace(trace, directory = "trace_2015_16")
```

[7]:
```
with model:
    results_df = az.summary(trace)
```

[7]:
```
player_index_map_2015 = pd.read_csv("../data/player_index_map_2015-16.csv")
player_index_map_2015.head()
```

[7]:
```
   Unnamed: 0  player_id  index      player_name
0           0     201952      0      Jeff Teague
1           1     203471      1  Dennis Schroder
2           2     203488      2     Mike Muscala
3           3     203145      3    Kent Bazemore
4           4     203503      4       Tony Snell
```

[8]:
```
# player_index_map_2015.loc[player_index_map_2015['index'] == 163]
player_index_map_2015.loc[player_index_map_2015['player_name'] == "Stephen␣
 ↪Curry"]
```

[8]:
```
     Unnamed: 0  player_id  index    player_name
405         405     201939    405  Stephen Curry
```

[9]:
```
print((results_df.loc[results_df['mean'] > 4]).sort_values(by=['mean']))
```

```
         mean     sd  hdi_3%  hdi_97%  mcse_mean  mcse_sd  ess_bulk  \
x[459]  4.032  2.114   0.198    8.061      0.032    0.026    4279.0
x[200]  4.053  2.348  -0.224    8.482      0.039    0.033    3540.0
x[32]   4.271  2.326  -0.041    8.618      0.033    0.031    5132.0
x[42]   4.316  2.268  -0.178    8.340      0.034    0.032    4427.0
x[183]  4.318  2.159   0.178    8.259      0.033    0.027    4200.0
x[405]  4.325  2.317  -0.121    8.792      0.034    0.030    4593.0
x[114]  4.430  2.629  -0.869    8.828      0.038    0.035    4715.0
x[256]  4.487  2.134   0.404    8.285      0.028    0.023    5986.0
x[439]  4.565  2.240   0.283    8.797      0.034    0.029    4253.0
x[201]  4.607  2.213   0.653    8.779      0.031    0.030    4954.0
x[413]  4.616  2.212   0.378    8.599      0.030    0.026    5356.0
x[427]  4.619  2.158   0.657    8.759      0.029    0.026    5501.0
x[48]   4.631  2.208   0.546    8.845      0.032    0.033    4845.0
x[111]  4.672  2.254   0.564    8.962      0.034    0.030    4306.0
x[329]  4.814  2.182   0.912    8.965      0.033    0.028    4382.0
x[304]  4.945  2.093   1.141    9.010      0.033    0.025    4009.0
x[78]   5.265  2.176   1.015    9.130      0.032    0.025    4761.0
x[23]   5.396  2.161   1.182    9.389      0.032    0.026    4666.0
x[303]  5.552  2.341   1.249   10.052      0.038    0.031    3826.0
```

```
x[27]     5.761  2.264   1.582   10.001       0.037    0.027    3859.0
x[138]    5.833  2.134   1.467    9.540       0.035    0.026    3655.0
x[38]     5.854  2.135   2.018   10.046       0.032    0.025    4492.0
x[163]    5.962  2.089   1.903    9.611       0.028    0.022    5285.0
x[455]    6.164  2.326   1.871   10.472       0.033    0.027    5003.0
x[9]      6.289  2.361   1.441   10.364       0.036    0.029    4391.0
x[35]     6.366  2.298   1.864   10.362       0.033    0.026    4873.0
x[82]     7.654  2.200   3.732   11.809       0.033    0.026    4341.0
x[93]     8.216  2.244   3.923   12.360       0.037    0.028    3641.0
sigma    81.049  0.308  80.392   81.589       0.004    0.003    4933.0

        ess_tail  r_hat
x[459]    1696.0    1.0
x[200]    1460.0    1.0
x[32]     1224.0    1.0
x[42]     1296.0    1.0
x[183]    1670.0    1.0
x[405]    1597.0    1.0
x[114]    1727.0    1.0
x[256]    1538.0    1.0
x[439]    1152.0    1.0
x[201]    1286.0    1.0
x[413]    1713.0    1.0
x[427]    1554.0    1.0
x[48]     1064.0    1.0
x[111]    1558.0    1.0
x[329]    1561.0    1.0
x[304]    1363.0    1.0
x[78]     1640.0    1.0
x[23]     1333.0    1.0
x[303]    1484.0    1.0
x[27]     1122.0    1.0
x[138]    1423.0    1.0
x[38]     1407.0    1.0
x[163]    1595.0    1.0
x[455]    1544.0    1.0
x[9]      1215.0    1.0
x[35]     1585.0    1.0
x[82]     1404.0    1.0
x[93]     1409.0    1.0
sigma     1195.0    1.0
```

[34]:

```python
with model:
    tmp = trace.get_values("x")

# np.shape(tmp)
# np.mean(tmp, axis = 0)
```

```
tmp_df = pd.DataFrame(tmp)
tmp_df.to_csv(r'../data/bayesian_posterior_samples_2015_16.csv')
```

[35]:
```
results_df.to_csv(r'../data/bayesian_results_df_2015_16.csv')
```