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. ~~In this paper~~ We work with data from the 2018-19 NBA season, though our methods should be able to be easily applied to any NBA season. ~~As mentioned earlier~~ We use Bayesian regression to model the Plus-Minus statistic for players, and we use a nested ridge regression to derive logical prior distributions for each player based on their team and contract value. [Tentative] The model we arrive at corrects for both teammate performance and overall team performance; we believe the model offers improvement on conventional methods for individual player performance. The model does, however, have some outliers where some role players are overvalued.

ok good, thanks

Introduction

Our client, Kostas Pelechrinis, is a Professor at the University of Pittsburgh who is interested in obtaining an improved estimate of the plus-minus statistic for NBA players. 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 is a very natural metric to use when measuring players' contribution to their teams simply by definition - plus-minus measures the net point differential for player A's team while player A 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.

Please add a basic citation for +/- .

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 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

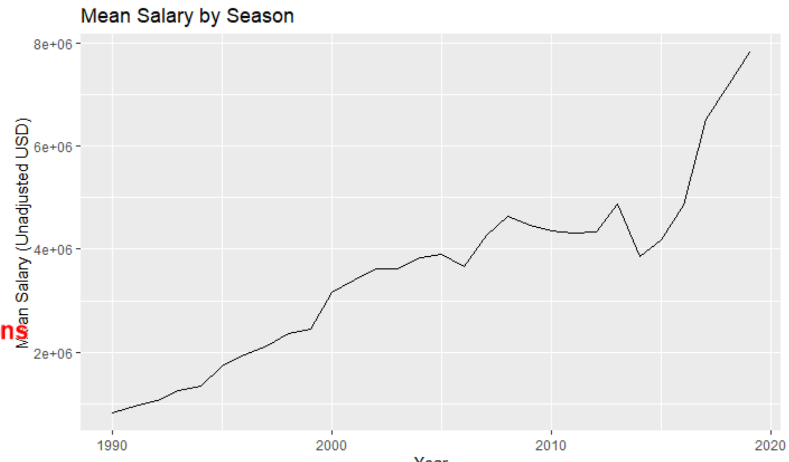
Contract Data

The first dataset contains information about player contracts. This data was obtained from two different sources: web-scraping data found on spotrak.com (2017-2019 seasons) and Kaggle (1990 - 2017 seasons). These two data sets were joined on player name, 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. The data relevant to this draft are from the 2017-2019 seasons.

and so mostly from spotrak?



would be more informative to adjust for inflation especially for a multiple-seasons analysis

Figure 1: blah blah blah (source: kaggle.com)

As seen in Figure ~~#~~1, the average salary from 1990 to 2019 has been increasing. This isn't a huge cause for concern as discussed later in the Methods section since our priors or strictly based on only the previous season and we are assuming we are blind to the current seasons data. If we used multiple seasons in our priors, we would need to adjust for this positive trend between time and player salary.

Games Data

move web address to list of references and put citation here (remember ASA style in both places)

Our NBA games data comes from 538 (https://github.com/fivethirtyeight/data/tree/master/nba-forecasts). This data is actually used by 538 to create elo rankings for each team updated after each game. The data goes back to 1946 and includes variables such as each team's pre and post elo scores as well as the final game scores and who had home court advantage. We will use this data to create a team rating system that will be used to create priors for our bayesian regression.

define this

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 1 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 1

Delete this line

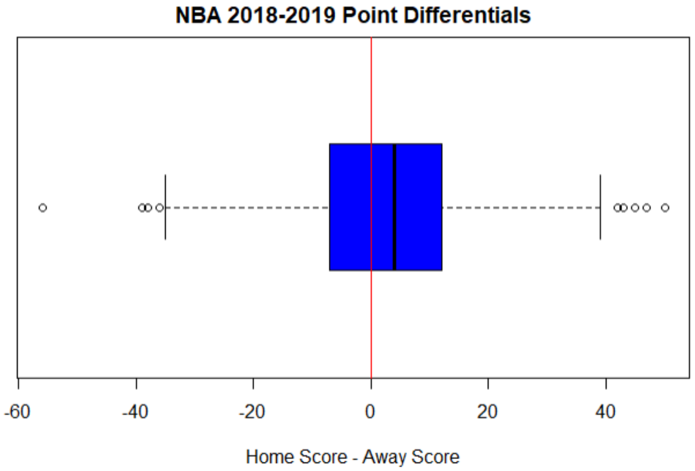


Figure X: blah blah

Figure number and caption below figure

Shifts Data

The last main dataset that we used was derived from NBA play-by-play data from eighthirtyfour (add citation here). The play-by-play dataset includes a number of different variables with only a few of interest to us: good, thanks.

- 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 ~~following~~ formula (include citation here, nbastuffer.com): (delete colon)

good, move web addr to list of refs and put citation here (both in ASA style)

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

lower case

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 (roughly 530 columns) 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 1s, while the five away players in a shift are denoted with -1s.

This dataset is used to train our Bayesian regression model, where the set of columns corresponding to the players forms our design matrix ***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

?

This section will include linear regression for team ratings as well as the ridge regression stuff for priors. When creating our ultimate priors to be used in bayesian regression, we split the data into rookies and veterans. ~~Naturally~~, T this means that linear and ridge regression were performed separately on the two classes as rookie contracts do not compare well to non-rookie contracts.

Linear Regression

Our team utilized linear regression in two ways:

- To derive team ratings for each season
- To derive standard errors for each player in our nested ridge regression model.

(described below in the subsection on ridge regression)

Use in Deriving Team Ratings

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.

Use in Deriving Standard Errors for all Players

Linear regression is also used to derive a standard error for the final prior mean for each player. The idea is that we want to have a unique prior mean and standard error for each player. A nested ridge regression gives us the prior mean, but since glmnet and regularized regression techniques don't provide good estimates of standard error, we used linear regression. This linear regression follows the same formula as the nested ridge regression, just without any penalization. We regress player coefficients generated from an initial ridge regression using a per 100 possession point differential variable on the contract priors and team rating priors for each player. This gives us an approximation for the standard error for each player.

N.b. for future work: Ridge regression is Bayesian regression with priors on the coefficients centered at zero, and the prior variance is the ridge regression "tuning parameter". Since you know how to fit a Bayesian regression, you could get SE's by fitting this Bayesian model directly.

Ridge Regression

Ridge regression was used in the following ways:

- Used to predict coefficients for each player using a per 100 possession point differential variable.
- These coefficients were then used as the dependent variable in another ridge regression that tries to predict this coefficient from team rating and contract prior.

Ridge regression is a method that we used heavily in the process of deriving meaningful prior distributions for NBA players. Specifically, we used a nested regression framework to compute logical priors for players in the following manner:

- Run a ridge regression with point differential per 100 possessions as the dependent variable, and our sparse matrix X as 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 representing their plus-minus estimate for a given season with no prior information.
- The coefficients from this ridge regression are then used as the *dependent variable* in another regression model where the covariates are contract value and team ratings. This will give us coefficient estimates for contract value and team ratings.
- To obtain a final prior mean for each player, we use this regression model and insert the player's contract value and his team's rating as the new values of the covariates.

this would be a pretty standard multilevel model (as in 36-617) with the first bullet here being level 1 and the second being level 2. Too bad you didn't try that!

(you could even fit it using the Bayesian methods you have learned...)

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 (citation here), we seek a model of the ~~following~~ form: (delete colon)

yup, thanks.

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

, (comma)

(lower case)

Where y is a vector containing the point differential in each shift, β_0 is a constant representing home-court advantage, (space) $X$is our sparse design matrix described above in the Data section, and (space) $\beta$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.

As a sanity check, it would be good to make

(a) a scatter plot of naive +/- vs the posterior means of your adjusted +/-

(b) parallel boxplots of the posterior distribution of the difference between your adjusted +/- and naive +/-, for each player

We implement this model in Python using the pymc3 package (include citation here for pycm3). Pymc3 provides a convenient framework for specifying prior distributions, allowing us to input our priors derived from the ridge regression 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 $\beta_0$and a distribution for the error term ϵ.

yup

Visualizing Results

The last goal of the project given to us by the client is to create an interactive visualization of the results. This application is now in its developmental phase, and will be done with R Shiny. The current prototype allows the user to select any number of players from a drop-down list from the 2018-2019 season. The application then displays our estimates of each selected player's plus-minus distribution using ggplot.

great! You guys have made good progress, and I look forward to seeing the shiny app!

Results

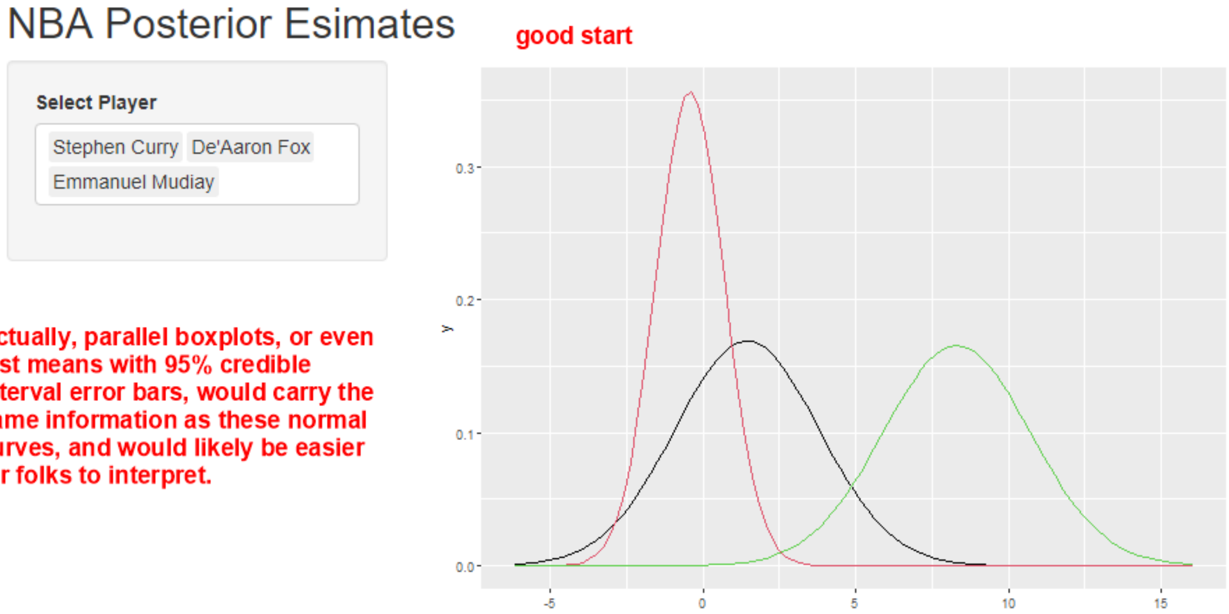
Plus-Minus Posterior Distributions with Bayesian Regression

So far we have been able to build a reasonable Bayesian regression model to estimate distributions of players' plus-minus statistics. 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.

[We are still finalizing some of our results/re-running our models one last time with some final edits, so more will be added here in the future]

ok

Visualization Prototype:



NBA Posterior Esimates

Select Player

Stephen Curry De'Aaron Fox Emmanuel Mudiay

good start

Actually, parallel boxplots, or even just means with 95% credible interval error bars, would carry the same information as these normal curves, and would likely be easier for folks to interpret.

The figure above shows an example output of when the user selects 3 players. Here, De'Aaron Fox is the black distribution, Emmanuel Mudiay is the red, and Stephen Curry is the green (currently there is no legend, but we will resolve this before the final). This is reasonable, as Curry is one of the best players in the NBA, whereas Fox is an up-and-coming star, and Mudiay is a bench player.

Discussion [Tentative]

ok

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 first component of our prior, the contract value, allowed us to adjust players based on how a team views the player's value. By doing this, we were able to mostly adjust for teammate's impact on a player's plus-minus. 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 more muted posterior distribution than when compared to the box plus-minus. Another factor we adjusted for was team rating, as superstar players on bad teams suffer from conventional methods. One such example is Bradley Beal on the Washington Wizards - a terrible team during this season, who has a box score plus-minus consistent with a decent starter, but has the 13th highest mean by our metric - a position more consistent with his star/superstar status.

showing graphs illustrating both points here would be really neat

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. There were some outliers, such as Brook Lopez and Donte Divencenzo, two important role players on the Milwaukee Bucks, were ranked very highly; in contrast, Giannis Antetekounpo, their

comparing your +/- with naive +/- and being able to explain both cases in which the two are similar as well as cases in which the two are different, would go some distance toward validating your methodology.

MVP teammate, was only the 8th best rating. We believe that this shows that the model is not perfect when correcting for teammates, but the overall trend is that role players that do not contribute much when playing with superstar teammates are no longer skewed upwards.

While a visualization can be informative, our group would also like to produce a probabilistic comparison between two players (i.e. on any given day, Stephen Curry has a 90% chance of being better than De'aaron Fox). In order to compare player distributions, we plan to utilize a Z-test (to be discussed with Prof. MacDonald). Future functionality could be to incorporate more seasons.

If you are doing MCMC here, you can directly sample from the posterior distribution of the difference in +/- of the two players and estimate the prob that the difference exceeds zero.

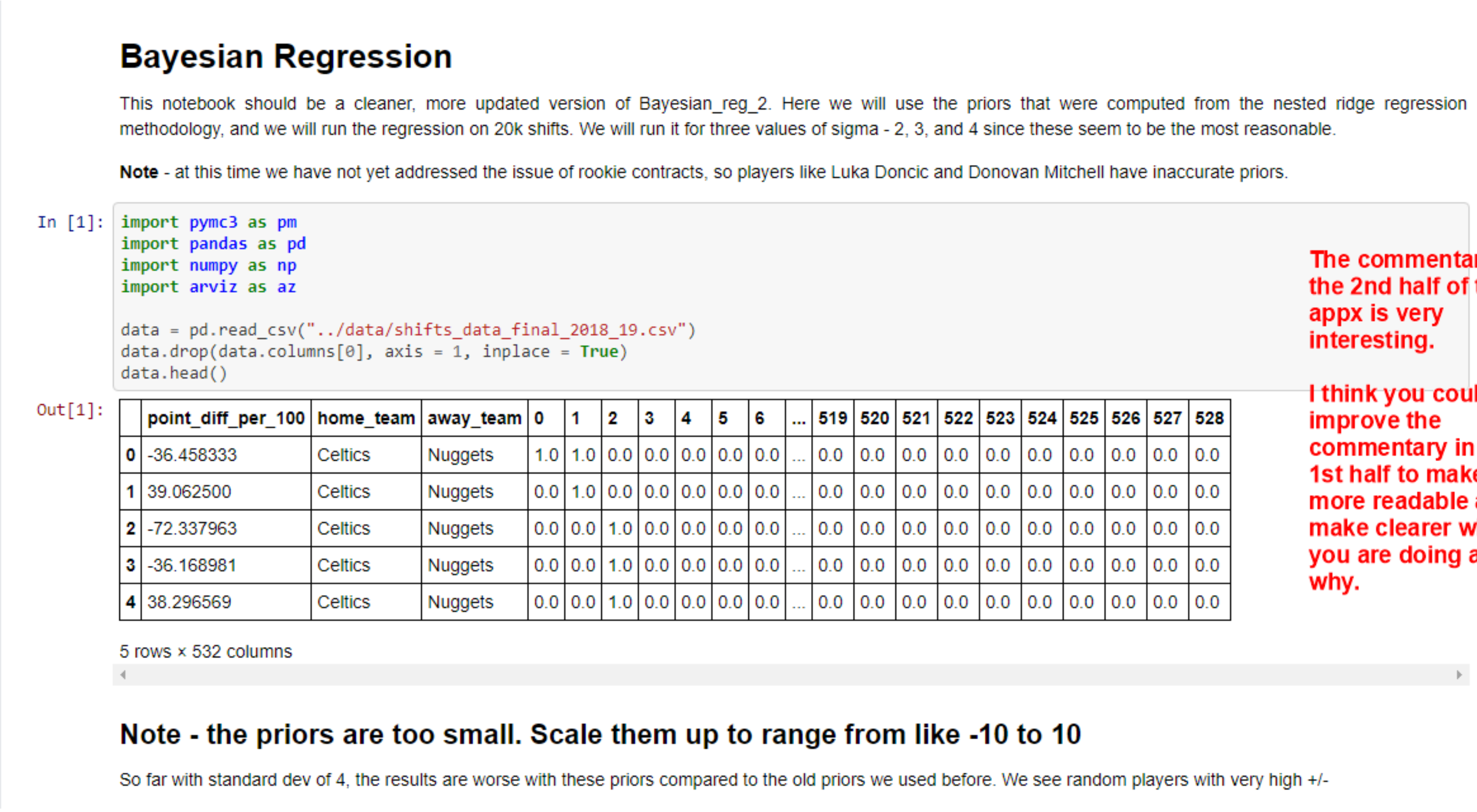
References

- Deshpande, S. & Jensen, S. (2016), *Estimating an NBA player's impact on his team's chances of winning*. Journal of Quantitative Analytics Sports.
- 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.
- https://eightthirtyfour.com/data
- https://www.nbastuffer.com/analytics101/possession/
- Spotrak.com
- https://www.kaggle.com/whitefero/nba-player-salary-19902017

good start on references. be sure to put them in ASA form.

TECHNICAL APPENDIX (INCOMPLETE) Still need to cite in main document

good, thanks for remembering



Bayesian Regression

This notebook should be a cleaner, more updated version of Bayesian_reg_2. Here we will use the priors that were computed from the nested ridge regression methodology, and we will run the regression on 20k shifts. We will run it for three values of sigma - 2, 3, and 4 since these seem to be the most reasonable.

Note - at this time we have not yet addressed the issue of rookie contracts, so players like Luka Doncic and Donovan Mitchell have inaccurate priors.

```
In [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_2018_19.csv")
        data.drop(data.columns[0], axis = 1, inplace = True)
        data.head()
```

Out[1]:

	point_diff_per_100	home_team	away_team	0	1	2	3	4	5	6	...	519	520	521	522	523	524	525	526	527	528
0	-36.458333	Celtics	Nuggets	1.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	39.062500	Celtics	Nuggets	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	-72.337963	Celtics	Nuggets	0.0	0.0	1.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	-36.168981	Celtics	Nuggets	0.0	0.0	1.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	38.296569	Celtics	Nuggets	0.0	0.0	1.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

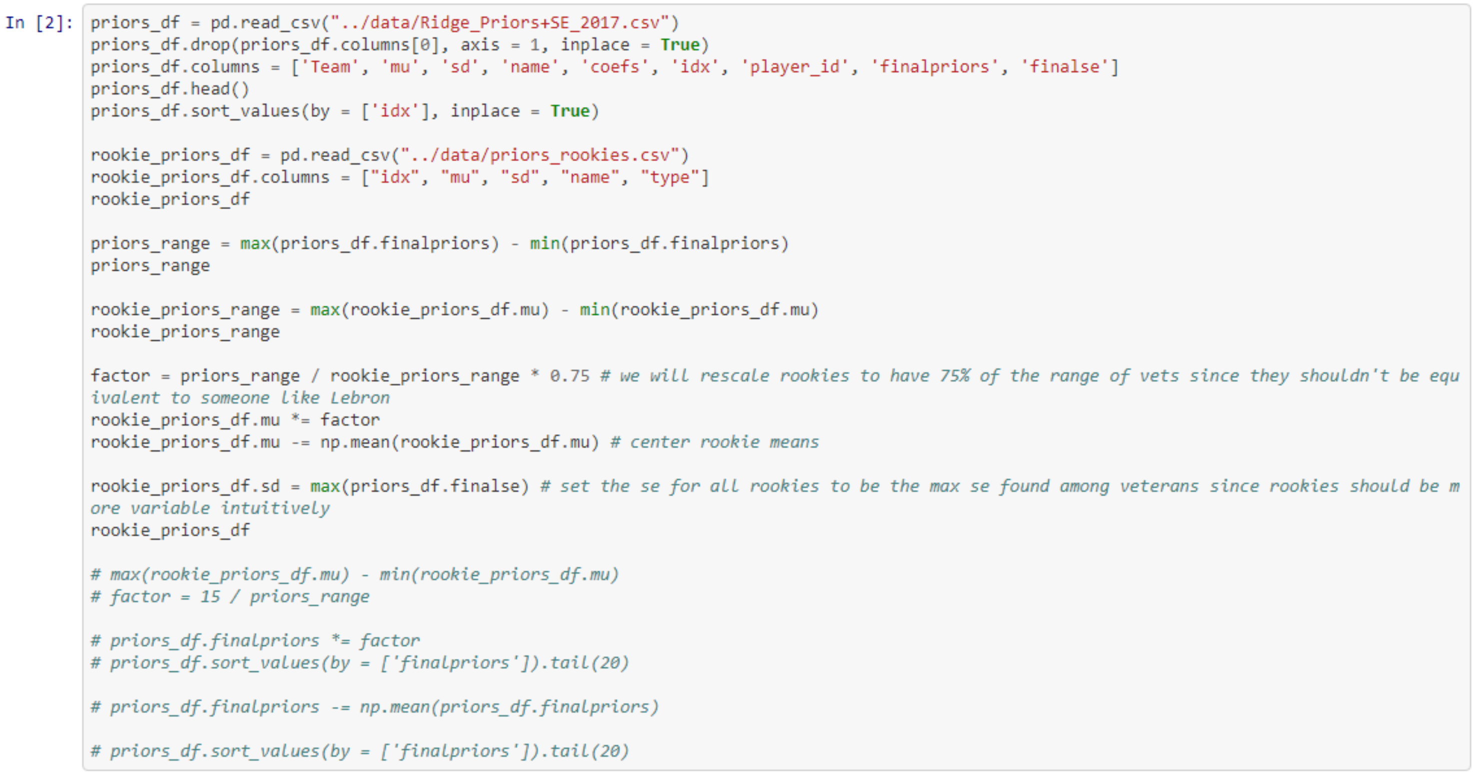
5 rows × 532 columns

The commentary in the 2nd half of the appx is very interesting.

I think you could improve the commentary in the 1st half to make it more readable and make clearer what you are doing and why.

Note - the priors are too small. Scale them up to range from like -10 to 10

So far with standard dev of 4, the results are worse with these priors compared to the old priors we used before. We see random players with very high +/-



```
In [2]: priors_df = pd.read_csv("../data/Ridge_Priors+SE_2017.csv")
        priors_df.drop(priors_df.columns[0], axis = 1, inplace = True)
        priors_df.columns = ['Team', 'mu', 'sd', 'name', 'coefs', 'idx', 'player_id', 'finalpriors', 'finalse']
        priors_df.head()
        priors_df.sort_values(by = ['idx'], inplace = True)

        rookie_priors_df = pd.read_csv("../data/priors_rookies.csv")
        rookie_priors_df.columns = ["idx", "mu", "sd", "name", "type"]
        rookie_priors_df

        priors_range = max(priors_df.finalpriors) - min(priors_df.finalpriors)
        priors_range

        rookie_priors_range = max(rookie_priors_df.mu) - min(rookie_priors_df.mu)
        rookie_priors_range

        factor = priors_range / rookie_priors_range * 0.75 # we will rescale rookies to have 75% of the range of vets since they shouldn't be equ
        ivalent to someone like Lebron
        rookie_priors_df.mu *= factor
        rookie_priors_df.mu -= np.mean(rookie_priors_df.mu) # center rookie means

        rookie_priors_df.sd = max(priors_df.finalse) # set the se for all rookies to be the max se found among veterans since rookies should be m
        ore variable intuitively
        rookie_priors_df

        # max(rookie_priors_df.mu) - min(rookie_priors_df.mu)
        # factor = 15 / priors_range

        # priors_df.finalpriors *= factor
        # priors_df.sort_values(by = ['finalpriors']).tail(20)

        # priors_df.finalpriors -= np.mean(priors_df.finalpriors)

        # priors_df.sort_values(by = ['finalpriors']).tail(20)
```

Out[2]:

	idx	mu	sd	name	type
0	1	2.249577	0.761081	Jayson Tatum	Rookie
1	2	1.237387	0.761081	Jaylen Brown	Rookie
2	3	-0.164070	0.761081	Terry Rozier	Rookie
3	14	-0.889821	0.761081	OG Anunoby	Rookie
4	16	-0.503590	0.761081	Delon Wright	Rookie
...	...	...	...	...	...
97	494	-0.510240	0.761081	Dario Saric	Rookie
98	504	-0.916798	0.761081	Aaron Holiday	Rookie
99	507	-0.622432	0.761081	Lonnie Walker IV	Rookie
100	514	1.625617	0.761081	Brandon Ingram	Rookie
101	515	-0.864193	0.761081	Chandler Hutchison	Rookie

102 rows × 5 columns

In [3]:

```python
# Now we need to go through priors_df and drop any rows that are for players with rookie contracts
drop_lst = []

for i in range(len(priors_df)):
    cur_idx = priors_df.iloc[i]['idx']
    if cur_idx in np.array(rookie_priors_df['idx']):
        drop_lst.append(i) # add this index to the list of indexes to be dropped

# priors_df.reset_index(drop = True, inplace = True)
priors_df.drop(drop_lst, inplace = True)

priors_df
```

Out[3]:

	Team	mu	sd	name	coefs	idx	player_id	finalpriors	finalse
70	Boston Celtics	1.666667	5	Marcus Morris	-4.003747	0	202694	-0.269767	0.259020
68	Boston Celtics	1.881800	5	Jayson Tatum	3.790331	1	1628369	-0.158154	0.253661
67	Boston Celtics	1.652160	5	Jaylen Brown	-8.338793	2	1627759	-0.277293	0.259416
73	Boston Celtics	0.662840	5	Terry Rozier	-2.256052	3	1626179	-0.790558	0.295269
308	Chicago Bulls	4.596167	5	Robin Lopez	3.294456	4	201577	1.249145	0.414105
...	...	...	...	...	...	...	...	...	...
179	Miami Heat	4.651333	5	James Johnson	-6.849357	509	201949	1.278399	0.251177
171	Charlotte Hornets	0.900000	5	Michael Carter-Williams	-4.064299	510	203487	-0.667793	0.239569
320	Phoenix Suns	1.050977	5	T.J. Warren	5.711320	513	203933	-0.590305	0.455433
214	Los Angeles Lakers	1.839800	5	Brandon Ingram	1.426551	514	1627742	-0.180359	0.214220
64	Boston Celtics	0.030953	5	Demetrius Jackson	-1.072410	516	1627743	-1.118385	0.325535

299 rows × 9 columns

In [4]:

```python
# Note - priors_df is missing any players with no contract data due to name inconsistencies. We will fill them in with zero means here
prior_means = np.zeros(529)

for i in range(len(prior_means)):
    if i in np.array(priors_df['idx']):
        prior_means[i] = priors_df.loc[priors_df['idx'] == i]['finalpriors'].iloc[0]
    elif i in np.array(rookie_priors_df['idx']):
        prior_means[i] = rookie_priors_df.loc[rookie_priors_df['idx'] == i]['mu'].iloc[0]
```

In [5]:

```
prior_sd = np.full(529, np.mean(priors_df.finalse)) # initialize all standard errors to the mean, then we'll edit below

for i in range(len(prior_sd)):
    if i in np.array(priors_df['idx']):
        prior_sd[i] = priors_df.loc[priors_df['idx'] == i]['finalse'].iloc[0]
    elif i in np.array(rookie_priors_df['idx']):
        prior_sd[i] = rookie_priors_df.loc[rookie_priors_df['idx'] == i]['sd'].iloc[0]



# prior_sd4 = np.full(529, 4)
# prior_sd3 = np.full(529, 3)
# prior_sd2 = np.full(529, 2)
```

In [6]:

```
# store home and away teams for potential use later when we incorporate team ratings
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()
```

Out[6]:

	point_diff_per_100	0	1	2	3	4	5	6	7	8	...	519	520	521	522	523	524	525	526	527	528
0	-36.458333	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	39.062500	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	-72.337963	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	-36.168981	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	38.296569	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

5 rows × 530 columns

In [7]:

```
# 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
```

Out[7]:

	point_diff	p0	p1	p2	p3	p4	p5	p6	p7	p8	...	p519	p520	p521	p522	p523	p524	p525	p526	p527	p528
0	-36.458333	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	39.062500	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	-72.337963	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	-36.168981	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	38.296569	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
33885	-52.083333	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	0.0	0.0	0.0	0.0	0.0
33886	38.868159	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	0.0	0.0	0.0	0.0	0.0
33887	60.562016	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	0.0	0.0	0.0
33888	-72.337963	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	0.0	0.0	0.0	0.0	0.0
33889	75.483092	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	0.0	0.0	0.0

33890 rows × 530 columns

Below we fit the Bayesian model. ONLY run this cell if we do not have a saved trace. If we have a saved trace, proceed further down and execute the cell to load a saved trace.

In [60]:

```
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_sd * 5 # 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 wil
l return an `arviz.InferenceData` object instead of a `MultiTrace` by default. You can pass return_inferencedata=True or return_inference
data=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]
```

```
100.00% [2000/2000 13:19<00:00 Sampling chain 0, 0 divergences]
```

```
/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)))
```

```
100.00% [2000/2000 14:20<00:00 Sampling chain 1, 0 divergences]
```

```
Sampling 2 chains for 1_000 tune and 1_000 draw iterations (2_000 + 2_000 draws total) took 1660 seconds.
```

Below we save the trace

In [9]:

```
with model:
    path = pm.save_trace(trace, directory = "main_model_trace")
```

In [61]:

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

Below we load the saved trace

In [27]:

```
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_sd # 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

    loaded_trace = pm.load_trace("main_model_trace")

    results_df = az.summary(loaded_trace)
```

In [10]:

```
# import the player map dictionary to go between index, player_id, and name
player_index_map = pd.read_csv("../data/player_index_map.csv")
```

In [77]:

```
player_index_map.loc[player_index_map.index == 212]
```

Out[77]:

	Unnamed: 0	player_id	index	player_name
212	212	203496.0	212	Robert Covington

In [62]:

```
print((results_df.loc[results_df['mean'] > 3]).sort_values(by=['mean']))
```

```
          mean     sd  hdi_3%  hdi_97%  mcse_mean  mcse_sd  ess_bulk  \
x[212]   3.022  1.591   0.076    6.025      0.028    0.023    3279.0   
x[412]   3.028  1.848  -0.513    6.528      0.033    0.029    3126.0   
x[358]   3.109  2.045  -0.505    7.196      0.028    0.029    5176.0   
x[42]    3.245  1.857  -0.341    6.561      0.031    0.024    3590.0   
x[219]   3.322  2.953  -2.205    8.675      0.053    0.056    3058.0   
x[28]    3.830  1.680   0.716    6.935      0.027    0.022    3812.0   
x[81]    3.894  1.631   0.951    7.021      0.026    0.020    4053.0   
x[335]   4.040  2.040   0.219    7.790      0.033    0.028    3833.0   
x[129]   4.336  1.784   0.734    7.451      0.033    0.026    2986.0   
x[30]    4.528  1.691   1.322    7.548      0.029    0.021    3451.0   
x[425]   4.528  1.557   1.743    7.586      0.024    0.019    4020.0   
x[112]   4.548  1.795   1.284    7.972      0.027    0.022    4299.0   
x[59]    4.602  1.727   1.332    7.820      0.029    0.022    3445.0   
x[95]    4.616  2.305   0.422    9.019      0.037    0.032    3856.0   
x[386]   5.074  1.798   1.779    8.412      0.026    0.021    4928.0   
x[225]   5.088  2.164   1.088    9.095      0.038    0.029    3140.0   
x[321]   5.124  2.048   1.567    8.970      0.033    0.028    3833.0   
x[462]   5.217  2.011   1.140    8.863      0.031    0.024    4315.0   
x[63]    6.528  1.810   3.181    9.926      0.028    0.021    4067.0   
x[207]   6.790  2.095   2.882   10.908      0.034    0.025    3949.0   
x[252]   7.275  2.157   3.329   11.379      0.037    0.029    3446.0   
x[317]   8.308  2.411   3.837   12.759      0.038    0.029    4022.0   
sigma   78.113  0.298  77.539   78.653      0.005    0.003    3808.0   

        ess_tail  r_hat  
x[212]    1264.0   1.01  
x[412]    1316.0   1.00  
x[358]    1556.0   1.00  
x[42]     1309.0   1.00  
x[219]    1244.0   1.00  
x[28]     1580.0   1.00  
x[81]     1580.0   1.00  
x[335]    1295.0   1.00  
x[129]    1279.0   1.00  
x[30]     1393.0   1.00  
x[425]    1232.0   1.00  
x[112]    1510.0   1.00  
x[59]     1492.0   1.00  
x[95]     1487.0   1.00  
x[386]    1287.0   1.00  
x[225]    1490.0   1.00  
x[321]    1411.0   1.00  
x[462]    1355.0   1.00  
x[63]     1608.0   1.00  
x[207]    1366.0   1.00  
x[252]    1474.0   1.00  
x[317]    1528.0   1.00  
sigma     1130.0   1.00  
```

Results from using the Ridge-derived priors for mean and sd for each player, running Bayesian Regression on full season

When we use nested ridge regression to derive prior means and standard deviations for each player, we see the following results for the Bayesian regression model:

- Steph Curry
- Paul Millsap
- LeBron James
- James Harden
- Gordon Hayward (this one is bizarre - Hayward was not good in 2018-19 or 2017-18 so his prior should not be high and his 2018/19 data should not boost him)
- Al Horford
- Damian Lillard
- Mike Conley
- DeMar DeRozan
- Carmelo Anthony (another bizarre one - Melo was not good)
- Markelle Fultz (another bad one - I think we're starting to see that priors are being weighed too heavily due to small standard errors)
- KD
- CP3
- Bradley Beal
- AD
- Andre Drummond
- Brook Lopez
- Giannis

Now - results after we scale up the standard deviations by a factor of 3

- Curry
- Millsap
- Harden
- Dame
- LeBron
- Hayward
- Al Horford
- Giannis
- Rudy Gobert
- Brook Lopez
- Bradley Beal
- Steven Adams
- Luka Doncic
- AD
- Andre Drummond
- PG
- Mike Conley
- CP3
- KD
- Danilo Galinari

When we scale up standard deviations by factor of 5, we see essentially the same results as above

The following cell shows how to get the distribution for a specific player

In this case we get the distribution for Steph Curry (index 317)

In [ ]:

```
import seaborn as sns

with model:
#     az.plot_trace(trace3)
#     print(np.shape(trace3['x']))
#     print(max(trace3['x'][:,317]))
    sns.distplot(trace['x'][:,317], hist = False)
```

In [ ]:

```
with model:
    az.plot_trace(trace)
```