Modeling basketball shot likelihoods with Bayesian Binomial-Logit models

Project work; CS-E5710: Bayesian Data Analysis

Rasmus Siljander, Juho Kuikka

6.12.2020

Contents

1. Introduction

Sports and predictions have always been intertwined - whether it is the casual sports fan proclaiming the results of the big game, the analyst wanting to highlight their expertise, or the bettor looking for the next payday, predicting sports outcomes have always been of key interest to many. In this regard, basketball is no different than any other sport. Games can come down to the last posession, where everything is decided by the outcome of one shot. As a bystander, it is a fascinating thought to be able to predict the outcome of that shot, and evaluate success likelihoods based on factors such as skill, location, and distance to basket.

This report aims to model the shot-making ability of professional basketball players. More specifically, we aspire to answer the following research question:

How does the location of the player relative to the basket affect the likelihood of a successful shot outcome?

This model evaluates three different techniques of the jump shot: 1) The traditional jump shot, where the player initiates the shooting motion from a near stand still 2) The fadeaway jump shot, where the player shoots the ball with simultaneous backwards motion 3) Pull-up jump shot, where the player is often moving with forward velocity while initiating the shot.

2. Data

The used data set (N = 129915) contained player shooting data from seasons between the years 2010-2018. The dataset contains three columns of interest: the shot type, distance from basket and the shot success flag (0 = miss; 1 = shot made). The number of recorded shots varied between shot types:

	Sample size (N)
Jump shot	50000
Fadeaway/Stepback	29915
Pullup	50000

The data was downloaded from the website nbasavant.com, which collected its data from the official NBA stats API as well as the ESPN shot tracker. We collected shooting data from many seasons to ensure that we have enough data replicates for all shot distances. The data was collected with certain constraints due to reasons we will discuss below.

In addition, a traditional train-test split was executed to allow for posterior predictive performance assessment for unseen observations. The dataset was formed by random sampling, and posterior predictive checking was performed by comparing models with the 20% (test) data.

Shot distance constraints

According to the NBA official rules website, the NBA three-point line is approximately 24 Feet from the basket. Since players want to maximize their shooting efficiency, shots will occur from as close to the basket as possible. Therefore, from a utility point of view, we expect that shot frequency will quickly dissipate once moving beyond the three-point line. As a result, since trying to model percentages beyond $\sim$ 26 feet offers little value, we restrict our data to shots under 30 feet.

Loading Packages and useful functions

Below are the functions and R libraries used in the project. The sourced functions were helper functions used to streamline the report such that the readability did not suffer by excess (and contextually redundant) code. The source code for the functions defined in 'project_functions' can be found in the Appendices.

```
library(rstan)
```

```
## Loading required package: StanHeaders

## Loading required package: ggplot2

## rstan (Version 2.21.2, GitRev: 2e1f913d3ca3)

## For execution on a local, multicore CPU with excess RAM we recommend calling
## options(mc.cores = parallel::detectCores()).
## To avoid recompilation of unchanged Stan programs, we recommend calling
## rstan_options(auto_write = TRUE)

## Do not specify '-march=native' in 'LOCAL_CPPFLAGS' or a Makevars file
```

```
library(ggplot2)
library(loo)
```

```
## This is loo version 2.3.1

## - Online documentation and vignettes at mc-stan.org/loo

## - As of v2.0.0 loo defaults to 1 core but we recommend using as many as possible. Use the 'cores' ar

## - Windows 10 users: loo may be very slow if 'mc.cores' is set in your .Rprofile file (see https://gi

##
## Attaching package: 'loo'

## The following object is masked from 'package:rstan':
##
##     loo
```

```
library(gridExtra)
library(bayesplot)
```

```
## This is bayesplot version 1.7.2

## - Online documentation and vignettes at mc-stan.org/bayesplot

## - bayesplot theme set to bayesplot::theme_default()

##    * Does _not_ affect other ggplot2 plots

##    * See ?bayesplot_theme_set for details on theme setting
```

```r
library(mcmcse)
```

```
## mcmcse: Monte Carlo Standard Errors for MCMC
## Version 1.4-1 created on 2020-01-29.
## copyright (c) 2012, James M. Flegal, University of California, Riverside
##                     John Hughes, University of Colorado, Denver
##                     Dootika Vats, University of Warwick
##                     Ning Dai, University of Minnesota
##  For citation information, type citation("mcmcse").
##  Type help("mcmcse-package") to get started.
```

```r
source('project_functions.R')
```

Here are peeks at the training and test data sets:

Loading data

```r
head(train$df_pooled)
```

```
##   distance throws successes
## 1       15   7875      3970
## 2       16   8845      4444
## 3       17   8970      4444
## 4       18   9032      4480
## 5       19   7907      3810
## 6       20   5814      2767
```

```r
head(test$df_pooled)
```

```
##   distance throws successes
## 1       15   2013      1011
## 2       16   2154      1076
## 3       17   2234      1124
## 4       18   2245      1107
## 5       19   1935       950
## 6       20   1482       678
```

3. Mathematical models:

Since the outcome of an individual throw T_i is Bernoulli distributed with a particular success likelihood p_i ($T_i \sim Bernoulli(p_i)$), with repeated number of trials, the variable accounting for made shots, S_i would be binomially distributed with suceess likelihood p_i:

$$S_i \sim Bin(n, p_i)$$

Our model assumes that a particular Bernoulli likelihood is logistically related player's distance to the basket x_b:

$$logit(p_i) = \alpha + \beta x_b$$

4. Stan Models

Pooled

Our first model is the pooled model, where we assume that all shot type samples are drawn from the same population. for the parameters α and β we have the same weakly informative prior $N(0, 10)$:

$$p_i \sim BinomialLogit(n, \alpha + \beta x_b)$$
$$\alpha \sim N(0, 10^2)$$
$$\beta \sim N(0, 10^2)$$

Hierarchical

The hierarchical model is a hybrid of the pooled and separate (where sample group is assumed to have been drawn from separate populations) model, allowing for the combination of the collective results of the pooled model with the individuality of the separate model. Furthermore, the hierarchical model reduces the parameter variance that is present in the separate model. In the hierarchical model, each shot type group follows the same (binomial logit) type of distribution, just like above, but they each have individual prior parameters α_j, β_j. These priors have a unique mean parameter μ_j, but share a common population standard deviation σ. All mean parameters μ_j are generated from a parent distribution with hyperparameters, in this case μ_0 and σ_0. The population standard deviation follows the SD parameter σ follows the same Gamma prior.

In this model, we use the familiar $N(0, 10)$ prior as a hyper parameter prior for μ_0:

$$p_{ij} \sim BinomialLogit(n, \alpha_j + \beta_j x_b)$$
$$\alpha_j \sim N(\mu_0, \sigma)$$
$$\beta_j \sim N(\mu_0, \sigma)$$
$$\mu_0 \sim N(0, 10)$$
$$\sigma \sim Gamma(1, 1)$$

5. Stan code

5.1 The pooled model

```
data {

  int<lower=0> N;              // Number of shot distances
  vector[N] distances;         // Vector of shot distances
  int throws[N];               // Throws per shot distance
  int successes[N];            // Number of successes per shot distance pair

  real mu0_alpha;              // Priors
  real sigma0_alpha;

  real mu0_beta;
  real sigma0_beta;

}

parameters {
  real alpha;                   // Model parameters
  real beta;
}

transformed parameters {
  vector[N] logit_p = alpha + beta * distances;
}

model {
  alpha ~ normal(mu0_alpha, sigma0_alpha);        // Weakly informative priors
  beta ~ normal(mu0_beta, sigma0_beta);

  successes ~ binomial_logit(throws, logit_p);
}


generated quantities {
  vector[N] log_lik;
  for (i in 1:N){
    log_lik[i] = binomial_logit_lpmf(successes[i]| throws[i], logit_p[i]);
  }
}
```

5.2 The hierarchical model

```
// Comparison of shot type groups with common variance

data {
  int<lower=0> N;                 // number of shot distances per group
  int<lower=0> J;                 // number of shot types

  vector[N] distances;            // Vector of shot distances
  int throws[N,J];                // Throws per shot distance and shot type
  int successes[N,J];             // Throws per shot distance and shot type

  real mu0_hyper;               // Priors
  real sigma0_hyper;

}

parameters {
  real mu0;                       // prior mean
  real<lower=0> sigma;            // prior std (constrained to be positive)
  vector[J] alpha;                // shot type alpha
  vector[J] beta;                 // shot type beta
}

model {
  mu0 ~ normal(mu0_hyper,sigma0_hyper);     // weakly informative prior
  sigma ~ gamma(1,1);                       // weakly informative prior

  for (i in 1:J) {
    alpha[i] ~ normal(mu0, sigma);  //parameters are modeled from group-specific distributions
    beta[i] ~ normal(mu0, sigma);
    successes[,i] ~ binomial_logit(throws[,i], alpha[i] + beta[i]*distances);
  }
}

generated quantities {
  vector[N*J] log_lik;
  for (j in 1:J) {
    for (i in 1:N) {
      log_lik[(j-1)*N + i] = binomial_logit_lpmf(successes[i,j]| throws[i,j],
      alpha[j] + beta[j]*distances[i]);
    }
  }
}
```

6. Execution of the Stan Code

The Stan model used default chain and iteration settings:

- 4 chains per model
- warm-up of 1000 iterations, 1000 iterations per chain

In addition to this, a seed of 1 was set to ensure that the results would be consistent with every test execution.

```
data_list_pooled = list(N = nrow(train$df_pooled)
                        , distances = train$df_pooled$distance
                        , throws = train$df_pooled$throws
                        , successes= train$df_pooled$successes
                        , mu0_alpha = 0
                        , sigma0_alpha = 10
                        , mu0_beta = 0
                        , sigma0_beta = 10)

data_list_hier = list(N = nrow(train$throw_data), J = 3
                      , distances = train$df_pooled$distance
                      , throws = train$throw_data
                      , successes= train$success_data
                      , mu0_hyper = 0
                      , sigma0_hyper = 10)
```

```
options(mc.cores = parallel::detectCores())
fit_pool = sampling(pool_model, data = data_list_pooled, seed = 1)
fit_hier = sampling(hier_model, data = data_list_hier, seed = 1)

params_pool = extract(fit_pool)
params_hier = extract(fit_hier)
```

7. Convergence diagnostics

7.1 Visual diagnostics

The first way of assessing convergence is a visual traceplot of the chains:

7.1.1 Pooled model

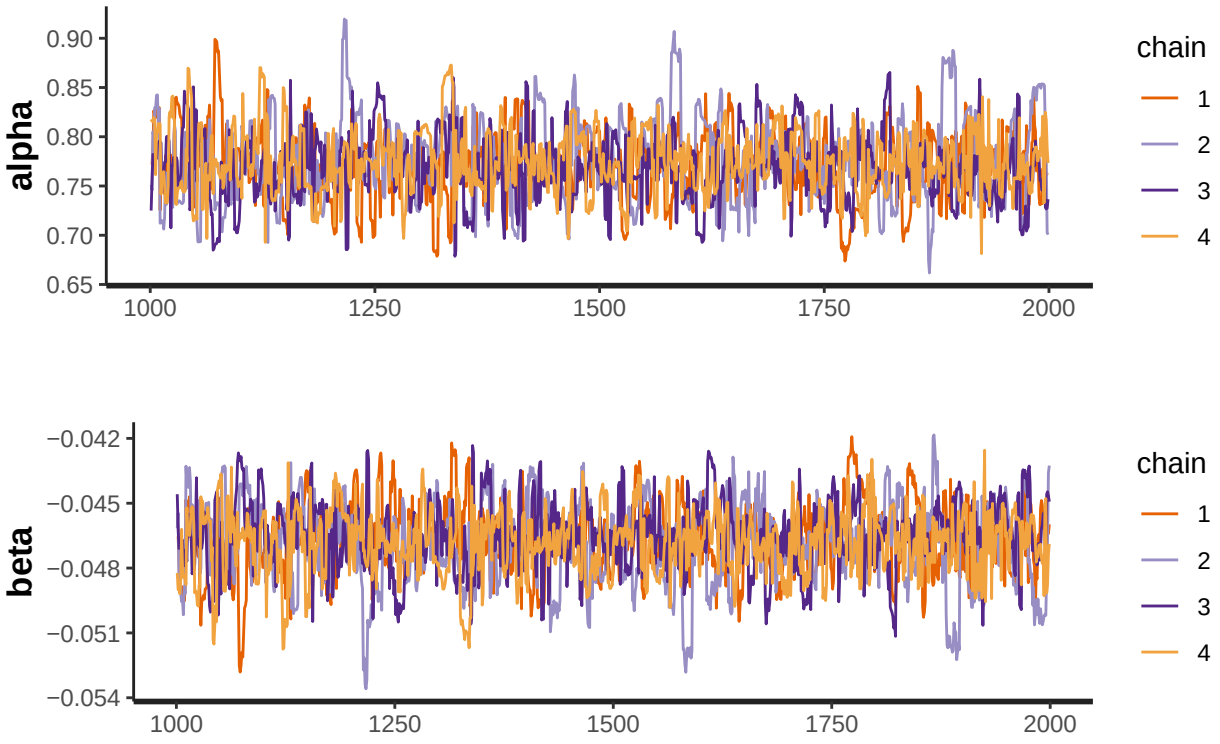
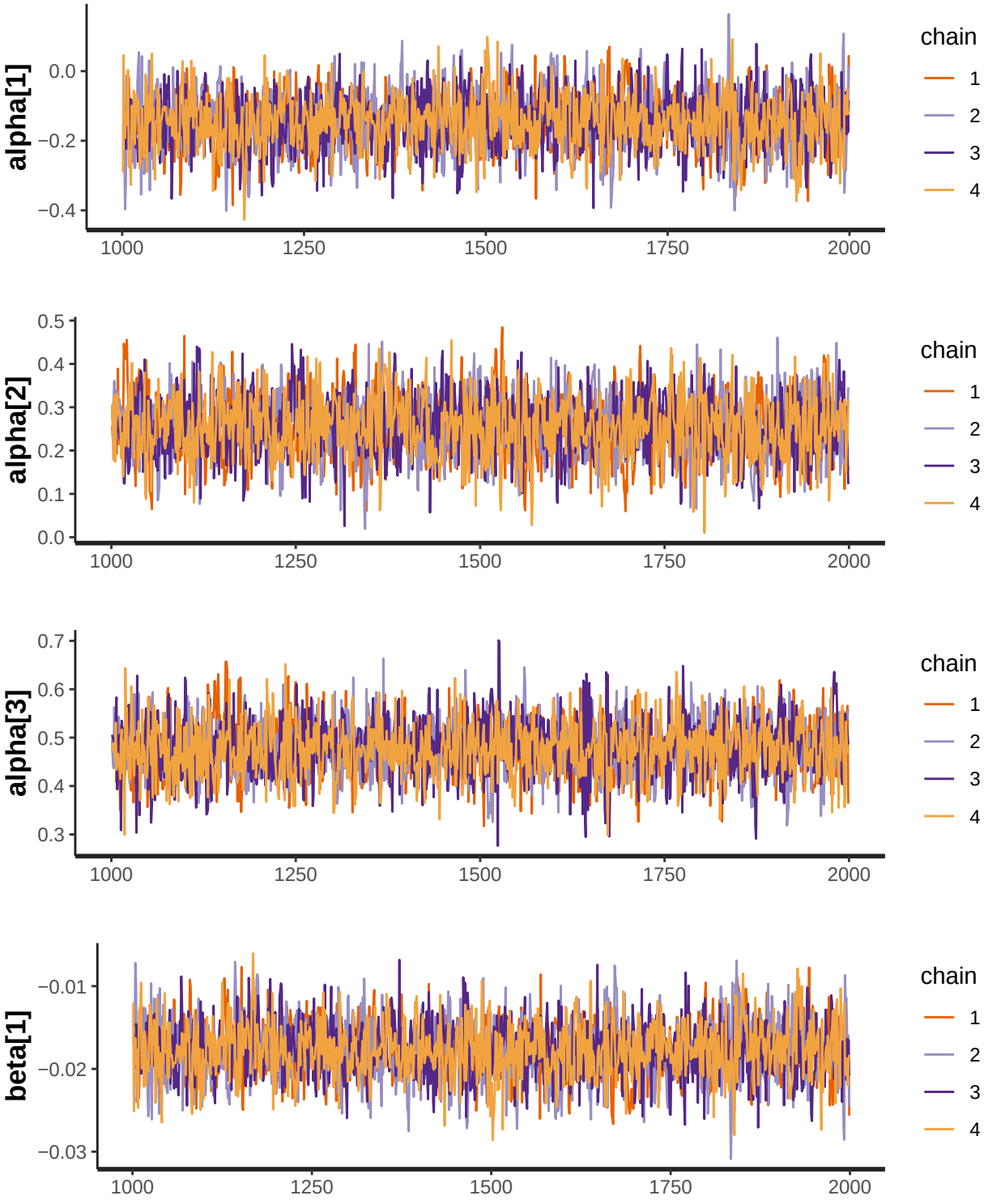


Figure 1: Traceplots of the pooled model parameters

Figure 1 above suggests that all chains seem to converge for both of the pooled model parameters.

7.1.2 Hierarchical model



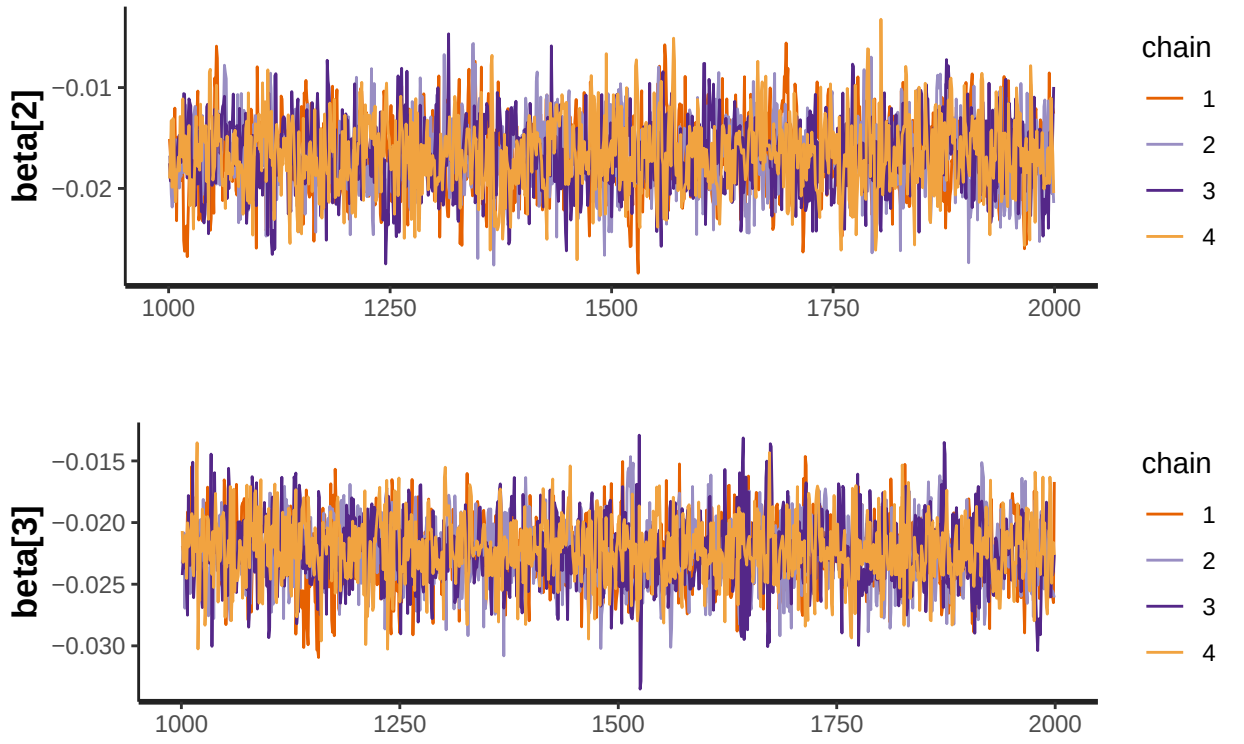


Figure 2: Traceplots of the hierarchical model parameters

By the traceplots in Figure 2, the hierarchical model chains seem to have converged as well. To ensure that the chains have absolutely converged, we evalute the potential scale reduction statistic $\hat{R}$ as well as the effective sample sizes (bulk and tail ESS).

7.2 Potential scale reduction statistic $\hat{R}$

The potential scale reduction statistic $\hat{R}$ assesses convergence by evaluating within-chain and between-chain variances (BDA3 book, p.278). As a rule of thumb, the closer $\hat{R}$ is to 1, the better the chains have converged. If $\hat{R}$ is less than 1.1 (BDA3 book, p. 287) or 1.05 (rstan output), we consider the model chains to have converged. We will assess these values below once all diagnostics have been introduced.

7.3 Effective sample size (ESS)

The effective sample size evaluates the general error of the mean and the effectiveness and stability of the estimates. Usually high sample variance results in a smaller ESS. In general, ESS, coupled with the Monte Carlo Standard Errors ($MCSE$) give "a sense of whether the simulations suffice for practical purposes" (BDA3 book, p. 594). If a simulation has a higher ESS, the mean have lower standard errors and more stable estimates. Below, we will evaluate ESS based on the two bulk and tail diagnostics. The R documentation for calculating those statistics suggest that these values are "improved versions" of the traditional Sample Size diagnostic. The bulk and tail functions split up the sample size evaluation into two separate sections, the bulk of the samples as well as the extreme sample quantiles at the tail of the distributions. For efficient estimates, both bulk and tail ESS should be over approximately 100 per chain. This means that our model ESS threshold is 400.

7.4 Divergence diagnostics

Since the traceplots seem to show chain convergence, we will briefly check the divergence diagnostics. This can be done with the rstan package (see https://mc-stan.org/rstan/reference/check_hmc_diagnostics.html for more details):

```
check_divergences(fit_pool)
```

```
## 0 of 4000 iterations ended with a divergence.
```

```
check_divergences(fit_hier)
```

```
## 0 of 4000 iterations ended with a divergence.
```

From the output above, we can see that no divergence was evident. For sanity, we will check the number of iterations that hit maximum tree depth:

```
check_treedepth(fit_pool)
```

```
## 0 of 4000 iterations saturated the maximum tree depth of 10.
```

```
check_treedepth(fit_hier)
```

```
## 0 of 4000 iterations saturated the maximum tree depth of 10.
```

The model chains seem to have converged nicely.

Pooled model summary

We can access all of these statistics with the rstan function **monitor**:

```
monitor(fit_pool)
```

```
## Inference for the input samples (4 chains: each with iter = 2000; warmup = 0):
##
##                  Q5      Q50      Q95     Mean  SD  Rhat Bulk_ESS Tail_ESS
## alpha           0.7      0.8      0.8      0.8 0.0  1.01      576      516
## beta            0.0      0.0      0.0      0.0 0.0  1.01      596      586
## logit_p[1]      0.1      0.1      0.1      0.1 0.0  1.01      639      673
## logit_p[2]      0.0      0.0      0.0      0.0 0.0  1.01      673      731
## logit_p[3]      0.0      0.0      0.0      0.0 0.0  1.01      735      789
## logit_p[4]     -0.1     -0.1     -0.1     -0.1 0.0  1.00      851     1015
## logit_p[5]     -0.1     -0.1     -0.1     -0.1 0.0  1.00     1132     1572
## logit_p[6]     -0.2     -0.2     -0.2     -0.2 0.0  1.00     1756     2233
## logit_p[7]     -0.2     -0.2     -0.2     -0.2 0.0  1.00     3232     2811
## logit_p[8]     -0.3     -0.3     -0.2     -0.3 0.0  1.00     3826     2789
## logit_p[9]     -0.3     -0.3     -0.3     -0.3 0.0  1.00     2670     2614
## logit_p[10]    -0.4     -0.3     -0.3     -0.3 0.0  1.00     1823     2196
## logit_p[11]    -0.4     -0.4     -0.4     -0.4 0.0  1.00     1347     1772
```

```
## logit_p[12]     -0.5     -0.4     -0.4     -0.4 0.0  1.00     1090     1377
## logit_p[13]     -0.5     -0.5     -0.5     -0.5 0.0  1.00      947     1132
## logit_p[14]     -0.6     -0.5     -0.5     -0.5 0.0  1.01      866     1037
## logit_p[15]     -0.6     -0.6     -0.6     -0.6 0.0  1.01      812      889
## logit_p[16]     -0.7     -0.6     -0.6     -0.6 0.0  1.01      774      913
## log_lik[1]     -10.3     -7.7     -5.9     -7.8 1.4  1.01      639      673
## log_lik[2]      -6.0     -5.0     -4.8     -5.1 0.4  1.01      796      731
## log_lik[3]      -5.2     -4.8     -4.8     -4.9 0.2  1.01     1136     1716
## log_lik[4]      -9.7     -7.9     -6.5     -8.0 1.0  1.00      851     1015
## log_lik[5]      -7.6     -6.5     -5.7     -6.6 0.6  1.00     1132     1572
## log_lik[6]      -8.8     -7.7     -6.8     -7.7 0.6  1.00     1756     2233
## log_lik[7]      -4.6     -4.4     -4.3     -4.4 0.1  1.00     3210     2933
## log_lik[8]     -12.3    -10.9     -9.7    -10.9 0.8  1.00     3826     2789
## log_lik[9]      -5.0     -4.7     -4.6     -4.7 0.2  1.00     2649     2713
## log_lik[10]     -5.1     -4.9     -4.8     -4.9 0.1  1.00     2199     2563
## log_lik[11]     -6.9     -5.5     -5.1     -5.7 0.6  1.00     1504     2059
## log_lik[12]     -7.5     -6.0     -5.1     -6.1 0.8  1.00     1094     1377
## log_lik[13]     -4.9     -4.5     -4.3     -4.5 0.2  1.00     1003     1132
## log_lik[14]     -6.6     -5.8     -5.2     -5.9 0.4  1.01      866     1037
## log_lik[15]     -3.9     -3.7     -3.5     -3.7 0.1  1.01      812      889
## log_lik[16]     -3.3     -3.2     -3.0     -3.2 0.1  1.01      774      913
## lp__        -71061.3 -71058.9 -71058.2 -71059.2 1.1  1.01      835      859
##
## For each parameter, Bulk_ESS and Tail_ESS are crude measures of
## effective sample size for bulk and tail quantities respectively (an ESS > 100
## per chain is considered good), and Rhat is the potential scale reduction
## factor on rank normalized split chains (at convergence, Rhat <= 1.05).
```

From the output above, we can see that all of the convergence diagnostics seem to be relatively stable and satisfy the convergence conditions. Since the output above supports our traceplot observations for the pooled model, we can conlcude that the model seems to have converged and is stable.

Further, we can assess the Monte Carlo Standard Error ($MSCE$) values, which can be calculated with the following formula (BDA3, p.267):

$$MCSE = \frac{s_\theta}{\sqrt{S}}$$

, or the sample standard deviation divided by the square root of the sample size. This can be calculated with the package **mcmcse**:

Pooled MCSE

```
mc_a = mcmcse::mcse(params_pool$alpha)
mc_b = mcmcse::mcse(params_pool$beta)
data.frame('Estimate' = c(mc_a$est, mc_b$est)
           , 'MCSE' = c(mc_a$se, mc_b$se)
           , row.names = c('alpha', 'beta'))
```

```
##         Estimate         MCSE
## alpha  0.7724908 5.762539e-04
## beta  -0.0467391 2.681046e-05
```

Hence, out modeled parameters for the pooled model are (according to appropriate $MCSE$ reporting standards):

- $\alpha = 0.772$
- $\beta = -0.0467$

Hierarchical model

Next, the hierarchical model:

```
monitor(fit_hier)
```

```
## Inference for the input samples (4 chains: each with iter = 2000; warmup = 0):
##
##                   Q5      Q50      Q95     Mean  SD  Rhat Bulk_ESS Tail_ESS
## mu0             -0.1      0.1      0.3      0.1 0.1  1.00     2185     1520
## sigma            0.2      0.3      0.6      0.3 0.1  1.00     1916     1769
## alpha[1]        -0.3     -0.1      0.0     -0.1 0.1  1.00     2209     1994
## alpha[2]         0.1      0.3      0.4      0.3 0.1  1.00     2163     2022
## alpha[3]         0.4      0.5      0.6      0.5 0.1  1.00     2743     2305
## beta[1]          0.0      0.0      0.0      0.0 0.0  1.00     2229     2129
## beta[2]          0.0      0.0      0.0      0.0 0.0  1.00     2239     2137
## beta[3]          0.0      0.0      0.0      0.0 0.0  1.00     2770     2202
## log_lik[1]      -6.0     -4.7     -3.9     -4.8 0.7  1.00     2262     2116
## log_lik[2]      -4.8     -4.0     -3.6     -4.1 0.4  1.00     2313     2072
## log_lik[3]      -4.2     -3.8     -3.7     -3.8 0.2  1.00     2441     2659
## log_lik[4]      -7.9     -6.5     -5.3     -6.5 0.8  1.00     2360     2314
## log_lik[5]      -4.3     -3.9     -3.7     -3.9 0.2  1.00     2495     2857
## log_lik[6]      -4.2     -3.9     -3.7     -3.9 0.2  1.00     2587     2906
## log_lik[7]      -3.7     -3.6     -3.6     -3.6 0.1  1.00     2813     3216
## log_lik[8]      -6.2     -5.3     -4.7     -5.3 0.5  1.00     3287     3234
## log_lik[9]     -12.8    -10.7     -9.0    -10.8 1.2  1.00     3858     2924
## log_lik[10]     -6.7     -5.6     -4.9     -5.7 0.6  1.00     4013     2772
## log_lik[11]     -5.9     -5.0     -4.8     -5.1 0.4  1.00     3490     3086
## log_lik[12]     -7.2     -5.7     -4.8     -5.8 0.8  1.00     3269     3247
## log_lik[13]     -5.1     -4.4     -4.1     -4.4 0.3  1.00     2982     3062
## log_lik[14]     -9.8     -8.4     -7.2     -8.4 0.8  1.00     2746     2823
## log_lik[15]     -4.5     -4.0     -3.7     -4.0 0.2  1.00     2621     2765
## log_lik[16]     -3.6     -3.3     -3.1     -3.3 0.2  1.00     2539     2667
## log_lik[17]     -6.0     -4.7     -4.3     -4.9 0.6  1.00     2463     3042
## log_lik[18]     -5.1     -4.4     -4.3     -4.5 0.3  1.00     2991     2947
## log_lik[19]     -5.7     -4.7     -4.3     -4.8 0.4  1.00     3059     2812
## log_lik[20]     -5.0     -4.5     -4.2     -4.5 0.3  1.00     3435     2714
## log_lik[21]     -6.2     -5.3     -4.6     -5.3 0.5  1.01     4077     2678
## log_lik[22]     -4.1     -3.9     -3.9     -3.9 0.1  1.00     2170     2267
## log_lik[23]    -11.8    -10.3     -9.0    -10.4 0.8  1.00     3666     2945
## log_lik[24]     -5.4     -4.7     -4.2     -4.8 0.4  1.00     3304     3161
## log_lik[25]     -4.0     -3.7     -3.5     -3.7 0.1  1.00     3073     3135
## log_lik[26]     -6.2     -4.9     -4.1     -5.0 0.6  1.00     2847     3090
## log_lik[27]     -5.0     -4.1     -3.9     -4.2 0.4  1.00     2930     3021
## log_lik[28]     -7.8     -6.1     -4.8     -6.2 0.9  1.00     2632     2921
## log_lik[29]     -4.4     -3.8     -3.3     -3.8 0.3  1.00     2571     2812
## log_lik[30]     -2.6     -2.4     -2.3     -2.4 0.1  1.00     2524     2738
```

```
## log_lik[31]     -2.3     -2.1     -2.0     -2.1 0.1  1.00     2488     2631
## log_lik[32]     -2.1     -2.0     -1.9     -2.0 0.1  1.00     2462     2636
## log_lik[33]     -5.1     -4.4     -4.3     -4.5 0.3  1.00     2895     2827
## log_lik[34]     -4.9     -4.5     -4.4     -4.5 0.2  1.00     2291     2757
## log_lik[35]     -5.3     -4.6     -4.4     -4.7 0.3  1.00     3085     3025
## log_lik[36]     -6.8     -5.6     -4.8     -5.6 0.6  1.00     3265     2806
## log_lik[37]     -6.9     -5.8     -5.0     -5.9 0.6  1.00     3550     2442
## log_lik[38]     -4.9     -4.5     -4.3     -4.5 0.2  1.00     3834     2444
## log_lik[39]     -4.2     -4.0     -4.0     -4.1 0.1  1.00     4086     2952
## log_lik[40]     -3.7     -3.7     -3.6     -3.7 0.0  1.00     2612     3061
## log_lik[41]     -9.5     -8.5     -7.6     -8.5 0.6  1.00     3661     3418
## log_lik[42]     -7.5     -6.0     -5.0     -6.1 0.8  1.00     3476     3491
## log_lik[43]     -7.6     -5.8     -4.7     -5.9 0.9  1.00     3330     3623
## log_lik[44]    -10.1     -7.8     -6.0     -7.9 1.2  1.00     3221     3336
## log_lik[45]     -4.1     -3.8     -3.6     -3.8 0.2  1.00     3140     3259
## log_lik[46]     -3.7     -3.4     -3.2     -3.4 0.1  1.00     3077     2903
## log_lik[47]     -3.4     -3.1     -2.9     -3.2 0.1  1.00     3036     2842
## log_lik[48]     -4.0     -3.7     -3.4     -3.7 0.2  1.00     2997     2767
## lp__        -70487.1 -70482.9 -70480.5 -70483.2 2.1  1.00     1577     2387
##
## For each parameter, Bulk_ESS and Tail_ESS are crude measures of
## effective sample size for bulk and tail quantities respectively (an ESS > 100
## per chain is considered good), and Rhat is the potential scale reduction
## factor on rank normalized split chains (at convergence, Rhat <= 1.05).
```

From the output above, it seems that the same holds for the hierarchical model: $\hat{R}$ seems to be very close to the optimal value of 1 and the ESS values are very high. Therefore, we conclude that the chains must are converged by the evidence of this output and the traceplots shown above.

The estimates and $MCSE$s for the hierarchical model:

```
## [1] "Jump Shot:"
##           Estimate         MCSE
## alpha1 -0.14609129 1.391781e-03
## beta1  -0.01771511 5.825243e-05
## [1] "Fadeaway:"
##           Estimate         MCSE
## alpha2  0.25633144 1.107546e-03
## beta2  -0.01674883 5.735007e-05
## [1] "Pullup:"
##           Estimate         MCSE
## alpha3  0.47842527 8.864385e-04
## beta3  -0.02238259 4.309139e-05
```

Hence, we have the following model parameters:

Jump shot

- $\alpha_1 = -0.15$
- $\beta_1 = -0.0177$

Fadeaway/stepback jump shot

- $\alpha_2 = 0.26$
- $\beta_2 = 0.0167$

Pullup jump shot

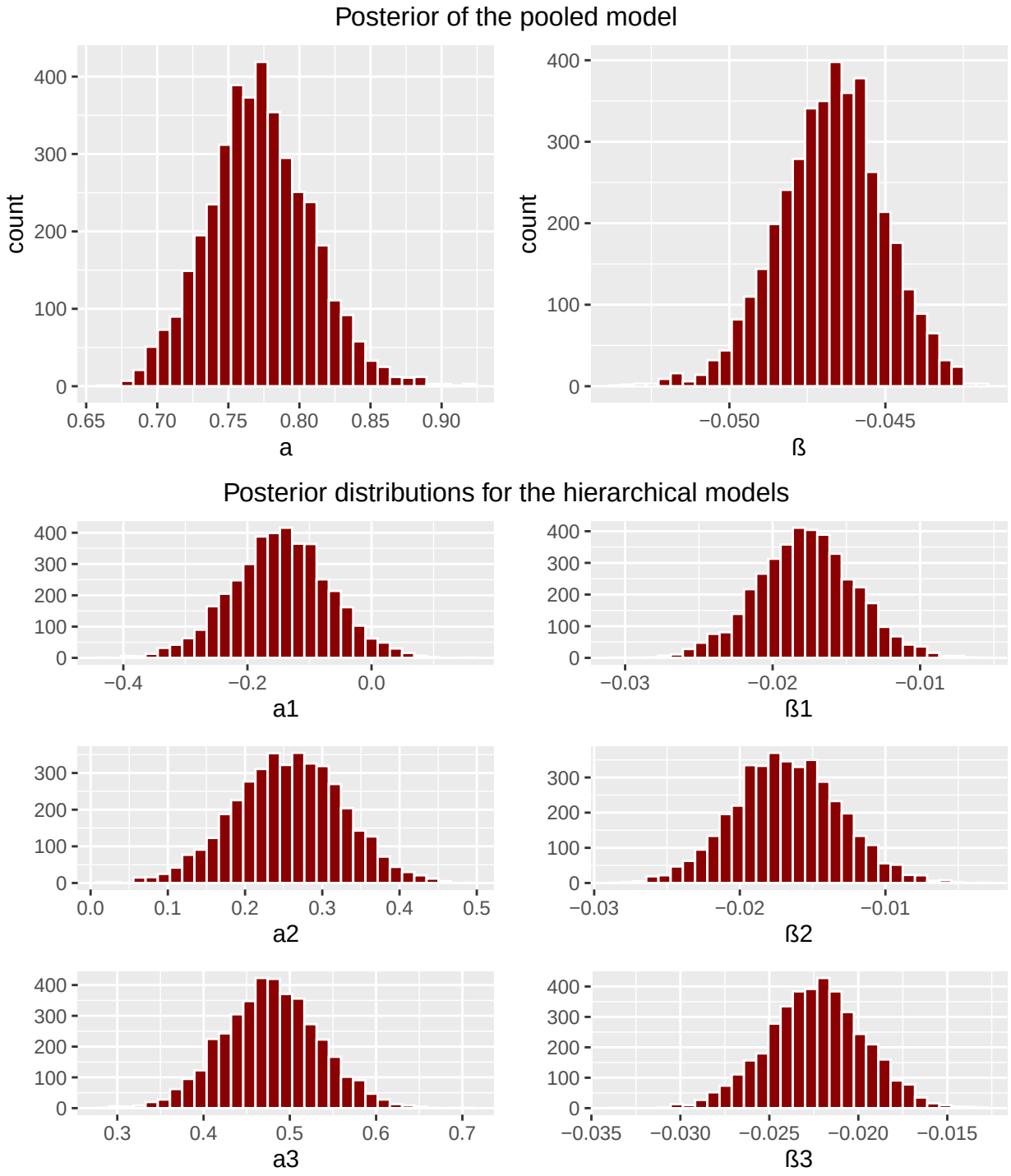
- $\alpha_3 = 0.478$
- $\beta_3 = -0.0224$

We will further assess these parameters in Sections below.

8. Posterior predictive checks and what was done to improve the model.

8.1 Posterior distributions

We can plot the posterior distributions



It can be observed that posterior distributions for pooled model and hierarchical models are sampled well.

For each 4000 sampled probabilities for α_i and β_i in each sampling, the probability is fitted to separate histograms.

8.2 Posterior Predictive Checking (PPC)

The most efficient way to do posterior predictive checking for our models is to plot the test data (as a plot of success/throw ratio vs distance) with the sampled posterior parameters. In the figures, 1000 posterior parameter pairs are randomly chosen, fitted on the test data and drawn. The parameter estimate (mean) will then be plotted in red to signify the true predictive estimate.

8.2.1 Pooled Model

```
plot_ppc(test$df_pooled, params_pool$alpha, params_pool$beta, 'Pooled Model')
```

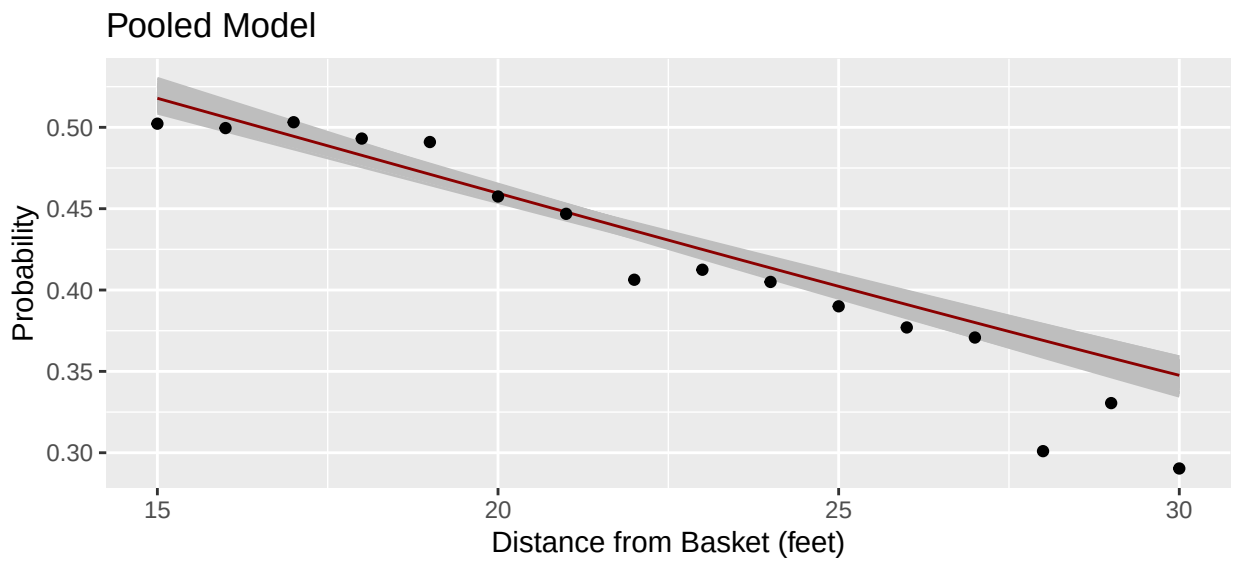
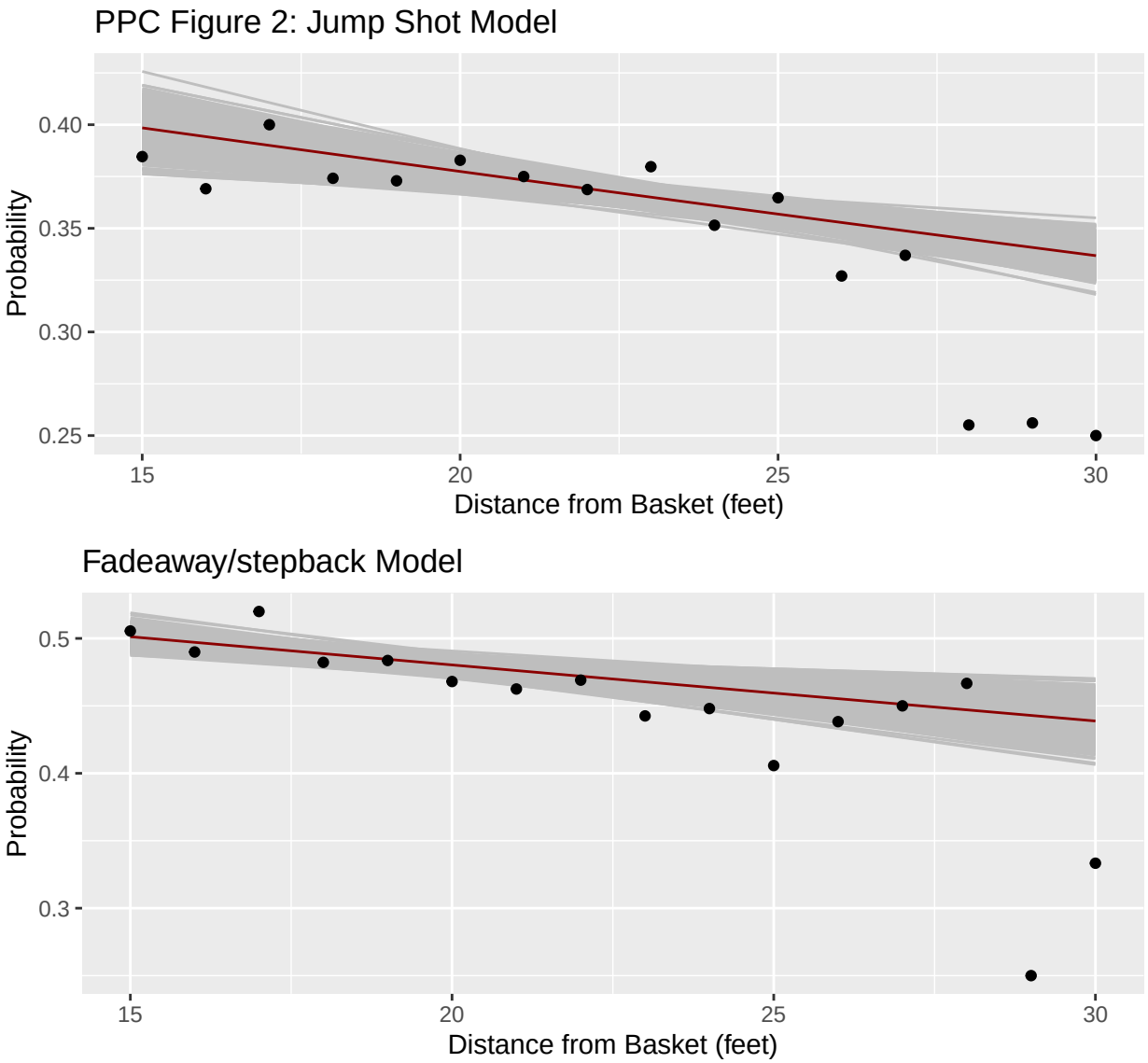


Figure 3: Posterior preditive plot of the pooled model

8.2.2 Hierarchical Model



Overall, it seems that the model is capable of moderate posterior predictability - we can develop posterior draws that are more or less descriptive of the data. Since these figures can also be used as tools in model comparison and performance assessment, we will further discuss them in the following sections.

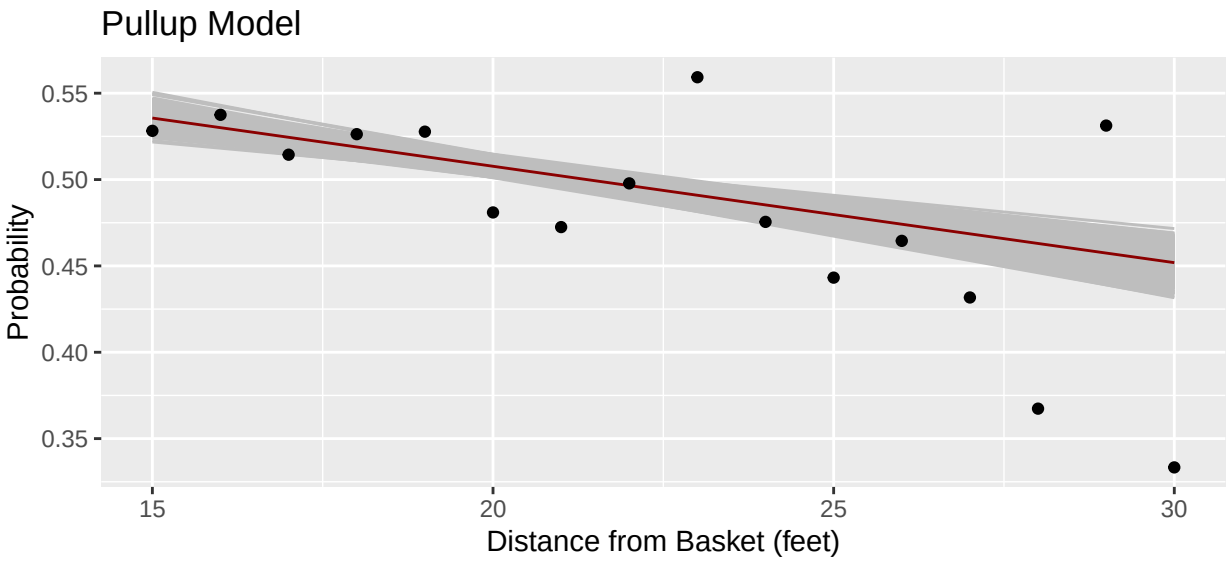


Figure 4: Posterior preditive plots of the hierarchical model

9. Model Comparison

9.1 PSIS-LOO Diagnostics

The majority of the model comparison diagnostics can be evaluated with the rstan function **loo**: it does most of the heavy lifting and returns different Pareto Smoothed Importance Sampling (PSIS) Leave-one-out (LOO) diagnostics.

```
loo_pool = loo(fit_pool, pars = 'log_lik')
loo_hier = loo(fit_hier, pars = 'log_lik')

loo_pool
```

```
##
## Computed from 4000 by 16 log-likelihood matrix
##
##          Estimate   SE
## elpd_loo    -98.1  8.8
## p_loo         6.2  2.3
## looic       196.1 17.6
## ------
## Monte Carlo SE of elpd_loo is NA.
##
## Pareto k diagnostic values:
##                          Count Pct.    Min. n_eff
## (-Inf, 0.5]   (good)     14    87.5%   189
##  (0.5, 0.7]   (ok)        1     6.2%   463
##    (0.7, 1]   (bad)       1     6.2%   8
##    (1, Inf)   (very bad)  0     0.0%   <NA>
## See help('pareto-k-diagnostic') for details.
```

```
loo_hier
```

```
##
## Computed from 4000 by 48 log-likelihood matrix
##
##          Estimate   SE
## elpd_loo   -243.7 13.8
## p_loo        12.7  2.5
## looic       487.4 27.5
## ------
## Monte Carlo SE of elpd_loo is 0.1.
##
## Pareto k diagnostic values:
##                            Count Pct.    Min. n_eff
## (-Inf, 0.5]   (good)     44    91.7%   835
##  (0.5, 0.7]   (ok)        4     8.3%   203
##    (0.7, 1]   (bad)       0     0.0%   <NA>
##    (1, Inf)   (very bad)  0     0.0%   <NA>
##
## All Pareto k estimates are ok (k < 0.7).
## See help('pareto-k-diagnostic') for details.
```

The $elpd_{loo}$ (expected log predictive density) value evaluates the predictive accuracy of the model, and the model with the larger $elpd_{loo}$ is usually referred to as the better model. From the output above, we can see that the pooled model $elpd_{loo}$ is approximately -98 where as the same value for the hierarchical model is -244. Therefore, based on this, it seems that the pooled model explains the data better. However, there is the issue of the size of the log-likelihood vector: since shots from the each shot type contribute to the number of successes of the pooled model observations, we have a different number of posterior log-likelihood points for each model. From the output above, one can see that the pooled model loos were calculated from a 4000x16 matrix while the hierarchical model used a 4000x48. This is paradoxical, since $elpd_{loo}$ is relative to the number of data log-likelihood observations (for more resources see Aki Vehtari's article that was covered suring Assignment 8), we can expect the 4000x48 matrix to yield significantly worse $elpd_{loo}$ values.

Overall, it is very difficult to compare $elpd_{loo}$ values of different sample sizes, but as a guideline, the sample size of the hierarchical $elpd_{loo}$ was three times as large as the sample size of the sample size. In comparison, three times the $elpd_{loo}$ of the pooled model is significantly larger than the $elpd_{loo}$ of the hierarchical. Although these multiplication-like operations may not be translatable for $elpd_{loo}$ values, this comparison shows that even though $elpd_{loo}$ would suggest it as such, we cannot conlcude that the pooled model is absolutely better on that value alone.Indeed, if we would separate the pooled data as separate samples for each shot type, we the $elpd_{loo}$ value of the poled model would be:

```
## Warning: Some Pareto k diagnostic values are too high. See help('pareto-k-diagnostic') for details.
```

```
##
## Computed from 4000 by 48 log-likelihood matrix
##
##          Estimate    SE
## elpd_loo   -844.7 104.2
## p_loo        67.0  18.6
## looic      1689.4 208.5
## ------
## Monte Carlo SE of elpd_loo is NA.
##
```

```
## Pareto k diagnostic values:
##                          Count Pct.    Min. n_eff
## (-Inf, 0.5]   (good)     42    87.5%   100
##  (0.5, 0.7]   (ok)        2     4.2%   99
##    (0.7, 1]   (bad)       0     0.0%   <NA>
##    (1, Inf)   (very bad)  4     8.3%   5
## See help('pareto-k-diagnostic') for details.
```

, which we can see is a lot larger than the $elpd_{loo}$ we had for each of the models defined above. However, this introduces a paradoxical problem in defining the pooled model, since pooled model used and defined earlier should as well be justified, assuming that pooled model assumption holds.

9.2 Pareto k-values

We can further evaluate the realibility of the PSIS-LOO estimates with the help of Pareto k-values ($\hat{k}$). Superficially, it is enough for us to know (BDA Assignment 8), that if all $\hat{k}$ values are approximately less than 0.7 the PSIS-LOO estimates can be considered reliable. From the outputs of the **loo** function we already received information about those values, but we can visualize them for easier understanding:

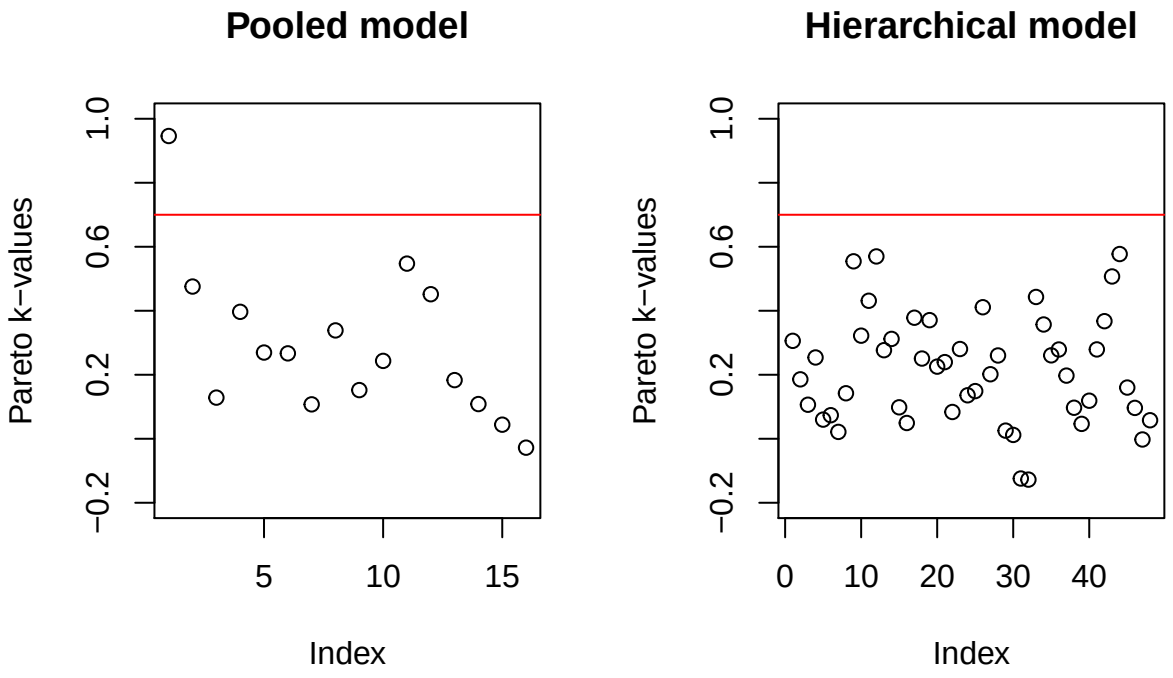


Figure 5: Plot of Pareto k-values of both models

It can be seen above that all except one (0.95) of the Pooled model $\hat{k}$-values are below 0.7. However, this is only one of the $\hat{k}$-values, and even though this decreases model similarity, we should not reject the model based on one outlier alone. On the other hand, all of the hierarchical $\hat{k}$-values seem to be ok, and we may conclude that the hierarchical PSIS-LOO estimates are reliable. Therefore, model estimates are mostly reliable and we can conclude that models are not distinguishable based on the $\hat{k}$-values.

10. Predictive performance assessment

The final model comparison can be done during predictive performance assessment, since it will help us distinguish the better model. The first evaluation tool will be the posterior predictive (PPC) plots.

10.1 Posterior predictive plots

In Section 8.2, we showed Figures 3 and 4, which were fitted posterior predictive figures. The figures showed the test data, the model parameter estimates, and a distribution of randomly sampled predictive posterior curves in grey. In a crude way, the grey area in those figures can be seen as a confidence interval - the larger the posterior uncertainty, the more variance in the model parameters and wider the grey "model confidence interval" area (since the curve parameters can assume a wider range of values). Figure 3 shows that the fitted model curves are concentrated around a much narrower range that the curves on Figure 4. This would suggest that the pooled model is a more efficient predictive estimator than the hierarchical model. This would support the (although uncomparable) $elpd_{loo}$ findings presented in Section 9.1.

From Figure 4 it can be see that all of the hierarchical shot type groups seem to have outliers in the end range of shot distance. For all of the model groups, these were points that the predictive model could not account for. This again suggests that the hierarchical model estimators are less efficient. Finally, the test data observations of hierarchical models seems to be significantly more noisy than the pooled model observations. Although this could simply be due to smaller sample size due to the partitioning of data into smaller groups, the increased noise of the data points further shows that according to our model, the predictability of the shots suffer when taking shot type into consideration.

10.2 Residuals

Plotting the residual (error) plots for the models:

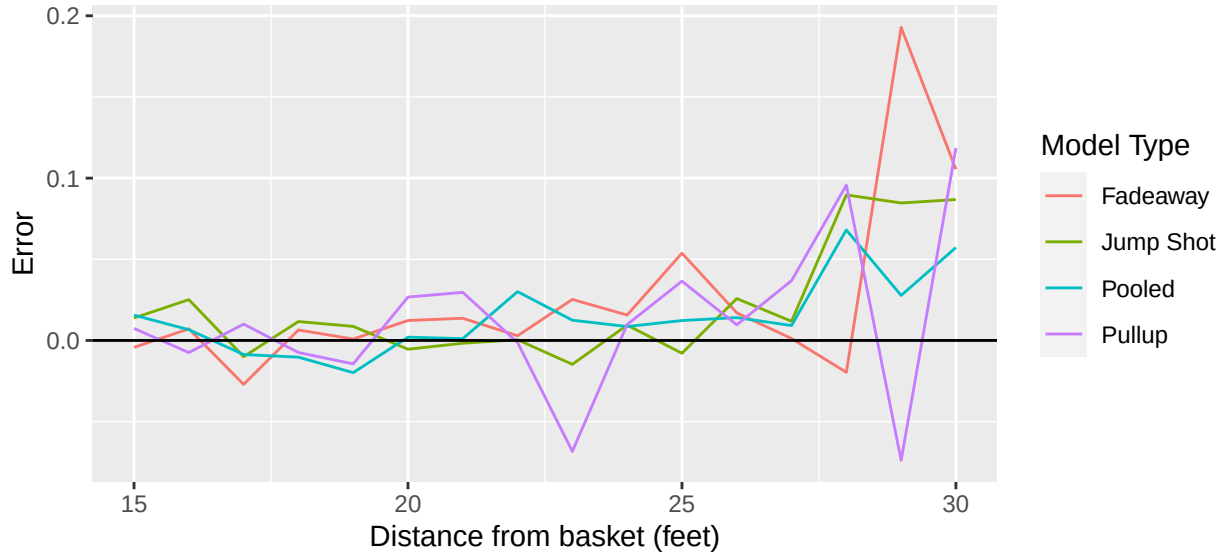


Figure 6: Model error plots

From Figure 6 above, we can wee that most of the models are relatively stationary (realtive errors do not differ from each other), until about 26 feet, after which the models diverge from the data. Of the four models, however, the pooled model seems to follow the data the best, as its residuals do not become too extreme. The Fadeaway model and the Jump Shot model seem to perform relatively up to par with the pooled model

up to this point, after which their errors diverge. The Pullup model seems to be the most poorly predictable as its residuals are the largest throughout.

Overall, it seems that pooled model is able to predict the shot likelihoods the best. Although the hierarchical model is able to distinguish differences between the shot type groups (estimated parameters are different, section 7), the model's predictive accuracy is poor in comparison to the pooled model. Moreover, the $elpd_{loo}$ values of the pooled model were (somewhat questionably) significantly larger, and the posterior predictive model curves were more narrowly defined.

11. Sensitivity analysis with respect to prior choices (i.e. checking whether the result changes a lot if prior is changed)

11.1 Pooled model

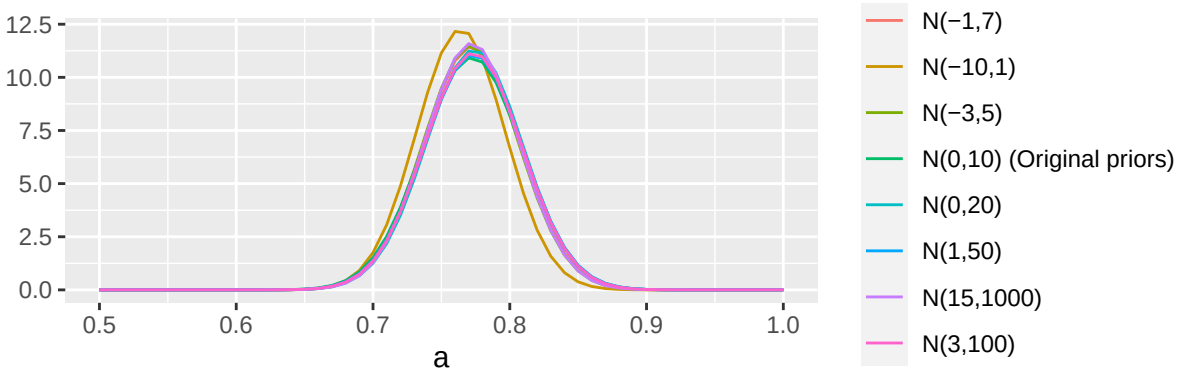
To assess model sensitivity, we compare changes to both parameters for the priors of α and β. We define a vector of 7 different μ_i values that we use to evaluate sensitivity:

$$\mu = [-10, -3, -1, 0, 1, 3, 15] \; \sigma = [1, 5, 7, 20, 50, 1000]$$

, in other words, we test the posterior changes for prior mean increments of two.

```
## Warning: Tail Effective Samples Size (ESS) is too low, indicating posterior variances and tail quanti
## Running the chains for more iterations may help. See
## http://mc-stan.org/misc/warnings.html#tail-ess
```

```
## Warning: Tail Effective Samples Size (ESS) is too low, indicating posterior variances and tail quanti
## Running the chains for more iterations may help. See
## http://mc-stan.org/misc/warnings.html#tail-ess
```



11.2 Hierarchical model

We will use the same test prior values for the hierarchical prior hyperparameters:

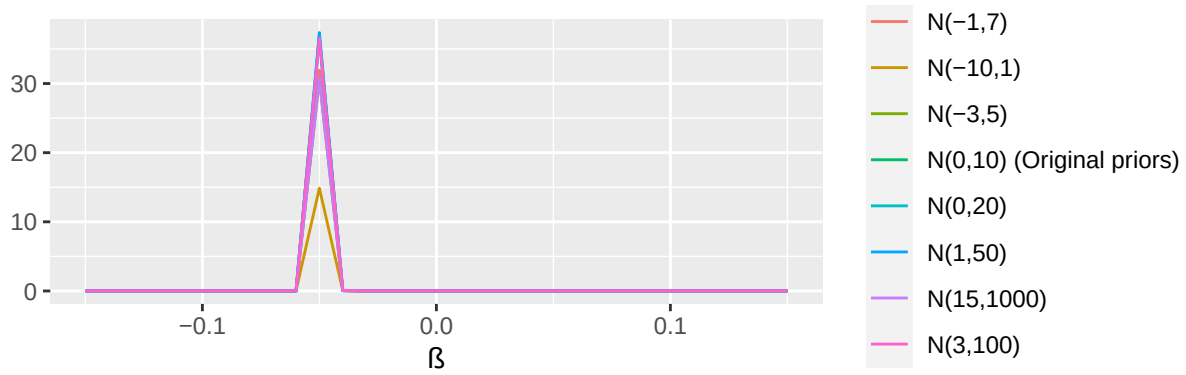


Figure 7: Sensitivity plots of the pooled model

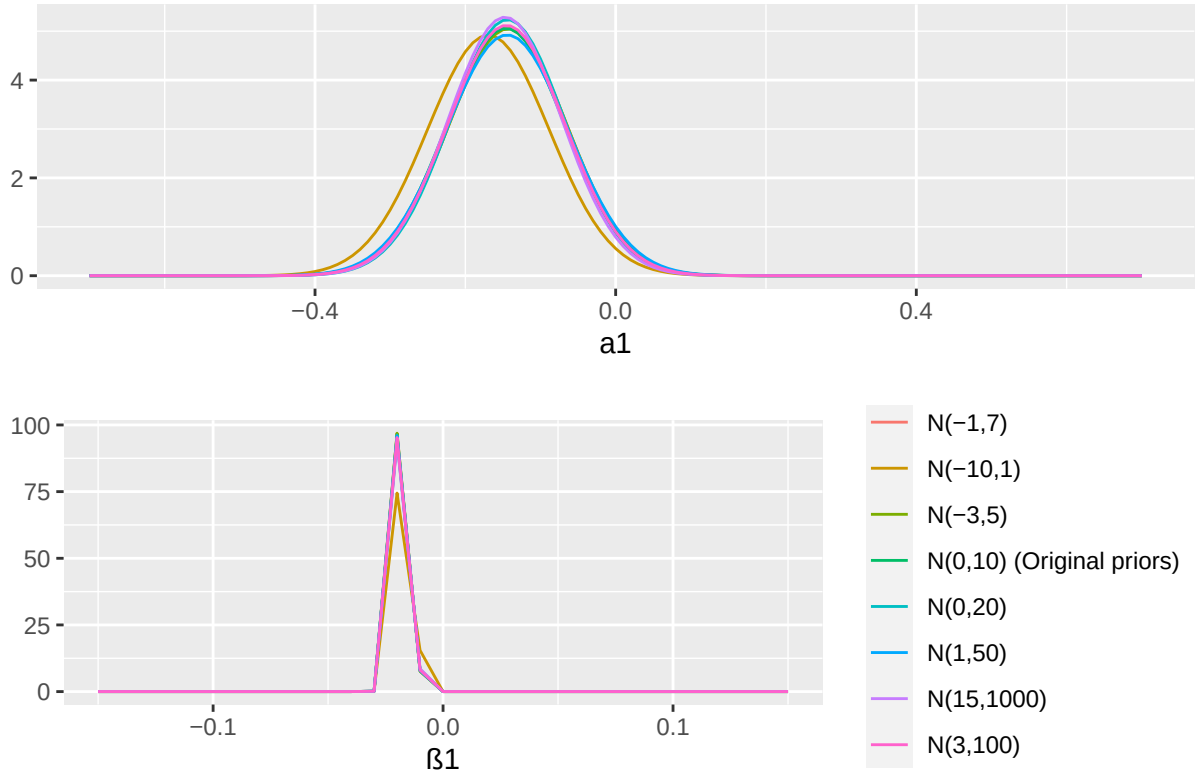
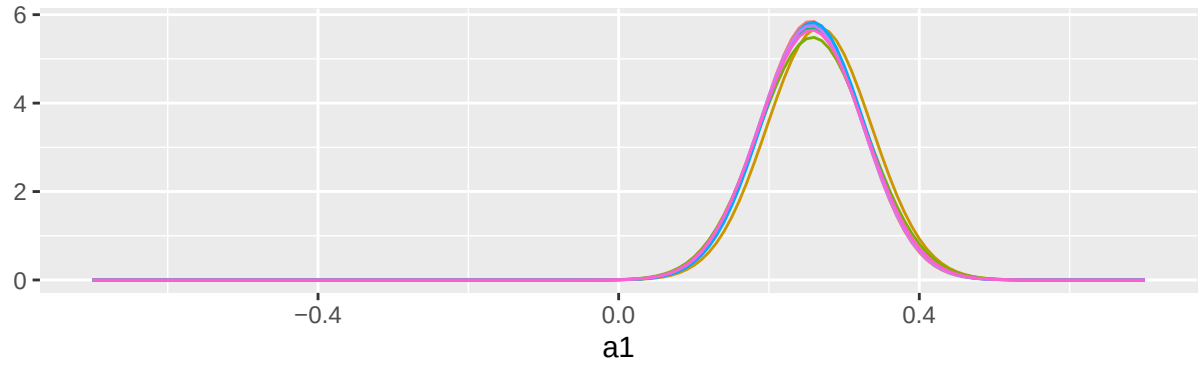
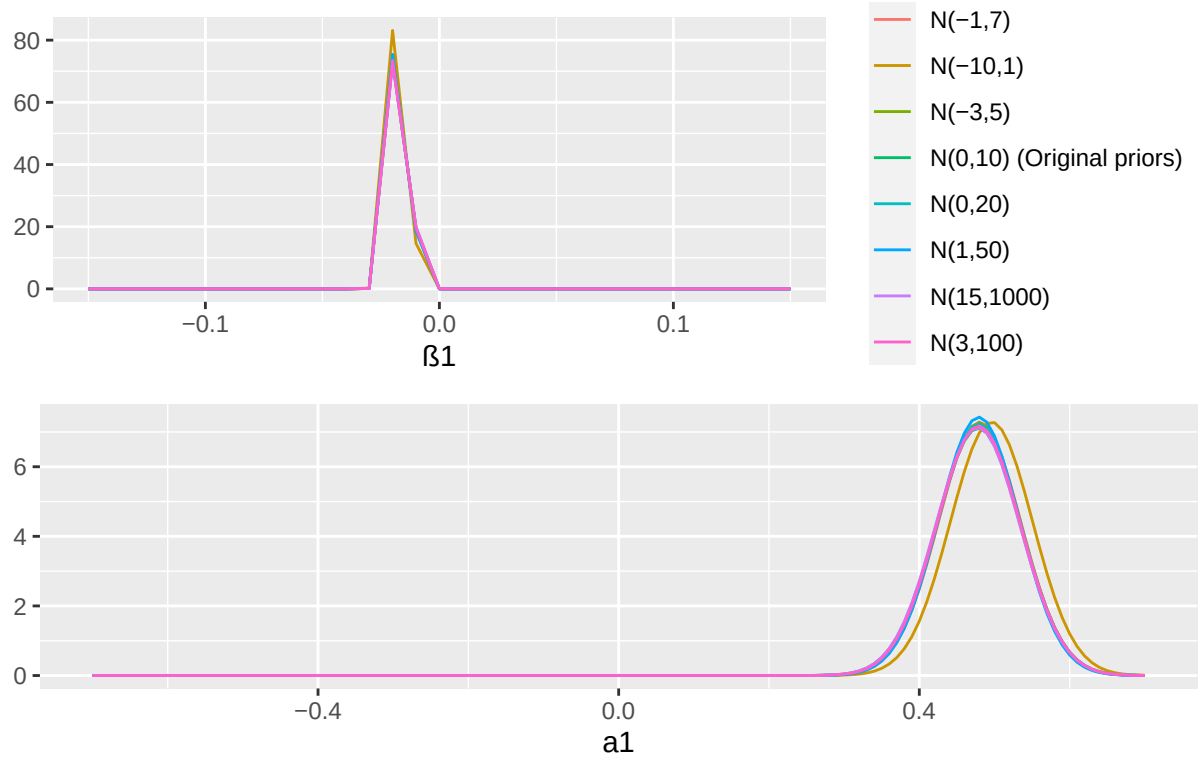


Figure 8: Sensitivity plot for the Jump Shot model



b1



From the Figures above, it can be observed that the changes in the values of the priors does not have a significant impact to the model. Therefore, twe can conclude that the models are not sensitive to the definition of our priors.

12. Discussion

12.1 Data

The data model was a little problematic because the sample size for Fadeaway/Stepback shot type varied compared to the two other shot types. Modeling could have been even more accurate if the amount of attempted shots would have been the same for each shot type. However, the amount of data for each shot type was sufficient for accurate analysis. In addition, the shot distances were quite widely partitioned into

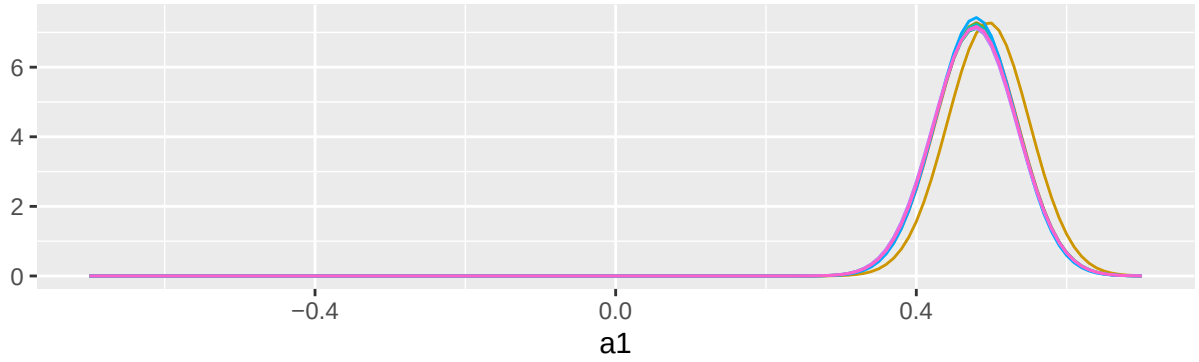


Figure 9: Sensitivity plot of the Pullup model

whole numbers. It would have been interesting to be able to analyze the effect of shot distance without this categorization of distances, especially close to the three point line. Another issue with our data and modeling was that the sample size for pooled model was three times as big as in hierarchical models. This was caused by division of the data to different shot types. This can have an effect to results.

Finally, a factor that increased model uncertainty was the compression of data into success ratios. Although the success ratio gave a better understanding of the shot-making relationship, the process compressed over 120000 data points into 16. This loss of information means that there was incerased predictive unceratinty, especially in relation to the hierachical model shot type groups. However, it was a necessary step, since that way we could actually fit the binomial logit model to the data.

12.2 Model Choice

As mentioned above, our model accuracy suffered from the loss of information due to the compression of data. A solution to this could have been to use independent Bernoulli Logit models for each individual distance, where our Bernoulli likelihood would be logistically related to player's distance to the basket. That way, we could maximize model predicitive efficiency, since for each distance, the more observations there are, the better the posterior predictive power. However, with this approach we would completely use the predictive ability for continuous distances between our discrete values. In the long run, since we cannot predict probabilities for distances that have not yet been modeled, this approach is not very applicable either.

Another possible alternative would have been to use a linear model. From the figures 3 and **??** we can see that the model relationship does not look sigmoidal (s-shaped curve), which is a trademark of logistical curves. In fact, the models look more linear than they do logistical. This could suggest that our mathematical model choice was wrong. However, the introduction of the linear model means that we lose the asymptotic property of probabilities - since with linear models values can be above 1 and below 0, the linear model could predict values that are not applicable at all. This suggest that the linear model is not optimal either.

Overall, perhaps it is unlikely that shot likelihoods could even be univariately modeled. Afterall, there are so many variables that go into a single basketball shot - player position, location, player skill, defender distance and skill, tactics, player fatigue and confidence, among others - and being able to predict basketball shots based on distance alone might not be plausible at all. Basketball is a fast-moving and intricate sport, and every little factor one can think of, not only shot distance, contributes to the big picture that one witnesses from beyond the court. Maybe the game of basketball requires more intricate modeling.

13. Conclusion what was learned from the data analysis.

In this report, we studied the effect of shot distance on successful shot outcomes. The data set included 129915 shot attempts during 2010-18 NBA seasons. Shot distance, shot type and shot made flag was collected. Our analysis focused on pooled and hierarchical Binomial Logit models. Model convergence analysis was with traceplots, Both models were run with scale reduction statistics and effective sample size diagnostics. Posterior predictive checking was performed by plotting the posterior distribution as well as plotting elementary posterior predictive model fits. As a result from posterior predictive checking, we concluded that the models are capable of moderate posterior predictability. Model comparison was done with PSIS-LOO diagnostics and analyzing pareto k-values. We used previous results in conjunction with the PSIS-LOO diagnostics to show that the pooled model fits the data better. Second to last, we used sensitivity analysis to show that our models were robust to changes in the definition of our weakly informative priors. Finally, we discussed model predictive issues that arose from data loss due to compression as well poor model choice. Even though we discussed that out mathematical model choice might not have been optimal, we our exploration into other mathematical models brought us to the conclusion that it is more likely that basketball is such a complex sport that it is unlikely that any univariate mathematical model is capable of efficient and accurate posterior prediction.

14. Self-reflection of what the group learned while making the project.

Our knowledge of fitting Stan models to data improved during the project. We also learned new methods of how to analyze the goodness of the model fit. For example, posterior predictive checking was rather new to us. We also learned how to work as a team in data analysis project. Finally, we learned that Bayesian Data Analysis is HARD. Building a proper Bayesian model involves numerous stages, steps and techniques that are very complicated and laborsome.

15. Sources

Websites

- https://www.nbasavant.com/shot_search.php
- https://official.nba.com/rule-no-1-court-dimensions-equipment/
- https://mc-stan.org/rstan/reference/check_hmc_diagnostics.html

Books and articles

- Bayesian Data Analysis Third edition (Aki Vehtari et al., 2015)
- BDA course assigments (especially 8)

15. Appendices

Helper functions source code

```
convert_df = function(data, distances, shot_types) {
  shot_names = names(shot_types)
  n_shots = length(shot_names)
  rowcount = length(distances)*n_shots
  n_distances = length(distances)

  empty_nx3 = matrix(rep(NA, rowcount), ncol = n_shots)
  successes_df = data.frame(empty_nx3)
  colnames(successes_df) = shot_names
  throws_df = successes_df
  data_df = matrix(matrix(rep(NA,3*rowcount), ncol = 4))
  for (i in 1:n_shots) {
    data_shot_type = data[data$action_type %in%  shot_types[[i]],]
    for(j in 1:n_distances) {
      id = data_shot_type$shot_distance == distances[j]
      data_distances = data_shot_type[id,]
      throws_df[j,i] = nrow(data_distances)
      successes_df[j,i] = sum(data_distances$shot_made_flag)
    }
  }

  data_df = data.frame('distance' = distances, 'throws' = rowSums(throws_df), 'successes' =  rowSums(su
  return(list('df_pooled' = data_df, 'throw_data' = throws_df, 'success_data' = successes_df))
}


read_data = function(filename) {
  data = read.csv(filename)
  data = data[, c('action_type','shot_made_flag', 'shot_distance')]

  distances = sort(unique(data$shot_distance))
  shot_types = list('jump' = 'Jump Shot', 'fade' = c('Fadeaway Bank Shot','Fadeaway Jump Shot', 'Step Ba
```

```
  set.seed(420)
  id_test = sample(1:nrow(data), size=0.2*nrow(data))

  data_test = data[id_test,]
  data_train = data[-id_test,]

  dfs_train = convert_df(data_train,distances, shot_types)
  dfs_test = convert_df(data_test,distances, shot_types)
  return(list('train' = dfs_train,'test' = dfs_test))
}

posterior_pooled = function(params) {
  df_params_pool = data.frame(params)
  alpha = '\u03b1'
  beta = '\u03b2'

  w_a = (max(params$alpha) - min(params$alpha))/30
  w_b = (max(params$beta) - min(params$beta))/30

  p_alpha = ggplot(df_params_pool , aes(x=alpha))+
    geom_histogram(color="white", fill="darkred", binwidth = w_a)+
    xlab(alpha)
  p_beta = ggplot(df_params_pool, aes(x=beta)) + geom_histogram(color="white", fill="darkred", binwidth

  return(grid.arrange(p_alpha, p_beta, nrow = 1, top = 'Posterior of the pooled model'))
}

posterior_hier = function(params) {
  params = data.frame(params)

  a1 = ggplot(params, aes(x = alpha.1)) + geom_histogram(color="white", fill="darkred", binwidth = bin_
  b1 = ggplot(params, aes(x = beta.1)) + geom_histogram(color="white", fill="darkred", binwidth = bin_w

  a2 = ggplot(params, aes(x = alpha.2)) + geom_histogram(color="white", fill="darkred", binwidth = bin_
  b2 = ggplot(params, aes(x = beta.2)) + geom_histogram(color="white", fill="darkred", binwidth = bin_w

  a3 = ggplot(params, aes(x = alpha.3)) + geom_histogram(color="white", fill="darkred", binwidth = bin_
  b3 = ggplot(params, aes(x = beta.3)) + geom_histogram(color="white", fill="darkred", binwidth = bin_w

  return(grid.arrange(a1, b1, a2, b2, a3, b3, nrow = 3, top = 'Posterior distributions for the hierarch
}

bin_width = function(vector, nbins) {
  return((max(vector) - min(vector))/nbins)
}

plot_pooled = function(data, params) {
  data$ratio = data$successes/data$throws
  data$sd = sqrt(data$ratio*(1- data$ratio)/data$throws)

  a = mean(params$alpha)
  b = mean(params$beta)
  d_plot = seq(15,30,0.01)
```

```
  rat_l = a + b*d_plot
  rat = exp(rat_l)/(1+exp(rat_l))
  data_pool = data.frame('distance' = d_plot, 'ratio' = rat)

  base_plot = ggplot(data, aes(x = distance, y = ratio)) + geom_point()
  base_plot = base_plot + geom_errorbar(aes(ymin = ratio-sd, ymax = ratio+sd))
  base_plot = base_plot + geom_line(data = data_pool
                                    , aes(x = distance, y = ratio), color = 'darkred')
  plot_whole = base_plot + ggtitle('Fitted Pooled Model') + ylab('Probability') + xlab('Distance (feet)
  return(plot_whole)
}



plot_ppc = function(data, alpha, beta, title) {
  set.seed(1)
  n_samples = 1000
  id = sample(1:length(alpha), n_samples)

  alphas = alpha[id]
  betas = beta[id]

  alphas = append(alphas, mean(alpha))
  betas = append(betas, mean(beta))
  d_plot = seq(15,30,0.01)

  data$ratio = data$successes/data$throws
  base_plot = ggplot(data, aes(x = distance, y = ratio)) + ylab('Probability') + xlab('Distance from Ba

  for (i in 1:n_samples+1){
    rat_l = alphas[i] + betas[i]*d_plot
    rat = exp(rat_l)/(1+exp(rat_l))
    data_plot = data.frame('distance' = d_plot, 'ratio' = rat)
    if (i <= n_samples) {
      col = 'grey'
    }
    else {
      col = 'darkred'
    }
    base_plot = base_plot + geom_line(data = data_plot
                                      , aes(x = distance, y = ratio), color = col)
  }
  base_plot = base_plot + geom_point()

  return(base_plot)
}



plot_hie = function(data, params, shot_type){
  if(shot_type == 'jump'){
    data$ratio = data$success_data$jump/data$throw_data$jump
```

```
  data$sd = sqrt(data$ratio*(1- data$ratio)/data$throw_data$jump)

  a = mean(params$alpha)
  b = mean(params$beta)
  d_plot = seq(15,30,0.01)

  rat_l = a + b*d_plot
  rat = exp(rat_l)/(1+exp(rat_l))
  data_hie = data.frame('distance' = d_plot, 'ratio' = rat)
  data_hie2 = data.frame('distance' = data$df$distance, 'ratio' = data$ratio, 'sd' = data$sd)

  base_plot = ggplot(data_hie2, aes(x = distance, y = ratio)) + geom_point()
  base_plot = base_plot + geom_errorbar(aes(ymin = ratio-sd, ymax = ratio+sd))
  base_plot = base_plot + geom_line(data = data_hie
                                    , aes(x = distance, y = ratio), color = 'darkred')
  plot_whole = base_plot + ggtitle('Fitted Hierarchical Jump Shot Model') + ylab('Probability') + xlab
  return(plot_whole)
}
else if(shot_type == 'fade'){
  data$ratio = data$success_data$fade/data$throw_data$fade
  data$sd = sqrt(data$ratio*(1- data$ratio)/data$throw_data$fade)

  a = mean(params$alpha)
  b = mean(params$beta)
  d_plot = seq(15,30,0.01)

  rat_l = a + b*d_plot
  rat = exp(rat_l)/(1+exp(rat_l))
  data_hie = data.frame('distance' = d_plot, 'ratio' = rat)
  data_hie2 = data.frame('distance' = data$df$distance, 'ratio' = data$ratio, 'sd' = data$sd)

  base_plot = ggplot(data_hie2, aes(x = distance, y = ratio)) + geom_point()
  base_plot = base_plot + geom_errorbar(aes(ymin = ratio-sd, ymax = ratio+sd))
  base_plot = base_plot + geom_line(data = data_hie
                                    , aes(x = distance, y = ratio), color = 'darkred')
  plot_whole = base_plot + ggtitle('Fitted Hierarchical Fade Shot Model') + ylab('Probability') + xlab
  return(plot_whole)

}
else if(shot_type == 'run'){
  data$ratio = data$success_data$run/data$throw_data$run
  data$sd = sqrt(data$ratio*(1- data$ratio)/data$throw_data$run)

  a = mean(params$alpha)
  b = mean(params$beta)
  d_plot = seq(15,30,0.01)

  rat_l = a + b*d_plot
  rat = exp(rat_l)/(1+exp(rat_l))
  data_hie = data.frame('distance' = d_plot, 'ratio' = rat)
  data_hie2 = data.frame('distance' = data$df$distance, 'ratio' = data$ratio, 'sd' = data$sd)

  base_plot = ggplot(data_hie2, aes(x = distance, y = ratio)) + geom_point()
```

```
    base_plot = base_plot + geom_errorbar(aes(ymin = ratio-sd, ymax = ratio+sd))
    base_plot = base_plot + geom_line(data = data_hie
                                      , aes(x = distance, y = ratio), color = 'darkred')
    plot_whole = base_plot + ggtitle('Fitted Hierarchical Running Shot Model') + ylab('Probability') + x
    return(plot_whole)
  }

}


calc_probs = function(a,b,x) {
  ly = a + x*b
  y = exp(ly)/(1+exp(ly))
  return(y)
}
```