Cross Model Analysis

Ben Moolman, Craig Orman, Ethan Pross

Data Prep and Cleaning

```
# Load and clean data
original_tbl <- read.csv("./NBA-BoxScores-2023-2024.csv") |>
  mutate(
    START_POSITION = na_if(START_POSITION, "") |> factor(),
    COMMENT = na_if(COMMENT, "") |> factor(),
    MIN = na_if(MIN, ""),
    MIN = str_replace(MIN, "([0-9]+)\\.[0-9]+:", "\\1:")
  )

# Filter to starters only
starting_dat <- original_tbl |>
  filter(!is.na(START_POSITION))

# Calculate team points per game
team_points <- original_tbl |>
  filter(!is.na(PTS)) |>
  group_by(GAME_ID, TEAM_ID) |>
  summarize(TeamPoints = sum(PTS), .groups = "drop")

# Join with itself to get opponent points
team_vs_opponent <- team_points |>
  inner_join(team_points, by = "GAME_ID", suffix = c("", ".opp")) |>
  filter(TEAM_ID != TEAM_ID.opp) |>
  rename(OPP_TEAM_ID = TEAM_ID.opp, OpponentPoints = TeamPoints.opp)

# Compute average opponent points allowed per team (DRTG)
team_drtg <- team_vs_opponent |>
  group_by(TEAM_ID) |>
  summarize(DRTG_proxy = mean(OpponentPoints), n_games = n(), .groups = "drop")

# Build opponent_map from distinct team-game pairs
game_team_pairs <- original_tbl |>
  select(GAME_ID, TEAM_ID) |>
  distinct()

# Create mapping of TEAM_ID and OPP_TEAM_ID for each game
opponent_map <- game_team_pairs |>
  inner_join(game_team_pairs, by = "GAME_ID") |>
  filter(TEAM_ID.x != TEAM_ID.y) |>
  rename(TEAM_ID = TEAM_ID.x, OPP_TEAM_ID = TEAM_ID.y)
```

```r
# Join with defensive ratings (DRTG)
opponent_map <- opponent_map |>
  left_join(team_drtg |> rename(OPP_TEAM_ID = TEAM_ID, OPP_DRTG = DRTG_proxy), by = "OPP_TEAM_ID")


# Merge opponent info into starting dataset and center DRTG
mean_drtg <- mean(team_drtg$DRTG_proxy)

starting_dat <- starting_dat |>
  left_join(opponent_map, by = c("GAME_ID", "TEAM_ID")) |>
  mutate(centered_OPP_DRTG = OPP_DRTG - mean_drtg)

# For posterior predictive
# lebron 2544
# steph curry 201939
player_id = 2544
# GSW 1610612744
# LAL 1610612747

opp_team = 1610612744

player_dat = starting_dat[starting_dat$PLAYER_ID == player_id, ]
player_dat = player_dat[player_dat$OPP_TEAM_ID == opp_team, ]
n_iter = 100000
burnin = 10000
player_name = player_dat$PLAYER_NAME[1]
opp_team_name = starting_dat$TEAM_ABBREVIATION[starting_dat$TEAM_ID == opp_team][1]

if(length(player_dat$FGM) < 1) {
  print("ERROR, NO OBSERVATIONS")
}
```

Model functions

Model 1

```r
log_q = function(theta, y, n) {
  if (theta < 0 || theta > 1) return(-Inf)
  sum(dbinom(y, size = n, prob = theta, log = TRUE)) + dbeta(theta, 5, 5, log = TRUE)
}

MH_beta_binom = function(current = 0.5, prop_sd = 0.05, n_vec, y_vec, n_iter = 1000) {
  samps = rep(NA, n_iter)
  for (i in 1:n_iter) {
    proposed = rnorm(1, current, prop_sd)
    logr = log_q(proposed, y = y_vec, n = n_vec) - log_q(current, y = y_vec, n = n_vec)
    if (log(runif(1)) < logr) current = proposed
    samps[i] = current
  }
  return(samps)
}
```

```
MH_beta_grid_search = function(current = 0.5, n_vec, y_vec, n_iter = 1000) {
  vals <- seq(0.005, 1, by = 0.005)
  effect_sizes <- data.frame(sd = vals, ess = NA)

  for (i in seq_along(vals)) {
    samps <- MH_beta_binom(current = current, prop_sd = vals[i],
                           n_vec = n_vec, y_vec = y_vec, n_iter = 5000)
    effect_sizes$ess[i] <- effectiveSize(samps)
  }

  best_sd <- effect_sizes$sd[which.max(effect_sizes$ess)]
  final_samps <- MH_beta_binom(current = current, prop_sd = best_sd,
                               n_vec = n_vec, y_vec = y_vec, n_iter = n_iter)

  return(list(
    samples = final_samps,
    best_sd = best_sd,
    ess_table = effect_sizes
  ))
}


# Create Model
Y <- player_dat$FGM
N_vec <- player_dat$FGA

model_1 <- MH_beta_grid_search(current = 0.5,
                               y_vec = Y,
                               n_vec = N_vec,
                               n_iter = n_iter)

#Decided on using max for n
p_samples = model_1$samples
model_1_samps = rbinom(n_iter-burnin, max(N_vec), p_samples[burnin:n_iter])
```

Model 2

```
### MODEL 2 ###
log_n_con = function(p, lambda, n, y) {
  if (all(is.nan(log( ((1-p)*lambda)^sum(n) ) - sum(log((factorial(n-y)))) ))){
    return(rep(-Inf,length(y)))
  }else{
    log( ((1-p)*lambda)^sum(n) ) - sum(log((factorial(n-y))))
  }
}
mcmc_model_2 = function(data, player_id, opp_team_id, n_iter=5000,
                        init_lambda = c(), init_n = c(), gamma=0.01) {
  # Gather true data
  player_dat = data[data$PLAYER_ID == player_id, ]
  player_dat = player_dat[player_dat$OPP_TEAM_ID == opp_team_id, ]
  y = player_dat$FGM
  true_n = player_dat$FGA
  def_factor<- exp(gamma*(data$centered_OPP_DRTG[1]))
```

```
  if(length(init_lambda) ==0) {
    init_lambda = mean(player_dat$FGA)
  }
  if(length(init_n) ==0) {
    init_n = mean(player_dat$FGA)
  }


  big_N<- length(y)
  lambda<- init_lambda
  n<- rep(init_n,big_N)

  # setting up lists/matrices for returning
  p_matrix<- matrix(NA, nrow=n_iter, ncol=big_N)
  lambda_list<- rep(lambda, n_iter)
  n_matrix<- matrix(NA, nrow=n_iter, ncol=big_N)
  y_new_list <- rep(NA, n_iter)
  n_new_list <- rep(NA, n_iter)

  for (i in 1:n_iter) {
    # sample p
    p_unscaled<- rbeta(big_N, 5 + sum(y), 5 + sum(n-y))
    p<- p_unscaled*def_factor

    # sample lambda
    lambda<- rgamma(1,shape=sum(n)-1/2,rate=big_N)

    # sample n
    n_prop<- rnorm(big_N, n, 1) # the third 1 is a tuning parameter
    logr<- log_n_con(p, lambda, n_prop, y)-log_n_con(p, lambda, n, y)
    for (j in 1:length(logr)) {
      if (is.finite(logr[j]) && log(runif(1))<logr[j]) {
        n[j]<- n_prop[j]
      }
    }

    # generate new values
    n_new = rpois(1, lambda)
    y_new <- rbinom(1, size = n_new, prob = mean(p))

    # save values
    p_matrix[i,] = p
    n_matrix[i,] = n
    lambda_list[i] = lambda
    n_new_list[i] = n_new
    y_new_list[i] =  y_new
  }
  return(data.frame(iteration=1:n_iter,
                    parameter=rep(c(paste("n[",1:big_N,"]", sep=""), "lambda", "n_new", "y_new", paste(
                    value=c(as.numeric(n_matrix),lambda_list,n_new_list,y_new_list, as.numeric(p_matrix
}

model_2 = suppressWarnings(mcmc_model_2(data = starting_dat,
```

```
                              player_id = player_id, #LeBron
                              opp_team_id = opp_team, #GSW
                              n_iter=n_iter,
                              gamma=0.01)) #Previously found to be good value


model_2_samps <- model_2$value[which(model_2$parameter=="y_new")][(burnin+1):n_iter]
```

model 3

```
### MODEL 3 ###
log_n_con = function(n, r, theta, y,p) {
  dummy<- sum(log((factorial(n+r-1)))) - sum(log((factorial(n-y)))) + log((1-theta)*((1-p)^sum(n)))
  if (all(is.nan(dummy))){
    return(rep(-Inf,length(n)))
  }else{
    return(dummy)
  }
}
log_r_con = function(n, r, theta, y) {
  dummy<- sum(log((factorial(n+r-1)))) - sum(log((factorial(r-1)))) + log(theta^sum(r)) + log(r^(length
  if (all(is.nan(dummy))){
    return(rep(-Inf,length(r)))
  }else{
    return(dummy)
  }
}
mcmc_model_3 = function(data, player_id, opp_team_id, gamma = 0.01,
                        n_iter=5000, init_r, init_theta, init_n, init_p,
                        prop_r_sd = 3.5, prop_n_sd = 3.5) {
  # Gather true data
  player_dat = data[data$PLAYER_ID == player_id, ]
  player_dat = player_dat[player_dat$OPP_TEAM_ID == opp_team_id, ]
  y = player_dat$FGM
  true_n = player_dat$FGA
  def_factor<- exp(gamma*(data$centered_OPP_DRTG[1]))

  # Start code
  big_N<- length(y)
  r<- init_r
  theta<- init_theta
  n<- rep(init_n,big_N)
  p<- init_p

  # setting up lists/matrices for returning
  r_list<- rep(NA, n_iter)
  theta_list<- rep(NA, n_iter)
  p_matrix<- matrix(NA, nrow=n_iter, ncol=big_N)
  n_matrix<- matrix(NA, nrow=n_iter, ncol=big_N)

  for (i in 1:n_iter) {
    # sample p
    p_unscaled<- rbeta(big_N, 5 + sum(y), 5 + sum(true_n-y))
```

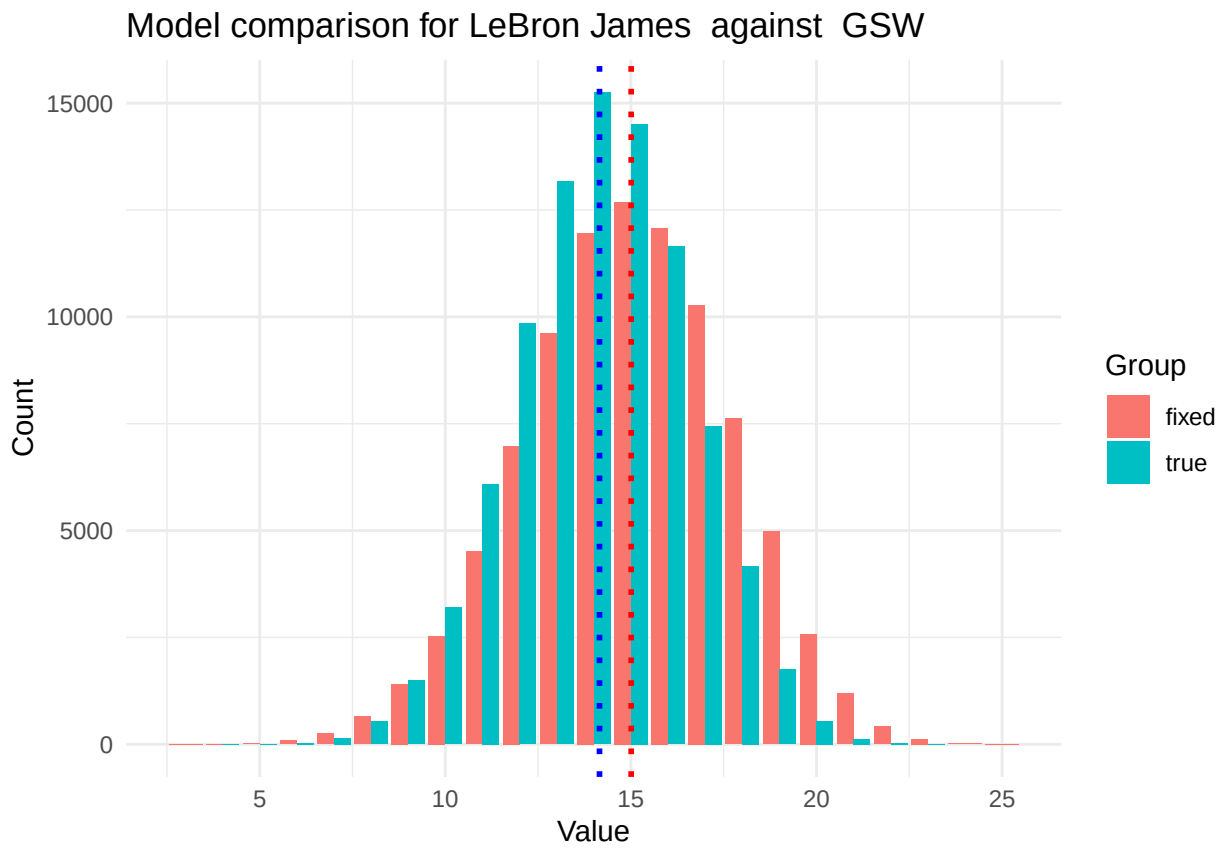
```
    p<- p_unscaled*def_factor

    # sample theta
    theta<- rbeta(1,sum(n)+1/2,r-1)

    # sample r
    r_prop<- rnorm(1, r, prop_r_sd) # the third 1 is a tuning parameter
    logr<- log_r_con(n, r_prop, theta, y)-log_r_con(n, r, theta, y)
    if (is.finite(logr)) {
      if (log(runif(1))<logr) {
        r<- r_prop
      }
    }

    # sample n
    n_prop<- rnorm(big_N, n, prop_n_sd) # the third 1 is a tuning parameter
    logr<- log_n_con(n_prop, r, theta, y,p)-log_n_con(n, r, theta, y,p)
    for (j in 1:length(logr)) {
      if (is.finite(logr[j])) {
        if (log(runif(1))<logr[j]) {
          n[j]<- n_prop[j]
        }
      }
    }

    # save values
    r_list[i] = r
    theta_list[i] = theta
    p_matrix[i,] = p
    n_matrix[i,] = n
  }
  # return(data.frame(iteration=1:n_iter,
  #                   r = r_list,
  #                   theta = theta_list,
  #                   p1 = p_matrix[,1],
  #                   p2 = p_matrix[,2],
  #                   p3 = p_matrix[,3],
  #                   n1 = n_matrix[,1],
  #                   n2 = n_matrix[,2],
  #                   n3 = n_matrix[,3]))
  return(data.frame(iteration=1:n_iter,
                    parameter=rep(c("r","theta",paste("p[",1:big_N,"]", sep=""),paste("n[",1:big_N,"]",
                    value=c(r_list,theta_list,as.numeric(p_matrix),as.numeric(n_matrix))))
}


init_r<- max(player_dat$FGA) # maybe tune?
init_theta<- mean(player_dat$FG_PCT) # maybe tune ?
init_p<- mean(player_dat$FG_PCT) # maybe tune ?
init_n<- round(mean(player_dat$FGA)) #maybe tune ?


model_3 =suppressWarnings(mcmc_model_3(starting_dat, player_id = player_id, opp_team_id = opp_team, gam
```

```
                            n_iter=n_iter, init_r,init_theta,init_n,init_p,
                            prop_r_sd=3.7, prop_n_sd = 3.7))

p1_culled<- model_3$value[which(model_3$parameter=="p[1]")][burnin:n_iter]
p2_culled<- model_3$value[which(model_3$parameter=="p[2]")][burnin:n_iter]
p3_culled<- model_3$value[which(model_3$parameter=="p[3]")][burnin:n_iter]
n1_culled<- model_3$value[which(model_3$parameter=="n[1]")][burnin:n_iter]
n2_culled<- model_3$value[which(model_3$parameter=="n[2]")][burnin:n_iter]
n3_culled<- model_3$value[which(model_3$parameter=="n[3]")][burnin:n_iter]

# getting the 10000 p's for sampling
posterior_mean_ps <- apply(data.frame(p1_culled,p2_culled,p3_culled),1,mean)

# getting the 10000 n's for sampling
posterior_mean_ns <- round(apply(data.frame(n1_culled,n2_culled,n3_culled),1,mean))

# making the y's using our posterior sample
model_3_samps <- rbinom(n_iter-burnin,posterior_mean_ns,posterior_mean_ps)
```

Creating our cool chart

```
true_samps = rbinom(n_iter-burnin,round(mean(player_dat$FGA)),mean(player_dat$FG_PCT))
df = data.frame(fixed = model_1_samps,
                true = true_samps)

df_long <- df |>
  pivot_longer(
    cols = everything(),
    names_to = "group",
    values_to = "value"
  )
title = paste("Model comparison for", player_name, " against ", opp_team_name)
df_long |>
  ggplot(aes(x = value, fill = group)) +
  geom_bar(position = "dodge") +
  labs(
    title = title,
    x = "Value",
    y = "Count",
    fill = "Group"
  ) +
  geom_vline(xintercept = mean(model_1_samps), color = "red", linetype = "dotted", size = 1) +
  geom_vline(xintercept = mean(true_samps), color = "blue", linetype = "dotted", size = 1) +
  theme_minimal()
```

```
## Warning: Using ‘size‘ aesthetic for lines was deprecated in ggplot2 3.4.0.
## i Please use ‘linewidth‘ instead.
## This warning is displayed once every 8 hours.
## Call ‘lifecycle::last_lifecycle_warnings()‘ to see where this warning was
## generated.
```


Model comparison for LeBron James against GSW
15000
10000
5000
0
Count
5
10
15
20
25
Value
Group
fixed
true