DATASCI451_Final_Project

2025-04-08

```r
library(rstan)
```

```
## Loading required package: StanHeaders

##
## rstan version 2.36.0.9000 (Stan version 2.36.0)

## 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)
## For within-chain threading using `reduce_sum()` or `map_rect()` Stan functions,
## change `threads_per_chain` option:
## rstan_options(threads_per_chain = 1)
```

```r
options(mc.cores = 4)
```

```r
olympics <- read.csv('01_data_preprocessing/data/olympics.csv')
head(olympics)
```

```
##   NOC Gold Silver Bronze Total Year X.Athletes X.Events IsHost Country
## 1 AUS    2      0      0     2    1          5        5      0       4
## 2 AUS    2      0      3     5    2          6        6      0       4
## 3 AUS    0      3      1     4    3          6        6      0       4
## 4 AUS    0      2      1     3    7         29       19      0       4
## 5 AUS    3      1      2     6    8         60       31      0       4
## 6 AUS    1      2      1     4    9         29       26      0       4
```

```r
# Held out 2024 Data
olympics_2024 <- olympics[olympics$Year == 33, ]
head(olympics_2024)
```

```
##     NOC Gold Silver Bronze Total Year X.Athletes X.Events IsHost Country
## 26  AUS   18     19     16    53   33        649      205      0       4
## 37  CHN   40     27     24    91   33        587      211      0       2
## 65  FRA   16     26     22    64   33        801      223      0       5
## 86  JPN   20     12     13    45   33        598      166      0       3
## 113 USA   40     44     42   126   33        854      234      0       1
```

```r
olympics <- olympics[olympics$Year != 33, ]

olympics$log_X.Athletes <- log(olympics$X.Athletes + 1)
olympics$log_X.Events <- log(olympics$X.Events + 1)
```

```r
# Correlation matrix of the variables
cor_matrix <- cor(olympics[, c("X.Athletes", "X.Events", "IsHost")])

library(corrplot)
```

```
## corrplot 0.95 loaded
```

```r
pdf("figures/correlation_matrix.pdf", width = 8, height = 6)

corrplot(
  cor_matrix,
  method = "color",
  tl.col = "black",
  tl.srt = 45,
  tl.cex = 0.9,
  tl.offset = 0.8,
  tl.pos = "lt",
  title = "Correlation Matrix of Predictive Variables",
  mar = c(1, 1, 2, 1),
  addgrid.col = "lightgray",
  col = colorRampPalette(c("skyblue", "white", "#004D99"))(100),
  cl.ratio = 0.2,
  cl.align.text = "c"
)
dev.off()
```

```
## pdf
##   2
```

```r
pdf("figures/histogram_medals.pdf", width = 8, height = 8)
par(mfrow = c(3, 1), mar = c(4, 4, 2, 1))
hist(olympics$Gold,
     col = "skyblue",
     border = "white",
     main = "Gold Medals",
     xlab = "Medals Counts",
     yaxt = "n",
     ylab = "")
text(x = max(olympics$Gold) * 0.738,
     y = max(hist(olympics$Gold, plot = FALSE)$counts * 0.8),
     labels = paste("Mean =", round(mean(olympics$Gold), 2),
                    "\nVar =", round(var(olympics$Gold), 2)),
     col = "red", cex = 1.2, adj = 0)

hist(olympics$Silver,
     col = "skyblue",
     border = "white",
```

```
     main = "Silver Medals",
     xlab = "Medals Counts",
     yaxt = "n",
     ylab = "")
text(x = max(olympics$Silver) * 0.7,
     y = max(hist(olympics$Silver, plot = FALSE)$counts * 0.8),
     labels = paste("Mean =", round(mean(olympics$Silver), 2),
                    "\nVar =", round(var(olympics$Silver), 2)),
     col = "red", cex = 1.2, adj = 0)

hist(olympics$Bronze,
     col = "skyblue",
     border = "white",
     main = "Bronze Medals",
     xlab = "Medals Counts",
     yaxt = "n",
     ylab = "")
text(x = max(olympics$Bronze) * 0.71,
     y = max(hist(olympics$Bronze, plot = FALSE)$counts * 0.8),
     labels = paste("Mean =", round(mean(olympics$Bronze), 2),
                    "\nVar =", round(var(olympics$Bronze), 2)),
     col = "red", cex = 1.2, adj = 0)
dev.off()
```

```
## pdf
##   2
```

$$
\begin{gathered}
\text{Total}_{c,y} \sim \text{NegBinomial}(\lambda_{c,y}, \phi) \\
\log(\lambda_{c,y}) = \alpha_c + \beta_0 + \beta_1 \text{X.Athletes}_{c,y} + \beta_2 \text{X.Events}_{c,y} + \beta_3 \text{IsHost}_{c,y} \\
\alpha_c \sim N(0, 5) \\
\beta_i \sim N(0, 5) \text{ for } i = 0, 1, 2, 3, 4 \\
\phi \sim \text{Gamma}(2, 0.1)
\end{gathered}
$$

```
stan_model_code <- "
data {
  int<lower=0> N; // number of observations
  int<lower=0> C; // number of countries
  int<lower=0> Y; // number of years
  array[C, Y] int<lower=0> Total; // total medals
  array[C, Y] real<lower=0> N_athletes; // number of athletes
  array[C, Y] real<lower=0> N_events; // number of events
  array[C, Y] int<lower=0, upper=1> IsHost; // host country
}
parameters {
  array[C] real alpha; // country-specific intercepts
  array[4] real beta; // coefficients for predictors
  real<lower=0> phi; // dispersion parameter
}
model {
  for (c in 1:C) {
```

```
    for (y in 1:Y) {
      Total[c, y] ~ neg_binomial_2_log(alpha[c] + beta[1] + beta[2] * N_athletes[c, y] + beta[3] * N_eve
    }
  }

  // Priors
  alpha ~ normal(0, 5);
  phi ~ gamma(2, 0.1);
  for (i in 1:4) {
    beta[i] ~ normal(0, 5);
  }
}

"
```

```
N = nrow(olympics)
C = length(unique(olympics$NOC))
Y = max(olympics$Year)
Gold = matrix(0, nrow = C, ncol = Y)
Silver = matrix(0, nrow = C, ncol = Y)
Bronze = matrix(0, nrow = C, ncol = Y)
N_athletes = matrix(0, nrow = C, ncol = Y)
N_events = matrix(0, nrow = C, ncol = Y)
IsHost = matrix(0, nrow = C, ncol = Y)
for (i in 1:C) {
  country_data = olympics[olympics$Country == i, ]
  for (j in 1:Y) {
    year_data = country_data[country_data$Year == j, ]
    if (nrow(year_data) > 0) {
      Gold[i, j] = year_data$Gold
      Silver[i, j] = year_data$Silver
      Bronze[i, j] = year_data$Bronze
      N_athletes[i, j] = year_data$log_X.Athletes
      N_events[i, j] = year_data$log_X.Events
      IsHost[i, j] = year_data$IsHost
    }
  }
}

gold_data_list = list(
  N = N,
  C = C,
  Y = Y,
  Total = Gold,
  N_athletes = N_athletes,
  N_events = N_events,
  IsHost = IsHost
)

silver_data_list = list(
  N = N,
  C = C,
  Y = Y,
```

```
  Total = Silver,
  N_athletes = N_athletes,
  N_events = N_events,
  IsHost = IsHost
)

bronze_data_list = list(
  N = N,
  C = C,
  Y = Y,
  Total = Bronze,
  N_athletes = N_athletes,
  N_events = N_events,
  IsHost = IsHost
)
```

```
model = stan_model(model_code = stan_model_code)

gold_fit <- sampling(object = model,
            data = gold_data_list,
            iter = 10000,
            warmup = 2000,
            seed = 451)
```

```
silver_fit <- sampling(object = model,
            data = silver_data_list,
            iter = 10000,
            warmup = 2000,
            seed = 451)
```

```
bronze_fit <- sampling(object = model,
            data = bronze_data_list,
            iter = 10000,
            warmup = 2000,
            seed = 451)
```

```
## Warning: There were 15 transitions after warmup that exceeded the maximum treedepth. Increase max_tre
## https://mc-stan.org/misc/warnings.html#maximum-treedepth-exceeded
```

```
## Warning: Examine the pairs() plot to diagnose sampling problems
```

```
# Analysis of the Gold Model
pdf("figures/gold_model.pdf", width = 8, height = 6)
plot(gold_fit)
```

```
## ci_level: 0.8 (80% intervals)
```

```
## outer_level: 0.95 (95% intervals)
```

```r
dev.off()
```

```
## pdf
##   2
```

```r
pdf("figures/gold_model_trace.pdf", width = 8, height = 6)
stan_trace(gold_fit, nrow = 3, ncol = 4)
dev.off()
```

```
## pdf
##   2
```

```r
pdf("figures/gold_model_hist.pdf", width = 8, height = 6)
stan_hist(gold_fit)
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

```r
dev.off()
```

```
## pdf
##   2
```

```r
summary(gold_fit)$summary
```

```
##                    mean      se_mean        sd          2.5%          25%
## alpha[1]  4.417132e-01 0.0266474722 2.0312162   -3.57129046   -0.8998364
## alpha[2]  1.116215e-01 0.0266001053 2.0315221   -3.91026463   -1.2257557
## alpha[3] -7.621047e-01 0.0266344804 2.0320469   -4.77660529   -2.1036917
## alpha[4] -8.649524e-01 0.0266275307 2.0309427   -4.87306367   -2.2038460
## alpha[5] -8.827668e-01 0.0266258715 2.0316342   -4.89068923   -2.2197583
## beta[1]  -1.980985e+00 0.0266578386 2.0449556   -5.99327607   -3.3325836
## beta[2]   8.240061e-01 0.0012116423 0.1352333    0.55903164    0.7330318
## beta[3]  -8.298702e-03 0.0013336193 0.1508828   -0.30494791   -0.1082781
## beta[4]   3.058219e-01 0.0009001547 0.1190484    0.07363778    0.2261483
## phi       1.605423e+01 0.0410980947 5.4119446    8.38420072   12.2217337
## lp__     -3.230419e+02 0.0228407143 2.3153946 -328.42727773 -324.3654839
##                     50%          75%       97.5%     n_eff      Rhat
## alpha[1]  4.372026e-01   1.79005012   4.4180594  5810.323 1.0007792
## alpha[2]  1.135166e-01   1.45457046   4.1043523  5832.791 1.0007729
## alpha[3] -7.635213e-01   0.58738031   3.2325254  5820.751 1.0007968
## alpha[4] -8.676188e-01   0.48065806   3.1151518  5817.462 1.0008014
## alpha[5] -8.879349e-01   0.46731865   3.1045263  5822.150 1.0007737
## beta[1]  -1.974073e+00  -0.63873956   2.0843084  5884.613 1.0008530
## beta[2]   8.230308e-01   0.91297188   1.0927002 12457.138 1.0000221
## beta[3]  -7.745153e-03   0.09253793   0.2856607 12800.176 0.9999696
## beta[4]   3.046984e-01   0.38514276   0.5420399 17490.920 1.0000045
## phi       1.516191e+01  18.77998188  29.3372129 17340.567 1.0001519
## lp__     -3.227044e+02 -321.35206883 -319.5297042 10276.156 1.0003389
```

```r
# Analysis of the Silver Model
pdf("figures/silver_model.pdf", width = 8, height = 6)
plot(silver_fit)
```

```
## ci_level: 0.8 (80% intervals)
```

```
## outer_level: 0.95 (95% intervals)
```

```r
dev.off()
```

```
## pdf
##   2
```

```r
pdf("figures/silver_model_trace.pdf", width = 8, height = 6)
stan_trace(silver_fit, nrow = 3, ncol = 4)
dev.off()
```

```
## pdf
##   2
```

```r
pdf("figures/silver_model_hist.pdf", width = 8, height = 6)
stan_hist(silver_fit)
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

```r
dev.off()
```

```
## pdf
##   2
```

```r
summary(silver_fit)$summary
```

```
##                    mean      se_mean        sd         2.5%          25%
## alpha[1]     0.21643170 0.0265477448 2.0339330   -3.7396161   -1.15619307
## alpha[2]    -0.05082450 0.0265625437 2.0344788   -4.0076937   -1.42909782
## alpha[3]    -0.77738152 0.0265401934 2.0331373   -4.7373074   -2.15503057
## alpha[4]    -0.69951951 0.0265527486 2.0333571   -4.6644180   -2.08405573
## alpha[5]    -0.68447342 0.0265461917 2.0330691   -4.6338251   -2.06760524
## beta[1]     -2.13765888 0.0266056278 2.0478966   -6.1118706   -3.53425290
## beta[2]      0.96110434 0.0010885979 0.1216557    0.7195194    0.87930458
## beta[3]     -0.15187818 0.0011626379 0.1317816   -0.4101343   -0.23967128
## beta[4]      0.08663597 0.0009334605 0.1159357   -0.1422264    0.00856328
## phi         22.11422028 0.0638303422 8.1706493   10.8471363   16.39614398
## lp__      -309.46047705 0.0220797454 2.2720483 -314.7245400 -310.78180171
##                    50%          75%       97.5%    n_eff     Rhat
## alpha[1]     0.23322106   1.60639516   4.1613699  5869.729 1.000834
## alpha[2]    -0.04062904   1.34324866   3.9026606  5866.337 1.000851
## alpha[3]    -0.76595959   0.61388289   3.1565584  5868.475 1.000839
## alpha[4]    -0.68432264   0.69456464   3.2519117  5864.194 1.000847
```

```
## alpha[5]   -0.67482682    0.70536400    3.2462458  5865.430 1.000832
## beta[1]    -2.13303610   -0.75415297    1.8537792  5924.736 1.000816
## beta[2]     0.96118297    1.04337083    1.2008926 12489.057 1.000184
## beta[3]    -0.15312491   -0.06383704    0.1101934 12847.558 1.000202
## beta[4]     0.08628104    0.16433763    0.3160563 15425.615 1.000023
## phi        20.60066478   26.08369107   42.1050653 16385.466 1.000277
## lp__     -309.13105838 -307.79725514 -305.9970707 10588.807 1.000101
```

```
# Analysis of the Bronze Model
pdf("figures/bronze_model.pdf", width = 8, height = 6)
plot(bronze_fit)
```

```
## ci_level: 0.8 (80% intervals)
```

```
## outer_level: 0.95 (95% intervals)
```

```
dev.off()
```

```
## pdf
##   2
```

```
pdf("figures/bronze_model_trace.pdf", width = 8, height = 6)
stan_trace(bronze_fit, nrow = 3, ncol = 4)
dev.off()
```

```
## pdf
##   2
```

```
pdf("figures/bronze_model_hist.pdf", width = 8, height = 6)
stan_hist(bronze_fit)
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

```
dev.off()
```

```
## pdf
##   2
```

```
summary(bronze_fit)$summary
```

```
##                   mean     se_mean        sd       2.5%         25%
## alpha[1]   -0.11396049 0.024260212 2.0281412 -4.1424320  -1.4764136
## alpha[2]   -0.35886712 0.024253484 2.0266685 -4.3931789  -1.7182528
## alpha[3]   -0.71351502 0.024253128 2.0275966 -4.7357349  -2.0766754
## alpha[4]   -0.63142871 0.024280330 2.0273180 -4.6739371  -1.9877145
## alpha[5]   -0.73454896 0.024247001 2.0272016 -4.7627661  -2.0933773
## beta[1]    -2.80163711 0.024388792 2.0501625 -6.7741859  -4.2086843
## beta[2]     1.11460335 0.001128093 0.1208378  0.8731453   1.0345871
## beta[3]    -0.16833498 0.001112039 0.1226876 -0.4058331  -0.2501244
```

```
## beta[4]    -0.05020952 0.000866808  0.1069868   -0.2590715   -0.1214385
## phi         31.58707975 0.089657859 11.8595755   15.0917085   23.1844099
## lp__      -293.73243443 0.022187818  2.2731499 -299.0101911 -295.0329132
##                     50%           75%         97.5%     n_eff      Rhat
## alpha[1]    -0.09918133    1.27306728    3.81435511  6988.874 1.000133
## alpha[2]    -0.34375133    1.02834819    3.56113180  6982.600 1.000141
## alpha[3]    -0.69511653    0.67904154    3.21682386  6989.202 1.000125
## alpha[4]    -0.61319786    0.75495317    3.29898205  6971.634 1.000124
## alpha[5]    -0.71887904    0.65159498    3.17628881  6990.011 1.000133
## beta[1]     -2.80895867   -1.42820986    1.27287850  7066.364 1.000115
## beta[2]      1.11459348    1.19578772    1.34993528 11474.022 1.000410
## beta[3]     -0.17020966   -0.08708980    0.07679603 12171.975 1.000279
## beta[4]     -0.05084803    0.01993483    0.16182676 15234.037 1.000020
## phi         29.44861237   37.57043973   60.56322056 17496.918 1.000139
## lp__      -293.38217395 -292.07069309 -290.31235813 10496.076 1.000378
```

```r
# Predict the data for 2028 using SLR
# Not used
pred_data <- function(data, Country, IsHost) {
  noc_data = data[data$Country == Country, ]
  lm.athletes = lm(X.Athletes ~ Year, data = noc_data)
  summary(lm.athletes)
  lm.events = lm(X.Events ~ Year, data = noc_data)
  summary(lm.events)
  pred = data.frame(
    Year = 34,
    log_X.Athletes = log(predict(lm.athletes, newdata = data.frame(Year = 34))),
    log_X.Events = log(predict(lm.events, newdata = data.frame(Year = 34))),
    IsHost = IsHost,
    Country = Country
  )
  return(pred)
}
```

```r
# Predict medals for countries in 2024
post_pred <- function(fit, new_data) {
  # Extract posterior samples
  posterior_samples <- extract(fit)
  alpha_samples <- posterior_samples$alpha[, 1]    # vector of MCMC samples for alpha[c]
  beta_samples  <- posterior_samples$beta          # matrix: iterations × 4 (or list of vectors)
  phi_samples   <- posterior_samples$phi     # vector of MCMC samples for phi

  # Calculate log(lambda)
  log_lambda <- alpha_samples +
                beta_samples[,1] +
                beta_samples[,2] * new_data$log_X.Athletes +
                beta_samples[,3] * new_data$log_X.Events +
                beta_samples[,4] * new_data$IsHost

  # Calculate lambda
  lambda_pred <- exp(log_lambda)

  # Generate predictions from Poisson distribution
  set.seed(451)  # for reproducibility
```

```r
  total_pred <- rnbinom(n = length(lambda_pred),
                        size = phi_samples,
                        mu = lambda_pred)

  return(total_pred)
}
```

```r
# Predict the medals for 2028
predict_for_country <- function(
  gold_fit,
  silver_fit,
  bronze_fit,
  olympics_data,
  country_id,
  is_host,
  country_name = "COUNTRY",
  figure_root_dir = "figures"
) {

  # 1. Predict the data for 2028
  new_data <- data.frame(
    Year = 33,
    log_X.Athletes = log(olympics_data$X.Athletes[olympics_data$Country == country_id]+1),
    log_X.Events = log(olympics_data$X.Events[olympics_data$Country == country_id]+1),
    IsHost = is_host,
    Country = country_name
  )

  # 2. Predict the medals using the posterior predictive distribution
  gold_pred   <- post_pred(gold_fit, new_data)
  silver_pred <- post_pred(silver_fit, new_data)
  bronze_pred <- post_pred(bronze_fit, new_data)

  # Create the directory for saving figures
  dir.create(file.path(figure_root_dir, country_name), showWarnings = FALSE, recursive = TRUE)

  # 3. Plot the posterior predictive distribution
  save_hist <- function(pred_values, medal_type = "Gold") {
    pdf_file <- file.path(figure_root_dir, country_name, paste0(tolower(medal_type), "_predictions.pdf"
    pdf(pdf_file, width = 8, height = 6)

    plotPost(pred_values, xlab = paste(medal_type, "Medals"),
             main = paste("Predicted", medal_type, "Medals for", country_name, "in 2024"),
             )

    dev.off()
  }

  # 4. Plot for each medal type
  save_hist(gold_pred,   medal_type = "Gold")
  save_hist(silver_pred, medal_type = "Silver")
  save_hist(bronze_pred, medal_type = "Bronze")
```

```r
  # 5. Return the summary
  out_list <- list(
    country = country_name,
    gold_pred = gold_pred,
    silver_pred = silver_pred,
    bronze_pred = bronze_pred
  )

  return(invisible(out_list))
}
```

```r
# Predict for USA
library(bayesboot)
usa_pred <- predict_for_country(
  gold_fit,
  silver_fit,
  bronze_fit,
  olympics_2024,
  country_id = 1,
  is_host = 0,
  country_name = "USA",
  figure_root_dir = "figures"
)

# Predict for CHN
chn_pred <- predict_for_country(
  gold_fit,
  silver_fit,
  bronze_fit,
  olympics_2024,
  country_id = 2,
  is_host = 0,
  country_name = "CHN",
  figure_root_dir = "figures"
)

# Predict for JPN
gbr_pred <- predict_for_country(
  gold_fit,
  silver_fit,
  bronze_fit,
  olympics_2024,
  country_id = 3,
  is_host = 0,
  country_name = "JPN",
  figure_root_dir = "figures"
)

# Predict for AUS
aus_pred <- predict_for_country(
  gold_fit,
  silver_fit,
  bronze_fit,
```

```
  olympics_2024,
  country_id = 4,
  is_host = 0,
  country_name = "AUS",
  figure_root_dir = "figures"
)

# Predict for FRA
fra_pred <- predict_for_country(
  gold_fit,
  silver_fit,
  bronze_fit,
  olympics_2024,
  country_id = 5,
  is_host = 1,
  country_name = "FRA",
  figure_root_dir = "figures"
)
```