The Bradley-Terry Model of Ranking via Paired Comparisons

Bob Carpenter

19 March 2018

Contents

Abstract

Bradley and Terry (1952) modeled paired comparison data, such as comparisons of pairs of products by consumers or outcomes of sporting contests between two players. The data is binary, with each paired comparison having a single "winner." The case study will provide simulated data and demonstrate the inference of contestant abilities and ranks as well as predicting the outcomes of future contests.

We will show how the Bradley-Terry likelihood may be paired with a prior to produce a Bayesian model for which it is possible to perform inference on parameters from observed data using full Bayesian inference (the so-called "Bayesian inversion"). This then allows inference for future matches.

After introducing a Bayesian Bradley-Terry model, we replace the simple fixed population prior with a hierarchical model in which we jointly estimate the population variation in ability along with the ability parameters. This allows tighter estimation of ability parameters and reduces uncertainty in future, unobserved outcome predictions.

Finally, we will introduce a Bradley-Terry model for team contests, where each team consists of multiple players. The assumption will be that each player has an ability and the team's ability is additive on the log odds scale.

The Bradley-Terry Model

Like all good probability stories, this one is a generative story. Nevertheless, we will begin historically with the likelihood introduced by Bradley

and Terry (1952) and earlier discussed by Zermelo (1929). The observed data is the outcome of matches between players and the (latent) parameters are the player abilities; we postpone from now the model of the abilities and concentrate on the model of match outcomes.

We will suppose that there are K players. Each contestant will have an ability $\alpha_k \in \mathbb{R}$. The probability that contestant i will beat contestant j is given by[1]

$$\Pr[i \text{ beats } j] = \text{logit}^{-1}(\alpha_i - \alpha_j).$$

Furthermore, the flexibility of the long-form data means not every pair of players need play each other and there may be multiple matches between the same two players.

play each other. If there are multiple matches between the same two players, the results are assumed to be independent.

[1] The log odds function

$$\text{logit} : (0,1) \to (-\infty, \infty)$$

is defined by

$$\text{logit}(u) = \log\left(\frac{u}{1-u}\right).$$

Its inverse,

$$\text{logit}^{-1} : (-\infty, \infty) \to (0,1),$$

is given by

$$\text{logit}^{-1}(v) = \frac{1}{1+\exp(-v)} = \frac{\exp(v)}{1+\exp(v)}.$$

Long form data

Because the data is "incomplete" (i.e., not all players need play each other), the easiest way to encode the data is in long form, e.g.,

n	player^0	player^1	y
1	1	2	1
2	1	13	0
3	1	5	0
4	2	9	0
5	3	4	1
6	2	9	1

The first column, labeled n is the match index. With N matches, $n \in 1, 2, \ldots, N$. The second two columns indicate which players participated in the match. The last column is the result $y_n \in \{0, 1\}$, indicating which player won the match. For example, the fourth row ($n = 4$) records a match between player 2 (player$^0 = 2$) and player 9 (player$^1 = 9$) in which player 2 won ($y = 0$).

Simulating data

In order to simulate data, we need to assume a distribution over the ability parameters. For simplicity and because we are not assuming anything about the structure of the data, we will assume they are distributed standard normal (mean zero, standard deviation one).[2]

[2] More elaborate models would be appropriate, if, for example, we knew that the population was made up out of two groups, amateurs and professionals, a mixture prior would be more appropriate.

```
# map (-infinity, infinty) -> (0, 1)
inv_logit <- function(u) 1 / (1 + exp(-u))

# map vector to vector with sum of zero
```

```
center <- function(u) u - sum(u) / length(u)

# parameters
K <- 50
alpha <- center(rnorm(K))

# observations
N <- K^2    # may not have this many matches in practice
player1 <- rep(NA, N)
player0 <- rep(NA, N)
for (n in 1:N) {
  players <- sample(1:K, 2)
  player0[n] <- players[1]
  player1[n] <- players[2]
}
log_odds_player1 <- alpha[player1] - alpha[player0]
prob_win_player1 <- inv_logit(log_odds_player1)
y <- rbinom(N, 1, prob_win_player1)
```

Rendering the first few rows of the data frame shows that it is in the same form as the long-form table shown in the previous section.

```
df <- data.frame(player0 = player0, player1 = player1, y = y)
head(df)
```

```
  player0 player1 y
1      23      49 1
2      39      16 1
3      37      14 0
4      15      47 0
5      31      14 0
6      30      38 1
```

Coding the Bradley-Terry model for maximum likelihood

Suppose have some data data in the appropriate long form. With RStan, we can write the model out directly in terms of its log likelihood. The first part of the program encodes the data, beginning with the constant number of players `K` and matches `N`. Then there are three parallel arrays of integers indicating the players in each match and the outcome.

```
data {
  int<lower = 0> K;                    // players
  int<lower = 0> N;                    // matches
  int<lower=1, upper = K> player1[N];   // player 1 for game n
```

```
  int<lower=1, upper = K> player0[N];   // player 0 for game n
  int<lower = 0, upper = 1> y[N];       // winner for match n
}
parameters {
  vector[K] alpha;                      // ability for player n
}
model {
  y ~ bernoulli_logit(alpha[player1] - alpha[player0]);
}
```

The parameters are coded as a K-vector, and the likelihood coded as defined above. The vectorized sampling statement is equivalent to the following less efficient loop form.[3]

```
for (n in 1:N)
  y[n] ~ bernoulli_logit(alpha[player1[n]] - alpha[player2[n]]);
```

[3] The loop form illustrates how the likelihood function is defined as

$$p(y \mid \alpha) = \prod_{n=1}^{N} \text{Bernoulli}\Big(y_n \;\Big|\; \text{logit}^{-1}\big(\alpha_{\text{player1}[n]} - \alpha_{\text{player0}[n]}$$

The distribution `bernoulli_logit` is the Bernoulli distribution with a parameter on the logit (log odds) scale, where for $y \in \{0, 1\}$ and $\theta \in (0, 1)$,

$$\text{Bernoulli}(y \mid \theta) \;=\; \begin{cases} \theta & \text{if } \; y = 1 \\ 1 - \theta & \text{if } \; y = 0. \end{cases}$$

and for $\alpha \in (-\infty, \infty)$,

$$\text{BernolliLogit}(y \mid \alpha) \;=\; \text{Bernoulli}(y \mid \text{logit}^{-1}(\alpha)).$$

Building in the link function makes the arithmetic more stable and the code less error prone. Thus rather than writing

```
y ~ bernoulli(inv_logit(u));
```

we encourage users to use the logit (log odds) parameterized version.

```
y ~ bernoulli_logit(u);
```

Maximum likelihood estimation

All that we need to do to fit the data with Stan is pack the data into a list, compile the model using `stan_model`, then find the maximum likelihood estimate θ^*, that is, the estimate for the parameter values that maximizes the probability of the match outcomes that were observed.

$$\begin{aligned} \theta^* &= \text{argmax}_\theta \, p(y|\theta) \\ &= \text{argmax}_\theta \, \textstyle\prod_{n=1}^{N} \text{Bernoulli}(y_n \mid \text{logit}^{-1}(\alpha_{\text{player1[n]}} - \alpha_{\text{player0[n]}}) \end{aligned}$$

```
mle_model_data <-
  list(K = K, N = N, player0 = player0, player1 = player1, y = y)
mle_model <-
  stan_model("individual-uniform.stan")
mle_model_estimates <-
  optimizing(mle_model, data = mle_model_data)
```

We can now see how well we recovered the parameters by plotting the maximum likelihood estimates against the true values, which takes some wrangling to wrench out of the fit object.

```
alpha_star <- mle_model_estimates$par[paste("alpha[", 1:K, "]", sep="")]
mle_fit_plot <-
  ggplot(data.frame(alpha = alpha, alpha_star = alpha_star),
         aes(x = alpha, y = alpha_star)) +
      geom_abline(slope = 1, intercept = 0, color="green", size =
      geom_point(size = 2) +
      ggtheme_tufte()
mle_fit_plot
```

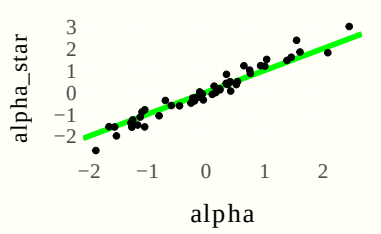

Figure 1: Fit of true value (horizontal axis) versus maximum likelihood estimate (vertical axis). The nearly linear relationship shows that maximum likelihood estimation does a good job estimating parameters with this number and scale of parameters and number of matches.

Ranking with the Bradley-Terry Model

Players (or other items) may be ranked using the Bradley-Terry model. In Stan, functions of parameters, such as rankings, may be coded in the generated quantities. So we only need to add the following lines to the end of our original program to define the ranking.

```
generated quantities {
  int<lower=1, upper=K> ranked[K] = sort_indices_desc(alpha);
}
```

This returns the sorting of the indexes in descending order.

```
ranked_players <-
  mle_model_estimates$par[paste("ranked[", 1:K, "]",
                                sep="")]
print(ranked_players, digits=0)
```

```
 ranked[1]  ranked[2]  ranked[3]  ranked[4]  ranked[5]  ranked[6]  ranked[7]
        16         30         49         18         48         26         32
 ranked[8]  ranked[9] ranked[10] ranked[11] ranked[12] ranked[13] ranked[14]
        22         29         40          7         43          5         20
ranked[15] ranked[16] ranked[17] ranked[18] ranked[19] ranked[20] ranked[21]
        13         23         11         31          3         50         38
```

```
ranked[22] ranked[23] ranked[24] ranked[25] ranked[26] ranked[27] ranked[28]
        17         46         28          9         41         42         10
ranked[29] ranked[30] ranked[31] ranked[32] ranked[33] ranked[34] ranked[35]
         2         34         44         37         21          6          4
ranked[36] ranked[37] ranked[38] ranked[39] ranked[40] ranked[41] ranked[42]
        25         33         14         12         19         35         15
ranked[43] ranked[44] ranked[45] ranked[46] ranked[47] ranked[48] ranked[49]
         1         36         27         47         39         45         24
ranked[50]
         8
```

The value of `ranked[1]` is the index of the top-ranked player, `ranked[2]` of the second-ranked player, and so on. We can print the rankings we derive as follows.

Bayesian Bradley-Terry Model

To convert our simple likelihood into a proper Bayesian model, we need a prior for the ability parameters. Such a prior will characterize the population of players in terms of the distribution of their abilities.

Population model for abilities

We follow Leonard (1977) in laying out simple non-conjugate priors for the ability parameters.[4]

$$\alpha_k \sim \text{Normal}(0, 1)$$

[4] This is in some sense cheating because it guarantees the model is well specified in the sense of exactly matching the data-generating process. In reality, models are almost always approximations and thus not perfectly well specified for the data sets they model.

Coding the Bayesian Bradley-Terry model

The form of the data does not change. The declarations of parameters is simplified and no longer centered by construction.

```
parameters {
  vector[K] alpha;                     // ability for player n
}
model {
  alpha ~ normal(0, 1);
  y ~ bernoulli_logit(alpha[player1] - alpha[player0]);
}
```

Instead of hard centering, the normal prior with location parameter zero will implicitly center the paremters around zero by assigning them higher density. The unit scale of the normal prior provides an indication of how much variation there is in player ability.

Fitting the model

```
individual_model <- stan_model("individual.stan")
individual_posterior <- sampling(individual_model, data = mle_model_data)
print(individual_posterior, "alpha", probs=c(0.05, 0.5, 0.95))
```

```
Inference for Stan model: individual.
4 chains, each with iter=2000; warmup=1000; thin=1;
post-warmup draws per chain=1000, total post-warmup draws=4000.

           mean se_mean   sd    5%   50%   95% n_eff Rhat
alpha[1]  -1.29    0.01 0.27 -1.75 -1.29 -0.86   960    1
alpha[2]  -0.21    0.01 0.28 -0.68 -0.20  0.25  1088    1
alpha[3]   0.32    0.01 0.27 -0.12  0.32  0.77   972    1
alpha[4]  -0.44    0.01 0.28 -0.90 -0.43  0.01  1049    1
alpha[5]   0.77    0.01 0.25  0.36  0.77  1.19   783    1
alpha[6]  -0.36    0.01 0.26 -0.79 -0.35  0.07   911    1
alpha[7]   0.94    0.01 0.28  0.50  0.94  1.41  1045    1
alpha[8]  -2.36    0.01 0.33 -2.92 -2.35 -1.83  1289    1
alpha[9]  -0.03    0.01 0.26 -0.45 -0.02  0.40  1012    1
alpha[10] -0.20    0.01 0.24 -0.60 -0.20  0.20   845    1
alpha[11]  0.37    0.01 0.26 -0.04  0.37  0.80   940    1
alpha[12] -0.83    0.01 0.28 -1.29 -0.82 -0.37   822    1
alpha[13]  0.46    0.01 0.27  0.02  0.46  0.89   918    1
alpha[14] -0.74    0.01 0.26 -1.19 -0.73 -0.33   801    1
alpha[15] -1.16    0.01 0.26 -1.61 -1.16 -0.73   876    1
alpha[16]  2.65    0.01 0.36  2.08  2.64  3.27  1502    1
alpha[17]  0.12    0.01 0.26 -0.31  0.12  0.54   875    1
alpha[18]  1.67    0.01 0.31  1.17  1.66  2.20  1311    1
alpha[19] -0.98    0.01 0.28 -1.43 -0.98 -0.52   951    1
alpha[20]  0.46    0.01 0.27  0.00  0.46  0.91   973    1
alpha[21] -0.34    0.01 0.27 -0.79 -0.34  0.10   971    1
alpha[22]  1.13    0.01 0.27  0.69  1.13  1.57  1027    1
alpha[23]  0.39    0.01 0.25 -0.02  0.39  0.79   793    1
alpha[24] -1.80    0.01 0.31 -2.31 -1.80 -1.30  1039    1
alpha[25] -0.57    0.01 0.26 -1.00 -0.57 -0.15   863    1
alpha[26]  1.39    0.01 0.31  0.89  1.39  1.88  1098    1
alpha[27] -1.46    0.01 0.27 -1.91 -1.45 -1.03   898    1
alpha[28]  0.03    0.01 0.26 -0.40  0.03  0.46   824    1
alpha[29]  1.12    0.01 0.28  0.67  1.12  1.59   959    1
alpha[30]  2.16    0.01 0.33  1.62  2.15  2.73  1501    1
alpha[31]  0.35    0.01 0.26 -0.09  0.34  0.78   952    1
alpha[32]  1.36    0.01 0.27  0.92  1.36  1.81   860    1
alpha[33] -0.57    0.01 0.28 -1.03 -0.57 -0.12   954    1
alpha[34] -0.24    0.01 0.24 -0.65 -0.24  0.15   851    1
```

```
alpha[35] -1.04    0.01 0.26 -1.49 -1.04 -0.61   949    1
alpha[36] -1.38    0.01 0.27 -1.83 -1.38 -0.94   911    1
alpha[37] -0.33    0.01 0.27 -0.77 -0.33  0.10   932    1
alpha[38]  0.16    0.01 0.25 -0.24  0.16  0.58   821    1
alpha[39] -1.45    0.01 0.27 -1.88 -1.45 -1.00  1030    1
alpha[40]  1.11    0.01 0.26  0.68  1.10  1.54   944    1
alpha[41] -0.07    0.01 0.27 -0.51 -0.07  0.38   964    1
alpha[42] -0.08    0.01 0.26 -0.51 -0.08  0.36   950    1
alpha[43]  0.80    0.01 0.25  0.39  0.80  1.21   876    1
alpha[44] -0.29    0.01 0.26 -0.72 -0.28  0.14   983    1
alpha[45] -1.47    0.01 0.27 -1.90 -1.47 -1.04   960    1
alpha[46]  0.06    0.01 0.25 -0.35  0.06  0.46   851    1
alpha[47] -1.45    0.01 0.27 -1.90 -1.45 -1.01   975    1
alpha[48]  1.47    0.01 0.30  0.99  1.47  1.97   916    1
alpha[49]  1.69    0.01 0.29  1.21  1.69  2.17   867    1
alpha[50]  0.27    0.01 0.28 -0.18  0.26  0.73   966    1

Samples were drawn using NUTS(diag_e) at Sat Nov 21 11:57:18 2020.
For each parameter, n_eff is a crude measure of effective sample size,
and Rhat is the potential scale reduction factor on split chains (at
convergence, Rhat=1).
```

For the posterior fit object, we are taking draws $\alpha^{(m)}$ from the posterior[5]

[5] Not indendently, but by using Markov chain Monte Carlo.

$$p(\alpha|y) \propto p(y|\alpha)\, p(\alpha).$$

To calculate Bayesian estimates, we take posterior means, which are guaranteed to minimize expected square error when the model is well specified.

$$\begin{aligned}
\hat{\alpha}_k &= \mathbb{E}\left[\alpha_k \mid y\right] \\
&= \int_{-\infty}^{\infty} \alpha_k \; p(\alpha_k|y) \; \mathrm{d}\alpha \\
&\approx \frac{1}{M} \sum_{m=1}^{M} \alpha_k^{(m)}
\end{aligned}$$

This is an example of full Bayesian inference, which is nearly always based on calculating conditional expectations of quantities of interest over the posterior. The second line defining the expectation shows the general form—a weighted average of the quantity of interest, α_k, over the posterior distribution $p(\alpha_k|y)$. This weighted average is calculated by Markov chain Monte Carlo (MCMC) using an average of the posterior draws.

The `extract()` function in RStan returns a structure from which the draws $\alpha^{(m)}$ may be extracted. These are easily converted into

posterior means.[6]

[6] They can be extracted even more easily using utility functions in RStan; this long form is just for pedagogical purposes.

```
alpha_hat <- rep(NA, K)
for (k in 1:K)
  alpha_hat[k] <- mean(extract(individual_posterior)$alpha[ , k])
```

These can then be scatterplotted against the true values just as the maximum likelihood was previously.

```
bayes_fit_plot <-
  ggplot(data.frame(alpha = alpha, alpha_hat = alpha_hat),
         aes(x = alpha, y = alpha_hat)) +
      geom_abline(slope = 1, intercept = 0, color="green", size =
      geom_point(size = 2) +
      ggtheme_tufte()
bayes_fit_plot
```

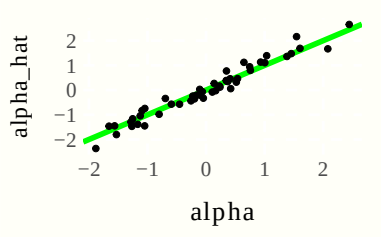

Figure 2: Fit of true value (horizontal axis) versus Bayesian (posterior mean) estimate (vertical axis). The nearly linear relationship shows that the Bayesian estimates also do a good job of fitting this data.

With this much data, the Bayesian estimates are very similar to the maximum likelihood estimates.

Ranking in the Bayesian model

Because of the way posterior quantities are calculated with uncertainty, it does not make sense to return a single answer for the ranking. Instead, we have a probabilistic notion of ranking which can be coded in Stan as follows.

```
generated quantities {
  int<lower=1, upper=K> ranking[K];       // rank of player ability
  {
    int ranked_index[K] = sort_indices_desc(alpha);
    for (k in 1:K)
      ranking[ranked_index[k]] = k;
  }
}
```

With this definition, `ranking[k]` holds the rank of player `k` (rather than the index of the player at rank `k` as before). This allows for full Bayesian inference for posterior uncertainty.

Intercepts and home-field advantage

With player abilities α_k centered around zero and the total predictor being additive in player abilities, this model has no intercept term. This is because there is symmetry between the player identified as player 0 and the player identified as player 1. If these identifiers are assigned randomly, the expected difference $\alpha_i - \alpha_j$ is zero.

In situations where there is a distinction between the players assigned as player 0 and as player 1, it will be necessary to introduce an intercept term to model the advantage of being player 1 (which may be negative and thus actually be a disadvantage). This intercept could model the advantage for playing the white pieces in chess and moving first, or the home-field advantage in basketball games. For this case study, we stick to random assignment and assume there is no advantage to player 0 or player 1 in any match.[7]

[7] Technically, the prior and likelihood with random assignment ensure the prior predictive expectation of y_n is zero. Such an assumption is easily tested.

Hierarchical Bayesian Bradley-Terry Model

Rather than hard-coding the prior for abilities, we can use a hierarchical prior to estimate the amount of variation in the population at the same time as we estimate the parameters themselves.

Priors and Hyperprior

We introduce a new parameter $\sigma > 0$ for the scale of variation in the population. We then use the scale σ in the prior for the population parameters,

$$\alpha_k \sim \mathsf{Normal}(0, \sigma).$$

We then need a hyperprior on σ to complete the model; we will somewhat arbitrarily choose a lognormal prior with scale 0.5.

$$\sigma \sim \mathsf{LogNormal}(0,\, 0.5).$$

When there are large numbers of players, player ability estimates and the estimate of the scale of variation should not be very sensitive to the choice of prior scale (see the exercises).

If there is not much variation among the abilities of the players in the population, σ will be estimated with a value near zero; otherwise, if the abilities are estimated to be widely varying, σ will be large. In the limit as $\sigma \to 0$, the model reverts to a complete pooling model where every player will have the same ability value (zero). In the other limit, as $\sigma \to \infty$, the model reverts to a model with no pooling, where the amount of variation in the population means no information can be used from the population of player abilities to help estimate an individual player's ability.

Coding the model

We add a parameter `sigma` for σ. It is important to include the lower bound of zero, because scales must be positive.[8]

[8] If parameters are not appropriately constrained to their support, sampling may fail to initialize or revert to inefficient rejection sampling behaviors. Constraints on data, and generated quantities are just for error checking.

```
parameters {
  real<lower = 0> sigma;               // scale of ability variation
  vector[K] alpha;                     // ability for player n
}
model {
  sigma ~ lognormal(0, 0.5);           // boundary avoiding
  alpha ~ normal(0, sigma);            // hierarchical
  y ~ bernoulli_logit(alpha[player1] - alpha[player0]);
}
```

The prior on `sigma` is coded as before, and now the prior on `alpha` uses `sigma` as its scale parameter.

Fitting the model

The model is fit just as before.

```
model_hier <- stan_model("individual-hierarchical.stan")
posterior_hier <- sampling(model_hier, data = mle_model_data)
```

The fit can be examined as before for convergence violations.

```
print(posterior_hier, c("sigma", "alpha"), probs=c(0.05, 0.5, 0.95))
```

```
Inference for Stan model: individual-hierarchical.
4 chains, each with iter=2000; warmup=1000; thin=1;
post-warmup draws per chain=1000, total post-warmup draws=4000.

           mean se_mean   sd    5%   50%   95% n_eff Rhat
sigma      1.14    0.00 0.13  0.95  1.13  1.36  3086    1
alpha[1]  -1.29    0.01 0.28 -1.77 -1.29 -0.84   850    1
alpha[2]  -0.20    0.01 0.28 -0.66 -0.20  0.26   894    1
alpha[3]   0.35    0.01 0.28 -0.10  0.34  0.81   738    1
alpha[4]  -0.44    0.01 0.29 -0.90 -0.43  0.04   917    1
alpha[5]   0.80    0.01 0.26  0.37  0.80  1.24   701    1
alpha[6]  -0.35    0.01 0.28 -0.81 -0.34  0.12   783    1
alpha[7]   0.97    0.01 0.28  0.53  0.97  1.45   762    1
alpha[8]  -2.40    0.01 0.36 -3.01 -2.39 -1.83  1363    1
alpha[9]  -0.01    0.01 0.28 -0.45 -0.01  0.45   736    1
alpha[10] -0.19    0.01 0.26 -0.61 -0.18  0.24   778    1
alpha[11]  0.39    0.01 0.27 -0.04  0.39  0.83   779    1
alpha[12] -0.83    0.01 0.29 -1.31 -0.83 -0.36   941    1
alpha[13]  0.49    0.01 0.28  0.04  0.47  0.96   738    1
alpha[14] -0.74    0.01 0.27 -1.18 -0.74 -0.29   771    1
alpha[15] -1.17    0.01 0.27 -1.63 -1.16 -0.73   759    1
alpha[16]  2.75    0.01 0.38  2.14  2.73  3.42  1256    1
```

```
alpha[17]  0.13    0.01 0.27 -0.32  0.13  0.60   786    1
alpha[18]  1.72    0.01 0.34  1.18  1.72  2.28  1261    1
alpha[19] -0.99    0.01 0.29 -1.47 -0.98 -0.53   906    1
alpha[20]  0.48    0.01 0.29  0.01  0.48  0.97   824    1
alpha[21] -0.33    0.01 0.29 -0.80 -0.33  0.14   882    1
alpha[22]  1.17    0.01 0.28  0.70  1.16  1.64   799    1
alpha[23]  0.41    0.01 0.26 -0.01  0.41  0.84   715    1
alpha[24] -1.83    0.01 0.33 -2.38 -1.82 -1.31  1020    1
alpha[25] -0.55    0.01 0.27 -1.01 -0.55 -0.09   799    1
alpha[26]  1.43    0.01 0.31  0.92  1.42  1.96   877    1
alpha[27] -1.47    0.01 0.29 -1.94 -1.47 -1.00   828    1
alpha[28]  0.05    0.01 0.27 -0.40  0.05  0.50   735    1
alpha[29]  1.16    0.01 0.29  0.68  1.15  1.64   889    1
alpha[30]  2.23    0.01 0.35  1.65  2.23  2.83  1364    1
alpha[31]  0.37    0.01 0.27 -0.08  0.37  0.82   759    1
alpha[32]  1.40    0.01 0.29  0.94  1.40  1.88   851    1
alpha[33] -0.57    0.01 0.29 -1.05 -0.57 -0.11   835    1
alpha[34] -0.23    0.01 0.27 -0.66 -0.22  0.20   764    1
alpha[35] -1.04    0.01 0.28 -1.49 -1.04 -0.59   891    1
alpha[36] -1.39    0.01 0.29 -1.87 -1.39 -0.92   879    1
alpha[37] -0.32    0.01 0.28 -0.78 -0.31  0.13   768    1
alpha[38]  0.18    0.01 0.26 -0.25  0.18  0.62   699    1
alpha[39] -1.46    0.01 0.30 -1.94 -1.45 -0.98   880    1
alpha[40]  1.13    0.01 0.27  0.70  1.13  1.59   787    1
alpha[41] -0.05    0.01 0.28 -0.51 -0.05  0.43   885    1
alpha[42] -0.07    0.01 0.27 -0.51 -0.06  0.38   724    1
alpha[43]  0.83    0.01 0.27  0.39  0.83  1.27   726    1
alpha[44] -0.28    0.01 0.27 -0.73 -0.28  0.18   797    1
alpha[45] -1.47    0.01 0.29 -1.96 -1.47 -1.01   872    1
alpha[46]  0.08    0.01 0.26 -0.35  0.08  0.51   692    1
alpha[47] -1.46    0.01 0.29 -1.95 -1.46 -0.99   875    1
alpha[48]  1.52    0.01 0.30  1.03  1.51  2.04   964    1
alpha[49]  1.74    0.01 0.31  1.27  1.74  2.25   962    1
alpha[50]  0.29    0.01 0.29 -0.18  0.29  0.77   874    1

Samples were drawn using NUTS(diag_e) at Sat Nov 21 11:57:24 2020.
For each parameter, n_eff is a crude measure of effective sample size,
and Rhat is the potential scale reduction factor on split chains (at
convergence, Rhat=1).
```

The $\hat{R}$ values are all near 1 and the effective sample sizes are reasonable, so we have no reason to distrust the fit.

Hierarchical, Bayesian, Team Bradley-Terry Model

Many extensions to the Bradley-Terry model are possible, as it is really nothing more than a multi-intercept logistic regression. We will consider a team-based model in which two teams made up of several players compete in a match. Or equivalently, when two baskets of consumer goods are compared to one another. The point of such a model will be to infer the ability of the players (goods in the baskets) from the performance of the teams (baskets) in which they participate.

Additivity

As before, each player is still modeled as having an undlerlying ability on the log odds scale. The ability of a team will be the sum of the abilities of its players. This makes sense when the contributions of players are independent, but is only an approximation when there are interaction effects (such as synergies or interference among participants on the team).

Identifiability

This model will be most appropriate in situations where team membership is varied. Many games among teams with the same sets of players will not be identified. That is, there will be know way to infer the relative contributions of the players on a team if the players only ever play with each other. This model could be applied to sports in which the players participating varies during a game, as long as the segments were comparable in terms of variance.[9]

[9] For example, continuous games like football (soccer, not American), hockey, or basketball might be divided into one minute segments where a winner was assigned based on the scoring during that interval; for most of these sports, a mechanism for accounting for ties would be necessary.

Data for team Bradley-Terry model

The data for the team Bradley-Terry model needs to indicate which players are on the teams playing each other. This will perhaps be easiest to see through a concrete simulation of a complete data set.

```
K <- 50                                 # players
J <- 3                                  # players / team
N <- K^2 * J                            # matches
sigma <- 1                              # scale of player ability variation
alpha <- center(rnorm(K, 0, sigma))     # player abilities, centered log odds
team0 <- matrix(NA, N, J)               # players on team 0
team1 <- matrix(NA, N, J)               # players on team 1
for (n in 1:N) {
   players <- sample(1:K, 2 * J)        # 1:K w/o replacement for distinct teams
   team0[n, 1:J] <- players[1:J]
```

```
   team1[n, 1:J] <- players[(J + 1) : (2 * J)]
}
y <- rep(NA, N)
for (n in 1:N) {
  alpha_team0 = sum(alpha[team0[n, 1:J]])
  alpha_team1 = sum(alpha[team1[n, 1:J]])
  y[n] <- rbinom(1, 1, inv_logit(alpha_team1 - alpha_team0))
}
team_data <- list(K = K, J = J, N = N, team0 = team0, team1 = team1, y = y)
```

There are K players in total, with J players on each team. There are a total of N matches. Here we use quite a few matches to make it easy to visualize the model fit (as it will be close to the true values). We set the population variation of abilities at $\sigma = 1$, and then generate the vector of player abilities $\alpha \sim \text{Normal}(0, \sigma)$. We then generate the teams by sampling J players for each team without replacement from among the K teams. Then y is sampled using a binomial random number generator given the sum of the team one's abilities minus the sum of team zero's.

Stan model

Coding up that generative process in Stan leads to the following model.

```
data {
  int<lower = 0> K;                       // players
  int<lower = 0> J;                       // players per team
  int<lower = 0> N;                       // matches
  int<lower = 1, upper = K> team0[N, J];  // team 0 players
  int<lower = 1, upper = K> team1[N, J];  // team 1 players
  int<lower = 0, upper = 1> y[N];         // winner
}
parameters {
  vector[K] alpha;
  real<lower = 0> sigma;
}
model {
  sigma ~ lognormal(0, 0.5);           // zero avoiding, weakly informative
  alpha ~ normal(0, sigma);                  // hierarchical, zero centered
  for (n in 1:N)                          // additive Bradley-Terry model
    y[n] ~ bernoulli_logit(sum(alpha[team1[n]]) - sum(alpha[team0[n]]));
}
generated quantities {
  int<lower=1, upper=K> ranking[K];       // rank of player ability
```

```
  {
    int ranked_index[K] = sort_indices_desc(alpha);
    for (k in 1:K)
      ranking[ranked_index[k]] = k;
  }
}
```

```
team_model <- stan_model("team.stan")
team_posterior <- sampling(team_model, data = team_data, init=0.5)
```

```
Warning: Bulk Effective Samples Size (ESS) is too low, indicating posterior means and medians may be unrel
Running the chains for more iterations may help. See
http://mc-stan.org/misc/warnings.html#bulk-ess
```

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

```
print(team_posterior, c("sigma", "alpha"), probs=c(0.05, 0.5, 0.95))
```

```
Inference for Stan model: team.
4 chains, each with iter=2000; warmup=1000; thin=1;
post-warmup draws per chain=1000, total post-warmup draws=4000.

           mean se_mean   sd    5%   50%   95% n_eff Rhat
sigma      0.98    0.01 0.10  0.83  0.98  1.16   177    1
alpha[1]   0.46    0.02 0.17  0.18  0.46  0.73   105    1
alpha[2]   0.66    0.02 0.17  0.39  0.67  0.94   105    1
alpha[3]  -1.27    0.02 0.17 -1.56 -1.26 -0.99   111    1
alpha[4]  -0.82    0.02 0.17 -1.10 -0.82 -0.56   106    1
alpha[5]   0.68    0.02 0.17  0.40  0.68  0.94   104    1
alpha[6]   1.94    0.02 0.18  1.64  1.95  2.23   117    1
alpha[7]   1.99    0.02 0.18  1.70  1.99  2.27   114    1
alpha[8]  -0.20    0.02 0.17 -0.49 -0.20  0.06   107    1
alpha[9]   0.81    0.02 0.17  0.52  0.81  1.08   106    1
alpha[10]  0.31    0.02 0.17  0.02  0.32  0.58   105    1
alpha[11] -0.28    0.02 0.17 -0.56 -0.28 -0.02   102    1
alpha[12] -0.02    0.02 0.17 -0.30 -0.01  0.25   103    1
alpha[13]  0.13    0.02 0.17 -0.15  0.13  0.39   112    1
alpha[14]  1.07    0.02 0.17  0.79  1.07  1.35   108    1
alpha[15] -0.54    0.02 0.17 -0.82 -0.53 -0.28   105    1
alpha[16] -0.55    0.02 0.17 -0.84 -0.55 -0.28   112    1
alpha[17]  0.84    0.02 0.17  0.55  0.84  1.11   103    1
alpha[18]  1.74    0.02 0.17  1.45  1.74  2.02   110    1
alpha[19] -0.51    0.02 0.17 -0.80 -0.50 -0.24   106    1
alpha[20] -1.11    0.02 0.17 -1.39 -1.11 -0.84   109    1
```

```
alpha[21] -0.36    0.02 0.17 -0.65 -0.35 -0.08   104    1
alpha[22] -0.11    0.02 0.17 -0.39 -0.11  0.16   105    1
alpha[23] -0.98    0.02 0.17 -1.27 -0.98 -0.71   103    1
alpha[24]  0.16    0.02 0.17 -0.12  0.16  0.42   108    1
alpha[25] -0.68    0.02 0.17 -0.96 -0.68 -0.42   106    1
alpha[26] -1.05    0.02 0.17 -1.35 -1.05 -0.78   104    1
alpha[27] -0.28    0.02 0.17 -0.55 -0.28 -0.02    99    1
alpha[28] -1.56    0.02 0.17 -1.85 -1.56 -1.28   112    1
alpha[29]  0.68    0.02 0.17  0.39  0.68  0.95   109    1
alpha[30]  0.38    0.02 0.17  0.10  0.38  0.64   106    1
alpha[31] -0.55    0.02 0.17 -0.83 -0.55 -0.27   110    1
alpha[32] -0.60    0.02 0.17 -0.88 -0.60 -0.34   103    1
alpha[33] -0.76    0.02 0.17 -1.03 -0.76 -0.49   103    1
alpha[34]  2.04    0.02 0.18  1.74  2.05  2.33   113    1
alpha[35]  0.30    0.02 0.17  0.01  0.31  0.57   103    1
alpha[36] -0.52    0.02 0.17 -0.80 -0.52 -0.25   102    1
alpha[37]  0.81    0.02 0.17  0.53  0.81  1.09   103    1
alpha[38]  1.47    0.02 0.17  1.18  1.47  1.75   109    1
alpha[39] -1.89    0.02 0.18 -2.19 -1.89 -1.61   116    1
alpha[40] -0.47    0.02 0.17 -0.74 -0.47 -0.21   107    1
alpha[41]  0.07    0.02 0.17 -0.21  0.08  0.35   107    1
alpha[42] -0.97    0.02 0.17 -1.25 -0.96 -0.69   106    1
alpha[43]  1.99    0.02 0.18  1.70  2.00  2.28   123    1
alpha[44] -0.69    0.02 0.17 -0.97 -0.69 -0.42   108    1
alpha[45] -1.23    0.02 0.17 -1.52 -1.22 -0.96   112    1
alpha[46]  0.25    0.02 0.17 -0.03  0.26  0.53   105    1
alpha[47] -0.57    0.02 0.17 -0.85 -0.57 -0.30   107    1
alpha[48]  0.14    0.02 0.17 -0.14  0.15  0.41   103    1
alpha[49] -1.18    0.02 0.17 -1.46 -1.17 -0.90   106    1
alpha[50] -0.05    0.02 0.17 -0.32 -0.05  0.21   109    1

Samples were drawn using NUTS(diag_e) at Sat Nov 21 11:59:05 2020.
For each parameter, n_eff is a crude measure of effective sample size,
and Rhat is the potential scale reduction factor on split chains (at
convergence, Rhat=1).
```

As before, we will pull out the Bayesian parameter estimates.

```
alpha_hat <- rep(NA, K)
for (k in 1:K)
  alpha_hat[k] <- mean(extract(team_posterior)$alpha[ , k])
```

and plot against the simulated values.

```
team_bayes_fit_plot <-
  ggplot(data.frame(alpha = alpha, alpha_hat = alpha_hat),
```

```
        aes(x = alpha, y = alpha_hat)) +
    geom_abline(slope = 1, intercept = 0, color="green", size =
    geom_point(size = 2) +
    ggtheme_tufte()
team_bayes_fit_plot
```

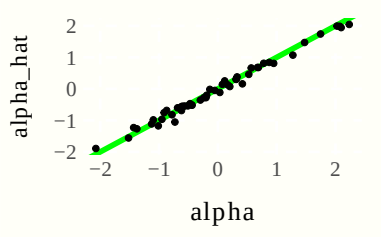

Figure 3: Fit of true value (horizontal axis) versus Bayesian (posterior mean) estimate (vertical axis) for team-based Bradley-Terry model. As with the previous model, inference for true abilities is very good.

Exercises

1. Explore the sensitivity of the prior by contrasting fits with priors on the scale parameter of $\sigma \sim \text{LogNormal}(0, 1)$ and $\sigma \sim \text{LogNormal}(0, 0.25)$. Then consider a simple half-Cauchy prior $\sigma \sim \text{Cauchy}(0, 1)$ and a half-normal prior $\sigma \sim \text{Normal}(0, 1)$.

2. Consider an extended Bradley-Terry model where there is a "home-field advantage." This may literally be a home-field advantage to the sports team playing in their home field (stadium, arena, etc.), or it may be something like the advantage to the player with the white pieces in chess. Assume that player 1 is the "home" player in the data coding and that there is a global intercept term representing this advantage on the log scale. Simulate data and fit it as in the above examples. Is the home-field advantage recovery? Given that we know home-field advantages are positive, does it make sense to constrain the home-field advantage parameter to be positive? If you do and the effect is fairly small and consistent with zero in the posterior, what happens to its estimate with the constraint versus without the constraint (assuming the same prior)?

References

- Bradley, Ralph Allan and Milton E. Terry. 1952. Rank analysis of incomplete block designs: I. The method of paired comparisons. *Biometrika* **39**(3/4): 324.
- Leonard, T. 1977. An alternative Bayesian approach to the Bradley-Terry model for paired comparisons. *Biometrics* 33(1):121–132.
- Zermelo, E. 1929. Die berechnung der turnier-ergebnisse als ein maximumproblem der wahrscheinlichkeitsrechnung. *Mathematische Zeitschrift* 29(1), 436–460.

Source code

All of the source code, data, text, and images for this case study are available on GitHub at

- stan-dev/example-models/knitr/bradley-terry

Session information

```
sessionInfo()
```

```
R version 4.0.3 (2020-10-10)
Platform: x86_64-apple-darwin17.0 (64-bit)
Running under: macOS Catalina 10.15.7

Matrix products: default
BLAS:   /Library/Frameworks/R.framework/Versions/4.0/Resources/lib/libRblas.dylib
LAPACK: /Library/Frameworks/R.framework/Versions/4.0/Resources/lib/libRlapack.dylib

locale:
[1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8

attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods   base

other attached packages:
[1] tufte_0.8            rstan_2.21.1         StanHeaders_2.21.0-6
[4] reshape_0.8.8        knitr_1.30           gridExtra_2.3
[7] ggplot2_3.3.2

loaded via a namespace (and not attached):
 [1] tidyselect_1.1.0   xfun_0.18          purrr_0.3.4        V8_3.3.1
 [5] colorspace_1.4-1   vctrs_0.3.4        generics_0.0.2     htmltools_0.5.0
 [9] stats4_4.0.3       loo_2.3.1          yaml_2.2.1         rlang_0.4.8
[13] pkgbuild_1.1.0     pillar_1.4.6       glue_1.4.2         withr_2.3.0
[17] matrixStats_0.57.0 lifecycle_0.2.0    plyr_1.8.6         stringr_1.4.0
[21] munsell_0.5.0      gtable_0.3.0       codetools_0.2-16   evaluate_0.14
[25] labeling_0.4.2     inline_0.3.16      callr_3.5.1        ps_1.4.0
[29] curl_4.3           parallel_4.0.3     fansi_0.4.1        Rcpp_1.0.5
[33] scales_1.1.1       RcppParallel_5.0.2 jsonlite_1.7.1     farver_2.0.3
[37] digest_0.6.27      stringi_1.5.3      processx_3.4.4     dplyr_1.0.2
[41] grid_4.0.3         cli_2.1.0          tools_4.0.3        magrittr_1.5
[45] tibble_3.0.4       crayon_1.3.4       pkgconfig_2.0.3    ellipsis_0.3.1
[49] prettyunits_1.1.1  assertthat_0.2.1   rmarkdown_2.5      R6_2.5.0
[53] compiler_4.0.3
```

Licenses