Using rstan

Jake Thompson

9/12/2018

1 The data

For this example, I have simulated an example data set with 500 respondents taking a 20 item test. Further, there is 5 percent missing data. *Stan* cannot handle missing data explicity. Therefore, we must turn the explicit missing data into implicit missing data. This can be done by transforming the data from wide format to long format, and filtering out any cases where the score is `NA`.

```
response_data <- read_csv(here("data/sample-data.csv"),
  col_types = cols(.default = col_integer())) %>%
  gather(key = "item_id", value = "score", -student_id) %>%
  mutate(item_id = as.integer(item_id)) %>%
  filter(!is.na(score)) %>%
  arrange(student_id, item_id)

response_data
## # A tibble: 28,531 x 3
##    student_id item_id score
##         <int>   <int> <int>
##  1          1       1     0
##  2          1       2     0
##  3          1       3     1
##  4          1       4     0
##  5          1       5     0
##  6          1       6     0
##  7          1       7     1
##  8          1       8     1
##  9          1       9     0
## 10          1      10     1
## # ... with 28,521 more rows
```

The data must then be formatted for use in *Stan*. The **rstan** package expects a named list, with the names corresponding to the data elements defined in the `.stan` file. Looking at `Stan/irt-3pl.stan`, we see the following (the full `.stan` file can be seen in Appendix A):

```
data {
  int<lower=1> J;                         // number of respondents
  int<lower=1> K;                         // number of items
  int<lower=1> N;                         // number of observations
  int<lower=1,upper=J> jj[N];             // respondent for observation n
  int<lower=1,upper=K> kk[N];             // item for observation n
  int<lower=0,upper=1> y[N];              // score for observation n
}
```

We can therefore define our data for the **rstan** package as a list with the elements `J`, `K`, `N`, `jj`, `kk`, and `y`.

```
stan_data = list(
  J = length(unique(response_data$student_id)),
  K = length(unique(response_data$item_id)),
  N = nrow(response_data),
  jj = response_data$student_id,
  kk = response_data$item_id,
  y = response_data$score
)
```

Now that the data has been correctly defined, we can estimate the model.

2 Model estimation

The model can be estimated by using the `stan` function. Here, we provide the path to the `.stan` script and our previously created data list. We also specify the number of chains, length of each chain, and the length of the warm-up or burn-in period. Because we have three chains, each with 2,000 iterations with the first 1,000 discarded, we will end up with total of 3,000 samples in our posteriors (1,000 from each chain).

```
model <- stan(file = here("Stan/irt-3pl.stan"), data = stan_data, chains = 3,
  iter = 2000, warmup = 1000, cores = 3)
```

3 Model analysis

We can view a summary of the model by using the summary function. By default, the summary method for `stanfit` objects returns a list with elements `summary` and `c_summary`, which provide the summary for merged chains and individual chains respectively. This summary provides us with the mean, Monte Carlo standard error, standard deviation, quantiles, effective sample size, and Rhat values.

```
model_summary <- summary(model, pars = c("theta", "b", "a", "c"),
    probs = c(0.025, 0.50, 0.975))$summary %>%
  as.data.frame() %>%
  rownames_to_column(var = "parameter") %>%
  as_data_frame() %>%
  separate(parameter, into = c("param", "index"), sep = "\\[|\\]",
    extra = "drop", convert = TRUE) %>%
  mutate_if(is.double, round, digits = 2)

model_summary
## # A tibble: 1,090 x 10
##    param index  mean se_mean    sd `2.5%` `50%` `97.5%` n_eff  Rhat
##    <chr> <int> <dbl>   <dbl> <dbl>  <dbl> <dbl>   <dbl> <dbl> <dbl>
##  1 theta     1 -0.93    0.01  0.5   -1.99 -0.89   -0.08  3000     1
##  2 theta     2 -0.16    0.01  0.39  -0.97 -0.14    0.55  3000     1
##  3 theta     3 -0.54    0.01  0.49  -1.65 -0.5     0.28  3000     1
##  4 theta     4  0.21    0.01  0.44  -0.74  0.25    0.98  3000     1
##  5 theta     5 -0.73    0.01  0.53  -1.9  -0.69    0.15  3000     1
##  6 theta     6 -1.38    0.01  0.6   -2.61 -1.36   -0.28  3000     1
##  7 theta     7 -0.77    0.01  0.59  -1.93 -0.76    0.28  3000     1
##  8 theta     8  1.18    0.01  0.49   0.26  1.17    2.17  3000     1
##  9 theta     9  1.07    0.01  0.46   0.21  1.05    1.99  3000     1
## 10 theta    10 -0.56    0.01  0.45  -1.56 -0.52    0.24  3000     1
## # ... with 1,080 more rows
```

3.1 Parameter recovery

We can then use our estimates to look at the parameter recovery of our model, to make sure it is specified correctly. Let's first look at the recovery of our person parameters (θ). The true values used to simulate the data are saved in the `data/` directory. The estimated values can be extracted from the `model_summary`. Finally, we can join the two sets of estimates together, and create a plot like in Figure 1.

```
true_theta <- read_csv(here("data/respondent-parameters.csv"),
  col_types = cols(respondent_id = col_integer(), theta = col_double())) %>%
  rename(true_theta = theta)

est_theta <- model_summary %>%
  filter(param == "theta") %>%
  select(respondent_id = index, est_theta = mean)

theta_recovery <- left_join(true_theta, est_theta, by = "respondent_id")
```

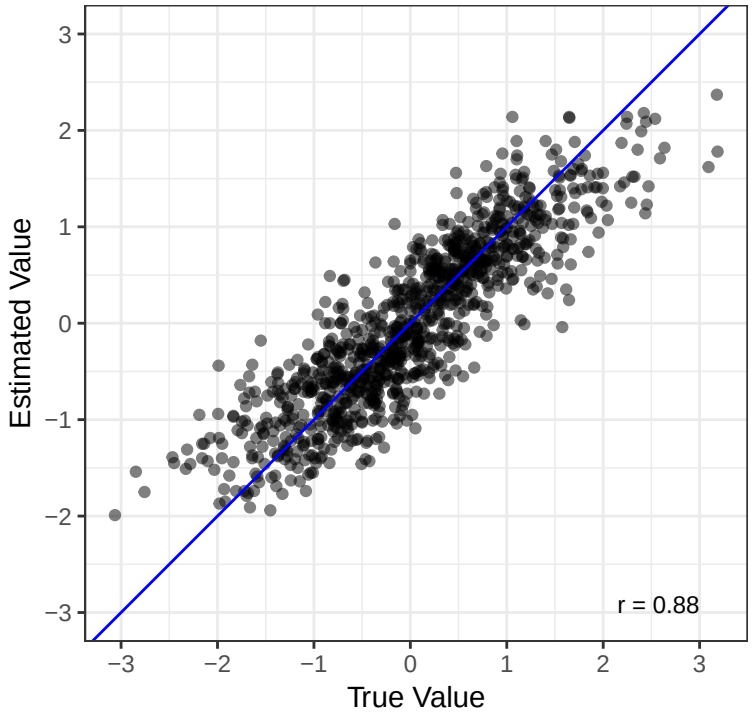

Figure 1: Recovery of θ parameter.

Here we can see that the correlation between our true values and estimated values is 0.88, indicating good recovery of the person parameters.

Similarly, we can examine the recovery of the item parameters. The true values for the item parameters can also be found in the `data/` directory, and the estimated values extracted from the `model_summary`. We can then create a facetted plot, as in Figure 2, to view the recovery for all item parameters at the same time.

```
true_item <- read_csv(here("data/item-parameters.csv"),
  col_types = cols(.default = col_double(), item_id = col_integer())) %>%
  gather(key = "param", value = "true_value", -item_id)

est_item <- model_summary %>%
  filter(param %in% c("b", "a", "c")) %>%
  select(param, item_id = index, est_value = mean)

item_recovery <- left_join(true_item, est_item, by = c("param", "item_id"))
```

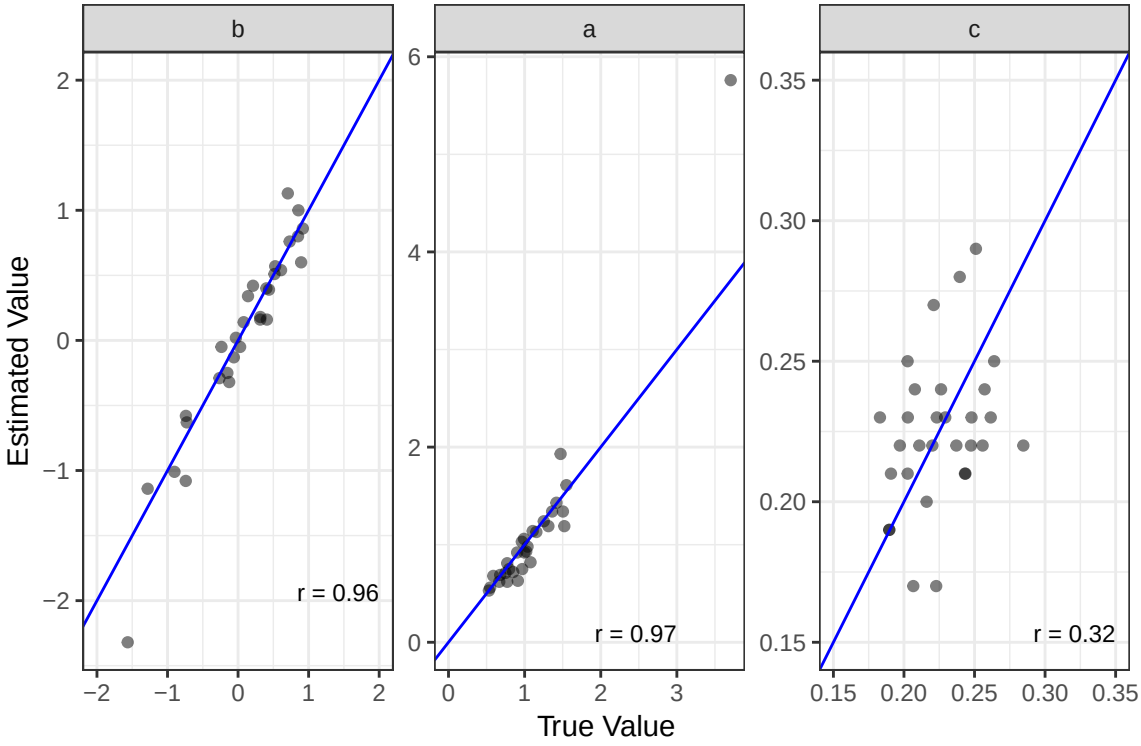

Figure 2: Recovery of item parameters

In Figure 2 we can see that we have excellent recovery for the b (r = 0.96) and a (r = 0.97) parameters, and moderate to poor recovery for the c parameters r = 0.32.

A *Stan* code for 3-PL item response theory model

```
data {
  int<lower=1> J;                          // number of respondents
  int<lower=1> K;                          // number of items
  int<lower=1> N;                          // number of observations
  int<lower=1,upper=J> jj[N];              // respondent for observation n
  int<lower=1,upper=K> kk[N];              // item for observation n
  int<lower=0,upper=1> y[N];               // score for observation n
}
parameters {
  real theta[J];
  real b[K];
  real<lower=0> a[K];
  real<lower=0,upper=1> c[K];
}
model {
  vector[N] eta;

  // priors
  theta ~ normal(0, 1);
  b ~ normal(0, 10);
  a ~ lognormal(0.5, 1);
  c ~ beta(5, 17);

  // model
  for (n in 1:N) {
    eta[n] = c[kk[n]] + (1 - c[kk[n]]) * inv_logit(a[kk[n]] * (theta[jj[n]] - b[kk[n]]));
  }
  y ~ bernoulli(eta);
}
```