The Exploded Logit Model

Michael Betancourt

November 2025

Table of contents

1 The Exploded Logit Model

The exploded logit model is a probabilistic model for observed rankings, where each ranking orders a set of items from best to worst, for some notion of best and worst.

In an exploded logit model the relative quality of each item is quantified with a quality parameter

$$\alpha_1, \ldots, \alpha_K,$$

similar to the setup of a pairwise comparison model.

Given the quality parameters we can then model each rank sequentially, considering only the remaining items at each step. First we model the first rank with a categorical model over all of the items,

$$p(r_1 = k) = \frac{\exp(\alpha_k)}{\sum_{k'=1}^{I} \exp(\alpha_{k'})}.$$

Next we model the second rank with a categorical model over all items except the top ranking item,

$$p(r_2 = k) = \frac{\exp(\alpha_k)}{\sum_{k' \neq r_1} \exp(\alpha_{k'})}.$$

Then we iterate, slowly whittling down the items until only one remains,

$$p(r_{k''} = k) = \frac{\exp(\alpha_k)}{\sum_{k' \neq r_1, \ldots, r_{k''-1}} \exp(\alpha_{k'})}.$$

Note that the final rank r_I does not contribute anything to the probabilistic model because it is completely determined by the first $K - 1$ ranks.

The main challenge with implementing this model in practice is not confusing ranks, items, and any indices we use to implement either in code. To ensure an efficient implementation we also have to take care to avoid redundant, and hence wasteful, calculations as we loop through the ranks.

Because ranking models are not easy to develop, the exploded logit model is popular in many applications. That said it does not always perform well. The main limitation of the exploded logit model is that the items behave the same for all ranks.

Consider, for example, using an exploded logit model to model the outcome of a race. The first rank captures first place, the second rank captures second place, and so on. In real races the strategy, and hence behavior, of racers often depends on their current position. For example a racer in first place might race less aggressively than racer in second or even last place. An exploded logit model is not able to capture this more nuanced behavior.

2 Setup

To demonstrate the implementation of this model we'll first need to setup our local `R` environment.

```
par(family="serif", las=1, bty="l",
    cex.axis=1, cex.lab=1, cex.main=1,
    xaxs="i", yaxs="i", mar = c(5, 5, 3, 1))

library(rstan)
rstan_options(auto_write = TRUE)            # Cache compiled Stan programs
options(mc.cores = parallel::detectCores()) # Parallelize chains
parallel:::setDefaultClusterOptions(setup_strategy = "sequential")
```

```
util <- new.env()
source('mcmc_analysis_tools_rstan.R', local=util)
source('mcmc_visualization_tools.R', local=util)
```

3 Simulate Data

Nine items makes for clean, symmetric plots.

```
K <- 9
delta_true <- c(0.0000000, 1.0163341, 0.3225904,
                0.2677161, 2.3349013, 2.0759525,
                1.5426314, 3.1642413, 4.0853040)
```

Simulating data is straightforward in theory, but requires some care in practice in order to properly track the indices of the remaining items. Here I effectively implement an array of decreasing side by moving the remaining item indices to the left and decreasing the total number of elements considered.

```
N <- 100
simu <- stan(file="stan_programs/simu_rankings.stan",
             algorithm="Fixed_param",
             data=list('N' = N, 'K' = K, 'delta_true' = delta_true),
             seed=8438338, warmup=0, iter=1, chains=1, refresh=0)
```

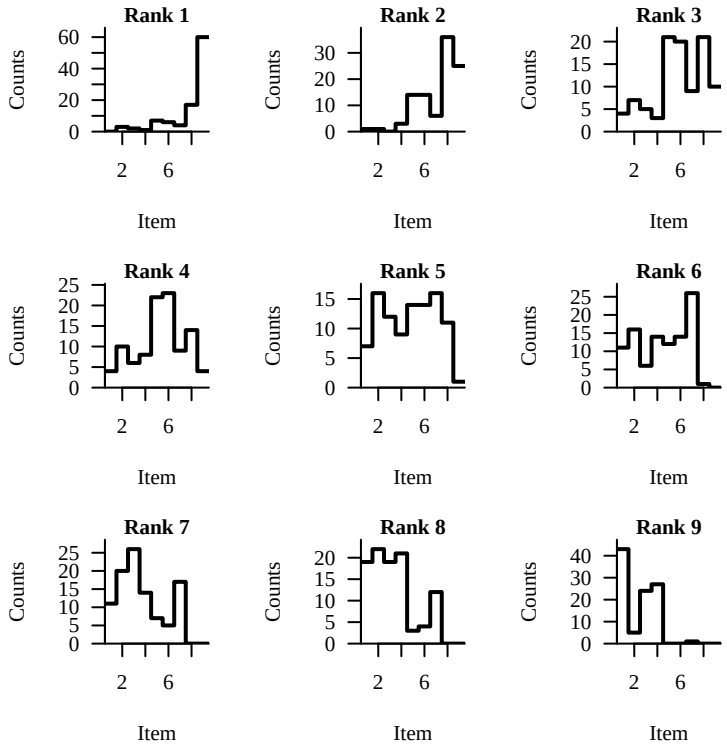
```
data <- list('N' = N, 'K' = K,
             'ranks' = extract(simu)$ranks[1,,])
```

4 Data Exploration

Because ranks are not one-dimensional – each rank consists of K values – the best way to visualize an ensemble of observed ranks is not straightforward.

Here I display a histogram of the items that appear in each rank across the ensemble.

```
par(mfrow=c(3, 3), mar=c(5, 5, 1, 1))

for (k in 1:data$K)
  util$plot_line_hist(data$ranks[,k], 0.5, data$K + 0.5, 1,
                      xlab="Item", main=paste("Rank", k))
```



5 Posterior Inferences

When implementing the exploded logit model I loop over the ranks in reverse, which allows me to buildup the probability normalizations iteratively without any wasted calculations.

Additionally I anchor the first item quality to zero to avoid redundancy, and the resulting inferential degeneracies.

```
fit <- stan(file="stan_programs/rankings.stan",
            data=data, seed=8438338,
            warmup=1000, iter=2024, refresh=250)
```

No signs of incomplete posterior quantification.

```
diagnostics <- util$extract_hmc_diagnostics(fit)
util$check_all_hmc_diagnostics(diagnostics)
```

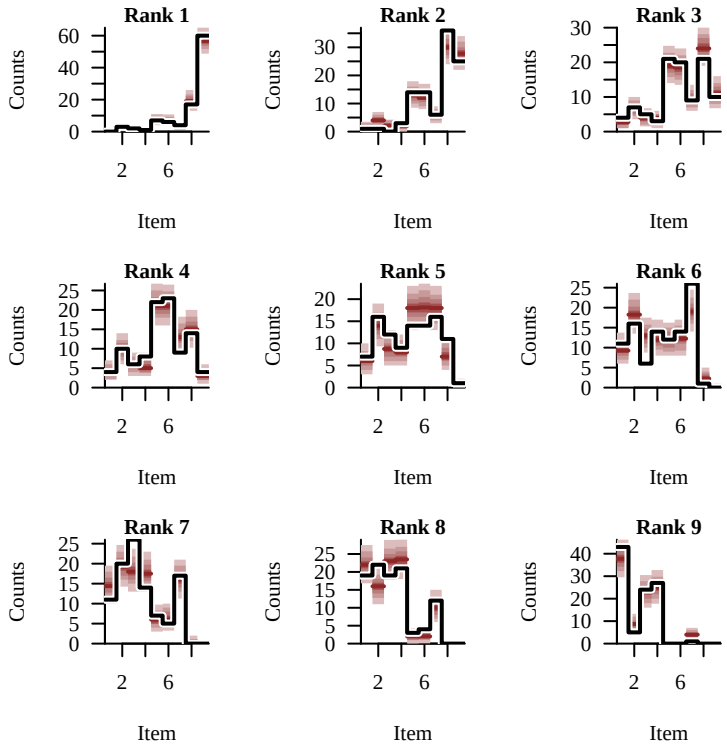
```
  All Hamiltonian Monte Carlo diagnostics are consistent with reliable
Markov chain Monte Carlo.
```

```
samples <- util$extract_expectand_vals(fit)
base_samples <- util$filter_expectands(samples,
                                       c('delta_free'),
                                       check_arrays=TRUE)
util$check_all_expectand_diagnostics(base_samples)
```

```
All expectands checked appear to be behaving well enough for reliable
Markov chain Monte Carlo estimation.
```

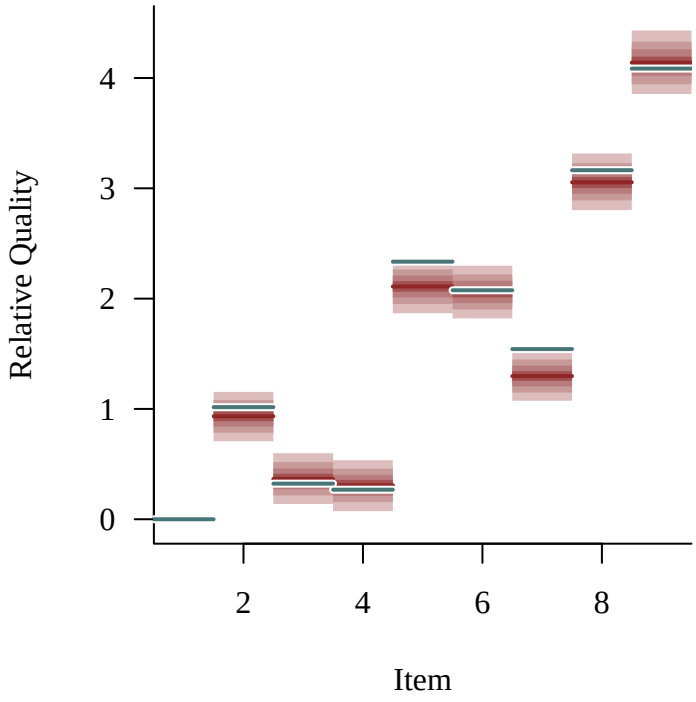
The posterior retrodictive checks are clean. Because we're simulating data and drawing inferences from the same model this is expected, but it's a nice cross check to make sure that we're implementing everything consistently.

```
par(mfrow=c(3, 3), mar=c(5, 5, 1, 1))

for (k in 1:data$K) {
  names <- sapply(1:data$N,
                  function(n)
                  paste0('ranks_pred[', n, ',', k, ']'))
  base_samples <- util$filter_expectands(samples, names)

  util$plot_hist_quantiles(base_samples, 'ranks_pred',
                           0.5, data$K + 0.5, 1,
                           baseline_values=data$ranks[,k],
                           xlab="Item", main=paste("Rank", k))
}
```



Also unsurprising, but nice to see, is the consistency of our inferences with the true model configuration from which the data were simulated.

```
par(mfrow=c(1, 1), mar=c(5, 5, 1, 1))

names <- sapply(1:data$K, function(k) paste0('delta[', k, ']'))
util$plot_disc_pushforward_quantiles(samples, names,
                                     baseline_values=delta_true,
                                     baseline_col=util$c_mid_teal,
                                     xlab="Item", ylab="Relative Quality")
```



License

Original Computing Environment

```
writeLines(readLines(file.path(Sys.getenv("HOME"), ".R/Makevars")))
```

```
CC=clang

CXXFLAGS=-O3 -mtune=native -march=native -Wno-unused-variable -Wno-unused-function -Wno-macr
CXX=clang++ -arch x86_64 -ftemplate-depth-256
```

```
CXX14FLAGS=-O3 -mtune=native -march=native -Wno-unused-variable -Wno-unused-function -Wno-ma
CXX14=clang++ -arch x86_64 -ftemplate-depth-256
```

```
sessionInfo()
```

```
R version 4.3.2 (2023-10-31)
Platform: x86_64-apple-darwin20 (64-bit)
Running under: macOS 15.6.1

Matrix products: default
BLAS:   /Library/Frameworks/R.framework/Versions/4.3-x86_64/Resources/lib/libRblas.0.dylib
LAPACK: /Library/Frameworks/R.framework/Versions/4.3-x86_64/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

time zone: America/New_York
tzcode source: internal

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

other attached packages:
[1] colormap_0.1.4     rstan_2.32.6       StanHeaders_2.32.7

loaded via a namespace (and not attached):
 [1] gtable_0.3.4       jsonlite_1.8.8     compiler_4.3.2     Rcpp_1.0.11
 [5] parallel_4.3.2     gridExtra_2.3      scales_1.3.0       yaml_2.3.8
 [9] fastmap_1.1.1      ggplot2_3.4.4      R6_2.6.1           curl_5.2.0
[13] knitr_1.45         tibble_3.2.1       munsell_0.5.0      pillar_1.9.0
[17] rlang_1.1.2        utf8_1.2.4         V8_4.4.1           inline_0.3.19
[21] xfun_0.41          RcppParallel_5.1.7 cli_3.6.2          magrittr_2.0.3
[25] digest_0.6.33      grid_4.3.2         lifecycle_1.0.4    vctrs_0.6.5
[29] evaluate_0.23      glue_1.6.2         QuickJSR_1.0.8     codetools_0.2-19
[33] stats4_4.3.2       pkgbuild_1.4.3     fansi_1.0.6        colorspace_2.1-0
[37] rmarkdown_2.25     matrixStats_1.2.0  tools_4.3.2        loo_2.6.0
[41] pkgconfig_2.0.3    htmltools_0.5.7
```

Stan
Program 1 `simu\_rankings.stan`

```
data {
  int<lower=1> N; // Number of observations
  int<lower=1> K; // Number of items

  vector[K] delta_true; // Relative item qualities
}

generated quantities {
  array[N, K] int<lower=1, upper=K> ranks;
  {
    for (n in 1:N) {
      int K_remaining = K;
      array[K] int remaining_items = linspaced_int_array(K, 1, K);
      for (k in 1:(K - 1)) {
        // Sample from remaining items
        array[K + 1 - k] int items = remaining_items[1:K_remaining];
        vector[K + 1 - k] p = softmax(delta_true[items]);
        int r = categorical_rng(p);
        ranks[n, k] = remaining_items[r];

        // Slide sampled item to end of remaining items
        for (kk in r:(K_remaining - 1)) {
          remaining_items[kk] = remaining_items[kk + 1];
        }
        remaining_items[K_remaining] = ranks[n, k];
        K_remaining -= 1;
      }
      ranks[n, K] = remaining_items[1];
    }
  }
}
```

Stan
Program 2 `rankings.stan`

```
data {
  int<lower=1> N; // Number of observations
  int<lower=1> K; // Number of items

  array[N, K] int<lower=1, upper=K> ranks; // Observed rankings
}

parameters {
  vector[K - 1] delta_free; // Relative item qualities
}

transformed parameters {
  vector[K] delta = append_row(0, delta_free);
}

model {
  target += normal_lpdf(delta_free | 0, 4 / 2.32);

  for (n in 1:N) {
    // Incorporate probability of each rank in reverse
    real log_norm = delta[ranks[n, K]];
    for (rk in 1:(K - 1)) {
      int k = K - rk;
      int r = ranks[n, k];
      log_norm = log_sum_exp(log_norm, delta[r]);
      target += delta[r] - log_norm;
    }
  }
}

generated quantities {
  array[N, K] int<lower=1, upper=K> ranks_pred;
  {
    for (n in 1:N) {
      int K_remaining = K;
      array[K] int remaining_items = linspaced_int_array(K, 1, K);
      for (k in 1:(K - 1)) {
        // Sample from remaining items
        array[K + 1 - k] int items = remaining_items[1:K_remaining];
        vector[K + 1 - k] p = softmax(delta[items]);
        int r = categorical_rng(p);
        ranks_pred[n, k] = remaining_items[r];

        // Slide sampled item to end of remaining items
        for (kk in r:(K_remaining - 1)) {
          remaining_items[kk] = remaining_items[kk + 1];
        }
        remaining_items[K_remaining] = ranks_pred[n, k];
        K_remaining -= 1;
      }
```