PREVIC Item selection

2023-07-20

Packages

Data

The original dataset contains response data of 379 items from 1190 participants. It is a quite large dataset. In the original paper, the authors used a set of 'computational expensive' approaches, that can hardly run on a 'personal computer' like mine. But at some point I think there is a certain extent of computation power waste, that one can actually conduct similar task with similar model performance (or even better with alternative reasonings/considerations.)

I'll quicky review the goal of the paper and start my alternative demo.

goal: i. I want to have a set of 'good' items, that my ultimate models follows 1PL (Rasch) settings (i.e., have better or equal performance even not estimating discrimination); ii. I want my items list can have a good difficulty distribution, that children from all ability ranges can have conduct informative 'perfect' item in the test (in Rasch model, 'perfect' item is the one with the same difficulty score as participants' ability);

```
data <- read_csv("../data/previc_data.csv")%>%
  mutate(age_group = factor(substr(age, 1,1)))
```

```
## Rows: 451010 Columns: 9
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (5): word, word_type, english, sex, subjID
## dbl (4): score, trial, aoa_rating_german, age
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
irt_dat <- data%>%
  select(subjID, word, score, sex, aoa_rating_german)

aoa <- data%>%distinct(word, .keep_all = T)%>%select(word, aoa_rating_german)

full <- data%>%
  group_by(subjID)%>%
  summarise(mean_full = mean(score))


target_words_orig <- read_csv("../data/final_item_list.csv") %>%
  pull(word)
```

```
## Rows: 89 Columns: 3
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (3): word, english, word_type
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
prior_2pl <-
  prior("normal(0, 2)", class = "b", nlpar = "eta") +
  prior("normal(0, 1)", class = "b", nlpar = "logalpha") +
  prior("constant(1)", class = "sd", group = "subjID", nlpar = "eta") +
  prior("normal(0, 3)", class = "sd", group = "word", nlpar = "eta") +
  prior("normal(0, 1)", class = "sd", group = "word", nlpar = "logalpha")

prior_1pl <-
  prior("normal(0, 1)", class = "sd", group = "subjID") +
  prior("normal(0, 3)", class = "sd", group = "word")

# Note: I mainly adopt the original prior settings, but I'm not sure why the original paper used (0,1)

formula_irt1 <- bf(
    score ~ 1 + (1 | word) + (1 | subjID)
  ) # 1pl



formula_irt2 <- bf(
    score ~ exp(logalpha) * eta,
    eta ~ 1 + (1 |i| word) + (1 | subjID),
    logalpha ~ 1 + (1 |i| word),
    nl = TRUE #2pl
  )
```

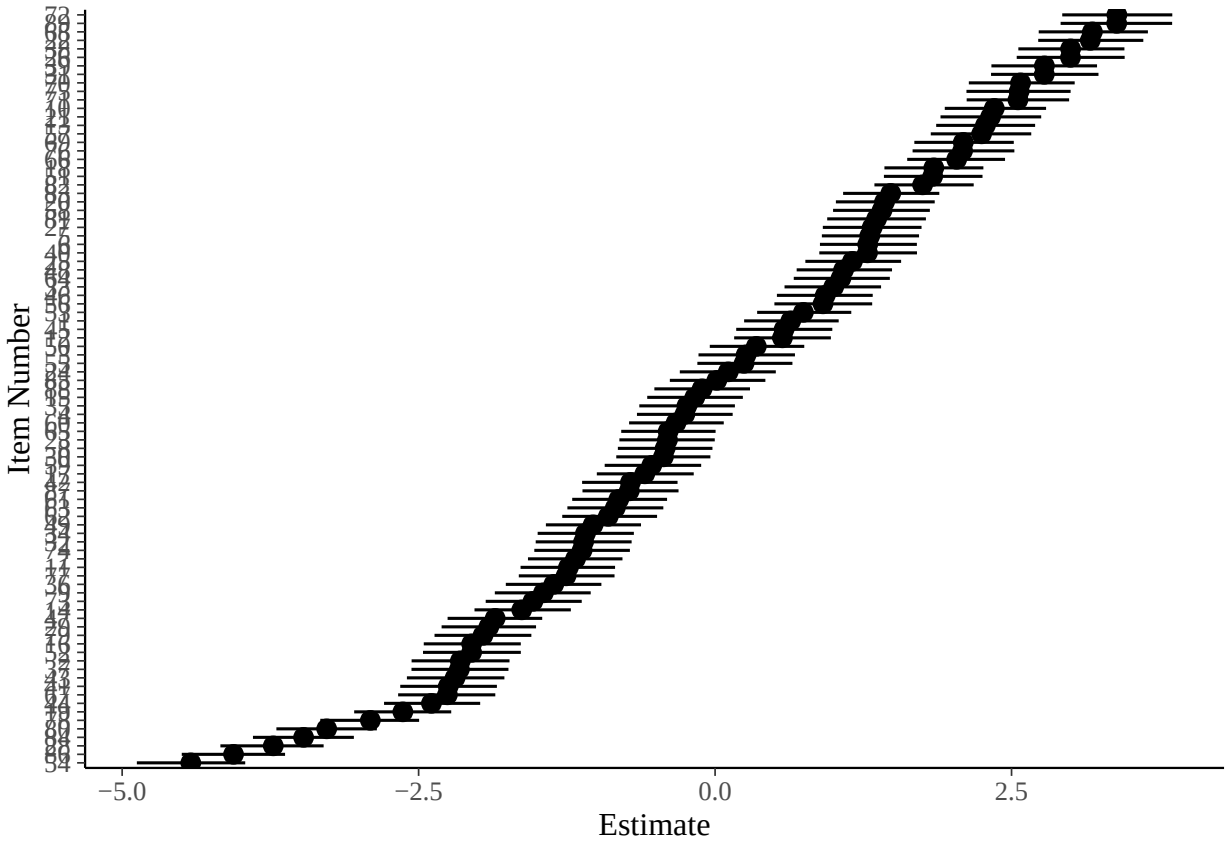
Original solution

Let's quickly review the original solution in the paper. Following a rigor (yet really expensive) workflow, authors get 89 items. The model includes difficulty distributions like this: It looks overall good, as it has a good distribution from -5 ~ 4, and all items difficulty posterior seems not too wide. But if someone is picky enough, might spot on that from -5 ~ -2.5, the distribution seems quite sparse. Another critique might be, the simulated annealing algorithm blindly optimizes for equidistant spacing via a stochastic process, fundamentally ignoring whether the resulting difficulty distribution makes practical sense. Consequently, it treats an item with a Rasch difficulty of -5 as a valid selection to fill a gap, even though such an item is likely too easy to provide good enough information.

```
irt_dat_ori <- irt_dat %>%
  filter(word %in% target_words_orig)

irt_ori <- brm(
  formula = formula_irt1,
  data = irt_dat_ori,
  family = brmsfamily("bernoulli"),
```

```
  prior = prior_1pl,
  control = list(adapt_delta = 0.95, max_treedepth = 15),
  cores = 4,
  chains = 4,
  iter = 2000,
  warmup = 1000,
  seed = 12345,
  backend = "cmdstanr",
  threads = threading(2),
  file = "../models/irt_ori.rds"
)
```

```
item_ori_1pl <- ranef(irt_ori)$word


item_ori_1pl[, , "Intercept"] %>%
  as_tibble() %>%
  rownames_to_column() %>%
  rename(item = "rowname") %>%
  mutate(item = reorder(item, Estimate)) %>%
  ggplot(aes(item, Estimate, ymin = Q2.5, ymax = Q97.5)) +
    geom_pointrange() +
    coord_flip() +
    labs(x = "Item Number")
```



Step 1: Rough screening

In original paper, authors fit a global BIRT for 379 items, and then calculate in/out fit for each iteration, getting the mode as the proxy of in/out fit of BIRT. Then they go for frequentist screening by applying 0.7/1.3 rule. For two reasons I don't like the solution: 1. This is the main reason - it's impossible to replicate this in my own computer. I tried to reduce the chain to 4, and iteration to 1000, with random sampling for 500 from 1179, still takes 1 hour to run and the model can't converge. It's too expensive. 2. I don't think MCMC/Bayesian approach here add any information for the rough screening. The major advantage for BIRT is that it estimates uncertainty by providing posterior instead of point estimate, while the posterior is completely missing in the paper. If we refer to BIRT tutorial papers (e.g., Bürkner, 2021; Zhang, 2025), people usually use "visual qualitative" solution, by spotting poor items if there is any with really wide posterior or really off estimates. While if the final goal is to use a point estimate to do rough screening, I can use frequentist IRT to do this naturally. I'll put in this way - in/out fit tells you how bad the model did by giving a point estimate; while in BIRT you can directly tell from the item difficulty posterior, the wider the poorer, if we want to directly use the mode/posterior mean.

As an affordable alternative, here I use TAM to fit a 1PL, and use 0.7/1.3 benchmark naturally.

```
wide_dat <- irt_dat %>%
  select(subjID, word, score) %>%
  distinct(subjID, word, .keep_all = TRUE) %>%
  pivot_wider(names_from = word, values_from = score) %>%
  column_to_rownames("subjID")

resp <- as.matrix(wide_dat)

cat(sprintf("Matrix ready: %d Persons x %d Items\n", nrow(resp), ncol(resp)))
```

```
## Matrix ready: 1190 Persons x 379 Items
```

```
item_ids <- colnames(resp)


mod_1pl <- tam.mml(resp, irtmodel = "1PL", verbose = FALSE)
fit_stats <- tam.fit(mod_1pl)
```

```
## Item fit calculation based on 15 simulations
## |**********|
## |----------|
```

```
res_1pl <- fit_stats$itemfit %>%
  as_tibble() %>%
  mutate(item_id = item_ids) %>%
  select(item_id, Infit_MSQ = Infit, Outfit_MSQ = Outfit)

good_fit_items <- res_1pl %>%
  filter(Infit_MSQ > 0.7 & Infit_MSQ < 1.3) %>%
  filter(Outfit_MSQ > 0.7 & Outfit_MSQ < 1.3)

cat(sprintf("1PL Filter (0.7 < MSQ < 1.3): %d items passed\n", nrow(good_fit_items)))
```

```
## 1PL Filter (0.7 < MSQ < 1.3): 212 items passed
```

We now use less than 10 seconds to fit a rough screening model, and have 211 items passed. The number is very similar to 212 in the original paper (where they use global BIRT + manually in/out fit for every iteration). We can check whether they provide similar information as well - 93% of them are the same. As a demo I won't go through the 7% difference in detail - and engineeringly, I assume these outliers will not be selected in final 90 anyway.

```
my_list <- good_fit_items$item_id

target_list <- readRDS("../original/fit_selected_items.rds") # 212 item list from original paper

common_items <- intersect(my_list, target_list)

jaccard <- length(common_items) / length(union(my_list, target_list))
jaccard
```

```
## [1] 0.9360731
```

Step2:

```
my_list <- good_fit_items$item_id


irt_dat_selected <- irt_dat %>%
  filter(word %in% my_list)
```

Now I follow the original paper's solution, to fit 1pl and 2pl for selected items. For convience, I change the setting for quicker fitting, it results in higher rhat and ess, but since this is a demo I'll ignore it.

```
irt1 <- brm(
  formula = formula_irt1,
  data = irt_dat_selected,
  family = brmsfamily("bernoulli"),
   prior = prior_1pl,
   init = 0,
   control = list(adapt_delta = 0.95, max_treedepth = 15),
   cores = 4,
   chains = 4,
   iter = 2000,
   warmup = 1000,
   seed = 1234,
   backend = "cmdstanr",
   threads = threading(2),
   file = "../models/irt1.rds" ) # runs approximately 50 mins

irt1 <- add_criterion(
  irt1,
  criterion = c("loo"),
  cores = 8,
  pointwise = TRUE,
  ndraws = 1000
```

```
)

saveRDS(irt1, "../models/irt1.rds")
```

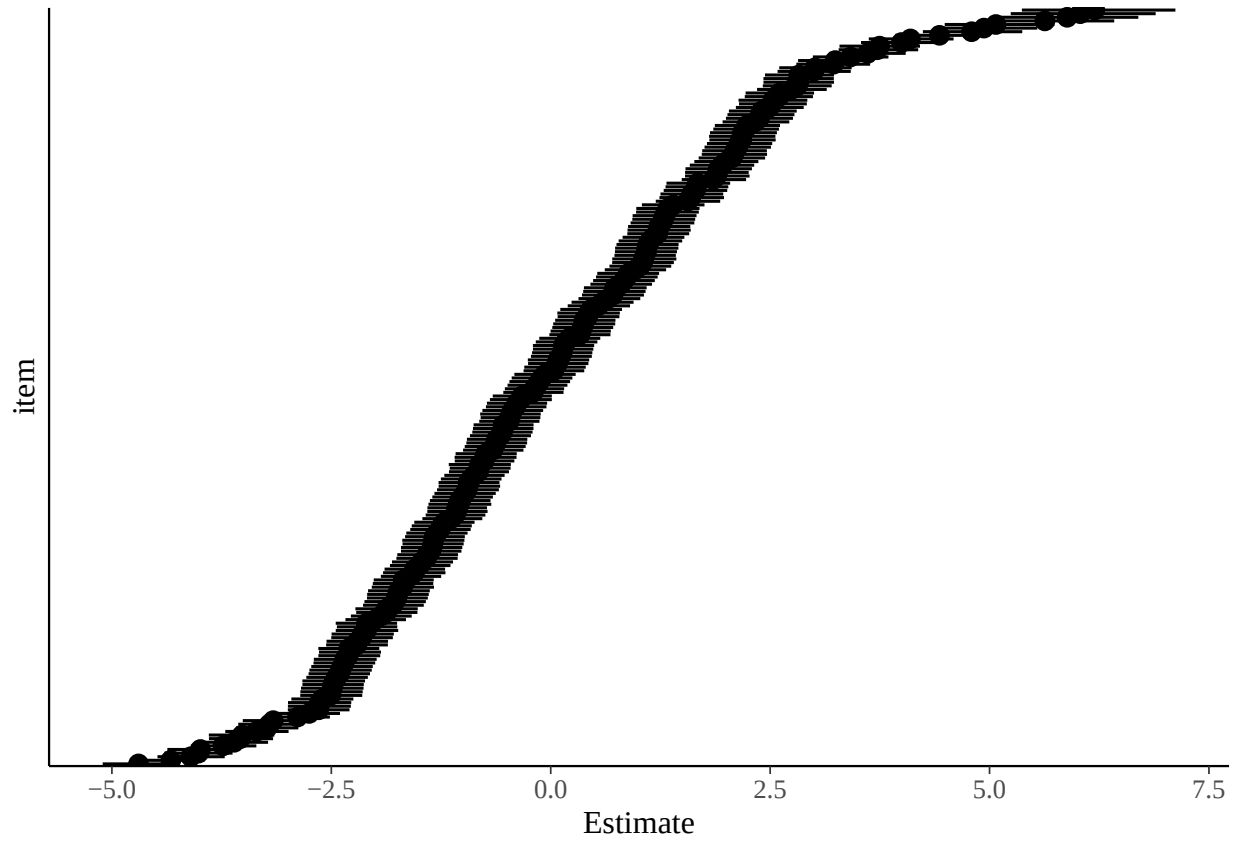
```
irt2 <- brm(
  formula = formula_irt2,
  data = irt_dat_selected,
  family = brmsfamily("bernoulli" , link = "logit"),
   prior = prior_2pl,
   init = 0,
   control = list(adapt_delta = 0.95, max_treedepth = 15),
   cores = 4,
   chains = 4,
   iter = 2000,
   warmup = 1000,
   seed = 1234,
   backend = "cmdstanr",
   threads = threading(2),
   file = "../models/irt2.rds" ) # runs approximately 2 hours

irt2 <- add_criterion(
  irt2,
  criterion = c("loo"),
  cores = 8,
  pointwise = TRUE,
  ndraws = 1000
)

saveRDS(irt1, "/Users/chi/Desktop/previc-optimization-demo/models/rt2.rds")
```

global model description

For the two global models, we can see from 1pl, we now have a set of items with difficulty range from -5 ~ 7, and for difficulty items (around 6), the posterior is relatively wide, indicating a potential uncertain fit. This is common, as if there are really hard word and there is few children can answer it correctly, the model have no enough information to have a robust estimate. For 2pl, we can see the discrimination ranges from 1~3. And this also results in 2pl way outperform 1pl (see loo compare), as the item pool overall is not rasch enough (recall rasch mdoel set discrimination at 1 by default.)

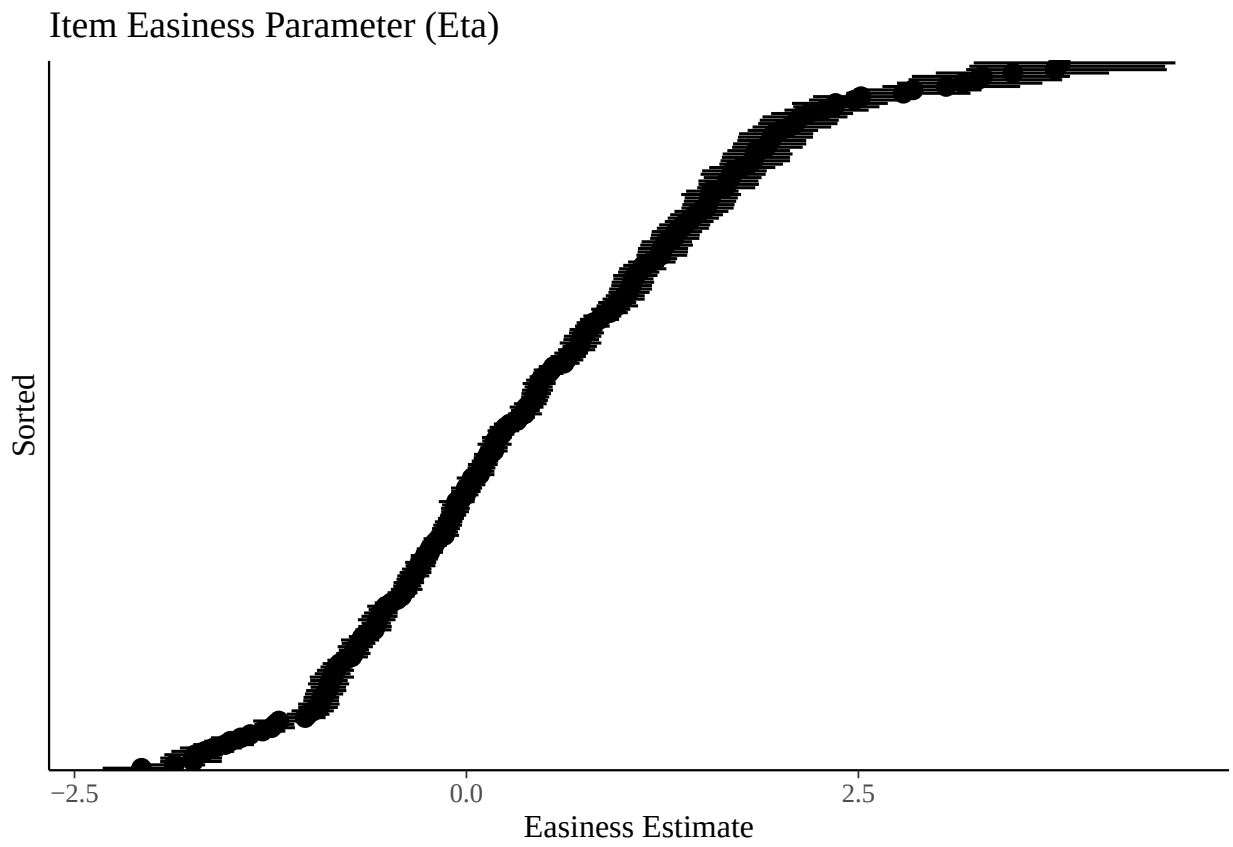
```
item_1pl <- ranef(irt1)$word


item_1pl[, , "Intercept"] %>%
  as_tibble() %>%
  rownames_to_column() %>%
  rename(item = "rowname") %>%
  mutate(item = reorder(item, Estimate)) %>%
  ggplot(aes(item, Estimate, ymin = Q2.5, ymax = Q97.5)) +
    geom_pointrange() +
    coord_flip() +
    theme(
      axis.text.y = element_blank(),
```

```
    axis.ticks.y = element_blank()
  )
```

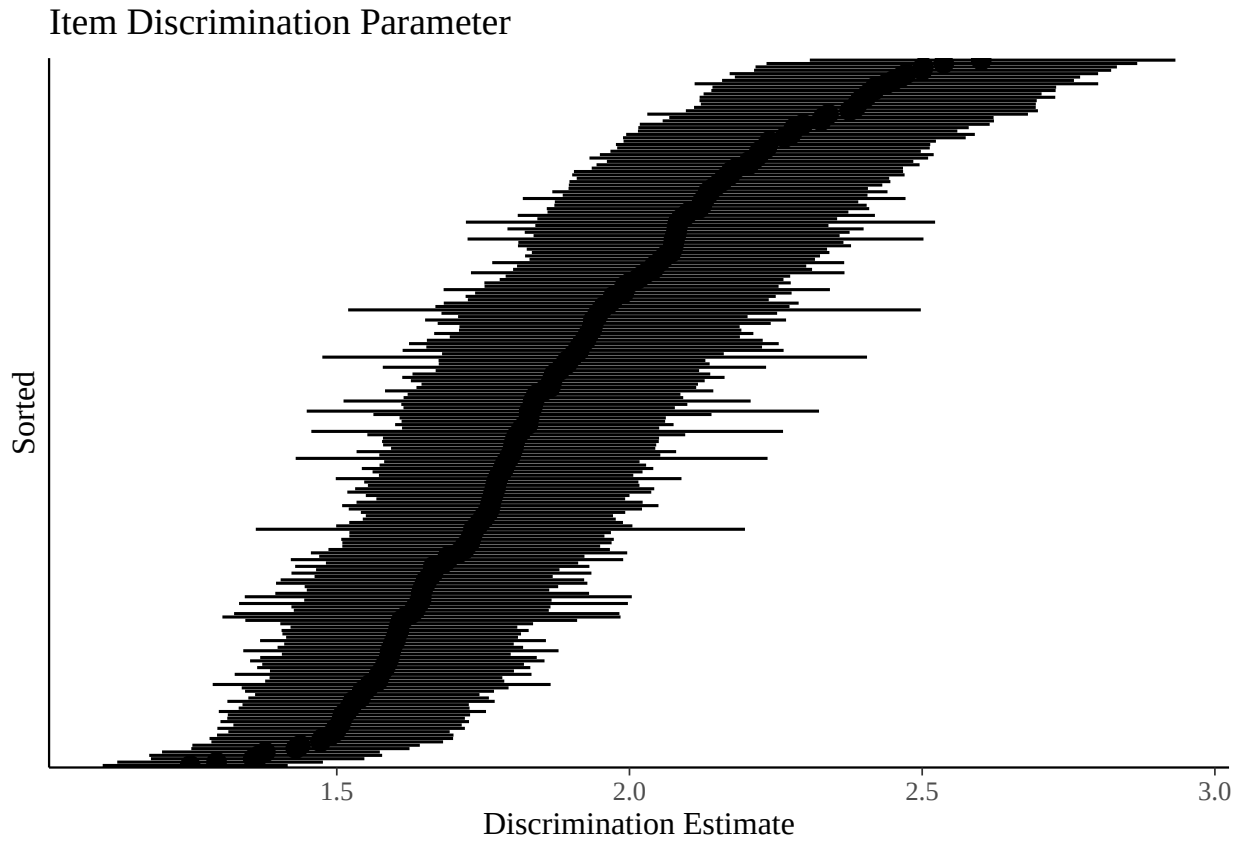



```
item_pars <- coef(irt2)$word

eta_df <- item_pars[, , "eta_Intercept"] %>%
  as_tibble(rownames = NA) %>%
  rownames_to_column(var = "item") %>%

  arrange(Estimate) %>%
  mutate(item = factor(item, levels = item)) %>%
  select(item, Estimate, Q2.5, Q97.5)


alpha_df <- item_pars[, , "logalpha_Intercept"] %>%
  exp() %>%
  as_tibble(rownames = NA) %>%
  rownames_to_column(var = "item") %>%
  arrange(Estimate) %>%
  mutate(item = factor(item, levels = item)) %>%
  select(item, Estimate, Q2.5, Q97.5)

p1 <- ggplot(eta_df, aes(x = item, y = Estimate, ymin = Q2.5, ymax = Q97.5)) +
  geom_pointrange() +
  coord_flip() +
```

```
  labs(
    x = "Sorted",
    y = "Easiness Estimate",
    title = "Item Easiness Parameter (Eta)"
  ) + theme(
      axis.text.y = element_blank(),
      axis.ticks.y = element_blank()
    )


p2 <- ggplot(alpha_df, aes(x = item, y = Estimate, ymin = Q2.5, ymax = Q97.5)) +
  geom_pointrange() +
  coord_flip() +
  labs(
    x = "Sorted",
    y = "Discrimination Estimate",
    title = "Item Discrimination Parameter"
  ) + theme(
      axis.text.y = element_blank(),
      axis.ticks.y = element_blank()
    )

p1
```



```
p2
```




```
loo(irt1,irt2)
```

```
## Output of model 'irt1':
##
## Computed from 1000 by 249900 log-likelihood matrix.
##
##          Estimate    SE
## elpd_loo -92596.6 276.1
## p_loo      1370.6   5.2
## looic    185193.1 552.2
## ------
## MCSE of elpd_loo is 1.2.
## MCSE and ESS estimates assume MCMC draws (r_eff in [0.5, 1.6]).
##
## All Pareto k estimates are good (k < 0.67).
## See help('pareto-k-diagnostic') for details.
##
## Output of model 'irt2':
##
## Computed from 1000 by 249900 log-likelihood matrix.
##
##          Estimate    SE
## elpd_loo -92095.3 275.7
```

```
## p_loo      1541.8   6.7
## looic    184190.6 551.3
## ------
## MCSE of elpd_loo is 1.3.
## MCSE and ESS estimates assume MCMC draws (r_eff in [0.5, 1.5]).
##
## All Pareto k estimates are good (k < 0.67).
## See help('pareto-k-diagnostic') for details.
##
## Model comparisons:
##      elpd_diff se_diff
## irt2    0.0       0.0
## irt1 -501.2      31.5
```

Step 3: Cost function

In the original paper, authors have a cost function assign to each items as the standard deviation for item difficulties was multiplied by – 1/3, Infit values by – 4, Outfit values by – 2 and modification indices by – 1/100. Basically, it is an aggregation of 3 considerations: i. difficulty (spacing); ii. fit stats (how good is the item?) iii. modification indices (how rasch is the item?).

Again, in this cost function, the uncertainty of items is ignored. Though items with significant uncertainty always being misfit, thus fit stats can reflect this, as we already conduct BIRT, I'll use width of posterior instead (estimating fit stats in BIRT is too expensive for me.) Plus, for MI, I'll use logalpha (discrimination) instead. I'll assign a penalty function to items depends on the distance to 1 (desirable Rasch discrimination).

And most importantly, authors use simulated annealing, as the spacing function is relative, it depends on different item list. This is fancinating, yet expensive and I believe is an overkill. We can easily turn this stochastic process to more linear calculation. For example, we'll expecting an Rasch assessment covering difficulty range from -3 ~ 3 (by convention), I'll expand this a bit based on the actual difficulty range in the global 1pl model (-4 ~ 5) as we have wider range items to choose, and these items still have quite stable estimate (for items, e.g., higher than 5, the posterior becomes oddly wide that we don't mind give up them). For n items we'd like to select, we assign a Target difficulty by spliting the whole range into n parts. Then we have a non-changeable spacing reference, namely difficulty to target.

Thus our new cost function can write as (DTT: difficulty to target; wp: width penalty; dp: disc penalty; For convenient I'll normalise each value):

$$C(i,k) = \lambda_1 \cdot \underbrace{\|\eta_i - \tau_k\|}_{\text{DTT}} + \lambda_2 \cdot \underbrace{\|\text{Width}_i\|}_{\text{WP}} + \lambda_3 \cdot \underbrace{\|\alpha_i - 1\|}_{\text{DP}}$$

In the original paper, authors use the algorithm to randomly have 5 best sets of items for each iterations. But my strategy is like more linear matching, as for each item the cost value is unchanged, thus each lambda setting will end up with one best item list. We can also consider to use different lambda settings to have competing item set, of course. Here we can set 1,1,1, reflecting that we equally focus on the three elements; or alternatively we can also do 3,2,3 or 3,1,3, as we already prescreening the item fit, which can reflect items' certainty, that makes WP less important. I'll do 3,2,3 as an example demo, but there is nothing wrong to try other settings if with proper reasoning.

```
pars_diff <- coef(irt1)$word[, , "Intercept"] %>%
  as.data.frame() %>%
  rownames_to_column("item") %>%
  transmute(
    item = item,
```

```
    eta = Estimate,
    width = Q97.5 - Q2.5
  )


pars_disc <- coef(irt2)$word[, , "logalpha_Intercept"] %>%
  as.data.frame() %>%
  rownames_to_column("item") %>%
  transmute(
    item = item,
    alpha = exp(Estimate)
  )

full_item_df <- pars_diff %>%
  inner_join(pars_disc, by = "item")
```

```
w_min <- min(full_item_df$width)
w_max <- max(full_item_df$width)
full_item_df$n_pw <- (full_item_df$width - w_min) / (w_max - w_min)

raw_pdisc <- abs(full_item_df$alpha - 1)
d_min <- min(raw_pdisc)
d_max <- max(raw_pdisc)
full_item_df$n_pdisc <- (raw_pdisc - d_min) / (d_max - d_min)
```

#Step 3: automatic item selection

Now our goal becomes select n items from 211, that i) we hope it is rasch enough, that 2pl doesn't outperform 1pl; ii) we hope these n items can be ideally distributed (according to cost function value competing). Naturally, we can use greedy matching, by assign the best item according to cost function value to each matching point. While this approach might have issue in order - if item t become the best for both point i and j, then i will be selected for the first occurring point. To make the selection process safe and replicable, we thus use Hungarian Algorithm instead.This method solves for the global optimum, minimizing the aggregate cost across all item slots simultaneously.

```
#n_list <- c(70, 80, 90, 100)

#run_selection_and_modeling <- function(target_n, full_item_df, irt_dat, base_irt1, base_irt2) {
  #targets <- seq(-4, 5, length.out = target_n)


  #n_items <- nrow(full_item_df)
  #dist_mat_raw <- abs(outer(full_item_df$eta, targets, "-")) #DTT
  #dist_mat_norm <- (dist_mat_raw - min(dist_mat_raw)) / (max(dist_mat_raw) - min(dist_mat_raw))

  #mat_pw    <- matrix(full_item_df$n_pw, nrow = n_items, ncol = target_n, byrow = FALSE) # PW
  #mat_pdisc <- matrix(full_item_df$n_pdisc, nrow = n_items, ncol = target_n, byrow = FALSE) #PD

  # 3:2:3
  #cost_matrix <- (3 * dist_mat_norm) + (2 * mat_pw) + (3 * mat_pdisc)

 # Hungarian algorithm
  #res <- lp.transport(
```

```
    #cost.mat = cost_matrix,
    #direction = "min",
    #row.signs = rep("<=", n_items), row.rhs = rep(1, n_items),
    #col.signs = rep("=", target_n), col.rhs = rep(1, target_n)
  #)

#if(res$status != 0) stop("Optimization failed!")

  #selected_indices <- which(rowSums(res$solution) > 0.5)
  #selected_items <- full_item_df$item[selected_indices]

  #dat_subset <- irt_dat %>% filter(word %in% selected_items)

# --- 1PL ---
  #file_name_1pl <- sprintf("../models/optimization/irt1_%d.rds", target_n)

  #fit_1pl <- update(
    #base_irt1,
    #newdata = dat_subset,
    #chains = 4, cores = 4,
    #iter = 2000, warmup = 1000,
    #backend = "cmdstanr",
    #file = NULL
  #)
  #fit_1pl <- add_criterion(fit_1pl, "loo", ndraws = 1000, cores = 8)
  #saveRDS(fit_1pl, file_name_1pl)

  # --- 2PL ---
  #file_name_2pl <- sprintf("../models/optimization/irt2_%d.rds", target_n)

  #fit_2pl <- update(
    #base_irt2,
    #newdata = dat_subset,
    #chains = 4, cores = 4,
    #iter = 2000, warmup = 1000,
    #backend = "cmdstanr",
    #file = NULL
  #)
  #fit_2pl <- add_criterion(fit_2pl, "loo", ndraws = 1000, cores = 4)
  #saveRDS(fit_2pl, file_name_2pl)

#}
```

```
#results_df <- map_dfr(n_list, function(n) {

  #run_selection_and_modeling(
   # target_n = n,
  #  full_item_df = full_item_df,
   # irt_dat = irt_dat,
  #  base_irt1 = irt1,
   # base_irt2 = irt2
  #)
#})
```

```
#print(results_df)
```

```
irt1_100 <- readRDS("../models/optimization/irt1_100.rds")
irt2_100  <- readRDS("../models/optimization/irt2_100.rds")
irt1_90 <- readRDS("../models/optimization/irt1_90.rds")
irt2_90  <- readRDS("../models/optimization/irt2_90.rds")
irt1_80 <- readRDS("../models/optimization/irt1_80.rds")
irt2_80  <- readRDS("../models/optimization/irt2_80.rds")
irt1_70 <- readRDS("../models/optimization/irt1_70.rds")
irt2_70  <- readRDS("../models/optimization/irt2_70.rds")
```

Item Size Result

For convenience, I only run item size for 70, 80, 90, 100. The result is quite different with the original paper, where the maximum item size that not violate Rasch setting is 90; but in my setting, even when select 100 items the rasch structure is still acceptable. Here, for comparasion purpose I choose to continue with size 90. Strictly speaking, here I need a DIF analysis to make sure all my items works well, I'll simply skip this for convenience. (Also I'm not so sure why the DIF model in original paper not including sex in fixed effect to rule out global group difference like this: 1 + sex + (0+ sex | word) + (1 | subjID), but I assume the result would be the same.)

```
loo_compare (irt1_70, irt2_70)
```

```
##         elpd_diff se_diff
## irt1_70  0.0       0.0
## irt2_70 -1.1       1.7
```

```
loo_compare (irt1_80, irt2_80)
```

```
##         elpd_diff se_diff
## irt2_80  0.0       0.0
## irt1_80 -1.7       2.2
```

```
loo_compare (irt1_90, irt2_90)
```

```
##         elpd_diff se_diff
## irt1_90  0.0       0.0
## irt2_90 -0.1       3.1
```

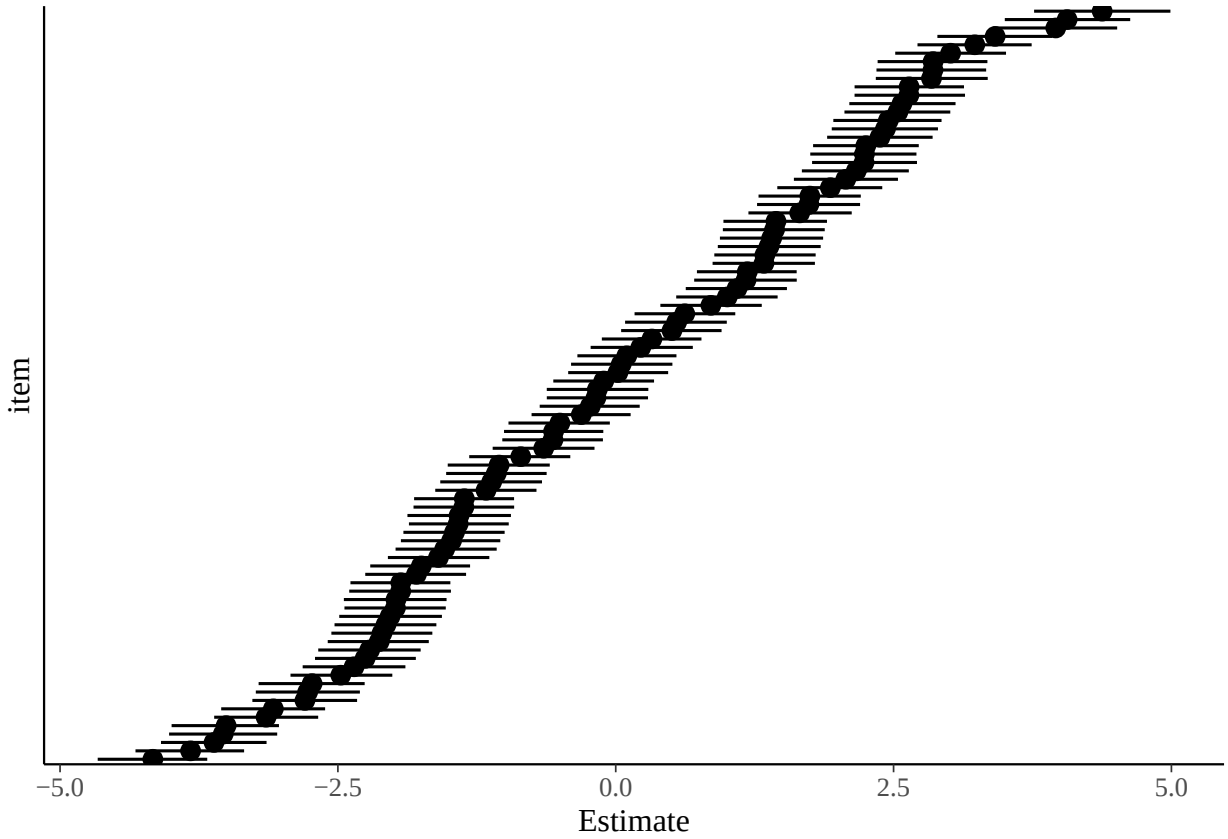
```
loo_compare (irt1_100, irt2_100)
```

```
##          elpd_diff se_diff
## irt2_100  0.0       0.0
## irt1_100 -3.5       3.8
```

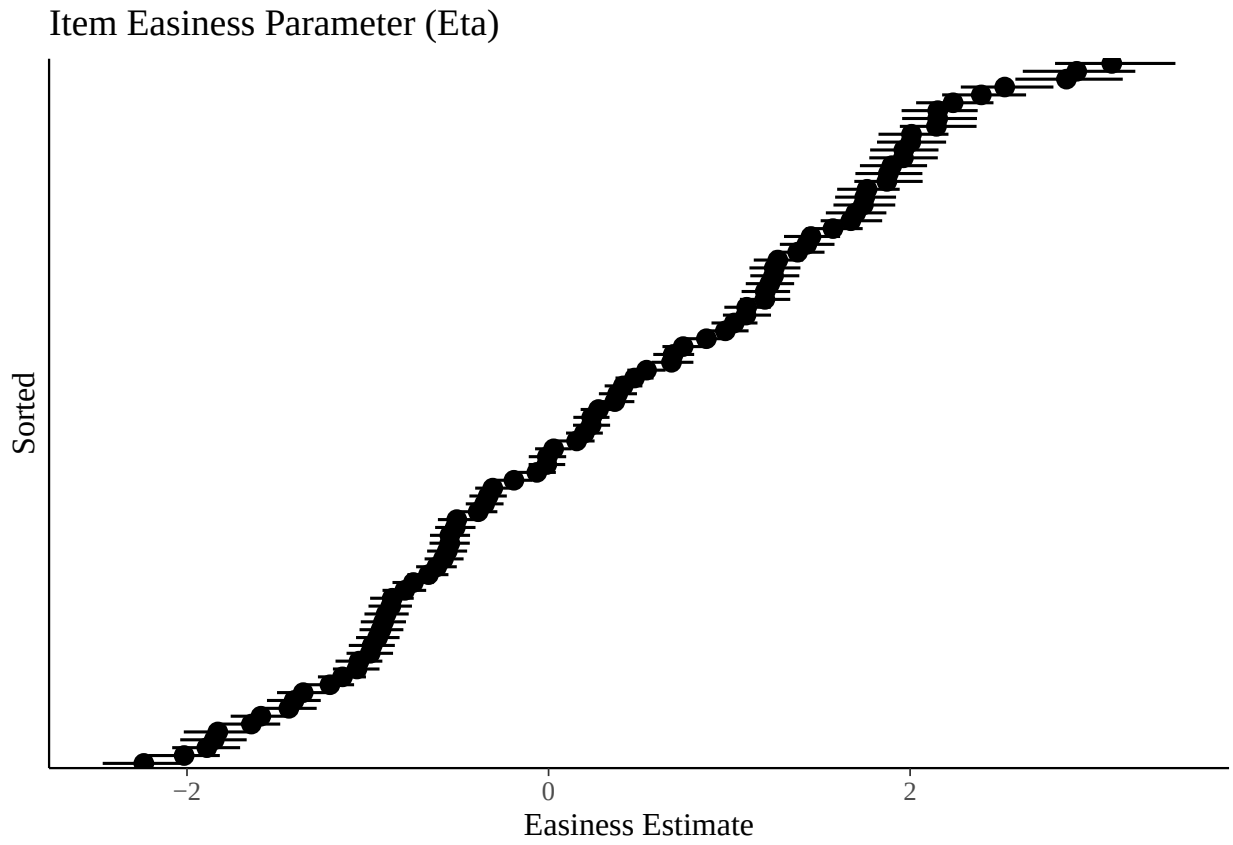
Step 4: result screening

We can see from the 1pl plot, that our model has a quite satisfying evenly distributed structure, with the 3 most difficult items being slightly off. This may be because I was overly optimistic about more challenging items (so change -4~5 to -4~4.5 might help). Or if I continue with this, I might manually delete the three outliers.In 2pl plot, it shows we now have items with similar discrimination from 1.5 ~ 1.7; and we still not yet pick all good items.

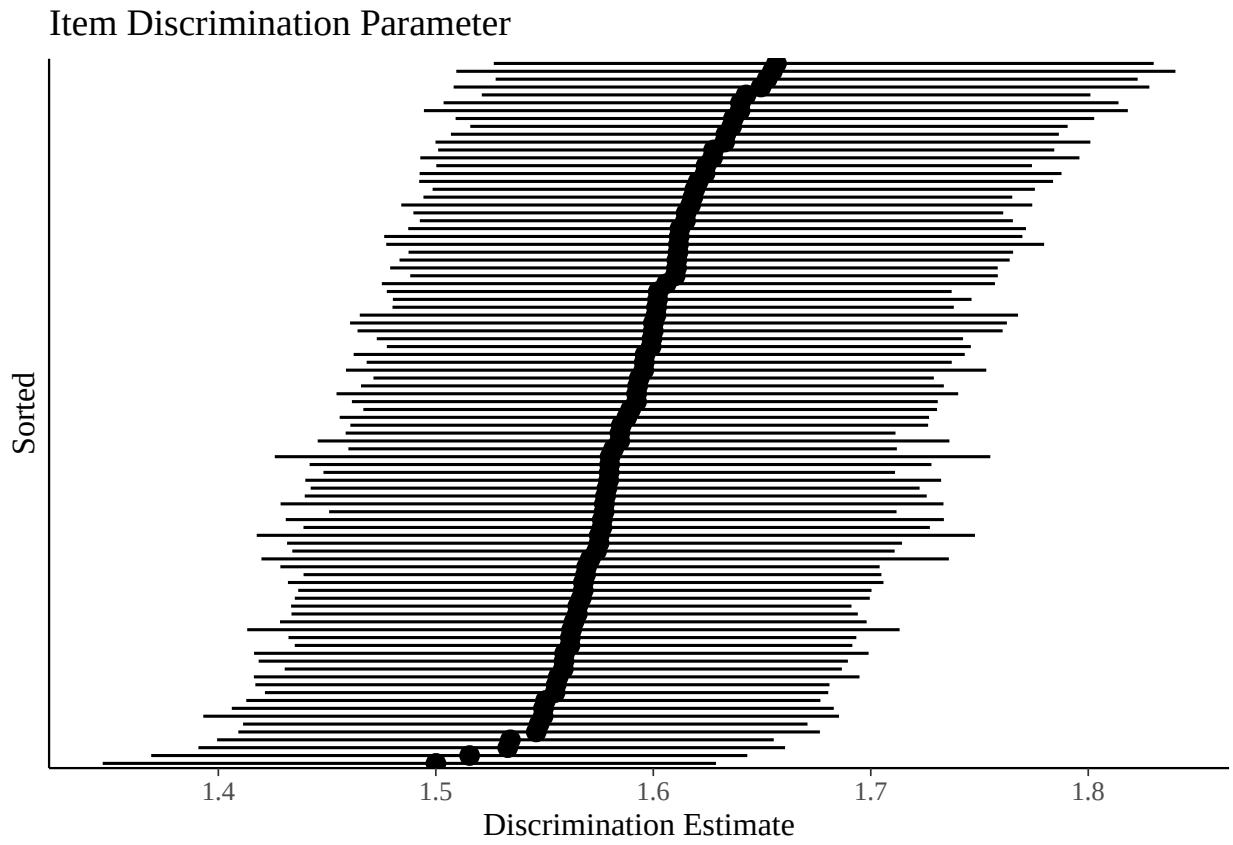
```
item_1pl_90 <- ranef(irt1_90)$word


item_1pl_90[, , "Intercept"] %>%
  as_tibble() %>%
  rownames_to_column() %>%
  rename(item = "rowname") %>%
  mutate(item = reorder(item, Estimate)) %>%
  ggplot(aes(item, Estimate, ymin = Q2.5, ymax = Q97.5)) +
    geom_pointrange() +
    coord_flip() +
    theme(
      axis.text.y = element_blank(),
      axis.ticks.y = element_blank()
    )
```



```r
item_pars_90 <- coef(irt2_90)$word

eta_df_90 <- item_pars_90[, , "eta_Intercept"] %>%
  as_tibble(rownames = NA) %>%
  rownames_to_column(var = "item") %>%

  arrange(Estimate) %>%
  mutate(item = factor(item, levels = item)) %>%
  select(item, Estimate, Q2.5, Q97.5)


alpha_df_90 <- item_pars_90[, , "logalpha_Intercept"] %>%
  exp() %>%
  as_tibble(rownames = NA) %>%
  rownames_to_column(var = "item") %>%
  arrange(Estimate) %>%
  mutate(item = factor(item, levels = item)) %>%
  select(item, Estimate, Q2.5, Q97.5)

p1 <- ggplot(eta_df_90, aes(x = item, y = Estimate, ymin = Q2.5, ymax = Q97.5)) +
  geom_pointrange() +
  coord_flip() +
  labs(
    x = "Sorted",
    y = "Easiness Estimate",
    title = "Item Easiness Parameter (Eta)"
  ) + theme(
      axis.text.y = element_blank(),
      axis.ticks.y = element_blank()
    )


p2 <- ggplot(alpha_df_90, aes(x = item, y = Estimate, ymin = Q2.5, ymax = Q97.5)) +
  geom_pointrange() +
  coord_flip() +
  labs(
    x = "Sorted",
    y = "Discrimination Estimate",
    title = "Item Discrimination Parameter"
  ) + theme(
      axis.text.y = element_blank(),
      axis.ticks.y = element_blank()
    )

p1
```




```
p2
```



Benchmark comparasion

But whether this is better than the original choice? I calculate the final difficulty spacing value of original final model and my 1pl_90 model, finding that my solution sacrifice around 10% of spacing performance. Yet I still take it as a win, given that this is an alternative 'affordable' solution that can run on a personal computer within one day; yet it can potentially contains items more than 90, thus we may regard this as sacrificing an optimal spacing solution in favour of including more good items.

```
calc_brms_spacing <- function(brms_model) {

  item_params <- ranef(brms_model)$word[, , "Intercept"][, "Estimate"]


  easinesses <- sort(item_params)


  nn_dists <- diff(easinesses)


  spacing_score <- -1 * sd(nn_dists) / 3

  return(spacing_score)
}


spacing_ori <- calc_brms_spacing(irt_ori)
```

```
spacing_90 <- calc_brms_spacing(irt1_90)

spacing_90
```

```
## [1] -0.03214383
```

```
spacing_ori
```

```
## [1] -0.02911598
```

Items selection comparasion

I also compare the original 89 choice and my 90 choice. There is of 43 in common, and the different item selection shows why my solution lose on the performance. For dropping items, the mean alpha was 1.92 while I chose alternative with alpha at 1.56. This means the original paper's cost function focus more on item fit and spacing, while don't mind select high discrimination item in it. This can also explain why 90 is the maximum for original solution, as it already include acute disc items. While my solution punish too much on bad disc items - this might not make sense because normally we believe items with low disc might be a problem, but high disc sometimes though being considered against rasch, but is a signal for good item. In this sense the lambda setting might give disc = 1 too much credit. And my manually setting difficulty range might be too wide to include bad items. I dropped 45 items with 1.54 difficulty 0.38 width, which are stable items at mid range, while I choose items with 0.43 width and 2.32 difficulty, which is more extreme distributed and unstable. Again, the lambda and difficulty range setting can be reset the fix this. I'm thinking doing (-4, 4) and (5, 4, 2) can help with mimicing a similar solution for original paper.

```
items_ori <- rownames(ranef(irt_ori)$word)
items_90  <- rownames(ranef(irt1_90)$word)

common    <- intersect(items_ori, items_90)
dropped   <- setdiff(items_ori, items_90)
added     <- setdiff(items_90, items_ori)

comparison_df <- full_item_df %>%
  mutate(
    status = case_when(
      item %in% common  ~ "Common (Both)",
      item %in% dropped ~ "Dropped (Original Only)",
      item %in% added   ~ "Added (My Algo)",
      TRUE              ~ "Not Selected"
    )
  ) %>%
  filter(status != "Not Selected")


summary_stats <- comparison_df %>%
  group_by(status) %>%
  summarise(
    Count = n(),
    Mean_Alpha = mean(alpha, na.rm=TRUE),
    Mean_Width = mean(width, na.rm=TRUE),
    SD_Eta     = sd(eta, na.rm=TRUE)
```

```
  )

print(summary_stats)
```

```
## # A tibble: 3 x 5
##   status                  Count Mean_Alpha Mean_Width SD_Eta
##   <chr>                   <int>      <dbl>      <dbl>  <dbl>
## 1 Added (My Algo)            47       1.56      0.433   2.32
## 2 Common (Both)              43       1.70      0.415   2.25
## 3 Dropped (Original Only)    45       1.92      0.385   1.55
```