Item Response Probit Model

Summer 2024

This report writes about the DIHOPIT model with the random effect deleted. It is also called the item response probit model in psychometric. I plan to write down the current information and try to integrate the psychometric and linear regression terms. This approach may help us further investigate how we can use this model to solve the urban-rural bias that exists in the current PCA approach.

Mathematical Formula

The DIHOPIT model without random effect can be written as

$$\begin{aligned} &y_i^* = X_i'\beta + \varepsilon_i \quad i = 1, \ldots, N \\ &\varepsilon_i \sim N(0,1) \end{aligned}$$

The observation mechanism for each indicator variable $a = 1, \ldots, A$:

$$\begin{aligned} y_i^a = 0 \text{ if } \quad -\infty < y_i^* \leq \tau^a \\ y_i^a = 1 \text{ if } \quad \tau^a < y_i^* \leq +\infty \end{aligned}$$

where τ^a is the threshold for the a^{th} indicator variable.

$$\begin{aligned} P(y_i^a = 1|X_i) &= P(X_i'\beta + \varepsilon_i > \tau^a) \\ &= P(\varepsilon_i > \tau^a - X_i'\beta) \\ &= 1 - P(\varepsilon_i \leq \tau^a - X_i'\beta) \\ &= 1 - \Phi(\tau^a - X_i'\beta) \\ &= \Phi(X_i'\beta - \tau^a) \end{aligned}$$

The same model has another form of expression in terms of

$$\begin{aligned} P(y_i^a = 1|\theta_i, a_a, b_a) = \Phi(a_a(\theta_i - b_a)) \quad a = 1, \ldots, A \\ \theta_i \sim N(0,1) \quad i = 1, \ldots, N \end{aligned}$$

where

θ_i is the latent trait (ability) of respondent i

y_i^a is the response of individual i to item a

$\Phi(\cdot)$ is the standard normal cumulative distribution function (probit function)

a_a is the discrimination parameter for item a

b_a is the difficulty parameter for item a

Simulation Result

```r
# Set seed for reproducibility
set.seed(123)
```

```r
# Step 1: Set up parameters
N <- 1000  # number of respondents
P <- 3     # number of predictor variables
A <- 5     # number of items/indicators

# Arbitrarily set beta and tau
beta <- c(0.8, -0.5, 1.2)
tau <- c(-1.5, -0.5, 0, 0.5, 1.5)

# Step 2: Generate latent variables
X <- matrix(rnorm(N*P), nrow=N, ncol=P)
colnames(X) <- paste0("X", 1:P)
epsilon <- rnorm(N)
y_star <- X %*% beta + epsilon

# Step 3: Check if y_star passes each threshold tau
Y <- matrix(0, nrow=N, ncol=A)
colnames(Y) <- paste0("Y", 1:A)
for(a in 1:A) {
  Y[,a] <- as.integer(y_star > tau[a])
}

# Create a data frame
df <- data.frame(
  id = 1:N,
  X,
  y_star = y_star,
  Y
)

# Print first few rows of the data frame
cat("\nFirst 5 rows of the data frame:\n")
```

```
##
## First 5 rows of the data frame:
```

```r
print(head(df, 5))
```

```
##   id          X1          X2         X3     y_star Y1 Y2 Y3 Y4 Y5
## 1  1 -0.56047565 -0.99579872 -0.5116037 -0.7147131  1  0  0  0  0
## 2  2 -0.23017749 -1.03995504  0.2369379  0.2924039  1  1  1  0  0
## 3  3  1.55870831 -0.01798024 -0.5415892 -0.8421155  1  0  0  0  0
## 4  4  0.07050839 -0.13217513  1.2192276  0.8882829  1  1  1  1  0
## 5  5  0.12928774 -2.54934277  0.1741359  4.1855549  1  1  1  1  1
```

Simulation by Rstan

```r
# Extract results
posterior <- extract(fit)
estimated_beta <- colMeans(posterior$beta)
estimated_tau <- colMeans(posterior$tau)

# Calculate standard errors
se_beta <- apply(posterior$beta, 2, sd)
```

```r
se_tau <- apply(posterior$tau, 2, sd)

cat(
  "Estimated beta coefficients (shared across all items):\n",
  paste(round(estimated_beta, 4), collapse = ", "), "\n\n",
  "Standard errors for beta:\n",
  paste(round(se_beta, 4), collapse = ", "), "\n\n",
  "Estimated thresholds (tau) for each item:\n",
  paste(round(estimated_tau, 4), collapse = ", "), "\n\n",
  "Standard errors for tau:\n",
  paste(round(se_tau, 4), collapse = ", "), "\n\n",
  "True beta coefficients:\n",
  paste(round(beta, 4), collapse = ", "), "\n\n",
  "True thresholds:\n",
  paste(round(tau, 4), collapse = ", "), "\n",
  sep = ""
)
```

```
## Estimated beta coefficients (shared across all items):
## 0.8352, -0.5131, 1.339
##
## Standard errors for beta:
## 0.0317, 0.0268, 0.0395
##
## Estimated thresholds (tau) for each item:
## -1.5905, -0.5065, -0.0168, 0.5718, 1.6509
##
## Standard errors for tau:
## 0.0676, 0.0544, 0.052, 0.0552, 0.0689
##
## True beta coefficients:
## 0.8, -0.5, 1.2
##
## True thresholds:
## -1.5, -0.5, 0, 0.5, 1.5
```

Simulation by Maximum Likelihood Estimation

$$L(\boldsymbol{\beta}, \boldsymbol{\tau} | \mathbf{X}, \mathbf{Y}) = \sum_{j=1}^{M} \sum_{i=1}^{N} [Y_{ij} \log(p_{ij}) + (1 - Y_{ij}) \log(1 - p_{ij})]$$
$$\text{where } p_{ij} = \Phi(\mathbf{X}_i \boldsymbol{\beta} - \tau_j)$$

```r
# Prepare data
X <- as.matrix(df[, c("X1", "X2", "X3")])
Y <- as.matrix(df[, paste0("Y", 1:A)])

# Log-likelihood function
log_likelihood <- function(params, X, Y) {
  beta <- params[1:3]
  tau <- params[4:8]

  ll <- 0
  for (j in 1:ncol(Y)) {
```

```r
    p <- pnorm(X %*% beta - tau[j])
    ll <- ll + sum(dbinom(Y[,j], size = 1, prob = p, log = TRUE))
  }
  return(-ll)  # Return negative log-likelihood for minimization
}

# Initial parameter values
initial_params <- c(rep(0, 3), rep(0, 5))  # 3 betas and 5 taus

# Optimize using optim
fit <- optim(par = initial_params,
             fn = log_likelihood,
             X = X,
             Y = Y,
             method = "BFGS",
             hessian = TRUE)

# Extract results
beta_estimates <- fit$par[1:3]
tau_estimates <- fit$par[4:8]

# Calculate standard errors
se <- sqrt(diag(solve(fit$hessian)))
beta_se <- se[1:3]
tau_se <- se[4:8]

# Print results
cat("Beta estimates:\n")
```

```
## Beta estimates:
```

```r
print(beta_estimates)
```

```
## [1]  0.8336015 -0.5117845  1.3357161
```

```r
cat("\nBeta standard errors:\n")
```

```
##
## Beta standard errors:
```

```r
print(beta_se)
```

```
## [1] 0.03150957 0.02716907 0.03942711
```

```r
cat("\nTau estimates:\n")
```

```
##
## Tau estimates:
```

```r
print(tau_estimates)
```

```
## [1] -1.58642369 -0.50527916 -0.01573883  0.56981898  1.64832564
```

```r
cat("\nTau standard errors:\n")
```

```
##
## Tau standard errors:
```

```r
print(tau_se)
```

```
## [1] 0.06760760 0.05356487 0.05189363 0.05399222 0.06861807
```

```r
# Compare with true values
cat("\nTrue beta coefficients:\n")
```

```
##
## True beta coefficients:
```

```r
print(beta)
```

```
## [1]  0.8 -0.5  1.2
```

```r
cat("\nTrue thresholds:\n")
```

```
##
## True thresholds:
```

```r
print(tau)
```

```
## [1] -1.5 -0.5  0.0  0.5  1.5
```