Package ‘ohoegdm’

February 24, 2022

Title  Ordinal Higher-Order Exploratory General Diagnostic Model for Polytomous Data

Version  0.1.0


URL  https://github.com/tmsalab/ohoegdm,
     https://tmsalab.github.io/ohoegdm/

BugReports  https://github.com/tmsalab/ohoegdm/issues

License  GPL (>= 2)

Encoding  UTF-8

RoxygenNote  7.1.2

LinkingTo  Rcpp, RcppArmadillo

Imports  Rcpp

Suggests  edmdata, covr

NeedsCompilation  yes

Author  Steven Andrew Culpepper [aut, cph]
         (<https://orcid.org/0000-0003-4226-6176>),
         James Joseph Balamuta [aut, cre, cph]
         (<https://orcid.org/0000-0003-2826-8458>)

Maintainer  James Joseph Balamuta <balamut2@illinois.edu>

Repository  CRAN

Date/Publication  2022-02-24 09:30:02 UTC

R topics documented:

  GenerateAtable ..........................................................  2
gen_bijectionvector ......................................................  2
ohoegdm .................................................................  3
sim_slcm ...............................................................  7
GenerateATable

Generate tables that store different design elements

Description

Each table provides a "cache" of pre-computed values.

Usage

GenerateATable(nClass, K, M, order)

Arguments

nClass  Number of Attribute Classes
K  Number of Attributes
M  Number of Responses
order  Highest interaction order to consider. Default model-specified k.

Details

This is an internal function briefly used to simulate data and, thus, has been exported into R as well as documented. Output from this function can change in future versions.

Value

Return a list containing the table caches for different parameters

gen_bijectionvector

Generate a vector to map polytomous vector to integers

Description

Converts class into a bijection to integers

Usage

gen_bijectionvector(K, M)

Arguments

K  Number of Attributes
M  Number of Response Categories

Value

Return a $K$-length vector containing the bijection vector.
**Description**

Performs the Gibbs sampling routine for an ordinal higher-order EGDM.

**Usage**

```r
ohoegdm(
  y,  # Ordinal Item Matrix
  k,  # Dimension to estimate for Q matrix
  m = 2,  # Number of Item Categories. Default is 2 matching the binary case.
  order = k,  # Highest interaction order to consider. Default model-specified k.
  sd_mh = 0.4,  # Metropolis-Hastings standard deviation tuning parameter.
  burnin = 1000L,  # Amount of Draws to Burn
  chain_length = 10000L,  # Number of Iterations for chain.
  l0 = c(1, rep(100, sum(choose(k, seq_len(order))))),  # Spike parameter. Default 1 for intercept and 100 coefficients
  l1 = c(1, rep(1, sum(choose(k, seq_len(order))))),  # Slab parameter. Default 1 for all values.
  m0 = 0,  # Additional tuning parameters.
  bq = 1
)
```

**Arguments**

- `y`: Ordinal Item Matrix
- `k`: Dimension to estimate for Q matrix
- `m`: Number of Item Categories. Default is 2 matching the binary case.
- `order`: Highest interaction order to consider. Default model-specified `k`.
- `sd_mh`: Metropolis-Hastings standard deviation tuning parameter.
- `burnin`: Amount of Draws to Burn
- `chain_length`: Number of Iterations for chain.
- `l0`: Spike parameter. Default 1 for intercept and 100 coefficients
- `l1`: Slab parameter. Default 1 for all values.
- `m0`, `bq`: Additional tuning parameters.

**Details**

The `estimates` list contains the mean information from the sampling procedure. Meanwhile, the `chain` list contains full MCMC values. Moreover, the `details` list provides information regarding the estimation call. Lastly, the `recovery` list stores values that can be used when assessing the method under a simulation study.
A `ohoegdm` object containing four named lists:

- **estimates**: Averaged chain iterations
  - `thetas`: Average theta coefficients
  - `betas`: Average beta coefficients
  - `deltas`: Average activeness of coefficients
  - `classes`: Average class membership
  - `m2lls`: Average negative two times log-likelihood
  - `omegas`: Average omega
  - `kappas`: Average category threshold parameter
  - `taus`: Average $K$-vectors of factor intercept
  - `lambdas`: Average $K$-vectors of factor loadings
  - `guessing`: Average guessing item parameter
  - `slipping`: Average slipping item parameter
  - `QS`: Average activeness of Q matrix entries
- **chain**: Chain iterations from the underlying C++ routine.
  - `thetas`: Theta coefficients iterations
  - `betas`: Beta coefficients iterations
  - `deltas`: Activeness of coefficients iterations
  - `classes`: Class membership iterations
  - `m2lls`: Negative two times log-likelihood iterations
  - `omegas`: Omega iterations
  - `kappas`: Category threshold parameter iterations
  - `taus`: $K$-vector of factor intercept iterations
  - `lambdas`: $K$-vector of factor loadings iterations
  - `guessing`: Guessing item parameter iterations
  - `slipping`: Slipping item parameter iterations
- **details**: Properties used to estimate the model
  - `n`: Number of Subjects
  - `j`: Number of Items
  - `k`: Number of Traits
  - `m`: Number of Item Categories.
  - `order`: Highest interaction order to consider. Default model-specified $k$.
  - `sd_mh`: Metropolis-Hastings standard deviation tuning parameter.
  - `l0`: Spike parameter
  - `l1`: Slab parameter
  - `m0, bq`: Additional tuning parameters
  - `burnin`: Number of Iterations to discard
  - `chain_length`: Number of Iterations to keep
  - `runtime`: Elapsed time algorithm run time in the C++ code.
- **recovery**: Assess recovery metrics under a simulation study.
  - `Q_item_encoded`: Per-iteration item encodings from Q matrix.
  - `MHsum`: Average acceptance from metropolis hastings sampler
Examples

# Simulation Study
if (requireNamespace("edmdata", quietly = TRUE)) {

# Q and Beta Design ----

# Obtain the full K3 Q matrix from edmdata
data("qmatrix_oracle_k3_j20", package = "edmdata")
Q_full = qmatrix_oracle_k3_j20

# Retain only a subset of the original Q matrix
removal_idx = -c(3, 5, 9, 12, 15, 18, 19, 20)
Q = Q_full[removal_idx, ]

# Construct the beta matrix by-hand
beta = matrix(0, 20, ncol = 8)

# Intercept
beta[, 1] = 1

# Main effects
beta[1:3, 2] = 1.5
beta[4:6, 3] = 1.5
beta[7:9, 5] = 1.5

# Setup two-way effects
beta[10, c(2, 3)] = 1
beta[11, c(3, 4)] = 1
beta[12, c(2, 5)] = 1
beta[13, c(2, 5)] = 1
beta[14, c(2, 6)] = 1
beta[15, c(3, 5)] = 1
beta[16, c(3, 5)] = 1
beta[17, c(3, 7)] = 1

# Setup three-way effects
beta[18:20, c(2, 3, 5)] = 0.75

# Decrease the number of Beta rows
beta = beta[removal_idx, ]

# Construct additional parameters for data simulation
Kappa = matrix(c(0, 1, 2), nrow = 20, ncol = 3, byrow = TRUE) # mkappa
lambda = c(0.25, 1.5, -1.25) # mlambdas
tau = c(0, -0.5, 0.5) # mtaus

# Simulation conditions ----
N = 100 # Number of Observations
J = nrow(beta) # Number of Items
M = 4 # Number of Response Categories
Malha = 2  # Number of Classes
K = ncol(Q)  # Number of Attributes
order = K  # Highest interaction to consider
sd_mtheta = 1  # Standard deviation for theta values

# Simulate data ----

# Generate theta values
theta = rnorm(N, sd = sdmtheta)

# Generate alphas
Zs = matrix(1, N, 1) %*% tau +
    matrix(theta, N, 1) %*% lambda +
    matrix(rnorm(N * K), N, K)
Alphas = 1 * (Zs > 0)

vv = gen_bijectionvector(K, Malha)
CLS = Alphas %*% vv
Atab = GenerateAtable(Malpha ^ K, K, Malha, order)$Atable

# Simulate item-level data
Ysim = sim_slcm(N, J, M, Malha ^ K, CLs, Atab, beta, Kappa)

# Establish chain properties
# Standard Deviation of MH. Set depending on sample size.
# If sample size is:
# - small, allow for larger standard deviation
# - large, allow for smaller standard deviation.
sd_mh = .4
burnin = 50  # Set for demonstration purposes, increase to at least 5,000 in practice.
chain_length = 100  # Set for demonstration purposes, increase to at least 40,000 in practice.

# Setup spike-slab parameters
l0s = c(1, rep(100, Malha ^ K - 1))
l1s = c(1, rep(1, Malha ^ K - 1))

my_model = ohoegdm::ohoegdm(
    y = Ysim,
    k = K,
    m = M,
    order = order,
    l0 = l0s,
    l1 = l1s,
    m0 = 0,
    bq = 1,
    sd_mh = sd_mh,
    burnin = burnin,
    chain_length = chain_length
)
}
`sim_slcm`  

**Simulate Ordinal Item Data from a Sparse Latent Class Model**

**Description**

Simulate Ordinal Item Data from a Sparse Latent Class Model

**Usage**

```r
sim_slcm(N, J, M, nClass, CLASS, Atable, BETA, KAPPA)
```

**Arguments**

- `N` Number of Observations
- `J` Number of Items
- `M` Number of Item Categories (2, 3, ..., M)
- `nClass` Number of Latent Classes
- `CLASS` A vector of N observations containing the class ID of the subject.
- `Atable` A matrix of dimensions $M^K \times M^O$ containing the attribute classes in bijection-form. Note, $O$ refers to the model’s highest interaction order.
- `BETA` A matrix of dimensions $J \times M^K$ containing the coefficients of the reparameterized $\beta$ matrix.
- `KAPPA` A matrix of dimensions $J \times M$ containing the category threshold parameters

**Value**

An ordinal item matrix of dimensions $N \times J$ with $M$ response levels.

**See Also**

- `ohoegdm`
Index

gen_bijectionvector, 2
GenerateAtable, 2

ohoeqdm, 3, 7

sim_slcm, 7