Package 'Qval'

January 20, 2025

Type Package

Title The Q-Matrix Validation Methods Framework

Version 1.1.1

Date 2025-01-10

Author Haijiang Qin [aut, cre, cph] (<https://orcid.org/0009-0000-6721-5653>),
Lei Guo [aut, cph] (<https://orcid.org/0000-0002-8273-3587>)

Maintainer Haijiang Qin <haijiang133@outlook.com>

Description
Provide a variety of Q-matrix validation methods for the generalized cognitive diagnosis models, including the method based on the generalized deterministic input, noisy, and gate model (G-DINA) by de la Torre (2011) <DOI:10.1007/s11336-011-9207-7> discrimination index (the GDI method) by de la Torre and Chiu (2016) <DOI:10.1007/s11336-015-9467-8>, the stepwise Wald test method (the Wald method) by Ma and de la Torre (2020) <DOI:10.1111/bmsp.12156>, the Hull method by N
jera et al. (2021) <DOI:10.1111/bmsp.12228>, the multiple logistic regression-based Q-matrix validation method (the MLR-B method) by Tu et al. (2022) <DOI:10.3758/s13428-022-01880-x>, the beta method based on signal detection theory by Li and Chen (2024) <DOI:10.1111/bmsp.12371> and Q-matrix validation based on relative fit index by Chen et la. (2013) <DOI:10.1111/j.1745-3984.2012.00185.x>. Different research methods and iterative procedures during Q-matrix validating are available.

License GPL-3

Depends R (>= 4.1.0)

Imports glmnet, GDINA, plyr, nloptr, MASS, Matrix, parallel, Rcpp,
stats

LinkingTo Rcpp

RoxygenNote 7.3.2

Encoding UTF-8

NeedsCompilation yes

Collate 'beta.R' 'CDM.R' 'convex.R' 'cov.R' 'fit.R' 'GDI.R' 'Hull.R'
'MLR-B.R' 'MLR.R' 'MLRlasso.R' 'Mmatrix.R' 'P.R' 'Pattern.R'
'priority.R' 'PVAF.R' 'plot.Hull.R' 'Qvalindex.R' 'R2.R'

'RcppExports.R' 'Rmatrix.R' 'sim.data.R' 'sim.MQ.R' 'sim.Q.R'
'utils.R' 'validation.R' 'Wald.R' 'Wald.test.R' 'zzz.R'

Repository CRAN

URL https://haijiangqin.com/Qval/

Date/Publication 2025-01-09 16:20:06 UTC

Contents

`CDM` *Parameter estimation for cognitive diagnosis models (CDMs) by MMLE/EM or MMLE/BM algorithm.*

Description

A function to estimate parameters for cognitive diagnosis models by MMLE/EM (de la Torre, 2009; de la Torre, 2011) or MMLE/BM (Ma & Jiang, 2020) algorithm.The function imports various functions from the `GDINA` package, parameter estimation for Cognitive Diagnostic Models was performed and extended. The `CDM` function not only accomplishes parameter estimation for most commonly used models (`GDINA`, `DINA`, `DINO`, `ACDM`, `LLM`, or `rRUM`) but also facilitates parameter estimation for the `LCDM` model (Henson, Templin, & Willse, 2008; Tu et al., 2022). Furthermore, it incorporates Bayes modal estimation (BM; Ma & Jiang, 2020) to obtain more reliable estimation results, especially in small sample sizes. The monotonic constraints are able to be satisfied.

Usage

```
CDM(
  Y,
  Q,
  model = "GDINA",
  method = "EM",
  mono.constraint = TRUE,
  maxitr = 2000,
  verbose = 1
)
```

Arguments

`Y`	A required `N` × `I` matrix or data.frame consisting of the responses of `N` individuals to × `I` items. Missing values need to be coded as `NA`.
`Q`	A required binary `I` × `K` containing the attributes not required or required, 0 or 1, to master the items. The `i`th row of the matrix is a binary indicator vector indicating which attributes are not required (coded by 0) and which attributes are required (coded by 1) to master item `i`.
`model`	Type of model to be fitted; can be `"GDINA"`, `"LCDM"`, `"DINA"`, `"DINO"`, `"ACDM"`, `"LLM"`, or `"rRUM"`. Default = `"GDINA"`.
`method`	Type of mtehod to estimate CDMs' parameters; one out of `"EM"`, `"BM"`. Default = `"EM"` However, `"BM"` is only avaible when `method` = `"GDINA"`.
`mono.constraint`	Logical indicating whether monotonicity constraints should be fulfilled in estimation. Default = `TRUE`.
`maxitr`	A vector for each item or nonzero category, or a scalar which will be used for all items to specify the maximum number of EM or BM cycles allowed. Default = `2000`.
`verbose`	Can be `0`, `1` or `2`, indicating to print no information, information for current iteration, or information for all iterations. Default = `1`.

Details

CDMs are statistical models that fully integrates cognitive structure variables, which define the response probability of subjects on questions by assuming the mechanism of action between attributes. In the dichotomous test, this probability is the probability of answering correctly. According to the specificity or generality of CDM assumptions, it can be divided into reduced CDM and saturated CDM.

Reduced CDMs possess special and strong assumptions about the mechanisms of attribute interactions, leading to clear interactions between attributes. Representative reduced models include the Deterministic Input, Noisy and Gate (DINA) model (Haertel, 1989; Junker & Sijtsma, 2001; de la Torre & Douglas, 2004), the Deterministic Input, Noisy or Gate (DINO) model (Templin & Henson, 2006), and the Additive Cognitive Diagnosis Model (A-CDM; de la Torre, 2011), the reduced Reparametrized Unified Model (r-RUM; Hartz, 2002), among others. Compared to reduced models, saturated models do not have strict assumptions about the mechanisms of attribute interactions. When appropriate constraints are applied, they can be transformed into various reduced models (Henson et al., 2008; de la Torre, 2011), such as the Log-Linear Cognitive Diagnosis Model (LCDM; Henson et al., 2009) and the general Deterministic Input, Noisy and Gate model (G-DINA; de la Torre, 2011).

The LCDM (Log-Linear Cognitive Diagnosis Model) is a saturated CDM fully proposed within the framework of cognitive diagnosis. Unlike simplified models that only discuss the main effects of attributes, it also considers the interactions between attributes, thus having more generalized assumptions about attributes. Its definition of the probability of correct response is as follows:

$$P(X_{pi}=1|\alpha_l)=\frac{\exp(\lambda_{i0}+\lambda_i^T\mathbf{h}(\mathbf{q_i},\alpha_\mathbf{l}))}{1+\exp(\lambda_{i0}+\lambda_i^T\mathbf{h}(\mathbf{q_i},\alpha_\mathbf{l}))}$$

$$\lambda_i^T\mathbf{h}(\mathbf{q_i},\alpha_\mathbf{l})=\lambda_{i0}+\sum_{k=1}^{K^*}\lambda_{ik}\alpha_{lk}q_{ik}+\sum_{k=1}^{K^*-1}\sum_{k'=k+1}^{K^*}\lambda_{ik}\lambda_{ik'}\alpha_{lk}\alpha_{lk'}q_{ik}q_{ik'}+\cdots+\lambda_{12\cdots K^*}\prod_{k=1}^{K^*}\alpha_{lk}q_{ik}$$

Where, $P(X_{pi}=1|\alpha_l)$ represents the probability of a subject with attribute mastery pattern α_l, where $l=1,2,\cdots,L$ and $L=2^{K^*}$, correctly answering item i. Here, K^* denotes the number of attributes in the collapsed q-vector, λ_{i0} is the intercept parameter, and $\lambda_i=(\lambda_{i1},\lambda_{i2},\cdots,\lambda_{i12},\cdots,\lambda_{i12\cdots K^*})$ represents the effect vector of the attributes. Specifically, λ_{ik} is the main effect of attribute k, $\lambda_{ikk'}$ is the interaction effect between attributes k and k', and $\lambda_{j12\cdots K}$ represents the interaction effect of all attributes.

The general Deterministic Input, Noisy and Gate model (G-DINA), proposed by de la Torre (2011), is a saturated model that offers three types of link functions: identity link, log link, and logit link, which are defined as follows:

$$P(X_{pi}=1|\alpha_l)=\delta_{i0}+\sum_{k=1}^{K^*}\delta_{ik}\alpha_{lk}+\sum_{k=1}^{K^*-1}\sum_{k'=k+1}^{K^*}\delta_{ik}\delta_{ik'}\alpha_{lk}\alpha_{lk'}+\cdots+\delta_{12\cdots K^*}\prod_{k=1}^{K^*}\alpha_{lk}$$

$$log(P(X_{pi}=1|\alpha_l))=v_{i0}+\sum_{k=1}^{K^*}v_{ik}\alpha_{lk}+\sum_{k=1}^{K^*-1}\sum_{k'=k+1}^{K^*}v_{ik}v_{ik'}\alpha_{lk}\alpha_{lk'}+\cdots+v_{12\cdots K^*}\prod_{k=1}^{K^*}\alpha_{lk}$$

$$logit(P(X_{pi}=1|\alpha_l))=\lambda_{i0}+\sum_{k=1}^{K^*}\lambda_{ik}\alpha_{lk}+\sum_{k=1}^{K^*-1}\sum_{k'=k+1}^{K^*}\lambda_{ik}\lambda_{ik'}\alpha_{lk}\alpha_{lk'}+\cdots+\lambda_{12\cdots K^*}\prod_{k=1}^{K^*}\alpha_{lk}$$

Where δ_{i0}, v_{i0}, and λ_{i0} are the intercept parameters for the three link functions, respectively; δ_{ik}, v_{ik}, and λ_{ik} are the main effect parameters of α_{lk} for the three link functions, respectively; $\delta_{ikk'}$, $v_{ikk'}$, and $\lambda_{ikk'}$ are the interaction effect parameters between α_{lk} and $\alpha_{lk'}$ for the three link functions, respectively; and $\delta_{i12\cdots K^*}$, $v_{i12\cdots K^*}$, and $\lambda_{i12\cdots K^*}$ are the interaction effect parameters of $\alpha_{l1}\cdots\alpha_{lK^*}$ for the three link functions, respectively. It can be observed that when the logit link is adopted, the G-DINA model is equivalent to the LCDM model.

Specifically, the A-CDM can be formulated as:

$$P(X_{pi}=1|\alpha_l)=\delta_{i0}+\sum_{k=1}^{K^*}\delta_{ik}\alpha_{lk}$$

The RRUM, can be written as:

$$log(P(X_{pi}=1|\alpha_l))=\lambda_{i0}+\sum_{k=1}^{K^*}\lambda_{ik}\alpha_{lk}$$

The item response function for LLM can be given by:

$$logit(P(X_{pi} = 1|\alpha_l)) = \lambda_{i0} + \sum_{k=1}^{K^*} \lambda_{ik}\alpha_{lk}$$

In the DINA model, every item is characterized by two key parameters: guessing (g) and slip (s). Within the traditional framework of DINA model parameterization, a latent variable η, specific to individual p who has the attribute mastery pattern α_l and item i, is defined as follows:

$$\eta_{li} = \prod_{k=1}^{K} \alpha_{lk}^{q_{ik}}$$

If individual p who has the attribute mastery pattern α_l has acquired every attribute required by item i, η_{pi} is given a value of 1. If not, η_{pi} is set to 0. The DINA model's item response function can be concisely formulated as such:

$$P(X_{pi} = 1|\alpha_l) = (1 - s_j)^{\eta_{li}} g_j^{(1-\eta_{li})} = \delta_{i0} + \delta_{i12\cdots K} \prod_{k=1}^{K^*} \alpha_{lk}$$

In contrast to the DINA model, the DINO model suggests that an individual can correctly respond to an item if they have mastered at least one of the item's measured attributes. Additionally, like the DINA model, the DINO model also accounts for parameters related to guessing and slipping. Therefore, the main difference between DINO and DINA lies in their respective η_{pi} formulations. The DINO model can be given by:

$$\eta_{li} = 1 - \prod_{k=1}^{K} (1 - \alpha_{lk})^{q_{lk}}$$

Value

An object of class `CDM.obj` is a `list` containing the following components:

`analysis.obj`	An `GDINA` object gained from `GDINA` package or an `list` after BM algorithm, depending on which estimation is used.
`alpha`	Individuals' attribute parameters caculated by EAP method (Huebner & Wang, 2011)
`P.alpha.Xi`	Individual posterior
`alpha.P`	Individuals' marginal mastery probabilities matrix (Tu et al., 2022)
`P.alpha`	Attribute prior weights for calculating marginalized likelihood in the last iteration
`model.fit`	Some basic model-fit indeces, including $Deviance$, $npar$, AIC, BIC

Author(s)

Haijiang Qin <Haijiang133@outlook.com>

References

de la Torre, J. (2009). DINA Model and Parameter Estimation: A Didactic. Journal of Educational and Behavioral Statistics, 34(1), 115-130. DOI: 10.3102/1076998607309474.

de la Torre, J., & Douglas, J. A. (2004). Higher-order latent trait models for cognitive diagnosis. Psychometrika, 69(3), 333-353. DOI: 10.1007/BF02295640.

de la Torre, J. (2011). The Generalized DINA Model Framework. Psychometrika, 76(2), 179-199. DOI: 10.1007/s11336-011-9207-7.

Haertel, E. H. (1989). Using restricted latent class models to map the skill structure of achievement items. Journal of Educational Measurement, 26(4), 301-323. DOI: 10.1111/j.1745-3984.1989.tb00336.x.

Hartz, S. M. (2002). A Bayesian framework for the unified model for assessing cognitive abilities: Blending theory with practicality (Unpublished doctoral dissertation). University of Illinois at Urbana-Champaign.

Henson, R. A., Templin, J. L., & Willse, J. T. (2008). Defining a Family of Cognitive Diagnosis Models Using Log-Linear Models with Latent Variables. Psychometrika, 74(2), 191-210. DOI: 10.1007/s11336-008-9089-5.

Huebner, A., & Wang, C. (2011). A note on comparing examinee classification methods for cognitive diagnosis models. Educational and Psychological Measurement, 71, 407-419. DOI: 10.1177/0013164410388832.

Junker, B. W., & Sijtsma, K. (2001). Cognitive assessment models with few assumptions, and connections with nonparametric item response theory. Applied Psychological Measurement, 25(3), 258-272. DOI: 10.1177/01466210122032064.

Ma, W., & Jiang, Z. (2020). Estimating Cognitive Diagnosis Models in Small Samples: Bayes Modal Estimation and Monotonic Constraints. Applied Psychological Measurement, 45(2), 95-111. DOI: 10.1177/0146621620977681.

Templin, J. L., & Henson, R. A. (2006). Measurement of psychological disorders using cognitive diagnosis models. Psychological methods, 11(3), 287-305. DOI: 10.1037/1082-989X.11.3.287.

Tu, D., Chiu, J., Ma, W., Wang, D., Cai, Y., & Ouyang, X. (2022). A multiple logistic regression-based (MLR-B) Q-matrix validation method for cognitive diagnosis models: A confirmatory approach. Behavior Research Methods. DOI: 10.3758/s13428-022-01880-x.

See Also

`validation`.

Examples

```
################################################################
#                          Example 1                           #
#            fit using MMLE/EM to fit the GDINA models         #
################################################################
set.seed(123)

library(Qval)

## generate Q-matrix and data to fit
K <- 5
```

```
I <- 30
example.Q <- sim.Q(K, I)
IQ <- list(
  P0 = runif(I, 0.0, 0.2),
  P1 = runif(I, 0.8, 1.0)
)
example.data <- sim.data(Q = example.Q, N = 500, IQ = IQ,
                         model = "GDINA", distribute = "horder")


## using MMLE/EM to fit GDINA model
example.CDM.obj <- CDM(example.data$dat, example.Q, model = "GDINA",
                       method = "EM", maxitr = 2000, verbose = 1)



################################################################
#                           Example 2                          #
#               fit using MMLE/BM to fit the DINA              #
################################################################
set.seed(123)

library(Qval)

## generate Q-matrix and data to fit
K <- 5
I <- 30
example.Q <- sim.Q(K, I)
IQ <- list(
  P0 = runif(I, 0.0, 0.2),
  P1 = runif(I, 0.8, 1.0)
)
example.data <- sim.data(Q = example.Q, N = 500, IQ = IQ,
                         model = "DINA", distribute = "horder")


## using MMLE/EM to fit GDINA model
example.CDM.obj <- CDM(example.data$dat, example.Q, model = "GDINA",
                       method = "BM", maxitr = 1000, verbose = 2)


################################################################
#                           Example 3                          #
#              fit using MMLE/EM to fit the ACDM               #
################################################################
set.seed(123)

library(Qval)

## generate Q-matrix and data to fit
K <- 5
I <- 30
example.Q <- sim.Q(K, I)
```

```
IQ <- list(
  P0 = runif(I, 0.0, 0.2),
  P1 = runif(I, 0.8, 1.0)
)
example.data <- sim.data(Q = example.Q, N = 500, IQ = IQ,
                         model = "ACDM", distribute = "horder")


## using MMLE/EM to fit GDINA model
example.CDM.obj <- CDM(example.data$dat, example.Q, model = "ACDM",
                       method = "EM", maxitr = 2000, verbose = 1)
```

fit *Calculate data fit indeces*

Description

Calculate relative fit indices (-2LL, AIC, BIC, CAIC, SABIC) and absolute fit indices (M_2 test) using the `testfit` function in the `GDINA` package.

Usage

```
fit(Y, Q, model = "GDINA")
```

Arguments

`Y`	A required `N` × `I` matrix or data.frame consisting of the responses of `N` individuals to `I` items. Missing values should be coded as NA.
`Q`	A required binary `I` × `K` matrix containing the attributes not required or required , coded as 0 or 1, to master the items. The `i`th row of the matrix is a binary indicator vector indicating which attributes are not required (coded as 0) and which attributes are required (coded as 1) to master item `i`.
`model`	Type of model to be fitted; can be `"GDINA"`, `"LCDM"`, `"DINA"`, `"DINO"`, `"ACDM"`, `"LLM"`, or `"rRUM"`. Default = `"GDINA"`.

Value

An object of class `list`. The list contains various fit indices:

`npar`	The number of parameters.
`-2LL`	The Deviance.
`AIC`	The Akaike information criterion.
`BIC`	The Bayesian information criterion.
`CAIC`	The consistent Akaike information criterion.

SABIC	The Sample size Adjusted BIC.
M2	A vector consisting of M_2 statistic, degrees of freedom, significance level, and $RMSEA_2$ (Liu, Tian, & Xin, 2016).
SRMSR	The standardized root mean squared residual (SRMSR; Ravand & Robitzsch, 2018).

Author(s)

Haijiang Qin <Haijiang133@outlook.com>

References

Khaldi, R., Chiheb, R., & Afa, A.E. (2018). Feed-forward and Recurrent Neural Networks for Time Series Forecasting: Comparative Study. In: Proceedings of the International Conference on Learning and Optimization Algorithms: Theory and Applications (LOPAL 18). Association for Computing Machinery, New York, NY, USA, Article 18, 1–6. DOI: 10.1145/3230905.3230946.

Liu, Y., Tian, W., & Xin, T. (2016). An application of M2 statistic to evaluate the fit of cognitive diagnostic models. Journal of Educational and Behavioral Statistics, 41, 3–26. DOI: 10.3102/1076998615621293.

Ravand, H., & Robitzsch, A. (2018). Cognitive diagnostic model of best choice: a study of reading comprehension. Educational Psychology, 38, 1255–1277. DOI: 10.1080/01443410.2018.1489524.

Examples

```
set.seed(123)

library(Qval)

## generate Q-matrix and data to fit
K <- 5
I <- 30
example.Q <- sim.Q(K, I)
IQ <- list(
  P0 = runif(I, 0.0, 0.2),
  P1 = runif(I, 0.8, 1.0)
)
example.data <- sim.data(Q = example.Q, N = 500, IQ = IQ, model = "GDINA", distribute = "horder")

## calculate fit indices
fit.indices <- fit(Y = example.data$dat, Q = example.Q, model = "GDINA")
print(fit.indices)
```

---

get.Mmatrix *Calculate* $\mathbf{M}$ *matrix*

Description

Calculate $\mathbf{M}$ matrix for stauted CDMs (de la Torre, 2011).

Usage

```
get.Mmatrix(K = NULL, pattern = NULL)
```

Arguments

`K`	The number of attributes. Can be NULL if `pattern` is passed to the function and is not NULL.
`pattern`	The knowledge state matrix containing all possible attribute mastery pattern. Can be gained from @seealso `attributepattern`. Also can be NULL if `K` is passed to the function and is not NULL.

Value

An object of class `matrix`.

Author(s)

Haijiang Qin <Haijiang133@outlook.com>

References

de la Torre, J. (2011). The Generalized DINA Model Framework. Psychometrika, 76(2), 179-199. DOI: 10.1007/s11336-011-9207-7.

Examples

```
library(Qval)

example.Mmatrix <-  get.Mmatrix(K = 5)
```

`get.priority` *Priority of Attribute*

Description

This function will provide the priorities of attributes for all items.

Usage

```
get.priority(Y = NULL, Q = NULL, CDM.obj = NULL, model = "GDINA")
```

Arguments

`Y`	A required `N` × `I` matrix or data.frame consisting of the responses of `N` individuals to `I` items. Missing values need to be coded as `NA`.
`Q`	A required binary `I` × `K` containing the attributes not required or required, 0 or 1, to master the items. The `i`th row of the matrix is a binary indicator vector indicating which attributes are not required (coded by 0) and which attributes are required (coded by 1) to master item `i`.
`CDM.obj`	An object of class `CDM.obj`. When it is not NULL, it enables rapid verification of the Q-matrix without the need for parameter estimation. @seealso `CDM`.
`model`	Type of model to fit; can be `"GDINA"`, `"LCDM"`, `"DINA"`, `"DINO"` , `"ACDM"`, `"LLM"`, or `"rRUM"`. Default = `"GDINA"`. @seealso `CDM`.

Details

The calculation of priorities is straightforward: the priority of an attribute is the regression coefficient obtained from a LASSO multinomial logistic regression, with the attribute as the independent variable and the response data from the subjects as the dependent variable. The formula is as follows:

$$\log[\frac{P(X_\pi = 1|\mathbf{\Lambda}_p)}{P(X_\pi = 0|\mathbf{\Lambda}_p)}] = logit[P(X_\pi = 1|\mathbf{\Lambda}_p)] = \beta_{i0} + \beta_{i1}\Lambda_{p1} + \ldots + \beta_{ik}\Lambda_{pk} + \ldots + \beta_{iK}\Lambda_{pK}$$

The LASSO loss function can be expressed as:

$$l_{lasso}(\mathbf{X}_i|\mathbf{\Lambda}) = l(\mathbf{X}_i|\mathbf{\Lambda}) - \lambda|\beta_i|$$

The priority for attribute i is defined as: $\mathbf{priority}_i = [\beta_{i1}, \ldots, \beta_{ik}, \ldots, \beta_{iK}]$

Value

A matrix containing all attribute priorities.

Examples

```
set.seed(123)
library(Qval)

## generate Q-matrix and data
K <- 5
I <- 20
IQ <- list(
  P0 = runif(I, 0.1, 0.3),
  P1 = runif(I, 0.7, 0.9)
)


Q <- sim.Q(K, I)
data <- sim.data(Q = Q, N = 500, IQ = IQ, model = "GDINA", distribute = "horder")
```

```
MQ <- sim.MQ(Q, 0.1)

CDM.obj <- CDM(data$dat, MQ)

priority <- get.priority(data$dat, Q, CDM.obj)
head(priority)
```

---

get.PVAF *Calculate $PVAF$*

Description

The function is able to caculate the proportion of variance accounted for ($PVAF$) for all items after fitting `CDM` or directly.

Usage

```
get.PVAF(Y = NULL, Q = NULL, CDM.obj = NULL, model = "GDINA")
```

Arguments

`Y`	A required `N` × `I` matrix or data.frame consisting of the responses of `N` individuals to `I` items. Missing values should be coded as `NA`.
`Q`	A required binary `I` × `K` matrix containing the attributes not required or required, coded as 0 or 1, to master the items. The `i`th row of the matrix is a binary indicator vector indicating which attributes are not required (coded as 0) and which attributes are required (coded as 1) to master item `i`.
`CDM.obj`	An object of class `CDM.obj`. Can can be NULL, but when it is not NULL, it enables rapid verification of the Q-matrix without the need for parameter estimation. @seealso `CDM`.
`model`	Type of model to be fitted; can be `"GDINA"`, `"LCDM"`, `"DINA"`, `"DINO"`, `"ACDM"`, `"LLM"`, or `"rRUM"`. Default = `"GDINA"`.

Details

The intrinsic essence of the GDI index (as denoted by ζ_2) is the weighted variance of all 2^{K*} attribute mastery patterns' probabilities of correctly responding to item i, which can be computed as:

$$\zeta^2 = \sum_{l=1}^{2^K} \pi_l (P(X_{pi} = 1|\alpha_l) - P_i^{mean})^2$$

where π_l represents the prior probability of mastery pattern l; $P_i^{mean} = \sum_{k=1}^{2^K} \pi_l P(X_{pi} = 1|\alpha_l)$ is the weighted average of the correct response probabilities across all attribute mastery patterns.

When the q-vector is correctly specified, the calculated ζ^2 should be maximized, indicating the maximum discrimination of the item.

Theoretically, ζ^2 is larger when $\mathbf{q}_i$ is either specified correctly or over-specified, unlike when $\mathbf{q}_i$ is under-specified, and that when $\mathbf{q}_i$ is over-specified, ζ^2 is larger than but close to the value of $\mathbf{q}_i$ when specified correctly. The value of ζ^2 continues to increase slightly as the number of over-specified attributes increases, until $\mathbf{q}_i$ becomes $\mathbf{q}_{i1:K}$. Thus, ζ^2/ζ^2_{max} is computed to indicate the proportion of variance accounted for by $\mathbf{q}_i$, called the $PVAF$.

Value

An object of class `matrix`, which consisted of $PVAF$ for each item and each possible attribute mastery pattern.

Author(s)

Haijiang Qin <Haijiang133@outlook.com>

References

de la Torre, J., & Chiu, C. Y. (2016). A General Method of Empirical Q-matrix Validation. Psychometrika, 81(2), 253-273. DOI: 10.1007/s11336-015-9467-8.

See Also

`validation`

Examples

```
library(Qval)

set.seed(123)

## generate Q-matrix and data
K <- 3
I <- 20
example.Q <- sim.Q(K, I)
IQ <- list(
  P0 = runif(I, 0.0, 0.2),
  P1 = runif(I, 0.8, 1.0)
)
example.data <- sim.data(Q = example.Q, N = 500, IQ = IQ, model = "GDINA", distribute = "horder")

## calculate PVAF directly
PVAF <-get.PVAF(Y = example.data$dat, Q = example.Q)
print(PVAF)

## caculate PVAF after fitting CDM
example.CDM.obj <- CDM(example.data$dat, example.Q, model="GDINA")
PVAF <-get.PVAF(CDM.obj = example.CDM.obj)
print(PVAF)
```

get.R2 *Calculate McFadden pseudo-R^2*

Description

The function is able to calculate the McFadden pseudo-R^2 (R^2) for all items after fitting `CDM` or directly.

Usage

```
get.R2(Y = NULL, Q = NULL, CDM.obj = NULL, model = "GDINA")
```

Arguments

`Y`	A required `N` × `I` matrix or data.frame consisting of the responses of `N` individuals to `I` items. Missing values should be coded as `NA`.
`Q`	A required binary `I` × `K` matrix containing the attributes not required or required, coded as 0 or 1, to master the items. The `i`th row of the matrix is a binary indicator vector indicating which attributes are not required (coded as 0) and which attributes are required (coded as 1) to master item `i`.
`CDM.obj`	An object of class `CDM.obj`. Can can be NULL, but when it is not NULL, it enables rapid verification of the Q-matrix without the need for parameter estimation. @seealso `CDM`.
`model`	Type of model to fit; can be `"GDINA"`, `"LCDM"`, `"DINA"`, `"DINO"`, `"ACDM"`, `"LLM"`, or `"rRUM"`. Default = `"GDINA"`.

Details

The McFadden pseudo-R^2 (McFadden in 1974) serves as a definitive model-fit index, quantifying the proportion of variance explained by the observed responses. Comparable to the squared multiple-correlation coefficient in linear statistical models, this coefficient of determination finds its application in logistic regression models. Specifically, in the context of the CDM, where probabilities of accurate item responses are predicted for each examinee, the McFadden pseudo-R^2 provides a metric to assess the alignment between these predictions and the actual responses observed. Its computation is straightforward, following the formula:

$$R_i^2 = 1 - \frac{\log(L_{im}}{\log(L_{i0})}$$

where $\log(L_{im}$ is the log-likelihood of the model, and $\log(L_{i0})$ is the log-likelihood of the null model. If there were N examinees taking a test comprising I items, then $\log(L_{im})$ would be computed as:

$$\log(L_{im}) = \sum_{p}^{N} \log \sum_{l=1}^{2^{K^*}} \pi(\alpha_l^* | X_p) P_i(\alpha_l^*)^{X_{pi}} (1 - P_i(\alpha_l^*))^{1-X_{pi}}$$

where $\pi(\alpha_l^*|X_p)$ is the posterior probability of examinee p with attribute profle α_l^* when their response vector is $\mathbf{X}_p$, and X_{pi} is examinee p's response to item i. Let X_i^{mean} be the average probability of correctly responding to item i across all N examinees; then $\log(L_{i0}$ could be computed as:

$$\log(L_{i0}) = \sum_{p}^{N} \log X_i^{mean X_{pi}} (1 - X_i^{mean})^{1-X_{pi}}$$

Value

An object of class `matrix`, which consisted of R^2 for each item and each possible attribute mastery pattern.

Author(s)

Haijiang Qin <Haijiang133@outlook.com>

References

McFadden, D. (1974). Conditional logit analysis of qualitative choice behavior. In P. Zarembka (Ed.), Frontiers in economics (pp.105–142). Academic Press.

Najera, P., Sorrel, M. A., de la Torre, J., & Abad, F. J. (2021). Balancing ft and parsimony to improve Q-matrix validation. British Journal of Mathematical and Statistical Psychology, 74, 110–130. DOI: 10.1111/bmsp.12228.

Qin, H., & Guo, L. (2023). Using machine learning to improve Q-matrix validation. Behavior Research Methods. DOI: 10.3758/s13428-023-02126-0.

See Also

`validation`

Examples

```
library(Qval)

set.seed(123)

## generate Q-matrix and data
K <- 3
I <- 20
example.Q <- sim.Q(K, I)
IQ <- list(
  P0 = runif(I, 0.0, 0.2),
  P1 = runif(I, 0.8, 1.0)
)
example.data <- sim.data(Q = example.Q, N = 500, IQ = IQ, model = "GDINA", distribute = "horder")

## calculate PVAF directly
PVAF <-get.PVAF(Y = example.data$dat, Q = example.Q)
print(PVAF)
```

```
## caculate PVAF after fitting CDM
example.CDM.obj <- CDM(example.data$dat, example.Q, model="GDINA")
PVAF <-get.PVAF(CDM.obj = example.CDM.obj)
print(PVAF)
```

get.Rmatrix *Restriction matrix*

Description

This function returns the restriction matrix (de la Torre, 2011; Ma & de la Torre, 2020) based on two q-vectors, where the two q-vectors can only differ by one attribute.

Usage

```
get.Rmatrix(Q.i, Q.i.k)
```

Arguments

`Q.i`	A q-vector
`Q.i.k`	Another q-vector

Value

A restriction matrix

References

de la Torre, J. (2011). The Generalized DINA Model Framework. Psychometrika, 76(2), 179-199. DOI: 10.1007/s11336-011-9207-7.

Ma, W., & de la Torre, J. (2020). An empirical Q-matrix validation method for the sequential generalized DINA model. British Journal of Mathematical and Statistical Psychology, 73(1), 142-163. DOI: 10.1111/bmsp.12156.

Examples

```
Q.i <- c(1, 1, 0)
Q.i.k <- c(1, 1, 1)

Rmatrix <- get.Rmatrix(Q.i, Q.i.k)

print(Rmatrix)
```

parallel_iter *A tool for the β Method*

Description

This function performs a single iteration of the β method for A item's validation. It is designed to be used in parallel computing environments to speed up the validation process of the β method. The function is a utility function for `validation`, and it should not be called independently by the user.

Usage

```
parallel_iter(
  i,
  Y,
  P.alpha.Xi,
  P.alpha,
  pattern,
  ri,
  Ni,
  Q.pattern.ini,
  model,
  criter,
  search.method,
  P_GDINA,
  Q.beta,
  L,
  K,
  alpha.P,
  get.MLRlasso,
  priority
)
```

Arguments

`i`	Item number that need to be validated.
`Y`	Observed data matrix for validation.
`P.alpha.Xi`	Individual posterior
`P.alpha`	Attribute prior weights.
`pattern`	The attribute mastery pattern matrix.
`ri`	A vector that contains the numbers of examinees in each knowledge state who correctly answered item i.
`Ni`	A vector that contains the total numbers of examinees in each knowledge state.
`Q.pattern.ini`	Initial pattern number for the model.
`model`	Model object used for fitting (e.g., GDINA).

criter	Fit criterion ("AIC", "BIC", "CAIC", or "SABIC").
search.method	Search method for model selection ("beta", "ESA", "SSA", or "PAA").
P_GDINA	Function to calculate probabilities for GDINA model.
Q.beta	Q-matrix for validation.
L	Number of latent pattern.
K	Number of attributes.
alpha.P	Individuals' marginal mastery probabilities matrix (Tu et al., 2022)
get.MLRlasso	Function for Lasso regression with multiple linear regression.
priority	Vector of priorities for PAA method search.

Value

An object of class `validation` is a `list` containing the following components:

fit.index.pre	The previous fit index value after applying the selected search method.
fit.index.cur	The current fit index value after applying the selected search method.
Q.pattern.cur	The pattern that corresponds to the optimal model configuration for the current iteration.
priority	The priority vector used in the PAA method, if applicable.

---

plot.Hull *Hull Plot*

Description

This function can provide the Hull plot. The points suggested by the Hull method are marked in red.

Usage

```
## S3 method for class 'Hull'
plot(x, i, ...)
```

Arguments

x	A `list` containing all the information needed to plot the Hull plot. It can be gotten from the `validation` object when `method = "Hull"`.
i	A numeric, which represents the item you want to plot Hull curve.
...	Additional arguments to be passed to the plotting function.

Value

None. This function is used for side effects (plotting).

Examples

```
set.seed(123)
library(Qval)

## generate Q-matrix and data
K <- 5
I <- 20
IQ <- list(
  P0 = runif(I, 0.1, 0.3),
  P1 = runif(I, 0.7, 0.9)
)


Q <- sim.Q(K, I)
data <- sim.data(Q = Q, N = 500, IQ = IQ, model = "GDINA", distribute = "horder")
MQ <- sim.MQ(Q, 0.1)

CDM.obj <- CDM(data$dat, MQ)

############### ESA ###############
Hull.obj <- validation(data$dat, MQ, CDM.obj, method = "Hull", search.method = "ESA")
Hull.fit <- Hull.obj$Hull.fit

## plot Hull curve for item 5
plot(Hull.fit, 5)

############### PAA ###############
Hull.obj <- validation(data$dat, MQ, CDM.obj, method = "Hull", search.method = "PAA")
Hull.fit <- Hull.obj$Hull.fit

## plot Hull curve for item 5
plot(Hull.fit, 5)
```

`sim.data` *generate response data*

Description

randomly generate response data matrix according to certen conditions, including attributes distribution, item quality, sample size, Q-matrix and cognitive diagnosis models (CDMs).

Usage

```
sim.data(
  Q = NULL,
  N = NULL,
  IQ = list(P0 = NULL, P1 = NULL),
  model = "GDINA",
  distribute = "uniform",
  control = NULL,
  verbose = TRUE
)
```

Arguments

`Q`	The Q-matrix. A random 30×5 Q-matrix (`sim.Q`) will be used if NULL.
`N`	Sample size. Default = 500.
`IQ`	A List contains tow I-length vectors: `P0` and `P1`.
`model`	Type of model to be fitted; can be `"GDINA"`, `"LCDM"`, `"DINA"`, `"DINO"`, `"ACDM"`, `"LLM"`, or `"rRUM"`.
`distribute`	Attribute distributions; can be `"uniform"` for the uniform distribution, `"mvnorm"` for the multivariate normal distribution (Chiu, Douglas, & Li, 2009) and `"horder"` for the higher-order distribution (Tu et al., 2022).
`control`	A list of control parameters with elements: • `sigma` A positive-definite symmetric matrix specifying the variance-covariance matrix when `distribute = "mvnorm"`. Default = 0.5 (Chiu, Douglas, & Li, 2009). • `cutoffs` A vector giving the cutoff for each attribute when `distribute = "mvnorm"`. Default = $k/(1+K)$ (Chiu, Douglas, & Li, 2009). • `theta` A vector of length N representing the higher-order ability for each examinee. By default, generate randomly from the normal distribution (Tu et al, 2022). • `a` The slopes for the higher-order model when `distribute = "horder"`. Default = 1.5 (Tu et al, 2022). • `b` The intercepts when `distribute = "horder"`. By default, select equally spaced values between -1.5 and 1.5 according to the number of attributes (Tu et al, 2022).
`verbose`	Logical indicating to print information or not. Default is `TRUE`

Value

Object of class `simGDINA`. An `simGDINA` object gained by `simGDINA` function form `GDINA` package. Elements that can be extracted using method extract include:

`dat`	An `N` × `I` simulated item response matrix.
`Q`	The Q-matrix.
`attribute`	An `N` × `K` matrix for inviduals' attribute patterns.
`catprob.parm`	A list of non-zero category success probabilities for each latent group.
`delta.parm`	A list of delta parameters.

`higher.order.parm`	Higher-order parameters.
`mvnorm.parm`	Multivariate normal distribution parameters.
`LCprob.parm`	A matrix of item/category success probabilities for each latent class.

Author(s)

Haijiang Qin <Haijiang133@outlook.com>

References

Chiu, C.-Y., Douglas, J. A., & Li, X. (2009). Cluster Analysis for Cognitive Diagnosis: Theory and Applications. Psychometrika, 74(4), 633-665. DOI: 10.1007/s11336-009-9125-0.

Tu, D., Chiu, J., Ma, W., Wang, D., Cai, Y., & Ouyang, X. (2022). A multiple logistic regression-based (MLR-B) Q-matrix validation method for cognitive diagnosis models:A confirmatory approach. Behavior Research Methods. DOI: 10.3758/s13428-022-01880-x.

Examples

```
################################################################
#                          Example 1                           #
#          generate data follow the uniform distrbution        #
################################################################
library(Qval)

set.seed(123)

K <- 5
I <- 10
Q <- sim.Q(K, I)

IQ <- list(
  P0 = runif(I, 0.0, 0.2),
  P1 = runif(I, 0.8, 1.0)
)

data <- sim.data(Q = Q, N = 10, IQ=IQ, model = "GDINA", distribute = "uniform")

print(data$dat)

################################################################
#                          Example 2                           #
#          generate data follow the mvnorm distrbution         #
################################################################
set.seed(123)
K <- 5
I <- 10
Q <- sim.Q(K, I)

IQ <- list(
  P0 = runif(I, 0.0, 0.2),
  P1 = runif(I, 0.8, 1.0)
)

example_cutoffs <- sample(qnorm(c(1:K)/(K+1)), ncol(Q))
data <- sim.data(Q = Q, N = 10, IQ=IQ, model = "GDINA", distribute = "mvnorm",
                 control = list(sigma = 0.5, cutoffs = example_cutoffs))

print(data$dat)

################################################################
#                          Example 3                           #
#           generate data follow the horder distrbution        #
################################################################
set.seed(123)
K <- 5
I <- 10
Q <- sim.Q(K, I)

IQ <- list(
  P0 = runif(I, 0.0, 0.2),
  P1 = runif(I, 0.8, 1.0)
)

example_theta <- rnorm(10, 0, 1)
example_b <- seq(-1.5,1.5,length.out=K)
data <- sim.data(Q = Q, N = 10, IQ=IQ, model = "GDINA", distribute = "horder",
                 control = list(theta = example_theta, a = 1.5, b = example_b))

print(data$dat)
```

sim.MQ *Simulate mis-specifications*

Description

simulate certen $rate$ mis-specifications in the Q-matrix.

Usage

```
sim.MQ(Q, rate, verbose = TRUE)
```

Arguments

Q	The Q-matrix (sim.Q) that need to simulate mis-specifications.
rate	The pecentage of mis-specifications in the$Q$.
verbose	Logical indicating to print information or not. Default is TRUE

Value

An object of class $matrix$.

Author(s)

Haijiang Qin <Haijiang133@outlook.com>

Examples

```
library(Qval)

set.seed(123)

Q <- sim.Q(5, 10)
print(Q)

MQ <- sim.MQ(Q, 0.1)
print(MQ)
```

`sim.Q` *generate a random Q-matrix*

Description

generate a $I * K$ Q-matrix randomly, which consisted of one-attribute q-vectors (0.5), two-attribute q-vectors (0.25), and three-attribute q-vectors (0.25). This function ensures that the generated Q-matrix contains at least two identity matrices as a priority.

Usage

```
sim.Q(K, I)
```

Arguments

K	The number of attributes of each item.
I	The number of items.

Value

An object of class $matrix$.

Author(s)

Haijiang Qin <Haijiang133@outlook.com>

References

Najera, P., Sorrel, M. A., de la Torre, J., & Abad, F. J. (2021). Balancing fit and parsimony to improve Q-matrix validation. Br J Math Stat Psychol, 74 Suppl 1, 110-130. DOI: 10.1111/bmsp.12228.

Examples

```
library(Qval)

set.seed(123)

Q <- sim.Q(5, 10)
print(Q)
```

`validation` *Perform Q-matrix validation methods*

Description

This function uses generalized Q-matrix validation methods to validate the Q-matrix, including commonly used methods such as GDI (de la Torre, & Chiu, 2016; Najera, Sorrel, & Abad, 2019; Najera et al., 2020), Wald (Ma, & de la Torre, 2020), Hull (Najera et al., 2021), and MLR-B (Tu et al., 2022). It supports different iteration methods (test level or item level; Najera et al., 2020; Najera et al., 2021; Tu et al., 2022) and can apply various attribute search methods (ESA, SSA, PAA; de la Torre, 2008; Terzi, & de la Torre, 2018). More see details.

Usage

```
validation(
  Y,
  Q,
  CDM.obj = NULL,
  par.method = "EM",
  mono.constraint = TRUE,
  model = "GDINA",
  method = "GDI",
  search.method = "PAA",
  maxitr = 1,
  iter.level = "test",
  eps = 0.95,
  alpha.level = 0.05,
  criter = NULL,
  verbose = TRUE
)
```

Arguments

`Y`	A required `N` × `I` matrix or data.frame consisting of the responses of `N` individuals to `I` items. Missing values need to be coded as `NA`.
`Q`	A required binary `I` × `K` containing the attributes not required or required, 0 or 1, to master the items. The `i`th row of the matrix is a binary indicator vector indicating which attributes are not required (coded by 0) and which attributes are required (coded by 1) to master item `i`.
`CDM.obj`	An object of class `CDM.obj`. When it is not NULL, it enables rapid verification of the Q-matrix without the need for parameter estimation. @seealso `CDM`.
`par.method`	Type of mtehod to estimate CDMs' parameters; one out of `"EM"`, `"BM"`. Default = `"EM"` However, `"BM"` is only avaible when `method = "GDINA"`.
`mono.constraint`	Logical indicating whether monotonicity constraints should be fulfilled in estimation. Default = `TRUE`.
`model`	Type of model to fit; can be `"GDINA"`, `"LCDM"`, `"DINA"`, `"DINO"` , `"ACDM"`, `"LLM"`, or `"rRUM"`. Default = `"GDINA"`. @seealso `CDM`.
`method`	The methods to validata Q-matrix, can be `"GDI"`, `"Wald"`, `"Hull"`, `"MLR-B"` and `"beta"`. The `"model"` must be `"GDINA"` when `method = "Wald"`. Please note that the β method has different meanings when applying different search algorithms. For more details, see section 'Search algorithm' below. Default = `"GDI"`. See details.
`search.method`	Character string specifying the search method to use during validation. **"ESA"** for exhaustive search algorithm. Can not for the `'Wald'` method. **"SSA"** for sequential search algorithm (see de la Torre, 2008; Terzi & de la Torre, 2018). This option can be used when the. It will be equal to `"forward"` when `method = "Wald"`. **"PAA"** for priority attribute algorithm. **"stepwise"** only for the `"Wald" method` **"beta"** only for the `"beta" method`
`maxitr`	Number of max iterations. Default = `1`.
`iter.level`	Can be `"item"` level, `"test.att"` or `"test"` level. Default = `"test"` and `"test.att"` can not for `"Wald"` and `"MLR-B"`. See details.
`eps`	Cut-off points of $PVAF$, will work when the method is `"GDI"` or `"Wald"`. Default = `0.95`. See details.
`alpha.level`	alpha level for the wald test. Default = `0.05`
`criter`	The kind of fit-index value. When `method = "Hull"`, it can be `R^2` for $R^2_{McFadden}$ @seealso `get.R2` or `'PVAF'` for the proportion of variance accounted for ($PVAF$) @seealso `get.PVAF`. When `method = "beta"`, it can be `'AIC'`, `'BIC'`, `'CAIC'` or `'SABIC'`. Default = `"PVAF"` for `'Hull'` and default = `"AIC"` for `'beta'`. See details.
`verbose`	Logical indicating to print iterative information or not. Default is `TRUE`

Value

An object of class `validation` is a `list` containing the following components:

`Q.orig`	The original Q-matrix that maybe contains some mis-specifications and need to be validate.
`Q.sug`	The Q-matrix that suggested by certain validation method.
`process`	A matrix that contains the modification process of each question during each iteration. Each row represents an iteration, and each column corresponds to the q-vector index of the respective question. The order of the indices is consistent with the row numbering in the matrix generated by the @seealso `attributepattern` function in the `GDINA` package.
`priority`	An `I × K` matrix that contains the priority of every attribute for each item. Only when the `search.method` is `"PAA"`, the value is availble. See details.
`Hull.fit`	A `list` containing all the information needed to plot the Hull plot, which is available only when `method = "Hull"`.
`iter`	The number of iteration.
`time.cost`	The time that CPU cost to finish the function.

The GDI method

The GDI method (de la Torre & Chiu, 2016), as the first Q-matrix validation method applicable to saturated models, serves as an important foundation for various mainstream Q-matrix validation methods.

The method calculates the proportion of variance accounted for ($PVAF$; @seealso `get.PVAF`) for all possible q-vectors for each item, selects the q-vector with a $PVAF$ just greater than the cut-off point (or Epsilon, EPS) as the correction result, and the variance ζ^2 is the generalized discriminating index (GDI; de la Torre & Chiu, 2016). Therefore, the GDI method is also considered as a generalized extension of the $delta$ method (de la Torre, 2008), which also takes maximizing discrimination as its basic idea. In the GDI method, ζ^2 is defined as the weighted variance of the correct response probabilities across all mastery patterns, that is:

$$\zeta^2 = \sum_{l=1}^{2^K} \pi_l (P(X_{pi} = 1|\alpha_l) - P_i^{mean})^2$$

where π_l represents the prior probability of mastery pattern l; $P_i^{mean} = \sum_{k=1}^{K} \pi_l P(X_{pi} = 1|\alpha_l)$ is the weighted average of the correct response probabilities across all attribute mastery patterns. When the q-vector is correctly specified, the calculated ζ^2 should be maximized, indicating the maximum discrimination of the item. However, in reality, ζ^2 continues to increase when the q-vector is over-specified, and the more attributes that are over-specified, the larger ζ^2 becomes. The q-vector with all attributes set to 1 (i.e., $\mathbf{q}_{1:K}$) has the largest ζ^2 (de la Torre, 2016). This is because an increase in attributes in the q-vector leads to an increase in item parameters, resulting in greater differences in correct response probabilities across attribute patterns and, consequently, increased variance. However, this increase in variance is spurious. Therefore, de la Torre et al. calculated $PVAF = \frac{\zeta^2}{\zeta^2_{1:K}}$ to describe the degree to which the discrimination of the current q-vector explains the maximum discrimination. They selected an appropriate $PVAF$ cut-off point to achieve a balance between q-vector fit and parsimony. According to previous studies, the $PVAF$

cut-off point is typically set at 0.95 (Ma & de la Torre, 2020; Najera et al., 2021). Najera et al. (2019) proposed using multinomial logistic regression to predict a more appropriate cut-off point for $PVAF$. The cut-off point is denoted as eps, and the predicted regression equation is as follows:

$$\log\left(\frac{eps}{1-eps}\right) = \text{logit}(eps) = -0.405 + 2.867 \cdot IQ + 4.840 \times 10^4 \cdot N - 3.316 \times 10^3 \cdot I$$

Where IQ represents the question quality, calculated as the negative difference between the probability of an examinee with all attributes answering the question correctly and the probability of an examinee with no attributes answering the question correctly ($IQ = -\{P(\mathbf{1}) - [1 - P(\mathbf{0})]\}$), and N and I represent the number of examinees and the number of questions, respectively.

The Wald method

The Wald method (Ma & de la Torre, 2020) combines the Wald test with $PVAF$ to correct the Q-matrix at the item level. Its basic logic is as follows: when correcting item i, the single attribute that maximizes the $PVAF$ value is added to a vector with all attributes set to $\mathbf{0}$ (i.e., $\mathbf{q} = (0, 0, \ldots, 0)$) as a starting point. In subsequent iterations, attributes in this vector are continuously added or removed through the Wald test. The correction process ends when the $PVAF$ exceeds the cut-off point or when no further attribute changes occur. The Wald statistic follows an asymptotic χ^2 distribution with a degree of freedom of $2^{K^*} - 1$.

The calculation method is as follows:

$$Wald = [\mathbf{R} \times P_i(\alpha)]^{'} (\mathbf{R} \times \mathbf{V}_i \times \mathbf{R})^{-1} [\mathbf{R} \times P_i(\alpha)]$$

$\mathbf{R}$ represents the restriction matrix; $P_i(\alpha)$ denotes the vector of correct response probabilities for item i; $\mathbf{V}_i$ is the variance-covariance matrix of the correct response probabilities for item i, which can be obtained by multiplying the $\mathbf{M}_i$ matrix (de la Torre, 2011) with the variance-covariance matrix of item parameters $\mathbf{\Sigma}_i$, i.e., $\mathbf{V}_i = \mathbf{M}_i \times \mathbf{\Sigma}_i$. The $\mathbf{\Sigma}_i$ can be derived by inverting the information matrix. Using the the empirical cross-product information matrix (de la Torre, 2011) to calculate $\mathbf{\Sigma}_i$.

$\mathbf{M}_i$ is a $2^{K^*} 2^{K^*}$ matrix that represents the relationship between the parameters of item i and the attribute mastery patterns. The rows represent different mastery patterns, while the columns represent different item parameters.

The Hull method

The Hull method (Najera et al., 2021) addresses the issue of the cut-off point in the GDI method and demonstrates good performance in simulation studies. Najera et al. applied the Hull method for determining the number of factors to retain in exploratory factor analysis (Lorenzo-Seva et al., 2011) to the retention of attribute quantities in the q-vector, specifically for Q-matrix validation. The Hull method aligns with the GDI approach in its philosophy of seeking a balance between fit and parsimony. While GDI relies on a preset, arbitrary cut-off point to determine this balance, the Hull method utilizes the most pronounced elbow in the Hull plot to make this judgment. The the most pronounced elbow is determined using the following formula:

$$st = \frac{(f_k - f_{k-1})/(np_k - np_{k-1})}{(f_{k+1} - f_k)/(np_{k+1} - np_k)}$$

where f_k represents the fit-index value (can be $PVAF$ @seealso `get.PVAF` or $R2$ @seealso `get.R2`) when the q-vector contains k attributes, similarly, f_{k-1} and f_{k+1} represent the fit-index value when the q-vector contains $k-1$ and $k+1$ attributes, respectively. np_k denotes the number of parameters when the q-vector has k attributes, which is 2^k for a saturated model. Likewise, np_{k-1} and np_{k+1} represent the number of parameters when the q-vector has $k-1$ and $k+1$ attributes, respectively. The Hull method calculates the st index for all possible q-vectors and retains the q-vector with the maximum st index as the corrected result. Najera et al. (2021) removed any concave points from the Hull plot, and when only the first and last points remained in the plot, the saturated q-vector was selected.

The MLR-B method

The MLR-B method proposed by Tu et al. (2022) differs from the GDI, Wald and Hull method in that it does not employ $PVAF$. Instead, it directly uses the marginal probabilities of attribute mastery for subjects to perform multivariate logistic regression on their observed scores. This approach assumes all possible q-vectors and conducts $2^K - 1$ regression modelings. After proposing regression equations that exclude any insignificant regression coefficients, it selects the q-vector corresponding to the equation with the minimum AIC fit as the validation result. The performance of this method in both the LCDM and GDM models even surpasses that of the Hull method, making it an efficient and reliable approach for Q-matrix correction.

The β method

The β method (Li & Chen, 2024) addresses the Q-matrix validation problem from the perspective of signal detection theory. Signal detection theory posits that any stimulus is a signal embedded in noise, where the signal always overlaps with noise. The β method treats the correct q-vector as the signal and other possible q-vectors as noise. The goal is to identify the signal from the noise, i.e., to correctly identify the q-vector. For a question i with the q-vector of the c-th type, the β index is computed as follows:

$$\beta_{jc} = \sum_{l=1}^{2^K} \left| \frac{r_{li}}{n_l} P_{ic}(\alpha_{\mathbf{l}}) - \left(1 - \frac{r_{li}}{n_l}\right) [1 - P_{ic}(\alpha_{\mathbf{l}})] \right| = \sum_{l=1}^{2^K} \left| \frac{r_{li}}{n_l} - [1 - P_{ic}(\alpha_{\mathbf{l}})] \right|$$

In the formula, r_{li} represents the number of examinees in knowledge state l who correctly answered item i, while n_l is the total number of examinees in knowledge state l. $P_{ic}(\alpha_{\mathbf{l}})$ denotes the probability that an examinee in knowledge state l answers item i correctly when the q-vector for item i is of the c-th type. In fact, $\frac{r_{li}}{n_l}$ is the observed probability that an examinee in knowledge state l answers item i correctly, and β_{jc} represents the difference between the actual proportion of correct answers for item i in each knowledge state and the expected probability of answering the item incorrectly in that state. Therefore, to some extent, β_{jc} can be considered as a measure of discriminability, and the β method posits that the correct q-vector maximizes β_{jc}, i.e.:

$$\mathbf{q}_i = \arg\max_{\mathbf{q}} \left(\beta_{jc} : \mathbf{q} \in \left\{ \mathbf{q}_{ic},\, c = 1, 2, \ldots, 2^K - 1 \right\} \right)$$

Therefore, essentially, β_{jc} is an index similar to GDI. Both increase as the number of attributes in the q-vector increases. Unlike the GDI method, the β method does not continue to compute $\beta_{jc}/\beta_{j[11\ldots1]}$ but instead uses the minimum AIC value to determine whether the attributes in the q-vector are sufficient. In Package Qval, parLapply will be used to accelerate the β method.

Please note that the β method has different meanings when applying different search algorithms. For more details, see section 'Search algorithm' below.

Iterative procedure

The iterative procedure that one item modification at a time is item level iteration (`"item"`) in (Najera et al., 2020, 2021), while the iterative procedure that the entire Q-matrix is modified at each iteration is test level iteration (`"test"`) (Najera et al., 2020; Tu et al., 2022).

The steps of the `item` level iterative procedure algorithm are as follows:

Step1 Fit the `CDM` according to the item responses and the provisional Q-matrix ($\mathbf{Q}^0$).

Step2 Validate the provisional Q-matrix and gain a suggested Q-matrix ($\mathbf{Q}^1$).

Step3 for each item, $PVAF_{0i}$ as the $PVAF$ of the provisional q-vector specified in $\mathbf{Q}^0$, and $PVAF_{1i}$ as the $PVAF$ of the suggested q-vector in $\mathbf{Q}^1$.

Step4 Calculate all items' $\Delta PVAF_i$, defined as $\Delta PVAF_i = |PVAF_{1i} - PVAF_{0i}|$

Step5 Define the hit item as the item with the highest $\Delta PVAF_i$.

Step6 Update $\mathbf{Q}^0$ by changing the provisional q-vector by the suggested q-vector of the hit item.

Step7 Iterate over Steps 1 to 6 until $\sum_{i=1}^{I} \Delta PVAF_i = 0$

When the Q-matrix validation method is `'MLR-B'`, `'Hull'` when `criter = 'R2'` or `'beta'`, $PVAF$ is not used. In this case, the criterion for determining which question's index will be replaced is AIC, R^2or AIC, respectively.

`test.level = 'test.att'` will use a method called the test-attribute iterative procedure (Najera et al., 2021), which modifies all items in each iteration while following the principle of minimizing changes in the number of attributes.

The steps of the `test` level iterative procedure algorithm are as follows:

Step1 Fit the `CDM` according to the item responses and the provisional Q-matrix ($\mathbf{Q}^0$).

Step2 Validate the provisional Q-matrix and gain a suggested Q-matrix ($\mathbf{Q}^1$).

Step3 Check whether $\mathbf{Q}^1 = \mathbf{Q}^0$. If `TRUE`, terminate the iterative algorithm. If `FALSE`, Update $\mathbf{Q}^0$ as $\mathbf{Q}^1$.

Step4 Iterate over Steps 1 and 3 until one of conditions as follows is satisfied: 1. $\mathbf{Q}^1 = \mathbf{Q}^0$; 2. Reach the max iteration (`maxitr`); 3. $\mathbf{Q}^1$ does not satisfy the condition that an attribute is measured by one item at least.

Search algorithm

Three search algorithms are available: Exhaustive Search Algorithm (ESA), Sequential Search Algorithm (SSA), and Priority Attribute Algorithm (PAA). ESA is a brute-force algorithm. When validating the q-vector of a particular item, it traverses all possible q-vectors and selects the most appropriate one based on the chosen Q-matrix validation method. Since there are 2^{K-1} possible q-vectors with K attributes, ESA requires 2^{K-1} searches.

SSA reduces the number of searches by adding one attribute at a time to the q-vector in a stepwise manner. Therefore, in the worst-case scenario, SSA requires $K(K-1)/2$ searches. The detailed steps are as follows:

Step 1 Define an empty q-vector $\mathbf{q}^0 = [00...0]$ of length K, where all elements are 0.

Step 2 Examine all single-attribute q-vectors, which are those formed by changing one of the 0s in $\mathbf{q}^0$ to 1. According to the criteria of the chosen Q-matrix validation method, select the optimal single-attribute q-vector, denoted as $\mathbf{q}^1$.

Step 3 Examine all two-attribute q-vectors, which are those formed by changing one of the 0s in $\mathbf{q}^1$ to 1. According to the criteria of the chosen Q-matrix validation method, select the optimal two-attribute q-vector, denoted as $\mathbf{q}^2$.

Step 4 Repeat this process until $\mathbf{q}^K$ is found, or the stopping criterion of the chosen Q-matrix validation method is met.

PAA is a highly efficient and concise algorithm that evaluates whether each attribute needs to be included in the q-vector based on the priority of the attributes. @seealso `get.priority`. Therefore, even in the worst-case scenario, PAA only requires K searches. The detailed process is as follows:

Step 1 Using the applicable CDM (e.g. the G-DINA model) to estimate the model parameters and obtain the marginal attribute mastery probabilities matrix $\mathbf{\Lambda}$

Step 2 Use LASSO regression to calculate the priority of each attribute in the q-vector for item i

Step 3 Check whether each attribute is included in the optimal q-vector based on the attribute priorities from high to low seriatim and output the final suggested q-vector according to the criteria of the chosen Q-matrix validation method.

It should be noted that the Wald method proposed by Ma & de la Torre (2020) uses a `"stepwise"` search approach. This approach involves incrementally adding or removing 1 from the q-vector and evaluating the significance of the change using the Wald test: 1. If removing a 1 results in non-significance (indicating that the 1 is unnecessary), the 1 is removed from the q-vector; otherwise, the q-vector remains unchanged. 2. If adding a 1 results in significance (indicating that the 1 is necessary), the 1 is added to the q-vector; otherwise, the q-vector remains unchanged. The process stops when the q-vector no longer changes or when the PVAF reaches the preset cut-off point (i.e., 0.95). Stepwise are unique search approach of the Wald method, and users should be aware of this. Since stepwise is inefficient and differs significantly from the extremely high efficiency of PAA, Package `Qval` also provides `PAA` for q-vector search in the Wald method. When applying the PAA version of the Wald method, the search still examines whether each attribute is necessary (by checking if the Wald test reaches significance after adding the attribute) according to attribute priority. The search stops when no further necessary attributes are found or when the PVAF reaches the preset cut-off point (i.e., 0.95). The "forward" search approach is another search method available for the Wald method, which is equivalent to `'SSA'`. When `'Wald'` uses `search.method = 'SSA'`, it means that the Wald method is employing the forward search approach. Its basic process is the same as `'stepwise'`, except that it does not remove elements from the q-vector. Therefore, the "forward" search approach is essentially SSA.

Please note that, since the β method essentially selects q-vectors based on AIC, even without using the iterative process, the β method requires multiple parameter estimations to obtain the AIC values for different q-vectors. Therefore, the β method is more time-consuming and computationally intensive compared to the other methods. Li and Chen (2024) introduced a specialized search approach for the β method, which is referred to as the β search (`search.method = 'beta'`). The number of searches required is $2^{K-2} + K + 1$, and the specific steps are as follows:

Step 1 For item i, sequentially examine the β values for each single-attribute q-vector, select the largest β_{most} and the smallest β_{least}, along with the corresponding attributes k_{most} and k_{least}. (K searches)

Step 2 Then, add all possible q-vectors (a total of $2^K - 1$) containing attribute k_{most} and not containing k_{least} to the search space $\mathbf{S}_i$ (a total of 2^{K-2})), and unconditionally add the saturated q-vector $[11 \dots 1]$ to $\mathbf{S}_i$ to ensure that it is tested.

Step 3 Select the q-vector with the minimum AIC from $\mathbf{S}_i$ as the final output of the β method. (The remaining $2^{K-2} + 1$ searches)

The `Qval` package also provides three search methods, ESA, SSA, and PAA, for the β method. When the β method applies these three search methods, Q-matrix validation can be completed without calculating any β values, as the β method essentially uses `AIC` for selecting q-vectors. For example, when applying ESA, the β method does not need to perform Step 1 of the β search and only needs to include all possible q-vectors (a total of $2^K - 1$) in $\mathbf{S}_i$, then outputs the corresponding q-vector based on the minimum AIC. When applying SSA or PAA, the β method also does not require any calculation of β values. In this case, the β method is consistent with the Q-matrix validation process described by Chen et al. (2013) using relative fit indices. Therefore, when the β method does not use β search, it is equivalent to the method of Chen et al. (2013). To better implement Chen et al. (2013)'s Q-matrix validation method using relative fit indices, the `Qval` package also provides BIC, $CAIC$, and $SABIC$ as alternatives to validate q-vectors, in addition to AIC.

Author(s)

Haijiang Qin <Haijiang133@outlook.com>

References

Chen, J., de la Torre, J., & Zhang, Z. (2013). Relative and Absolute Fit Evaluation in Cognitive Diagnosis Modeling. Journal of Educational Measurement, 50(2), 123-140. DOI: 10.1111/j.1745-3984.2012.00185.x

de la Torre, J., & Chiu, C. Y. (2016). A General Method of Empirical Q-matrix Validation. Psychometrika, 81(2), 253-273. DOI: 10.1007/s11336-015-9467-8.

de la Torre, J. (2008). An Empirically Based Method of Q-Matrix Validation for the DINA Model: Development and Applications. Journal of Education Measurement, 45(4), 343-362. DOI: 10.1111/j.1745-3984.2008.00069.x.

Li, J., & Chen, P. (2024). A new Q-matrix validation method based on signal detection theory. British Journal of Mathematical and Statistical Psychology, 00, 1–33. DOI: 10.1111/bmsp.12371

Lorenzo-Seva, U., Timmerman, M. E., & Kiers, H. A. (2011). The Hull method for selecting the number of common factors. Multivariate Behavioral Research, 46, 340–364. DOI: 10.1080/00273171.2011.564527.

Ma, W., & de la Torre, J. (2020). An empirical Q-matrix validation method for the sequential generalized DINA model. British Journal of Mathematical and Statistical Psychology, 73(1), 142-163. DOI: 10.1111/bmsp.12156.

McFadden, D. (1974). Conditional logit analysis of qualitative choice behavior. In P. Zarembka (Ed.), Frontiers in economics (pp. 105–142). New York, NY: Academic Press.

Najera, P., Sorrel, M. A., & Abad, F. J. (2019). Reconsidering Cutoff Points in the General Method of Empirical Q-Matrix Validation. Educational and Psychological Measurement, 79(4), 727-753. DOI: 10.1177/0013164418822700.

Najera, P., Sorrel, M. A., de la Torre, J., & Abad, F. J. (2020). Improving Robustness in Q-Matrix Validation Using an Iterative and Dynamic Procedure. Applied Psychological Measurement, 44(6), 431-446. DOI: 10.1177/0146621620909904.

Najera, P., Sorrel, M. A., de la Torre, J., & Abad, F. J. (2021). Balancing fit and parsimony to improve Q-matrix validation. British Journal of Mathematical and Statistical Psychology, 74 Suppl 1, 110-130. DOI: 10.1111/bmsp.12228.

Terzi, R., & de la Torre, J. (2018). An Iterative Method for Empirically-Based Q-Matrix Validation. International Journal of Assessment Tools in Education, 248-262. DOI: 10.21449/ijate.40719.

Tu, D., Chiu, J., Ma, W., Wang, D., Cai, Y., & Ouyang, X. (2022). A multiple logistic regression-based (MLR-B) Q-matrix validation method for cognitive diagnosis models: A confirmatory approach. Behavior Research Methods. DOI: 10.3758/s13428-022-01880-x.

Examples

```
################################################################
#                          Example 1                           #
#             The GDI method to validate Q-matrix              #
################################################################
set.seed(123)

library(Qval)

## generate Q-matrix and data
K <- 4
I <- 20
example.Q <- sim.Q(K, I)
IQ <- list(
  P0 = runif(I, 0.0, 0.2),
  P1 = runif(I, 0.8, 1.0)
)
example.data <- sim.data(Q = example.Q, N = 500, IQ = IQ,
                         model = "GDINA", distribute = "horder")

## simulate random mis-specifications
example.MQ <- sim.MQ(example.Q, 0.1)


## using MMLE/EM to fit CDM model first
example.CDM.obj <- CDM(example.data$dat, example.MQ)

## using the fitted CDM.obj to avoid extra parameter estimation.
Q.GDI.obj <- validation(example.data$dat, example.MQ, example.CDM.obj, method = "GDI")


## also can validate the Q-matrix directly
Q.GDI.obj <- validation(example.data$dat, example.MQ)

## item level iteration
Q.GDI.obj <- validation(example.data$dat, example.MQ, method = "GDI",
                        iter.level = "item", maxitr = 150)
```

```
## search method
Q.GDI.obj <- validation(example.data$dat, example.MQ, method = "GDI",
                        search.method = "ESA")

## cut-off point
Q.GDI.obj <- validation(example.data$dat, example.MQ, method = "GDI",
                        eps = 0.90)

## check QRR
print(zQRR(example.Q, Q.GDI.obj$Q.sug))




################################################################
#                          Example 2                           #
#             The Wald method to validate Q-matrix             #
################################################################
set.seed(123)

library(Qval)

## generate Q-matrix and data
K <- 4
I <- 20
example.Q <- sim.Q(K, I)
IQ <- list(
  P0 = runif(I, 0.0, 0.2),
  P1 = runif(I, 0.8, 1.0)
)
example.data <- sim.data(Q = example.Q, N = 500, IQ = IQ, model = "GDINA",
                         distribute = "horder")

## simulate random mis-specifications
example.MQ <- sim.MQ(example.Q, 0.1)


## using MMLE/EM to fit CDM first
example.CDM.obj <- CDM(example.data$dat, example.MQ)

## using the fitted CDM.obj to avoid extra parameter estimation.
Q.Wald.obj <- validation(example.data$dat, example.MQ, example.CDM.obj, method = "Wald")


## also can validate the Q-matrix directly
Q.Wald.obj <- validation(example.data$dat, example.MQ, method = "Wald")

## check QRR
print(zQRR(example.Q, Q.Wald.obj$Q.sug))
```

```
################################################################
#                          Example 3                           #
#             The Hull method to validate Q-matrix             #
################################################################
set.seed(123)

library(Qval)

## generate Q-matrix and data
K <- 4
I <- 20
example.Q <- sim.Q(K, I)
IQ <- list(
  P0 = runif(I, 0.0, 0.2),
  P1 = runif(I, 0.8, 1.0)
)
example.data <- sim.data(Q = example.Q, N = 500, IQ = IQ, model = "GDINA",
                         distribute = "horder")

## simulate random mis-specifications
example.MQ <- sim.MQ(example.Q, 0.1)


## using MMLE/EM to fit CDM first
example.CDM.obj <- CDM(example.data$dat, example.MQ)

## using the fitted CDM.obj to avoid extra parameter estimation.
Q.Hull.obj <- validation(example.data$dat, example.MQ, example.CDM.obj, method = "Hull")


## also can validate the Q-matrix directly
Q.Hull.obj <- validation(example.data$dat, example.MQ, method = "Hull")

## change PVAF to R2 as fit-index
Q.Hull.obj <- validation(example.data$dat, example.MQ, method = "Hull", criter = "R2")

## check QRR
print(zQRR(example.Q, Q.Hull.obj$Q.sug))




################################################################
#                          Example 4                           #
#             The MLR-B method to validate Q-matrix            #
################################################################
set.seed(123)

library(Qval)

## generate Q-matrix and data
K <- 4
I <- 20
```

```
example.Q <- sim.Q(K, I)
IQ <- list(
  P0 = runif(I, 0.0, 0.2),
  P1 = runif(I, 0.8, 1.0)
)
example.data <- sim.data(Q = example.Q, N = 500, IQ = IQ, model = "GDINA",
                         distribute = "horder")

## simulate random mis-specifications
example.MQ <- sim.MQ(example.Q, 0.1)


## using MMLE/EM to fit CDM first
example.CDM.obj <- CDM(example.data$dat, example.MQ)

## using the fitted CDM.obj to avoid extra parameter estimation.
Q.MLR.obj <- validation(example.data$dat, example.MQ, example.CDM.obj, method = "MLR-B")


## also can validate the Q-matrix directly
Q.MLR.obj <- validation(example.data$dat, example.MQ, method  = "MLR-B")

## check QRR
print(zQRR(example.Q, Q.MLR.obj$Q.sug))
```

`Wald.test` *Wald.test for two q-vecotrs*

Description

This function flexibly provides the Wald test for any two q-vectors of a given item in the Q-matrix, but requires that the two q-vectors differ by only one attribute. Additionally, this function needs to accept a `CDM.obj`.

Usage

```
Wald.test(CDM.obj, Q.i, Q.i.k, i = 1)
```

Arguments

`CDM.obj`	An object of class `CDM.obj`. @seealso `CDM`.
`Q.i`	A q-vector
`Q.i.k`	Another q-vector
`i`	the item you focusing on

Details

$$Wald = [\mathbf{R} \times P_i(\alpha)]^{'} (\mathbf{R} \times \mathbf{V}_i \times \mathbf{R})^{-1} [\mathbf{R} \times P_i(\alpha)]$$

Value

An object of class `list` containing the following components:

`Wald.statistic`	The statistic of the Wald test.
`p.value`	The p value

Examples

```
set.seed(123)

K <- 3
I <- 20
N <- 500
IQ <- list(
  P0 = runif(I, 0.0, 0.2),
  P1 = runif(I, 0.8, 1.0)
)
Q <- sim.Q(K, I)
data <- sim.data(Q = Q, N = N, IQ = IQ, model = "GDINA", distribute = "horder")

CDM.obj <- CDM(data$dat, Q)

Q.i <- c(1, 0, 0)
Q.i.k <- c(1, 1, 0)

## Discuss whether there is a significant difference when
## the q-vector of the 2nd item in the Q-matrix is Q.i or Q.i.k.
Wald.test.obj <- Wald.test(CDM.obj, Q.i, Q.i.k, i=2)

print(Wald.test.obj)
```

`zOSR` *Caculate over-specifcation rate (OSR)*

Description

Caculate over-specifcation rate (OSR)

Usage

```
zOSR(Q.true, Q.sug)
```

Arguments

`Q.true`	The true Q-matrix.
`Q.sug`	The Q-matrix that has being validated.

Details

The OSR is defned as:

$$OSR = \frac{\sum_{i=1}^{I} \sum_{k=1}^{K} I(q_{ik}^{t} < q_{ik}^{s})}{IK}$$

where q_{ik}^{t} denotes the kth attribute of item `i` in the true Q-matrix (`Q.true`), q_{ik}^{s} denotes kth attribute of item `i` in the suggested Q-matrix(`Q.sug`), and $I(\cdot)$ is the indicator function.

Value

A numeric (OSR index).

Examples

```
library(Qval)

set.seed(123)

example.Q1 <- sim.Q(5, 30)
example.Q2 <- sim.MQ(example.Q1, 0.1)
OSR <- zOSR(example.Q1, example.Q2)
print(OSR)
```

`zQRR` *Caculate Q-matrix recovery rate (QRR)*

Description

Caculate Q-matrix recovery rate (QRR)

Usage

```
zQRR(Q.true, Q.sug)
```

Arguments

`Q.true`	The true Q-matrix.
`Q.sug`	A The Q-matrix that has being validated.

Details

The Q-matrix recovery rate (QRR) provides information on overall performance, and is defned as:

$$QRR = \frac{\sum_{i=1}^{I} \sum_{k=1}^{K} I(q_{ik}^{t} = q_{ik}^{s})}{IK}$$

where q_{ik}^{t} denotes the kth attribute of item i in the true Q-matrix ($Q.true$), q_{ik}^{s} denotes kth attribute of item i in the suggested Q-matrix($Q.sug$), and $I(\cdot)$ is the indicator function.

Value

A numeric (QRR index).

Examples

```
library(Qval)

set.seed(123)

example.Q1 <- sim.Q(5, 30)
example.Q2 <- sim.MQ(example.Q1, 0.1)
QRR <- zQRR(example.Q1, example.Q2)
print(QRR)
```

`zTNR` *Calculate true negative rate (TNR)*

Description

Calculate true negative rate (TNR)

Usage

```
zTNR(Q.true, Q.orig, Q.sug)
```

Arguments

`Q.true`	The true Q-matrix.
`Q.orig`	The Q-matrix need to be validated.
`Q.sug`	The Q-matrix that has being validated.

Details

TNR is defined as the proportion of correct elements which are correctly retained:

$$TNR = \frac{\sum_{i=1}^{I}\sum_{k=1}^{K} I(q_{ik}^{t} = q_{ik}^{s} | q_{ik}^{t} \neq q_{ik}^{o})}{\sum_{i=1}^{I}\sum_{k=1}^{K} I(q_{ik}^{t} \neq q_{ik}^{o})}$$

where q_{ik}^{t} denotes the kth attribute of item `i` in the true Q-matrix (`Q.true`), q_{ik}^{o} denotes kth attribute of item `i` in the original Q-matrix(`Q.orig`), q_{ik}^{s} denotes kth attribute of item `i` in the suggested Q-matrix(`Q.sug`), and $I(\cdot)$ is the indicator function.

Value

A numeric (TNR index).

Examples

```
library(Qval)

set.seed(123)

example.Q1 <- sim.Q(5, 30)
example.Q2 <- sim.MQ(example.Q1, 0.1)
example.Q3 <- sim.MQ(example.Q1, 0.05)
TNR <- zTNR(example.Q1, example.Q2, example.Q3)

print(TNR)
```

`zTPR` *Caculate true-positive rate (TPR)*

Description

Caculate true-positive rate (TPR)

Usage

```
zTPR(Q.true, Q.orig, Q.sug)
```

Arguments

`Q.true`	The true Q-matrix.
`Q.orig`	The Q-matrix need to be validated.
`Q.sug`	The Q-matrix that has being validated.

Details

TPR is defned as the proportion of correct elements which are correctly retained:

$$TPR = \frac{\sum_{i=1}^{I}\sum_{k=1}^{K} I(q_{ik}^{t} = q_{ik}^{s} | q_{ik}^{t} = q_{ik}^{o})}{\sum_{i=1}^{I}\sum_{k=1}^{K} I(q_{ik}^{t} = q_{ik}^{o})}$$

where q_{ik}^{t} denotes the kth attribute of item i in the true Q-matrix (`Q.true`), q_{ik}^{o} denotes kth attribute of item `i` in the original Q-matrix(`Q.orig`), q_{ik}^{s} denotes kth attribute of item `i` in the suggested Q-matrix(`Q.sug`), and $I(\cdot)$ is the indicator function.

Value

A numeric (TPR index).

Examples

```
library(Qval)

set.seed(123)

example.Q1 <- sim.Q(5, 30)
example.Q2 <- sim.MQ(example.Q1, 0.1)
example.Q3 <- sim.MQ(example.Q1, 0.05)
TPR <- zTPR(example.Q1, example.Q2, example.Q3)

print(TPR)
```

---

zUSR *Caculate under-specifcation rate (USR)*

Description

Caculate under-specifcation rate (USR)

Usage

```
zUSR(Q.true, Q.sug)
```

Arguments

`Q.true`	The true Q-matrix.
`Q.sug`	A The Q-matrix that has being validated.

Details

The USR is defned as:

$$USR = \frac{\sum_{i=1}^{I} \sum_{k=1}^{K} I(q_{ik}^{t} > q_{ik}^{s})}{IK}$$

where q_{ik}^t denotes the kth attribute of item `i` in the true Q-matrix (`Q.true`), q_{ik}^s denotes kth attribute of item `i` in the suggested Q-matrix(`Q.sug`), and $I(\cdot)$ is the indicator function.

Value

A numeric (USR index).

Examples

```
library(Qval)

set.seed(123)

example.Q1 <- sim.Q(5, 30)
example.Q2 <- sim.MQ(example.Q1, 0.1)
USR <- zUSR(example.Q1, example.Q2)
print(USR)
```

`zVRR` *Caculate vector recovery ratio (VRR)*

Description

Caculate vector recovery ratio (VRR)

Usage

```
zVRR(Q.true, Q.sug)
```

Arguments

`Q.true`	The true Q-matrix.
`Q.sug`	A The Q-matrix that has being validated.

Details

The VRR shows the ability of the validation method to recover q-vectors, and is determined by

$$VRR = \frac{\sum_{i=1}^{I} I(\mathbf{q}_{i}^{t} = \mathbf{q}_{i}^{s})}{I}$$

where $\mathbf{q}_i^t$ denotes the $\mathbf{q}$-vector of item `i` in the true Q-matrix (`Q.true`), $\mathbf{q}_i^s$ denotes the $\mathbf{q}$-vector of item `i` in the suggested Q-matrix(`Q.sug`), and $I(\cdot)$ is the indicator function.

Value

A numeric (VRR index).

Examples

```
library(Qval)

set.seed(123)

example.Q1 <- sim.Q(5, 30)
example.Q2 <- sim.MQ(example.Q1, 0.1)
VRR <- zVRR(example.Q1, example.Q2)
print(VRR)
```

Index