Package: guess (via r-universe)

June 7, 2026

Title Adjust Estimates of Learning for Guessing

Version 0.5.0

Description Provides tools to adjust estimates of learning for guessing-related bias in educational and survey research. Implements standard guessing correction methods and a sophisticated latent class model that leverages informative pre-post test transitions to account for guessing behavior. The package helps researchers obtain more accurate estimates of actual learning when respondents may guess on closed-ended knowledge items. For theoretical background and empirical validation, see Cor and Sood (2018) <https://gsood.com/research/papers/guess.pdf>.

URL https://github.com/finite-sample/guess, https://finite-sample.github.io/guess/

BugReports https://github.com/finite-sample/guess/issues

Depends R (>= 4.0.0)

Imports Rsolnp, stats, checkmate

License MIT + file LICENSE

LazyData true

VignetteBuilder knitr

Suggests knitr (>= 1.11), rmarkdown, testthat (>= 3.0.0), lintr, covr

RoxygenNote 7.3.3

Encoding UTF-8

Language en-US

Config/testthat/edition 3

Repository https://finite-sample.r-universe.dev

Date/Publication 2026-04-08 18:06:25 UTC

RemoteUrl https://github.com/finite-sample/guess

RemoteRef HEAD

RemoteSha 6397e949dd77b2711a56e0bcb00faac5c423cae4

Contents

`calculate_expected_values` *Calculate expected values for goodness of fit test*

Description

Calculate expected values for goodness of fit test

Usage

```
calculate_expected_values(gamma_i, params, total_obs, model_type = "nodk")
```

Arguments

`gamma_i`	item-specific gamma value
`params`	estimated parameters for the item
`total_obs`	total observations for the item
`model_type`	”nodk” or ”dk” model

Value

vector of expected values

`coef.guess_fit` *Extract coefficients from guess_fit*

Description

Extract coefficients from guess_fit

Usage

```
## S3 method for class 'guess_fit'
coef(object, ...)
```

Arguments

`object`	guess_fit object
`...`	ignored

Value

parameter matrix

cross_sectional_irt *Cross-sectional IRT learning probability*

Description

Converts the difference in ability estimates to a probability scale using the logistic function. Provides a comparable metric to posterior_learned() but without using the transition structure.

Usage

```
cross_sectional_irt(pre_test, pst_test, method = "logit", scale = 1)
```

Arguments

pre_test	data.frame of pre-test responses
pst_test	data.frame of post-test responses
method	character: ”logit” (default) or ”rasch”
scale	numeric scaling factor for ability difference (default 1)

Value

numeric vector of learning probabilities in [0, 1]

Examples

```
sim <- simulate_lca(n = 100, gk = 0.30, seed = 123, return_classes = TRUE)
p_learned_cs <- cross_sectional_irt(sim$pre, sim$post)
cor(p_learned_cs, sim$learned)
```

cross_sectional_learning *Cross-sectional learning estimate*

Description

Estimates learning as the difference in ability between post and pre test. This ignores the transition structure that the LCA model uses.

Usage

```
cross_sectional_learning(pre_test, pst_test, method = "logit")
```

Arguments

`pre_test`	data.frame of pre-test responses
`pst_test`	data.frame of post-test responses
`method`	character: ”logit” (default) or ”rasch”

Value

numeric vector of learning estimates (theta_post - theta_pre)

Examples

```
sim <- simulate_lca(n = 100, gk = 0.30, seed = 123, return_classes = TRUE)
learning_cs <- cross_sectional_learning(sim$pre, sim$post)
cor(learning_cs, sim$learned)
```

`cv_individuals` *K-fold cross-validation over individuals*

Description

Splits individuals into k folds, fits on training, evaluates on held-out.

Usage

```
cv_individuals(pre_test, pst_test, k = 5L, priors = NULL, seed = NULL)
```

Arguments

`pre_test`	data.frame of pre-test responses
`pst_test`	data.frame of post-test responses
`k`	integer number of folds
`priors`	optional numeric vector of starting parameters
`seed`	optional integer random seed

Value

list with fold_results, mean_ll, total_ll, perplexity, se

cv_items — *K-fold cross-validation over items*

Description

Splits items into k folds, fits on training items, evaluates on held-out items.

Usage

```
cv_items(transmatrix, k = 5L, priors = NULL, seed = NULL)
```

Arguments

`transmatrix`	numeric matrix from multi_transmat()
`k`	integer number of folds
`priors`	optional numeric vector of starting parameters
`seed`	optional integer random seed

Value

list with fold_results, mean_ll, total_ll, perplexity, se

estimate_ability — *Estimate ability from single timepoint (cross-sectional)*

Description

Estimates person ability using simple proportion correct (logit-transformed) or Rasch-style IRT. Ignores the transition structure between time points.

Usage

```
estimate_ability(responses, method = "logit", difficulty = NULL)
```

Arguments

`responses`	data.frame of binary responses (0/1)
`method`	character: ”logit” (default) or ”rasch”
`difficulty`	numeric vector of item difficulties (for rasch method)

Value

numeric vector of ability estimates (length = n individuals)

Examples

```
sim <- simulate_lca(n = 100, seed = 123)
theta_pre <- estimate_ability(sim$pre)
theta_post <- estimate_ability(sim$post)
```

`fit_model` *Goodness of fit statistics for transition matrix data*

Description

Chi-square goodness of fit between true and model based multivariate distribution. Handles both data with and without don't know responses automatically.

Usage

```
fit_model(pre_test, pst_test, g, est_param, force9 = FALSE)

fit_dk(pre_test, pst_test, g, est_param, force9 = FALSE)

fit_nodk(pre_test, pst_test, g, est_param)
```

Arguments

`pre_test`	data.frame carrying pre_test items
`pst_test`	data.frame carrying pst_test items
`g`	estimates of gamma produced from `lca_cor`
`est_param`	estimated parameters produced from `lca_cor`
`force9`	Optional. Force 9-column format even if no DK responses. Default is FALSE.

Details

Unified Goodness of Fit Statistics

Value

matrix with two rows: top row carrying chi-square value, bottom row p-values

Examples

```
## Not run:
# Fit model first
transmatrix <- multi_transmat(pre_test, pst_test)
res <- lca_cor(transmatrix)

# Calculate goodness of fit
fit_stats <- fit_model(pre_test, pst_test, res$params[nrow(res$params), ],
```

```
                        res$params[-nrow(res$params), ])

## End(Not run)
```

`format_transition_matrix` *Format transition matrix result with appropriate row and column names*

Description

Format transition matrix result with appropriate row and column names

Usage

```
format_transition_matrix(transition_list, n_items, add_aggregate = FALSE)
```

Arguments

`transition_list`	list of transition vectors
`n_items`	number of items
`add_aggregate`	whether to add aggregate row

Value

formatted matrix

`group_adj` *Group Level Adjustment That Accounts for Propensity to Guess*

Description

Adjusts observed 1s based on propensity to guess (based on observed 0s) and item level gamma. You can also put in your best estimate of hidden knowledge behind don't know responses.

Usage

```
group_adj(pre = NULL, pst = NULL, gamma = NULL, dk = 0.03)
```

Arguments

pre	pre data frame. Required. Each vector within the data frame should only take values 0, 1, and 'd'.
pst	pst data frame. Required. Each vector within the data frame should only take values 0, 1, and 'd'.
gamma	probability of getting the right answer without knowledge
dk	Numeric. Between 0 and 1. Hidden knowledge behind don't know responses. Default is .03.

Value

nested list of pre and post adjusted responses, and adjusted learning estimates

Examples

```
pre_test_var <- data.frame(pre = c(1,0,0,1,"d","d",0,1,NA))
pst_test_var <- data.frame(pst = c(1,NA,1,"d",1,0,1,1,"d"))
gamma <- c(.25)
group_adj(pre_test_var, pst_test_var, gamma)
```

---

lca_adj *Person Level Adjustment*

Description

Adjusts observed 1s based on item level parameters of the LCA model. Currently only takes data with Don't Know. And treats don't know responses as true confessions on ignorance. If NAs are observed in the data, they are treating as acknowledgments of ignorance.

Usage

```
lca_adj(pre = NULL, pst = NULL)
```

Arguments

pre	pre data frame
pst	pst data frame

Value

list of pre and post adjusted responses

Examples

```
pre_test_var <- data.frame(pre = c(1, 0, 0, 1, "d", "d", 0, 1, NA))
pst_test_var <- data.frame(pst = c(1, NA, 1, "d", 1, 0, 1, 1, "d"))
lca_adj(pre_test_var, pst_test_var)
```

lca_cor *Calculate item level and aggregate learning*

Description

guesstimate

Usage

```
lca_cor(
  transmatrix = NULL,
  nodk_priors = c(0.3, 0.1, 0.1, 0.25),
  dk_priors = c(0.3, 0.1, 0.2, 0.05, 0.1, 0.1, 0.05, 0.25)
)
```

Arguments

transmatrix	transition matrix returned from multi_transmat
nodk_priors	Optional. Vector of length 4. Priors for the parameters for model that fits data without Don't Knows
dk_priors	Optional. Vector of length 8. Priors for the parameters for model that fits data with Don't Knows

Value

list with two items: parameter estimates and estimates of learning

Examples

```
# Without DK
pre_test <- data.frame(item1 = c(1, 0, 0, 1, 0), item2 = c(1, NA, 0, 1, 0))
pst_test <- pre_test + cbind(c(0, 1, 1, 0, 0), c(0, 1, 0, 0, 1))
transmatrix <- multi_transmat(pre_test, pst_test)
res <- lca_cor(transmatrix)
```

lca_fit *Fit LCA model from individual-level data*

Description

Convenience wrapper: creates transition matrix and fits model.

Usage

```
lca_fit(pre_test, pst_test, ...)
```

Arguments

`pre_test`	data.frame of pre-test responses
`pst_test`	data.frame of post-test responses
`...`	passed to lca_cor()

Value

output from lca_cor()

`lca_irt` *Estimate LCA model with IRT difficulty parameterization*

Description

Fits an LCA model where item difficulty is parameterized using IRT-style difficulty parameters instead of raw gamma (guessing probability). This allows difficulty to be unbounded on the real line, which can improve optimization and makes difficulty parameters more interpretable.

Usage

```
lca_irt(
  transmatrix = NULL,
  base_rate = 0.25,
  nodk_priors = c(0.35, 0.3, 0.35, 0),
  dk_priors = c(0.25, 0.15, 0.1, 0.1, 0.15, 0.1, 0.15, 0)
)
```

Arguments

`transmatrix`	Transition matrix returned from `multi_transmat`
`base_rate`	Numeric. Minimum guessing probability (random chance). Default 0.25 (1/4 for 4-choice items). This is the floor for gamma when difficulty $\to +\infty$.
`nodk_priors`	Optional. Vector of length 4. Starting values for (gg, gk, kk, difficulty). First 3 must sum to 1.
`dk_priors`	Optional. Vector of length 8. Starting values for DK model. First 7 must sum to 1.

Details

IRT-Parameterized LCA Estimation

The relationship between difficulty (d) and gamma is:

$$\gamma = base\_rate + (1 - base\_rate) \cdot logistic(-d)$$

Where logistic(x) = 1/(1+exp(-x)). This means:

- d = 0: gamma = base_rate + 0.5*(1-base_rate) (middle difficulty)
- d → +∞: gamma → base_rate (hard item, random guessing)
- d → -∞: gamma → 1 (easy item, always correct even when guessing)

Value

A guess_fit object with additional components:

`params`	Parameter matrix with "difficulty" row instead of "gamma"
`gamma`	Derived gamma values from difficulty (added for convenience)
`learning`	Learning estimates (gk or gk + kd)

Examples

```
# Simulate data with known difficulty
sim <- simulate_lca(n = 500, n_items = 3, difficulty = c(1, 0, -1), seed = 123)
transmatrix <- multi_transmat(sim$pre, sim$post)

# Fit with IRT parameterization
fit_irt <- lca_irt(transmatrix)
fit_irt$params["difficulty", ]  # Should recover approximately c(1, 0, -1)
fit_irt$gamma                   # Derived gamma values
```

`lca_se` *Bootstrapped standard errors of effect size estimates*

Description

Bootstrapped Standard Errors

Usage

```
lca_se(
  pre_test = NULL,
  pst_test = NULL,
  n_resamples = 100,
  seed = 31415,
  force9 = FALSE
)
```

Arguments

`pre_test`	data.frame carrying pre_test items
`pst_test`	data.frame carrying pst_test items
`n_resamples`	number of resamples, default is 100
`seed`	random seed, default is 31415
`force9`	Optional. Force 9-column format even if no DK responses. Default is FALSE.

Value

list with:

`se_params`	standard errors of parameters by item
`avg_effects`	mean learning estimates
`se_effects`	standard error of learning by item

Examples

```
pre_test <- data.frame(pre_item1 = c(1, 0, 0, 1, 0), pre_item2 = c(1, NA, 0, 1, 0))
pst_test <- data.frame(
  pst_item1 = pre_test[, 1] + c(0, 1, 1, 0, 0),
  pst_item2 = pre_test[, 2] + c(0, 1, 0, 0, 1)
)
## Not run: lca_se(pre_test, pst_test, n_resamples = 10, seed = 31415)
```

`log_likelihood` *Calculate log-likelihood for transition data*

Description

Calculate log-likelihood for transition data

Usage

```
log_likelihood(params, data)
```

Arguments

`params`	numeric vector of length 4 (nodk) or 8 (dk)
`data`	numeric vector of transition counts

Value

scalar log-likelihood

Examples

```
params <- c(0.4, 0.3, 0.3, 0.25)
data <- c(x00 = 10, x01 = 5, x10 = 3, x11 = 12)
log_likelihood(params, data)
```

`multi_transmat` *Creates a transition matrix for each item.*

Description

Needs an 'interleaved' dataframe (see interleave function). Pre-test item should be followed by corresponding post-item item etc. Don't knows must be coded as NA. Function handles items without don't know responses. The function is used internally. It calls transmat.

Usage

```
multi_transmat(
  pre_test = NULL,
  pst_test = NULL,
  subgroup = NULL,
  force9 = FALSE,
  agg = FALSE
)
```

Arguments

`pre_test`	Required. data.frame carrying responses to pre-test questions.
`pst_test`	Required. data.frame carrying responses to post-test questions.
`subgroup`	a Boolean vector identifying the subset. Default is NULL.
`force9`	Optional. There are cases where DK data doesn't have DK. But we need the entire matrix. By default it is FALSE.
`agg`	Optional. Boolean. Whether or not to add a row of aggregate transitions at the end of the matrix. Default is FALSE.

Details

multi_transmat: transition matrix of all the items

Value

matrix with rows = total number of items + 1 (last row contains aggregate distribution across items) number of columns = 4 when no don't know, and 9 when there is a don't know option

Examples

```
pre_test <- data.frame(pre_item1 = c(1,0,0,1,0), pre_item2 = c(1,NA,0,1,0))
pst_test <- data.frame(pst_item1 = pre_test[,1] + c(0,1,1,0,0),
             pst_item2 = pre_test[,2] + c(0,1,0,0,1))
multi_transmat(pre_test, pst_test)
```

`nona` *No NAs*

Description

Converts NAs to 0s

Usage

```
nona(vec = NULL)
```

Arguments

`vec`	Required. Character or Numeric vector.

Value

Character vector.

Examples

```
x <- c(NA, 1, 0); nona(x)
x <- c(NA, "dk", 0); nona(x)
```

`perplexity_individuals` *Calculate perplexity from individual-level data*

Description

Calculate perplexity from individual-level data

Usage

```
perplexity_individuals(lca_result, pre_test, pst_test, per_individual = FALSE)
```

Arguments

`lca_result`	output from lca_cor() or lca_fit()
`pre_test`	data.frame of pre-test responses
`pst_test`	data.frame of post-test responses
`per_individual`	logical; return per-individual perplexity?

Value

numeric scalar or vector

perplexity_items *Calculate perplexity from aggregated item data*

Description

Lower perplexity indicates better model fit.

Usage

```
perplexity_items(lca_result, transmatrix, item = NULL)
```

Arguments

lca_result	output from lca_cor() or numeric parameter vector
transmatrix	numeric matrix of transition counts (items x cells)
item	optional integer; specific item index (NULL = aggregate)

Value

numeric scalar perplexity

Examples

```
## Not run:
transmatrix <- multi_transmat(pre_test, pst_test)
res <- lca_cor(transmatrix)
perplexity_items(res, transmatrix)

## End(Not run)
```

posterior_class_probs *Compute posterior class probabilities*

Description

Uses Bayes' rule to compute P(class | data) for each individual. The LCA model uses the joint transition structure across all items to separate true learning from lucky guessing.

Usage

```
posterior_class_probs(lca_result, pre_test, pst_test)
```

Arguments

`lca_result`	output from lca_cor() or lca_fit()
`pre_test`	data.frame of pre-test responses
`pst_test`	data.frame of post-test responses

Value

data.frame with columns P_gg, P_gk, P_kk (rows = individuals)

Examples

```
sim <- simulate_lca(n = 100, gk = 0.30, seed = 123, return_classes = TRUE)
fit <- lca_fit(sim$pre, sim$post)
posteriors <- posterior_class_probs(fit, sim$pre, sim$post)
head(posteriors)
```

`posterior_learned` *Compute posterior probability of learning*

Description

Returns P(gk | data) for each individual, representing the probability that the individual truly learned (vs. guessing or already knowing).

Usage

```
posterior_learned(lca_result, pre_test, pst_test)
```

Arguments

`lca_result`	output from lca_cor() or lca_fit()
`pre_test`	data.frame of pre-test responses
`pst_test`	data.frame of post-test responses

Value

numeric vector of P(learned | data) for each individual

Examples

```
sim <- simulate_lca(n = 100, gk = 0.30, seed = 123, return_classes = TRUE)
fit <- lca_fit(sim$pre, sim$post)
p_learned <- posterior_learned(fit, sim$pre, sim$post)
cor(p_learned, sim$learned)
```

print.guess_cv *Print method for guess_cv*

Description

Print method for guess_cv

Usage

```
## S3 method for class 'guess_cv'
print(x, ...)
```

Arguments

x	guess_cv object
...	ignored

Value

invisible(x)

print.guess_fit *Print method for guess_fit*

Description

Print method for guess_fit

Usage

```
## S3 method for class 'guess_fit'
print(x, ...)
```

Arguments

x	guess_fit object
...	ignored

Value

invisible(x)

`simulate_lca` *Simulation Functions for LCA Models*

Description

Functions to generate simulated pre/post test data from known LCA parameters for validation and parameter recovery studies. Simulate Pre-Post Test Data (No DK Model)

Usage

```
simulate_lca(
  n,
  n_items = 1,
  gg = 0.35,
  gk = 0.3,
  kk = 0.35,
  gamma = 0.25,
  difficulty = NULL,
  base_rate = 0.25,
  seed = NULL,
  return_classes = FALSE
)
```

Arguments

`n`	Integer. Number of individuals to simulate.
`n_items`	Integer. Number of test items. Default 1.
`gg`	Numeric. Proportion in guess->guess state (stable ignorance). Default 0.35.
`gk`	Numeric. Proportion in guess->know state (LEARNED). Default 0.30.
`kk`	Numeric. Proportion in know->know state (stable knowledge). Default 0.35.
`gamma`	Numeric. Probability of guessing correctly. Can be scalar (same for all items) or vector of length n_items. Default 0.25.
`difficulty`	Numeric vector. Optional IRT difficulty parameters. If provided, gamma is computed as base_rate + (1 - base_rate) * plogis(-difficulty). Higher difficulty = harder item (lower gamma). Ignored if NULL.
`base_rate`	Numeric. Minimum guessing probability (random chance). Used when difficulty is specified. Default 0.25 (1/4 for 4-choice items).
`seed`	Optional integer. Random seed for reproducibility.
`return_classes`	Logical. If TRUE, also return true latent class assignments. Default FALSE for backward compatibility.

Details

Generates simulated pre/post test data from a latent class model with known parameters. Useful for parameter recovery validation studies.

The model simulates three latent classes: - **gg (guess->guess)**: Don't know at both times. Responses are random guesses. - **gk (guess->know)**: Learned between tests. Random guess pre, correct post. - **kk (know->know)**: Know at both times. Correct responses at both times.

Parameters must satisfy: gg + gk + kk = 1 (constraint enforced automatically).

When difficulty is specified, gamma values are derived using an IRT-like transformation: gamma_i = base_rate + (1 - base_rate) * plogis(-difficulty_i). This means: - difficulty = 0: gamma = base_rate + 0.5 * (1 - base_rate) (middle) - difficulty → +∞: gamma → base_rate (hard item, random guessing) - difficulty → -∞: gamma → 1 (easy item, always correct)

Value

List with components:

`pre`	Data frame of pre-test responses (0/1 for each item)
`post`	Data frame of post-test responses (0/1 for each item)
`true_class`	(If return_classes=TRUE) Factor with levels "gg", "gk", "kk"
`learned`	(If return_classes=TRUE) Logical vector: TRUE if individual is in gk class

Examples

```
# Simulate data with 30% learning
sim <- simulate_lca(n = 500, gg = 0.35, gk = 0.30, kk = 0.35, gamma = 0.25, seed = 123)
fit <- lca_fit(sim$pre, sim$post)
fit$params["gk", ]  # Should be close to 0.30

# Multi-item simulation
sim_multi <- simulate_lca(n = 500, n_items = 3, seed = 456)

# Item-specific gamma (vector)
sim_vec <- simulate_lca(n = 500, n_items = 3, gamma = c(0.2, 0.25, 0.3), seed = 789)

# IRT-style difficulty parameters
sim_irt <- simulate_lca(n = 500, n_items = 3, difficulty = c(1, 0, -1), seed = 101)

# Return true class assignments for validation
sim_classes <- simulate_lca(n = 500, gk = 0.30, seed = 123, return_classes = TRUE)
table(sim_classes$true_class)
mean(sim_classes$learned)  # Should be close to 0.30
```

`simulate_lca_dk` *Simulate Pre-Post Test Data (DK Model)*

Description

Generates simulated pre/post test data from a latent class model with Don't Know responses.

Usage

```
simulate_lca_dk(
  n,
  n_items = 1,
  gg = 0.25,
  gk = 0.15,
  gd = 0.1,
  kg = 0.1,
  kk = 0.15,
  kd = 0.1,
  dd = 0.15,
  gamma = 0.25,
  difficulty = NULL,
  base_rate = 0.25,
  seed = NULL
)
```

Arguments

`n`	Integer. Number of individuals to simulate.
`n_items`	Integer. Number of test items. Default 1.
`gg`	Numeric. Proportion: guess->guess (stable ignorance). Default 0.25.
`gk`	Numeric. Proportion: guess->know (learned). Default 0.15.
`gd`	Numeric. Proportion: guess->dk. Default 0.10.
`kg`	Numeric. Proportion: know->guess (forgot). Default 0.10.
`kk`	Numeric. Proportion: know->know (stable knowledge). Default 0.15.
`kd`	Numeric. Proportion: know->dk. Default 0.10.
`dd`	Numeric. Proportion: dk->dk. Default 0.15.
`gamma`	Numeric. Probability of guessing correctly. Can be scalar (same for all items) or vector of length n_items. Default 0.25.
`difficulty`	Numeric vector. Optional IRT difficulty parameters. If provided, gamma is computed as base_rate + (1 - base_rate) * plogis(-difficulty). Higher difficulty = harder item (lower gamma). Ignored if NULL.
`base_rate`	Numeric. Minimum guessing probability (random chance). Used when difficulty is specified. Default 0.25 (1/4 for 4-choice items).
`seed`	Optional integer. Random seed for reproducibility.

Details

The DK model has 7 latent classes representing transitions between guess (g), know (k), and don't know (d) states: - **gg**: guess both times - **gk**: guess -> know (learned) - **gd**: guess -> dk - **kg**: know -> guess (forgot) - **kk**: know -> know - **kd**: know -> dk - **dd**: dk -> dk

Parameters must sum to 1 (constraint enforced automatically).

When difficulty is specified, gamma values are derived using an IRT-like transformation: gamma_i = base_rate + (1 - base_rate) * plogis(-difficulty_i).

Value

List with two data frames:

`pre`	Pre-test responses (character: "0", "1", or "d")
`post`	Post-test responses (character: "0", "1", or "d")

Examples

```
# Simulate DK data
sim <- simulate_lca_dk(n = 500, gk = 0.15, seed = 123)
fit <- lca_fit(sim$pre, sim$post)
fit$params["gk", ]  # Should be close to 0.15

# Item-specific gamma (vector)
sim_vec <- simulate_lca_dk(n = 500, n_items = 3, gamma = c(0.2, 0.25, 0.3), seed = 456)

# IRT-style difficulty parameters
sim_irt <- simulate_lca_dk(n = 500, n_items = 3, difficulty = c(1, 0, -1), seed = 789)
```

`stnd_cor` *Standard Guessing Correction for Learning*

Description

Estimate of learning adjusted with standard correction for guessing. Correction is based on number of options per question. The function takes separate pre-test and post-test dataframes. Why do we need dataframes? To accomodate multiple items. The items can carry NA (missing). Items must be in the same order in each dataframe. Assumes that respondents are posed same questions twice. The function also takes a `lucky` vector — the chance of getting a correct answer if guessing randomly. Each entry is 1/(number of options). The function also optionally takes a vector carrying names of the items. By default, the vector carrying adjusted learning estimates takes same item names as the pre_test items. However you can assign a vector of names separately via `item_names`.

Usage

```
stnd_cor(pre_test = NULL, pst_test = NULL, lucky = NULL, item_names = NULL)
```

Arguments

pre_test	Required. data.frame carrying responses to pre-test questions.
pst_test	Required. data.frame carrying responses to post-test questions.
lucky	Required. A vector. Each entry is 1/(number of options)
item_names	Optional. A vector carrying item names.

Value

a list of three vectors, carrying pre-treatment corrected scores, post-treatment scores, and adjusted estimates of learning

Examples

```
# Without DK
pre_test <- data.frame(item1 = c(1,0,0,1,0), item2 = c(1,NA,0,1,0))
pst_test <- pre_test + cbind(c(0,1,1,0,0), c(0,1,0,0,1))
lucky <- rep(.25, 2); stnd_cor(pre_test, pst_test, lucky)
# With DK
pre_test <- data.frame(item1 = c(1,0,0,1,0,'d',0), item2 = c(1,NA,0,1,0,'d','d'))
pst_test <- data.frame(item1 = c(1,0,0,1,0,'d',1), item2 = c(1,NA,0,1,0,1,'d'))
lucky <- rep(.25, 2); stnd_cor(pre_test, pst_test, lucky)
```

---

summary.guess_cv *Summary method for guess_cv*

Description

Summary method for guess_cv

Usage

```
## S3 method for class 'guess_cv'
summary(object, ...)
```

Arguments

object	guess_cv object
...	ignored

Value

invisible summary

summary.guess_fit *Summary method for guess_fit*

Description

Summary method for guess_fit

Usage

```
## S3 method for class 'guess_fit'
summary(object, ...)
```

Arguments

object	guess_fit object
...	ignored

Value

invisible summary object

transmat *transmat: Cross-wave transition matrix*

Description

Prints Cross-wave transition matrix and returns the vector behind the matrix. Missing values are treated as ignorance. Don't know responses need to be coded as 'd'.

Usage

```
transmat(pre_test_var, pst_test_var, subgroup = NULL, force9 = FALSE)
```

Arguments

pre_test_var	Required. A vector carrying pre-test scores of a particular item. Only
pst_test_var	Required. A vector carrying post-test scores of a particular item
subgroup	Optional. A Boolean vector indicating rows of the relevant subset.
force9	Optional. There are cases where DK data doesn't have DK. But we need the entire matrix. By default it is FALSE.

Value

a numeric vector. Assume 1 denotes correct answer, 0 and NA incorrect, and d 'don't know.' When there is no don't know option and no missing, the entries are: x00, x10, x01, x11 When there is a don't know option, the entries of the vector are: x00, x10, xd0, x01, x11, xd1, xd0, x1d, xdd

Examples

```
pre_test_var <- c(1,0,0,1,0,1,0)
pst_test_var <- c(1,0,1,1,0,1,1)
transmat(pre_test_var, pst_test_var)

# With NAs
pre_test_var <- c(1,0,0,1,"d","d",0,1,NA)
pst_test_var <- c(1,NA,1,"d",1,0,1,1,"d")
transmat(pre_test_var, pst_test_var)
```

---

validate_compatible_dataframes

Validate that two data frames have compatible dimensions

Description

Validate that two data frames have compatible dimensions

Usage

```
validate_compatible_dataframes(pre_test, pst_test)
```

Arguments

pre_test	pre-test data frame
pst_test	post-test data frame

Value

TRUE if valid, throws error otherwise

---

validate_gamma

Validate gamma parameter

Description

Validate gamma parameter

Usage

```
validate_gamma(gamma)
```

Arguments

gamma	probability parameter

Value

TRUE if valid, throws error otherwise

`validate_lucky_vector` *Validate lucky vector for standard correction*

Description

Validate lucky vector for standard correction

Usage

```
validate_lucky_vector(lucky, n_items)
```

Arguments

`lucky`	vector of guessing probabilities
`n_items`	number of items to validate against

Value

TRUE if valid, throws error otherwise

`validate_priors` *Validate prior parameters*

Description

Validate prior parameters

Usage

```
validate_priors(priors, expected_length, param_name)
```

Arguments

`priors`	vector of prior parameters
`expected_length`	expected length of priors vector
`param_name`	name of parameter for error messages

Value

TRUE if valid, throws error otherwise

`validate_recovery` *Validate Parameter Recovery via Monte Carlo Simulation*

Description

Performs Monte Carlo simulations to assess parameter recovery of the LCA model. Useful for validating estimator performance.

Usage

```
validate_recovery(true_params, n = 500, n_items = 2, n_sims = 100, seed = NULL)
```

Arguments

`true_params`	Named numeric vector of true parameters. For no-DK model: c(gg=, gk=, kk=, gamma=) For DK model: c(gg=, gk=, gd=, kg=, kk=, kd=, dd=, gamma=)
`n`	Integer. Sample size per simulation. Default 500.
`n_items`	Integer. Number of items. Default 2.
`n_sims`	Integer. Number of Monte Carlo simulations. Default 100.
`seed`	Optional integer. Random seed for reproducibility.

Value

Data frame with one row per parameter containing columns: parameter (name), true_value, mean_estimate, bias (mean estimate minus true), rmse (root mean squared error), se (standard deviation of estimates), and coverage_95 (proportion of times 95

Examples

```
## Not run:
# Validate no-DK model recovery
results <- validate_recovery(
  c(gg = 0.35, gk = 0.30, kk = 0.35, gamma = 0.25),
  n = 500, n_sims = 50
)
print(results)

# Validate DK model recovery
results_dk <- validate_recovery(
  c(gg = 0.25, gk = 0.15, gd = 0.10, kg = 0.10,
    kk = 0.15, kd = 0.10, dd = 0.15, gamma = 0.25),
  n = 500, n_sims = 50
)

## End(Not run)
```

`validate_transition_values` *Validate transition matrix values*

Description

Validate transition matrix values

Usage

```
validate_transition_values(pre_test_var, pst_test_var)
```

Arguments

`pre_test_var`	pre-test variable vector
`pst_test_var`	post-test variable vector

Value

TRUE if valid, throws error otherwise

Index