

Package: guess (via r-universe)

September 28, 2026

Title Adjust Estimates of Learning for Guessing

Version 0.8.0

Description Provides tools to adjust estimates of learning for guessing-related bias in educational and survey research. Implements standard guessing correction methods and latent class models that leverage 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 (2016) <<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 (>= 2.0.1), stats, checkmate

License MIT + file LICENSE

VignetteBuilder knitr

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

Encoding UTF-8

Language en-US

Config/testthat/edition 3

Config/roxygen2/version 8.0.0

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

Date/Publication 2026-09-28 02:01:39 UTC

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

RemoteRef HEAD

RemoteSha e261d766c25e35d285348ef74cede9876e901560

Contents

assess_item_lca_fit	2
calculate_expected_values	4
coef.guess_fit	4
count_item_transitions	5
count_transitions	6
cross_sectional_learning	6
cross_sectional_learning_score	7
cv_individual_lca	8
estimate_logit_score	9
fit_item_lca	10
fit_item_lca_counts	11
fit_person_lca	12
format_transition_matrix	13
group_adj	14
lca_adj	15
lca_se	16
nona	17
posterior_class_probs	18
posterior_learned	18
print.guess_cv	19
print.guess_fit	19
print.guess_gof	20
score_individual_lca	20
score_item_lca	21
simulate_lca	22
simulate_lca_dk	24
stnd_cor	25
summary.guess_cv	26
summary.guess_fit	27
validate_compatible_dataframes	28
validate_gamma	28
validate_guessing_probability	29
validate_priors	29
validate_recovery	30
validate_transition_values	31

Index	32
--------------	-----------

assess_item_lca_fit	<i>Assess goodness of fit for an item-level latent-class model</i>
---------------------	--

Description

Computes per-item Pearson diagnostics between paired observed responses and the cell counts implied by a [fit_item_lca()] or [fit_item_lca_counts()] result. It is an in-sample specification diagnostic, not a measure of held-out predictive performance. Use [score_item_lca()] with independent transition counts or [cv_individual_lca()] for held-out evaluation.

The binary model is saturated: its three free cell probabilities are exactly accounted for by its three free parameters, so it has zero degrees of freedom and no p-value. The don't-know model has one remaining over-identifying restriction and therefore one degree of freedom.

Usage

```
assess_item_lca_fit(
  fit,
  pre_test,
  post_test,
  ...,
  na_as = c("dk", "missing"),
  missing_action = c("omit", "error")
)
```

Arguments

fit	A 'guess_fit' object returned by [fit_item_lca()] or [fit_item_lca_counts()].
pre_test	Data frame containing one pre-test item per column.
post_test	Data frame containing the corresponding post-test items.
...	Must be empty. Its presence requires optional arguments to be named.
na_as	Classification of 'NA' responses: "dk" treats them as observed don't-know responses and "missing" treats them as structural missingness.
missing_action	How to handle structural missingness: "omit" excludes incomplete pairs and "error" rejects them.

Value

A 'guess_gof' object with 'statistics', a data frame containing the unrounded Pearson statistic, degrees of freedom, p-value, and observation count for each item; plus 'observed', 'expected', and 'residuals' matrices.

References

Pearson, K. (1900). On the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling. **Philosophical Magazine**, 50(302), 157–175. doi:10.1080/14786440009463897.

Cor, M. K., and Sood, G. (2016). Guessing and Forgetting: A Latent Class Model for Measuring Learning. **Political Analysis**, 24(2), 226–242.

Examples

```
sim <- simulate_lca_dk(n = 500, n_items = 2, seed = 123)
fit <- fit_item_lca(sim$pre, sim$post)
assess_item_lca_fit(fit, sim$pre, sim$post)
```

```
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

count_item_transitions

Count transitions for matched test items

Description

Apply [count_transitions()] to each named item shared by pre-test and post-test data. The response-cell schema is inferred after applying 'na_as'.

Usage

```
count_item_transitions(
  pre_test,
  post_test,
  ...,
  subgroup = NULL,
  include_aggregate = FALSE,
  na_as = c("dk", "missing"),
  missing_action = c("omit", "error")
)
```

Arguments

pre_test	Data frame containing one pre-test item per column.
post_test	Data frame containing the corresponding post-test items.
...	Must be empty. Its presence requires optional arguments to be named.
subgroup	Optional logical vector selecting observations to count.
include_aggregate	Whether to append an 'aggregate' row containing column sums across items.
na_as	Classification of 'NA' responses: "dk" treats them as observed don't-know responses and "missing" treats them as structural missingness.
missing_action	How to handle structural missingness: "omit" excludes incomplete pairs and "error" rejects them.

Value

A visible integer matrix with items in rows and inferred transition cells in columns.

count_transitions	<i>Count cross-wave response transitions</i>
-------------------	--

Description

Count paired transitions between pre-test and post-test item responses. The response-cell schema is inferred after applying 'na_as': binary data use four cells and observed don't-know responses use nine.

Usage

```
count_transitions(
  pre_test,
  post_test,
  ...,
  subgroup = NULL,
  na_as = c("dk", "missing"),
  missing_action = c("omit", "error")
)
```

Arguments

pre_test	Required vector of pre-test item responses.
post_test	Required vector of post-test item responses.
...	Must be empty. Its presence requires optional arguments to be named.
subgroup	Optional logical vector selecting observations to count.
na_as	Classification of 'NA' responses: "dk" treats them as observed don't-know responses and "missing" treats them as structural missingness.
missing_action	How to handle structural missingness: "omit" excludes incomplete pairs and "error" rejects them.

Value

A visible named integer vector of binary or don't-know transition cells in canonical order.

cross_sectional_learning	<i>Cross-sectional learning estimate</i>
--------------------------	--

Description

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

Usage

```
cross_sectional_learning(
  pre_test,
  post_test,
  na_as = c("dk", "missing"),
  missing_action = c("omit", "error")
)
```

Arguments

pre_test	data.frame of pre-test responses
post_test	data.frame of post-test responses
na_as	classification of NA responses
missing_action	structural missingness handling

Value

numeric vector of learning scores (post - 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)
```

cross_sectional_learning_score
Cross-Sectional Learning Score

Description

Applies the logistic function to the difference in cross-sectional logit scores. The result is bounded in $[0, 1]$, but is not a calibrated probability of learning and is not an IRT estimate.

Usage

```
cross_sectional_learning_score(
  pre_test,
  post_test,
  scale = 1,
  na_as = c("dk", "missing"),
  missing_action = c("omit", "error")
)
```

Arguments

pre_test	data.frame of pre-test responses
post_test	data.frame of post-test responses
scale	numeric scaling factor for the score difference (default 1)
na_as	classification of NA responses
missing_action	structural missingness handling

Value

numeric vector of learning scores in [0, 1]

Examples

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

cv_individual_lca	<i>Cross-validate an item LCA fit over individuals</i>
-------------------	--

Description

Splits respondents into ‘k’ folds, fits each model on the remaining respondents, and scores the held-out respondents. Each fold must fit and converge; a failed fold stops with its fold number rather than silently changing the estimand by omitting its held-out responses.

Usage

```
cv_individual_lca(
  pre_test,
  post_test,
  ...,
  k = 5L,
  seed = NULL,
  start = NULL,
  control = list(),
  na_as = c("dk", "missing"),
  missing_action = c("omit", "error")
)
```

Arguments

pre_test	Data frame containing one pre-test item per column.
post_test	Data frame containing the corresponding post-test items.
...	Must be empty.

k	Number of folds; an integer from 2 through the number of rows.
seed	Optional integer seed used only to assign folds. It does not alter the caller's random-number-generator state.
start	Optional named feasible starting vector passed to [fit_item_lca_counts()].
control	List passed to the 'control' argument of [fit_item_lca_counts()].
na_as	Classification of 'NA' responses: "dk" treats them as observed don't-know responses and "missing" treats them as structural missingness.
missing_action	How to handle structural missingness: "omit" excludes incomplete pairs and "error" rejects them.

Value

A 'guess_cv' object. 'fold_results' records every fold and 'fold_id' maps each input row to its held-out fold. Aggregate likelihood and perplexity are weighted by held-out observed item pairs. No standard error is reported because fold scores are dependent training-set refits, not independent replicate estimates.

estimate_logit_score *Estimate a Cross-Sectional Logit Score*

Description

Computes the empirical logit of each person's proportion correct. The denominator-aware half-count correction keeps all-observed scores finite. This is a descriptive score, not a fitted item-response model.

Usage

```
estimate_logit_score(
  responses,
  na_as = c("dk", "missing"),
  missing_action = c("omit", "error")
)
```

Arguments

responses	data.frame of binary responses (0/1)
na_as	classification of NA responses
missing_action	structural missingness handling

Details

Observed d/DK responses are scored as incorrect in this binary correctness baseline. They remain a distinct response category in the LCA functions. The empirical-logit correction is from Cox and Snell (1989, Section 2.1.6); respondents with no observed responses receive 'NA'.

Value

numeric vector of logit scores (length = n individuals)

References

Cox, D. R., and Snell, E. J. (1989). **Analysis of Binary Data** (2nd ed.). Chapman and Hall.

Examples

```
sim <- simulate_lca(n = 100, seed = 123)
score_pre <- estimate_logit_score(sim$pre)
score_post <- estimate_logit_score(sim$post)
```

fit_item_lca

Fit item-wise latent-class models to paired responses

Description

Creates item transition counts from paired responses, then estimates the constrained latent-class model of Cor and Sood (2016) independently for each item. This is equivalent to calling [count_item_transitions()] followed by [fit_item_lca_counts()].

Usage

```
fit_item_lca(
  pre_test,
  post_test,
  ...,
  subgroup = NULL,
  include_aggregate = FALSE,
  na_as = c("dk", "missing"),
  missing_action = c("omit", "error"),
  start = NULL,
  control = list()
)
```

Arguments

pre_test	Data frame containing one pre-test item per column.
post_test	Data frame containing the corresponding post-test items.
...	Must be empty. Its presence requires optional arguments to be named.
subgroup	Optional logical vector selecting observations to fit.
include_aggregate	Whether to fit the item-total transition counts and store that fit separately in 'aggregate'.

na_as	Classification of 'NA' responses: "dk" treats them as observed don't-know responses and "missing" treats them as structural missingness.
missing_action	How to handle structural missingness: "omit" excludes incomplete pairs and "error" rejects them.
start	Optional named feasible starting vector passed to [fit_item_lca_counts()].
control	List passed to the 'control' argument of [Rsolnp::solnp()].

Value

A 'guess_fit' object. 'diagnostics' records optimizer results by item. An aggregate fit, when requested, is stored separately and excluded from item metadata.

References

Cor, M. K., and Sood, G. (2016). Guessing and Forgetting: A Latent Class Model for Measuring Learning. *Political Analysis**, 24(2), 226–242.

Galanos, A., and Ye, Y. (2025). *Rsolnp: General Non-Linear Optimization**. R package version 2.0.1.

Examples

```
sim <- simulate_lca(n = 500, n_items = 2, seed = 123)
fit_item_lca(sim$pre, sim$post)
```

fit_item_lca_counts	<i>Fit item-wise latent-class models to transition counts</i>
---------------------	---

Description

Estimate the constrained latent-class model developed by Cor and Sood (2016) independently for each item. The model is fitted with [Rsolnp::solnp()] and requires its reported convergence code to be zero.

Usage

```
fit_item_lca_counts(transition_counts, ..., start = NULL, control = list())
```

Arguments

transition_counts	Named matrix of nonnegative whole-number transition counts returned by [count_item_transitions()].
...	Must be empty. Its presence requires optional arguments to be named.
start	Optional named feasible starting vector. When 'NULL', latent class weights start uniformly and 'gamma' starts at its empirical transition-ratio estimate. A boundary estimate is moved inward by machine tolerance; an unidentified ratio starts at the admissible midpoint.
control	List passed to the 'control' argument of [Rsolnp::solnp()].

Value

A ‘guess_fit’ object. ‘diagnostics’ records optimizer results by item. If ‘transition_counts’ contains an ‘aggregate’ row, its fit is stored separately in ‘aggregate’ and excluded from item metadata.

References

Cor, M. K., and Sood, G. (2016). Guessing and Forgetting: A Latent Class Model for Measuring Learning. *Political Analysis*, 24(2), 226–242.

Galanos, A., and Ye, Y. (2025). *Rsolnp: General Non-Linear Optimization*. R package version 2.0.1.

Examples

```
pre_test <- data.frame(item1 = c(1, 0, 0, 1, 0))
post_test <- data.frame(item1 = c(1, 0, 1, 1, 0))
transition_counts <- count_item_transitions(pre_test, post_test)
fit_item_lca_counts(transition_counts)
```

fit_person_lca

Fit a joint person-level latent-class model

Description

Fits one latent transition class per person across repeated items. Class proportions are shared across items, while guessing probabilities are item-specific. Parameters are estimated jointly by expectation-maximization. This is a package-specific joint extension of the item-level response model, not the item-wise estimator developed by Cor and Sood (2016).

Usage

```
fit_person_lca(
  pre_test,
  post_test,
  ...,
  missing_action = c("omit", "error"),
  start = NULL,
  max_iterations = 1000L,
  tolerance = sqrt(.Machine$double.eps)
)
```

Arguments

pre_test	Data frame containing one binary pre-test item per column.
post_test	Data frame containing the corresponding binary post-test items. Items are paired by name, not position.
...	Must be empty. Its presence requires optional arguments to be named.

missing_action	How to handle structural missingness: "omit" excludes incomplete pairs and "error" rejects them.
start	Optional named list with 'class_priors' (named 'gg', 'gk', 'kk') and item-named 'gamma'. When 'NULL', [fit_item_lca()] supplies a deterministic item-level initialization.
max_iterations	Maximum EM iterations before an error is raised.
tolerance	Strictly positive finite convergence tolerance for the maximum absolute parameter change.

Details

The model accepts binary responses and structural missingness only. It assumes each person has one 'gg', 'gk', or 'kk' trajectory across all items and that observed item pairs are conditionally independent given that trajectory.

Value

An object of class 'guess_person_fit' containing shared class proportions, item-specific guessing probabilities, person-level posterior probabilities, log-likelihood, and convergence information. 'n_obs' is the number of observed item-response pairs.

References

- Cor, M. K., and Sood, G. (2016). Guessing and Forgetting: A Latent Class Model for Measuring Learning. *Political Analysis*, 24(2), 226–242.
- Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). Maximum Likelihood from Incomplete Data via the EM Algorithm. *Journal of the Royal Statistical Society: Series B*, 39(1), 1–38.

Examples

```
sim <- simulate_lca(n = 500, n_items = 4, seed = 123)
fit <- fit_person_lca(sim$pre, sim$post)
fit$class_priors
head(fit$posterior)
```

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,
  item_names,
  include_aggregate = FALSE
)
```

Arguments

```
transition_list      list of transition vectors
item_names           item names in output row order
include_aggregate    whether to add an 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 an item-level guessing probability. You can also put in your best estimate of hidden knowledge behind don't know responses.

Usage

```
group_adj(
  pre_test = NULL,
  post_test = NULL,
  guessing_probability = NULL,
  knowledge_given_dont_know = 0.03,
  na_as = c("dk", "missing"),
  missing_action = c("omit", "error")
)
```

Arguments

```
pre_test      Pre-test data frame. Required. Each vector within the data frame should only
               take values 0, 1, and 'd'.
post_test     Post-test data frame. Required. Each vector within the data frame should only
               take values 0, 1, and 'd'.
```

guessing_probability	Probability of getting the right answer without knowledge.
knowledge_given_dont_know	Numeric probability of hidden knowledge conditional on an observed don't-know response. Must be between 0 and 1. Defaults to 0.03.
na_as	Classification of NA responses: "dk" (the default) treats them as observed don't know responses; "missing" treats them as structural missingness.
missing_action	How to handle structural missingness: "omit" excludes it and "error" rejects it.

Value

A list with 'adjusted_responses', containing 'pre_test' and 'post_test' data frames, and 'mean_learning', the item-level mean adjusted change.

Examples

```
pre_test_var <- data.frame(item = c(1, 0, 0, 1, "d", "d", 0, 1, NA))
post_test_var <- data.frame(item = c(1, NA, 1, "d", 1, 0, 1, 1, "d"))
guessing_probability <- c(.25)
group_adj(pre_test_var, post_test_var, guessing_probability)
```

lca_adj	<i>Person Level Adjustment</i>
---------	--------------------------------

Description

Adjusts observed Is 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. By default, NAs are treated as acknowledgments of ignorance.

Usage

```
lca_adj(
  pre_test = NULL,
  post_test = NULL,
  na_as = c("dk", "missing"),
  missing_action = c("omit", "error")
)
```

Arguments

pre_test	Pre-test data frame.
post_test	Post-test data frame.
na_as	Classification of NA responses: "dk" (the default) treats them as observed don't know responses; "missing" treats them as structural missingness.
missing_action	How to handle structural missingness: "omit" excludes it and "error" rejects it.

Value

list of pre and post adjusted responses

Examples

```
pre_test <- data.frame(item = c(1, 0, 0, 1, "d", "d", 0, 1, NA))
post_test <- data.frame(item = c(1, NA, 1, "d", 1, 0, 1, 1, "d"))
lca_adj(pre_test, post_test)
```

lca_se	<i>Bootstrapped standard errors of effect size estimates</i>
--------	--

Description

Bootstrapped Standard Errors

Usage

```
lca_se(
  pre_test = NULL,
  post_test = NULL,
  n_resamples = 100,
  seed = NULL,
  na_as = c("dk", "missing"),
  missing_action = c("omit", "error")
)
```

Arguments

pre_test	data.frame carrying pre_test items
post_test	data.frame carrying post-test items
n_resamples	number of resamples, default is 100
seed	Optional integer seed. When ‘NULL’, resampling uses the current random-number-generator state.
na_as	Classification of NA responses: “dk” (the default) treats them as observed don’t know responses; “missing” treats them as structural missingness.
missing_action	How to handle structural missingness: “omit” excludes incomplete pairs and “error” rejects them.

Value

list with:

- parameter_standard_error
standard errors of parameters by item
- mean_learning
mean learning estimates
- learning_standard_error
standard error of learning by item

Examples

```
pre_test <- data.frame(item1 = c(1, 0, 0, 1, 0), item2 = c(1, NA, 0, 1, 0))
post_test <- data.frame(
  item1 = pre_test[, 1] + c(0, 1, 1, 0, 0),
  item2 = pre_test[, 2] + c(0, 1, 0, 0, 1)
)
## Not run:
lca_se(pre_test, post_test, n_resamples = 10, seed = 123)

## End(Not run)
```

nona

Normalize NA Responses

Description

Classifies NAs as observed don't know responses by default. If NAs represent structural missingness, they can instead be omitted or rejected.

Usage

```
nona(
  vec = NULL,
  na_as = c("dk", "missing"),
  missing_action = c("omit", "error")
)
```

Arguments

vec Required. Character or numeric vector.

na_as Classification of NA responses: "dk" (the default) or "missing".

missing_action How to handle structural missingness: "omit" or "error".

Value

Character vector.

Examples

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

posterior_class_probs *Compute posterior class probabilities*

Description

Extracts $P(\text{class} \mid \text{response vector})$ for each individual from an explicitly fitted person/item model.

Usage

```
posterior_class_probs(fit)
```

Arguments

fit output from fit_person_lca()

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 <- fit_person_lca(sim$pre, sim$post)
posteriors <- posterior_class_probs(fit)
head(posteriors)
```

posterior_learned *Compute posterior probability of learning*

Description

Returns $P(\text{gk} \mid \text{data})$ for each individual, representing the probability that the individual truly learned (vs. guessing or already knowing).

Usage

```
posterior_learned(fit)
```

Arguments

fit output from fit_person_lca()

Value

numeric vector of $P(\text{learned} \mid \text{data})$ for each individual

Examples

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

print.guess_cv	<i>Print method for guess_cv</i>
----------------	----------------------------------

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	<i>Print method for guess_fit</i>
-----------------	-----------------------------------

Description

Print method for guess_fit

Usage

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

Arguments

x	guess_fit object
...	ignored

Value

invisible(x)

<code>print.guess_gof</code>	<i>Print method for item-level goodness-of-fit diagnostics</i>
------------------------------	--

Description

Print method for item-level goodness-of-fit diagnostics

Usage

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

Arguments

<code>x</code>	A ‘guess_gof’ object.
<code>...</code>	Ignored.

Value

‘x’, invisibly.

<code>score_individual_lca</code>	<i>Score an item LCA fit on individual responses</i>
-----------------------------------	--

Description

Evaluates a fitted item-level latent-class model on paired individual responses. The aggregate likelihood equals `[score_item_lca()]` applied to transition counts made from the same responses.

Usage

```
score_individual_lca(
  fit,
  pre_test,
  post_test,
  ...,
  na_as = c("dk", "missing"),
  missing_action = c("omit", "error")
)
```

Arguments

<code>fit</code>	A ‘guess_fit’ object returned by <code>[fit_item_lca_counts()]</code> or <code>[fit_item_lca()]</code> .
<code>pre_test</code>	Data frame containing one pre-test item per column.
<code>post_test</code>	Data frame containing the corresponding post-test items.
<code>...</code>	Must be empty.
<code>na_as</code>	Classification of ‘NA’ responses: “dk” treats them as observed don’t-know responses and “missing” treats them as structural missingness.
<code>missing_action</code>	How to handle structural missingness: “omit” excludes incomplete pairs and “error” rejects them.

Value

A ‘guess_individual_score’ object. ‘individual_scores’ contains one row per respondent; rows with no observed item pairs have ‘NA’ mean score and perplexity. Aggregate fields are observation-weighted across all observed item pairs.

<code>score_item_lca</code>	<i>Score an item-level latent-class fit against transition counts</i>
-----------------------------	---

Description

Computes log scores and perplexity for each named item and in aggregate. The supplied counts may be the fit data or independent evaluation data; this function does not infer that provenance. Scores are only out-of-sample when the counts were not used to estimate ‘fit’.

Usage

```
score_item_lca(fit, transition_counts)
```

Arguments

<code>fit</code>	A ‘guess_fit’ object returned by <code>[fit_item_lca()]</code> or <code>[fit_item_lca_counts()]</code> .
<code>transition_counts</code>	Named matrix of nonnegative whole-number transition counts with exactly the same item names as ‘fit’.

Value

A ‘guess_item_score’ object containing ‘item_scores’, a data frame of item-level log likelihood, observation count, mean log likelihood, and perplexity; plus aggregate ‘total_log_likelihood’, ‘n_observations’, ‘mean_log_likelihood’, and ‘perplexity’.

References

Cor, M. K., and Sood, G. (2016). Guessing and Forgetting: A Latent Class Model for Measuring Learning. **Political Analysis**, 24(2), 226–242.

Examples

```
sim <- simulate_lca(n = 500, n_items = 2, seed = 123)
transition_counts <- count_item_transitions(sim$pre, sim$post)
fit <- fit_item_lca_counts(transition_counts)
score_item_lca(fit, transition_counts)
```

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 difficulty-link scores. If provided, gamma is computed as $\text{base_rate} + (1 - \text{base_rate}) * \text{plogis}(-\text{difficulty})$. 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 a logistic transformation: $\gamma_i = \text{base_rate} + (1 - \text{base_rate}) * \text{plogis}(-\text{difficulty}_i)$. This means: - difficulty = 0: $\gamma = \text{base_rate} + 0.5 * (1 - \text{base_rate})$ (middle) - difficulty -> +Inf: $\gamma \rightarrow \text{base_rate}$ (hard item, random guessing) - difficulty -> -Inf: $\gamma \rightarrow 1$ (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 <- fit_item_lca(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)

# Difficulty-link scores
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,
  kk = 0.15,
  dg = 0.1,
  dk = 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.
kk	Numeric. Proportion: know->know (stable knowledge). Default 0.15.
dg	Numeric. Proportion: dk->guess. Default 0.10.
dk	Numeric. Proportion: dk->know (learned). 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 difficulty-link scores. If provided, gamma is computed as $\text{base_rate} + (1 - \text{base_rate}) * \text{plogis}(-\text{difficulty})$. 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 - **kk**: know -> know - **dg**: dk -> guess - **dk**: dk -> know (learned) - **dd**: dk -> dk

The know -> guess and know -> dk classes are absent by design. The model is identified by the assumption that people do not lose knowledge over a short informative process, which sets both to zero. Learning is gk + dk.

Parameters must sum to 1 (constraint enforced automatically).

When difficulty is specified, gamma values are derived using a logistic transformation: $\text{gamma}_i = \text{base_rate} + (1 - \text{base_rate}) * \text{plogis}(-\text{difficulty}_i)$.

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 = 5000, gk = 0.15, seed = 123)
fit <- fit_item_lca(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)

# Difficulty-link scores
sim_irt <- simulate_lca_dk(n = 500, n_items = 3, difficulty = c(1, 0, -1), seed = 789)
```

std_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 guessing_probability 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,
  post_test = NULL,
  guessing_probability = NULL,
  item_names = NULL,
  na_as = c("dk", "missing"),
  missing_action = c("omit", "error")
)
```

Arguments

<code>pre_test</code>	Required. data.frame carrying responses to pre-test questions.
<code>post_test</code>	Required. data.frame carrying responses to post-test questions.
<code>guessing_probability</code>	Required. A vector. Each entry is 1/(number of options)
<code>item_names</code>	Optional. A vector carrying item names.
<code>na_as</code>	Classification of NA responses: "dk" (the default) treats them as observed don't know responses; "missing" treats them as structural missingness.
<code>missing_action</code>	How to handle structural missingness: "omit" excludes it and "error" rejects it.

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))
post_test <- pre_test + cbind(c(0, 1, 1, 0, 0), c(0, 1, 0, 0, 1))
guessing_probability <- rep(.25, 2)
stnd_cor(pre_test, post_test, guessing_probability)

# With DK
pre_test <- data.frame(item1 = c(1, 0, 0, 1, 0, "d", 0), item2 = c(1, NA, 0, 1, 0, "d", "d"))
post_test <- data.frame(item1 = c(1, 0, 0, 1, 0, "d", 1), item2 = c(1, NA, 0, 1, 0, 1, "d"))
guessing_probability <- rep(.25, 2)
stnd_cor(pre_test, post_test, guessing_probability)
```

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	<i>Summary method for guess_fit</i>
-------------------	-------------------------------------

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

validate_compatible_dataframes	<i>Validate that two data frames have compatible dimensions</i>
--------------------------------	---

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	<i>Validate gamma parameter</i>
----------------	---------------------------------

Description

Validate gamma parameter

Usage

```
validate_gamma(gamma)
```

Arguments

gamma	probability parameter
-------	-----------------------

Value

TRUE if valid, throws error otherwise

`validate_guessing_probability`*Validate guessing probabilities for standard correction*

Description

Validate guessing probabilities for standard correction

Usage

```
validate_guessing_probability(guessing_probability, n_items)
```

Arguments

<code>guessing_probability</code>	vector of guessing probabilities
<code>n_items</code>	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

<code>priors</code>	vector of prior parameters
<code>expected_length</code>	expected length of priors vector
<code>param_name</code>	name of parameter for error messages

Value

TRUE if valid, throws error otherwise

validate_recovery	<i>Validate Parameter Recovery via Monte Carlo Simulation</i>
-------------------	---

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 strictly interior true parameters. For no-DK model: c(gg=, gk=, kk=, gamma=) For DK model: c(gg=, gk=, gd=, kk=, dg=, dk=, 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), estimate_sd (Monte Carlo standard deviation of estimates), and mcse (Monte Carlo standard error of the mean estimate).

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, kk = 0.20,
    dg = 0.10, dk = 0.10, dd = 0.10, 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

`assess_item_lca_fit`, [2](#)

`calculate_expected_values`, [4](#)

`coef.guess_fit`, [4](#)

`count_item_transitions`, [5](#)

`count_transitions`, [6](#)

`cross_sectional_learning`, [6](#)

`cross_sectional_learning_score`, [7](#)

`cv_individual_lca`, [8](#)

`estimate_logit_score`, [9](#)

`fit_item_lca`, [10](#)

`fit_item_lca_counts`, [11](#)

`fit_person_lca`, [12](#)

`format_transition_matrix`, [13](#)

`group_adj`, [14](#)

`lca_adj`, [15](#)

`lca_se`, [16](#)

`nona`, [17](#)

`posterior_class_probs`, [18](#)

`posterior_learned`, [18](#)

`print.guess_cv`, [19](#)

`print.guess_fit`, [19](#)

`print.guess_gof`, [20](#)

`score_individual_lca`, [20](#)

`score_item_lca`, [21](#)

`simulate_lca`, [22](#)

`simulate_lca_dk`, [24](#)

`stnd_cor`, [25](#)

`summary.guess_cv`, [26](#)

`summary.guess_fit`, [27](#)

`validate_compatible_dataframes`, [28](#)

`validate_gamma`, [28](#)

`validate_guessing_probability`, [29](#)

`validate_priors`, [29](#)

`validate_recovery`, [30](#)

`validate_transition_values`, [31](#)