

Package: latents (via r-universe)

October 15, 2026

Title Multilevel Latent Profile, Class and Transition Models

Version 0.9.30

Description Provides functions for fitting multilevel latent profile analysis, latent class, and latent transition models for continuous, categorical, and mixed-type variables. Models are estimated by expectation-maximization, handle missing data, and provide standard and robust standard errors. Model selection is supported by information criteria, entropy, classification diagnostics, and a parametric bootstrap likelihood ratio test. Covariates can be related to latent class membership using three-step methods with correction for classification error. Latent transition models estimate transitions between latent profiles across ordered occasions and are compatible with the 'tna' package for network and sequence analysis. The multilevel mixture formulation follows Vermunt (2003)
<[doi:10.1111/j.0081-1750.2003.t01-1-00131.x](https://doi.org/10.1111/j.0081-1750.2003.t01-1-00131.x)>.

URL <https://pak.dynasite.org/latents/>,
<https://github.com/mohsaqr/latents>

BugReports <https://github.com/mohsaqr/latents/issues>

License GPL-3

Encoding UTF-8

Depends R (>= 4.1.0)

Imports stats, grDevices, utils

Suggests testthat (>= 3.0.0), knitr, rmarkdown, tna, mclust, mice, ggplot2, MASS, withr

VignetteBuilder knitr

Config/testthat/edition 3

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

LazyData true

Repository <https://mohsaqr.r-universe.dev>

Date/Publication 2026-10-15 17:01:09 UTC

RemoteUrl <https://github.com/mohsaqr/latents>

RemoteRef HEAD

RemoteSha b91424dd6ab4be0f94e5ab6f2948e635debc5719

Contents

bootstrap_lrt	3
compare_models	5
course_engagement	7
descriptives	9
diagnostics	11
enumerate_classes	13
enumerate_lca	15
enumerate_lpa	16
enumerate_regressions	18
fit_staged	20
get_candidate	22
get_group_tna	24
get_results	25
get_tna	34
growth_schools	35
growth_scores	36
latents-conditions	37
lca	43
lpa	44
lta	45
mixture_regression	50
multilca	55
multilpa	56
parameter_inference.multilpa_additive	65
parameter_inference.multilpa_transitions	71
plot.latents_family_enumeration	73
pool_imputations	81
prior_control	84
r3step	85
report	88
sensitivity	89
simulate.latents_growth_mixture	91
srl	93
starting_values	94
student_esm	95
study_hours	97
three_step	98
vcov.multilpa_covariates	101

bootstrap_lrt

*Parametric bootstrap likelihood-ratio comparison***Description**

Simulates indicators under the null model while preserving observed group sizes, refits both models, and compares their likelihood differences. Models must differ by exactly one individual profile or one group class, with the other count, covariance structure and centering mode fixed. Grand-mean centering is repeated in every simulated refit. Person-centred fits are refused because this model does not specify a generative distribution for the group baselines removed by that transformation. This is a native parametric bootstrap, not an implementation of Mplus TECH14. It does not use a chi-square reference distribution. Any failed/nonconverged or reversed replicate makes the p-value NA, avoiding silent deletion of difficult fits.

Usage

```
bootstrap_lrt(
  null_model,
  alternative_model,
  data = NULL,
  iter = 199L,
  n_starts = 10L,
  max_iter = 1000L,
  tol = 1e-08,
  seed = NULL
)
```

Arguments

<code>null_model</code>	Smaller, converged <code>multilpa()</code> model, with or without membership covariates.
<code>alternative_model</code>	Larger model fitted to exactly the same data.
<code>data</code>	Optional. The data both models were fitted to, used to verify both fitted likelihoods; when omitted it is rebuilt from what the null model stores.
<code>iter</code>	Number of simulated datasets (at least two; use many for inference).
<code>n_starts</code>	Number of starts for each simulated fit.
<code>max_iter</code>	Maximum EM iterations for each simulated fit.
<code>tol</code>	Relative likelihood convergence tolerance.
<code>seed</code>	Optional seed: any whole number set. <code>seed()</code> accepts. The caller's random-number state is restored on exit.

Details

Fits made with `missing = "fiml"` are supported when both models were: every replicate is given the observed data's missing cells before it is refitted, preserving the pattern of available measurements. Calibration requires missingness independent of the indicators and latent classes, conditional on any fixed covariates. Carrying a fixed mask does not reproduce a general MAR mechanism that depends on observed indicators, or an MNAR mechanism. FIML estimation under MAR does not by itself validate this bootstrap under MAR; the retained size study covers MCAR only.

Membership-covariate fits (`multilpa(profile_covariates = , group_covariates = )`) are supported: the covariates are held at their observed values, group classes are drawn from the fitted group logits and profiles from each row's profile logits, as the model conditions on the covariates. Both models must use the same profile covariates, group covariates and slope specification, except that a one-group-class null, which cannot carry group covariates or slopes by group class, may be compared with an alternative that adds them. When data is omitted it is rebuilt from the alternative, whose columns include the null's. A covariate replicate counts as valid when its likelihood has converged even if a membership logit is still drifting, as it does for an empty or separated class in an over-fitted alternative: the statistic reads only the maximized likelihood, which such a fit has reached. The replicate table's `logits_settled` column records whether both refits' logits had also converged. The original models must be fully converged.

Models fitted with `fixed`, including those from `fit_staged()`, are supported: every replicate is refitted with the same blocks held at the same values, so the simulated statistics compare the same two constrained models as the observed statistic does. The p-value is then conditional on that measurement solution, which is treated as known and whose own uncertainty is not propagated. A constrained pair whose nesting cannot be established is refused rather than given a p-value.

Value

An object of class `multilpa_bootstrap_lrt`, carrying the observed statistic, the finite-simulation corrected p-value, its Monte Carlo standard error, the measurement blocks held fixed in every fit, and one record per replicate. Read it with the verbs that describe it: `as.data.frame()` gives the single-row test result, `get_results(what = "replicates")` one row per simulated replicate, `summary()` the test beside the replicate diagnostics, and `plot()` the simulated null distribution with the observed statistic marked. Inspect failed starts, boundary flags and likelihood replication before interpreting results.

Conditions

`latents_bad_nesting` when only one of the two models holds measurement blocks fixed, when they hold different blocks, when they hold the same blocks at different values, when they hold a measurement fixed while differing in the number of profiles (the held blocks then have different shapes, so the null is not a restriction of the alternative), or when the alternative does not estimate more free parameters than the null. `latents_failed_replicates` is warned when some replicate fails validation, and the p-value is NA. `latents_unsupported_bootstrap` refuses person-centred fits, for which this simulator has no group-baseline distribution. `latents_unsupported_prior` refuses fits estimated with a prior: their likelihoods are evaluated at posterior modes, not ML estimates. `latents_incomparable_models` when the two models differ in their observations, grouping, covariance structure, centering or missing-data handling; `latents_bad_data` when the data have missing indicators and the models were not fitted with `missing = "fiml"`; `latents_bad_nesting`

also when two covariate models use different covariates or slope specifications (other than a one-group-class null).

Examples

```
set.seed(1)
example_data <- data.frame(
  school = rep(seq_len(10), each = 10),
  score = rnorm(100, rep(c(-2, 2), each = 50))
)
smaller <- multilpa(example_data, "score", "school", n_profiles = 1,
  n_group_classes = 1, n_starts = 2, seed = 1)
larger <- multilpa(example_data, "score", "school", n_profiles = 2,
  n_group_classes = 1, n_starts = 2, seed = 1)
# `iter` is small so the example runs quickly; use many more for inference.
comparison <- bootstrap_lrt(smaller, larger, iter = 9, n_starts = 1,
  max_iter = 2000, tol = 1e-6, seed = 1)

comparison
```

compare_models

Compare fitted models on the same data

Description

One row per model with its log likelihood, number of parameters, information criteria, classification quality and convergence, so alternative specifications (for example trajectory classes alone against a growth mixture model) are compared without assembling tables by hand. `delta_bic` is each model's BIC above the smallest, and `bic_weight` the Schwarz weight, $\exp(-\text{delta}/2)$ normalized over the models compared (Wagenmakers and Farrell 2004): the approximate posterior probability of each model if one of them is true, with equal prior probabilities.

Usage

```
compare_models(..., n_profiles = NULL, n_group_classes = NULL, cutoff = 0.8)
```

Arguments

<code>...</code>	Either an enumeration grid from <code>enumerate_classes()</code> followed by the candidates to compare (none for all of them, or a selection through <code>n_profiles</code> and <code>n_group_classes</code>); or two or more fits from <code>multilpa()</code> , <code>lpa()</code> , <code>lca()</code> , <code>lta()</code> or <code>get_candidate()</code> ; or two or more fits from <code>mixture_regression()</code> (with or without random). Name fits to label the rows; unnamed fits are labelled by their expression, candidates by their name.
<code>n_profiles</code> , <code>n_group_classes</code>	With a grid, the class counts of the candidates to compare, instead of naming them; <code>NULL</code> takes every count. Candidates whose fit failed are left out, with a message.
<code>cutoff</code>	For profile and class models, the average posterior probability below which a class is counted in <code>avepp_below</code> and <code>group_avepp_below</code> . Not used for mixture regressions.

Details

Information criteria are comparable only on the same data: every model must be fitted to the same outcome, persons (or groups) and observations, or the comparison is refused.

Candidates of an enumeration grid are compared by naming them, as the grid's candidate column shows them: `compare_models(grid, "classes3_groups4", "classes3_groups5")`, or by their class counts, `compare_models(grid, c(3, 4), c(3, 5))` (one number is one group class). `n_profiles` and `n_group_classes` select candidates instead: `compare_models(grid, n_profiles = 3)` compares every candidate with 3 profiles; `compare_models(grid)` compares them all. Latent class models are compared the same way; their `n_profiles` column counts classes. Fits from `multilpa()`, `lpa()`, `lca()`, `lta()` or `get_candidate()` can also be passed directly.

Value

A base data.frame of class `latents_comparison` with one row per model. For mixture regressions: `model`, `n_classes`, `random`, `n_parameters`, `log_likelihoood`, `aic`, `bic`, `sabic`, `icl`, `entropy`, `smallest_share`, `delta_bic`, `bic_weight`, `converged`. For profile and class models: `model`, `n_profiles`, `n_group_classes`, `structure` (the covariance structure, NA without continuous indicators), `n_parameters`, `log_likelihoood`, every information criterion of `enumerate_classes()`'s candidates table, `profile_entropy`, `group_entropy`, `avepp_min`, `avepp_max`, `avepp_mean`, `avepp_below` and the same four prefixed `group_`, `smallest_share` (the smallest estimated profile share, relative to all observation mass including any noise component), `converged`, `n_best_replicated`, and `delta_bic` and `bic_weight` on `bic_groups` (the BIC that counts groups as the independent units; on a single-level model it is the ordinary BIC). The numbers are the ones the grid's candidates table reports. The printout marks the row with the lowest BIC; that is a fact about the criterion, not a choice. Raises `latents_bad_argument` for fits it cannot compare, for a group-class family grid (compare those in their own table), for candidates named together with `n_profiles` or `n_group_classes`, and for `n_profiles` or `n_group_classes` without a grid; `latents_unknown_candidate` for a name or counts not in the grid; `latents_failed_candidate` for a named candidate that could not be fitted; and `latents_incomparable_models` for fits to different data. A cutoff outside (0, 1] fails the argument checks.

References

Wagenmakers, E.-J., & Farrell, S. (2004). AIC model selection using Akaike weights. *Psychonomic Bulletin & Review*, 11, 192–196.

Examples

```
# Fixed trajectories against growth curves with a random intercept,
# for the first 60 students
few_students <- subset(growth_scores, student <= 60)
trajectories <- mixture_regression(score ~ wave, few_students, n_classes = 2,
                                  id = "student", class_level = "group",
                                  seed = 1)
growth <- mixture_regression(score ~ wave, few_students, n_classes = 2,
                             id = "student", class_level = "group",
                             random = "intercept", seed = 1)
compare_models(trajectories = trajectories, growth = growth)
```

```
# Candidates of an enumeration grid, by name, by class counts, or selected
grid <- enumerate_classes(subset(course_engagement, student <= 40),
  c("browse", "lectures"), "student",
  n_profiles = 2:3, n_group_classes = 1:2,
  model = "VVI", n_starts = 2, seed = 1)
compare_models(grid, "profiles2_groups2", "profiles3_groups2")
compare_models(grid, c(2, 2), c(3, 2))
compare_models(grid, n_group_classes = 2)
```

course_engagement

Student engagement across a sequence of courses

Description

A simulated dataset with the structure this package exists for, and all of it in one place: enrolments nested in students, ordered within each student, with a covariate that predicts which pattern an enrolment follows.

Usage

```
course_engagement
```

Format

A data frame with 1422 rows (106 students, 32 courses, up to 15 enrolments each) and 11 columns:

student Integer student identifier, 1 to 106. The nesting unit; pass it as `id`.

course Integer course identifier, 1 to 32. Which course the enrolment was in.

sequence Integer, the position of this course in the student's own order, 1 upwards. Pass it as `time`. Students who took fewer courses have shorter sequences, so the panel is ragged.

browse Numeric, course page views on the $\log_{1p}(\text{events})$ scale, standardized within course.

lectures Numeric, lecture videos viewed on the $\log_{1p}(\text{events})$ scale, standardized within course.

forum_read Numeric, forum posts read on the $\log_{1p}(\text{events})$ scale, standardized within course.

forum_post Numeric, forum posts written on the $\log_{1p}(\text{events})$ scale, standardized within course.

attendance Numeric, days the student was active in the course as on the $\log_{1p}(\text{events})$ scale, standardized within course. It is **generated from the four click measures** and then blurred, because a student is recorded present on a day precisely because they clicked something, and a day is a coarse unit – one tick however much happened inside it. The indicators are therefore locally dependent on purpose: attendance correlates 0.51 to 0.67 with the click measures overall and 0.16 to 0.26 within a single profile, which is more association than a shared profile alone would produce. See the note on local dependence below.

previous_grade Numeric, the grade the student earned in the course *before* this one, standardised across the cohort. At a student's first course an entry grade stands in its place. It is a predictor of engagement here, not an outcome of it: pass it as `profile_covariates` to `multilpa(profile_covariates = )`, or as the predictor to `r3step()`.

engagement Factor, "disengaged" or "engaged", the pattern the row was generated from. This is the truth behind the data, not something a study would observe; it is shipped so a fitted model can be checked against what produced it. It is deliberately not called `profile`: `get_results(fit, "assignments")` adds a column of that name, and a dataset already carrying it would make the verb refuse to join its own output.

student_type Factor, "committed" or "wavering", the kind of student the row's student was generated as. The truth behind the group classes, shipped for the same reason.

Details

Each row is one student's enrolment in one course. A student takes up to fifteen courses, and sequence orders them, so the same rows support a cross-sectional two-level model (`id = "student"`), a latent transition model (`time = "sequence"`), and a membership-covariate model (`profile_covariates = "previous_grade"`).

Two kinds of student were generated. Both contain both patterns of engagement; they differ in the *mix*, which is the two-level quantity this package estimates. A committed student is engaged in about four enrolments in five, a wavering student in about one in four. Pooled across the sample the engaged share describes neither kind, which is the point: recovering the split is what a two-level model does and a pooled one cannot.

Local independence does not hold here, on purpose

A latent profile model with a diagonal residual covariance assumes the indicators are independent given the profile. `attendance` is built from the click measures, so for these data that assumption is false, and the package finds it: `get_results(fit, "residuals")` reports a largest residual of 0.21 $p = 1.7e-15$, and `covariance_model = "full"` improves the log likelihood from -7618.5 to -7494.7 for twenty more parameters, which every information criterion prefers.

This is deliberate. A bundled dataset that satisfies local independence exactly can only ever show those verbs agreeing that nothing is wrong; here the diagnostic has a real violation to find, its cause is known, and the fix can be seen to work. One consequence is worth knowing before it surprises you: under a diagonal model `enumerate_classes()` keeps preferring more classes, because extra classes absorb residual association the model is not allowed to express directly.

The five activity indicators are **standardized within course**: each value is that enrolment's distance from its own course's mean, in that course's standard deviations. Every indicator therefore has mean zero and standard deviation one within every course, and a browse of 1.2 reads as "1.2 standard deviations more active than the average enrolment in that course". This is the scale a learning-analytics reader meets, because activity volume is not comparable across courses: a course with weekly quizzes generates more clicks than a seminar whatever its students are like, so an analyst compares a student with their coursemates rather than with the whole cohort.

In this simulation the courses do not in fact differ in activity volume, so the standardization sets the scale rather than removing a course effect. It is stated here rather than left implicit because a reader who assumed otherwise would misread what the course column does.

The indicators are simulated directly on the $\log_{1p}(\text{events})$ scale, the scale of a logged activity export; no raw counts are generated. They are standardized on that scale, not as counts. Standardizing is affine within a course, so it leaves the within-course skew exactly as it found it, and raw learning-analytics counts are strongly right-skewed: on an unlogged draft of these data the skew alone looked like an extra class, exactly as a floor or ceiling effect does. The log removes the skew; the standardization puts the courses on a common footing afterwards.

Source

Simulated by `data-raw/course-engagement.R`, seed 2026. The simulation is calibrated to the shape of a real learning-analytics export – its scales, its spreads, its correlations and how persistent engagement is across a student’s courses – but only those aggregate constants were used. No row, identifier or value of any real student is present, and the source is not distributed.

See Also

`vignette("lpa", package = "latents")` for the analysis these data are used in, `multilpa()` for the two-level model, `lta()` for the sequence, and `multilpa(profile_covariates = )` for `previous_grade`.

Examples

```
fit <- multilpa(
  course_engagement,
  vars = c("browse", "lectures", "forum_read", "forum_post", "attendance"),
  id = "student", n_profiles = 2, n_group_classes = 2, n_starts = 4,
  seed = 1
)
as.data.frame(fit)
get_results(fit, what = "profile_probabilities")
# The generating truth ships alongside, so recovery can be checked directly.
get_results(fit, what = "assignments", data = course_engagement,
            truth = "engagement")
```

descriptives

*Describe the variables a model uses, or will use***Description**

One row per variable, with the summaries you would look at before fitting and the one that decides whether a two-level model has anything to find: the intraclass correlation, the share of each variable’s variance that lies *between* groups rather than within them.

Usage

```
descriptives(x, ...)

## S3 method for class 'data.frame'
descriptives(x, vars = NULL, id = NULL, by = NULL, ...)

## S3 method for class 'multilpa'
descriptives(x, by = NULL, ...)

## S3 method for class 'multilpa_transitions'
descriptives(x, by = NULL, ...)
```

```
## S3 method for class 'multilpa_covariates'
descriptives(x, by = NULL, ...)
```

Arguments

<code>x</code>	A data frame, or a fitted model of this package. A fit carries the columns it was built from, so <code>vars</code> and <code>id</code> are read from it.
<code>...</code>	Passed between methods.
<code>vars</code>	Character vector of column names to describe. Repeated names are described once. Columns must be vectors; matrix-valued columns raise <code>latents_bad_data</code> . For a data frame, defaults to every numeric column that is not <code>id</code> .
<code>id</code>	Optional. The column identifying the groups rows are nested in. Supplying it adds <code>n_groups</code> and <code>icc</code> ; omitting it leaves them out.
<code>by</code>	Optional. "profile" or "group_class" on a fitted model, or a column name on a data frame: describe each variable once per level of it, rather than once overall. Rows whose stratifier is NA are neither dropped nor folded into another level: they form their own stratum, labelled NA in the <code>by</code> column and reported last, so that the strata account for every input row exactly once. The stratifier is never compared with <code>==</code> , so a missing value cannot select rows it does not belong to.

Details

An ICC near zero suggests little between-group mean variation in that variable. It does not rule out group classes that differ in other variables, profile prevalence, or other features of their distributions.

Value

A base `data.frame`, one row per variable, or one row per variable and by level, with the columns

- `variable` character: the column described.
- `n` integer: observations with a value.
- `n_missing` integer: observations without one.
- `mean,sd,min,max` numeric, and `NA_real_` for a variable that is not numeric.
- `n_distinct` integer: distinct observed values, which is the useful summary where a mean is not.
- `n_groups,icc` present only when `id` is known. `icc` is the one-way random-effects estimate, and may be slightly negative when the between-group mean square falls below the within-group one; that is the estimator reporting no group structure, not an error. Rows with a missing group ID do not contribute to either statistic.

A `by` column comes first, named after what it splits on and holding that level's own value, NA for the stratum of rows whose stratifier is missing. For every variable, `n + n_missing` summed over the strata equals the number of input rows; that invariant is asserted before the table is returned. Describing by a class the model assigned keeps that assignment out of the described frame, so an indicator of the caller's own that is named `profile` or `group_class` is summarized as itself, not replaced by the assignment. A `by` whose name is one of the summary's own columns cannot be told apart from them and raises `latents_bad_data`.

References

Bliese, P. D. (2000). Within-group agreement, non-independence, and reliability. In K. J. Klein & S. W. J. Kozlowski (Eds.), *Multilevel theory, research, and methods in organizations*.

See Also

`multilpa()` to fit, `diagnostics()` for the after-the-fit counterpart.

Examples

```
descriptives(
  course_engagement,
  vars = c("browse", "lectures", "forum_read", "forum_post", "attendance"),
  id = "student"
)

# A missing stratifier is a stratum of its own, not a silent loss and not
# missingness invented in a variable that has none.
descriptives(data.frame(cohort = c("A", "A", "B", NA), score = 1:4),
  vars = "score", by = "cohort")

fit <- multilpa(
  course_engagement,
  vars = c("browse", "lectures", "forum_read", "forum_post", "attendance"),
  id = "student", n_profiles = 2, n_group_classes = 2, n_starts = 4,
  seed = 1
)
descriptives(fit)
descriptives(fit, by = "profile")
```

diagnostics

Every classification diagnostic, in one call

Description

`summary()` reports what a model estimated. This reports whether to believe it: how sharply the posterior separates the classes, how big each class actually is, how confidently each unit was assigned, and whether the within-profile independence the model assumes survives contact with the data.

Usage

```
diagnostics(x, data = NULL, plots = TRUE, by = c("profile", "overall"), ...)

## S3 method for class 'multilpa_diagnostics'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)

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

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

Arguments

<code>x</code>	A fitted model of this package.
<code>data</code>	Optional. The data the model was fitted to. A fit carries the columns it was built from, so this is only needed to override them.
<code>plots</code>	TRUE, the default, also draws the four classification plots (effective profile sizes, posterior probabilities of the assigned profiles, case entropy, and the average posterior probability matrix), as printing <code>plot()</code> on the result does. Without the <code>ggplot2</code> package a message says so and the tables are still returned. FALSE returns the tables only.
<code>by</code>	"profile", the default, assesses each profile's bivariate residuals separately; "overall" pools them. It is the one argument of a gathered table this function forwards, because it is the one that changes what a gathered table means rather than which fit it is taken from. The level the classification tables use is not an argument here: it follows from whether the fit has discrete group classes.
<code>...</code>	For <code>diagnostics()</code> , nothing further is accepted. An argument this function cannot forward raises an error of class <code>latents_bad_argument</code> naming it, rather than being dropped on the way to a table that then means something other than what was asked for. For <code>plot()</code> , nothing further is accepted either; style the returned plots with <code>ggplot2</code> .
<code>row.names, optional</code>	Passed to the base generic; <code>row.names</code> is applied to the returned table.

Details

It gathers the four classification tables of `get_results()` — "entropy", "classification", "average_posteriors" and "residuals" — and prints one line of reading per diagnostic rather than four differently shaped tables. The tables themselves come back from `get_results()` on the result, or on the fit.

Value

An object of class `multilpa_diagnostics`. Read its tables with `get_results(result, what = )`, which offers "entropy", "classification", "average_posteriors", "residuals" and "all", and never with `$`. A model family that has no bivariate residuals leaves that table out of "all", and asking for it by name raises `latents_no_group_classes`.

`print()` returns the object invisibly, having printed one line per diagnostic: relative entropy, smallest class, lowest average posterior and largest residual, at each level the fit has. `plot()` returns every classification plot as a `latents_plots` list of `ggplot` objects — the effective profile sizes, the posterior probability of each case's assigned profile, each case's relative entropy, and the average posterior probability matrix — that draws all four when printed. `as.data.frame()` returns the entropy table, the primary one.

See Also

[get_results\(\)](#) for these tables and every other one, [descriptives\(\)](#) for the before-the-fit counterpart, and [summary\(\)](#) for what the model estimated rather than whether to trust it.

Examples

```
fit <- multilpa(
  course_engagement,
  vars = c("browse", "lectures", "forum_read", "forum_post", "attendance"),
  id = "student", n_profiles = 2, n_group_classes = 2, n_starts = 4,
  seed = 1
)
quality <- diagnostics(fit)
quality
get_results(quality, what = "classification")
get_results(diagnostics(fit, by = "overall"), what = "residuals")
```

enumerate_classes	<i>Enumerate numbers of individual profiles and group classes</i>
-------------------	---

Description

Fits every requested combination and retains errors and warnings alongside successful fits. Information criteria are descriptive: neither the smallest BIC nor high entropy guarantees the correct number of classes. Unconverged, boundary, and unreplicated fits are reported, not silently selected.

Usage

```
enumerate_classes(
  data,
  vars,
  id,
  n_profiles = 1:4,
  n_group_classes = 1:3,
  model = "basic",
  seed = NULL,
  family = "profiles",
  between_variance = c("varying", "equal"),
  time = NULL,
  ...
)
```

Arguments

data	Data frame.
vars	Continuous indicator names.

<code>id</code>	The column that identifies the groups the observations are nested in (students in schools, reports in students). <code>NULL</code> enumerates single-level models, with a message; <code>enumerate_lpa()</code> and <code>enumerate_lca()</code> fit those by name.
<code>n_profiles</code>	Positive integer profile counts to try.
<code>n_group_classes</code>	Positive integer group-class counts to try. With <code>id = NULL</code> it is 1 and may be left out.
<code>model</code>	The covariance structures to cross with the class counts. "basic", the default, fits the four structures that combine variances equal or varying across profiles with covariances absent or present: "EEI" (equal variances, no covariances), "VVI" (varying variances, no covariances), "EEE" (one covariance matrix shared by all profiles) and "VVV" (a covariance matrix for each profile). They are the choices that matter most in practice, since omitting covariances between correlated indicators makes the grid favour additional profiles. "all" fits all 14 structures of Celeux and Govaert (1995). Otherwise name the structures by their three-letter codes, as <code>multilpa()</code> 's <code>model</code> takes them: the letters give the volume, shape and orientation of each profile's covariance matrix, each equal (E) or varying (V) across profiles, with I for no covariances. <code>NULL</code> crosses no structures and fits whatever the other arguments ask for. With one continuous indicator the structures reduce to equal or varying variance, and codes that coincide are fitted once; with only categorical indicators there is no structure to cross. A structure set through <code>variance_model</code> , <code>covariance_model</code> , <code>volume</code> , <code>shape</code> or <code>orientation</code> replaces the default.
<code>seed</code>	Optional reproducible seed for each fit.
<code>family</code>	"profiles" (the default) enumerates the profile model over <code>n_profiles</code> , <code>n_group_classes</code> and <code>model</code> . One or more of the group-class families "additive", "dispersion" and "additive_dispersion" enumerates those instead, over <code>family</code> , <code>n_group_classes</code> and <code>between_variance</code> ; <code>n_profiles</code> and <code>model</code> are then refused, and the result is a <code>latents_family_enumeration</code> read with <code>get_results()</code> and <code>get_candidate()</code> .
<code>between_variance</code>	For group-class families: the between-variance restrictions to cross, "varying", "equal" or both (the default). The dispersion family is always fitted with "equal".
<code>time</code>	The occasion column: enumerate latent transition models (<code>lta()</code>) over <code>n_profiles</code> (default 2:4) and <code>n_group_classes</code> (default 1), holding every other <code>lta()</code> argument given in ... fixed. <code>model</code> , when given, crosses those covariance structures. The result is a <code>latents_transition_enumeration</code> , read with <code>get_results()</code> and <code>get_candidate()</code> .
...	Further arguments to <code>multilpa()</code> (or to <code>lta()</code> with <code>time</code>); for group-class families only <code>n_starts</code> , <code>max_iter</code> , <code>tol</code> and <code>min_variance</code> .

Value

An object of class `multilpa_enumeration`. Read it with the verbs that describe it rather than by reaching into it: `as.data.frame()` gives one row per candidate model with every criterion and diagnostic (including both entropies and, at each level, the smallest, largest and mean average posterior probability and the number of classes below a cutoff), `get_results(x, "criteria")` names

the candidate each criterion's minimum falls on, `get_results(x, "classification")` gives every class's average posterior probability and relative entropy for every candidate, `get_results(x, "deltas")` and `"knees"` give the model-selection evidence, `summary()` prints each criterion's minimum and every table, `plot()` draws the information criteria (and, with what, classification quality and model-selection views), `get_candidate()` returns the fitted model of one candidate, and `compare_models()` puts candidates side by side. Each candidate has a name, the first column of the table (`"profiles3_groups2"`), which these verbs take.

Conditions

`latents_bad_argument` when model names an unknown structure, when it is given together with another way of setting the structure, or when it is given although every indicator is categorical.

References

Celeux, G., & Govaert, G. (1995). Gaussian parsimonious clustering models. *Pattern Recognition*, 28(5), 781–793.

See Also

`get_candidate()` to take one fitted model out of the grid, `summary()` for the criterion-by-criterion comparison.

Examples

```
set.seed(1)
d <- data.frame(g = rep(1:10, each = 10), y = rnorm(100))
candidates <- enumerate_classes(d, "y", "g", n_profiles = 1:2,
                               n_group_classes = 1, n_starts = 2, seed = 1)
as.data.frame(candidates)
summary(candidates)

# Single level, crossing class counts with two named structures:
single <- enumerate_lpa(iris, c("Sepal.Length", "Sepal.Width",
                               "Petal.Length", "Petal.Width"),
                       n_profiles = 1:3, model = c("VVV", "EEE"),
                       n_starts = 2, seed = 1)
summary(single)
```

enumerate_lca

Compare latent class models

Description

Fits single-level latent class models with each number of classes in `n_classes` and returns them in one table for comparison by information criteria, entropy and the diagnostics of each fit. Every indicator is categorical unless named in ordinal or count, as in `lca()`. It is `enumerate_classes()` with `id = NULL`, and it returns the same object; the number of classes is its `n_profiles` column.

Usage

```
enumerate_lca(data, vars, n_classes = 1:4, ...)
```

Arguments

<code>data</code>	A data frame with one row per observation.
<code>vars</code>	Names of the categorical indicator columns.
<code>n_classes</code>	Positive whole numbers of classes to compare.
<code>...</code>	Further arguments for <code>multilca()</code> , such as <code>missing</code> , <code>n_starts</code> , <code>seed</code> and <code>min_probability</code> , except <code>id</code> , <code>n_group_classes</code> and <code>categorical</code> .

Details

For observations nested in groups, use `enumerate_classes()` with the grouping column as `id` and the indicators as `categorical`.

Value

An object of class `multilpa_enumeration`, as `enumerate_classes()` returns: `as.data.frame()` gives one row per candidate, `summary()` the candidate each criterion prefers, `plot()` the criteria against the number of classes, and `get_candidate()` one fitted candidate, named by its `n_profiles`.

Conditions

`latents_bad_argument` when `id`, `n_group_classes`, `categorical` or `model` is passed.

See Also

`lca()` to fit one model; `enumerate_lpa()` for continuous indicators.

Examples

```
models <- enumerate_lca(student_esm, c("time_with_friends",
                                     "on_social_media", "sports"),
                       n_classes = 1:3, n_starts = 2, seed = 1)
summary(models)
```

enumerate_lpa

Compare latent profile models

Description

Fits single-level latent profile models with each number of profiles in `n_profiles` under each covariance structure in `model`, and returns them in one table for comparison by information criteria, entropy and the diagnostics of each fit. It is `enumerate_classes()` with `id = NULL`, and it returns the same object.

Usage

```
enumerate_lpa(data, vars, n_profiles = 1:4, model = "basic", ...)
```

Arguments

<code>data</code>	A data frame with one row per observation.
<code>vars</code>	Names of the indicator columns.
<code>n_profiles</code>	Positive whole numbers of profiles to compare.
<code>model</code>	The covariance structures: "basic", the default, for the four that combine equal or varying variances with covariances absent or present (EEI, VVI, EEE, VVV); "all" for all 14; or their three-letter codes. See enumerate_classes() .
<code>...</code>	Further arguments for multilpa() , such as <code>missing</code> , <code>n_starts</code> , <code>seed</code> and <code>categorical</code> , except <code>id</code> and <code>n_group_classes</code> .

Details

For observations nested in groups, use [enumerate_classes\(\)](#) with the grouping column as `id`, which also crosses the number of group classes.

Value

An object of class `multilpa_enumeration`, as [enumerate_classes\(\)](#) returns: [as.data.frame\(\)](#) gives one row per candidate, [summary\(\)](#) the candidate each criterion prefers, [plot\(\)](#) the criteria against the number of profiles, and [get_candidate\(\)](#) one fitted candidate.

Conditions

`latents_bad_argument` when `id` or `n_group_classes` is passed, or when `model` names an unknown structure.

References

Celeux, G., & Govaert, G. (1995). Gaussian parsimonious clustering models. *Pattern Recognition*, 28(5), 781–793.

See Also

[lpa\(\)](#) to fit one model; [enumerate_lca\(\)](#) for categorical indicators; [bootstrap_lrt\(\)](#) to test one number of profiles against another.

Examples

```
models <- enumerate_lpa(srl, c("cognitive_strategies", "intrinsic_value",
                             "self_efficacy", "self_regulation",
                             "test_anxiety"),
                       n_profiles = 1:3, n_starts = 2, seed = 1)

summary(models)
plot(models)
```

enumerate_regressions *Compare mixture regressions with different numbers of classes*

Description

Fits `mixture_regression()` for every requested number of classes (and of group classes, for the two-level model) and returns their fit statistics in one table. Optionally adds a parametric bootstrap likelihood-ratio test of $k - 1$ against k classes (McLachlan 1987; Nylund, Asparouhov and Muthen 2007): outcomes are simulated from the $k - 1$ class fit, both models are refitted to every replicate, and the p-value is the share of replicate statistics at least as large as the observed one, $(1 + b) / (1 + B)$.

Usage

```
enumerate_regressions(
  formula,
  data,
  n_classes = 1:4,
  n_group_classes = 1L,
  bootstrap = 0L,
  bootstrap_starts = 3L,
  seed = NULL,
  ...
)

## S3 method for class 'latents_regression_enumeration'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)

## S3 method for class 'latents_regression_enumeration'
get_results(
  x,
  what = c("fit", "model"),
  n_classes = NULL,
  n_group_classes = 1L,
  ...
)

## S3 method for class 'latents_regression_enumeration'
print(x, digits = 4L, ...)
```

Arguments

formula	A model formula outcome ~ predictors. Factors, interactions and offset() are supported.
data	A data frame.
n_classes	Integer vector of class counts to fit.

n_group_classes	Integer vector of group-class counts (two-level model; needs id). Crossed with n_classes.
bootstrap	Number of bootstrap replicates for the likelihood-ratio test; 0 skips it.
bootstrap_starts	Random starts for each bootstrap refit.
seed	NULL or an integer seed. The caller's random state is restored afterwards.
...	Further arguments to <code>mixture_regression()</code> , such as family, id, class_level, common, membership or variance.
x	A <code>latents_regression_enumeration</code> object.
row.names, optional	Unused; part of the generic.
what	"fit" for the comparison table, "model" for one fitted model.
digits	Significant digits.

Value

An object of class `latents_regression_enumeration`. Its table (from `as.data.frame()` or `get_results(x, "fit")`) has one row per model: `n_classes`, `n_group_classes`, `log_likelihood`, `n_parameters`, `aic`, `bic`, `bic_rows`, `sabic`, `icl`, `entropy`, `smallest_share`, `converged`, `n_best_replicated`, `best_bic` (the minimum-BIC row), and, with `bootstrap > 0`, `blrt_statistic`, `blrt_p_value` and `blrt_replicates` (the successful replicates) and `blrt_flagged` (the failed, unconverged, separated or reversed-likelihood replicates). The p-value is NA if any requested replicate fails validation. `get_results(x, "model", n_classes = , n_group_classes = )` returns one fitted model.

Conditions

`latents_bad_argument` for an empty or invalid grid. A model that cannot be fitted at some count (for example every start degenerates) gets a row of NA with its condition message in note, rather than stopping the comparison.

References

McLachlan, G. J. (1987). On bootstrapping the likelihood ratio test statistic for the number of components in a normal mixture. *Applied Statistics*, 36, 318–324.

Nylund, K. L., Asparouhov, T., & Muthen, B. O. (2007). Deciding on the number of classes in latent class analysis and growth mixture modeling: A Monte Carlo simulation study. *Structural Equation Modeling*, 14, 535–569.

Examples

```
classes <- enumerate_regressions(score ~ hours, data = study_hours,
                                n_classes = 1:3, n_starts = 3, seed = 1)

classes

with_test <- enumerate_regressions(score ~ hours, data = study_hours,
                                    n_classes = 1:2, n_starts = 3, seed = 1,
                                    bootstrap = 19)
```

```
as.data.frame(with_test)
```

fit_staged

Fit a two-level model in stages, holding the measurement solution

Description

Estimates the measurement model once, on its own, and then estimates the group-class structure with that measurement held fixed. This is the staged workflow used when the measurement solution is meant to be decided before the grouping question is asked, so that adding group classes cannot move the profiles they are meant to describe.

Usage

```
fit_staged(
  data,
  vars,
  id,
  n_profiles,
  n_group_classes = 2L,
  measurement = NULL,
  variance_model = c("varying", "equal"),
  n_starts = 10L,
  max_iter = 1000L,
  tol = 1e-08,
  min_variance = 1e-06,
  seed = NULL,
  missing = c("error", "fiml"),
  covariance_model = c("diagonal", "full"),
  categorical = character(),
  min_probability = 1e-10,
  time = NULL
)
```

Arguments

data	A data frame containing indicators and a group identifier.
vars	Unique character vector of continuous indicator column names.
id	Name of the observed group identifier column. Character, factor, or numeric identifiers are supported; missing identifiers are not. Required here: this model has a second level by construction, so it has no single-level form and does not take <code>multilpa()</code> 's <code>id = NULL</code> .
n_profiles	Positive integer number of individual profiles.
n_group_classes	Number of latent group classes to estimate in the second stage. Must be at least two; one would leave nothing to estimate.

measurement	Optional fitted multilpa model to use as the first stage, so that a measurement solution already chosen and inspected is carried forward rather than refitted. Its profiles, indicators and measurement options must match the ones requested here, and it must have one group class: a multi-class fit already estimates group structure and is not a measurement-only first stage. When NULL, the first stage is fitted here with one group class.
variance_model	Either "varying" (profile-specific variances or covariance matrices) or "equal" (shared across profiles).
n_starts	Number of EM starts, used for both stages.
max_iter	Nonnegative integer maximum number of EM updates per start. max_iter = 0 performs no update. With start, it returns the model evaluated at exactly those values, whatever n_starts is, so that logLik() scores a parameter set supplied from elsewhere rather than the best of some random initializations that were never asked for. Without start there is nothing to evaluate at, so the random initializations are scored and the highest is returned.
tol	Positive relative log-likelihood tolerance. Convergence requires absolute change no greater than tol * (1 + abs(previous log likelihood)).
min_variance	Positive lower bound on each variance, or each covariance eigenvalue for full covariance, in squared input units. This defines a constrained maximum-likelihood problem. Bound-active estimates are explicitly reported and generate a warning.
seed	Optional random seed: any whole number set.seed() accepts, negative ones included. With a supplied seed, the caller's random-number state is restored on exit.
missing	"error" rejects missing indicators; "fiml" maximizes the observed-data likelihood under an ignorable missingness mechanism (MAR). Missing indicators are integrated out, not filled in for likelihood fitting. This holds with membership covariates too; the covariates themselves must be complete (latents_bad_data).
covariance_model	"diagonal" assumes conditional independence; "full" estimates within-profile residual covariances.
categorical	Character vector naming indicators to treat as categorical. Each is modelled by unrestricted, profile-specific response probabilities over its observed categories, which is the latent class measurement model. Binary, ordinal and unordered indicators are all handled by the same unrestricted parameterization; numeric, integer, logical, character and factor columns are accepted. Indicators not named here stay Gaussian, so naming a subset fits a mixed-mode model.
min_probability	Positive lower bound on every categorical response probability, defining a constrained maximum-likelihood problem in the same way min_variance does for Gaussian indicators.
time	Optional name of a column giving each observation's position within its group, such as a wave, occasion or course number. The model does not use it; it is stored so that get_results(x, "sequences"), get_results(x, "sequence_summary") and plot(what = "sequences") can read the assignments back in order. Values must be complete and unique within each group.

Details

The second stage treats the measurement solution as known, so its Wald standard errors (`parameter_inference()`'s default), information criteria and likelihood-ratio comparisons are conditional on that solution and are narrower than they would be if the measurement had been estimated jointly. `parameter_inference(fit, method = "bootstrap")` removes that conditioning for the standard errors and intervals: every re-sample of the groups is staged again, measurement first, so the replicates carry the first stage's sampling variability into the second stage, and the table also reports the measurement parameters with their own errors. A measurement supplied by the caller is re-estimated on each resample with the same options, since the resample is new data. The conditioning of the criteria and likelihood comparisons is a property of staging itself, and it is the reason both parameter counts are reported: compare staged fits with one another using `n_parameters`, and compare a staged fit with a jointly estimated one using `n_parameters_with_measurement`, remembering that the staged likelihood is not the joint maximum and the comparison is descriptive.

Value

A `multilpa` object for the second stage, so that every accessor, diagnostic and method works on it unchanged. Its measurement parameters are exactly the first stage's. It additionally carries fixed, naming the held blocks; `n_parameters`, counting only the parameters this stage estimated; and `n_parameters_with_measurement`, which adds the held measurement back. Use `get_results()` with `what = "stages"` for a tidy two-row summary of both stages.

See Also

`multilpa()` with `fixed` for finer control over which blocks are held, and `starting_values()` with `what = "measurement"` for the values a stage hands on.

Examples

```
set.seed(7)
example_data <- data.frame(
  school = rep(seq_len(30), each = 8),
  score_a = stats::rnorm(240), score_b = stats::rnorm(240)
)
staged <- fit_staged(example_data, c("score_a", "score_b"), "school",
  n_profiles = 2, n_group_classes = 2, n_starts = 2,
  seed = 1)
get_results(staged, what = "stages")
get_results(staged, what = "profile_probabilities")
```

get_candidate

Take one fitted model out of an enumeration grid or a comparison

Description

An enumeration keeps every candidate it fitted, and `compare_models()` keeps every model it compared. This returns one of them, so that a caller who wants to plot, summarise or test a particular candidate names it by its class counts instead of indexing the grid by position.

Usage

```
get_candidate(x, n_profiles, n_group_classes = 1L, model = NULL)
```

```
candidate_fit(x, n_profiles, n_group_classes = 1L, model = NULL)
```

Arguments

<code>x</code>	An enumeration grid from <code>enumerate_classes()</code> (a profile, latent class, transition or group-class family grid), or a comparison from <code>compare_models()</code> . On a family grid, give the name, or <code>n_group_classes</code> with <code>model</code> .
<code>n_profiles</code>	Number of individual profiles identifying the candidate, the candidate's name ("profiles3_groups4"), or its two class counts (<code>c(3, 4)</code>). For a comparison, the label of the row, as its <code>model</code> column shows it.
<code>n_group_classes</code>	Number of group classes identifying the candidate.
<code>model</code>	The covariance model identifying the candidate, needed when <code>enumerate_classes()</code> was given several models and the class counts alone name several candidates. Counts that match more than one candidate raise <code>latents_unknown_candidate</code> , listing their names.

Details

A candidate is named by its name, `get_candidate(grid, "classes3_groups4")`, as the grid's candidate column shows it; by its class counts, `get_candidate(grid, 3, 4)` or `get_candidate(grid, c(3, 4))`; or, on a grid that crosses covariance structures, by its counts and `model`. From a comparison, name the row: `get_candidate(comparison, "classes3_groups4")`. The same `model` is `grid$classes3_groups4`.

`candidate_fit()` is the name this function had in `latents 0.9.21`. It is still accepted with the same arguments and without a warning, so code written for that release keeps working; it forwards to `get_candidate()`, which new code uses.

Value

The fitted model for that cell of the grid: an object of class `multilpa`, the same kind of object `multilpa()` returns, so the verbs for a fitted model apply to it.

Conditions

`latents_unknown_candidate` for a name or class counts not in the grid, for counts that match several candidates (the message lists their names), and for a label that is not a row of a comparison; `latents_failed_candidate` when the candidate is in the grid but its fit raised an error, so no model exists to return; `latents_bad_argument` for a name or pair of counts given together with `n_group_classes` or `model`, or for a value that names no candidate; `latents_incomplete_fit` for a comparison that no longer carries its models. Counts that are not whole positive numbers raise an error from the argument checks. None of these returns `NULL`, because a `NULL` would flow silently into whatever the caller did next.

Examples

```
set.seed(1)
d <- data.frame(g = rep(1:10, each = 10), y = rnorm(100))
candidates <- enumerate_classes(d, "y", "g", n_profiles = 1:2,
                                n_group_classes = 1, n_starts = 2, seed = 1)
get_candidate(candidates, n_profiles = 2, n_group_classes = 1, model = "VVI")
# This grid crosses two structures, so the names carry them:
candidates
get_candidate(candidates, "profiles2_groups1_VVI")
compared <- compare_models(candidates, "profiles1_groups1_VVI",
                            "profiles2_groups1_VVI")
get_candidate(compared, "profiles2_groups1_VVI")
```

get_group_tna	<i>A transition network for each latent group class</i>
---------------	---

Description

Builds one `tna::tna()` model per latent group class of a fitted transition model, collected into the `group_tna` object `tna`'s grouped verbs expect. A class row with no expected outgoing moves keeps the fit's unestimated transition probabilities and raises `latents_empty_transition_row`.

Usage

```
get_group_tna(x, ...)

## S3 method for class 'multilpa_transitions'
get_group_tna(x, label = "Group class", ...)

## S3 method for class 'multilpa_lta'
get_group_tna(x, ...)

## S3 method for class 'multilpa'
get_group_tna(x, label = "Group class", ...)

## S3 method for class 'multilpa_covariates'
get_group_tna(x, label = "Group class", ...)
```

Arguments

<code>x</code>	A fitted model from <code>lta()</code> , or a <code>multilpa()</code> / <code>multilca()</code> fit made with <code>time =</code> , whose modal profile sequences are then used.
<code>...</code>	Passed to <code>tna::tna()</code> for each class.
<code>label</code>	What the classes are called in <code>tna</code> 's output.

Value

An object of class `group_tna`, one `tna` model per latent class: `centralities()` returns one tidy table with a group column, `compare()` contrasts two classes, and `plot()` draws them together.

See Also

`get_tna()` for one network over the whole sample.

Examples

```
if (requireNamespace("tna", quietly = TRUE)) {
  activity <- c("browse", "lectures", "forum_read")
  moves <- lta(course_engagement, vars = activity, id = "student",
              time = "sequence", n_profiles = 2,
              n_group_classes = 2, n_starts = 2, seed = 1)
  get_group_tna(moves)
}
```

get_results

Every table a fitted model holds, from one verb

Description

One verb returns every tidy table this package computes, named by what. The tables were once eleven separate verbs and seven `as.data.frame()` methods with their own what catalogues; a reader had to know which verb owned which table before they could ask for it. Here the object decides what it can offer and what names it, so the same call shape reads a measurement model, a classification diagnostic and a transition matrix.

Usage

```
get_results(x, what = NULL, ...)

## Default S3 method:
get_results(x, what = NULL, ...)

## S3 method for class 'multilpa'
get_results(x, what = NULL, ...)

## S3 method for class 'multilpa_covariates'
get_results(x, what = NULL, ...)

## S3 method for class 'multilpa_transitions'
get_results(x, what = NULL, ...)

## S3 method for class 'multilpa_enumeration'
get_results(x, what = NULL, ...)
```

```
## S3 method for class 'latents_comparison'
get_results(x, what = NULL, ...)

## S3 method for class 'multilpa_bootstrap_lrt'
get_results(x, what = NULL, ...)

## S3 method for class 'multilpa_start'
get_results(x, what = NULL, ...)

## S3 method for class 'multilpa_diagnostics'
get_results(x, what = NULL, ...)

## S3 method for class 'summary_multilpa'
get_results(x, what = NULL, ...)

## S3 method for class 'summary_multilpa_covariates'
get_results(x, what = NULL, ...)

## S3 method for class 'summary_multilpa_transitions'
get_results(x, what = NULL, ...)

## S3 method for class 'summary_multilpa_enumeration'
get_results(x, what = NULL, ...)

## S3 method for class 'summary_multilpa_bootstrap_lrt'
get_results(x, what = NULL, ...)
```

Arguments

<code>x</code>	A fitted model of this package, an enumeration grid, a bootstrap comparison, a starting-value set, a classification-diagnostics object, or the summary of any of them.
<code>what</code>	Which table to return, or "all" for every table the object can produce. The tables an object offers depend on its class and are listed under <i>Tables</i> below; asking for one it does not have raises <code>latents_bad_argument</code> naming the ones it does. <code>NULL</code> , the default, gives the object's primary table: the measurement model for a fit, the candidate grid for an enumeration, the test row for a bootstrap comparison, and the entropy summary for a diagnostics object.
<code>...</code>	Arguments for the requested table. Each table accepts only the arguments listed for it below, and any other name raises <code>latents_bad_argument</code> saying which that table takes, rather than being dropped on the way to a table that would then answer a different question.

Details

Every object also answers the usual generics: `print()` shows a compact overview, `summary()` collects its main tables (taking `level` and `vcov_type` where the model has standard errors), and `as.data.frame()` returns its primary table, the same as `get_results(x)`.

Value

A base data.frame for a single what, whose columns are given under *Tables*, or a named list of such data frames for what = "all".

Tables

Every fitted model offers these:

"profiles" The Gaussian measurement model, one row per profile and continuous indicator: profile, indicator, mean, variance, standard_deviation. Zero rows when every indicator is categorical. Takes scale, and data, which adds mean_standard_error and variance_standard_error.

"responses" The categorical measurement model, one row per profile, categorical indicator and category: profile, indicator, category, probability, threshold. The threshold is $qlogis(P(y \leq \text{category}))$ and is NA_real_ for each indicator's final category, where the cumulative probability is one. Takes data, which adds probability_standard_error.

"covariances" The within-profile residual covariance matrices, one row per profile and ordered pair of continuous indicators: profile, indicator, indicator_2, covariance. The diagonal parameterization carries no covariance array but does state a covariance matrix – the variances on the diagonal and exact zeros off it – and it is reported explicitly, so one shape answers both parameterizations.

"counts" Effective class memberships, one row per class at each level the fit has: level, class, effective_count (the summed posteriors) and effective_proportion. These differ from the modal counts in "classification" whenever entropy is below one.

"model" One row describing the whole fit: its dimensions, likelihood, parameter counts, aic, bic_groups, bic_individual and its convergence and boundary diagnostics. The columns are those of the family: a covariate fit reports its predictor counts, because padding those onto a two-level fit would leave the common case mostly NA. Comparing fits across families is what "information_criteria" is for, and that table does have one column set for every family.

"posteriors" One row per individual and profile: row, group, profile, posterior, modal. Takes format.

"assignments" One row per observation, the columns the fit was built from (or those of data) followed by profile, group_class where the model has one, uncertainty, and one posterior_profile_* column per profile. uncertainty is one minus the posterior of the assigned profile, which is what modal assignment discards. Takes data and truth; truth returns a recovery cross-tabulation instead, described below.

"classification" Classification quality, one row per class and level: level, class, n_modal, proportion_modal, estimated_n, estimated_proportion, average_posterior, odds_correct_classification. Takes level. Under sampling weights estimated_n and estimated_proportion (and so the odds of correct classification) are weighted posterior totals; the modal and average posterior columns describe the units supplied.

"average_posteriors" For the units assigned to each class, their mean posterior of belonging to every class: level, assigned_class, class, n_assigned, average_posterior. Each assigned class's rows sum to one. Conditions on the *assigned* class, where "classification_errors" conditions on the *true* one; the two are different numbers, not two spellings of one table. Takes level.

"classification_errors" The probability that a unit truly in one class is assigned to another: level, true_class, assigned_class, probability, summing to one within each true_class. Takes level. On a membership-covariate fit this is a marginal summary over the fitted units, not a classification-error adjustment conditional on covariates. A fit with a noise component raises `latents_unsupported_noise`, as `three_step()` does, and "all" and `summary()` leave the table out.

"bch_weights" The BCH inverse-error weights, one row per unit and class: level, unit, assigned_class, class, weight. Takes level; weight a regression with one level, not with "both". On a membership-covariate fit these invert the marginal error matrix and are not a conditional three-step correction. Refused for a fit with a noise component, like "classification_errors".

"entropy" One row per level: level, n_classes, n_units, entropy_sum, relative_entropy.

"residuals" Bivariate residuals for every indicator pair, testing the within-profile independence the measurement model assumes: profile, indicator_1, indicator_2, kind, observed, expected, residual, effective_n, statistic, df, p_value, p_adjusted. Takes data, by and adjust.

"information_criteria" Every likelihood-penalty criterion the package computes. Takes format and definitions.

"sequences" The assignments in occasion order, one row per observation: group, group_class, time, profile. Takes format. Raises `latents_no_time` for a fit made without time.

"sequence_summary" One row per group class, summarizing how much data it contributes: group_class, groups, observations, mean_length, median_length, shortest, longest, complete, gaps. Raises `latents_no_time` for a fit made without time.

"starts" One row per EM start: start, log_likelihood, converged, iterations, error and, where the family records it, boundary.

"data" The columns of the original data the model was fitted to, one row per observation in input order: the identifier, the occasion where there is one, and the indicators under their original names. Columns a model never saw are not here, which is why `three_step()` and `r3step()` still take data.

Every family with discrete group classes adds:

"group_posteriors" One row per observed group and group class: group, group_size, log_likelihood, group_class, posterior, modal. Takes format.

And each family adds its own:

"profile_probabilities", "stages" A multilpa fit. "profile_probabilities" has one row per group class and profile: group_class, profile, probability, group_class_probability. "stages" has one row per estimation stage: stage, group_classes, fixed, log_likelihood, parameters, parameters_with_measurement, converged. An ordinary fit has a single "joint" row and a `fit_staged()` result has two; fixed is `NA_character_` for a stage that estimated every block, never a sentinel such as "none".

"coefficients" A covariate fit. One row per estimated membership coefficient: level ("profile" or "group"), outcome, term, parameter (always "logit"), estimate. The standard errors are not here; `parameter_inference()` reports them.

"transitions", "initial", "sequence_lengths" A transition fit. "transitions" has one row per group class and ordered pair of profiles, with group_class, from, to, probability, expected_count, stable, estimated and group_class_probability; it takes estimated and stable to restrict it. "initial" has one row per group class and profile: group_class, profile, probability, prevalence, group_class_probability. "sequence_lengths" has one row per group: group, group_class, occasions, observations, complete.

"candidates", "criteria", "classification" An enumeration grid. "criteria" has one row per information criterion and sample-size convention: criterion, convention, candidate (the candidate its minimum falls on), n_profiles, n_group_classes, model and value. "candidates" has one row per candidate model: its name (candidate, first), its class counts, its covariance model, log likelihood, parameter count, every criterion under both sample-size conventions, both entropies, the average posterior probability (AvePP) summaries of each level – avepp_min, avepp_max, avepp_mean, avepp_below and the same four prefixed group_ – and the convergence, boundary, replication, warning and error diagnostics. Failed candidates are retained with NA estimates and their error text. "candidates" takes cutoff (default 0.8): avepp_below and group_avepp_below count the classes whose AvePP is below it. A class with no modally assigned unit has no AvePP; it is left out of the smallest, largest and mean and counted as below the cutoff. A level with one class has NA summaries. "classification" has one row per fitted candidate, level and class: candidate, n_profiles, n_group_classes, model, then the columns of a fit's "classification" table and that level's relative_entropy. Failed candidates and levels with a single class contribute no rows.

"deltas", "knees" Model-selection evidence of an enumeration grid, each taking using (default "bic_groups"). Neither selects a model. "deltas" has one row per candidate: candidate, its class counts and model, criterion, value, delta_profile (how much the criterion fell from the converged candidate with one profile fewer and the same group classes and structure; positive means this candidate is lower) with from_profile naming that candidate, and delta_group_class with from_group_class for one group class fewer. value and both deltas are NA for a candidate that did not converge, and a delta is NA when the smaller neighbour is missing or did not converge. "knees" has one row per line of the grid: direction ("profiles", along the profiles at fixed group classes; "group_classes", along the group classes at fixed profiles; "parameters", every candidate by its number of parameters, one line per structure), line, criterion, from, to, n_points, knee_candidate (the maximum chord-distance knee: with both axes rescaled to 0-1, the candidate furthest below the straight line from the first to the last, measured perpendicular to it), distance, knee_without_largest (the knee when the last candidate of the line is left out), stable and note. This is a simplified geometric rule inspired by Kneedle, without its smoothing or sensitivity thresholds; a very small positive distance can therefore still name a knee. A line needs three candidates for a knee and four for knee_without_largest and stable; otherwise they are NA and note says why.

"comparison", "classification" A `compare_models()` result. "comparison" is its table, one row per model. "classification", for profile, class, membership-covariate and transition models, has one row per model, level and class: model, then the columns of a fit's "classification" table and that level's relative_entropy; levels with one class are left out. The models themselves come back with `get_candidate()`.

"test", "replicates" A bootstrap comparison. "test" is the one-row result; "replicates" has one row per simulated dataset.

"covariances" A starting-value set. One row per profile and ordered pair of continuous indicators: profile, indicator, indicator_2, covariance.

Tables of the other model families

Each family's first table is its default what; "all" returns a named list of every table (without "recovery", which needs data and truth). Tables with intervals take level, and vcov_type ("observed", "robust" or "opg"; NULL reuses the type stored with a regression fit). The usual generics answer as well: `coef()`, `vcov(type = )`, `confint(parm = , level = )`, `logLik()` and `nobs()`.

Mixture regressions (`mixture_regression()`) "coefficients" (one row per class and term, with Wald statistics, `p_adjusted` by Benjamini-Hochberg over non-intercepts and `exp_estimate` for the binomial, Poisson and ordinal families; ordinal thresholds lead each class as terms "threshold:<lower>|<upper>"), "classes", "membership" (multinomial logits and odds ratios), "group_classes" (two-level fits), "fit" (bic penalised by independent units, `bic_rows` by rows as flexmix reports it), "assignments", "groups", "fitted", "starts" (stage "screened" or "completed"), "classification" and "recovery" (by = "class" or "group_class").

Growth mixtures (`mixture_regression(random = )`) The tables above plus "random" (random-effect variances, standard deviations and correlations, with intervals on the log and Fisher-z scales), "random_effects" (one row per person), "trajectories" (each class's mean trajectory with a band) and "individual" (each person's own predicted curve); the last two take time. A multilevel growth mixture adds "group_classes" and "clusters", and its "fit" counts clusters in bic (persons in `bic_persons`).

Additive fits (`multilpa(family = "additive")`) "parameters" (class means, between- and within-group variances and class weights; variance intervals on the log scale, weights on the logit scale; adjust sets the p-value adjustment, "none" by default), "group_classes", "groups", "intercepts" and "conditional_intercepts" (each group's posterior intercept, overall or given each class), "assignments", "classification", "fit", "recovery" and "starts". effective_groups in "group_classes" is $w^2 (1 - w) / \text{Var}(w)$, the number of perfectly classified groups that would estimate the class weight as precisely; below 50 the class's intervals carry a `latents_weak_class` warning.

Cross-level fits (`multilpa(family = "full_cross_level")` or "restricted_cross_level") "profiles" (individual profiles, then the group classes' means of group means), "group_classes", "composition" (profile shares within each class: model-implied for the full family, a posterior-weighted description for the restricted one), "groups", "assignments", "fit" and "starts". The likelihood is a working likelihood, comparable only between cross-level fits of the same data.

General transition fits (`lta()` with `transitions = "occasion"`, **transition or initial covariates**, or `measurement = "occasions"`) "transitions" (model-implied, averaged over the groups observed), "transition_coefficients" (log odds of each move against staying, with odds ratios), "second_order_transitions" and "second_order_coefficients" (order = 2), "initial" and "initial_coefficients", "profiles", "group_classes", "assignments", "fit" and "starts".

Pooled imputations (`pool_imputations()`) "estimates" (the pooled table), "imputations" (each imputation's aligned estimates) and "fits" (one row per imputation).

Transition and group-class family grids (`enumerate_classes(time = )`, `enumerate_classes(family = )`) "candidates" (one row per candidate, its name first, with its likelihood, criteria, entropy and diagnostics) or "best", the row with the lowest criterion ("bic" by groups, the default,

"aic" or "bic_individual"): the lowest value of a criterion, not a choice of model. With one group class every family is the same model, so those rows share a likelihood. Their candidates are taken out by name with `get_candidate()`; transition grids can also be compared with `compare_models()`.

What "all" returns

A named list of every table the object can produce, in catalogue order, built from the same definitions a single what uses, so the two can never disagree. It takes no further arguments: every table is built with its defaults, and supplying anything else raises `latents_bad_argument`. A table this particular object cannot produce is left out rather than erroring – a sequence table for a fit made without time, bivariate residuals for a family with no discrete group classes – so the names of the list say what was available. Only the classed refusals that mean *this object has no such table* are skipped; anything else propagates.

Arguments the tables take

These reach the requested table through ..., and each table takes only the ones listed for it under *Tables* above.

data A data frame with one row per observation of the fit, in the order it was fitted in. NULL uses the columns the fit carries. For "profiles" and "responses" it triggers `parameter_inference()` and adds a standard error beside every estimate.

scale For "profiles", "raw" reports the estimates in input units and "standardized" divides each indicator's deviation from its grand mean by that indicator's observed standard deviation, which is what `plot(x, scale = "standardized")` draws. Every standard error beside an estimate is divided by the same constant, so the whole row is on one scale.

format For the posterior tables and "sequences", "long" gives one row per unit and class and "wide" one row per unit with one column per class. For "information_criteria", "wide" is the one-row reporting shape and "long" gives one row per criterion and sample-size convention.

level Which level of classification to report: "individuals", "groups" or "both". NULL, the default, gives "both" for a fit that has discrete group classes and "individuals" for one that does not, which is the same rule `diagnostics()` follows.

truth For "assignments", one or more column names of data holding a known label to check the assignments against. See *Recovery* below.

by For "residuals", "profile" assesses each profile separately and "overall" pools them.

adjust For "residuals", the multiplicity correction applied across the indicator pairs.

definitions For "information_criteria" in long format, TRUE adds the formula and the reference for each criterion.

estimated, stable For "transitions", TRUE or FALSE restricts the table to the rows that were estimated from observed occupancy, or to the diagonal. NULL keeps every row.

Recovery

what = "assignments" with **truth** returns a cross-tabulation of the model's labels against known ones instead of the per-observation table: one row per class and truth value, with columns assignment

(which model label the column was compared against), class, truth (the column name), value, n and proportion, the share of the units carrying that truth value that were assigned to that class. Several truth columns may be named at once and are stacked.

Each truth column is compared against the level it describes, which is read off the data rather than guessed: a column that takes one value within every group is a property of the group and is compared against group_class; a column that varies inside any group cannot be a group-level label and is compared against profile. The assignment column records which comparison was made, so the choice is never silent. Group-level recovery counts each group once, even when groups have different numbers of observations. Unused factor levels are omitted, because their within-truth proportions are undefined. Every column is compared against profile for a fit with no discrete group classes.

The labels are the model's own and carry no order, so a recovery table whose mass sits off the diagonal is a relabelling, not a failure.

Information criteria

Let q be the number of free parameters, n the chosen sample size, and EN the classification entropy of the level matching that convention. The criteria are deviance = $-2L$, aic = $-2L + 2q$, bic = $-2L + q \log(n)$, sabic = $-2L + q \log((n + 2) / 24)$, caic = $-2L + q (\log(n) + 1)$, awe = $-2(L - EN) + 2q (1.5 + \log(n))$, icl = $-2L + q \log(n) + 2 EN$, kic = $-2L + 3(q + 1)$, and clc = $-2L + 2 EN$. Note that clc uses the entropy sum, as icl and awe here do; tidyLPA reports a CLC built from relative entropy instead, which is bounded by one and so penalizes almost nothing, and the two numbers are not comparable.

Every criterion that depends on a sample size is reported under both multilevel conventions, "groups" and "individuals", and uses group-level posteriors under the first and individual-level posteriors under the second. That is a stated per-level choice, not a unique multilevel definition: use one convention consistently across compared candidates. deviance, aic and kic depend on no sample size and carry convention = NA_character_. clc depends on none either, but its convention selects which level's classification uncertainty it penalizes, so it is reported once per convention. Individual sample sizes exclude rows with no observed indicators. Where the group classification entropy is undefined, group-level awe, icl and clc are NA.

Classification quality

The odds of correct classification for class k is $(p / (1 - p)) / (r / (1 - r))$, where p is the average posterior in the assigned class and r is the model-estimated class proportion. Values near one indicate classification no better than the class proportion alone; values of five or more are conventionally read as adequate separation. It is NA_real_ wherever it is undefined, that is whenever average_posterior is missing, or it or estimated_proportion is zero or one, so a single-class solution always reports NA_real_. average_posterior is itself NA_real_ for a class with no modal members, where the mean is taken over nothing and is undefined rather than zero. Modal assignment discards classification uncertainty, which is why n_modal and estimated_n differ whenever entropy is below one.

relative_entropy is $1 - EN / (n \log K)$, on the zero-to-one scale, so higher is sharper; entropy_sum is the raw EN the entropy-penalized criteria use, so lower is sharper. relative_entropy is NA_real_ when a level has a single class, where it is undefined rather than perfect.

A fit with individual profiles but no discrete group classes returns individuals only for level = "both", and raises latents_no_group_classes for level = "groups".

Bivariate residuals

The measurement model assumes the indicators are independent within a profile. "residuals" tests that pair by pair, weighting every observation by the posterior the fit holds for it, so a frame in any other row order would pair each observation with someone else's posterior; the alignment is checked rather than assumed. A large residual is evidence of local dependence between two indicators, which the enumeration grid will otherwise absorb by asking for an extra profile. The p-values are approximate, and p_adjusted applies the multiplicity correction named by adjust across the pairs. A fit with sampling weights uses those weights in its descriptive associations and effective_n. Design-based residual inference is not implemented: statistic, df, p_value and p_adjusted are NA for weighted fits rather than unweighted tests attached to weighted moments.

References

Schwarz, G. (1978). Estimating the dimension of a model. *Annals of Statistics*, 6, 461–464. Sclove, S. L. (1987). Application of model-selection criteria to some problems in multivariate analysis. *Psychometrika*, 52, 333–343. Bozdogan, H. (1987). Model selection and Akaike's information criterion. *Psychometrika*, 52, 345–370. Banfield, J. D., & Raftery, A. E. (1993). Model-based Gaussian and non-Gaussian clustering. *Biometrics*, 49, 803–821. Biernacki, C., & Govaert, G. (1997). Using the classification likelihood to choose the number of clusters. *Computing Science and Statistics*, 29, 451–457. Cavanaugh, J. E. (1999). A large-sample model selection criterion based on Kullback's symmetric divergence. *Statistics and Probability Letters*, 42, 333–343. Biernacki, C., Celeux, G., & Govaert, G. (2000). Assessing a mixture model for clustering with the integrated completed likelihood. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22, 719–725. Bolck, A., Croon, M., & Hagenaars, J. (2004). Estimating latent structure models with categorical variables. *Political Analysis*, 12, 3–27. Nagin, D. S. (2005). *Group-Based Modeling of Development*. Harvard University Press. Vermunt, J. K. (2010). Latent class modeling with covariates: two improved three-step approaches. *Political Analysis*, 18, 450–469.

See Also

`as.data.frame()`, which returns the primary table alone, `summary()`, which prints every table, and `diagnostics()` for the classification tables gathered with a condensed reading.

Examples

```
fit <- multilpa(
  course_engagement,
  vars = c("browse", "lectures", "forum_read", "forum_post", "attendance"),
  id = "student", n_profiles = 2, n_group_classes = 2, n_starts = 4,
  seed = 1
)
get_results(fit)
get_results(fit, what = "profile_probabilities")
get_results(fit, what = "entropy")
names(get_results(fit, what = "all"))

# `engagement` and `student_type` are the kinds each row was simulated from,
# which the model never saw. Both recovery checks are one call.
get_results(fit, what = "assignments", data = course_engagement,
  truth = c("engagement", "student_type"))
```

get_tna	<i>A transition network for the whole sample</i>
---------	--

Description

Builds one `tna::tna()` model from a fitted transition model, aggregated over every latent group class.

Usage

```
get_tna(x, ...)

## S3 method for class 'multilpa_transitions'
get_tna(x, ...)

## S3 method for class 'multilpa_lta'
get_tna(x, ...)

## S3 method for class 'multilpa'
get_tna(x, ...)

## S3 method for class 'multilpa_covariates'
get_tna(x, ...)
```

Arguments

x	A fitted model from <code>lta()</code> .
...	Passed to <code>tna::tna()</code> .

Details

The aggregate is formed from the expected transition counts, summed across classes and then normalised, rather than by averaging the class transition matrices. The two differ: averaging weights each class by how probable it is, and summing counts weights it by how much transition mass it actually contributes, which is what a marginal transition probability means. A class holding a tenth of the students but a fifth of the observed moves counts for the latter. If a profile has no expected outgoing moves in any class, its transition row has no count-based estimate; the network uses the fitted class rows averaged by class probability and warns that the row is unestimated. It never interprets absent moves as a certain self-transition.

A general `lta()` fit – with occasion-varying or covariate-dependent transitions, a second order, movers and stayers, or ordinal, count or weighted data – has no single transition matrix, and is refused with `latents_unsupported_tna`; its transitions are in `get_results(x, "transitions")`.

For a `multilpa()` or `multilca()` fit made with `time =`, there is no estimated transition matrix: the network is built by `tna::tna()` from each group's sequence of modal profiles in time order, the observed transitions between the profiles the observations were assigned to. `get_group_tna()` splits those sequences by each group's modal group class. A fit made without `time` has no order and is refused.

Value

An object of class `tna`, as `tna::tna()` returns: every verb of that package applies to it, including `centralities()`, `communities()`, `cliques()` and `plot()`.

See Also

`get_group_tna()` for one network per latent class.

Examples

```
if (requireNamespace("tna", quietly = TRUE)) {
  activity <- c("browse", "lectures", "forum_read")
  moves <- lta(course_engagement, vars = activity, id = "student",
               time = "sequence", n_profiles = 2,
               n_group_classes = 2, n_starts = 2, seed = 1)

  get_tna(moves)
}
```

growth_schools

Reading growth of students nested in schools

Description

A simulated multilevel longitudinal dataset made for the multilevel growth mixture model. 600 students in 40 schools (15 per school) sit a reading test on five occasions (wave 0 to 4). Two kinds of student follow two trajectories: *improving* (starts near 48, gains 3 points a wave) and *plateauing* (starts near 56, gains 0.3). Students vary around their trajectory (random intercept SD 3, random slope SD 0.6, correlation -0.2; residual SD 2). Schools are of two types: in *supportive* schools 80% of students improve, in *struggling* schools 25% do, and a school's support programme (standardized, measured once) makes it more likely to be supportive. About 8% of tests were missed at random.

Usage

```
growth_schools
```

Format

A data frame with 2780 rows (600 students, 40 schools) and 7 columns:

school Integer school identifier, 1 to 40. Pass it as `cluster`.

student Integer student identifier, 1 to 600. Pass it as `id`.

wave Integer test occasion, 0 to 4. The time variable.

score Reading score.

programme Standardized school support programme, constant within school.

school_type Factor, "supportive" or "struggling": the school's generating type. Not for fitting; kept to check recovery.

trajectory Factor, "improving" or "plateauing": the student's generating kind. Not for fitting; kept to check recovery.

Details

The school types are what a multilevel growth mixture model's group classes recover: `mixture_regression(..., cluster = "school", n_group_classes = 2)`, with `group_membership = "programme"`.

Source

Simulated by `data-raw/growth-schools.R` in the source repository.

See Also

`mixture_regression()`, `get_results()`.

Examples

```
descriptives(growth_schools, c("score", "programme"))

fit <- mixture_regression(score ~ wave, growth_schools, n_classes = 2,
  id = "student", class_level = "group",
  random = "wave", random_covariance = "equal",
  cluster = "school", n_group_classes = 2,
  group_membership = "programme",
  n_starts = 2, seed = 1)
get_results(fit, "group_classes")
get_results(fit, "recovery", data = growth_schools, truth = "trajectory")
```

growth_scores

Achievement growth of students with three kinds of trajectory

Description

A simulated longitudinal dataset made for growth models. 300 students sit an achievement test on up to seven occasions (wave 0 to 6). Three kinds of student follow three trajectories: *improving* (starts near 48, gains 3.5 points a wave), *stable* (starts near 62, gains 0.5) and *declining* (starts near 58, loses 2.5). Within each kind, students vary around the trajectory: each has their own start and rate of change (random intercept SD 4, random slope SD 0.8, correlation -0.3, the same spread in every kind), with residual SD 3. Motivation, measured once, makes a student more likely to be improving and less likely to be declining. About 12% of tests were missed at random, so the panel is unbalanced.

Usage

growth_scores

Format

A data frame with 1844 rows (300 students) and 5 columns:

student Integer student identifier, 1 to 300. Pass it as `id`.

wave Integer test occasion, 0 to 6. The time variable.

score Achievement score.

motivation Standardized motivation, constant within student.

trajectory Factor, "improving", "stable" or "declining": the student's generating kind. Not for fitting; kept to check recovery.

Details

Because students vary within their kind, a trajectory model without random effects needs more classes than the three that generated the data; a growth mixture model with random intercepts and slopes recovers three.

Source

Simulated by `data-raw/growth-scores.R` in the source repository.

See Also

[mixture_regression\(\)](#), [compare_models\(\)](#).

Examples

```
descriptives(growth_scores, c("score", "motivation"))

fit <- mixture_regression(score ~ wave, growth_scores, n_classes = 3,
  id = "student", class_level = "group",
  random = "wave", random_covariance = "equal",
  n_starts = 4, seed = 1)
get_results(fit, "recovery", data = growth_scores, truth = "trajectory")
```

latents-conditions *Conditions this package raises*

Description

Every error a caller might reasonably want to handle carries a class, so it can be caught by what went wrong rather than by matching message text, which changes between versions. This page is the contract: the classes below are stable, the messages are not.

Value

Nothing; this page documents conditions rather than a function.

Input and data contracts

- `latents_bad_data` The data frame, the indicator columns or the group column break the input contract: too few rows, a duplicated or absent column name, an indicator that is not a numeric vector, an indicator that is constant or wholly unobserved, missing or non-finite values under `missing = "error"`, or a missing group identifier.
- `latents_bad_argument` A scalar argument is outside its contract: `n_profiles`, `n_group_classes`, `n_starts`, `max_iter`, `tol`, `min_variance`, or a seed or seeds that is not a whole number `set.seed()` accepts.
- `latents_bad_time` time is missing, or repeats a position within a group, so the ordering it is supposed to supply does not exist.
- `latents_bad_categorical` A categorical indicator cannot be encoded, or `min_probability` leaves no room for its number of categories.
- `latents_bad_start` Supplied starting values are missing a block the model needs, do not have the shape it requires, or carry labels that name an indicator or a category this fit does not have.
- `latents_bad_fixed` A fixed request names a block the model does not have, or one that start does not supply, or start is absent.
- `latents_bad_stage` A measurement fit offered to `fit_staged()` has already estimated multiple group classes, or was fitted to different indicators, profiles or measurement options than the stage requested.
- `latents_bad_transition` A transition model was asked for with fewer than two profiles, or with no group observed at two occasions.
- `latents_bad_inference_data` The data passed to a verb alongside a fit do not reproduce the fit: a different number of rows, different values, or a different group ordering. Raised for a plain multilpa fit and for a covariate fit alike.
- `latents_no_indicator_data` A fit carries no stored indicators, so the request cannot be answered without the original data.
- `latents_incomplete_fit` A fit is missing fields the request needs, usually because it was made by an older version. Raised instead of letting a list subset by an absent name yield a NULL element keyed NA, which would leave a hole in the result.

Estimation and identification

- `latents_unidentified` The requested model cannot be identified from these data: more profiles than distinct observed indicator rows, more group classes than groups, or more than one group class with only one profile or only singleton groups.
- `latents_all_starts_failed` Every EM start raised, so no fit was produced. The message carries the distinct start errors.
- `latents_empty_profile` A profile has no observed responses for an indicator, so its response probabilities are not identified; or a group class has no effective membership, so the profile prevalence within it is not defined.
- `latents_too_few_groups` Fewer independent groups than the quantities being reported, so the covariance cannot be formed. Raised by robust `parameter_inference()`, and by `three_step()` and `r3step()`, whose cluster-robust contributions sum to zero at the estimate and so span at most one dimension fewer than the number of groups.

- `latents_inseparable_classes` Classes cannot be separated well enough for the requested three-step correction.
- `latents_no_converge` A fit required by the request did not converge, and using it would report an unconverged estimate as final. Raised for a plain multilpa fit and for a covariate fit alike.
- `latents_boundary_fit` A variance or a response probability sits at its bound, or a mixing probability is zero, so the parameter is on the edge of its space and the asymptotic normal approximation Wald inference relies on does not hold. The standard error is refused rather than reported.
- `latents_singular_information` The observed information is not positive definite, or is numerically singular, so it cannot be inverted and no standard error can be formed.
- `latents_unknown_candidate` The requested profile and group-class combination was never enumerated.
- `latents_failed_candidate` An enumerated candidate could not be fitted. The message carries that candidate's error.
- `latents_no_group_classes` A verb that needs discrete group classes was called on a model that has none. Every model family this version fits has them, so this is a guard against a future family rather than a refusal the current surface can produce.

Model comparison

- `latents_bad_nesting` Two models are not nested in the way the comparison requires, including a `bootstrap_lrt()` pair whose held measurement blocks differ, are held at different values, or are held in only one of the two models.
- `latents_incomparable_models` Two models were fitted to different data, or differ in covariance structure or centering mode where the comparison requires them to match.
- `latents_reversed_likelihood` The model with more parameters has the lower likelihood, so at least one fit is at a local optimum and the comparison is meaningless.
- `latents_bad_outcome` A distal outcome given to `three_step()` is not of a type the requested method can handle, varies within a group when a group-level outcome is requested, or is a measurement indicator that already helped define the classes.
- `latents_bad_covariate` A predictor given to `r3step()` varies within a group when group-level membership is predicted, makes the design rank deficient, or is a measurement indicator that already helped define the classes. The counterpart of `latents_bad_outcome`.
- `multilpa_indicator_reused` A measurement indicator was offered as an external variable, to `three_step()` as an outcome or to `r3step()` as a predictor. Raised alongside `latents_bad_outcome` or `latents_bad_covariate`, so one handler catches the mistake from either verb.
- `latents_bad_scores` Score contributions could not be formed for the robust sandwich.

Unsupported requests

These name capabilities the package does not have. They are raised in place of returning a number that would be wrong.

- `latents_unsupported_inference` Standard errors are not available for this particular fit. Raised for a covariate fit with categorical indicators, where no score is implemented for the response probabilities, for fits made by an older version, for a fit recording a covariance structure code

this version does not know, and by `parameter_inference(method = "bootstrap")` for a fit holding a measurement block, whose held values came from a fit these data do not resample, and by `parameter_inference()` for a fit made with prior, which sits at a posterior mode rather than a likelihood maximum.

`latents_unsupported_prior` `multilpa(prior = )` was asked for a combination the conjugate prior is not defined for: the VEE, EVE, VVE or EVV covariance structure (`mclust` defines none), categorical indicators, a fixed block, membership covariates, or missing indicator values.

`latents_unsupported_noise` `multilpa(noise = TRUE)` was asked for more than one group class, categorical indicators, a fixed block, membership covariates or missing indicator values; or a verb that does not yet account for a noise component — `parameter_inference()`, `three_step()`, `r3step()`, `bootstrap_lrt()`, `starting_values()`, bivariate residuals, `fit_staged()` or a posterior plot — was given a fit that has one.

`latents_unsupported_structure` `multilpa()` was asked for a covariance structure other than EEI, VVI, EEE or VVV together with `profile_covariates` or `group_covariates`, which fit those four only.

`latents_bad_weights` `weights` does not name a column, or its values are missing, negative, non-finite, vary within a unit, or are positive for fewer than two units.

`latents_unsupported_weights` A weighted fit was given to a verb or option that does not account for sampling weights: `bootstrap_lrt()`, `enumerate_regressions(bootstrap = )`, `three_step()`, `r3step()`, prior or noise; or a non-sandwich `vcov_type` was asked of one.

`latents_unsupported_indicator` ordinal or count indicators were combined with an option that does not take them yet: membership covariates, `start`, `fixed`, prior or noise.

`latents_pooling_failed` An imputation's fit or inference failed in `pool_imputations()`, or the imputations' fits report different parameters. Rubin's rules need every imputation, so none is dropped; the message names the imputation and carries the original reason.

`latents_bootstrap_failed` Fewer than two resamples produced a usable fit, so there is nothing to read a spread from. Raised by `parameter_inference(method = "bootstrap")`; the message carries the first reason a resample gave.

`latents_no_free_parameters` The fit holds every parameter it has, so there is no free coordinate to report a standard error for. Raised by `parameter_inference()` and `vcov()` on a fully held fit.

`latents_held_parameter` A named parameter was held fixed by this fit, so it has no sampling distribution and no confidence interval. Raised by `confint()` when `parm` names a held coordinate.

`latents_unsupported_sensitivity` `sensitivity()` was called on a model family whose refit needs arguments the shared refit does not carry, whose profile labels cannot yet be aligned between two fits, or a directly fixed fit whose original free starts cannot be replayed.

`latents_unsupported_three_step` `three_step()` or `r3step()` was called on a membership-covariate fit. Its classification errors depend on the fitted predictors, but the correction here uses one unconditional error matrix.

`latents_unsupported_bootstrap` `bootstrap_lrt()` was called on a person-centred fit. Group baselines were removed before estimation and no distribution for them is available to simulate raw data under the null model.

latents_unsupported_tna `get_tna()` or `get_group_tna()` was called on a general transition fit (a `lta()` fit with occasion-varying or covariate-dependent transitions, a second order, movers and stayers, or ordinal, count or weighted data), which has no single transition matrix to hand over; its transitions are in `get_results(x, "transitions")`.

latents_no_plot `plot(x, what = "all")` was called on an object whose plot method names no views to draw.

latents_no_time A sequence verb was called on a fit made without time, so it carries no ordering.

latents_missing_package A verb needs a package listed in `Suggests` that is not installed. Raised by `get_tna()` and `get_group_tna()` when the tna package is absent; the message names the package and how to install it.

Plotting

latents_no_continuous, latents_no_categorical The requested panel needs indicators of a kind this fit does not have.

latents_nothing_to_describe `descriptives()` was given no variables to describe: either vars named none, or the frame has no numeric column outside id and by to fall back on.

latents_nothing_to_plot The requested panel has no rows to draw.

latents_unknown_category, latents_unknown_criterion A named category or information criterion does not exist in this fit.

latents_bad_scale An unrecognised plotting scale was requested.

Warnings

A fit that is returned but qualified warns rather than errors, so that a simulation loop can catch the qualification it cares about and let the others through.

latents_failed_starts Some, but not all, EM starts raised. The surviving starts are in `get_results(fit, "starts")`.

latents_not_identified `mixture_regression()` with a binary outcome and one trial per class assignment: a mixture of Bernoulli regressions is not identified there.

latents_missing_data `mixture_regression()` with `missing = "error"` met a missing value in a variable the model uses.

latents_rows_dropped Warning. `mixture_regression()` with `missing = "omit"` dropped incomplete rows; the message states how many.

latents_no_valid_start Every `mixture_regression()` start degenerated: a class emptied or its variance collapsed.

latents_degenerate_start Warning. Some `mixture_regression()` starts degenerated and were set aside.

latents_separation Warning. A `mixture_regression()` logit or log-link coefficient diverged (perfect separation); it and its standard error are unreliable.

latents_em_decrease The `mixture_regression()` EM likelihood decreased beyond roundoff. This indicates a defect and should be reported.

latents_unconverged The best start had not converged when `max_iter` was reached, so the returned estimate is not a maximum.

- `latents_boundary` A variance, or a covariance eigenvalue, sits at `min_variance`, so the fit is bound-active and its standard errors are not valid at that parameter.
- `latents_small_classes` A profile or group class has effective membership below one, so it is supported by less than one observation.
- `latents_weak_class` `multilpa(family = "additive")` only: a group class is supported by fewer than 50 effective groups (see `effective_groups` in `get_results()`), because it is small, poorly separated, or both. Standard errors are still reported, but intervals for class means and weights can be miscalibrated.
- `latents_failed_replicates` Some bootstrap replicates failed validation, so the bootstrap p-value is NA.
- `latents_single_level` A message, not a warning: `id = NULL` was passed, so the fit has one observation per unit and no second level. Raised by `multilpa()` on every such fit and once by an `enumerate_classes()` grid; `suppressMessages()` or a `latents_single_level` handler silences it.
- `latents_bootstrap_dropped` Some resamples did not produce a usable fit and were left out of the interval. Raised by `parameter_inference(method = "bootstrap")`, naming how many, so the count the interval rests on is never quietly smaller than `iter`.
- `latents_sensitivity_dropped` One or more seeds did not produce a fit in `sensitivity()`. Their rows are NA rather than absent, and the warning names how many failed and the first reason, so the table is never quietly shorter than `seeds`.
- `latents_extreme_coefficients` A membership logit coefficient is large enough that the class is close to separated, so the estimate is driven by scaling, a sparse class or separation rather than by the data.
- `latents_empty_transition_row` A profile is never occupied before a final occasion, so its transition row is uniform by construction rather than estimated. `get_results(fit, "transitions", estimated = FALSE)` lists them. The transition-network handoff also warns when it includes such a row, because it cannot be read as evidence of persistence.
- `latents_unverified_alignment` A supplied data frame has too little fitted information to establish row order, for example only a repeated group identifier. The result is returned on the caller's assurance that the rows are in fitting order.

Examples

```
set.seed(1)
example_data <- data.frame(
  school = rep(seq_len(8), each = 6),
  score_a = stats::rnorm(48), score_b = stats::rnorm(48)
)
fit <- multilpa(example_data, c("score_a", "score_b"), "school",
  n_profiles = 2, n_group_classes = 1, n_starts = 2, seed = 1)
# A sequence verb on a fit made without `time` is catchable by class.
tryCatch(get_results(fit, "sequences"),
  latents_no_time = function(condition) {
    "this fit carries no ordering"
  })
```

lca	<i>Latent class analysis</i>
-----	------------------------------

Description

Fits a single-level latent class model: each observation belongs to one of `n_classes` unobserved classes, and within a class the discrete indicators are independent. Categorical indicators have unrestricted response probabilities; ordinal and count in `...` choose the structured measurement models described by `multilpa()`. It is the model `multilca()` fits with `id = NULL`, and it returns the same object, so every accessor, plot and inference verb of the package applies to it. In the results the classes are labelled `profile_1`, `profile_2`, and so on, as throughout the package.

Usage

```
lca(data, vars, n_classes, ...)
```

Arguments

<code>data</code>	A data frame with one row per observation.
<code>vars</code>	Names of the discrete indicator columns. Categories are taken from the data; ordinal and count in <code>...</code> identify structured discrete indicators and the remaining columns are categorical.
<code>n_classes</code>	Number of classes, a positive whole number.
<code>...</code>	Further arguments for <code>multilca()</code> , such as <code>missing</code> , <code>n_starts</code> , <code>seed</code> , <code>min_probability</code> and <code>profile_covariates</code> , except <code>id</code> and <code>n_group_classes</code> , which a single-level model does not have.

Details

For observations nested in groups, use `multilca()` with the grouping column as `id`, which adds latent classes of groups above the classes of observations.

Value

A fitted `multilpa` object, as `multilca()` returns.

Conditions

`latents_bad_argument` when `id` or `n_group_classes` is passed; the conditions `multilca()` raises otherwise.

References

Lazarsfeld, P. F., & Henry, N. W. (1968). *Latent Structure Analysis*. Houghton Mifflin.

See Also

[lpa\(\)](#) for continuous indicators; [multilca\(\)](#) for the two-level model; [enumerate_lca\(\)](#) to compare numbers of classes.

Examples

```
fit <- lca(student_esm, c("time_with_friends", "on_social_media",
                        "tv_video_games", "sports"),
          n_classes = 2, n_starts = 3, seed = 1)
fit
plot(fit, what = "heatmap")
```

lpa

*Latent profile analysis***Description**

Fits a single-level latent profile model: a finite mixture of multivariate normal distributions, in which each observation belongs to one of `n_profiles` unobserved profiles and each profile has its own mean on every indicator. It is the model [multilpa\(\)](#) fits with `id = NULL`, and it returns the same object, so every accessor, plot and inference verb of the package applies to it.

Usage

```
lpa(data, vars, n_profiles, ...)
```

Arguments

<code>data</code>	A data frame with one row per observation.
<code>vars</code>	Names of the indicator columns. Numeric columns are continuous indicators; name categorical ones in categorical.
<code>n_profiles</code>	Number of profiles, a positive whole number.
<code>...</code>	Further arguments for multilpa() : the covariance structure (<code>variance_model</code> , <code>covariance_model</code> , or <code>volume</code> , <code>shape</code> , <code>orientation</code>), <code>missing</code> , <code>categorical</code> , <code>n_starts</code> , <code>seed</code> , <code>prior</code> , <code>noise</code> , <code>profile_covariates</code> and the others it documents, except <code>id</code> and <code>n_group_classes</code> , which a single-level model does not have.

Details

For observations nested in groups (students in schools, occasions in persons), use [multilpa\(\)](#) with the grouping column as `id`, which adds latent classes of groups above the profiles.

Value

A fitted `multilpa` object, as [multilpa\(\)](#) returns.

Conditions

latents_bad_argument when id or n_group_classes is passed; the conditions `multilpa()` raises otherwise.

References

Oberski, D. (2016). Mixture models: Latent profile and latent class analysis. In J. Robertson & M. Kaptein (Eds.), *Modern Statistical Methods for HCI* (pp. 275–287). Springer.

See Also

`lca()` for categorical indicators; `multilpa()` for the two-level model; `enumerate_lpa()` to compare numbers of profiles.

Examples

```
fit <- lpa(srl, c("cognitive_strategies", "intrinsic_value",
               "self_efficacy", "self_regulation", "test_anxiety"),
          n_profiles = 2, variance_model = "equal",
          covariance_model = "full", n_starts = 3, seed = 1)
fit
plot(fit, what = "raincloud")
```

lta

Latent transition analysis

Description

Fits latent profiles to repeated observations of the same group and estimates the probabilities of moving between them from one occasion to the next. The measurement model is that of `multilpa()` – Gaussian, categorical or mixed indicators, invariant across occasions – so profiles keep the same meaning at every occasion and a change of profile is a change of state rather than a change of definition. With more than one group class, each class carries its own initial distribution and its own transition matrix, which is how groups that differ in their dynamics are separated from groups that differ only in where they start.

Usage

```
lta(
  data,
  vars,
  id,
  n_profiles,
  time,
  n_group_classes = 1L,
  variance_model = c("varying", "equal"),
  n_starts = 10L,
```

```

max_iter = 1000L,
tol = 1e-08,
min_variance = 1e-06,
seed = NULL,
missing = c("error", "fiml"),
covariance_model = c("diagonal", "full"),
categorical = character(),
min_probability = 1e-10,
occasions = c("observed", "grid"),
select_start = c("likelihood", "converged"),
transitions = c("homogeneous", "occasion"),
transition_covariates = character(),
initial_covariates = character(),
measurement = c("invariant", "occasion"),
order = 1L,
model = NULL,
mover_stayer = FALSE,
weights = NULL,
ordinal = character(),
count = character(),
count_model = c("poisson", "negative_binomial"),
count_dispersion = c("varying", "equal")
)

```

Arguments

<code>data</code>	A data frame containing indicators and a group identifier.
<code>vars</code>	Unique character vector of continuous indicator column names.
<code>id</code>	Name of the observed group identifier column. Character, factor, or numeric identifiers are supported; missing identifiers are not. Required here: this model has a second level by construction, so it has no single-level form and does not take <code>multilpa()</code> 's <code>id = NULL</code> .
<code>n_profiles</code>	Integer number of individual profiles, at least two.
<code>time</code>	Name of the column giving each observation's occasion within its group. Required: this model is defined by the ordering. Values must be complete and unique within each group.
<code>n_group_classes</code>	Positive integer number of latent group classes. One gives an ordinary single-level latent transition model; more than one fits a mixture of transition models over groups.
<code>variance_model</code>	Either "varying" (profile-specific variances or covariance matrices) or "equal" (shared across profiles).
<code>n_starts</code>	Positive integer number of EM starts. The first is deterministic: Ward's hierarchical clustering of the standardized indicators (on at most 2000 evenly spaced rows), each cluster described by its own means and spread, which is the start <code>mclust</code> 's name refers to and finds solutions whose profiles differ in shape. The rest are k-means from random centres. When <code>start</code> is supplied, it supplies the

	first start instead; remaining starts are random initializations. It is ignored when <code>max_iter = 0</code> and <code>start</code> is supplied: that call evaluates the supplied parameters and nothing else, so exactly one start is run whatever <code>n_starts</code> says.
<code>max_iter</code>	Nonnegative integer maximum number of EM updates per start. <code>max_iter = 0</code> performs no update. With <code>start</code> , it returns the model evaluated at exactly those values, whatever <code>n_starts</code> is, so that <code>logLik()</code> scores a parameter set supplied from elsewhere rather than the best of some random initializations that were never asked for. Without <code>start</code> there is nothing to evaluate at, so the random initializations are scored and the highest is returned.
<code>tol</code>	Positive relative log-likelihood tolerance. Convergence requires absolute change no greater than <code>tol * (1 + abs(previous log likelihood))</code> .
<code>min_variance</code>	Positive lower bound on each variance, or each covariance eigenvalue for full covariance, in squared input units. This defines a constrained maximum-likelihood problem. Bound-active estimates are explicitly reported and generate a warning.
<code>seed</code>	Optional random seed: any whole number <code>set.seed()</code> accepts, negative ones included. With a supplied seed, the caller's random-number state is restored on exit.
<code>missing</code>	"error" rejects missing indicators; "fiml" maximizes the observed-data likelihood under an ignorable missingness mechanism (MAR). Missing indicators are integrated out, not filled in for likelihood fitting. This holds with membership covariates too; the covariates themselves must be complete (<code>latents_bad_data</code>).
<code>covariance_model</code>	"diagonal" assumes conditional independence; "full" estimates within-profile residual covariances.
<code>categorical</code>	Character vector naming indicators to treat as categorical. Each is modelled by unrestricted, profile-specific response probabilities over its observed categories, which is the latent class measurement model. Binary, ordinal and unordered indicators are all handled by the same unrestricted parameterization; numeric, integer, logical, character and factor columns are accepted. Indicators not named here stay Gaussian, so naming a subset fits a mixed-mode model.
<code>min_probability</code>	Positive lower bound on every categorical response probability, defining a constrained maximum-likelihood problem in the same way <code>min_variance</code> does for Gaussian indicators.
<code>occasions</code>	How a group's occasions are placed on the transition grid. "observed" numbers each group's own observations consecutively, so a transition always links two adjacent observations. "grid" places them on the grid of every position seen in the data, so a group that skips a position still consumes a transition across the gap and contributes no measurement information at it. The two agree whenever every group is observed at every position.
<code>select_start</code>	Which start the fit reports. "likelihood", the default, takes the highest log likelihood across starts, preferring a converged start only among those tied with it to within a relative $1e-10$. "converged" takes the highest log likelihood among the converged starts whenever at least one converged, and falls back to every start when none did. Use it when an unconverged start edges out a converged one by a negligible amount, which happens when a membership logit creeps

along a flat ridge of the likelihood: the converged start is then a maximum that the standard errors can work with, and the other is not. `summary()` and `get_results(fit, "starts")` show every start either way.

<code>transitions</code>	"homogeneous" (the default) uses one transition matrix for every occasion; "occasion" estimates a separate one for each move (occasion 1 to 2, 2 to 3, ...).
<code>transition_covariates</code>	Names of columns whose values shift the transition probabilities through a multinomial logit per origin profile: the log odds of moving to each other profile rather than staying. A column may change over occasions; the value at the destination occasion is used. Factors are expanded to indicator columns.
<code>initial_covariates</code>	Names of columns that shift the initial profile distribution (a multinomial logit, reference: the last profile), read at each group's first occasion.
<code>measurement</code>	"invariant" (the default) keeps the profiles' means, variances and response probabilities equal across occasions; "occasion" estimates them separately at each occasion, so a profile is defined by its position in the sequence of transitions rather than by one fixed measurement model.
<code>order</code>	1 (the default) or 2. With 2, the profile at each occasion from the third on depends on the profiles at the two previous occasions (a second-order Markov chain); the first move stays first order.
<code>model</code>	A covariance structure as <code>multilpa()</code> takes it ("EEI", "VVI", "VEI", ... ; <code>mclust</code> 's codes). Standard errors are given for EEI, VVI, EEE and VVV in the homogeneous model and for the diagonal EEI and VVI with the extensions; other structures are refused (<code>latents_unsupported_inference</code>).
<code>mover_stayer</code>	TRUE adds a class of stayers: groups that never change profile (identity transitions) and have their own initial profile distribution (Goodman's mover-stayer model). <code>n_group_classes</code> then counts the mover classes; the stayer class is reported as "stayers".
<code>weights</code>	NULL, or the name of a column of data holding one sampling weight per id (constant over its occasions). As in <code>multilpa()</code> : pseudo maximum likelihood with the weights scaled to sum to the number of id units, sandwich standard errors by default, and integer weights equal to repeating each sequence. A weighted fit uses the general transition engine and returns a <code>multilpa_lta</code> .
<code>ordinal, count, count_model, count_dispersion</code>	Ordinal and count indicators, as in <code>multilpa()</code> ; with <code>measurement = "occasion"</code> their parameters are estimated per occasion. Read them with <code>get_results(fit, "ordinal")</code> and <code>get_results(fit, "count_means")</code> . A fit with them uses the general transition engine.

Details

This is latent transition analysis (LTA); with more than one group class it is a mixture over transition patterns, so the classes are trajectories rather than states. `multilpa()` is the cross-sectional counterpart.

By default transitions are first order and homogeneous over occasions: the probability of moving from one profile to another does not depend on the occasion or on earlier profiles. Measurement

parameters are shared across occasions and across group classes. A profile that no group occupies before its final occasion leaves its transition row without information; the row is then uniform by construction rather than estimated. Such a row is warned about when the model is fitted, is flagged by the estimated column of the transition table and is listed by `get_results(fit, "transitions", estimated = FALSE)`. Profile labels are arbitrary and are not comparable across fits without alignment.

`max_iter = 0` updates nothing: the starting values are evaluated and returned with the expectation they imply, and the fit reports `converged = FALSE` after zero iterations. Every reported quantity, including `expected_count` and the implied profile prevalence, is then read off that single expectation.

Value

A `multilpa_transitions` object containing means, variances, optional covariances and `response_probabilities`, `initial_probabilities` (group classes by profiles), `transition_probabilities` (profiles by profiles by group classes, rows indexing the profile moved from), `group_probabilities`, posterior matrices, classifications, log likelihood, information criteria, restart diagnostics and convergence history. Use `get_results()` for the tidy transition table and for the other tables. No standard errors, likelihood-ratio tests or guarantees of global optimality are given for this model family.

With `transitions = "occasion"`, `transition_covariates`, `initial_covariates`, `measurement = "occasion"` or `order = 2` the result is a `multilpa_lta` object instead, read with `get_results()`: transition and initial logit coefficients with Wald standard errors (observed, robust or OPG, from analytic scores), model-implied transition probabilities per occasion, profiles per occasion. Missing indicators (`missing = "fiml"`) and covariance structures are supported; diagonal fits are finished by a quasi-Newton search on the exact likelihood, others by EM. Agreement with Mplus (User's Guide examples 8.13, 8.14; occasion-specific thresholds), `depmixS4` and `LMest` is recorded in `equivalence/lta-extensions/` of the source repository.

References

- Collins, L. M., & Lanza, S. T. (2010). Latent class and latent transition analysis. Wiley.
- Vermunt, J. K. (2003). Multilevel latent class models. *Sociological Methodology*, 33, 213–239. doi:10.1111/j.0081-1750.2003.t01-1-00131.x.

See Also

`get_results()` for the fitted transition probabilities and for the assignments in order, and `multilpa()` for the cross-sectional model this one shares its measurement parameters with.

Examples

```
# Students take their courses in their own order, which `sequence` records,
# so the same enrolments that fit a cross-sectional model fit a transition
# one. Engagement mostly persists from one course to the next.
fit <- lta(
  course_engagement,
  vars = c("browse", "lectures", "forum_read", "forum_post", "attendance"),
  id = "student", n_profiles = 2, time = "sequence", n_starts = 2, seed = 1
)
```

```
get_results(fit, what = "transitions")
summary(fit)
```

mixture_regression	<i>Fit a finite mixture of regressions</i>
--------------------	--

Description

Fits a regression whose coefficients differ across latent classes: the data are assumed to come from `n_classes` subpopulations, each with its own regression of the outcome on the predictors, and the model estimates the regressions, the class sizes and every row's posterior class membership at once. This is clusterwise, latent class or mixture regression (DeSarbo and Cron 1988; Wedel and DeSarbo 1995).

Usage

```
mixture_regression(
  formula,
  data,
  n_classes,
  family = c("gaussian", "binomial", "poisson", "negative_binomial", "ordinal"),
  id = NULL,
  class_level = c("observation", "group"),
  n_group_classes = 1L,
  common = NULL,
  membership = ~1,
  group_membership = ~1,
  variance = c("varying", "equal"),
  n_starts = 10L,
  max_iter = 1000L,
  tol = 1e-08,
  min_variance = 1e-06,
  seed = NULL,
  missing = c("error", "omit"),
  select_start = c("likelihood", "converged"),
  vcov_type = c("observed", "robust", "opg", "none"),
  weights = NULL,
  random = NULL,
  random_covariance = c("varying", "equal", "proportional"),
  random_diagonal = FALSE,
  cluster = NULL
)
```

Arguments

formula	A model formula outcome ~ predictors. Factors, interactions and <code>offset()</code> are supported.
---------	--

data	A data frame.
n_classes	Number of regression classes.
family	"gaussian", "binomial", "poisson", "negative_binomial" or "ordinal" (proportional-odds cumulative logit; see <i>Families</i>).
id	NULL, or the name of the column identifying groups (persons, schools, ...).
class_level	"observation" (each row has its own class) or "group" (all rows of a group share a class; needs id).
n_group_classes	Number of second-level group classes, for class_level = "observation" with id. 1 fits no second level.
common	NULL, or the predictor terms whose coefficients are shared by every class: variable names such as "age", or a one-sided formula such as ~ age. Each term must appear in formula.
membership	Covariates predicting class membership (a multinomial logit, the first class as the reference): variable names such as "age", or a one-sided formula (~ 1, the default, for none). Row-level for "observation"; constant within groups for "group".
group_membership	Group-level covariates predicting the group class, for the two-level model, as names or a one-sided formula. Must be constant within groups.
variance	"varying" (a residual standard deviation, or a negative-binomial dispersion, per class) or "equal" (one shared). Gaussian and negative-binomial families only.
n_starts	Number of random starts, besides one start built from the residuals of a pooled regression. Every start runs 50 EM iterations; the better half (at least two) continue to convergence.
max_iter	Maximum EM iterations per start.
tol	Relative convergence tolerance on the log likelihood.
min_variance	Floor for a Gaussian residual variance, relative to the outcome's variance. A class that reaches it is fitting too few rows exactly; such starts are set aside.
seed	NULL or an integer seed. The caller's random state is restored afterwards.
missing	"error" refuses rows with missing values in any variable the model uses; "omit" drops them with a latents_rows_dropped warning stating how many.
select_start	"likelihood" keeps the start with the highest likelihood; "converged" prefers the best start that converged.
vcov_type	Covariance of the estimates stored with the fit: "observed" (inverse observed information), "robust" (sandwich, clustered on the top-level unit, or on id for a single-level fit given id) or "opg" (outer product of the scores). "none" skips inference.
weights	NULL, or the name of a numeric column of data with a sampling weight per independent unit: per row without id, per id group otherwise (constant within it). Pseudo maximum likelihood with the weights scaled to sum to the number of units; vcov_type defaults to "robust" and refuses "observed" and "opg".

random	NULL (no random effects), or the random effects within classes: variable names such as "time" (a random intercept and a random slope on time) or "intercept" (random intercepts only), or a one-sided formula such as ~ 1 + time (needed for anything names cannot say, such as ~ 0 + time, a slope without a random intercept). See the growth mixture section.
random_covariance	How the random-effect covariance differs across classes: "varying" (one per class), "equal" (one shared matrix, as <code>lcmm::hlme()</code> with <code>nwg = FALSE</code>) or "proportional" (a shared matrix times a class-specific scale, the last class's being one; <code>nwg = TRUE</code>).
random_diagonal	TRUE for uncorrelated random effects (a diagonal covariance, as <code>idiag = TRUE</code> in <code>lcmm</code>).
cluster	NULL, or the name of a column grouping the persons (<code>id</code>) into clusters, such as schools, for the multilevel growth mixture model (with <code>random</code>): each cluster belongs to one of <code>n_group_classes</code> group classes, which shifts how probable each trajectory class is for its persons. The clusters are the independent units of the likelihood, the standard errors and the BIC; with <code>n_group_classes = 1</code> the model is the single-level growth mixture with clusters as those units, comparable by BIC with more group classes. See the growth mixture section.

Value

An object of class `latents_mixture_regression`, or `latents_growth_mixture` when `random` is given. Read it with `as.data.frame()` (the coefficient table) or `get_results()` (every table, by name), and use `predict()`, `plot()`, `summary()`, `coef()`, `vcov()`, `confint()`, `logLik()` and `nobs()` on it. The tables are described on `get_results()`.

Nesting

`class_level` and `n_group_classes` choose one of three likelihoods.

Single level (`id = NULL`) Every row is its own unit and has its own class.

`class_level = "group"` Every row of a group (`id`) belongs to the same class, so a class is a kind of *group*: the regression mixture for repeated measures, `flexmix(y ~ x | id)`. Each group's evidence is the product of its rows' densities.

`class_level = "observation"` **with** `id` **and** `n_group_classes >= 2` Rows keep their own class, and a second-level group class shifts how probable each regression class is within its groups (Vermunt 2003). With `n_group_classes = 1` the groups only enter the "robust" standard errors, which are then clustered on `id`.

Growth mixture models

With repeated measures of persons (`id`, `class_level = "group"`), a time predictor in formula gives every class its own trajectory: latent class growth analysis (Nagin 2005). `random` adds random effects *within* classes, so persons scatter around their class trajectory: the growth mixture model (Verbeke and Lesaffre 1996; Muthen and Shedden 1999). Within class k , person i 's outcomes are

$$y_i = X_i\beta_k + Z_ib_i + e_i, \quad b_i \sim N(0, G_k), \quad e_i \sim N(0, \sigma_k^2 I),$$

with Z built from random. The likelihood is exact (Gaussian, no numerical integration); estimation is EM with the random effects as missing data, finished by quasi-Newton with analytic scores. Such a fit has class `latents_growth_mixture`; its tables, plots and inference are described on `get_results()` and `plot()`. Random effects need the gaussian family, `id` and `class_level = "group"`.

With `cluster`, persons are nested in clusters (students in schools) and the multilevel growth mixture model is fitted: each cluster belongs to a group class h , and the trajectory-class logits of its persons have a group-class-specific intercept and shared covariate slopes,

$$L = \prod_j \sum_h \omega_h \prod_{i \in j} \sum_k \pi_{k|h} f_k(y_i)$$

(Asparouhov and Muthen 2008; Vermunt 2003). `group_membership` names cluster-level covariates of the group class. The group classes are read with `get_results(fit, "group_classes")` and `"clusters"`.

Families

"gaussian" (identity link, a residual standard deviation per class, or one shared under variance = "equal"), "binomial" (logit link; the outcome is 0/1, logical, a two-level factor whose second level is the success, or `cbind(successes, failures)`), "poisson" (log link; `offset()` terms in the formula are honoured) and "negative_binomial" (log link, NB2: variance $\mu + \alpha \mu^2$ with a dispersion α per class, or one shared under variance = "equal", for overdispersed counts; a dispersion estimated at zero is the Poisson limit and raises `latents_boundary`).

"ordinal" fits a cumulative-logit (proportional-odds) regression within each class (McCullagh 1980),

$$P(Y \leq c \mid x, k) = F(t_{kc} - x' \beta_k), \quad c = 1, \dots, C - 1,$$

with ordered thresholds $t_{k1} < \dots < t_{k,C-1}$ per class in place of the intercept, and the parametrization of `MASS::polr()`: a positive slope moves the class towards higher categories. The outcome is an ordered factor, a factor (its levels taken in order) or whole-number categories; the categories are those observed, at least two. common terms share their slopes across classes, and `offset()` is honoured; the formula's intercept is replaced by the thresholds, so it must not be removed. The thresholds appear in the coefficient table as terms "threshold:<lower>|<upper>", and `exp_estimate` of a slope is the cumulative odds ratio of a higher category. The class "mean" of an ordinal outcome (fitted values, `predict()`, trajectories) is the expected category score $\sum_c c P(Y = c)$, the categories scored 1 to C in order; `predict(type = "probabilities")` gives the category probabilities. With one outcome per class assignment the classes are identified by how continuous predictors shift the category probabilities: a mixture with no predictors, or with two categories (then the logistic mixture of "binomial"), is refused with `latents_not_identified` unless `class_level = "group"` gives each class assignment several outcomes.

A mixture of Bernoulli regressions with one trial per unit is not identified – any mixture of Bernoullis is again a Bernoulli – so a binary outcome requires `class_level = "group"` (several rows per class assignment) or binomial counts with more than one trial per row. The request is refused otherwise.

Conditions

`latents_bad_argument` for an inconsistent specification; `latents_bad_data` for an outcome outside the family's support or a covariate that varies within a group where it may not; `latents_not_identified`


```

get_results(by_student, "fit")

# Students nested in schools, each school in one of two group classes:
schools <- mixture_regression(score ~ wave, growth_schools, n_classes = 2,
                             id = "student", class_level = "group",
                             random = "wave", random_covariance = "equal",
                             cluster = "school", n_group_classes = 2,
                             n_starts = 2, seed = 1)
get_results(schools, "group_classes")

```

multilca

Fit a two-level latent class model

Description

Two-level latent class analysis for categorical indicators: `multilpa()` with discrete indicators in `vars` treated as categorical unless named in `ordinal` or `count` through `...`. Each profile is described by unrestricted response probabilities for categorical items and the corresponding structured parameters for ordinal and count items, and the group classes differ in how probable each profile is for their observations. For models that combine categorical and continuous indicators, call `multilpa()` and name the categorical ones in its `categorical` argument.

Usage

```
multilca(data, vars, id, n_profiles, n_group_classes = 2L, ...)
```

Arguments

<code>data</code>	A data frame with one row per observation.
<code>vars</code>	Names of the categorical indicator columns. Numeric, integer, logical, character and factor columns are accepted; factor levels keep their declared order and other types are sorted.
<code>id</code>	Name of the group identifier column, or <code>NULL</code> for a single-level model.
<code>n_profiles</code>	Number of observation-level latent classes (profiles).
<code>n_group_classes</code>	Number of group-level latent classes. Defaults to 2, or to 1 when <code>id = NULL</code> (a single-level model has no second level).
<code>...</code>	Further arguments to <code>multilpa()</code> , such as <code>n_starts</code> , <code>seed</code> , <code>missing</code> , <code>tol</code> or <code>profile_covariates</code> . <code>ordinal</code> and <code>count</code> can name structured discrete indicators; the remaining indicators are categorical. <code>categorical</code> cannot be supplied.

Value

A fitted model of class `multilpa` (or `multilpa_covariates` when membership covariates are given), exactly as `multilpa()` returns it with the corresponding discrete indicator types. Categorical response probabilities are in `get_results(fit, "responses")`, one row per profile, item and category.

Conditions

`latents_bad_argument` when `categorical` is supplied, since its columns are derived from `vars`, ordinal and count here. Every condition of `multilpa()` can also be raised.

See Also

`multilpa()` for continuous and mixed indicators, and `vignette("lca", package = "latents")`.

Examples

```
activities <- c("time_with_friends", "on_social_media", "tv_video_games")
fit <- multilca(subset(student_esm, day <= 1), vars = activities,
               id = "student", n_profiles = 2, n_group_classes = 2,
               n_starts = 1, seed = 1)
get_results(fit, "responses")
get_results(fit, "profile_probabilities")
```

multilpa

Fit a two-level latent profile model

Description

Fits individual Gaussian profiles nested within observed groups. A discrete latent group class determines the profile proportions. Profile means and variances are shared across group classes (measurement invariance), and indicators are independent conditional on individual profile membership by default. Full residual covariance and observed-data maximum likelihood for missing indicators are available.

Usage

```
multilpa(
  data,
  vars,
  id,
  n_profiles,
  n_group_classes = 2L,
  profile_covariates = character(),
  group_covariates = character(),
  variance_model = c("varying", "equal"),
  n_starts = 10L,
```

```

max_iter = 1000L,
tol = 1e-08,
min_variance = 1e-06,
seed = NULL,
start = NULL,
missing = c("error", "fiml"),
profile_slopes = c("shared", "group_class"),
covariance_model = c("diagonal", "full"),
categorical = character(),
min_probability = 1e-10,
time = NULL,
fixed = character(),
centering = c("none", "person", "grand"),
volume = NULL,
shape = NULL,
orientation = NULL,
model = NULL,
select_start = c("likelihood", "converged"),
prior = NULL,
noise = FALSE,
acceleration = c("squarem", "none"),
family = c("profiles", "additive", "dispersion", "additive_dispersion",
  "restricted_cross_level", "full_cross_level"),
between_variance = c("varying", "equal"),
weights = NULL,
ordinal = character(),
count = character(),
count_model = c("poisson", "negative_binomial"),
count_dispersion = c("varying", "equal")
)

```

Arguments

data	A data frame containing indicators and a group identifier.
vars	Unique character vector of continuous indicator column names.
id	Name of the observed group identifier column. Character, factor, or numeric identifiers are supported; missing identifiers are not. id has no default: omitting it raises <code>latents_bad_argument</code>, because a forgotten grouping would otherwise be fitted as a different model without saying so. Passing <code>id = NULL</code> explicitly fits a single-level model: the observations are treated as independent, each row is its own unit, and <code>n_group_classes</code> becomes one. That fit raises a <code>latents_single_level</code> message saying so. This is the ordinary Gaussian or latent-class mixture that the two-level model reduces to, and every verb of this package works on it. The unit column is fabricated internally as <code>.observation</code>; it is not returned by <code>get_results(x, "data")</code>, and a data that already has a column of that name raises <code>latents_bad_data</code>. Asking for more than one group class without an <code>id</code> raises <code>latents_bad_argument</code>, because one observation per unit leaves no composition for a second-level class

	to differ in.
n_profiles	Positive integer number of individual profiles.
n_group_classes	Positive integer number of latent group classes. With <code>id = NULL</code> it is one, and naming anything else is an error.
profile_covariates, group_covariates	Names of numeric columns of data predicting individual-profile and group-class membership through multinomial logits, with the final class as reference. Naming either one fits the one-step covariate model and returns a <code>multilpa_covariates</code> object: profile intercepts differ by group class, profile slopes are shared across group classes unless <code>profile_slopes = "group_class"</code> , and a <code>group_covariate</code> must be constant within each group. Covariates enter in their supplied units, so centre or scale them beforehand if that is what you want. This is one-step maximum likelihood, not a regression on assigned classes; <code>three_step()</code> and <code>r3step()</code> are the staged alternatives. The covariate path supports neither <code>start</code> nor <code>fixed</code> , and refuses them by name rather than ignoring them; it supports <code>missing = "fiml"</code> for the indicators, but covariates must be complete. Use one for an ordinary, pooled LPA.
variance_model	Either "varying" (profile-specific variances or covariance matrices) or "equal" (shared across profiles).
n_starts	Positive integer number of EM starts. The first is deterministic: Ward's hierarchical clustering of the standardized indicators (on at most 2000 evenly spaced rows), each cluster described by its own means and spread, which is the start <code>mclust</code> 's name refers to and finds solutions whose profiles differ in shape. The rest are k-means from random centres. When <code>start</code> is supplied, it supplies the first start instead; remaining starts are random initializations. It is ignored when <code>max_iter = 0</code> and <code>start</code> is supplied: that call evaluates the supplied parameters and nothing else, so exactly one start is run whatever <code>n_starts</code> says.
max_iter	Nonnegative integer maximum number of EM updates per start. <code>max_iter = 0</code> performs no update. With <code>start</code> , it returns the model evaluated at exactly those values, whatever <code>n_starts</code> is, so that <code>logLik()</code> scores a parameter set supplied from elsewhere rather than the best of some random initializations that were never asked for. Without <code>start</code> there is nothing to evaluate at, so the random initializations are scored and the highest is returned.
tol	Positive relative log-likelihood tolerance. Convergence requires absolute change no greater than <code>tol * (1 + abs(previous log likelihood))</code> .
min_variance	Positive lower bound on each variance, or each covariance eigenvalue for full covariance, in squared input units. This defines a constrained maximum-likelihood problem. Bound-active estimates are explicitly reported and generate a warning.
seed	Optional random seed: any whole number set.seed() accepts, negative ones included. With a supplied seed, the caller's random-number state is restored on exit.
start	Optional list of means and variances (profiles by indicators), <code>profile_probabilities</code> (group classes by profiles), and <code>group_probabilities</code> (vector). Starting probabilities must be positive. For full covariance, supply covariances (indicators

	by indicators by profiles); variances may be omitted or must match their diagonals. A categorical model also takes <code>response_probabilities</code> , a list of one profiles-by-categories matrix per indicator named in <code>categorical</code> . That list is read by position unless it is labelled: name its elements after the indicators, or its columns after the categories, and each block is matched to the indicator and category its labels name, so a differently ordered categorical or a differently ordered set of factor levels cannot attach a distribution to the wrong item. A label naming an indicator or a category this fit does not have raises <code>latents_bad_start</code> rather than being aligned by position.
<code>missing</code>	"error" rejects missing indicators; "fiml" maximizes the observed-data likelihood under an ignorable missingness mechanism (MAR). Missing indicators are integrated out, not filled in for likelihood fitting. This holds with membership covariates too; the covariates themselves must be complete (<code>latents_bad_data</code>).
<code>profile_slopes</code>	"shared" (the default) gives each profile covariate one slope per profile logit, common to every group class. "group_class" lets each group class have its own slopes, so a covariate can predict profile membership differently in different kinds of group (a cross-level interaction); its coefficients are reported with terms such as <code>z:group_class_1</code> . It adds (<code>n_group_classes - 1</code>) times as many slope parameters again, each estimated from the groups in its class, so it needs enough groups per class. Naming it without <code>profile_covariates</code> raises <code>latents_bad_argument</code> .
<code>covariance_model</code>	"diagonal" assumes conditional independence; "full" estimates within-profile residual covariances.
<code>categorical</code>	Character vector naming indicators to treat as categorical. Each is modelled by unrestricted, profile-specific response probabilities over its observed categories, which is the latent class measurement model. Binary, ordinal and unordered indicators are all handled by the same unrestricted parameterization; numeric, integer, logical, character and factor columns are accepted. Indicators not named here stay Gaussian, so naming a subset fits a mixed-mode model.
<code>min_probability</code>	Positive lower bound on every categorical response probability, defining a constrained maximum-likelihood problem in the same way <code>min_variance</code> does for Gaussian indicators.
<code>time</code>	Optional name of a column giving each observation's position within its group, such as a wave, occasion or course number. The model does not use it; it is stored so that <code>get_results(x, "sequences")</code> , <code>get_results(x, "sequence_summary")</code> and <code>plot(what = "sequences")</code> can read the assignments back in order. Values must be complete and unique within each group.
<code>fixed</code>	Character vector naming measurement blocks to hold at the values <code>start</code> supplies, instead of estimating them: any of "means", "variances" and "response_probabilities", or "measurement" for every block the model has. Under <code>covariance_model = "full"</code> , "variances" holds the residual covariance matrices. A held block stays exactly as supplied, in every restart, and stops counting towards <code>n_parameters</code> , so this is a different model rather than a different starting point for the same one. <code>start</code> must carry the named blocks; <code>starting_values(fit, what = "measurement")</code> produces them, and <code>fit_staged()</code> wraps the whole two-stage workflow in one

call. The mixing parameters cannot be held: they are what a fixed-measurement fit is for.

centering How to centre the continuous indicators before fitting. "none", the default, fits them as supplied. "person" subtracts each group's own mean from its rows, so a value reads as a deviation from that unit's average and the profiles become profiles of *change* rather than of level: this is the within-person, person-mean-centred design (Quintana, 2021; Voelkle, Brose, Schmiedek, & Lindenberger, 2014). "grand" subtracts one mean per indicator, which moves the origin without touching the within-group structure. The offsets are kept on the fit, so `get_results(x, "data")` still returns the columns you supplied and every verb that checks row alignment still checks it. Centring removes exactly the between-unit variation, so "person" refuses with `latents_bad_data` when it leaves an indicator constant — which is what happens when a unit has one observation of it. With "person" the group classes become types of *change pattern*, not types of unit.

volume, shape, orientation The covariance structure, in the three pieces it is made of. Each profile's covariance decomposes as $\Sigma_k = \lambda_k D_k A_k D_k'$: a *volume* $\lambda_k = |\Sigma_k|^{1/d}$, an *orientation* D_k of eigenvectors, and a *shape* A_k , diagonal with determinant one. Constraining the three across profiles gives the fourteen models `mclust` names with three letters, and this package fits all of them.

volume is "equal" or "varying". shape is "equal", "varying" or "spherical", the last making every indicator's spread equal within a profile, which leaves no orientation to constrain. orientation is "axis" (axis-parallel, a diagonal covariance), "equal" (one orientation shared by every profile) or "varying". Each is NULL by default, which follows `variance_model` and `covariance_model`, so a call that names none of them fits exactly what it always did.

volume	shape	orientation	model
equal	spherical	—	EII
varying	spherical	—	VII
equal/varying	equal/varying	axis	E EI, V EI, E VI, V VI
equal/varying	equal/varying	equal	E EE, V EE, E VE, V VE
equal/varying	equal/varying	varying	E EV, V EV, E VV, V VV

The estimates are those of Celeux and Govaert (1995); the two models with a shared orientation and a free shape, EVE and VVE, have no closed form and use the minorize-maximize step of Browne and McNicholas (2014). Parameter counts match `mclust`'s own for all fourteen.

Anything other than EEI, VVI, EEE or VVV is maximized across every profile at once, so it cannot be combined with a held variances block. `parameter_inference()` reports Wald standard errors for all fourteen: a constrained structure is differentiated in its own free coordinates (log volumes, determinant-one log shapes and orientations), which have exactly as many dimensions as the structure has parameters, and carried to the reported variances and covariances by the delta method. `method = "bootstrap"` is available for all fourteen as well.

model	The covariance structure named by its mclust code, one of "EII", "VII", "EEI", "VEI", "EVI", "VVI", "EEE", "VEE", "EVE", "VVE", "EEV", "VEV", "EVV", "VVV": the first letter is the volume, the second the shape and the third the orientation, each equal (E) or varying (V) across profiles, with I for axis-aligned (no covariances). The same codes <code>enumerate_classes()</code> takes. Use it instead of <code>variance_model/covariance_model</code> or <code>volume/shape/orientation</code> , not together with them.
select_start	Which start the fit reports. "likelihood", the default, takes the highest log likelihood across starts, preferring a converged start only among those tied with it to within a relative $1e-10$. "converged" takes the highest log likelihood among the converged starts whenever at least one converged, and falls back to every start when none did. Use it when an unconverged start edges out a converged one by a negligible amount, which happens when a membership logit creeps along a flat ridge of the likelihood: the converged start is then a maximum that <code>parameter_inference()</code> can work with, and the other is not. <code>summary()</code> and <code>get_results(fit, "starts")</code> show every start either way.
prior	NULL, the default, for maximum likelihood, or <code>prior_control()</code> for maximum a posteriori estimation of the Gaussian means and covariances under the conjugate prior of Fraley and Raftery (2007) — mclust's <code>priorControl()</code> . The hyperparameters default to mclust's <code>defaultPrior()</code> computed from the indicators being fitted, and each M-step is mclust's own, so a fit from the same start reproduces <code>mclust::Mclust(..., prior = priorControl())</code> . It is defined for EII, VII, EEI, VEI, EVI, VVI, EEE, EEV, VEV and VVV (mclust has none for VEE, EVE, VVE and EVV), for complete continuous indicators without fixed or membership covariates; anything else raises <code>latents_unsupported_prior</code> . The mixing proportions are not given a prior. Following mclust, the fit's <code>log_likelihood</code> is the <i>unpenalized</i> log likelihood evaluated at the posterior mode, and <code>aic</code> , <code>bic</code> and their variants are computed from it with the usual parameter count; EM convergence is judged on its relative change. A prior's MAP step can lower that likelihood, so the decrease check made under maximum likelihood is not applied. <code>parameter_inference(method = "wald")</code> is refused for such a fit (<code>latents_unsupported_inference</code>), because the likelihood's score is not zero at a posterior mode; <code>method = "bootstrap"</code> refits every resample with the same prior. The fit records the request as <code>prior</code> and the resolved hyperparameters, in the indicators' units, as <code>prior_parameters</code> .
noise	FALSE, the default, or TRUE to add a noise component: one more mixture component with a constant density $1 / V$ over the hypervolume V of the data, which absorbs observations no Gaussian profile explains — outliers and scatter — instead of letting them distort a profile (Banfield & Raftery, 1993; mclust's <code>initialization = list(noise =)</code> and <code>Vinv</code>). V is mclust's <code>hypvol()</code> : the smaller of the volumes of the data's axis-aligned and principal-component-aligned bounding boxes. The component has no measurement parameters; its mixing proportion is estimated with the others, and, as mclust counts it, the model has two more parameters (the proportion and the hypervolume), so its criteria equal mclust's. A random start puts a tenth of the mass on the noise; a <code>start</code> supplies <code>noise_probability</code> beside <code>profile_probabilities</code> , the two summing to one. Available for continuous, complete indicators with one group class (<code>id = NULL</code> , or <code>n_group_classes = 1</code>) and any of the fourteen covariance structures,

alone or with prior; otherwise `latents_unsupported_noise`. In the fit, the noise component is profile 0, as in `mclust`'s classification: `subject_profiles` is 0 for an observation assigned to it, `get_results(fit, "assignments")` adds a `posterior_noise` column, `get_results(fit, "posteriors")` and `get_results(fit, "profile_probabilities")` have profile-0 rows, and the classification diagnostics count it as a class. `means`, `variances`, `covariances`, `profile_probabilities` and `subject_posteriors` describe the Gaussian profiles only, so the latter two sum to one minus the noise share; the fit adds `noise_probability`, `noise_posteriors`, `n_noise` and `hypervolume`. Verbs that do not yet account for the component — `parameter_inference()`, `three_step()`, `r3step()`, `bootstrap_lrt()`, `starting_values()`, bivariate residuals, `fit_staged()` and the posterior plots — refuse such a fit with `latents_unsupported_noise`.

acceleration "squarem", the default, accelerates EM with SQUAREM (Varadhan & Roland, 2008): each cycle extrapolates along the last two EM steps and finishes with an ordinary EM step, and falls back to plain EM whenever the extrapolated point would lower the likelihood, so the iteration stays monotone and converges to the same kind of maximum. The convergence test is read over a whole cycle, so it is never looser than plain EM's. "none" runs plain EM, which follows the same path as other EM implementations from the same start (`mclust`, `Mplus`) and is the choice for step-by-step reproduction. `iterations` and `max_iter` count EM steps either way. Fits with prior, and models with membership covariates, always use plain EM. For EVE and VVE, whose shared orientation has no closed form, each EM iteration takes a fixed number of warm-started orientation steps rather than solving it (a generalized EM step), under either setting; this keeps every iteration monotone and makes the EM map a fixed function, which SQUAREM requires.

family The model family. "profiles", the default, is the model described above: individual profiles whose prevalences differ across group classes. The other three are group-class-only families with no individual profiles (Houle, Morin & Harvey, 2026): each group has a Gaussian intercept per indicator, drawn from its group class's distribution, and its members vary around it.

- "additive": group classes differ in their means (and, optionally, in between-group variances); one within-group variance per indicator is shared by every class.
- "dispersion": group classes differ in their within-group variances; means and between-group variances are shared.
- "additive_dispersion": group classes differ in means and in within-group variances.

These families take only `data`, `vars`, `id`, `n_group_classes`, `between_variance`, `n_starts`, `max_iter`, `tol`, `min_variance` and `seed`; any other argument is refused with `latents_bad_argument`. Indicators must be complete and continuous, and at least one group must have more than one row (`latents_unidentified` otherwise). Between variances estimated at zero are reached by a boundary maximization with a Karush-Kuhn-Tucker check and flagged (`latents_boundary`); interior fits are finished by Newton steps. The result has class `multilpa_additive`; `bic` uses the number of groups and `bic_individual` the number of rows. Read it with `get_results()`; `summary()`, `plot()`, `coef()`, `vcov()`, `confint()` and

`parameter_inference()` work on it. These families are experimental: enumeration over families, bootstrap inference, missing data and covariates are not yet available. See the [group-class and cross-level models article](#).

Two cross-level families estimate individual profiles and group classes from the same indicators, following the manifest-aggregation specification of Houle et al. (2026, supplement): each group's means of the indicators are between-level indicators of its group class, with class-specific means and variances.

- "restricted_cross_level": profile prevalences are the same in every group class, so profiles and group classes are estimated separately; the profile composition of each class is reported descriptively.
- "full_cross_level": profile prevalences differ across group classes (as in "profiles"), and the group means also inform the classes.

These take data, vars, id, n_profiles, n_group_classes, variance_model (individual profiles), between_variance (group classes), n_starts, max_iter, tol, min_variance and seed. The group means are computed from the same ratings, so the likelihood is the specification's working likelihood: comparable between cross-level fits of the same data, not with the other families. Standard errors are not available (latents_unsupported_inference). The result has class `multilpa_cross_level`, read with `get_results()`.

between_variance

For the additive and additive-dispersion families: "varying" (the default) estimates between-group variances per group class, "equal" one set shared by all classes. The dispersion family holds them equal; "varying" is refused there. For the cross-level families, the variances of the group means within each group class.

weights

NULL, or the name of a numeric column of data holding a sampling weight for each independent unit: each id group of a two-level fit, each row of a single-level one. A two-level weight must be constant within its group (latents_bad_weights otherwise); within-unit weights are not supported. The fit maximizes the pseudo log likelihood $\sum_j w_j \log L_j$ (Skinner, 1989) with the weights scaled to sum to the number of units, as Mplus does, so integer weights give the fit to the data with each unit repeated that many times and the criteria stay on the sample's scale. Standard errors are the sandwich: `parameter_inference()` defaults to `vcov_type = "robust"` and refuses "observed" and "opg" (latents_unsupported_weights), and `method = "bootstrap"` resamples the units with their weights. Classification tables report each unit's own posterior; effective counts and proportions are weighted. `bootstrap_lrt()`, `three_step()`, `r3step()`, prior and noise refuse a weighted fit. Works with every family, membership covariates, categorical indicators and missing = "fiml".

ordinal

Names of indicators in vars to model as ordinal: an (ordered) factor, or whole numbers, whose categories are ordered by factor level or by value. Each follows an adjacent-category logit with category intercepts shared by every profile and one location per profile, $\log P(k | c) / P(k - 1 | c) = a_k - a_{(k-1)} + \eta_{c,k}$, with the last profile's location fixed at zero: $(K - 1) + (C - 1)$ parameters per indicator against C (K - 1) for a categorical one. This is Latent GOLD's default ordinal model, against which it is checked. Read the category probabilities and locations with `get_results(fit, "ordinal")`.

count	Names of indicators in vars holding non-negative whole numbers, modelled as Poisson with one mean per profile (or negative binomial, see count_model); read the means with <code>get_results(fit, "count_means")</code> . Ordinal and count indicators take <code>missing = "fiml"</code> , weights, two-level fits, membership covariates, Wald and bootstrap inference, <code>bootstrap_lrt()</code> , <code>predict()</code> and <code>lta()</code> ; start, fixed, prior, noise and the group-class families do not take them yet (<code>latents_unsupported_indicator</code>).
count_model	"poisson" (the default) or "negative_binomial": NB2, with mean μ and variance $\mu + \alpha \mu^2$, for counts more variable than a Poisson within a profile (Latent GOLD's poisson overdispersed). A dispersion estimated at zero is the Poisson limit, a boundary fit (<code>latents_boundary</code> warning; Wald inference is refused there).
count_dispersion	For <code>count_model = "negative_binomial"</code> : one dispersion per profile ("varying", the default) or one shared by every profile ("equal").

Details

Infinite and constant observed indicators are rejected. Each indicator must have at least two distinct observed values. No rows are silently dropped. In FIML mode, fully missing individuals contribute no direct measurement likelihood but receive posterior probabilities from their group's information. Observation counts retain these individuals; the individual-level BIC excludes them through `n_informative`, while the group-level BIC uses all observed groups. Missing patterns can prevent parameter identification; no general identification guarantee is made. Initialization alone uses indicator-mean filling. EM uses conditional Gaussian sufficient statistics and observed marginal densities. More than one group class requires more than one profile and at least one group with multiple individuals; these checks are necessary but do not establish identification. By default the highest finite likelihood across starts is returned, even if that start did not converge; `select_start = "converged"` prefers a converged start, and `converged` and `starts` show which was chosen. Profile and group-class labels are arbitrary.

Value

An `multilpa` object containing means, variances, optional covariances (indicators by indicators by profiles), `profile_probabilities`, `group_probabilities`, posterior matrices, classifications, log likelihood, information criteria, restart diagnostics, and convergence history. Individual rows retain their input order; groups retain first-occurrence order. `bic` and `bic_groups` use the observed group count; `bic_individual` uses `n_informative`, the number of rows carrying at least one observed indicator. These are alternative conventions, not interchangeable criteria. `boundary` identifies variance bounds; `small_classes` flags effective memberships below one. The fit itself carries no standard errors: `parameter_inference()` computes them from the fit and the data it was fitted to, and `bootstrap_lrt()` tests nested models. No guarantee of global optimality is given, whatever `n_starts` is used. Read the tidy form with `as.data.frame()`; what selects which table.

References

Vermunt, J. K. (2003). Multilevel latent class models. *Sociological Methodology*, 33, 213–239. doi:10.1111/j.0081-1750.2003.t01-1-00131.x.

Banfield, J. D., & Raftery, A. E. (1993). Model-based Gaussian and non-Gaussian clustering. *Biometrics*, 49, 803–821. doi:10.2307/2532201.

Houle, S. A., Morin, A. J. S., & Harvey, J.-F. (2026). Multilevel latent profile analyses: A comprehensive guide. *Organizational Research Methods*. doi:10.1177/10944281261469432.

Skinner, C. J. (1989). Domain means, regression and multivariate analysis. In C. J. Skinner, D. Holt, & T. M. F. Smith (Eds.), *Analysis of complex surveys* (pp. 59–87). Wiley.

Asparouhov, T. (2005). Sampling weights in latent variable modeling. *Structural Equation Modeling*, 12, 411–434. doi:10.1207/s15328007sem1203_4.

Fraley, C., & Raftery, A. E. (2007). Bayesian regularization for normal mixture estimation and model-based clustering. *Journal of Classification*, 24, 155–181. doi:10.1007/s00357-007-0004-5.

See Also

`multilca()` for a model in which every indicator is categorical.

Examples

```
# Course sessions nested in students: engagement profiles of sessions, and
# classes of students that differ in how often they are engaged.
activity <- c("browse", "lectures", "forum_read", "forum_post", "attendance")
fit <- multilpa(course_engagement, activity, "student",
               n_profiles = 2, n_group_classes = 2, n_starts = 4,
               seed = 42)

summary(fit)
as.data.frame(fit)
get_results(fit, what = "profile_probabilities")
```

parameter_inference.multilpa_additive

Tidy inference for a covariate fit

Description

Standard errors, tests and intervals for every free parameter of `multilpa(profile_covariates = )`, at all three levels at once: the measurement model, the profile logits and the group-class logits. Without them a membership coefficient cannot be reported, because nothing distinguishes a real effect from separation.

Computes ordinary maximum-likelihood standard errors from the inverse observed Hessian and delta-method Wald confidence intervals. These are local, asymptotic estimates conditional on the selected numbers of classes. They are not robust sandwich errors and do not account for model selection. Mixture boundary solutions and unidentified Hessians do not admit this calculation.

Usage

```
## S3 method for class 'mutilpa_additive'
parameter_inference(
  x,
  data = NULL,
  level = 0.95,
  step = 1e-04,
  vcov_type = c("observed", "robust", "opg"),
  adjust = .mutilpa_p_adjust_methods,
  method = c("wald", "bootstrap"),
  ...
)

## S3 method for class 'mutilpa_covariates'
parameter_inference(
  x,
  data = NULL,
  level = 0.95,
  step = 1e-04,
  vcov_type = c("observed", "robust", "opg"),
  adjust = .mutilpa_p_adjust_methods,
  method = c("wald", "bootstrap"),
  iter = 199L,
  n_starts = 10L,
  max_iter = 1000L,
  tol = 1e-08,
  seed = NULL,
  boundary = c("error", "fix")
)

## S3 method for class 'mutilpa_cross_level'
parameter_inference(x, ...)

parameter_inference(
  x,
  data = NULL,
  level = 0.95,
  step = 1e-04,
  vcov_type = c("observed", "robust", "opg"),
  adjust = .mutilpa_p_adjust_methods,
  method = c("wald", "bootstrap"),
  iter = 199L,
  n_starts = 10L,
  max_iter = 1000L,
  tol = 1e-08,
  seed = NULL,
  boundary = c("error", "fix")
)
```

```

## S3 method for class 'mutilpa'
parameter_inference(
  x,
  data = NULL,
  level = 0.95,
  step = 1e-04,
  vcov_type = c("observed", "robust", "opg"),
  adjust = .mutilpa_p_adjust_methods,
  method = c("wald", "bootstrap"),
  iter = 199L,
  n_starts = 10L,
  max_iter = 1000L,
  tol = 1e-08,
  seed = NULL,
  boundary = c("error", "fix")
)

## S3 method for class 'mutilpa_lta'
parameter_inference(
  x,
  data = NULL,
  level = 0.95,
  step = 1e-04,
  vcov_type = c("observed", "robust", "opg"),
  adjust = .mutilpa_p_adjust_methods,
  method = c("wald", "bootstrap"),
  iter = 199L,
  n_starts = 10L,
  max_iter = 1000L,
  tol = 1e-08,
  seed = NULL,
  boundary = c("error", "fix")
)

```

Arguments

<code>x</code>	A converged mutilpa fit with inactive variance bounds.
<code>data</code>	Optional. The data frame the model was fitted to; when omitted it is rebuilt from the indicators, identifiers and occasions the fit stores, which round-trip exactly. Supplying it is the stronger check that the caller still holds that frame. A supplied frame is checked exactly against the stored indicators and identifiers; for an older fit without stored indicators, the likelihood provides a weaker consistency check.
<code>level</code>	Confidence level strictly between zero and one.
<code>step</code>	Positive finite-difference step for the analytic score derivative.
<code>vcov_type</code>	"observed" inverts the observed information. "robust" returns the Huber-White sandwich $A^{-1} B A^{-1}$, where B accumulates the outer product of the per-group scores. Groups are the independent units, so robust errors relax the

	assumption that the Gaussian within-group model is correctly specified. They do not relax the assumption that groups are independent. "opg" inverts the outer product of the per-group scores (the BHHH or outer-product-of-gradients estimate), which estimates the information by the variance of the score rather than by the curvature of the likelihood. It is the estimator glca reports, so it reproduces that package's standard errors; the two agree with "observed" in large samples and can differ noticeably in small ones, where "observed" is usually the more reliable. "robust" and "opg" need more groups than parameters.
adjust	Multiplicity correction applied to p_value to produce p_adjusted, one of the methods <code>stats::p.adjust()</code> accepts. The default "none" leaves the two columns equal: a correction changes what a p-value means, so it is applied only when it is asked for, and the method that was applied is recorded in the adjust attribute. The correction is taken over the tests the table actually reports; bounded parameters carry no test and do not count towards the family.
method	How uncertainty is measured. "wald", the default, inverts the observed information in the estimation coordinates, and is available for all fourteen covariance structures. EEI, VVI, EEE and VVV are estimated in log variances or log-Cholesky coordinates; the ten structures that constrain the volume, the shape or the orientation across profiles are estimated in log volumes, log shapes whose determinant-one constraint is written into the coordinates, and Cayley-transform orientations (or, for VEE and EVV, determinant-one log-Cholesky factors), so the information has one row per free parameter of the structure. The reported variances and covariances are then carried from those coordinates by the delta method, and their natural-scale covariance is singular, as it is for the class probabilities. "bootstrap" resamples <i>groups</i> with replacement, refits inside the same covariance family, undoes the relabelling each refit comes back with, and reports the percentile interval and the standard deviation of the replicates. It is available for all fourteen structures. Groups are the resampling unit rather than rows, so the interval carries the same independence assumption as <code>vcov_type = "robust"</code> and not the stronger one "observed" makes.
...	Unused by most methods; accepted so every method shares the generic's signature.
iter	Number of resamples when <code>method = "bootstrap"</code>, at least two. The default 199 is enough for a standard error; a 95% percentile interval is steadier at 999 and above.
n_starts, max_iter, tol	Passed to each bootstrap refit. Each replicate is a fresh fit, so <code>n_starts</code> buys the same protection against a local maximum here as it does in <code>multilpa()</code>, at the same multiple of the cost.
seed	Optional seed for the resampling: any whole number <code>set.seed()</code> accepts; a fraction raises <code>latents_bad_argument</code>. The stream is restored on exit, so a seeded call leaves the caller's random state exactly as it found it.
boundary	What to do when categorical response probabilities sit on their lower bound, <code>min_probability</code>. There the likelihood is flat in the logit and the information is singular, so "error", the default, refuses with <code>latents_boundary_fit</code>. "fix" holds those probabilities at the bound and reports every other parameter conditionally on them, the convention Mplus follows; the held probabilities are

left out of the table (or, for a covariate fit, reported with NA standard errors) and named in the result's `fixed_at_bound` attribute.

Details

Class-membership coefficients are on the multinomial logit scale with the final profile and the final group class as references, so a coefficient is a log odds against that reference. The tests are Wald tests and inherit the usual caveat: they are unreliable for a coefficient driven to the boundary by separation, which the size of the estimate and its standard error together will reveal.

For a general transition fit (an `lta()` fit with covariate, occasion-varying or second-order transitions), the table has one row per transition, initial and second-order logit: `block`, `group_class`, `from`, `to`, `term`, `estimate`, `standard_error`, `statistic`, `p_value`, `p_adjusted`, `conf_low`, `conf_high` (and previous for a second-order fit). `method = "bootstrap"` resamples persons, refits, and matches every replicate's profiles (on the measurement) and group classes to the original's, rebasing the initial logits when the reference profile moves; it needs data, and works where Wald inference is refused (for example full covariance structures). Transition inference holds nothing at a bound: with no bound active `boundary = "fix"` gives the same table as `"error"`, and a fit with an active bound raises `latents_boundary_fit` under either.

Value

A base data.frame with one row per free parameter and the same columns, in the same order, that `parameter_inference()` returns for every other fitted class: `level` ("measurement", "profile" or "group"), `outcome` (which profile or group class the coefficient predicts, or which profile a measurement parameter belongs to), `term`, `parameter` ("mean", "variance", "covariance", "response" or "coefficient"), `estimate`, `standard_error`, `statistic`, `p_value`, `p_adjusted`, `conf_low` and `conf_high`. Variances and covariances are reported in their natural units, with standard errors carried through the delta method from the log and log-Cholesky coordinates on which they are estimated. Categorical indicators contribute one "response" row per profile, indicator and category, named `indicator:category`: the probability of that category, with its standard error carried from the multinomial logits it is estimated on, and no Wald test, as for the covariate-free model.

A base data.frame with one row per reported parameter and the columns `level` ("measurement", "profile" or "group"), `outcome`, `term`, `parameter`, `estimate`, `standard_error`, `statistic`, `p_value`, `p_adjusted`, `conf_low` and `conf_high`. Estimates are on the natural scale: variances in their own units, probabilities as probabilities. A parameter whose null value is on the boundary of its space – a variance, a probability, a residual variance on the covariance diagonal – carries NA for `statistic`, `p_value` and `p_adjusted`, because a Wald test against that boundary is not a question worth asking; the interval still is. All class probabilities are reported; their sum constraints make the natural covariance singular. Wald intervals for probabilities are formed on the logit scale and for variances (and covariance diagonals) on the log scale, then mapped back with the same standard error, so they stay inside the parameter's range and widen towards the far side of a bound; every other interval is `estimate +/- z * standard_error`. `confint()` returns the same intervals. A probability at or below $1e-6$ is treated as on its bound (refused, or held by `boundary = "fix"`).

A fit made with `fixed` reports only the parameters it estimated: a held measurement block is a constant of this likelihood, so it contributes no row here and no row or column to the information matrix. The table is on the natural scale, where every class probability is reported, so it has one row per estimated natural coefficient: `n_parameters` rows plus one for each set of probabilities whose reference category the estimation scale drops. The held values still enter the likelihood at

the values they were held at, so every standard error, interval and p-value is conditional on that measurement solution and does not propagate its uncertainty. See `fit_staged()` for what that conditioning means. For a `fit_staged()` result, `method = "bootstrap"` does propagate it: each resample refits both stages, and the table has a row for every parameter, the measurement included (attribute `stages_resampled` is 2).

With `method = "bootstrap"` the table has the same columns and the same rows, `estimate` is still the fitted value, `standard_error` is the standard deviation of the replicates and `conf_low/conf_high` are the percentile interval. `statistic`, `p_value` and `p_adjusted` are NA throughout: putting a normal approximation back on top of the replicates the interval was read from is the assumption this path exists to avoid, so the interval is the inference. Its attributes are `covariance` (the covariance of the replicates, which `vcov()` returns), `level`, `method`, `iter`, `n_valid`, `replicates` (the kept replicates, one row each), `messages` (why a resample was dropped, NA where it was not), `structure` and `fixed`; `covariance_unconstrained` is NULL, because the estimation-scale chart is what this path does without.

Diagnostics of the fit as a whole travel as attributes rather than as columns repeated down every row: `covariance` and `covariance_unconstrained` (natural and estimation-scale covariance of the estimates), `hessian`, `gradient`, `scaled_score`, `condition_ratio`, `level`, `step`, `vcov_type`, `fixed` (the measurement blocks this fit held, `character()` when none) and `adjust`. When `vcov_type` is "robust", `group_scores` holds the groups-by-parameters score matrix and `scaling_correction` the MLR scaling correction factor $\text{tr}(A^{-1} B) / q$, used for scaled likelihood-ratio difference tests.

Conditions

`latents_no_converge` for an unconverged fit, `latents_boundary_fit` when a variance, a response probability or a mixing probability sits at its bound, `latents_singular_information` when the observed information cannot be inverted, `latents_no_free_parameters` when fixed held every parameter, `latents_bad_inference_data` when a supplied data does not reproduce the fit, and `latents_too_few_groups` for `vcov_type = "robust"` with fewer independent groups than reported parameters. A fit whose score is still far from zero is reported with a `latents_unconverged` warning rather than refused. A fit made with `multilpa(prior = )` is refused by the Wald path with `latents_unsupported_inference` (it sits at a posterior mode, where the likelihood's score is not zero), and a fit with a noise component by either path with `latents_unsupported_noise`.

`method = "bootstrap"` adds `latents_unsupported_inference` for a fit that holds a measurement block through `multilpa(fixed = )` — the held values came from another fit and resampling these data does not resample them; a `fit_staged()` result is bootstrapped by staging again instead — and `latents_bootstrap_failed` when fewer than two resamples produced a usable fit, whose message carries the first reason one gave. Resamples that fail or do not converge are dropped with a `latents_bootstrap_dropped` warning naming how many, rather than being silently left out of the count.

Examples

```
# Two separated profiles, membership driven by `x` and by the group's class.
set.seed(5)
school <- rep(seq_len(16), each = 8)
high_class <- rep(rep(c(FALSE, TRUE), length.out = 16), each = 8)
x <- rnorm(128)
```

```

profile <- ifelse(
  runif(128) < plogis(-1 + 2 * high_class + 0.8 * x), 2L, 1L
)
example_data <- data.frame(
  school = school, x = x,
  y1 = rnorm(128, ifelse(profile == 2L, 2, -2), 0.7),
  y2 = rnorm(128, ifelse(profile == 2L, 1.5, -1.5), 0.7)
)
fit <- mutilpa(example_data, c("y1", "y2"), "school", n_profiles = 2,
  n_group_classes = 2, profile_covariates = "x",
  n_starts = 2, seed = 1)
parameter_inference(fit, example_data)
set.seed(42)
example_data <- data.frame(
  school = rep(seq_len(10), each = 10),
  score_a = rnorm(100), score_b = rnorm(100)
)
fit <- mutilpa(example_data, c("score_a", "score_b"), "school",
  n_profiles = 2, n_group_classes = 1, n_starts = 2, seed = 1)
parameter_inference(fit)

# Many tests in one table: name the correction, do not apply one by stealth.
parameter_inference(fit, adjust = "BH")

# A structure that constrains the shape across profiles (VEI) is charted in
# its own free coordinates, so it has Wald standard errors too.
shaped <- mutilpa(example_data, c("score_a", "score_b"), "school",
  n_profiles = 2, n_group_classes = 1, n_starts = 2, seed = 1,
  volume = "varying", shape = "equal", orientation = "axis")
parameter_inference(shaped)

# The bootstrap resamples schools and refits inside the same family.
parameter_inference(shaped, method = "bootstrap", iter = 10, n_starts = 1,
  seed = 1)

```

```
parameter_inference.mutilpa_transitions
```

Wald inference for a latent transition model

Description

Standard errors, tests and intervals for every parameter of an `lta()` fit: the measurement model, the group-class shares, each group class's initial profile distribution and its transition matrix. The scores follow the Fisher identity (each is an expected count from the forward-backward pass minus the count the fitted probabilities imply), and the observed information is the numerical Jacobian of that analytic score.

Usage

```
## S3 method for class 'mutilpa_transitions'
```

```

parameter_inference(
  x,
  data = NULL,
  level = 0.95,
  step = 1e-04,
  vcov_type = c("observed", "robust", "opg"),
  adjust = .multilpa_p_adjust_methods,
  method = c("wald", "bootstrap"),
  iter = 199L,
  n_starts = 10L,
  max_iter = 1000L,
  tol = 1e-08,
  seed = NULL,
  boundary = c("error", "fix")
)

## S3 method for class 'multilpa_transitions'
vcov(
  object,
  data = NULL,
  step = 1e-04,
  vcov_type = c("observed", "robust", "opg"),
  scale = c("natural", "unconstrained"),
  ...
)

## S3 method for class 'multilpa_transitions'
confint(object, parm, level = 0.95, data = NULL, ...)

```

Arguments

<code>x</code>	A fitted <code>multilpa_transitions</code> model from <code>lta()</code> .
<code>data</code>	Optional. The data frame the model was fitted to. The fit stores its indicators, codes, groups and occasions, so it is not needed; when given it must reproduce them.
<code>level</code>	Confidence level of the intervals.
<code>step</code>	Finite-difference step for the observed information.
<code>vcov_type</code>	"observed" (observed information), "robust" (the sandwich clustered on groups, which are the sequences) or "opg" (the outer product of the group scores).
<code>adjust</code>	Multiplicity correction for <code>p_adjusted</code> , one of the methods <code>stats::p.adjust()</code> accepts.
<code>method</code>	"wald" (the default) or "bootstrap". The bootstrap resamples persons with replacement, refits the same specification, and matches each replicate's profiles and group classes to the original's; <code>standard_error</code> is the replicates' standard deviation, <code>conf_low</code> and <code>conf_high</code> their percentile interval, and <code>statistic</code> , <code>p_value</code> and <code>p_adjusted</code> are NA. It needs data.

iter, n_starts, max_iter, tol, seed	Bootstrap resamples, the controls of each refit, and an optional seed (the caller's random state is restored).
boundary	Accepted for the generic. Transition inference holds nothing at a bound: with no bound active "fix" gives the same table as "error", and a fit with an active bound raises latents_boundary_fit under either.
object	A fitted multilpa_transitions model.
scale	"natural" (probabilities) or "unconstrained" (logits).
...	Unused.
parm	Parameters to report, as names or positions of the natural coefficients; all by default.

Value

A base data.frame with one row per natural parameter and the columns of `parameter_inference()`: level, outcome, term, parameter, estimate, standard_error, statistic, p_value, p_adjusted, conf_low and conf_high, named as `coef()` names them. An initial probability (parameter = "initial_probability", level = "profile") has outcome the profile and term the group class; a transition probability ("transition_probability") has outcome the profile moved to and term group_class_h:profile_j, the group class and the profile moved from. Probabilities and variances carry no Wald test. Attributes covariance (natural scale), covariance_unconstrained, vcov_type and level.

Conditions

latents_no_converge for an unconverged fit; latents_boundary_fit when a variance, a response, initial or transition probability, or a group-class share sits at its bound, or a transition row was never informed (a profile no group occupies before its last occasion); latents_singular_information when the information cannot be inverted; latents_bad_inference_data when a supplied data does not reproduce the fit; latents_too_few_groups for robust or OPG errors with no more groups than parameters; latents_bad_argument for method = "bootstrap" without data.

Examples

```
fit <- lta(subset(course_engagement, student <= 40),
          c("browse", "lectures", "forum_read"), "student",
          n_profiles = 2, time = "sequence", n_starts = 2, seed = 1)
parameter_inference(fit)
```

```
plot.latents_family_enumeration
```

Plot latents results

Description

Every result of this package plots with `plot()`, which returns a `ggplot` object to print, save with `ggplot2::ggsave()` or restyle with `+ ggplot2::theme()`. The formerly base-graphics plots of mixture regressions, pooled imputations and bootstrap tests draw when called, including inside loops, and return their `ggplot` invisibly to avoid drawing twice at the console. The main views also have functions of their own, whose arguments an editor completes: `plot_profiles()` (profile means, or categorical means, medians and modes), `plot_responses()` (categorical response probabilities), `plot_prevalence()` (profile shares within each group class), `plot_diagnostics()` (classification quality), `plot_transitions()` (transition matrix), `plot_enumeration()` (information criteria, classification quality and model-selection evidence across a grid) and `plot_tree()` (how profiles split as more are added). Each draws exactly what the matching `plot(x, what = )` draws. `plot_views()` lists the views of a fitted model, and "enumeration" and "tree" for a grid, each with a one-line description; the other grid views are listed under what below.

Usage

```
## S3 method for class 'latents_family_enumeration'
plot(
  x,
  what = c("bic", "aic", "bic_individual"),
  main = NULL,
  subtitle = NULL,
  criterion = NULL,
  ...
)

## S3 method for class 'multilpa_additive'
plot(
  x,
  what = c("means", "variances", "intercepts"),
  level = 0.95,
  main = NULL,
  subtitle = NULL,
  ...
)

## S3 method for class 'latents_comparison'
plot(x, main = NULL, subtitle = NULL, ...)

## S3 method for class 'multilpa_cross_level'
plot(
  x,
  what = c("profiles", "group_means", "composition"),
  main = NULL,
  subtitle = NULL,
  ...
)
```

```
## S3 method for class 'multilpa_bootstrap_lrt'
plot(x, main = NULL, subtitle = NULL, palette = NULL, style = NULL, ...)

## S3 method for class 'latents_growth_mixture'
plot(
  x,
  what = c("trajectories", "individuals", "coefficients", "random", "classification",
    "posteriors"),
  time = NULL,
  facet = FALSE,
  persons = 12L,
  max_persons = 120L,
  spread = 0.8,
  level = 0.95,
  main = NULL,
  subtitle = NULL,
  ...
)

## S3 method for class 'latents_transition_enumeration'
plot(
  x,
  what = c("bic", "aic", "bic_individual"),
  main = NULL,
  subtitle = NULL,
  criterion = NULL,
  ...
)

## S3 method for class 'multilpa_lta'
plot(x, main = NULL, subtitle = NULL, ...)

## S3 method for class 'latents_mixture_regression'
plot(
  x,
  what = NULL,
  predictor = NULL,
  level = 0.95,
  main = NULL,
  subtitle = NULL,
  time = NULL,
  facet = FALSE,
  persons = 12L,
  max_persons = 120L,
  ...
)

plot_profiles(
```

```

    x,
    statistic = c("mean", "median", "mode"),
    scale = c("raw", "standardized"),
    intervals = TRUE,
    labels = TRUE,
    data = NULL,
    main = NULL,
    subtitle = NULL
)

plot_responses(
  x,
  category = "last",
  intervals = TRUE,
  labels = TRUE,
  data = NULL,
  main = NULL,
  subtitle = NULL
)

plot_prevalence(x, labels = TRUE, main = NULL, subtitle = NULL)

plot_diagnostics(
  x,
  what = c("all", "sizes", "posteriors", "entropy", "avepp"),
  main = NULL,
  subtitle = NULL
)

plot_transitions(x, main = NULL, subtitle = NULL)

plot_tree(x, main = NULL, subtitle = NULL)

## S3 method for class 'multilpa'
plot(
  x,
  what = c("profiles", "bars", "heatmap", "raincloud", "parallel", "pairs", "responses",
    "probabilities", "sequences", "sizes", "entropy", "posteriors", "avepp", "all"),
  data = NULL,
  scale = c("raw", "standardized"),
  category = "last",
  labels = TRUE,
  intervals = TRUE,
  cell_labels = TRUE,
  main = NULL,
  subtitle = NULL,
  statistic = c("mean", "median", "mode"),
  ...

```

```

)

## S3 method for class 'multilpa_enumeration'
plot(
  x,
  what = c("aic", "bic_groups", "bic_individual", "icl_individual"),
  criterion = NULL,
  combine = TRUE,
  labels = TRUE,
  mark_minimum = TRUE,
  main = NULL,
  subtitle = NULL,
  cutoff = 0.8,
  using = "bic_groups",
  ...
)

plot_enumeration(
  x,
  what = c("aic", "bic_groups", "bic_individual", "icl_individual"),
  combine = TRUE,
  labels = TRUE,
  mark_minimum = TRUE,
  main = NULL,
  subtitle = NULL,
  cutoff = 0.8,
  using = "bic_groups"
)

## S3 method for class 'multilpa_covariates'
plot(
  x,
  what = c("profiles", "bars", "heatmap", "raincloud", "parallel", "pairs", "responses",
    "sequences", "sizes", "entropy", "posteriors", "avepp", "all"),
  data = NULL,
  scale = c("raw", "standardized"),
  category = "last",
  labels = TRUE,
  intervals = TRUE,
  cell_labels = TRUE,
  main = NULL,
  subtitle = NULL,
  statistic = c("mean", "median", "mode"),
  ...
)

plot_views()

```

```
## S3 method for class 'latents_pooled'
plot(
  x,
  what = c("profile", "group", "measurement"),
  main = NULL,
  subtitle = NULL,
  level = NULL,
  ...
)

## S3 method for class 'multilpa_transitions'
plot(
  x,
  what = c("transitions", "profiles", "bars", "heatmap", "responses", "sequences",
    "sizes", "entropy", "posteriors", "avepp", "all"),
  data = NULL,
  scale = c("raw", "standardized"),
  category = "last",
  labels = TRUE,
  cell_labels = TRUE,
  main = NULL,
  subtitle = NULL,
  ...
)
```

Arguments

<code>x</code>	The object to plot: a profile, class or transition model for the named fit views and <code>plot_diagnostics()</code> , an enumeration from <code>enumerate_classes()</code> , <code>enumerate_lpa()</code> or <code>enumerate_lca()</code> for <code>plot_enumeration()</code> and <code>plot_tree()</code> , a transition fit from <code>lta()</code> for <code>plot_transitions()</code> .
<code>what</code>	For <code>plot_diagnostics()</code> : "all" (default), "sizes" (effective profile counts), "posteriors" (posterior probability of each case's assigned profile), "entropy" (each case's entropy relative to a flat posterior) or "avepp" (average posterior probability matrix). A noise fit supports "sizes" only; "all" returns that view, and the other diagnostics refuse with <code>latents_unsupported_noise</code> . For <code>plot_enumeration()</code> : criterion columns of <code>as.data.frame(x)</code> , such as "bic_groups", or "all"; the default draws AIC, both BICs and ICL over individuals. Or classification quality across the candidates, in the words of a single model's plot: "entropy" (relative entropy), "avepp" (the lowest AvePP, with the cutoff dashed) or "diagnostics" (both), at each level with two or more classes. Or model-selection evidence, none of which selects a model: "deltas" (how much the criterion in using fell with each added profile and each added group class; see <code>get_results(x, "deltas")</code> and "knees"), "heatmap" (the criterion over profiles and group classes, above the lowest), "smallest" (the smallest class share, 5% dashed) or "trust" (convergence, boundary and replication of every candidate); "selection" draws all of them and "diagnostics", as a list.

main, subtitle	Title and subtitle. NULL uses the view's own.
criterion	The former name for enumeration criteria. On a profile grid, use alone or with what = "enumeration"; other grid families use "bic", "aic" or "bic_individual".
...	Additional arguments. Profile/class plot methods refuse unknown names; methods that ignored them in 0.9.21 warn with latents_unused_argument, as described below.
level	Confidence level for additive, regression and growth views (default 0.95). On pooled plots, the former name for what: "profile", "group" or "measurement".
palette	For bootstrap plots, a colour vector; NULL uses the standard histogram and observed-statistic colours.
style	The former bootstrap drawing style; unused with a warning.
time	For growth/trajectory views, the numeric time column. NULL uses the first numeric variable in the fitted random or regression design.
facet	For trajectories, FALSE draws all classes together; TRUE draws one panel per class.
persons	For individual trajectories, the number of persons to show (default 12), or a vector of their identifiers.
max_persons	The maximum observed trajectories behind the class means (default 120); zero hides them.
spread	For growth trajectories, the random-effect spread of persons around the class curve, from zero (hidden) to less than one; default 0.8.
predictor	For the fitted mixture-regression view, a numeric predictor in the formula. NULL uses its first numeric predictor.
statistic	"mean" (default), "median" or "mode". They coincide for continuous indicators; for categorical ones they summarise the fitted probabilities on the numeric category scale, without intervals.
scale	"raw" keeps each indicator in its own units; "standardized" divides its deviation from the observed mean by its observed standard deviation.
intervals	For profile, bar and response views, TRUE draws 95% intervals when the fit has standard errors; otherwise the subtitle says why there are none.
labels	TRUE labels each series directly; FALSE uses a legend.
data	Optional. The data the model was fitted to, used only to compute intervals from a different frame.
category	Which category's probability to draw: "last" (default), "first", or a single label or index.
cell_labels	For sequence views, TRUE prints each assigned profile in its cell while the display is sparse enough to hold the text.
combine	TRUE draws several criteria as panels of one plot; FALSE returns one plot per criterion.
mark_minimum	TRUE rings each criterion's lowest value among converged candidates. It marks an extremum; it does not select a model. Information criteria only.
cutoff	For plot_enumeration(what = "avepp"), "diagnostics" or "selection", the AvePP cutoff drawn as a dashed line.

using For `plot_enumeration(what = "deltas")`, "heatmap" or "selection", the information criterion they read (default "bic_groups"); "smallest" and "trust" do not use one.

Details

Profiles keep one order (largest first), one set of Okabe-Ito colours and one point shape each across every view. The plots need `ggplot2`, which `latents` suggests rather than requires.

Value

A `ggplot` object. `plot_diagnostics(what = "all")`, `plot(fit, what = "all")` on a fitted model, `plot_enumeration(combine = FALSE)` with several criteria, and the enumeration views when more than one is asked for (including "selection") return a `latents_plots` list of them, which draws every one when printed. `plot_views()` returns a `data.frame` with one row per view: type (the value for `what`), group and description.

What `plot()` draws

Each object takes `what =` for its views, main and subtitle, and the arguments named here. Profile/class methods refuse an argument they do not use with `latents_bad_argument`; methods that ignored it in 0.9.21 instead draw with a `latents_unused_argument` warning, as listed below.

Fitted profile and class models From `multilpa()`, `multilca()`, `lpa()`, `lca()`, covariate models and `lta()`: "profiles" (default), "bars", "heatmap", "raincloud", "parallel", "pairs" (each profile's 95% covariance ellipse), "responses", "probabilities" (not for covariate models), "sequences" (fits with time; `cell_labels = TRUE` prints the profile in each cell), "transitions" (transition fits), "sizes", "entropy", "posteriors", "avepp" and "all". Transition fits draw point estimates and take no intervals or statistic.

Enumerations Criterion columns or "all" (with `combine`, `labels` and `mark_minimum`), "tree", the classification-quality views "entropy", "avepp" and "diagnostics" (with `cutoff`), and the model-selection views "deltas" and "heatmap" (which read the criterion in `using`), "smallest" and "trust"; "selection" draws those four and "diagnostics". Transition and group-class family grids take "bic" (default), "aic" or "bic_individual".

Additive fits "means" (each group class's mean with its Wald interval), "variances" (within-beside between-group variances) or "intercepts" (each group's posterior intercept); level sets the interval.

Cross-level fits "profiles", "group_means" or "composition" (profile shares within each group class).

Mixture regressions "fitted" (default: the outcome against a numeric predictor, one line per class), "coefficients" or "posteriors"; level sets the intervals. Fits that classify persons (`class_level = "group"`) draw the growth views below instead.

Growth mixtures "trajectories" (default: class mean trajectories with bands; `facet`, `max_persons`, `spread`), "individuals" (persons: a number or identifiers), "coefficients", "random" (predicted random effects with each class's 95% ellipse) or "classification"; time names the variable drawn along, level the band.

Pooled imputations "profile" (default), "group" or "measurement" parameters: each imputation's estimate beside the pooled one with its interval.

Model comparisons, bootstrap tests, diagnostics One view each: the criteria of each compared model, the simulated null distribution with the observed statistic, and the four views of `plot_diagnostics()`.

Names from latents 0.9.21

Code written for that release keeps working. Enumeration plots still take `criterion` (now `what`; on a profile grid `what = "enumeration"` is the default criteria view), a pooled fit's plot still takes `level` (now `what`), and a bootstrap test's plot still takes `palette` (the histogram and observed-line colours). On an enumeration, `criterion` with any `what` other than `"enumeration"` is an error of class `latents_bad_argument` (with `"tree"`, a warning, as the tree uses no criterion); `level` together with `what` on a pooled fit is an error of the same class.

Errors

`latents_missing_package` without `ggplot2`; `latents_bad_argument` for an argument the object's plot does not take; the plots of additive, cross-level, transition (`lta()`), growth-mixture, mixture-regression, pooled, comparison, bootstrap and transition or group-class family grids instead warn with class `latents_unused_argument` and draw, as they ignored such arguments in 0.9.21 (this includes a bootstrap plot's former style); `latents_nothing_to_plot` when the fit lacks what a view needs (for example posteriors or entropy of a one-profile fit); `latents_no_continuous`, `latents_no_categorical` and `latents_no_time` when a view needs indicators or occasions the fit does not have; `latents_unknown_criterion` for a criterion not in the grid.

References

Satopaa, V., Albrecht, J., Irwin, D. and Raghavan, B. (2011). Finding a "kneedle" in a haystack: detecting knee points in system behavior. *31st International Conference on Distributed Computing Systems Workshops*, 166-171.

Zappia, L. and Oshlack, A. (2018). Clustering trees: a visualization for evaluating clusterings at multiple resolutions. *GigaScience*, 7(7), giy083.

Examples

```
plot_views()
if (requireNamespace("ggplot2", quietly = TRUE)) {
  fit <- multilpa(course_engagement, c("browse", "lectures", "forum_read"),
    "student", n_profiles = 2, n_group_classes = 2,
    n_starts = 2, seed = 1)
  plot_profiles(fit)
  plot_diagnostics(fit, what = "avepp")
}
```

Description

Fits `multilpa()` to every completed data set, aligns each fit's profile and group-class labels to the first one, and pools the estimates with Rubin's rules. This is the route for missing *covariates*, which `missing = "fiml"` cannot integrate out: the model conditions on covariates, so it has no distribution for them to integrate over. Impute them first (for example with the `mice` package, including the indicators and the group identifier in the imputation model), then pass the completed data here.

Usage

```
pool_imputations(
  imputed,
  vars,
  id,
  n_profiles,
  ...,
  level = 0.95,
  vcov_type = c("observed", "robust", "opg"),
  adjust = .multilpa_p_adjust_methods
)
```

Arguments

<code>imputed</code>	The completed data sets: a list of at least two data frames with the same columns and rows in the same order, or a <code>mids</code> object from the <code>mice</code> package (which is then completed here).
<code>vars, id, n_profiles</code>	As in <code>multilpa()</code> .
<code>...</code>	Further arguments for <code>multilpa()</code> , such as <code>n_group_classes</code> , <code>profile_covariates</code> , <code>group_covariates</code> , <code>categorical</code> , <code>n_starts</code> and <code>seed</code> . The same arguments fit every imputation.
<code>level</code>	Confidence level of the pooled intervals.
<code>vcov_type, adjust</code>	As in <code>parameter_inference()</code> , applied to every imputation's fit.

Details

Pooling is on the reported (natural) scale: for each parameter the pooled estimate is the mean over imputations, and its variance is the mean within-imputation variance plus $(1 + 1/m)$ times the between-imputation variance. Degrees of freedom follow Rubin (1987) with a large complete-data sample, which is what maximum-likelihood inference assumes. Pooled intervals are symmetric t intervals on this scale; unlike the transformed Wald intervals of `parameter_inference()`, they can extend outside the parameter space for probabilities or variances. A structural constant, such as the sole class's probability of one, has zero within- and between-imputation variance and zero missing information. If within-imputation variance is zero but between-imputation variance is positive, `riv` is infinite and `fmi` is one.

Value

An object of class `latents_pooled`. `as.data.frame()` and `get_results(x, "estimates")` give one row per parameter, with the columns of `parameter_inference()` (`level`, `outcome`, `term`, `parameter`, `estimate`, `standard_error`, `statistic`, `p_value`, `p_adjusted`, `conf_low`, `conf_high`) plus `df` (Rubin's degrees of freedom), `within` and `between` (the two variance components), `riv` (the relative increase in variance due to missingness) and `fmi` (the fraction of missing information). `statistic` and `p_value` are NA where `parameter_inference()` reports no test (variances and probabilities). `get_results(x, "imputations")` gives each imputation's aligned estimates, one row per imputation and parameter, and `get_results(x, "fits")` one row per imputation with its log likelihood, convergence and the label permutation that aligned it.

Choosing the imputation model

Rubin's rules are only as good as the imputations. The covariate and the indicators are related through the latent classes, so an imputation model that is linear in the indicators (mice's defaults) misses that structure: in `validation/pool-imputations-coverage.R` (a membership slope with 30% of its covariate missing at random given the indicators, 200 replications, ten imputations) 95% intervals covered 0.925 under predictive mean matching and 0.935 under Bayesian linear regression, against 0.995 when the covariate was approximately drawn from its conditional distribution using known generating parameters. That oracle arm omits uncertainty in the imputation parameters and is not a validation of proper multiple imputation or nominal interval coverage. The study is one design, not a general guarantee for any method. Use an imputation model that accounts for the latent structure and clustering, includes the indicators and relevant nonlinearities, and propagates parameter uncertainty. Imputing separately within assigned classes alone ignores classification uncertainty. Complete-case coverage was 0.955 in that design, but selecting on the indicators changes their within-class distributions; the usual logistic case-control slope argument does not guarantee unbiased slopes when the outcome is latent and its measurement model is estimated.

Conditions

`latents_bad_data` when imputed is not a list of at least two data frames with identical columns and row counts; `latents_pooling_failed` when an imputation's fit or inference fails (its message names the imputation and carries the original reason), since Rubin's rules need every imputation; `latents_missing_package` for a `mids` object without `mice` installed. Warnings from the individual fits are passed on as they are raised.

References

- Rubin, D. B. (1987). *Multiple Imputation for Nonresponse in Surveys*. Wiley.
- Meng, X.-L. (1994). Multiple-imputation inferences with uncongenial sources of input. *Statistical Science*, 9(4), 538–558.
- van Buuren, S. (2018). *Flexible Imputation of Missing Data* (2nd ed.), section 5.2. Chapman & Hall/CRC.

Examples

```
set.seed(1)
# Two stand-in completed data sets. With real missing covariates, pass the
```

```
# `mids` object mice::mice() returns instead.
completed <- lapply(1:2, function(i) {
  within(subset(course_engagement, student <= 40),
    previous_grade <- previous_grade + rnorm(length(previous_grade), sd = 0.1))
})
pooled <- pool_imputations(completed, c("browse", "lectures", "forum_read"),
  "student", n_profiles = 2, n_group_classes = 1,
  profile_covariates = "previous_grade",
  n_starts = 2, seed = 1)

pooled
get_results(pooled, "fits")
```

prior_control

Conjugate prior for Gaussian mixture estimation

Description

Requests maximum a posteriori (MAP) estimation of the Gaussian profile means and covariances under the conjugate prior of Fraley and Raftery (2007), exactly as `mclust::priorControl()` does for `mclust::Mclust()`. Pass the result as `multilpa(prior = )`. The prior keeps covariances away from singularity, which is what lets a many-profile or many-indicator model be fitted where maximum likelihood degenerates.

Usage

```
prior_control(shrinkage = 0.01, mean = NULL, dof = NULL, scale = NULL)
```

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

Arguments

shrinkage	Non-negative prior precision factor for the means, kappa; mclust's default 0.01. Zero leaves the means unshrunk.
mean	Optional prior mean, one value per continuous indicator, in the indicators' own units. Naming one needs a positive shrinkage.
dof	Optional prior degrees of freedom, a single number.
scale	Optional prior scale: a single positive number for an axis-parallel or spherical structure, or a positive-definite d by d matrix for an ellipsoidal one.
x	A <code>latents_prior</code> object.
...	Ignored.

Details

Every argument left NULL takes `mclust::defaultPrior()`'s value, computed from the indicators being fitted: the prior mean is their column means, the degrees of freedom are $d + 2$ for d indicators, and the scale is $(1/G)^{(2/d)}$ times the sample covariance (ellipsoidal structures) or times the average sample variance (axis-parallel and spherical structures), for G profiles.

The updates follow `mclust`'s fitted-model convention, including its structure-specific departures from a textbook conjugate update. For EII, scale does not enter the variance update. For EEI, shrinkage and mean enter the variance calculation but do not shrink the fitted means. For VEV, dof does not enter the update. EEI, EVI and EEV count the mean-prior contribution once rather than once per profile. These conventions are retained for agreement with `mclust`; the controls therefore have different effects across covariance structures.

Value

An object of class `latents_prior`: a list with `shrinkage`, `mean`, `dof` and `scale`, the last three NULL where the default is to be computed from the data.

`print()` returns `x` invisibly.

Conditions

Raises `latents_bad_argument` for a negative or non-finite shrinkage, a mean with a zero shrinkage, or a non-positive dof or scale.

References

Fraley, C., & Raftery, A. E. (2007). Bayesian regularization for normal mixture estimation and model-based clustering. *Journal of Classification*, 24, 155–181. doi:10.1007/s0035700700045

See Also

`multilpa()`, whose prior argument takes this.

Examples

```
prior_control()
prior_control(shrinkage = 0)
```

r3step

Covariates predicting class membership, corrected for misclassification

Description

The R3STEP approach. A multinomial logit of class membership on covariates, fitted after the measurement model rather than alongside it, but treating the assigned class as an error-prone indicator of the true one with the error rates held fixed at what step one found. Regressing the modal class directly instead attenuates every coefficient, because some units are in the wrong class and the covariate cannot explain why.

Usage

```
r3step(
  x,
  data,
  covariates,
  level = c("individuals", "groups"),
  ci_level = 0.95,
  vcov_type = c("robust", "observed"),
  adjust = c("BH", "holm", "hochberg", "hommel", "bonferroni", "BY", "none"),
  by_group_class = FALSE
)
```

Arguments

<code>x</code>	A covariate-free fitted <code>multilpa()</code> or <code>lta()</code> model. A fit that already uses membership covariates cannot be corrected by the fixed, unconditional classification-error matrix used here.
<code>data</code>	The data frame carrying the covariates, in the fit's row order. Shared fit columns are checked row by row; a frame with too little identifying information warns that alignment cannot be verified.
<code>covariates</code>	Character vector of numeric columns not used as measurement indicators in <code>x</code> . For <code>level = "groups"</code> each must be constant within a group.
<code>level</code>	"individuals" predicts profile membership, "groups" predicts group-class membership.
<code>ci_level</code>	Confidence level for the intervals.
<code>vcov_type</code>	"robust", the default, uses the sandwich clustered on the fit's groups, which is the honest choice when the covariates are measured on observations nested inside them, and matches the default of <code>three_step()</code> . "observed" uses the observed information; for the pooled regression it treats every observation as independent, and with <code>by_group_class = TRUE</code> it is the model-based variance of the two-level likelihood. "robust" needs more independent groups than the regression has coefficients, because the group score contributions sum to zero at the estimate and so span at most one dimension fewer than there are groups; with too few it is refused with <code>latents_too_few_groups</code> rather than reporting a variance that is singular, or, with a single group, numerically zero. "observed" remains available there.
<code>adjust</code>	Multiplicity correction applied across the covariate terms, passed to <code>stats::p.adjust()</code> . "BH" by default; "none" leaves the p-values uncorrected. The family is every covariate term of every non-reference class, which is the set of tests this call computes; the intercepts are not part of it.
<code>by_group_class</code>	FALSE, the default, fits one pooled multinomial logit with a single intercept per profile. TRUE fits the two-level form of the one-step model instead: every group class has its own profile intercepts, the covariate slopes are shared across group classes, and a group's observations share one latent group class whose proportions are estimated alongside. Differences between groups that the covariates do

not explain are then absorbed by the group classes rather than left in the residual, where the pooled regression attenuates the slopes and inflates their clustered standard errors. For `level = "individuals"` on a covariate-free `multilpa()` fit only; refused with `latents_bad_argument` otherwise.

Details

The error matrix is held fixed rather than estimated jointly, which is what makes this a three-step method and what keeps the covariates from reshaping the classes. On a `lta()` fit, `level = "individuals"` models the marginal profile at each occasion. It does not regress initial-state or transition probabilities on the covariates. `vcov_type = "robust"` clusters the occasion scores by group; "observed" treats them as independent.

Over 60 replications of a two-profile design with a true log-odds slope of 1.2 and intercept -0.3, this recovered the slope with a bias of -0.002 and the intercept with a bias of +0.006, where a logistic regression on the modal class was biased by -0.163 on the slope. The correction costs some precision: the standard deviation of the slope across replications was 0.127 against 0.100 for the naive fit. The nominal 95% intervals covered the true slope in 95% of those replications.

That coverage was measured where the profiles separate well. The error matrix is still treated as known rather than estimated, so intervals should be expected to run narrow where classification is poorer; check `get_results(x, "classification_errors")` before relying on them.

Value

A base `data.frame` with one row per non-reference class and term, and the columns `level`, `outcome`, `term`, `estimate`, `standard_error`, `statistic`, `p_value`, `p_adjusted`, `conf_low` and `conf_high`. `p_value` is uncorrected and `p_adjusted` carries the correction named by `adjust`, which is also recorded in the result's `adjust` attribute; it is NA on the intercept rows, which are not part of the tested family. A coefficient with zero standard error has an undefined Wald statistic and p-value, reported as `NA_real_` rather than infinity or zero. Coefficients are log odds against the final class, which is the reference, matching `multilpa(profile_covariates = )`; that class is named in the result's `reference_class` attribute, and the variance that was used in its `vcov_type` attribute. With `by_group_class = TRUE` the intercept rows are one per group class, named `group_class_1`, `group_class_2`, ..., followed by the shared slopes and, at `level = "groups"`, the group-class logits against the last group class; the result also carries `by_group_class` and the maximised `log_likelihoood` as attributes.

References

Vermunt, J. K. (2010). Latent class modeling with covariates: two improved three-step approaches. *Political Analysis*, 18, 450–469. Asparouhov, T., & Muthen, B. (2014). Auxiliary variables in mixture modeling: three-step approaches using Mplus. *Structural Equation Modeling*, 21, 329–341.

See Also

`three_step()` for a distal outcome, and `multilpa(profile_covariates = )` for the one-step alternative that estimates everything jointly.

Examples

```
set.seed(21)
g <- rep(seq_len(50), each = 12)
x <- rnorm(600)
truth <- 1L + as.integer(runif(600) < plogis(-0.3 + 1.2 * x))
example_data <- data.frame(
  g = g, x = x,
  a = rnorm(600, ifelse(truth == 2L, 1.2, -1.2)),
  b = rnorm(600, ifelse(truth == 2L, 1.2, -1.2))
)
fit <- multilpa(example_data, c("a", "b"), "g", n_profiles = 2,
  n_group_classes = 1, n_starts = 4, seed = 1)
r3step(fit, example_data, "x")
```

report

Everything about a fit, in one call

Description

Prints the estimates, then the classification diagnostics, then draws every plot the fit supports. A first look, not a substitute for the verbs: each section is what [summary\(\)](#), [diagnostics\(\)](#) and [plot\(\)](#) return, and the tables are reachable from those rather than from here.

Usage

```
report(
  x,
  data = NULL,
  plots = TRUE,
  by = c("profile", "overall"),
  rows = 10L,
  ...
)
```

Arguments

<code>x</code>	A fitted model of this package.
<code>data</code>	Optional. The data the model was fitted to; a fit carries the columns it was built from.
<code>plots</code>	TRUE, the default, draws the plots. FALSE prints only.
<code>by</code>	Passed to diagnostics() , and from there to the bivariate residuals: "profile", the default, assesses each profile separately, "overall" pools them.
<code>rows</code>	How many rows of each of the summary's tables to print, passed to <code>print(summary(x))</code> . A first look at a fit with thousands of observations would otherwise be mostly posteriors.
<code>...</code>	Nothing further is accepted. An argument this function cannot forward raises an error of class <code>latents_bad_argument</code> naming it, before anything has been printed, rather than being dropped.

Value

The fitted model, invisibly. Called for the printing and drawing.

What it prints

`summary()`, which is every table the fit can produce, then `descriptives()`, then the condensed reading of `diagnostics()`, then every plot view the fit supports. Each section is what that verb returns, and the tables are reached from `get_results()` rather than from here.

See Also

`summary()`, `diagnostics()`, `descriptives()`, `plot_views()`.

Examples

```
fit <- multilpa(
  course_engagement,
  vars = c("browse", "lectures", "forum_read", "forum_post", "attendance"),
  id = "student", n_profiles = 2, n_group_classes = 2, n_starts = 4,
  seed = 1
)
report(fit, plots = FALSE)
```

sensitivity

Does the solution survive a different seed?

Description

A latent profile model is fitted by EM from random starts, so what comes back is a local maximum reached from one particular set of them. `n_starts` reports how many starts *within a single seed's stream* reached the best likelihood; it cannot say whether a different stream would have found a different mode. This verb refits the model under each of several seeds and reports what changed.

Usage

```
sensitivity(
  x,
  data = NULL,
  seeds = 1:10,
  n_starts = NULL,
  max_iter = NULL,
  tol = NULL,
  tolerance = NULL
)
```

Arguments

<code>x</code>	A fitted multilpa model to use as the reference. A <code>fit_staged()</code> result varies the second-stage starts while holding its first-stage measurement solution fixed. A directly fitted model with measurement blocks held is refused: its original free starting values are not retained, so replaying the same seed would not replay the same search.
<code>data</code>	Optional. The data the model was fitted to, in its original row order. A fit carries the columns it was built from, so this is only needed to override them. Shared identifier and indicator columns are checked against the fit before assignment agreement is calculated.
<code>seeds</code>	Seeds to refit under: at least two distinct whole numbers of either sign, as <code>set.seed()</code> accepts. Anything else raises <code>latents_bad_argument</code> . The reference fit's own seed may be among them, in which case that row reproduces it and is the arithmetic check that the refit is the same model.
<code>n_starts</code>	Starts per refit. Defaults to the number the reference fit used, so each seed gets the same search effort it did.
<code>max_iter, tol</code>	Passed to each refit; default to the reference fit's.
<code>tolerance</code>	Two log likelihoods within this of each other count as the same optimum. Defaults to the reference fit's own replication tolerance, which is what it used to decide whether its starts agreed.

Details

Three things are worth reading off the result. Whether every seed reached the same maximised log likelihood, which is what optimum counts. Whether the seeds that reached it converged. And how many observations were assigned to a different profile, which is what agreement measures, after the arbitrary profile labels of each refit have been matched to the reference fit's – two fits of the same mixture can be identical and still number their profiles differently, so comparing labels directly would report disagreement that is not there.

A seed whose refit fails contributes a row with NA estimates rather than being dropped silently, and a `latents_sensitivity_dropped` warning names how many failed, so the table is never quietly shorter than seeds.

Value

A base `data.frame`, one row per seed, with columns `seed`, `log_likelihood`, `converged`, `iterations`, `optimum` (1 for the best maximum found across the seeds, 2 for the next distinct one, and so on), `best` (whether this seed reached optimum 1) and `agreement` (the proportion of observations given the same profile as the reference fit, after label alignment). A failed refit has NA throughout except `seed`.

References

Hipp, J. R., & Bauer, D. J. (2006). Local solutions in the estimation of growth mixture models. *Psychological Methods*, 11, 36–53.

See Also

[enumerate_classes\(\)](#) to vary the number of classes rather than the seed, and [parameter_inference\(\)](#) for uncertainty about the parameters of a single solution.

Examples

```
set.seed(7)
example_data <- data.frame(
  school = rep(seq_len(12), each = 10),
  score_a = c(rnorm(60, -1.5), rnorm(60, 1.5)),
  score_b = c(rnorm(60, -1), rnorm(60, 1))
)
fit <- multilpa(example_data, c("score_a", "score_b"), "school",
  n_profiles = 2, n_group_classes = 1, n_starts = 3, seed = 1)
sensitivity(fit, seeds = 1:3, n_starts = 3)
```

```
simulate.latents_growth_mixture
```

Predict from and simulate a fitted model

Description

`predict()` evaluates a fit on new rows. For [multilpa\(\)](#) and [multilca\(\)](#) fits without membership covariates it is the E-step at the estimates, applied through the fit's own preparation (categorical levels, centering, missing-data handling), so predicting the training rows reproduces the fit's posteriors exactly. In a two-level model the groups in newdata are new groups, each classified from its own rows. For [mixture_regression\(\)](#) fits it returns fitted outcomes or class probabilities. Growth mixtures refuse with `latents_unsupported_prediction`; their fitted curves are the "trajectories" and "individual" tables of [get_results\(\)](#).

Usage

```
## S3 method for class 'latents_growth_mixture'
simulate(object, nsim = 1, seed = NULL, ...)

## S3 method for class 'latents_growth_mixture'
predict(object, ...)

## S3 method for class 'latents_mixture_regression'
predict(
  object,
  newdata = NULL,
  type = c("response", "class_response", "posterior", "probabilities"),
  ...,
  trials = NULL
)
```

```
## S3 method for class 'latents_mixture_regression'
simulate(object, nsim = 1, seed = NULL, ...)

## S3 method for class 'multilpa'
predict(object, newdata = NULL, type = c("class", "posterior", "density"), ...)
```

Arguments

<code>object</code>	A fitted model.
<code>nsim</code>	Number of simulated outcome vectors.
<code>seed</code>	NULL uses and advances the caller's random stream. An integer makes simulation reproducible and restores the caller's random state.
<code>...</code>	Unused.
<code>newdata</code>	NULL for the fitted rows, or a data frame with the indicator or predictor columns (and id and membership covariates where the model uses them).
<code>type</code>	For a profile or class model: "class" (modal profile and group class with their posteriors), "posterior" (every profile's probability as well) or "density" (each row's log density, marginal over group classes, for scoring or held-out likelihood). For a mixture regression: "response" (the mean outcome over classes; expected successes for a binomial count response), "class_response" (one row per row and class), "posterior" (class probabilities given the outcome, which newdata must contain) or "probabilities" (each category under each class; ordinal family only).
<code>trials</code>	For binomial mixture-regression predictions, the number of trials: one non-negative whole number, a vector with one value per row, or a column name in the prediction data. NULL uses the fitted trials when newdata = NULL, or infers them from the response columns when present. A count response otherwise requires trials; a Bernoulli fit defaults to one trial. For type = "posterior", any explicit value must agree with the observed response. Predictions remain on the expected success-count scale; trials = 1 requests probabilities for each trial.

Details

`simulate()` draws new outcomes on the fitted design of a mixture regression or growth mixture: class memberships from the fitted mixing model (group classes first in a two-level model), random effects from each class's covariance, then outcomes from each row's class regression.

Value

A base `data.frame` with one row per row of newdata (per row and class for "class_response", per row, class and category for "probabilities"), starting with row, its position. `simulate()` returns one row per fitted row and one column per simulation, `sim_1`, `sim_2`, ...

Conditions

`latents_bad_data` when newdata lacks an indicator or the id column, has a category the fit never saw, or a missing value the fit cannot handle, or a regression design cannot be evaluated. Binomial

count predictions without trial counts also raise `latents_bad_data`; `latents_bad_argument` for "probabilities" outside the ordinal family; `latents_unsupported_prediction` for `predict()` on a growth mixture.

Examples

```
fit <- mixture_regression(score ~ hours, data = study_hours, n_classes = 2,
                        n_starts = 2, seed = 1)
predict(fit, data.frame(hours = c(2, 10)), type = "class_response")
head(simulate(fit, nsim = 2, seed = 1))
```

srl

Self-regulated learning scores simulated by a large language model

Description

Scores on five constructs of the Motivated Strategies for Learning Questionnaire (MSLQ) for 300 simulated respondents. Each score is the mean of the construct's Likert items, on a scale from 1 to 7. The responses were **generated by a large language model (GPT)**, not collected from students: they come from a study of how language models answer psychological surveys, and are used here because they are a well-structured, openly shareable cross-sectional example. They should not be read as evidence about students.

Usage

```
srl
```

Format

A data frame with 300 rows and 5 columns:

```
cognitive_strategies cognitive strategy use
intrinsic_value intrinsic value of the course
self_efficacy self-efficacy
self_regulation self-regulation
test_anxiety test anxiety
```

Details

The five constructs are strongly correlated (about 0.8, except test anxiety), which makes the data a good illustration of how the choice of covariance structure shapes a latent profile analysis.

Source

The SRL_GPT data of the `psychnets` package, with descriptive column names. Saqr, M. (2025). Delving into the psychology of machines: Exploring the structure of self-regulated learning via LLM-generated survey responses. *Computers in Human Behavior*, 173, 108769. doi:10.1016/j.chb.2025.108769

Examples

```
descriptives(srl, names(srl))
```

starting_values	<i>Build starting values for a multilevel latent profile fit</i>
-----------------	--

Description

Returns the starting-value list `multilpa()` accepts, taken from a fitted model or from any object that already carries the parameter blocks, such as a retained reference solution. This exists so that callers never assemble the list by hand, and so that starting values are validated where they are built rather than deep inside the fitting loop.

Usage

```
starting_values(
  x,
  covariance = c("auto", "drop", "keep"),
  what = c("all", "measurement")
)
```

Arguments

x	A fitted multilpa model, or a list carrying <code>profile_probabilities</code> , <code>group_probabilities</code> , and whichever measurement blocks the model uses: means with variances or covariances for continuous indicators, <code>response_probabilities</code> for categorical ones. An all-categorical model needs no Gaussian block. Any other elements are ignored.
covariance	"auto" keeps covariances when the object carries them, "drop" always returns the diagonal parameterization, and "keep" requires covariances and fails when they are absent.
what	"all" returns the measurement and mixing blocks. "measurement" omits <code>profile_probabilities</code> and <code>group_probabilities</code> , which are sized for the model that produced them and must not be carried into a staged fit that changes the number of group classes. Pass the result as <code>start</code> alongside <code>fixed</code> in <code>multilpa()</code> .

Value

An object of class `multilpa_start`: the list `multilpa()` accepts as `start`, with elements `means`, `variances`, `profile_probabilities`, and `group_probabilities`, plus covariances when the full-covariance parameterization is returned and `response_probabilities` when the object carries a categorical measurement model. `what = "measurement"` drops `profile_probabilities` and `group_probabilities`. The Gaussian blocks are unnamed and indexed by position, matching what `multilpa()` expects of `start`; read them with `as.data.frame()`, which labels the positions, rather than out of the list. `response_probabilities` is the exception: the list keeps each indicator's name and each block keeps its category names, because those labels are what lets `multilpa()` attach a

block to the item it was estimated for rather than to whichever item happens to sit in that position. A fit that names its categorical indicators in another order, or that has other categories, therefore refuses the start with `latents_bad_start` instead of reporting a likelihood for a measurement model it never held. The list payload is unchanged by the class, so a `multilpa_start` can be passed straight back as `start`.

See Also

`get_results()` for the values as tidy tables.

Examples

```
set.seed(7)
example_data <- data.frame(
  school = rep(seq_len(12), each = 10),
  score_a = rnorm(120), score_b = rnorm(120)
)
fit <- multilpa(example_data, c("score_a", "score_b"), "school",
               n_profiles = 2, n_group_classes = 1, n_starts = 2, seed = 1)
starting_values(fit)
as.data.frame(starting_values(fit))
refit <- multilpa(example_data, c("score_a", "score_b"), "school",
                  n_profiles = 2, n_group_classes = 1, n_starts = 1,
                  start = starting_values(fit))
logLik(refit)
```

student_esm

Leisure activities of university students in daily life

Description

Experience-sampling data on what university students did between prompts when they were not studying. Prompts are nested within students, so the data suit a two-level latent class analysis: each prompt belongs to an activity profile, and students differ in how often their prompts fall in each profile.

Usage

`student_esm`

Format

A data frame with 2582 rows (100 students) and 15 columns:

student Integer student identifier, 1 to 100. The nesting unit; pass it as `id`.

day Integer study day, 0 to 13.

beep Integer prompt of the day, 1 to 5 among the prompts kept.

time_with_friends Factor, "no" or "yes": spent time with friends since the previous prompt.

on_social_media Factor, "no" or "yes": used social media.

tv_video_games Factor, "no" or "yes": watched TV or played video games.

listened_music Factor, "no" or "yes": listened to music.

sports Factor, "no" or "yes": did sports.

walking Factor, "no" or "yes": went for a walk.

reading Factor, "no" or "yes": read.

part_time_job Factor, "no" or "yes": worked in a part-time job.

happy Integer rating of feeling happy, 1 to 7.

relaxed Integer rating of feeling relaxed, 1 to 7.

worried Integer rating of feeling worried, 1 to 7.

exhausted Integer rating of feeling exhausted, 1 to 7.

Details

The students answered six prompts a day for fourteen days. At a prompt where the student had not studied since the previous prompt, the questionnaire asked which of eight leisure activities they had done. The study asked these items only at non-study prompts, so the dataset contains those prompts alone. Each row is one answered non-study prompt with all eight activity items and all four affect ratings observed.

The dataset is a subset of the original study: 100 of its students, drawn with a fixed seed from the students with at least 15 such prompts, with every such prompt they answered. Each student contributes 15 to 51 prompts (median 24). The eight activity items are categorical indicators; the four affect ratings are continuous-scored and can be combined with them in a mixed measurement model. worried has a floor: 49.7% of its ratings are 1, so a profile of low-worry prompts can have zero variance on it and reach the min_variance bound.

Source

openESM dataset 0062, deposited by Neubauer and Schmiedek at Zenodo, [doi:10.5281/zenodo.17347974](https://doi.org/10.5281/zenodo.17347974), under the Creative Commons Attribution 4.0 licence (CC-BY 4.0), and retrieved through the openESM database (<https://openesmdata.org>). The subset is produced by `data-raw/student-esm.R` in the source repository; identifiers are renumbered and the values are otherwise unchanged.

References

Neubauer, A. B., & Schmiedek, F. (2024). Approaching academic adjustment on multiple time scales. *Zeitschrift für Erziehungswissenschaft*, 27, 147–168. [doi:10.1007/s11618023011828](https://doi.org/10.1007/s11618023011828)

See Also

`multilpa()` with `categorical =` for the two-level latent class model.

Examples

```
head(student_esm)

activities <- c("time_with_friends", "on_social_media", "tv_video_games",
               "listened_music", "sports", "walking", "reading",
               "part_time_job")
first_week <- subset(student_esm, day <= 6)
lca <- multilpa(first_week, vars = activities, id = "student",
               n_profiles = 2, n_group_classes = 2,
               categorical = activities, n_starts = 3, seed = 1)
get_results(lca, what = "responses")
get_results(lca, what = "profile_probabilities")
```

study_hours

Study hours and quiz scores under two study strategies

Description

A simulated dataset for `mixture_regression()`, built so that every nesting of the mixture regression has a model that generated some of its columns, and the truth is kept for checking recovery.

Usage

```
study_hours
```

Format

A data frame with 900 rows (150 students by 6 weeks) and 10 columns:

student Integer student identifier, 1 to 150.

week Integer study week, 1 to 6.

hours Hours studied that week, 0 to 12.

sleep Hours slept the night before the quiz.

motivation Standardized motivation, constant within student.

score Quiz score.

passed Integer 0/1: passed the weekly check.

questions Count of questions the student asked that week.

strategy Factor, "deep" or "surface": the week's generating strategy. Not for fitting; kept to check recovery.

student_type Factor, "steady" or "erratic": the student's generating kind. Not for fitting; kept to check recovery.

Details

150 students each report six study weeks. In every week a student studies with one of two strategies, and the strategy decides how hours become a quiz score: a deep week gains about 4.5 points per hour from a base of 35, a surface week under one point per hour from a base of 55. Students are of two kinds that differ in their *mix* of strategies – steady students study deeply in about 85% of weeks, erratic ones in about 25% – and higher motivation makes a student more likely to be steady. More sleep makes a deep week more likely.

So `score ~ hours` is a two-class mixture at the level of the week (`class_level = "observation"`), whose class shares differ by student (the two-level model, `id = "student"`, `n_group_classes = 2`). `passed` depends on the student's kind, not the week's strategy, so it is a mixture at the level of the student (`class_level = "group"`). `questions` is a count whose Poisson regression depends on the week's strategy.

Source

Simulated by `data-raw/study-hours.R` in the source repository.

See Also

`mixture_regression()`, `enumerate_regressions()`.

Examples

```
fit <- mixture_regression(score ~ hours, data = study_hours, n_classes = 2,
                          n_starts = 3, seed = 1)
get_results(fit, "coefficients")
```

three_step

Relate a latent class to an outcome it did not help define

Description

The three-step approach: the measurement model is fitted first, units are assigned to classes, and only then is the outcome brought in, corrected for the fact that some assignments are wrong. Doing it in one step instead lets the outcome pull the classes toward itself; doing it naively by modal class instead attenuates every difference. This does neither.

Usage

```
three_step(
  x,
  data,
  outcome,
  level = c("individuals", "groups"),
  method = c("bch", "proportional", "modal"),
  ci_level = 0.95,
  contrast = c("none", "pairs"),
```

```

adjust = c("BH", "holm", "hochberg", "hommel", "bonferroni", "BY", "none"),
vcov_type = c("cluster", "independent")
)

```

Arguments

x	A covariate-free fitted <code>multilpa()</code> or <code>lta()</code> model. A fit that already uses membership covariates cannot be corrected by the fixed, unconditional classification-error matrix used here.
data	The data frame carrying the outcome, in the fit's row order. Shared fit columns are checked row by row; a frame with too little identifying information warns that alignment cannot be verified.
outcome	Name of a numeric outcome column not used as a measurement indicator in x. For <code>level = "groups"</code> it must be constant within each group.
level	"individuals" relates the outcome to profiles, "groups" to group classes.
method	"bch" applies the Bolck-Croon-Hagenaars weights; "proportional" weights by the posteriors, which is simpler and attenuated; "modal" is the naive assignment, shown for comparison.
ci_level	Confidence level for the intervals.
contrast	"none" (the default) reports one outcome mean per class. "pairs" reports the difference between every pair of classes instead, which is the quantity a three-step analysis is usually run to test, with a statistic and a p-value against a difference of zero.
adjust	Multiplicity correction applied across the pairwise differences, passed to <code>stats::p.adjust()</code> . "BH" by default; "none" leaves the p-values uncorrected. Ignored for <code>contrast = "none"</code> , which tests nothing.
vcov_type	"cluster", the default, sums the influence contributions within the fit's groups and takes those groups as the independent units, which is the honest choice when the outcome is measured on observations nested inside them. "independent" treats every observation as its own independent unit instead. That ignores the nesting the model was fitted to and is only defensible when there is no nesting left to ignore, such as a fit with a single group; it is never substituted silently, and the choice is recorded in the result's <code>vcov_type</code> attribute.

Details

The correction assumes the outcome is independent of the assigned class given the true one, which is what makes a three-step method valid. On a `lta()` fit, `level = "individuals"` uses one row per occasion and relates the outcome to the profile at that occasion. The default group-clustered variance accounts for repeated occasions within a group. This is a marginal profile-outcome analysis; it does not estimate an outcome effect on transitions or a sequence-level outcome model.

A cluster-robust variance is the sum of one outer product per independent unit, and those contributions sum to zero at the estimate, so it has rank at most one less than the number of units. With a single group it is exactly zero and with as many groups as classes it is singular. Both are refused with `latents_too_few_groups` rather than reported as a very small standard error; `vcov_type = "independent"` is the labelled way to ask for the unclustered variance instead.

The error matrix is treated as known rather than estimated, so the intervals are optimistic and the amount is worth stating. Over 60 replications of a two-profile design with 8% misclassification and a true class difference of 10, "bch" recovered that difference with a bias of -0.13 against -1.28 for "modal" and -1.82 for "proportional", and its nominal 95% intervals covered the truth 85% of the time. Treat the point estimate as close to unbiased and the interval as somewhat too narrow; the other two methods are biased enough that their intervals covered the truth in none of the 60.

Value

A base `data.frame`. For `contrast = "none"` it has one row per class and the columns `level`, `method`, `class`, `estimate`, `standard_error`, `conf_low`, `conf_high` and `effective_n`. It carries no statistic or `p_value`, unlike `r3step()`: an outcome mean has no meaningful null value to be tested against, and a p-value for "this class's mean is zero" would answer a question nobody asked. The comparison that does have a null is the difference between two classes, and that is what `contrast = "pairs"` returns: one row per pair with the columns `level`, `method`, `class`, `reference_class`, `estimate` (the mean of class minus the mean of `reference_class`), `standard_error`, `statistic`, `p_value`, `p_adjusted`, `conf_low` and `conf_high`, matching `r3step()`. A contrast with zero standard error has no defined test statistic or p-value; those columns are `NA_real_` rather than a spurious finite test.

Standard errors are cluster-robust in both shapes under the default `vcov_type = "cluster"`, taking the fit's groups as the independent units, so they remain honest when the outcome is measured on observations nested inside those groups; `vcov_type = "independent"` gives the unclustered variance instead. The variance that was used is recorded in the result's `vcov_type` attribute, and for `contrast = "pairs"` the correction applied to `p_adjusted` is recorded in its `adjust` attribute. A `contrast = "none"` table tests nothing, so it carries no `adjust` attribute.

References

Vermunt, J. K. (2010). Latent class modeling with covariates: two improved three-step approaches. *Political Analysis*, 18, 450–469. Bakk, Z., & Vermunt, J. K. (2016). Robustness of stepwise latent class modeling with continuous distal outcomes. *Structural Equation Modeling*, 23, 20–31.

Examples

```
set.seed(11)
group <- rep(seq_len(40), each = 10)
truth <- 1L + as.integer(runif(400) > 0.5)
example_data <- data.frame(
  g = group,
  a = rnorm(400, ifelse(truth == 2L, 1.1, -1.1)),
  b = rnorm(400, ifelse(truth == 2L, 1.1, -1.1)),
  y = rnorm(400, ifelse(truth == 2L, 10, 0))
)
fit <- multilpa(example_data, c("a", "b"), "g", n_profiles = 2,
  n_group_classes = 1, n_starts = 4, seed = 1)
three_step(fit, example_data, "y")
three_step(fit, example_data, "y", contrast = "pairs")
```

vcov.multilpa_covariates

Coefficients, covariance, intervals and likelihood of latents fits

Description

coef(), vcov(), confint(), logLik() and nobs() for multilevel profile and class models ([multilpa\(\)](#)), covariate models, latent transition models and, for vcov(), pooled imputations. Mixture regressions, growth mixtures and the additive, cross-level and general transition families document the same methods with their [get_results\(\)](#) page.

Usage

```
## S3 method for class 'multilpa_covariates'
vcov(
  object,
  data = NULL,
  step = 1e-04,
  vcov_type = c("observed", "robust", "opg"),
  scale = c("natural", "unconstrained"),
  boundary = c("error", "fix"),
  ...
)

## S3 method for class 'multilpa_covariates'
coef(object, scale = c("natural", "unconstrained"), ...)

## S3 method for class 'multilpa_covariates'
confint(object, parm, level = 0.95, data = NULL, ...)

## S3 method for class 'multilpa_covariates'
logLik(object, ...)

## S3 method for class 'multilpa_covariates'
nobs(object, ...)

## S3 method for class 'multilpa'
coef(object, scale = c("natural", "unconstrained"), ...)

## S3 method for class 'multilpa'
vcov(object, data = NULL, scale = c("natural", "unconstrained"), ...)

## S3 method for class 'multilpa'
confint(object, parm, level = 0.95, data = NULL, ...)

## S3 method for class 'multilpa'
logLik(object, ...)
```

```
## S3 method for class 'multilpa'
nobs(object, ...)

## S3 method for class 'latents_pooled'
vcov(object, ...)

## S3 method for class 'multilpa_transitions'
logLik(object, ...)

## S3 method for class 'multilpa_transitions'
nobs(object, ...)

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

Arguments

object	A fitted model: from <code>multilpa()</code> (class <code>multilpa</code>), a covariate model (<code>multilpa_covariates</code>), a latent transition model (<code>multilpa_transitions</code>), or pooled imputations (<code>latents_pooled</code> , <code>vcov()</code> only).
data	Optional. The data frame the model was fitted to. When omitted it is rebuilt from what the fit stores (indicators, identifiers, occasions and designs), which round-trips to floating-point precision; supplying it is the stronger check that the caller still holds that frame.
step	Finite-difference step for the observed information (covariate models).
vcov_type	"observed", "robust" or "opg", as for <code>parameter_inference()</code> (covariate models).
scale	"natural", the default, or "unconstrained". Natural coefficients are in the units <code>coef()</code> reports: means and variances in the indicators' units, probabilities as probabilities, with the covariance carried from the estimation scale by the delta method. Unconstrained is the scale the model is estimated on: log variances (diagonal), log-Cholesky coordinates (full covariance), log volumes, shapes and orientations (a structure that constrains them across profiles), and baseline-category logits for mixing and response probabilities. In a covariate model, means and membership coefficients are the same on both scales.
boundary	"error" or "fix", as for <code>parameter_inference()</code> : "fix" holds response probabilities on their bound, whose rows and columns are then zero (covariate models).
...	For <code>vcov()</code> and <code>confint()</code> of a <code>multilpa</code> fit, and <code>confint()</code> of a covariate model, passed to <code>parameter_inference()</code> (for example <code>method = "bootstrap"</code> , <code>vcov_type = "robust"</code> or <code>step</code>). Ignored otherwise.
parm	Optional coefficient names or indices; defaults to every estimated coefficient. Naming a coefficient that fixed held raises <code>latents_held_parameter</code> , because a held value has no interval.
level	Confidence level strictly between zero and one.

Value

coef(): a named numeric vector. vcov(): a named symmetric matrix over the same parameters. confint(): a matrix with one row per parameter and the lower and upper bounds as columns. logLik(): a "logLik" object with df and nobs attributes. nobs(): the number of groups, the independent units of the likelihood.

See Also

[parameter_inference\(\)](#) for the tidy table of estimates, standard errors and intervals.

Examples

```
fit <- multilpa(course_engagement, c("browse", "lectures", "forum_read"),
              "student", n_profiles = 2, n_group_classes = 2,
              n_starts = 2, seed = 1)

coef(fit)
logLik(fit)
nobs(fit)
```

Index

* datasets

- course_engagement, [7](#)
 - growth_schools, [35](#)
 - growth_scores, [36](#)
 - srl, [93](#)
 - student_esm, [95](#)
 - study_hours, [97](#)
-
- as.data.frame(), [4](#), [14](#), [16](#), [17](#), [19](#), [33](#), [52](#), [83](#), [94](#)
 - as.data.frame.latents_family_enumeration (get_results), [25](#)
 - as.data.frame.latents_growth_mixture (get_results), [25](#)
 - as.data.frame.latents_mixture_regression (get_results), [25](#)
 - as.data.frame.latents_pooled (get_results), [25](#)
 - as.data.frame.latents_regression_enumeration (enumerate_regressions), [18](#)
 - as.data.frame.latents_table (get_results), [25](#)
 - as.data.frame.latents_transition_enumeration (get_results), [25](#)
 - as.data.frame.mutilpa (get_results), [25](#)
 - as.data.frame.mutilpa_additive (get_results), [25](#)
 - as.data.frame.mutilpa_bootstrap_lrt (get_results), [25](#)
 - as.data.frame.mutilpa_covariates (get_results), [25](#)
 - as.data.frame.mutilpa_cross_level (get_results), [25](#)
 - as.data.frame.mutilpa_diagnostics (diagnostics), [11](#)
 - as.data.frame.mutilpa_enumeration (get_results), [25](#)
 - as.data.frame.mutilpa_lta (get_results), [25](#)
 - as.data.frame.mutilpa_start (get_results), [25](#)
 - as.data.frame.mutilpa_transitions (get_results), [25](#)
 - as.data.frame.summary_mutilpa (get_results), [25](#)
 - as.data.frame.summary_mutilpa_additive (get_results), [25](#)
 - as.data.frame.summary_mutilpa_bootstrap_lrt (get_results), [25](#)
 - as.data.frame.summary_mutilpa_covariates (get_results), [25](#)
 - as.data.frame.summary_mutilpa_enumeration (get_results), [25](#)
 - as.data.frame.summary_mutilpa_transitions (get_results), [25](#)
-
- bootstrap_lrt, [3](#)
 - bootstrap_lrt(), [17](#), [39](#), [40](#), [62–64](#)
 - candidate_fit (get_candidate), [22](#)
 - coef(), [52](#), [73](#)
 - coef.latents_growth_mixture (get_results), [25](#)
 - coef.latents_mixture_regression (get_results), [25](#)
 - coef.mutilpa (vcov.mutilpa_covariates), [101](#)
 - coef.mutilpa_additive (get_results), [25](#)
 - coef.mutilpa_covariates (vcov.mutilpa_covariates), [101](#)
 - coef.mutilpa_lta (get_results), [25](#)
 - coef.mutilpa_transitions (vcov.mutilpa_covariates), [101](#)
 - compare_models, [5](#)
 - compare_models(), [15](#), [22](#), [23](#), [29](#), [31](#), [37](#)
 - confint(), [40](#), [52](#)
 - confint.latents_growth_mixture (get_results), [25](#)

- confint.latents_mixture_regression
 (get_results), 25
- confint.mutilpa
 (vcov.mutilpa_covariates), 101
- confint.mutilpa_additive
 (get_results), 25
- confint.mutilpa_covariates
 (vcov.mutilpa_covariates), 101
- confint.mutilpa_transitions
 (parameter_inference.mutilpa_transitions), 5, 6, 15, 16, 45, 80
 71
- course_engagement, 7
- descriptives, 9
- descriptives(), 13, 41, 89
- diagnostics, 11
- diagnostics(), 11, 31, 33, 88, 89
- enumerate_classes, 13
- enumerate_classes(), 5, 6, 8, 15–17, 23, 42,
 61, 78, 91
- enumerate_lca, 15
- enumerate_lca(), 14, 17, 44, 78
- enumerate_lpa, 16
- enumerate_lpa(), 14, 16, 45, 78
- enumerate_regressions, 18
- enumerate_regressions(), 54, 98
- fit_staged, 20
- fit_staged(), 4, 28, 38, 40, 59, 62, 70, 90
- get_candidate, 22
- get_candidate(), 5, 6, 14–17, 29, 31
- get_group_tna, 24
- get_group_tna(), 35, 41
- get_results, 25
- get_results(), 12–14, 22, 36, 42, 49, 52, 53,
 62, 63, 89, 91, 95, 101
- get_results.latents_regression_enumeration
 (enumerate_regressions), 18
- get_tna, 34
- get_tna(), 25, 41
- growth_schools, 35
- growth_scores, 36
- latents-as-data-frame (get_results), 25
- latents-conditions, 37
- latents-model-methods
 (vcov.mutilpa_covariates), 101
- latents-plots
 (plot.latents_family_enumeration),
 73
- latents-predict
 (simulate.latents_growth_mixture),
 91
- latents-print (get_results), 25
- latents-summary (get_results), 25
- lca, 43
- lca(), 5, 6, 15, 16, 45, 80
- logLik(), 52
- logLik.latents_growth_mixture
 (get_results), 25
- logLik.latents_mixture_regression
 (get_results), 25
- logLik.mutilpa
 (vcov.mutilpa_covariates), 101
- logLik.mutilpa_additive (get_results),
 25
- logLik.mutilpa_covariates
 (vcov.mutilpa_covariates), 101
- logLik.mutilpa_cross_level
 (get_results), 25
- logLik.mutilpa_lta (get_results), 25
- logLik.mutilpa_transitions
 (vcov.mutilpa_covariates), 101
- lpa, 44
- lpa(), 5, 6, 17, 44, 80
- lta, 45
- lta(), 5, 6, 9, 14, 24, 30, 34, 41, 64, 69, 71,
 72, 78, 80, 86, 87, 99
- mixture_regression, 50
- mixture_regression(), 5, 18, 19, 30, 36, 37,
 41, 91, 97, 98
- mutilca, 55
- mutilca(), 16, 24, 34, 43, 44, 65, 80, 91
- mutilpa, 56
- mutilpa(), 3, 5, 6, 9, 11, 14, 17, 22–24, 34,
 42–45, 48, 49, 55, 56, 68, 80, 82,
 85–87, 91, 94, 96, 99, 101, 102
- nobs(), 52
- nobs.latents_growth_mixture
 (get_results), 25
- nobs.latents_mixture_regression
 (get_results), 25
- nobs.mutilpa
 (vcov.mutilpa_covariates), 101

- nobs.multilpa_additive (get_results), 25
- nobs.multilpa_covariates
 - (vcov.multilpa_covariates), 101
- nobs.multilpa_cross_level
 - (get_results), 25
- nobs.multilpa_lta (get_results), 25
- nobs.multilpa_transitions
 - (vcov.multilpa_covariates), 101
- parameter_inference
 - (parameter_inference.multilpa_additive), 65
- parameter_inference(), 22, 28, 31, 38, 40, 60–64, 69, 73, 82, 83, 91, 102, 103
- parameter_inference.multilpa_additive, 65
- parameter_inference.multilpa_transitions, 71
- plot(), 4, 15–17, 52, 53, 88
- plot.latents_comparison
 - (plot.latents_family_enumeration), 73
- plot.latents_family_enumeration, 73
- plot.latents_growth_mixture
 - (plot.latents_family_enumeration), 73
- plot.latents_mixture_regression
 - (plot.latents_family_enumeration), 73
- plot.latents_pooled
 - (plot.latents_family_enumeration), 73
- plot.latents_transition_enumeration
 - (plot.latents_family_enumeration), 73
- plot.multilpa
 - (plot.latents_family_enumeration), 73
- plot.multilpa_additive
 - (plot.latents_family_enumeration), 73
- plot.multilpa_bootstrap_lrt
 - (plot.latents_family_enumeration), 73
- plot.multilpa_covariates
 - (plot.latents_family_enumeration), 73
- plot.multilpa_cross_level
 - (plot.latents_family_enumeration), 73
- plot.multilpa_diagnostics
 - (diagnostics), 11
- plot.multilpa_enumeration
 - (plot.latents_family_enumeration), 73
- plot.multilpa_lta
 - (plot.latents_family_enumeration), 73
- plot.multilpa_transitions
 - (plot.latents_family_enumeration), 73
- plot_diagnostics
 - (plot.latents_family_enumeration), 73
- plot_enumeration
 - (plot.latents_family_enumeration), 73
- plot_prevalence
 - (plot.latents_family_enumeration), 73
- plot_profiles
 - (plot.latents_family_enumeration), 73
- plot_responses
 - (plot.latents_family_enumeration), 73
- plot_transitions
 - (plot.latents_family_enumeration), 73
- plot_tree
 - (plot.latents_family_enumeration), 73
- plot_views
 - (plot.latents_family_enumeration), 73
- plot_views(), 89
- pool_imputations, 81
- pool_imputations(), 30, 40
- predict(), 52, 64
- predict.latents_growth_mixture
 - (simulate.latents_growth_mixture), 91
- predict.latents_mixture_regression
 - (simulate.latents_growth_mixture), 91
- predict.multilpa
 - (simulate.latents_growth_mixture), 91

[91](#)
 print.latents_family_enumeration
 (get_results), [25](#)
 print.latents_growth_mixture
 (get_results), [25](#)
 print.latents_mixture_regression
 (get_results), [25](#)
 print.latents_plots(get_results), [25](#)
 print.latents_pooled(get_results), [25](#)
 print.latents_prior(prior_control), [84](#)
 print.latents_regression_enumeration
 (enumerate_regressions), [18](#)
 print.latents_table(get_results), [25](#)
 print.latents_transition_enumeration
 (get_results), [25](#)
 print.mutilpa(get_results), [25](#)
 print.mutilpa_additive(get_results),
 [25](#)
 print.mutilpa_bootstrap_lrt
 (get_results), [25](#)
 print.mutilpa_covariates
 (get_results), [25](#)
 print.mutilpa_cross_level
 (get_results), [25](#)
 print.mutilpa_diagnostics
 (diagnostics), [11](#)
 print.mutilpa_enumeration
 (get_results), [25](#)
 print.mutilpa_lta(get_results), [25](#)
 print.mutilpa_start(get_results), [25](#)
 print.mutilpa_transitions
 (get_results), [25](#)
 print.summary_latents_growth_mixture
 (get_results), [25](#)
 print.summary_latents_mixture_regression
 (get_results), [25](#)
 print.summary_mutilpa(get_results), [25](#)
 print.summary_mutilpa_additive
 (get_results), [25](#)
 print.summary_mutilpa_bootstrap_lrt
 (get_results), [25](#)
 print.summary_mutilpa_covariates
 (get_results), [25](#)
 print.summary_mutilpa_enumeration
 (get_results), [25](#)
 print.summary_mutilpa_transitions
 (get_results), [25](#)
 prior_control, [84](#)
 prior_control(), [61](#)
 r3step, [85](#)
 r3step(), [7](#), [28](#), [38–40](#), [58](#), [62](#), [63](#), [100](#)
 report, [88](#)
 sensitivity, [89](#)
 sensitivity(), [40](#), [42](#)
 simulate.latents_growth_mixture, [91](#)
 simulate.latents_mixture_regression
 (simulate.latents_growth_mixture),
 [91](#)
 srl, [93](#)
 starting_values, [94](#)
 starting_values(), [22](#), [40](#), [62](#)
 stats::p.adjust(), [68](#), [72](#), [86](#), [99](#)
 student_esm, [95](#)
 study_hours, [97](#)
 summary(), [4](#), [13](#), [15–17](#), [33](#), [52](#), [88](#), [89](#)
 summary.latents_family_enumeration
 (get_results), [25](#)
 summary.latents_growth_mixture
 (get_results), [25](#)
 summary.latents_mixture_regression
 (get_results), [25](#)
 summary.latents_pooled(get_results), [25](#)
 summary.latents_transition_enumeration
 (get_results), [25](#)
 summary.mutilpa(get_results), [25](#)
 summary.mutilpa_additive
 (get_results), [25](#)
 summary.mutilpa_bootstrap_lrt
 (get_results), [25](#)
 summary.mutilpa_covariates
 (get_results), [25](#)
 summary.mutilpa_cross_level
 (get_results), [25](#)
 summary.mutilpa_enumeration
 (get_results), [25](#)
 summary.mutilpa_lta(get_results), [25](#)
 summary.mutilpa_transitions
 (get_results), [25](#)
 three_step, [98](#)
 three_step(), [28](#), [38–40](#), [58](#), [62](#), [63](#), [86](#), [87](#)
 tna::tna(), [24](#), [34](#), [35](#)
 vcov(), [40](#), [52](#)
 vcov.latents_growth_mixture
 (get_results), [25](#)

`vcov.latents_mixture_regression`
 (`get_results`), [25](#)
`vcov.latents_pooled`
 (`vcov.multilpa_covariates`), [101](#)
`vcov.multilpa`
 (`vcov.multilpa_covariates`), [101](#)
`vcov.multilpa_additive` (`get_results`), [25](#)
`vcov.multilpa_covariates`, [101](#)
`vcov.multilpa_lta` (`get_results`), [25](#)
`vcov.multilpa_transitions`
 (`parameter_inference.multilpa_transitions`),
 [71](#)