Package 'RobustLPA'

August 21, 2026

Title Robust Latent Profile Analysis

Version 1.0.0

Description Provides a comprehensive toolset for estimating Latent Profile Analysis (LPA) models that are robust to multivariate outliers and missing data. By integrating a high-performance 'C++' engine via 'RcppArmadillo', it reliably extracts latent profiles using both Expectation-Maximization (EM) and Markov Chain Monte Carlo (MCMC) Bayesian estimation. The EM engine implements a Full Information Maximum Likelihood (FIML) approach, Huber weighting, and LASSO regularization with k-fold cross-validation for optimal penalty tuning. The MCMC engine utilizes a Bayesian Lasso approach with Laplace priors, the same Huber down-weighting available in the EM engine, multiple chains (4 by default), and classic Gelman-Rubin/effective sample size convergence diagnostics. It supports multiple geometric variance-covariance models, along with functions for bootstrapped likelihood ratio tests (BLRT), BCH auxiliary variable analysis, and plotting.
For methodological details on the Bootstrapped Likelihood Ratio Test, see Nylund et al. (2007) <doi:10.1080/10705510701575396>. For robust clustering methods, see Garcia-Escudero et al. (2010) <doi:10.1007/s11634-010-0064-5>.
For BCH auxiliary variable analysis, see Bolck et al. (2004) <doi:10.1093/pan/mph001>.

License GPL (>= 3)

Encoding UTF-8

RoxygenNote 8.0.0

LinkingTo Rcpp, RcppArmadillo

Imports Rcpp, ggplot2, stats, bayesplot, coda

Suggests parallel, knitr, rmarkdown

VignetteBuilder knitr

Depends R (>= 3.5)

LazyData true

NeedsCompilation yes

Author Valerio Riccardo Aquila [aut, cre] (ORCID:
<https://orcid.org/0009-0004-2231-2141>)

Maintainer Valerio Riccardo Aquila <valerio_aquila@hotmail.it>

Repository CRAN

Date/Publication 2026-08-20 23:02:26 UTC

Contents

bch_robust 2
blrt_robust 5
estimate_profiles_robust 6
neuro_data 8
plot_mcmc_chains 9
plot_robust_lpa 10
print.robust_lpa 11
robust_lpa 12
robust_mean 16
summary.robust_lpa 17

Index **19**

`bch_robust` *BCH Method for Auxiliary Continuous Variables*

Description

Applies the 3-step Bolck-Croon-Hagenaars (BCH) method to test the relationship between robust latent profiles and a continuous auxiliary (distal outcome) variable, adjusting for classification error in the profile assignments. Optionally adds a bootstrap correction (`correction = "bootstrap"`) for the F-test's main weakness: treating the classification matrix **D** as known/fixed (see Details).

Usage

```
bch_robust(
  model,
  aux_var,
  correction = c("none", "bootstrap"),
  n_boot = 200,
  cores = 1
)
```

Arguments

`model`	A fitted robust LPA model object returned by `robust_lpa`.
`aux_var`	A numeric vector of the continuous auxiliary (distal outcome) variable, of length `nrow(model$probabilities)`. `NA` values are dropped listwise before the BCH calculations.
`correction`	String, either `"none"` (default; the fixed-D F-test only) or `"bootstrap"` (also compute the bootstrap classification-uncertainty correction described in Details). Ignored (with `$Bootstrap_Correction = NULL`) when `"none"`.

`n_boot` Integer, number of bootstrap resamples to use when `correction = "bootstrap"` (default `200`; at least `10`). Each resample refits the full `robust_lpa` model, so runtime scales linearly with `n_boot`. Ignored when `correction = "none"`.

`cores` Integer, number of CPU cores to use to run the `n_boot` bootstrap refits in parallel when `correction = "bootstrap"` (default `1`, sequential); uses the same backend as `robust_lpa`'s own `cores` argument. Ignored when `correction = "none"`.

Details

Let $\hat{p}_{ig}$ be the posterior probability that observation i belongs to profile g (`model$probabilities`), and let $\hat{C}_i$ be its modal (hard) assignment (`model$assignments`). The classification probability matrix **D** is estimated as

$$D_{g,c} = P(\hat{C} = c \mid C = g) \approx \frac{1}{N_c} \sum_{i:\, \hat{C}_i = c} \hat{p}_{ig}$$

(Bolck, Croon, & Hagenaars, 2004). The BCH weight matrix is $W = D^{-1}$. The classification-error-corrected mean of the auxiliary variable Y for profile g is

$$\hat{\mu}_g^{BCH} = \frac{\sum_{i=1}^{n} W_{g,\hat{C}_i} Y_i}{\sum_{i=1}^{n} W_{g,\hat{C}_i}}$$

(Vermunt, 2010, eq. 13-15; Bolck et al., 2004), summed over *every* observation: each contributes to every profile's mean with a (possibly negative) cross-class weight $W_{g,\hat{C}_i}$, which is what removes the attenuation bias of a naive "reweight only your own class" analysis.

The main (`$ANOVA_Table`) significance test is a one-way weighted ANOVA on the equivalent "long" data set (one row per observation per profile, weighted by $W_{g,\hat{C}_i}$), fit by direct weighted normal equations rather than `stats::lm()`/`stats::aov()`, because the BCH weights are frequently negative and base R's weighted-least-squares machinery cannot handle that.

The fixed-D caveat, and the bootstrap correction. `$ANOVA_Table`'s F-test treats **D** as known/fixed. Bolck et al. (2004) and Vermunt (2010) both note that this understates the true uncertainty, because **D** is itself estimated from the step-1 model; Vermunt (2010) reports that naive (uncorrected) BCH p-values can be "much too small," particularly with poorly separated profiles or small samples. The literature's analytic fix is a "sandwich" (pseudo-likelihood) variance correction (Bakk, Oberski, & Vermunt, 2014), which requires the Fisher information of the step-1 mixture log-likelihood – intractable to derive analytically here for the Huber-robust EM and Bayesian-Lasso MCMC engines. Setting `correction = "bootstrap"` instead approximates that correction *nonparametrically*: it resamples observations with replacement, refits the entire step-1 `robust_lpa` model (using the exact same specification as `model`, via its stored `$call_args`) and recomputes **D**/**W**/the profile means on each resample, so the resulting bootstrap variability genuinely reflects step-1 estimation uncertainty (unlike the fixed-D F-test). This yields bootstrap standard errors and percentile confidence intervals for `Profile_Means`, plus a Wald chi-square test of "all profile means equal" using the bootstrap covariance – reported in `$Bootstrap_Correction`, and preferable to `$ANOVA_Table`'s p-value for publication-grade inference. It is *not* the Bakk et al. (2014) analytic formula; treat it as a practical approximation with the same goal (each refit's arbitrary profile labels are first aligned to `model`'s via a nearest-mean matching, to avoid mixing different real-world profiles together across resamples – see the package source for details). It is off (`"none"`) by default because it requires `n_boot` additional full model refits and is therefore substantially slower; use `cores > 1` to parallelize it.

Value

A list containing:

Profile_Means Named numeric vector of BCH bias-corrected profile means of `aux_var`.

ANOVA_Table A data.frame with `Df`, `Sum_Sq`, `Mean_Sq`, `F_value`, and `p_value` for the "Class" and "Residuals" rows (see the fixed-D caveat in Details).

Classification_Matrix The `G` x `G` matrix **D** of classification probabilities.

Classification_Weights The `G` x `G` BCH weight matrix $W = D^{-1}$.

N_Used Integer, the number of observations retained after removing missing `aux_var` values.

Bootstrap_Correction `NULL` unless `correction = "bootstrap"`, in which case a list with `n_boot_used`, `n_boot_failed`, `SE` and `CI_lower`/`CI_upper` (per profile), and the overall `Wald_stat`/`Wald_df`/`Wald_p_value` test of equal profile means (see Details).

References

Bolck, A., Croon, M., & Hagenaars, J. (2004). Estimating latent structure models with categorical variables: One-step versus three-step estimators. *Political Analysis*, 12(1), 3-27. doi:10.1093/pan/mph001

Vermunt, J. K. (2010). Latent class modeling with covariates: Two improved three-step approaches. *Political Analysis*, 18(4), 450-469. doi:10.1093/pan/mpq025

Bakk, Z., Oberski, D. L., & Vermunt, J. K. (2014). Relating latent class assignments to external variables: Standard errors for correct inference. *Political Analysis*, 22(4), 520-540. doi:10.1093/pan/mpu003

Examples

```
data(neuro_data)
# Fit the model on Memory and RT_Stroop only
x <- scale(as.matrix(neuro_data[, c("Memory", "RT_Stroop")]))
fit <- suppressWarnings(robust_lpa(data = x, G = 2, model = 1, n_starts = 3))
summary(fit)  # profile means for the two fitted variables
# Test RT_TMT (not used to fit the model) as an auxiliary outcome
bch_res <- bch_robust(fit, neuro_data$RT_TMT)
bch_res$Profile_Means
bch_res$ANOVA_Table


# Add the bootstrap classification-uncertainty correction (slower: refits
# the model n_boot times). A small n_boot here is just for a fast demo --
# use several hundred for publication-grade inference.
bch_res_boot <- suppressWarnings(bch_robust(fit, neuro_data$RT_TMT,
                                            correction = "bootstrap", n_boot = 30))
bch_res_boot$Bootstrap_Correction
```

blrt_robust *Bootstrapped Likelihood Ratio Test for Robust LPA*

Description

Compares a robust LPA model with `G` profiles against a null model with `G - 1` profiles using parametric bootstrapping (Nylund et al., 2007): the null model is fit to the observed data, data are simulated from it, and both the null and alternative models are refit to each simulated dataset to build a reference distribution for the likelihood ratio test statistic under `H0`. Supports FIML simulation conditions (the missingness pattern of the observed data is replicated in every simulated dataset).

Usage

```
blrt_robust(
  data,
  G,
  model = 6,
  engine = "EM",
  n_samples = 50,
  n_starts = 2,
  cores = 1,
  ...
)
```

Arguments

`data`	A matrix or data.frame.
`G`	The number of profiles for the alternative hypothesis (compared against `G - 1`).
`model`	An integer (1 to 6) specifying the variance-covariance parameterization (see `robust_lpa`).
`engine`	String, either `"EM"` (default) or `"MCMC"`; passed through to every internal call to `robust_lpa` so that the observed and bootstrap-refit models use the same estimation engine.
`n_samples`	Number of bootstrap samples. Default is 50 for speed; 200+ is recommended for publications.
`n_starts`	Number of starts for the EM algorithm execution (ignored when `engine = "MCMC"`).
`cores`	Integer, number of CPU cores to use to run the `n_samples` bootstrap replicates in parallel (default `1`, sequential). Each replicate fits two independent `robust_lpa` models (null and alternative) on its own simulated dataset, so replicates are "embarrassingly parallel". Uses the same `parallel::mclapply()` / `parallel::makeCluster()` PSOCK backend as `robust_lpa`'s own `cores` argument (see there for details); falls back to sequential execution with a `warning()` if the **parallel** package is unavailable. If you also pass `cores` through `...` to `robust_lpa` (to parallelize each replicate's EM restarts / MCMC chains too),

keep the product of the two `cores` values at or below your machine's core count to avoid oversubscription; for most uses it is simplest to parallelize only at this (bootstrap-replicate) level.

`...` Additional arguments passed on to `robust_lpa` (e.g. `max_iter`, `tol`, `mcmc_iter`, `n_chains`, `prior_laplace`, `robust`, `alpha`) – for instance, pass `robust = FALSE` here to run the BLRT with classical (non-robust) estimation throughout, for either engine. Note that with `engine = "MCMC"` every one of the `2 * (n_samples + 1)` internal `robust_lpa` calls this function makes will run `n_chains` chains each; consider passing a smaller `n_chains` and/or `mcmc_iter` than the `robust_lpa` defaults, and/or using `cores > 1` above, to keep the BLRT's bootstrap loop tractable.

Value

A list containing:

LRT_Observed The observed likelihood ratio test statistic (non-negative).

Bootstrap_LRTs Numeric vector of the successfully-fit bootstrap replicates.

p_value The empirical p-value, computed with the standard "+1" small-sample correction (`(sum(Bootstrap_LRTs >= LRT_Observed) + 1) / (length(Bootstrap_LRTs) + 1)`), which avoids reporting an (impossible) exact p-value of 0 from a finite bootstrap.

Bootstrap_Failures Integer, how many of the `n_samples` bootstrap replicates failed to converge and were excluded from `Bootstrap_LRTs` / `p_value`.

References

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(4), 535-569. doi:10.1080/10705510701575396

Examples

```
# Fast demonstration of the robust BLRT: is a 2nd profile justified over 1?
data(neuro_data)
x <- scale(as.matrix(neuro_data[, c("Memory", "RT_Stroop")]))
blrt_res <- suppressWarnings(blrt_robust(x, G = 2, model = 1, n_samples = 2, n_starts = 3))
# Print the summary of the results
blrt_res
```

`estimate_profiles_robust`

Estimate Robust Latent Profile Models Across Profiles and Models

---

Description

Estimate Robust Latent Profile Models Across Profiles and Models

Usage

```
estimate_profiles_robust(
  data,
  n_profiles = 1:3,
  models = c(1, 2, 3, 4, 5, 6),
  engine = "EM",
  cores = 1,
  n_starts = 5,
  lambda = 0,
  tune_lasso = FALSE,
  k_folds = 5,
  lambda_grid = c(0.01, 0.05, 0.1, 0.2),
  ...
)
```

Arguments

`data`	A matrix or data.frame.
`n_profiles`	A vector of integers specifying the number of profiles to run.
`models`	A vector of LPA models to run.
`engine`	String. Either "EM" or "MCMC".
`cores`	Integer. Number of CPU cores to use for parallel processing *across* the requested `n_profiles` x `models` grid (and across `k_folds` within `tune_lasso`). This is a different axis of parallelism from `robust_lpa`'s own `cores` argument (which parallelizes across EM restarts / MCMC chains *within* a single model fit); if you also pass `cores` through `...` to `robust_lpa`, keep the product of the two roughly at or below your machine's core count to avoid oversubscription.
`n_starts`	Number of initializations per model.
`lambda`	Fixed penalty for LASSO.
`tune_lasso`	Logical. If TRUE, finds optimal lambda via cross-validation.
`k_folds`	Number of folds for cross-validation.
`lambda_grid`	Vector of penalty values to test.
`...`	Additional arguments passed on to every internal `robust_lpa` call (e.g. `max_iter`, `tol`, `robust`, `alpha`, `cores`, and, when `engine = "MCMC"`, `mcmc_iter`/`n_chains`/`prior_laplace`) – for instance, pass `robust = FALSE` here to compare model/profile combinations using classical (non-robust) estimation throughout, for either engine.

Value

A list containing the fit comparison table and the estimated models.

Examples

```
# Quick evaluation of multiple profiles
data(neuro_data)
x <- scale(as.matrix(neuro_data[, c("Memory", "RT_Stroop")]))
```

```
res <- suppressWarnings(estimate_profiles_robust(x, n_profiles = 1:2, models = 1, n_starts = 3))
res$fit_table
# Each element of `res$models` is a `robust_lpa` object with print/summary methods
summary(res$models[[1]])
```

`neuro_data` *Simulated Neuropsychological Dataset for Robust LPA*

Description

A synthetic dataset of neuropsychological test scores and reaction times for two latent groups, "Healthy" and "Pathological", designed as a worked example for every estimation path in this package: the Expectation-Maximization and MCMC engines, all six variance-covariance parameterizations, robust vs. classical estimation, LASSO regularization, model/profile selection (`estimate_profiles_robust`), the bootstrapped likelihood ratio test (`blrt_robust`), and the BCH auxiliary-variable method (`bch_robust`).

Usage

```
neuro_data
```

Format

A data frame with 250 rows and 7 variables:

ID Unique identifier for each participant.

True_Profile The true latent group, `"Healthy"` (n = 150) or `"Pathological"` (n = 100). Not used for estimation (LPA is unsupervised); included so that recovered profiles can be checked against ground truth, e.g. `table(neuro_data$True_Profile, fit$assignments)`.

Memory Simulated memory test score. Differs in mean between groups.

Attention Simulated attention test score. Identical distribution in both groups (no group signal); a noise variable.

Executive_Functions Simulated executive functions score. Identical distribution in both groups (no group signal); a noise variable.

RT_Stroop Reaction time in milliseconds. Differs in mean, variance, and correlation with `RT_TMT` between groups; a subset of Pathological observations carry an additional outlying shift.

RT_TMT Reaction time in milliseconds. Differs in mean, variance, and correlation with `RT_Stroop` between groups; a subset of Pathological observations carry an additional outlying shift.

Details

The two groups differ in more than location: `Attention` and `Executive_Functions` have *identical* means and variances in both groups (deliberate noise variables, carrying no group signal), while `Memory`, `RT_Stroop`, and `RT_TMT` differ in mean between groups, and `RT_Stroop`/`RT_TMT` additionally differ in variance and in their correlation with each other (0.35 in Healthy vs. 0.90 in Pathological). This last feature is intentional: it is a genuine, whole-group difference in covariance structure (not merely in means), so that `model = 6` (a fully unconstrained covariance matrix per profile) is the

best-fitting parameterization for this dataset by BIC at G = 2 – run estimate_profiles_robust(scale(neuro_data[, 3:7]),models = 1:6, n_profiles = 1:3) and inspect $fit_table to see this directly. A more parsimonious model (e.g. model = 3, a single covariance matrix shared across profiles) fits these data measurably worse, illustrating why the six parameterizations exist and how to choose among them.

On top of this, 6% of the Pathological observations (chosen at random) receive an additional, positive, randomly-sized shift on RT_Stroop and RT_TMT (drawn from a Gamma distribution, so the contamination varies in severity rather than landing on a single fixed value) – measurement-error-like outliers on top of the two groups' otherwise multivariate-normal structure. These are what robust = TRUE (the default of robust_lpa) down-weights via Huber-type estimation; compare robust = TRUE vs. robust = FALSE fits to see their effect on the estimated Pathological-profile covariance.

The contamination magnitude and rate were calibrated (by direct grid search across the six variance-covariance models, replicated over multiple random seeds) so that fitting model = 6 with G = 2 reliably wins by BIC over both more-parsimonious models at G = 2 and less-parsimonious models at G = 3, and recovers True_Profile with better than 95% accuracy in the reference implementation. See data-raw/generate_neuro_data.R (in the package sources, not installed) for the full generative code, the exact parameter values, and the fixed random seed used to build this exact copy of the dataset.

Source

Simulated data for testing and documentation purposes; see data-raw/generate_neuro_data.R in the package sources for the full, reproducible generative code.

---

plot_mcmc_chains *Plot MCMC Trace for Robust LPA Models*

Description

Draws multi-chain trace plots for the MCMC engine of robust_lpa using **bayesplot**. By default, the profile means, profile variances (the diagonal of each covariance matrix), and mixing proportions are shown; use pars to select a subset.

Usage

```
plot_mcmc_chains(model, pars = NULL)
```

Arguments

model	A fitted model object returned by robust_lpa with engine = "MCMC".
pars	Optional character vector of parameter names to visualize (a subset of the default "mu[...]" / "sigma[...]" / "pi[...]" names described in Details). Default NULL plots all of them.

Details

robust_mcmc_cpp (called internally by robust_lpa(engine = "MCMC"), once per chain) returns its draws as a nested list (mu_chain, sigma_chain, pi_chain), not the array/matrix format bayesplot::mcmc_trace() expects. This function reshapes the raw per-chain draws stored in model$mcmc_draws$chains into an [iterations, chains, parameters] array before calling bayesplot::mcmc_trace(), so every chain is shown overlaid as a separate colored trace – the standard visual convergence check (well-mixed, overlapping chains suggest convergence; chains that stay visually separated suggest they have not converged, consistent with a high Gelman-Rubin $\hat{R}$; see model$mcmc_diagnostics).

Parameter names follow the pattern "mu[g,j]" (mean of variable j in profile g), "sigma[g,j]" (variance of variable j in profile g), and "pi[g]" (mixing proportion of profile g). Off-diagonal covariance terms are not included by default to keep the default plot readable; inspect model$mcmc_draws$chains[[1]]$sigma directly if you need those.

Value

A ggplot object generated by bayesplot::mcmc_trace().

Examples

```
data(neuro_data)
x <- scale(as.matrix(neuro_data[, c("Memory", "RT_Stroop")]))
fit <- suppressWarnings(robust_lpa(x, G = 2, model = 2, engine = "MCMC",
                                   mcmc_iter = 200, n_chains = 2))
summary(fit)  # profile means and MCMC convergence diagnostics
plot_mcmc_chains(fit)
```

---

plot_robust_lpa *Plot Robust Latent Profiles*

Description

Automatically generates a professional profile plot using ggplot2 from an estimated robust LPA model.

Usage

```
plot_robust_lpa(
  model,
  which_model = NULL,
  title = "Robust Latent Profiles",
  xlab = "Variables",
  ylab = "Value",
  var_labels = NULL,
  legend_title = "Class"
)
```

Arguments

model	Either a single fitted model object returned by robust_lpa, or the list returned by estimate_profiles_robust. In the latter case, the model with the lowest BIC in model$fit_table is selected automatically (a message reports which one), unless which_model is given.
which_model	Optional string, the name of a specific model to plot when model is an estimate_profiles_robust result (one of names(model$models), e.g. "model_6_profiles_3"). Ignored when model is already a single fitted model.
title	The title of the plot. Default is "Robust Latent Profiles".
xlab	The x-axis label. Default is "Variables".
ylab	The y-axis label. Default is "Value".
var_labels	A character vector to manually rename the variables on the X axis. Default is NULL (auto-detect).
legend_title	The title of the legend. Default is "Class".

Value

A ggplot object.

Examples

```
data(neuro_data)
x <- scale(as.matrix(neuro_data[, c("Memory", "RT_Stroop")]))
fit <- suppressWarnings(robust_lpa(data = x, G = 2, model = 1, n_starts = 3))
print(fit)  # concise overview (print.robust_lpa())
plot_robust_lpa(fit)
```

---

print.robust_lpa *Print a Fitted Robust LPA Model*

Description

A short, four-line-or-fewer overview of a model fitted by robust_lpa: engine/model/profiles/N, the headline fit indices (log-likelihood, AIC, BIC, entropy), the mixing proportions, and, for the MCMC engine, the chain configuration. It deliberately omits profile means and full diagnostics – use summary.robust_lpa for those.

Usage

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

Arguments

x	A robust_lpa object, as returned by robust_lpa.
...	Currently ignored (present for S3 consistency with the generic print).

Value

x, invisibly.

See Also

robust_lpa, summary.robust_lpa

Examples

```
data(neuro_data)
x <- scale(as.matrix(neuro_data[, c("Memory", "RT_Stroop")]))
fit <- suppressWarnings(robust_lpa(x, G = 2, model = 2, n_starts = 3, max_iter = 20))
print(fit)
```

---

robust_lpa *Fit a Single Robust Latent Profile Analysis Model*

Description

Estimates a Latent Profile Analysis (Gaussian mixture) model that is robust to multivariate outliers (via Huber down-weighting) and to missing data (via a Full Information Maximum Likelihood, FIML, available-case treatment), using either an EM or an MCMC (Bayesian Lasso) engine. Both engines share the same robust/alpha Huber down-weighting mechanism, on by default (see the "Robust estimation" section below). The MCMC engine runs n_chains independent chains (4 by default) and reports classic multi-chain convergence diagnostics (Gelman-Rubin $\hat{R}$ and effective sample size) in $mcmc_diagnostics. Set cores > 1 to run the EM engine's random restarts, or the MCMC engine's chains, in parallel.

Usage

```
robust_lpa(
  data,
  G,
  model = 6,
  engine = "EM",
  max_iter = 100,
  tol = 1e-06,
  n_starts = 5,
  lambda = 0,
  mcmc_iter = 2000,
  prior_laplace = 0.1,
  robust = TRUE,
  alpha = 0.05,
  n_chains = 4,
  cores = 1
)
```

Arguments

`data`	A matrix or data.frame of observations (numeric columns only; `NA` is allowed and triggers the FIML code path).
`G`	The number of latent profiles to extract (a single positive integer).
`model`	An integer (1 to 6) specifying the variance-covariance parameterization, following the same numbering convention as tidyLPA / mclust: **1** Equal variances across profiles, covariances fixed to 0 (diagonal, shared across profiles – each variable keeps its own variance level, constrained to be the same in every profile; this is *not* a single isotropic/spherical variance shared across variables too, despite that being a common shorthand for this model elsewhere). **2** Varying variances across profiles, covariances fixed to 0 (diagonal, profile-specific). **3** Equal variances and equal covariances across profiles (one shared full covariance matrix). **4** Varying variances, equal covariance *structure* (shared correlation matrix, profile-specific variances). **5** Equal variances, varying covariances (shared variances, profile-specific correlation matrices). **6** Fully unconstrained: each profile has its own variances and covariances (default).
`engine`	String. Either `"EM"` (default) or `"MCMC"`.
`max_iter`	Maximum number of EM iterations. Ignored when `engine = "MCMC"`.
`tol`	Tolerance for EM convergence (on the observed-data log-likelihood). Ignored when `engine = "MCMC"`.
`n_starts`	Number of random EM initializations; the fit with the highest log-likelihood across starts is returned. Ignored when `engine = "MCMC"`.
`lambda`	Non-negative soft-thresholding (LASSO-type) penalty applied to the profile means, via direct per-coordinate soft-thresholding of the (Huber- and posterior-probability-weighted) mean at every M-step – see `robust_m_step`/`robust_update_cpp()`. This shrinks each mean component toward zero on the scale of the (as supplied) `data`; it is only statistically meaningful as a sparsity-inducing penalty on centered/scaled data (so that zero corresponds to "no departure from the grand mean"), and, because the threshold is a flat amount rather than one rescaled by each profile's variance/sample size, it is a computationally convenient approximation to (not an exact coordinate-wise solution of) the corresponding L1-penalized weighted log-likelihood except when profile covariances are close to the identity – as they will be on standardized data with roughly independent variables. Standardize `data` first if you intend to use `lambda > 0`; a `warning()` is raised if `data` does not look centered/scaled. Default `0` (no shrinkage).
`mcmc_iter`	Number of iterations per MCMC chain (see `n_chains`). The first half of each chain is discarded as burn-in before computing posterior summaries and convergence diagnostics. Ignored when `engine = "EM"`.

`prior_laplace` Positive numeric, the Laplace (Bayesian Lasso) shrinkage/rate hyperparameter for the profile means under the MCMC engine (denoted λ in Park & Casella, 2008; *not* a dispersion "scale" in the usual sense – **larger** values induce **more** shrinkage of the means toward zero, analogous to a bigger `lambda` in the EM engine). Ignored when `engine = "EM"`.

`robust` Logical. If `TRUE` (default), robust (outlier down-weighted) estimation is used regardless of `engine`: Huber weights, computed from per-observation squared Mahalanobis distances, down-weight outliers when accumulating the sufficient statistics used for each profile's mean/covariance – in the EM engine's M-step (see `robust_m_step`) and, in the same spirit, in the MCMC engine's Gibbs updates (see "Robust estimation" below). If `FALSE`, this down-weighting is skipped in both engines, which reduce to classical (non-robust) maximum-likelihood / Bayesian estimation for a Gaussian mixture – useful if you want to fit a standard LPA/GMM with the same interface, missing-data handling, and model parameterizations as the rest of this package.

`alpha` Significance level for the Huber down-weighting threshold (observations with squared Mahalanobis distance beyond the `1 - alpha` chi-squared quantile are down-weighted). Smaller values down-weight fewer, more extreme points; larger values down-weight more aggressively. Default `0.05`. Ignored when `robust = FALSE`.

`n_chains` Number of independent MCMC chains to run (default `4`). Running multiple chains from independent starting values is what makes the Gelman-Rubin $\hat{R}$ diagnostic possible (it is a between- vs. within-chain comparison and is undefined for a single chain). Posterior summaries (`means`, `covariances`, `proportions`) pool the post-burn-in draws of all chains together. Ignored when `engine = "EM"`.

`cores` Integer, number of CPU cores to use for parallel estimation *within this single* `robust_lpa()` call (default `1`, sequential). For `engine = "EM"`, parallelizes across the `n_starts` random restarts. For `engine = "MCMC"`, parallelizes across the `n_chains` chains. Uses `parallel::mclapply()` (forking) on Unix-alikes, and a `parallel::makeCluster()` PSOCK cluster – with independent per-worker RNG streams via `parallel::clusterSetRNGStream()` – on Windows; falls back to sequential execution with a `warning()` if the **parallel** package is unavailable. This is a different axis of parallelism from, and should not usually be combined with (nested parallelism can oversubscribe your CPU), the `cores` argument of `estimate_profiles_robust`, which instead parallelizes across `G`/`model` combinations.

Value

A list with S3 class `"robust_lpa"` (see `print.robust_lpa` and `summary.robust_lpa` for concise and detailed views of the fit) containing:

engine The estimation engine used.

means A list of length `G` with the estimated profile means.

covariances A list of length `G` with the estimated profile covariance matrices.

proportions Numeric vector of length `G` with the estimated mixing proportions.

probabilities An `n` x `G` matrix of posterior profile-membership probabilities.

fit A one-row data.frame with `Model`, `Profiles`, `LogLik`, `Parameters`, `AIC`, `BIC`, `SABIC`, `Entropy`, `Min_Size`, and `Max_Size`.

assignments Integer vector of length `n` with the most likely profile for each observation.

mcmc_draws (MCMC engine only) a list with `chains` (the raw per-chain draws, as consumed by `plot_mcmc_chains`), `n_chains`, `mcmc_iter`, and `burnin`.

mcmc_diagnostics (MCMC engine only) a data.frame with one row per scalar parameter (`Parameter`, `Rhat`, `ESS`); see "MCMC convergence diagnostics" below. `NULL` if the **coda** package is unavailable or there are too few post-burn-in iterations.

data The numeric matrix actually fit (`data` coerced via `as.matrix()`).

call_args A named list of every argument controlling this fit (`G`, `model`, `engine`, ...), for internal reuse – e.g. `bch_robust`'s `correction = "bootstrap"` refits this exact specification on resampled data via `do.call(robust_lpa, modifyList(call_args, list(data = new_data)))`.

Robust estimation

Both engines share the same outlier-down-weighting idea, adapted to how each one accumulates information:

- **EM**: at every M-step, each profile's mean/covariance is recomputed from Huber-down-weighted, posterior-probability-weighted observations (`robust_m_step`).
- **MCMC**: at every Gibbs sweep, after observations are allocated to profiles, a Huber weight is computed for each observation from its squared Mahalanobis distance to its *currently assigned* profile's previous-sweep mean/covariance (the same `alpha` chi-squared cutoff used by the EM engine). These weights down-weight the sufficient statistics that drive that sweep's mean/covariance updates, so an outlying observation contributes a smaller effective sample size to its profile's posterior. As in the EM engine, the mixing-proportion update is unaffected (it uses the raw allocation counts) and the allocation step itself is not down-weighted. Setting `robust = FALSE` recovers a standard (non-robust) Gibbs sampler for the same Bayesian-Lasso Gaussian mixture. A row of `data` with no observed variables at all is still (re)allocated to a profile every sweep, drawn from the current mixing proportions (its posterior given zero data), rather than being left permanently stuck in a single profile.

MCMC convergence diagnostics

When `engine = "MCMC"`, `$mcmc_diagnostics` reports, for every scalar parameter (`"mu[g,j]"`, `"sigma[g,j]"`, `"pi[g]"`):

- **Rhat**: the classic Gelman-Rubin potential scale reduction statistic (Gelman & Rubin, 1992), comparing between- and within-chain variance on the post-burn-in draws. Values noticeably above 1.1 are the classic rule-of-thumb warning sign of non-convergence, and trigger a `warning()`. `NA` when `n_chains = 1` (undefined for a single chain).
- **ESS**: the classic (autocorrelation/spectral-density-based) effective sample size, i.e. how many independent draws the autocorrelated post-burn-in draws (pooled across chains) are worth.

Computing these requires the **coda** package; `$mcmc_diagnostics` is `NULL` (with a `warning()`) if it is not installed.

References

Gelman, A., & Rubin, D. B. (1992). Inference from iterative simulation using multiple sequences. *Statistical Science*, 7(4), 457-472. doi:10.1214/ss/1177011136

Park, T., & Casella, G. (2008). The Bayesian Lasso. *Journal of the American Statistical Association*, 103(482), 681-686. doi:10.1198/016214508000000337

See Also

`estimate_profiles_robust` to fit and compare many `G` / `model` combinations at once, `blrt_robust` for a bootstrapped likelihood ratio test to choose the number of profiles, and `plot_mcmc_chains` to inspect MCMC chains.

Examples

```
# Fast demonstration on the bundled `neuro_data` dataset (standardized,
# as recommended -- see `lambda` and `prior_laplace` above).
data(neuro_data)
x <- scale(as.matrix(neuro_data[, c("Memory", "Attention", "Executive_Functions",
                                    "RT_Stroop", "RT_TMT")]))
# Huber-weighted robust estimation can occasionally emit a log-likelihood
# decrease warning as an expected side effect of down-weighting outliers
# mid-fit (see the "Robust estimation" section above); wrapped in
# suppressWarnings() below for a clean example, not because it signals a
# problem with the fit.
fit <- suppressWarnings(robust_lpa(x, G = 2, model = 6, n_starts = 2, max_iter = 30))
fit           # print.robust_lpa(): concise overview of the fit
summary(fit)  # summary.robust_lpa(): profile means/sizes and full fit indices

# `neuro_data` injects extra, variable-magnitude outliers into RT_Stroop
# and RT_TMT for a subset of the Pathological group; compare robust
# (default) vs. classical (robust = FALSE) estimation of the profile means.
fit_robust <- suppressWarnings(robust_lpa(x, G = 2, model = 6, n_starts = 2, max_iter = 30))
fit_classical <- robust_lpa(x, G = 2, model = 6, n_starts = 2, max_iter = 30, robust = FALSE)
sapply(fit_robust$means, `[`, "RT_Stroop")
sapply(fit_classical$means, `[`, "RT_Stroop")


# MCMC engine: 4 chains (default), with a small mcmc_iter for speed, and
# the resulting Gelman-Rubin / ESS convergence diagnostics.
fit_mcmc <- suppressWarnings(robust_lpa(x, G = 2, model = 6, engine = "MCMC",
                                        mcmc_iter = 200, n_chains = 4))
summary(fit_mcmc)  # includes $mcmc_diagnostics (Rhat / ESS) in the printout
```

`robust_mean` *Calculate a Simple Robust (Trimmed) Mean*

Description

Computes a one-step trimmed centroid: an observation is included in the average only if its Euclidean distance to the coordinate-wise median of `data` is below `threshold`. This is a quick, easy-to-reason-about robust location estimate, not an iterative M-estimator; for the full robust mixture-model estimation used elsewhere in this package, see `robust_lpa`.

Usage

```
robust_mean(data, threshold = 10)
```

Arguments

`data`	A matrix or data.frame of numeric observations.
`threshold`	Maximum Euclidean distance to the coordinate-wise median for an observation to be included in the average. Default `10`.

Details

`threshold` is a distance *from the coordinate-wise median of* `data`, not from the origin – so a sensible value depends on the scale and spread of your variables. A reasonable starting point is a small multiple of a typical per-variable standard deviation times `sqrt(ncol(data))` (roughly the scale of a Euclidean distance across all variables); `mahalanobis`-based thresholds (as used internally by `robust_lpa`) account for correlation and scale automatically and are preferable when variables are on very different scales.

Value

A numeric vector representing the robust mean of the variables. If no observation falls within `threshold` of the median, returns a vector of zeros with a warning.

Examples

```
data(neuro_data)
x <- scale(as.matrix(neuro_data[, c("Memory", "RT_Stroop")]))
r_mean <- robust_mean(x, threshold = 3)

# Print the calculated robust means
r_mean
```

`summary.robust_lpa` *Summarize a Fitted Robust LPA Model*

Description

Builds a compact summary of a model fitted by robust_lpa, limited to the information most people actually need to interpret a fit: per-profile means, profile sizes/mixing proportions, the headline fit indices, and, for the MCMC engine, the Gelman-Rubin $\hat{R}$ / effective sample size convergence range. Returns an object of class "summary.robust_lpa" with its own print method (print.summary.robust_lpa), following the usual summary()/print(summary()) convention used throughout R (e.g. summary.lm). For the full per-parameter Rhat/ESS table, use object$mcmc_diagnostics directly.

Usage

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

Arguments

object	A robust_lpa object, as returned by robust_lpa.
...	Currently ignored (present for S3 consistency with the generic summary).

Value

An object of class "summary.robust_lpa", a list with engine, model, G, n, means (a variables x profiles matrix), sizes (a data.frame of profile sizes/mixing proportions), fit (the one-row fit-indices data.frame, restricted to the headline columns), and, for the MCMC engine only, mcmc_info (chain configuration and convergence diagnostics).

See Also

robust_lpa, print.robust_lpa

Examples

```
data(neuro_data)
x <- scale(as.matrix(neuro_data[, c("Memory", "RT_Stroop")]))
fit <- suppressWarnings(robust_lpa(x, G = 2, model = 2, n_starts = 3, max_iter = 20))
summary(fit)
```

Index

∗ **datasets**
neuro_data, 8

bch_robust, 2, *8*, *15*
blrt_robust, 5, *8*, *16*

estimate_profiles_robust, 6, *8*, *11*, *14*, *16*

mahalanobis, *17*

neuro_data, 8

plot_mcmc_chains, 9, *15, 16*
plot_robust_lpa, 10
print, *11*
print.robust_lpa, 11, *14*, *18*
print.summary.robust_lpa, *18*

robust_lpa, *2, 3*, *5–7*, *9*, *11, 12*, 12, *17, 18*
robust_m_step, *13–15*
robust_mean, 16

summary, *18*
summary.robust_lpa, *11, 12*, *14*, 17