Package: RobustLPA (via r-universe)

October 28, 2026

Title Robust Latent Profile Analysis

Version 1.1.0

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. Robustness is obtained either by Huber-type down-weighting or by mixtures of multivariate t distributions (a likelihood-based robust model, see Peel and McLachlan (2000) <doi:10.1023/A:1008981510081>). Missing data are handled by full information maximum likelihood with the exact EM treatment of incomplete observations (data augmentation in the MCMC engine). The EM engine also supports LASSO regularization with k-fold cross-validation for penalty tuning; the MCMC engine uses a Bayesian Lasso with Laplace priors, multiple chains, Gelman-Rubin/effective sample size diagnostics and the widely applicable information criterion. It supports six geometric variance-covariance models, along with functions for bootstrapped likelihood ratio tests (BLRT), BCH auxiliary variable analysis, and plotting. For longitudinal data, it fits robust growth mixture models and latent class growth analysis (Muthen and Shedden (1999) <doi:10.1111/j.0006-341X.1999.00463.x>) for one or several outcomes measured on unbalanced occasions, with Gaussian, Huber-weighted or multivariate-t (Pinheiro, Liu and Wu (2001) <doi:10.1198/10618600152628059>) latent classes, by EM and MCMC, and optional adaptive LASSO penalties (Zou (2006) <doi:10.1198/016214506000000735>) that identify stable trajectories and the outcomes that differentiate the classes. 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

LinkingTo Rcpp, RcppArmadillo

Imports Rcpp, ggplot2, stats, utils, bayesplot, coda

Suggests parallel, knitr, rmarkdown, testthat (>= 3.1.5), lme4, nlme

Config/testthat/edition 3

VignetteBuilder knitr

Depends R (>= 3.6)

LazyData true

NeedsCompilation yes

Config/roxygen2/version 8.0.0

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

Maintainer Valerio Riccardo Aquila <valerio_aquila@hotmail.it>

Config/pak/sysreqs libicu-dev

Repository https://vraquila.r-universe.dev

Date/Publication 2026-09-27 22:50:15 UTC

RemoteUrl https://github.com/cran/RobustLPA

RemoteRef HEAD

RemoteSha f818f52a1d9f00f65fd46b9d8981b4b355587a54

Contents

bch_robust . . . 3
blrt_gmm_robust . . . 5
blrt_robust . . . 7
estimate_gmm_robust . . . 9
estimate_profiles_robust . . . 10
neuro_data . . . 12
neuro_long . . . 13
plot_mcmc_chains . . . 14
plot_robust_gmm . . . 15
plot_robust_lpa . . . 16
print.robust_gmm . . . 17
print.robust_lpa . . . 18
robust_gmm . . . 19
robust_lpa . . . 25
robust_mean . . . 31
summary.robust_gmm . . . 32
summary.robust_lpa . . . 33

Index **34**

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 model returned by `robust_lpa` (latent profiles) or `robust_gmm` (latent classes of trajectories).
`aux_var`	A numeric vector of the continuous auxiliary (distal outcome) variable, of length `nrow(model$probabilities)`: one value per observation for `robust_lpa()` fits, one value per person, in the order of `model$ids`, for `robust_gmm()` fits (e.g. a baseline characteristic, or a distal outcome). `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 error matrix **D** is estimated as

$$D_{t,s} = P(\hat{C} = s \mid C = t) \approx \frac{\sum_{i:\,\hat{C}_i = s} \hat{p}_{it}}{\sum_{i=1}^{n} \hat{p}_{it}}$$

(Bolck, Croon, & Hagenaars, 2004; Vermunt, 2010), so each row of **D** sums to 1. The BCH weight matrix is $W = D^{-1}$, and the classification-error-corrected mean of the auxiliary variable Y for profile t is

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

summed over *every* observation: each contributes to every profile's mean with a (possibly negative) cross-class weight $W_{\hat{C}_i,t}$, which is what removes the attenuation bias of a naive comparison of modal-assignment groups. This is unbiased under the BCH assumption that Y is independent of the modal assignment given the true profile.

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 $W_{\hat{C}_i,g}$), 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); for `robust_gmm()` fits whole persons (with all their occasions) are resampled (a cluster bootstrap). 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 by an exact optimal assignment of standardized profile means, to avoid mixing different real-world profiles together across resamples). Each refit runs sequentially (`cores = 1`) inside its bootstrap replicate, whatever `cores` the original model used, so parallelism happens only across replicates. 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` classification error matrix **D**, with `D[t, s]` = P(assigned `s` | true profile `t`); rows sum to 1.

Classification_Weights The `G x G` BCH weight matrix $W = D^{-1}$ (rows: assigned profile; columns: true profile).

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")]))
set.seed(1)
fit <- 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 <- bch_robust(fit, neuro_data$RT_TMT, correction = "bootstrap", n_boot = 30)
bch_res_boot$Bootstrap_Correction
```

`blrt_gmm_robust` *Bootstrapped Likelihood Ratio Test for Robust Growth Mixture Models*

Description

Tests a growth mixture model with `G` classes against the same model with `G - 1` classes by parametric bootstrap (McLachlan, 1987; Nylund, Asparouhov & Muthen, 2007): data are simulated from the fitted `G - 1`-class model – for the persons, occasions and outcomes actually observed, so the visit schedule and the missingness pattern are reproduced exactly – and both models are refitted to every simulated dataset. With `robust_method = "t"` the data are simulated from the fitted multivariate-t model and the test compares two proper t-mixture likelihoods; with the Huber estimator the statistic is a heuristic.

Usage

```
blrt_gmm_robust(
  data,
  id,
  time,
  outcomes,
  G,
  n_samples = 50,
  n_starts = 3,
  cores = 1,
  ...
)
```

Arguments

data, id, time, outcomes	See robust_gmm.
G	Number of classes of the alternative model (at least 2).
n_samples	Number of bootstrap samples (default 50; use 200 or more for publication).
n_starts	Number of EM starts for every fit.
cores	Number of cores across bootstrap samples.
...	Further arguments passed to robust_gmm (the engine is always EM and penalties are not allowed).

Value

A list with LRT_Observed, Bootstrap_LRTs, p_value (with the "+1" correction) and Bootstrap_Failures.

References

McLachlan, G. J. (1987). On bootstrapping the likelihood ratio test statistic for the number of components in a normal mixture. *Journal of the Royal Statistical Society: Series C*, 36(3), 318-324. doi:10.2307/2347790

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

See Also

robust_gmm, estimate_gmm_robust

Examples

```
data(neuro_long)
set.seed(1)
blrt_gmm_robust(neuro_long, id = "ID", time = "Year", outcomes = "Memory",
                G = 2, n_samples = 5, n_starts = 2)
```

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`, `robust_method`, `nu`, `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.

Details

Data are simulated from the fitted null model in the same family that is being fitted: a Gaussian mixture for classical and Huber fits, and a multivariate-t mixture (with the fitted `nu`) for `robust_method = "t"` fits. With `robust_method = "t"` the test compares two proper t-mixture likelihoods and is the recommended robust option. With the Huber estimator the statistic is computed from Gaussian log-likelihoods evaluated at robust (non-ML) estimates and the null data are simulated without contamination, so the test is a heuristic whose calibration on contaminated data is not guaranteed.

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")]))
set.seed(1)
blrt_res <- blrt_robust(x, G = 2, model = 1, n_samples = 5, n_starts = 3)
# Print the summary of the results
blrt_res
```

`estimate_gmm_robust` *Estimate Robust Growth Mixture Models Across Numbers of Classes*

Description

Fits `robust_gmm` for every number of classes in `n_classes` (and every random-effect specification in `random`), optionally selecting the LASSO penalties of each model by BIC or by K-fold cross-validation over persons, and collects the fit indices in one table.

Usage

```
estimate_gmm_robust(
  data,
  id,
  time,
  outcomes,
  n_classes = 1:3,
  random = "slope",
  cores = 1,
  tune_penalty = c("none", "bic", "cv"),
  penalty = c("both", "growth", "diff"),
  z_grid = c(1.5, 2, 2.5, 3, 4),
  k_folds = 5,
  ...
)
```

Arguments

`data`, `id`, `time`, `outcomes`	See `robust_gmm`.
`n_classes`	Integer vector of numbers of classes to fit.
`random`	Character vector of random-effect specifications to fit (see `robust_gmm`).
`cores`	Number of cores used across the grid of models (results are identical for any value given the same `set.seed()`).
`tune_penalty`	`"none"` (default; penalties, if any, are taken from `...`), `"bic"` or `"cv"`.
`penalty`	Which penalties to tune: `"both"` (default), `"growth"` or `"diff"`.
`z_grid`	Candidate thresholds (see Details); `0` (no penalty) is always included.
`k_folds`	Number of folds for `tune_penalty = "cv"`.
`...`	Further arguments passed to `robust_gmm` (e.g. `degree`, `robust_method`, `group_diff`, `relax`, `n_starts`, `engine`).

Details

With `tune_penalty = "bic"` or `"cv"`, the penalty levels are chosen from `z_grid`: each value z gives $\lambda = z^2/N$ (N = number of persons). With the default adaptive weights of `robust_gmm`, a growth term (or class difference) is then set to zero approximately when its Wald statistic is below z in absolute value, which makes the grid directly interpretable (e.g. $z = 2$ or $z = \sqrt{\log N}$). `"bic"` fits every candidate on the full data and keeps the lowest BIC (whose parameter count accounts for the zeros and fusions); `"cv"` keeps the candidate with the highest held-out log-likelihood over `k_folds` folds of persons and refits it on the full data.

Value

A list with `fit_table` (one row per model), `models` (named `"G<g>_<random>"`) and, when penalties are tuned, `tuning` (the criterion of every candidate).

See Also

`robust_gmm`, `blrt_gmm_robust`

Examples

```
data(neuro_long)
set.seed(1)
res <- estimate_gmm_robust(neuro_long, id = "ID", time = "Year",
                           outcomes = "Memory", n_classes = 1:2, n_starts = 2)
res$fit_table
```

`estimate_profiles_robust`

Estimate Robust Latent Profile Models Across Profiles and Models

---

Description

Fits every combination of `n_profiles` and `models` with `robust_lpa` and collects their fit indices in one table.

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. This is a different axis of parallelism from robust_lpa's own cores argument; if you also pass cores through ..., keep the product of the two at or below your machine's core count. Results are identical for any value of cores given the same set.seed().
n_starts	Number of initializations per model.
lambda	Fixed penalty for LASSO (EM engine).
tune_lasso	Logical. If TRUE, selects lambda from lambda_grid by k-fold cross-validation of the held-out observed-data log-likelihood (EM engine only; ignored with a warning for engine = "MCMC", whose shrinkage is governed by prior_laplace).
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, robust_method, nu, alpha, init, and, when engine = "MCMC", mcmc_iter/n_chains/prior_laplace).

Value

A list containing fit_table (one row per successfully fitted model, with a Lambda column) and models (the fitted robust_lpa objects, named "model_<m>_profiles_<G>").

Examples

```
data(neuro_data)
x <- scale(as.matrix(neuro_data[, c("Memory", "RT_Stroop")]))
set.seed(1)
res <- estimate_profiles_robust(x, n_profiles = 1:2, models = 1, n_starts = 3)
res$fit_table
summary(res$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 about 93% accuracy (classical, Huber or t estimation, model = 6, G = 2).

Source

Simulated data for testing and documentation purposes.

---

neuro_long *Simulated Longitudinal Neuropsychological Dataset for Robust Growth Mixture Models*

Description

A synthetic longitudinal dataset (long format: one row per person and annual visit) designed as a worked example for robust_gmm: three latent classes of cognitive change, three outcomes, unbalanced follow-up with missed visits and drop-out, and a few gross data-entry errors.

Usage

```
neuro_long
```

Format

A data frame with one row per person and visit, and 8 variables:

ID Person identifier.

Year Years since baseline (0 to 5).

Memory, Executive, Speed Simulated test scores (NA when not observed).

True_Class The true latent class (not used for estimation).

Age Age at baseline (constant within person).

Biomarker A baseline biomarker level (constant within person).

Details

400 persons are assigned to three latent classes ("Stable", "Slow decline", "Fast decline"; about 50/30/20%) and followed for up to six annual visits (`Year` 0 to 5). `Memory` and `Executive` (T-score-like metric) decline at class-specific rates (Memory: 0, -2 and -5 points per year; Executive: 0, -1.5 and -4 points per year), whereas `Speed` declines by 0.5 points per year *in every class* (an outcome that does not differentiate the classes, useful to illustrate the group LASSO of `robust_gmm(lambda_diff = , group_diff = TRUE)`; the "Stable" class has exactly zero slopes on Memory and Executive, useful to illustrate `lambda_growth`). Within classes, persons have correlated random intercepts (SD 5) and slopes (SD 0.4) on every outcome, and residual errors with SD 2.5. After each visit a person drops out with a probability that increases as the last observed Memory score decreases (missing at random); 8% of the follow-up visits are missed and 5% of the single test scores are missing. For 4% of the persons one score is corrupted by a gross error of 25 to 40 points. `Age` and `Biomarker` are baseline characteristics that differ between the classes, for illustrating `bch_robust` on a `robust_gmm()` fit. The generating code (with its fixed seed) is installed with the package: run `source(system.file("scripts", "generate_neuro_long.R", package = "RobustLPA"))` to rebuild the data set.

Source

Simulated; see Details.

Examples

```
data(neuro_long)
head(neuro_long)
table(table(neuro_long$ID))   # number of visits per person
```

---

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` or `robust_gmm` with `engine = "MCMC"`. For growth mixture fits the parameters are the class trajectories (`"beta[g,outcome:term]"`), random-effect variances (`"D[...]"`), residual variances (`"sigma2[...]"`), class proportions (`"pi[g]"`) and, if estimated, `"nu"`, all on the original scale.
`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

mcmc_chain_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), "pi[g]" (mixing proportion of profile g), and, for robust_method = "t" fits with an estimated nu, "nu" (the t degrees of freedom). Off-diagonal covariance terms are not included by default to keep the default plot readable; inspect model$mcmc_draws$chains[[1]]$sigma_chain 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_gmm *Plot the Class Trajectories of a Robust Growth Mixture Model*

Description

Draws, for every outcome, the estimated mean trajectory of each latent class over the observed time range, optionally over the observed individual trajectories coloured by modal class.

Usage

```
plot_robust_gmm(
  model,
  outcomes = NULL,
  individual = TRUE,
  max_individuals = 300,
  title = "Latent class trajectories"
)
```

Arguments

model	A robust_gmm object.
outcomes	Optional subset of outcomes to plot (default: all).
individual	Logical; draw the observed individual trajectories (default TRUE).
max_individuals	Maximum number of persons whose trajectories are drawn (a random subset if there are more). Default 300.
title	Plot title.

Value

A ggplot object.

See Also

robust_gmm

Examples

```
data(neuro_long)
set.seed(1)
fit <- robust_gmm(neuro_long, id = "ID", time = "Year", outcomes = "Memory",
                  G = 2, n_starts = 2)
plot_robust_gmm(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_gmm *Print a Fitted Robust Growth Mixture Model*

Description

Compact overview of a model fitted by robust_gmm: engine, number of classes and persons, outcomes, random-effect and robustness settings, headline fit indices, class proportions and, if used, the LASSO penalties.

Usage

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

Arguments

x	A robust_gmm object.
...	Currently ignored.

Value

x, invisibly.

See Also

robust_gmm, summary.robust_gmm

---

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 <- robust_lpa(x, G = 2, model = 2, n_starts = 3, max_iter = 20)
print(fit)
```

robust_gmm *Robust Growth Mixture Model (Latent Class Growth Analysis)*

Description

Fits a growth mixture model (GMM) – a finite mixture of linear mixed-effects models for longitudinal data (Verbeke & Lesaffre, 1996; Muthen & Shedden, 1999) – or, with `random = "none"`, a latent class growth analysis (LCGA; Nagin, 1999), with one or several outcomes measured repeatedly on the same persons. Each latent class ("profile of change") has its own polynomial mean trajectory for every outcome; within a class, persons deviate from it through correlated random effects and residual errors. The model can be estimated by maximum likelihood (EM engine) or by Gibbs sampling (MCMC engine), in a classical (Gaussian), Huber-weighted or multivariate-t (robust) version, optionally with LASSO penalties on the class trajectories.

Usage

```
robust_gmm(
  data,
  id,
  time,
  outcomes,
  G,
  degree = 1,
  random = c("slope", "intercept", "none"),
  re_structure = c("full", "block", "diagonal"),
  re_cov = c("equal", "varying"),
  resid_var = c("equal", "varying"),
  engine = "EM",
  robust = TRUE,
  robust_method = c("huber", "t"),
  nu = NULL,
  alpha = 0.05,
  lambda_growth = 0,
  lambda_diff = 0,
  group_diff = FALSE,
  adaptive = TRUE,
  relax = FALSE,
  n_starts = 5,
  max_iter = 500,
  tol = 1e-08,
  init = c("kmeans", "random"),
  mcmc_iter = 2000,
  n_chains = 4,
  cores = 1
)
```

Arguments

data	A data.frame in long format: one row per person and occasion, with the person identifier, the time variable and the outcome columns (NA allowed in the outcomes).
id, time	Names of the person-identifier and time columns.
outcomes	Character vector with the names of one or more outcome columns (modelled jointly).
G	Number of latent classes.
degree	Degree of the polynomial trajectory in time (default 1, linear).
random	Random effects within classes: "slope" (random intercept and slope for every outcome; default), "intercept" (random intercept only) or "none" (no random effects: latent class growth analysis).
re_structure	Covariance structure of the random effects: "full" (all correlated, default), "block" (correlated within an outcome, independent across outcomes) or "diagonal".
re_cov	Random-effect covariance "equal" across classes (default, the usual and more stable choice) or "varying".
resid_var	Residual variances "equal" across classes (default) or "varying".
engine	"EM" (default) or "MCMC".
robust	Logical; FALSE fits the classical Gaussian model.
robust_method	"huber" (default) or "t"; see "Robust estimation".
nu	Degrees of freedom of the t model: NULL (default) estimates it, a number fixes it.
alpha	Huber significance level.
lambda_growth, lambda_diff	Non-negative LASSO penalty levels (see "LASSO penalties"); default 0 (no penalty).
group_diff	Logical; group-wise (by outcome) difference penalty.
adaptive	Logical; adaptive-Lasso weights from the unpenalized fit (default TRUE; see "LASSO penalties").
relax	Logical; refit without penalty keeping the selected zeros and fusions (EM engine only).
n_starts	Number of EM starts (also used for the EM fit that initializes the MCMC engine).
max_iter	Maximum number of EM iterations.
tol	Relative convergence tolerance on the (penalized) log-likelihood.
init	Initialization of the EM starts: "kmeans" (k-means on person-level least-squares trajectories; default) or "random" (G persons drawn at random as class centres, every person assigned to the nearest one).
mcmc_iter	Iterations per MCMC chain (first half discarded).
n_chains	Number of MCMC chains.
cores	Number of cores for the EM starts / MCMC chains. Results are identical for any value given the same set.seed().

Value

An object of class `"robust_gmm"`, a list with:

engine, robust_method, nu Estimation settings (`nu`: the estimated or fixed t degrees of freedom, `NULL` otherwise).

coefficients A list of `G` matrices (outcomes x polynomial terms): the class mean trajectories.

random_cov A list of `G` random-effect covariance matrices (`NULL` for `random = "none"`).

residual_var A `G` x `K` matrix of residual variances.

proportions Class proportions.

probabilities, assignments Posterior class probabilities (persons x classes) and modal classes, in the order of `ids`.

ids The person identifiers, in the order used by every person-level output (use it to align auxiliary variables for `bch_robust`).

weights Per-person robustness weights, averaged over classes with the posterior probabilities. With `robust_method = "huber"` they lie in (0, 1] (1 = full weight). With `robust_method = "t"` they are the E-step weights $(\nu + n_i)/(\nu + \delta_i)$, where n_i is the number of observed values and δ_i the squared Mahalanobis distance of person i: they exceed 1 for persons closer to their class trajectory than expected and are small for outlying persons. All 1 when `robust = FALSE`.

random_effects Posterior means of each person's random effects under his/her modal class.

fit One-row data.frame: `Classes`, `LogLik`, `Parameters`, `AIC`, `BIC` (with $\log N$, N = persons), `SABIC`, `Entropy`, `Min_Size`, `Max_Size`, the penalty levels, and `WAIC` (MCMC).

penalty Penalty settings and the selected zeros / fusions.

converged, iterations EM convergence information.

mcmc_draws, mcmc_diagnostics, waic MCMC output (see `plot_mcmc_chains`).

spec, internal, call_args, data Model specification, standardized-scale estimates, arguments and data used for refitting.

Model

For person i in class g, the stacked vector y_i of all his/her observed values (all outcomes, all occasions) is

$$y_i = X_i\beta_g + Z_i b_i + e_i, \quad b_i \sim N(0, D_g/u_i), \quad e_i \sim N(0, R_g/u_i),$$

where X_i contains, for each observation of outcome k at time t, the polynomial basis $(1, t, \ldots, t^{degree})$ in the block of outcome k; Z_i the first 1 (`random = "intercept"`) or 2 (`random = "slope"`) of those terms; R_g is diagonal with one residual variance per outcome; and $u_i = 1$ (Gaussian) or $u_i \sim \mathrm{Gamma}(\nu/2, \nu/2)$ (multivariate t: Pinheiro, Liu & Wu, 2001). Marginally, y_i is Gaussian or multivariate t with mean $X_i\beta_g$ and scale $V_{ig} = Z_i D_g Z_i' + R_g$.

Persons may be observed at different times, on different numbers of occasions, and not on every outcome at every occasion: each person contributes the exact likelihood of the values actually observed, so the estimates are maximum likelihood under missing-at-random missingness (including drop-out that depends on earlier observed values). Every outcome is standardized internally (all reported estimates are on the original scale); the time variable is used as supplied, so center it where the intercept should be interpreted (e.g. years since baseline).

Robust estimation

`robust_method = "t"` A mixture of multivariate-t linear mixed models: the random effects and the errors of a person share the latent scale u_i, so a person whose trajectory is far from every class (or who has a few gross errors) is down-weighted as a whole through $E[u_i] = (\nu + n_i)/(\nu + d_i)$, with d_i the Mahalanobis distance of y_i and n_i its length. It is a proper likelihood model, fitted by ECM with an ECME update of ν, so information criteria, the bootstrapped likelihood ratio test (`blrt_gmm_robust`) and the BCH method are used as intended. Recommended.

`robust_method = "huber"` Huber weights computed from d_i with a $\chi^2_{n_i}$ cutoff (`alpha`) down-weight outlying persons in the M-step; an estimating-equation approach whose reported log-likelihood is the Gaussian one evaluated at the robust estimates (a heuristic for information criteria).

LASSO penalties

Two penalties on the class trajectories (fixed effects) are available, separately or together; both act on the standardized outcome scale and are expressed per person, i.e. the EM engine maximizes

$$\ell(\theta)/N - \lambda_{growth} \sum_{g,k,l \geq 1} |\beta_{gkl}| - \lambda_{diff} \sum_{g,k,l} |\beta_{gkl} - \bar{\beta}_{kl}|,$$

with $\bar{\beta}_{kl}$ the (unweighted) mean over classes.

`lambda_growth` Sparse trajectories: the growth terms (slope, quadratic, ...) of every class and outcome are shrunk toward 0, intercepts are not penalized. A coefficient set exactly to 0 means that the class does not change on that outcome (e.g. a "stable" class). This is the Lasso of Tibshirani (1996) applied to the fixed effects of a mixture of regressions (Khalili & Chen, 2007; Du et al., 2013).

`lambda_diff` Sparse class differences: every coefficient of every class is shrunk toward the across-class mean, so that coefficients (or, with `group_diff = TRUE`, whole outcomes) on which the classes do not differ are fused. With `group_diff = TRUE` the penalty is $\lambda_{diff} \sum_k w_k \| S_k (B_k - 1\bar{\beta}_k') \|$, where S_k scales every coefficient by the square root of its information (so that intercepts and slopes are penalized on comparable scales; Simon & Tibshirani, 2012) and w_k is the adaptive weight; it removes outcome k from the class separation altogether when its group is set to zero – the grouped variable-selection penalty of Xie, Pan & Shen (2008), which extends the penalized model-based clustering of Pan & Shen (2007) used by `robust_lpa`'s `lambda`.

By default (`adaptive = TRUE`) the penalties are adaptive (Zou, 2006; Wang & Leng, 2008 for the group version): every coefficient, deviation or outcome group is weighted by the reciprocal of its unpenalized estimate, which gives consistent selection and makes the penalty level interpretable – a term is set to zero approximately when its Wald statistic is below $\sqrt{N\lambda}$ in absolute value, so `lambda = z^2 / N` corresponds to a threshold `z` (e.g. `4 / N` for $|z| < 2$). The penalized fit is started from the unpenalized maximum-likelihood solution (which also supplies the weights). The penalized fixed-effects step of each iteration is solved exactly (to numerical tolerance) by ADMM (Boyd et al., 2011); the number of fixed-effect parameters used by AIC/BIC is the number of free coefficients left by the zeros and fusions (the generalized-Lasso degrees of freedom; Tibshirani & Taylor, 2011). Penalized estimates are shrunk toward zero / toward each other: `relax = TRUE` refits the model without penalty while keeping the selected zeros and fusions (the relaxed Lasso; Meinshausen, 2007), which is preferable for reporting and inference. Choose `lambda_growth` / `lambda_diff` with `estimate_gmm_robust` (BIC or cross-validation over persons). In the MCMC

engine the same penalties become Bayesian-Lasso (Laplace) priors with rate $N\lambda$ times the adaptive weight of each term (Park & Casella, 2008; Bayesian group Lasso of Kyung et al., 2010, when `group_diff = TRUE`), whose posterior mode is the EM penalized estimate; `relax` does not apply to the MCMC engine.

Estimation

The EM engine uses an alternating ECM algorithm (Meng & van Dyk, 1997): each iteration updates the class proportions and the class trajectories by (penalized) weighted generalized least squares with the random effects integrated out, then, after a fresh E-step, the residual variances, the random-effect covariances and (t model) ν; every step increases the (penalized) observed-data log-likelihood. The iterations are accelerated by SQUAREM (Varadhan & Roland, 2008) with a monotonicity safeguard. Several starts are run (k-means on person-level least-squares trajectories by default) and the best is kept.

MCMC details

The Gibbs sampler allocates persons with the random effects and latent scales integrated out, then draws ν (t model; random-walk Metropolis with the latent scales integrated out, Gamma(2, 0.1) prior), the latent scales, the class trajectories with the random effects integrated out, the random effects, the random-effect covariance matrices, the residual variances (inverse-gamma(1, 0.1) on the standardized scale) and the mixing proportions (Dirichlet(1, ..., 1)). The random-effect covariances have a scaled inverse-Wishart prior (O'Malley & Zaslavsky, 2008), $D = \mathrm{diag}(a)\Psi\mathrm{diag}(a)$ with $\Psi \sim IW(q+1, I)$ per block and $a_j \sim N(0, A_j^2)$ (A_j = 5 standardized units for intercepts, divided by the standard deviation of the time term for slopes), sampled by parameter expansion (Liu & Wu, 1999; Gelman et al., 2008), which avoids the slow mixing of small variance components. The fixed effects have vague normal priors, combined with the Bayesian-Lasso priors when penalties are requested. The chains start from the EM fit with perturbed trajectories, draws are relabeled to that EM solution by an optimal assignment of the class mean trajectories, and the first half of each chain is discarded. The Huber option is a heuristic in the MCMC engine, as in `robust_lpa`.

References

Boyd, S., Parikh, N., Chu, E., Peleato, B., & Eckstein, J. (2011). Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, 3(1), 1-122. doi:10.1561/2200000016

Du, Y., Khalili, A., Neslehova, J. G., & Steele, R. J. (2013). Simultaneous fixed and random effects selection in finite mixture of linear mixed-effects models. *Canadian Journal of Statistics*, 41(4), 596-616. doi:10.1002/cjs.11192

Gelman, A., van Dyk, D. A., Huang, Z., & Boscardin, W. J. (2008). Using redundant parameterizations to fit hierarchical models. *Journal of Computational and Graphical Statistics*, 17(1), 95-122. doi:10.1198/106186008X287337

Khalili, A., & Chen, J. (2007). Variable selection in finite mixture of regression models. *Journal of the American Statistical Association*, 102(479), 1025-1038. doi:10.1198/016214507000000590

Kyung, M., Gill, J., Ghosh, M., & Casella, G. (2010). Penalized regression, standard errors, and Bayesian lassos. *Bayesian Analysis*, 5(2), 369-411. doi:10.1214/10BA607

Liu, J. S., & Wu, Y. N. (1999). Parameter expansion for data augmentation. *Journal of the American Statistical Association*, 94(448), 1264-1274. doi:10.1080/01621459.1999.10473879

Meinshausen, N. (2007). Relaxed Lasso. *Computational Statistics & Data Analysis*, 52(1), 374-393. doi:10.1016/j.csda.2006.12.019

Meng, X.-L., & van Dyk, D. (1997). The EM algorithm – an old folk-song sung to a fast new tune. *Journal of the Royal Statistical Society: Series B*, 59(3), 511-567. doi:10.1111/14679868.00082

Muthen, B., & Shedden, K. (1999). Finite mixture modeling with mixture outcomes using the EM algorithm. *Biometrics*, 55(2), 463-469. doi:10.1111/j.0006341X.1999.00463.x

Nagin, D. S. (1999). Analyzing developmental trajectories: A semiparametric, group-based approach. *Psychological Methods*, 4(2), 139-157. doi:10.1037/1082989X.4.2.139

O'Malley, A. J., & Zaslavsky, A. M. (2008). Domain-level covariance analysis for multilevel survey data with structured nonresponse. *Journal of the American Statistical Association*, 103(484), 1405-1418. doi:10.1198/016214508000000724

Pan, W., & Shen, X. (2007). Penalized model-based clustering with application to variable selection. *Journal of Machine Learning Research*, 8, 1145-1164.

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

Pinheiro, J. C., Liu, C., & Wu, Y. N. (2001). Efficient algorithms for robust estimation in linear mixed-effects models using the multivariate t distribution. *Journal of Computational and Graphical Statistics*, 10(2), 249-276. doi:10.1198/10618600152628059

Simon, N., & Tibshirani, R. (2012). Standardization and the group lasso penalty. *Statistica Sinica*, 22(3), 983-1001.

Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B*, 58(1), 267-288. doi:10.1111/j.25176161.1996.tb02080.x

Tibshirani, R. J., & Taylor, J. (2011). The solution path of the generalized lasso. *The Annals of Statistics*, 39(3), 1335-1371. doi:10.1214/11AOS878

Varadhan, R., & Roland, C. (2008). Simple and globally convergent methods for accelerating the convergence of any EM algorithm. *Scandinavian Journal of Statistics*, 35(2), 335-353. doi:10.1111/j.14679469.2007.00585.x

Verbeke, G., & Lesaffre, E. (1996). A linear mixed-effects model with heterogeneity in the random-effects population. *Journal of the American Statistical Association*, 91(433), 217-221. doi:10.1080/01621459.1996.10476679

Wang, H., & Leng, C. (2008). A note on adaptive group lasso. *Computational Statistics & Data Analysis*, 52(12), 5277-5286. doi:10.1016/j.csda.2008.05.006

Xie, B., Pan, W., & Shen, X. (2008). Variable selection in penalized model-based clustering via regularization on grouped parameters. *Biometrics*, 64(3), 921-930. doi:10.1111/j.15410420.2007.00955.x

Zou, H. (2006). The adaptive lasso and its oracle properties. *Journal of the American Statistical Association*, 101(476), 1418-1429. doi:10.1198/016214506000000735

See Also

`estimate_gmm_robust` (number of classes and penalty selection), `blrt_gmm_robust`, `plot_robust_gmm`, `bch_robust` (relating the classes to baseline or distal variables), `plot_mcmc_chains`.

Examples

```
data(neuro_long)
set.seed(1)
fit <- robust_gmm(neuro_long, id = "ID", time = "Year",
                  outcomes = c("Memory", "Executive"), G = 2,
                  robust_method = "t", n_starts = 2)
fit
summary(fit)
plot_robust_gmm(fit)


# Sparse class differences (group LASSO by outcome), relaxed refit
# (adaptive weights: lambda = z^2 / N removes groups with |z| below ~z)
N <- length(unique(neuro_long$ID))
fit_l <- robust_gmm(neuro_long, id = "ID", time = "Year",
                    outcomes = c("Memory", "Executive", "Speed"), G = 3,
                    robust_method = "t", lambda_diff = 9 / N,
                    group_diff = TRUE, relax = TRUE, n_starts = 2)
fit_l$penalty$outcome_selected

# MCMC engine
fit_b <- robust_gmm(neuro_long, id = "ID", time = "Year",
                    outcomes = "Memory", G = 2, engine = "MCMC",
                    robust_method = "t", mcmc_iter = 400, n_chains = 2)
summary(fit_b)
```

`robust_lpa` *Fit a Single Robust Latent Profile Analysis Model*

Description

Estimates a Latent Profile Analysis (finite mixture) model that is robust to multivariate outliers and handles missing data by full-information maximum likelihood, using either an EM or an MCMC (Bayesian Lasso) engine. Two robust estimators are available (see "Robust estimation" below): Huber-type down-weighting (`robust_method = "huber"`, the default) and a mixture of multivariate t distributions (`robust_method = "t"`), which is a proper likelihood-based model. The MCMC engine runs `n_chains` independent chains (4 by default) and reports Gelman-Rubin $\hat{R}$, effective sample size and WAIC. 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,
  robust_method = c("huber", "t"),
  nu = NULL,
  init = c("kmeans", "random")
)
```

Arguments

`data`	A matrix or data.frame of observations (numeric columns only; `NA` is allowed, see "Missing data").
`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). Both engines implement exactly the same six parameterizations. Models 4 and 5 have no closed-form M-step; the EM engine fits them by conditional maximization steps that never decrease the likelihood.
`engine`	String. Either `"EM"` (default) or `"MCMC"`.
`max_iter`	Maximum number of EM iterations. With `engine = "MCMC"`, used for the preliminary EM fit that initializes the chains.
`tol`	Tolerance for EM convergence (absolute change of the observed-data log-likelihood between iterations).

`n_starts` Number of EM initializations; the fit with the highest log-likelihood across starts is returned. With `engine = "MCMC"`, the preliminary EM fit uses `min(n_starts, 5)` starts.

`lambda` Non-negative soft-thresholding (LASSO-type) penalty applied to the profile means, via direct per-coordinate soft-thresholding of the (robustness- and posterior-probability-weighted) mean at every M-step. 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, 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 log-likelihood except when profile covariances are close to the identity. 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). Ignored by the MCMC engine (see `prior_laplace`).

`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; **larger** values induce **more** shrinkage of the means toward zero). Ignored when `engine = "EM"`.

`robust` Logical. If `TRUE` (default), robust estimation is used with the method selected by `robust_method`. If `FALSE`, both engines reduce to classical (non-robust) Gaussian-mixture estimation with the same interface, missing-data handling and parameterizations.

`alpha` Significance level for the Huber down-weighting threshold (observations with squared Mahalanobis distance beyond the `1 - alpha` chi-squared quantile are down-weighted). Default `0.05`. Used only when `robust = TRUE` and `robust_method = "huber"`.

`n_chains` Number of independent MCMC chains to run (default 4). Ignored when `engine = "EM"`.

`cores` Integer, number of CPU cores to use for parallel estimation *within this single* `robust_lpa()` call (default 1). For `engine = "EM"`, parallelizes across the `n_starts` random restarts; for `engine = "MCMC"`, across the `n_chains` chains. Uses `parallel::mclapply()` on Unix-alikes and a PSOCK cluster on Windows. Results are identical for any value of `cores` given the same `set.seed()` (see "Reproducibility").

`robust_method` Either `"huber"` (default) or `"t"`. See "Robust estimation". Ignored when `robust = FALSE`.

`nu` Degrees of freedom of the multivariate-t components when `robust_method = "t"`. `NULL` (default) estimates a single `nu` shared by all profiles (EM: ECME update maximizing the observed-data log-likelihood over `[1, 200]`; MCMC: Metropolis step under a Gamma(2, 0.1) prior). A positive number fixes it (e.g. `nu = 4`).

`init` Initialization of the EM starts: `"kmeans"` (default; each start runs k-means with different random centers on standardized, mean-imputed data) or `"random"` (random soft partitions, as in version 1.0.0). Random soft partitions start from almost identical profiles, so they need several `n_starts` and can end in an empty profile; `"kmeans"` is recommended.

Value

A list with S3 class `"robust_lpa"` (see `print.robust_lpa` and `summary.robust_lpa`) containing:

engine The estimation engine used.

robust_method `"none"`, `"huber"` or `"t"`.

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

covariances A list of length `G` with the estimated profile covariance matrices (for `robust_method = "t"`: the scale matrices; the covariance of a `t` component is `nu / (nu - 2)` times the scale matrix for `nu > 2`).

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

nu The (estimated or fixed) `t` degrees of freedom; `NULL` unless `robust_method = "t"`. Posterior median for the MCMC engine.

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

weights Numeric vector of length `n`: each observation's robustness weight, averaged over profiles with its posterior probabilities; small values flag outliers. Huber weights lie in (0, 1] (1 = full weight). With `robust_method = "t"` they are the E-step weights $(\nu + p_i)/(\nu + \delta_i)$ (p_i observed variables, δ_i squared Mahalanobis distance), which exceed 1 for observations close to the profile mean. All 1 when `robust = FALSE`.

fit A one-row data.frame with `Model`, `Profiles`, `LogLik`, `Parameters`, `AIC`, `BIC`, `SABIC`, `Entropy`, `Min_Size`, `Max_Size` and, for the MCMC engine, `WAIC`. `Parameters` counts every free parameter; only with the EM engine and `lambda > 0` are the mean components shrunk exactly to zero left out.

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

converged, iterations (EM engine) whether the selected start met `tol` before `max_iter`, and how many iterations it used.

mcmc_draws (MCMC engine only) a list with `chains` (the raw per-chain draws, relabeled to a common profile ordering), `n_chains`, `mcmc_iter`, `burnin`, and `nu_estimated`.

mcmc_diagnostics (MCMC engine only) a data.frame with one row per scalar parameter (`Parameter`, `Rhat`, `ESS`).

waic (MCMC engine only) a list with `WAIC`, `lppd` and `p_waic`.

data The numeric matrix actually fit.

call_args A named list of every argument controlling this fit, used to refit the identical specification on new data (e.g. by `bch_robust`'s bootstrap correction).

Robust estimation

`robust_method = "huber"` At every M-step each observation's contribution to a profile's mean and covariance is down-weighted by a Huber weight computed from its squared Mahalanobis distance to that profile's *current* (already robust) estimates, so the down-weighting accumulates across iterations and converges to an iteratively reweighted M-estimator. This is an estimating-equation approach, not a likelihood: the reported `LogLik` (and hence AIC/BIC/SABIC and `blrt_robust`) is the Gaussian mixture log-likelihood evaluated at the robust estimates, which is a useful heuristic but not the maximized objective. Huber-type estimators also have a limited breakdown point: a large, compact cluster of outliers can still mask itself.

`robust_method = "t"` Each profile is a multivariate t distribution (McLachlan & Peel, 1998; Peel & McLachlan, 2000), whose heavier tails automatically down-weight outlying observations through the latent-scale weights `(nu + p) / (nu + d)`. This is a proper likelihood-based model fitted by ECME, so `LogLik`, AIC/BIC, the BLRT and the BCH method are all used as intended; it is also markedly more resistant than Huber weighting to gross outliers. `nu` is estimated by default and counts as one extra parameter.

In the MCMC engine, `"t"` is implemented as an exact sampler (latent Gamma scales; `nu` updated with the latent scales integrated out), whereas `"huber"` remains a heuristic that down-weights sufficient statistics and does not target a well-defined posterior; prefer `"t"` for Bayesian inference.

Missing data

Rows with missing values contribute their exact observed-data likelihood (the marginal density of their observed entries) to the E-step, and the M-step uses the exact EM treatment of incomplete multivariate data: missing entries are replaced by their conditional expectations given the observed ones and the conditional covariance is added to the scatter matrix (Ghahramani & Jordan, 1994; Liu & Rubin, 1995). The resulting estimates are maximum likelihood under missing-at-random (MAR) missingness. The MCMC engine uses the equivalent data-augmentation step (missing entries are drawn from their conditional distribution every sweep). Rows with no observed variables are allowed and are allocated according to the mixing proportions only.

MCMC details

The chains are initialized from a preliminary EM fit (same model and robust method), with the profile means of each chain independently perturbed by half a within-profile standard deviation so that the chains start from dispersed points (as the Gelman-Rubin diagnostic assumes). Priors: Laplace (Bayesian Lasso) on the means, Dirichlet(1, ..., 1) on the mixing proportions, inverse-gamma(1, 1) on variances (models 1, 2 and the scales of models 4-5), inverse-Wishart(p + 1, I) on full covariance matrices (models 3 and 6), a uniform distribution on correlation matrices (models 4-5), and Gamma(2, 0.1) on `nu`. All conditionals are sampled exactly except the correlation matrices of models 4-5, which use random-walk Metropolis steps (step sizes adapted during burn-in only). Label switching is resolved by relabeling every draw to the EM solution (pivotal reordering) with an exact optimal assignment on standardized mean distances. `$mcmc_diagnostics` reports the Gelman-Rubin $\hat{R}$ and the effective sample size of every scalar parameter; `$fit$WAIC` is the widely applicable information criterion (Watanabe, 2010), computed from up to 1000 posterior draws and preferable to the plug-in AIC/BIC for comparing Bayesian fits.

Reproducibility

Every random quantity (EM starts, MCMC chains, bootstrap replicates) is generated from a seed drawn from R's random number stream before any parallel dispatch, so `set.seed(1); robust_lpa(..., cores = 1)` and `set.seed(1); robust_lpa(..., cores = 4)` give identical results on the same machine. Across operating systems, compilers or linear-algebra libraries, results may differ in the last digits (and, when two solutions are almost equally good, in the selected start or the order of the profiles).

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

Ghahramani, Z., & Jordan, M. I. (1994). Supervised learning from incomplete data via an EM approach. *Advances in Neural Information Processing Systems*, 6, 120-127.

Liu, C., & Rubin, D. B. (1995). ML estimation of the t distribution using EM and its extensions, ECM and ECME. *Statistica Sinica*, 5(1), 19-39.

Peel, D., & McLachlan, G. J. (2000). Robust mixture modelling using the t distribution. *Statistics and Computing*, 10(4), 339-348. doi:10.1023/A:1008981510081

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

Watanabe, S. (2010). Asymptotic equivalence of Bayes cross validation and widely applicable information criterion in singular learning theory. *Journal of Machine Learning Research*, 11, 3571-3594.

See Also

`estimate_profiles_robust` to fit and compare many `G` / `model` combinations at once, `blrt_robust` for a bootstrapped likelihood ratio test, `bch_robust` to relate profiles to distal outcomes, and `plot_mcmc_chains` to inspect MCMC chains.

Examples

```
data(neuro_data)
x <- scale(as.matrix(neuro_data[, c("Memory", "Attention", "Executive_Functions",
                                    "RT_Stroop", "RT_TMT")]))
set.seed(1)
fit <- robust_lpa(x, G = 2, model = 6, n_starts = 2, max_iter = 50)
fit           # print.robust_lpa(): concise overview of the fit
summary(fit)  # summary.robust_lpa(): profile means/sizes and fit indices

# Multivariate-t mixture: a likelihood-based robust alternative
fit_t <- robust_lpa(x, G = 2, model = 6, n_starts = 2, max_iter = 50,
                    robust_method = "t")
fit_t$nu
head(sort(fit_t$weights))  # smallest weights = most outlying observations

# Compare with classical (non-robust) estimation
fit_classical <- robust_lpa(x, G = 2, model = 6, n_starts = 2, max_iter = 50,
```

```
                              robust = FALSE)
sapply(fit_t$means, `[`, "RT_Stroop")
sapply(fit_classical$means, `[`, "RT_Stroop")


# MCMC engine: 4 chains (default), with a small mcmc_iter for speed
fit_mcmc <- robust_lpa(x, G = 2, model = 6, engine = "MCMC",
                       robust_method = "t", mcmc_iter = 300, n_chains = 4)
summary(fit_mcmc)  # includes Rhat / ESS ranges and WAIC
```

`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_gmm` *Summarize a Fitted Robust Growth Mixture Model*

Description

Summarize a Fitted Robust Growth Mixture Model

Usage

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

Arguments

`object`	A `robust_gmm` object.
`...`	Currently ignored.

Value

An object of class `"summary.robust_gmm"`: a list with the class trajectories (a data.frame with one row per class and outcome), class sizes, variance components, fit indices, penalty information and, for the MCMC engine, the range of the convergence diagnostics.

See Also

`robust_gmm`

Examples

```
data(neuro_long)
set.seed(1)
fit <- robust_gmm(neuro_long, id = "ID", time = "Year", outcomes = "Memory",
                  G = 2, n_starts = 2)
summary(fit)
```

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, method (a description of the estimation method), converged/iterations (EM 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, including WAIC for the MCMC engine), 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 <- robust_lpa(x, G = 2, model = 2, n_starts = 3, max_iter = 20)
summary(fit)
```

Index

∗ **datasets**
neuro_data, 12
neuro_long, 13

bch_robust, 3, *12*, *14*, *21*, *24*, *28*, *30*
blrt_gmm_robust, 5, *10*, *22*, *24*
blrt_robust, 7, *12*, *29*, *30*

estimate_gmm_robust, *6*, 9, *22*, *24*
estimate_profiles_robust, 10, *12*, *17*, *30*

mahalanobis, *31*

neuro_data, 12
neuro_long, 13

plot_mcmc_chains, 14, *21*, *24*, *30*
plot_robust_gmm, 15, *24*
plot_robust_lpa, 16
print, *18*
print.robust_gmm, 17
print.robust_lpa, 18, *28*, *33*
print.summary.robust_lpa, *33*

robust_gmm, *3*, *6*, *9*, *10*, *13*, *14*, *16–18*, 19, *32*
robust_lpa, *3*, *4*, *7*, *8*, *10*, *11*, *13*, *14*, *17*, *18*, *22*, *23*, 25, *31*, *33*
robust_mean, 31

summary, *33*
summary.robust_gmm, *18*, 32
summary.robust_lpa, *18*, *28*, 33