

Package ‘LUCIDus’

September 2, 2026

Title LUCID with Multiple Omics Data

Version 3.2.1

Description Implements Latent Unknown Clusters By Integrating Multi-omics Data (LUCID; Peng (2019) <[doi:10.1093/bioinformatics/btz667](https://doi.org/10.1093/bioinformatics/btz667)>) for integrative clustering with exposures, multi-omics data, and health outcomes. Supports three integration strategies: early, parallel, and serial. Provides model fitting and tuning, lasso-type regularization for exposure and omics feature selection, handling of missing data, including both sporadic and complete-case patterns, prediction, and g-computation for estimating causal effects of exposures, bootstrap inference for uncertainty estimation, and S3 summary and plot methods. For the multi-omics integration framework, see Jia (2024) <<https://journal.r-project.org/articles/RJ-2024-012/RJ-2024-012.pdf>>. For the missing-data imputation mechanism, see Jia (2024) <[doi:10.1093/bioadv/vbae123](https://doi.org/10.1093/bioadv/vbae123)>.

Depends R (>= 3.6.0)

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.3

LazyData true

Suggests testthat (>= 3.0.0), knitr, rmarkdown, plotly, mix, visdat

Config/testthat/edition 3

VignetteBuilder knitr

Imports mclust, nnet, boot, jsonlite, networkD3, progress, stats, utils, glasso, glmnet, ggplot2, grDevices

URL <https://journal.r-project.org/articles/RJ-2024-012/RJ-2024-012.pdf>,
<https://doi.org/10.1093/bioadv/vbae123>

NeedsCompilation no

Author Qiran Jia [aut, cre] (ORCID: <<https://orcid.org/0000-0002-0790-5967>>),
Yinqi Zhao [aut] (ORCID: <<https://orcid.org/0000-0003-2413-732X>>),
David Conti [ths] (ORCID: <<https://orcid.org/0000-0002-2941-7833>>),
Jesse Goodrich [ctb] (ORCID: <<https://orcid.org/0000-0001-6615-0472>>)

Maintainer Qiran Jia <qiranjia@usc.edu>
Repository CRAN
Date/Publication 2026-09-02 17:10:14 UTC

Contents

analyze_missing_pattern	2
boot_lucid	3
check_imputation_quality	5
check_na	7
estimate_lucid	8
get_cluster_assignment	12
get_selected_G	13
get_selected_Z	14
get_top_omics_features	15
lucid	16
plot.early_lucid	18
plot.lucid_parallel	19
plot.lucid_serial	20
plot_cluster_omic_profile	21
predict_lucid	23
print.sumlucid_early	25
print.sumlucid_parallel	26
print.sumlucid_serial	27
safe_impute	27
simulated_HELIX_data	28
sim_data	29
summary.lucid_parallel	30
summary.lucid_serial	30
summary_lucid	31
tune_lucid	33
Index	35

analyze_missing_pattern
<i>Describe the missing-data pattern of an omics matrix</i>

Description

Summarises where missingness sits in an omics layer before a LUCID model is fitted, so that the choice between listwise and sporadic handling can be made from the data rather than assumed. Reports missingness by feature and by subject, flags the features and subjects that are more than half missing, and counts the distinct missingness patterns present.

The number of distinct patterns is the diagnostic that matters most for cost: the observed-data likelihood is evaluated once per pattern, so a matrix with few patterns (largely listwise missingness) is far cheaper to fit than one of the same sparsity spread over many patterns.

Usage

```
analyze_missing_pattern(Z)
```

Arguments

Z An N by M omics matrix, or an object coercible to one by `as.matrix`. Missing values are NA.

Value

A list with components:

col_missingness Proportion missing for each of the M features.

row_missingness Proportion missing for each of the N subjects.

high_miss_cols Integer indices of features more than half missing.

high_miss_rows Integer indices of subjects more than half missing.

n_complete Number of subjects with no missing feature.

n_patterns Number of distinct missingness patterns, counting the complete pattern if any subject is complete.

total_missing Proportion of missing cells over the whole matrix.

See Also

[check_na](#), which classifies subjects into the complete / sporadic / listwise categories the EM algorithm branches on.

Examples

```
Z <- matrix(rnorm(200), nrow = 20)
Z[1:3, 1] <- NA
Z[5, ] <- NA
analyze_missing_pattern(Z)[c("n_complete", "n_patterns", "total_missing")]
```

Description

Generate R bootstrap replicates of LUCID parameters and derive confidence interval (CI) based on bootstrap. Bootstrap replicates are generated by nonparametric resampling, implemented with the ordinary method of `boot::boot`. Supports `lucid_model = "early"`, `lucid_model = "parallel"`, and `lucid_model = "serial"`.

Usage

```
boot_lucid(
  G,
  Z,
  Y,
  lucid_model = NULL,
  CoG = NULL,
  CoY = NULL,
  model,
  conf = 0.95,
  R = 100,
  verbose = FALSE,
  min_valid = 2L
)
```

Arguments

G	Exposures, a numeric vector, matrix, or data frame. Categorical variable should be transformed into dummy variables. If a matrix or data frame, rows represent observations and columns correspond to variables.
Z	Omics data: for LUCID early integration, a numeric matrix/data frame; for LUCID in parallel, a list of numeric matrices/data frames. Rows correspond to observations and columns correspond to variables.
Y	Outcome, a numeric vector. Categorical variable is not allowed. Binary outcome should be coded as 0 and 1.
lucid_model	Optional; "early", "parallel", or "serial". Auto-detected from class(model) when omitted (the normal case), so this rarely needs to be set explicitly – it exists for backward compatibility with scripts written before auto-detection. If supplied, it is cross-checked against model's actual class and an error is raised on a mismatch. Bootstrap inference is implemented for all three model types.
CoG	Optional, covariates to be adjusted for estimating the latent cluster. A numeric vector, matrix or data frame. Categorical variable should be transformed into dummy variables.
CoY	Optional, covariates to be adjusted for estimating the association between latent cluster and the outcome. A numeric vector, matrix or data frame. Categorical variable should be transformed into dummy variables.
model	A LUCID model fitted by estimate_lucid. If the fitted model uses nonzero penalties, boot_lucid will automatically refit a zero-penalty model as fall-back because bootstrap inference is only supported for $Rho_G = Rho_Z_{Mu} = Rho_Z_{Cov} = 0$.
conf	A numeric scalar between 0 and 1 to specify confidence level(s) of the required interval(s).
R	An integer to specify number of bootstrap replicates for LUCID model. If feasible, it is recommended to set $R \geq 1000$.
verbose	A flag indicates whether detailed information is printed in console. Default is FALSE.

min_valid Minimum number of bootstrap replicates that must yield finite estimates before confidence limits can be formed. The default, 2, is the mathematical floor. Replicates that fail are counted and warned about, and a small number of replicates raises a warning that the limits are unstable, but neither suppresses the limits; only fewer than `min_valid` surviving replicates yields NA limits.

Value

A list containing:

beta Bootstrap CI table(s) for G-to-X effects. For `lucid_model = "parallel"`, this is a list by omics layer and includes the multinomial intercept plus exposures in G (not CoG).

mu Bootstrap CI table(s) for cluster-specific means of omics features. For `lucid_model = "parallel"`, this is a list by omics layer.

gamma Bootstrap CI table for X-to-Y parameters.

stage For `lucid_model = "serial"`, a list of stage-wise CI tables (each stage contains beta, mu, and gamma for the final stage only).

bootstrap The boot object returned by `boot::boot`.

Examples

```
# use simulated data (a small subset keeps the example quick)
G <- sim_data$G[1:150, , drop = FALSE]
Z <- sim_data$Z[1:150, , drop = FALSE]
Y_normal <- sim_data$Y_normal[1:150]

# fit lucid model
fit1 <- estimate_lucid(G = G, Z = Z, Y = Y_normal, lucid_model = "early",
family = "normal", K = 2,
seed = 1008, max_itr = 20, max_tot_itr = 50)

# conduct bootstrap resampling (lucid_model is auto-detected from fit1's class)
# a small R keeps the example quick; `conf` sets the CI level (default 0.95)
boot1 <- suppressWarnings(
  boot_lucid(G = G, Z = Z, Y = Y_normal, model = fit1, R = 3, conf = 0.9)
)
```

check_imputation_quality

Check whether imputed values are distributionally plausible

Description

Compares the values that were filled in against the values that were actually observed, feature by feature, and flags an imputation that has shifted the centre or the spread of the data far enough to distort a subsequent LUCID fit. This is a sanity check on the imputation, not a measure of its accuracy: the true values are unknown, so agreement in distribution is all that can be assessed.

For each feature, the imputed values are compared with the observed ones through a standardised mean difference, $(\bar{x}_{imp} - \bar{x}_{obs})/s_{obs}$, and a spread ratio s_{imp}/s_{obs} . Features with no observed values or no imputed values are skipped. The reported `mean_diff` is the mean *absolute* standardised difference across features, and `sd_ratio` the mean spread ratio. A feature that is constant where observed contributes a `sd_ratio` of 1 if its imputed values are also constant, and `Inf` if they are not.

The imputation is declared invalid when `mean_diff` exceeds 2 (the filled values sit more than two observed standard deviations from the observed centre), or when `sd_ratio` falls outside `[0.3, 3]` – below that range indicates the near-constant imputation that mean-filling produces at high missingness, which biases cluster covariances towards singularity.

Usage

```
check_imputation_quality(original, imputed)
```

Arguments

<code>original</code>	The data matrix before imputation, containing NA.
<code>imputed</code>	The same matrix after imputation, with identical dimensions and column order. A dimension mismatch is a warning, not an error, and returns <code>is_valid = FALSE</code> .

Value

A list with components:

`is_valid` TRUE if neither threshold was breached and at least one feature could be compared.
`mean_diff` Mean absolute standardised mean difference across comparable features, or NA if none.
`sd_ratio` Mean ratio of imputed to observed standard deviation, or NA if none.
`warning` NULL when valid; otherwise a string naming the thresholds that were breached.

See Also

[safe_impute](#) for the imputations this is meant to check.

Examples

```
Z <- matrix(rnorm(200), nrow = 20)
Z_na <- Z; Z_na[1:5, 1] <- NA
check_imputation_quality(Z_na, safe_impute(Z_na, method = "mean"))
```

check_na

*Classify each subject's omics missingness pattern***Description**

Assigns every subject to one of the three missingness categories the LUCID EM algorithm branches on, and reports whether imputation is required at all. The categories follow the incomplete-omics extension of LUCID:

complete (code 1) Every feature observed. Contributes the ordinary complete-data term to the likelihood.

sporadic (code 2) Some but not all features observed. These are the subjects that require imputation: the missing coordinates are integrated out against the fitted cluster model in the I-step.

listwise (code 3) No feature observed for this layer. The omics term drops out of that subject's likelihood entirely, so the subject still informs the exposure and outcome models but needs no imputation.

The distinction matters for cost as well as correctness: `impute_flag` is TRUE only when at least one *sporadic* subject exists, and a dataset whose missingness is purely listwise is fitted without any imputation step.

A warning is issued for any feature more than half missing, per layer.

Usage

```
check_na(Z, lucid_model = c("early", "parallel"))
```

Arguments

<code>Z</code>	For <code>lucid_model = "early"</code> , an N by M omics matrix. For <code>"parallel"</code> , a list of such matrices, one per layer, all with the same number of rows; anything else is an error.
<code>lucid_model</code>	Either <code>"early"</code> or <code>"parallel"</code> . A serial fit calls this once per stage rather than passing <code>"serial"</code> here.

Value

For `"early"`, a list with `index` (an N by M logical matrix that is TRUE where observed), `indicator_na` (the length-N vector of codes 1, 2, 3 above), `impute_flag` (a single logical), and `missing_analysis` (the [analyze_missing_pattern](#) result).

For `"parallel"`, `index`, `indicator_na` and `layer_analyses` are lists with one element per layer, `impute_flag` is a logical vector over layers, and `cross_layer_summary` adds `n_layers`, `n_observations`, `features_per_layer`, `missing_pattern_counts` (a table of the joint across-layer pattern, so that subjects missing an entire layer can be distinguished from those missing scattered features in several) and `total_missing_prop`, the proportion of missing *cells* pooled over layers rather than the mean of per-layer rates, which would weight a one-feature layer as heavily as a fifty-feature one.

See Also

[analyze_missing_pattern](#) for the per-layer detail.

Examples

```
Z <- matrix(rnorm(200), nrow = 20)
Z[1:2, 1] <- NA # sporadic
Z[20, ] <- NA # listwise
table(check_na(Z, lucid_model = "early")$indicator_na)
```

estimate_lucid

Fit LUCID models with one or multiple omics layers

Description

EM algorithm to estimate LUCID with one or multiple omics layers

Usage

```
estimate_lucid(
  lucid_model = c("early", "parallel", "serial"),
  G,
  Z,
  Y,
  CoG = NULL,
  CoY = NULL,
  K,
  init_omic.data.model = "EEV",
  useY = TRUE,
  tol = 0.001,
  max_itr = 1000,
  max_tot_itr = 10000,
  Rho_G = 0,
  Rho_Z_Mu = 0,
  Rho_Z_Cov = 0,
  family = c("normal", "binary"),
  seed = 123,
  init_impute = c("lod", "mix"),
  init_par = c("mclust", "random"),
  verbose = FALSE,
  n_starts = 1L
)
```

Arguments

lucid_model Specifying LUCID model, "early" for early integration, "parallel" for lucid in parallel, "serial" for lucid in serial

G	an N by P matrix representing exposures
Z	Omics data, if "early", an N by M matrix; If "parallel", a list, each element i is a matrix with N rows and P_i features; If "serial", a list, each element i is a matrix with N rows and p_i features or a list with two or more matrices with N rows and a certain number of features
Y	a length N vector
CoG	an N by V matrix representing covariates to be adjusted for G -> X
CoY	an N by K matrix representing covariates to be adjusted for X -> Y
K	Number of latent clusters. If "early", an integer greater or equal to 2; If "parallel", an integer vector, same length as Z, with each element being an integer greater or equal to 2; If "serial", a list, each element is either an integer like that for "early" or an list of integers like that for "parallel", same length as Z
init_omic.data.model	a vector of strings specifies the geometric model of omics data. If NULL, See more in ?mclust::mclustModelNames
useY	logical, if TRUE, EM algorithm fits a supervised LUCID; otherwise unsupervised LUCID.
tol	stopping criterion for the EM algorithm
max_itr	Maximum iterations of the EM algorithm. If the EM algorithm iterates more than max_itr without converging, the EM algorithm is forced to stop.
max_tot_itr	Max number of total iterations for estimate_lucid function. estimate_lucid may conduct EM algorithm for multiple times if the algorithm fails to converge.
Rho_G	A scalar. This parameter is the LASSO penalty to regularize exposure coefficients in the G-to-X model. CoG adjustment covariates are included unpenalized. If user wants to tune the penalty, use the wrapper function lucid. Penalty tuning is supported for "early" and "parallel". For "serial", only scalar penalty inputs are supported.
Rho_Z_Mu	A scalar. This parameter is the LASSO penalty to regularize cluster-specific means for omics data (Z). If user wants to tune the penalty, use the wrapper function lucid. Penalty tuning is supported for "early" and "parallel". For "serial", only scalar penalty inputs are supported.
Rho_Z_Cov	A scalar. This parameter is the graphical LASSO penalty to estimate sparse cluster-specific variance-covariance matrices for omics data (Z). If user wants to tune the penalty, use the wrapper function lucid. Penalty tuning is supported for "early" and "parallel". For "serial", only scalar penalty inputs are supported.
family	The distribution of the outcome
seed	Random seed to initialize the EM algorithm
init_impute	Method to initialize the imputation of missing values in LUCID. lod (the default) initializes the imputation via replacing missing values by LOD / sqrt(2), where LOD is determined by the minimum of each variable in omics data; mix uses mclust::imputeData to implement EM Algorithm for Unrestricted General Location Model via the mix package to impute the missing values in omics data. mix is archived on CRAN and must be installed manually (e.g. from the CRAN Archive) to use this option; a request for init_impute = "mix" without mix installed raises an informative error.

<code>init_par</code>	For "early", an interface to initialize EM algorithm, if <code>mclust</code> , initiate the parameters using the <code>mclust</code> package, if <code>random</code> , initiate the parameters by drawing from a uniform distribution; For "parallel", <code>mclust</code> is the default for quick convergence; For "serial", each sub-model follows the above depending on it is a "early" or "parallel"
<code>verbose</code>	Logging level for fitting progress. If <code>FALSE</code> , concise start/finish status lines are printed. If <code>TRUE</code> , detailed iteration-level traces (including log-likelihood updates) are printed.
<code>n_starts</code>	Number of independent random starts for the EM algorithm (default 1). The EM algorithm converges only to a local optimum, so with <code>n_starts > 1</code> the model is fitted from that many starting points and the fit with the highest observed-data log-likelihood is returned. Per-start log-likelihoods are recorded in <code>em_control\$start_loglik</code> , which is worth inspecting: a wide spread indicates the likelihood surface is multi-modal and that a single start would have been unreliable.

Value

An object of class `early_lucid`, `lucid_parallel` or `lucid_serial` according to `lucid_model`. All three are lists; the components common to every fit are:

res_Beta Estimates of the exposure-to-cluster ($G \rightarrow X$) association. For "early", a K by $(1 + P + V)$ matrix of multinomial logistic coefficients, cluster 1 as reference. For "parallel" and "serial", a list holding the fitted object and the coefficient matrix per layer or stage.

res_Mu Cluster-specific omics means (the μ of $X \rightarrow Z$). A K by M matrix for "early"; a list by layer or stage otherwise.

res_Sigma Cluster-specific omics variance-covariance matrices (the σ of $X \rightarrow Z$). A list of K matrices for "early"; a list by layer or stage otherwise.

res_Gamma Estimates of the cluster-to-outcome ($X \rightarrow Y$) association, holding beta (absolute cluster levels), the reference-coded `cluster_effect` contrasts printed by `summary()`, any covariate coefficients, the residual sigma for a normal outcome, and the parameterization used.

inclusion.p Posterior probability of cluster membership for each observation, r_{ij} of Eq 3. An N by K matrix for "early"; a list by layer or stage otherwise.

K Number of latent clusters: an integer for "early", a list of integers for "parallel" and "serial".

var.names Names of the G , Z and Y variables, as `list(Gnames, Znames, Ynames)`.

init_omic.data.model The `mclust` geometric model used for the omics covariances.

family Outcome distribution, "normal" or "binary".

useY Whether the outcome was used in fitting, i.e. whether the model is supervised.

Z The omics data. For "early" and "parallel" this is the data the model was fitted to, so sporadically missing cells hold their imputed values and listwise-missing rows remain NA. For "serial" it is the omics data *as supplied*, still containing every missing value: imputation happens inside each stage, so the imputed omics for stage i are in `submodel[[i]]$Z`.

init_impute The imputation method used to initialize missing omics values.

init_par The parameter-initialization method used.

Rho The penalties actually applied, as `list(Rho_G, Rho_Z_Mu, Rho_Z_Cov)`.

missing_summary How much omics data was missing and in what pattern, using the taxonomy of the incomplete-omics extension: `complete_rows`, `listwise_rows` (a whole omics layer missing for that subject) and `sporadic_rows` (some features missing), the corresponding proportions, and cell-level counts `total_missing_cells`, `sporadic_missing_cells` and `prop_total_missing_cells`. A list by layer for "parallel"; for "serial", `n_stages` and a per-stage breakdown.

em_control The stopping controls used (`tol`, `max_itr`, `max_tot_itr`), which bootstrap refits reuse, together with convergence diagnostics. For "early" and "parallel" these are `converged` (FALSE, with a warning, if the fit exhausted `max_itr` without meeting `tol`), `n_iter`, `n_restart`, `loglik_trace`, `n_starts` and `n_starts_ok`. A "serial" fit runs no EM loop of its own, so it reports `converged` (TRUE only if every sub-model converged), `n_iter` (the total across sub-models), the per-sub-model `submodel_converged` and `submodel_n_iter`, and `submodel_loglik_trace` (each sub-model's own `loglik_trace`, by stage); the full diagnostics for a stage are in `submodel[[i]]$em_control`. For all three model types, the log-likelihood trace is checked for monotonicity as it is recorded: a decrease beyond a small, majorization-step-aware slack triggers a `warning()` naming the iteration (or stage) and the two values, since that indicates a numerical problem rather than expected EM behaviour.

The remaining components appear only for some model types:

likelihood The observed-data log-likelihood at the returned estimates, present for all three model types. For "early" and "parallel" this is the single joint log-likelihood from the EM fit. A "serial" fit runs no joint EM loop – it is a sequence of conditionally fitted stages – so its likelihood is instead the *sum* of each stage's own log-likelihood (matching `cal_loglik_serial()`); the individual per-stage values are in `submodel[[i]]$likelihood`.

select Feature-selection indicators, present for all three model types. `select$selectG` and `select$selectZ` are logical vectors of retained exposures and omics features. For "parallel", `select$selectG` is the exposure-wise union across layers (selected in at least one layer), `select$selectG_layer` holds the per-layer exposure selection, and `select$selectZ` holds the per-layer omics selection (a list with at least one layer's selection reported). For "serial", `select` is **stage 1's own selection only** (`submodel[[1]]$select`) – stage 1 is the only stage whose G is the user's actual exposures; every later stage's G is the previous stage's posterior cluster-membership probabilities, so `Rho_G` is always 0 there and that stage's `selectG` is not a meaningful exposure-selection result (its `selectZ` still is). The complete per-stage selection record remains available at `submodel[[i]]$select` for every `i`.

N Number of observations. Present for "parallel" and "serial"; for "early", use `nrow(fit$inclusion.p)`.

z The E-step responsibilities over the joint cluster configuration across layers, before they are marginalised into `inclusion.p`. Present for "parallel" only – nothing in the package reads this field, and there is no plan to extend it to "serial".

res_Delta Estimates of the between-stage cluster transition associations, one element per transition (length `n_stages - 1`). Element `i` is the coefficient object of stage `i + 1` fitted with stage `i`'s cluster assignment in place of the exposures, so it has the same structure as `res_Beta` for that stage. Present for "serial" only.

submodel The fitted sub-models, one per stage, each itself an `early_lucid` or `lucid_parallel` object. Present for "serial" only.

Examples

```

i <- 1008
set.seed(i)
G <- matrix(rnorm(500), nrow = 100)
Z1 <- matrix(rnorm(1000), nrow = 100)
Z2 <- matrix(rnorm(1000), nrow = 100)
Z3 <- matrix(rnorm(1000), nrow = 100)
Z <- list(Z1 = Z1, Z2 = Z2, Z3 = Z3)
Y <- rnorm(100)
CoY <- matrix(rnorm(200), nrow = 100)
CoG <- matrix(rnorm(200), nrow = 100)
fit1 <- estimate_lucid(G = G, Z = Z, Y = Y, K = list(2, 2, 2),
  lucid_model = "serial",
  family = "normal",
  seed = i,
  CoG = CoG, CoY = CoY,
  useY = TRUE,
  max_itr = 20, max_tot_itr = 50)

```

get_cluster_assignment

Extract the hard cluster assignment from a fitted LUCID model

Description

Computes the maximum-a-posteriori cluster label for every observation directly from `model$inclusion.p`, with no need to re-run prediction or supply `G/Z/Y` again. Shaped exactly like `predict_lucid()`'s own `pred.x`: a numeric vector for early, a list by layer for parallel, and a list by stage (recursively shaped) for serial. Labels run 1, ..., K, matching Eq 21 and the row names used by `summary()`.

Usage

```
get_cluster_assignment(model)
```

Arguments

`model` A fitted `early_lucid`, `lucid_parallel`, or `lucid_serial` object.

Value

A numeric vector (early), a named list by layer (parallel), or a named list by stage (serial, each element shaped like the above depending on that stage's own type).

Examples

```

idx <- 1:200
G <- sim_data$G[idx, ]
Z <- sim_data$Z[idx, ]
Y_normal <- sim_data$Y_normal[idx, ]

```

```
fit <- estimate_lucid(G = G, Z = Z, Y = Y_normal, lucid_model = "early",
                    family = "normal", K = 2,
                    max_itr = 10, max_tot_itr = 30)
table(get_cluster_assignment(fit))
```

get_selected_G	<i>Extract selected (retained) exposures from a fitted LUCID model</i>
----------------	--

Description

Reads `model$select$selectG` (or, for a serial model, stage 1's own selection), auto-detecting the model type from `class(model)`. Stage 1 is the only stage in a serial chain whose "G" is the cohort's actual exposures – from stage 2 on, "G" is the previous stage's posterior cluster probabilities, so there is nothing there for an exposure-selection result to describe (see `estimate_lucid`'s `@return` for the full rationale). This is exactly the same field `estimate_lucid()` already returns; this function only adds the class-based dispatch and the union/per-layer/per-stage bookkeeping so a caller doesn't have to.

Usage

```
get_selected_G(model, layer = NULL)
```

Arguments

<code>model</code>	A fitted <code>early_lucid</code> , <code>lucid_parallel</code> , or <code>lucid_serial</code> object.
<code>layer</code>	For a parallel model only: which layer's own exposure selection to return (an integer index or a layer name). If <code>NULL</code> (the default), returns the union across layers (selected in at least one layer) – ignored for early and serial.

Value

A named logical vector, one entry per exposure, `TRUE` where retained.

Examples

```
idx <- 1:200
G <- sim_data$G[idx, ]
Z <- sim_data$Z[idx, ]
Y_normal <- sim_data$Y_normal[idx, ]
fit <- estimate_lucid(G = G, Z = Z, Y = Y_normal, lucid_model = "early",
                    family = "normal", K = 2, Rho_G = 0.1,
                    max_itr = 10, max_tot_itr = 30)
get_selected_G(fit)
```

get_selected_Z	<i>Extract selected (retained) omics features from a fitted LUCID model</i>
----------------	---

Description

Reads `model$select$selectZ`, auto-detecting the model type from `class(model)` and collapsing any per-cluster selection matrix (a parallel-model layer's `selectZ` can be a $K \times M$ matrix rather than a plain vector) to one logical value per feature via an internal helper. Unlike exposure selection, every stage of a serial model has a meaningful omics selection, so this returns a per-stage breakdown rather than one stage's alone.

Usage

```
get_selected_Z(model, layer = NULL, stage = NULL)
```

Arguments

<code>model</code>	A fitted <code>early_lucid</code> , <code>lucid_parallel</code> , or <code>lucid_serial</code> object.
<code>layer</code>	For a parallel model (or a serial stage that is itself a parallel sub-model): which layer's own omics selection to return. If <code>NULL</code> (the default), returns a named list, one entry per layer.
<code>stage</code>	For a serial model only: which stage's own omics selection to return. If <code>NULL</code> (the default), returns a named list, one entry per stage (each shaped like this function's early/parallel return, depending on that stage's own type).

Value

A named logical vector (early; parallel with `layer` given), a named list of logical vectors (parallel with `layer = NULL`), or a named list of per-stage results (serial).

Examples

```
idx <- 1:200
G <- sim_data$G[idx, ]
Z <- sim_data$Z[idx, ]
Y_normal <- sim_data$Y_normal[idx, ]
fit <- estimate_lucid(G = G, Z = Z, Y = Y_normal, lucid_model = "early",
  family = "normal", K = 2, Rho_Z_Mu = 5,
  max_itr = 10, max_tot_itr = 30)
get_selected_Z(fit)
```

get_top_omics_features

Extract the top-N most important omics features from a fitted LUCID model

Description

Reuses `plot_cluster_omic_profile()`'s own feature-ranking criterion (see its documentation for what "separation" means) rather than introducing a second ranking rule: this is the same score `plot_cluster_omic_profile()` sorts features by, just returned as data instead of a plot. One panel is produced per relevant unit – the whole omics matrix for early, one per layer for parallel, and for serial one per stage (or one per layer within a stage that is itself parallel).

Usage

```
get_top_omics_features(
  model,
  top_n = 10,
  importance = c("separation", "range", "sd")
)
```

Arguments

<code>model</code>	A fitted <code>early_lucid</code> , <code>lucid_parallel</code> , or <code>lucid_serial</code> object.
<code>top_n</code>	Number of top features to return per panel (default 10). If a panel has fewer features than <code>top_n</code> , all of them are returned.
<code>importance</code>	Ranking criterion: "separation" (between-cluster spread over within-cluster SD, the default), "range", or "sd" of the cluster means – identical meaning to <code>plot_cluster_omic_profile()</code> own importance argument.

Value

A named list, one entry per panel (layer/stage), each a named numeric vector of the top `top_n` features by importance, sorted descending.

Examples

```
idx <- 1:200
G <- sim_data$G[idx, ]
Z <- sim_data$Z[idx, ]
Y_normal <- sim_data$Y_normal[idx, ]
fit <- estimate_lucid(G = G, Z = Z, Y = Y_normal, lucid_model = "early",
  family = "normal", K = 2,
  max_itr = 10, max_tot_itr = 30)
get_top_omics_features(fit, top_n = 3)
```

lucid	<i>Fit a lucid model for integrated analysis on exposure, outcome and multi-omics data, allowing for tuning</i>
-------	---

Description

Fit a lucid model for integrated analysis on exposure, outcome and multi-omics data, allowing for tuning

Usage

```
lucid(
  G,
  Z,
  Y,
  CoG = NULL,
  CoY = NULL,
  family = c("normal", "binary"),
  K = 2,
  lucid_model = c("early", "parallel", "serial"),
  Rho_G = 0,
  Rho_Z_Mu = 0,
  Rho_Z_Cov = 0,
  verbose_tune = FALSE,
  ...
)
```

Arguments

G	Exposures, a numeric vector, matrix, or data frame. Categorical variable should be transformed into dummy variables. If a matrix or data frame, rows represent observations and columns correspond to variables.
Z	Omics data. If "early", an N by M matrix. If "parallel", a list, each element i is a matrix with N rows and P_i features. If "serial", a list, each element i is either a matrix with N rows and p_i features, or a list with two or more matrices with N rows.
Y	Outcome, a numeric vector. Categorical variable is not allowed. Binary outcome should be coded as 0 and 1.
CoG	Optional, covariates to be adjusted for estimating the latent cluster. A numeric vector, matrix or data frame. Categorical variable should be transformed into dummy variables.
CoY	Optional, covariates to be adjusted for estimating the association between latent cluster and the outcome. A numeric vector, matrix or data frame. Categorical variable should be transformed into dummy variables.
family	Distribution of outcome. For continuous outcome, use "normal"; for binary outcome, use "binary". Default is "normal".

K	Number of latent clusters to be tuned. For lucid_model = "early", number of latent clusters (should be greater or equal than 2). Either an integer or a vector of integer. If K is a vector, model selection on K is performed. For lucid_model = "parallel", a list with vectors of integers or just integers, same length as Z, if the element itself is a vector, model selection on K is performed; For lucid_model = "serial", a list, each element is either an integer or an list of integers, same length as Z, if the smallest element (integer) itself is a vector, model selection on K is performed
lucid_model	Specifying LUCID model, "early" for early integration, "parallel" for lucid in parallel, "serial" for lucid in serial
Rho_G	A scalar or a vector. This parameter is the LASSO penalty to regularize exposure coefficients in the G-to-X model; CoG covariates are not penalized. If it is a vector, lucid will call tune_lucid to conduct model selection and variable selection. User can try penalties from 0 to 1. Penalty tuning is supported for "early" and "parallel". For "serial", only scalar penalty inputs are supported.
Rho_Z_Mu	A scalar or a vector. This parameter is the LASSO penalty to regularize cluster-specific means for omics data (Z). If it is a vector, lucid will call tune_lucid to conduct model selection and variable selection. User can try penalties from 1 to 100. Penalty tuning is supported for "early" and "parallel". For "serial", only scalar penalty inputs are supported.
Rho_Z_Cov	A scalar or a vector. This parameter is the graphical LASSO penalty to estimate sparse cluster-specific variance-covariance matrices for omics data (Z). If it is a vector, lucid will call tune_lucid to conduct model selection and variable selection. User can try penalties from 0 to 1. Penalty tuning is supported for "early" and "parallel". For "serial", only scalar penalty inputs are supported.
verbose_tune	A flag to print details of tuning process.
...	Other parameters passed to estimate_lucid

Value

A fitted LUCID model of class `early_lucid`, `lucid_parallel` or `lucid_serial` – the candidate with the lowest BIC when K or any penalty is given as a vector, and otherwise simply the single fitted model. The components are those documented in [estimate_lucid](#), with one addition:

selection Present for "early" only, and only when a non-zero penalty selected a strict subset of the input variables. Records what the tuned penalties dropped, as `selectG` and `selectZ` (logical vectors over the *original* inputs), the corresponding `Gnames` and `Znames`, and the tuned `Rho` that produced the selection. Because `lucid` refits the selected model unpenalized on the retained features, the model's own `select` component describes the refit dimensions and indexes `res_Beta` and `res_Mu`; use `selection` to see what was dropped from the original data.

Examples

```
# LUCID early integration (quick smoke example)
G <- sim_data$G[1:80, , drop = FALSE]
Z <- sim_data$Z[1:80, , drop = FALSE]
Y <- sim_data$Y_normal[1:80]
```

```

fit_early <- lucid(
  G = G, Z = Z, Y = Y,
  lucid_model = "early", family = "normal", K = 2,
  max_itr = 30, max_tot_itr = 60, seed = 1008
)

# LUCID in parallel (two layers)
i <- 1008
set.seed(i)
G <- matrix(rnorm(240), nrow = 80)
Z1 <- matrix(rnorm(320), nrow = 80)
Z2 <- matrix(rnorm(320), nrow = 80)
Z <- list(Z1 = Z1, Z2 = Z2)
CoY <- matrix(rnorm(160), nrow = 80)
CoG <- matrix(rnorm(160), nrow = 80)
Y <- rnorm(80)
fit_parallel <- lucid(
  G = G, Z = Z, Y = Y, K = list(2, 2),
  CoG = CoG, CoY = CoY, lucid_model = "parallel",
  family = "normal", seed = i,
  max_itr = 30, max_tot_itr = 60
)

```

plot.early_lucid	<i>Visualize an early-integration LUCID model through a Sankey diagram</i>
------------------	--

Description

Draws the fitted model as a Sankey diagram: exposures flow into the latent clusters, and the clusters flow on into the omics features and the outcome. Each node is either a variable (exposure, omics or outcome) or a latent cluster, and its colour indicates which. Each link is an estimated association: its width is the magnitude of the effect and its colour the sign, so the diagram shows at a glance which exposures drive which cluster and how that cluster differs in the omics layer.

Only exposures and omics features retained by the model are drawn, so a penalized fit yields a correspondingly sparser diagram.

Usage

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

Arguments

x	A LUCID model fitted by estimate_lucid or lucid , of class early_lucid.
...	Appearance options, all optional:
	G_color Colour of the exposure nodes (default "dimgray").

X_color Colour of the latent-cluster nodes (default "#eb8c30").
Z_color Colour of the omics nodes (default "#2fa4da").
Y_color Colour of the outcome node (default "#afa58e").
pos_link_color Colour of links with a positive effect (default "#67928b").
neg_link_color Colour of links with a negative effect (default "#d1e5eb").
fontsize Node label size in points (default 7).

Value

An HTML widget created by [sankeyNetwork](#). It renders when printed, in the RStudio viewer or a browser, and can be written to a standalone file with `htmlwidgets::saveWidget`.

Model types

Implemented for early integration only. A `lucid_parallel` or `lucid_serial` fit has a registered method ([plot.lucid_parallel](#)/[plot.lucid_serial](#)), but it raises an error: the parallel and serial diagrams are still under development.

Examples

```
# prepare data (a small subset keeps the example quick)
G <- sim_data$G[1:150, ]
Z <- sim_data$Z[1:150, ]
Y_normal <- sim_data$Y_normal[1:150, , drop = FALSE]

# plot lucid model
fit1 <- estimate_lucid(G = G, Z = Z, Y = Y_normal, lucid_model = "early",
  CoY = NULL, family = "normal", K = 2, seed = 1008,
  max_itr = 20, max_tot_itr = 50)
plot(fit1)

# change node color
plot(fit1, G_color = "yellow")
plot(fit1, Z_color = "red")

# change link color
plot(fit1, pos_link_color = "red", neg_link_color = "green")
```

<code>plot.lucid_parallel</code>	<i>Sankey diagram for a parallel-integration LUCID model (not yet implemented)</i>
----------------------------------	--

Description

Sankey diagram for a parallel-integration LUCID model (not yet implemented)

Usage

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

Arguments

x A LUCID model fitted with lucid_model = "parallel".

... Accepted for consistency with `plot.early_lucid`'s appearance options, but unused.

Value

Does not return: always raises an error.

plot.lucid_serial	<i>Sankey diagram for a serial-integration LUCID model (not yet implemented)</i>
-------------------	--

Description

Sankey diagram for a serial-integration LUCID model (not yet implemented)

Usage

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

Arguments

x A LUCID model fitted with lucid_model = "serial".

... Accepted for consistency with `plot.early_lucid`'s appearance options, but unused.

Value

Does not return: always raises an error.

plot_cluster_omic_profile

Plot per-cluster omics profiles

Description

Shows which omics features distinguish the latent clusters, and in which direction, for an early, parallel or serial fit. A parallel or serial model produces one plot per omics layer, returned as a named list, so no figure has to accommodate every layer at once.

The default rendering is a cluster-by-feature heatmap in the style used for single-cell cluster markers: features on the vertical axis ordered by how strongly they separate the clusters, clusters across the top, and fill showing how high or low each cluster sits for that feature.

Usage

```
plot_cluster_omic_profile(
  x,
  type = c("heatmap", "bar"),
  top_n = 10,
  importance = c("separation", "range", "sd"),
  layer_names = NULL,
  layer_colors = NULL,
  cluster_labels = NULL,
  scale = TRUE
)
```

Arguments

x	A fitted LUCID model: early_lucid, lucid_parallel or lucid_serial.
type	"heatmap" (default) or "bar".
top_n	Number of features to show per panel, default 10. A layer with fewer features than this shows all of them rather than erroring.
importance	Ranking measure; see <i>Which features are shown</i> .
layer_names	Character vector of layer or stage names, used as plot subtitles. Defaults to the names of the omics list the model was fitted to, falling back to "Layer 1", "Layer 2" and so on.
layer_colors	One colour per layer. The heatmap uses it as the high end of its diverging scale, and the bar plot as the darkest of a sequential ramp across clusters – which is what lets the bar plot handle any number of clusters without recycling colours.
cluster_labels	Optional labels for the clusters, defaulting to "Cluster 1", "Cluster 2" and so on. Must have one entry per cluster.
scale	If TRUE (default) the fill is a per-feature z-score across clusters; if FALSE it is the fitted cluster mean. At K = 2 the default instead centres without rescaling, because a two-value z-score is degenerate – see <i>What the colour means</i> .

Value

A named list of ggplot objects, one per omics layer – length one for an early fit. Each carries the data it drew as the attribute "profile_data": a data frame of feature, cluster, value (what is plotted), mean (the fitted cluster mean), sd (within-cluster standard deviation) and score (the importance value the ranking used), so the ranking can be extracted without recomputing it. That data frame also carries a "centred" attribute recording whether the fill was centred rather than z-scored, which is what happens by default at K = 2.

Which features are shown

Only the top_n most discriminating features per panel are drawn, ranked by importance:

"separation" **(default)** The spread of the cluster means divided by the typical within-cluster spread, $\text{sd}_k(\mu_{kj})/\sqrt{\sigma_{kj}^2}$. This is the only option that accounts for noise: a feature whose cluster means differ by two units is uninformative if its within-cluster standard deviation is also two. It is scale-free, so features on different scales are comparable, and it is defined the same way for any number of clusters.

"range" $\max_k \mu_{kj} - \min_k \mu_{kj}$, in the data's own units. The most directly interpretable, and the quantity the package's own feature selection thresholds.

"sd" The standard deviation of the cluster means, without standardizing by within-cluster spread.

A feature the model deselected has identical means across clusters and so scores zero under all three, sorting last.

What the colour means

With scale = TRUE (the default) the fill is a z-score computed across clusters within each feature, so the palette is centred at zero and a feature's own baseline does not dominate. With scale = FALSE the fill is the fitted cluster mean on the data's original scale.

Two clusters are a special case. A z-score over two values is always exactly $\pm 1/\sqrt{2}$, so at K = 2 every tile would saturate and the fill would carry the sign of the difference but nothing about its size. When K = 2 and scale is left at its default, the fill is therefore centred without rescaling – each cluster mean minus that feature's average across clusters – which keeps the palette diverging while restoring magnitude. Passing scale = TRUE explicitly overrides this and gives the degenerate z-score. The legend always names whichever quantity was used, and the returned data records it in the "centred" attribute.

The underlying means and within-cluster standard deviations are always available on the returned object, see *Value*.

Examples

```
# a small subset keeps the example quick
G <- sim_data$G[1:150, , drop = FALSE]
Z <- sim_data$Z[1:150, , drop = FALSE]
Y <- sim_data$Y_normal[1:150]

fit <- estimate_lucid(G = G, Z = Z, Y = Y, lucid_model = "early",
  family = "normal", K = 2, seed = 1008,
```

```

max_itr = 20, max_tot_itr = 50)

p <- plot_cluster_omic_profile(fit)
p[[1]]

# bar rendering, and more features
plot_cluster_omic_profile(fit, type = "bar", top_n = 15)[[1]]

# the ranking behind the figure
head(unique(attr(p[[1]], "profile_data")[, c("feature", "score")]))

```

predict_lucid

*Predict Cluster Assignment and Outcome From a Fitted LUCID Model***Description**

Predict cluster assignment and outcome using new data on G, Z, and optional Y. If `g_computation = TRUE`, prediction uses only the G-to-X path from the fitted model and returns counterfactual-style predictions under modified G. This function can also be used to extract latent cluster assignments when using the training data as input.

Usage

```

predict_lucid(
  model,
  lucid_model = NULL,
  G,
  Z = NULL,
  Y = NULL,
  CoG = NULL,
  CoY = NULL,
  response = TRUE,
  g_computation = FALSE,
  verbose = FALSE
)

```

Arguments

<code>model</code>	A model fitted and returned by estimate_lucid
<code>lucid_model</code>	Optional; "early", "parallel", or "serial". Auto-detected from <code>class(model)</code> when omitted (the normal case), so this rarely needs to be set explicitly – it exists for backward compatibility with scripts written before auto-detection. A serial model must have at least two stages to be predicted; a single-stage serial model is a fully equivalent early or parallel model and should be fitted as one.

G	Exposures, a numeric vector, matrix, or data frame. Categorical variable should be transformed into dummy variables. If a matrix or data frame, rows represent observations and columns correspond to variables.
Z	Omics data, and required for every model type unless <code>g_computation = TRUE</code> . If "early", an N by M matrix. If "parallel", a list, each element i is a matrix with N rows and P_i features. If "serial", a list, each element i is a matrix with N rows and p_i features (or a list with two or more matrices with N rows and a certain number of features). The requirement is not arbitrary: the E-step forms the posterior from the omics likelihood, so with no Z there is nothing to condition on. <code>g_computation = TRUE</code> is a different estimator, not a way around this – it drops the omics and outcome terms and uses the exposure path alone – which is why it is the one mode that accepts <code>Z = NULL</code> .
Y	Outcome, a numeric vector. Categorical variable is not allowed. Binary outcome should be coded as 0 and 1.
CoG	Optional, covariates to be adjusted for estimating the latent cluster. A numeric vector, matrix or data frame. Categorical variable should be transformed into dummy variables.
CoY	Optional, covariates to be adjusted for estimating the association between latent cluster and the outcome. A numeric vector, matrix or data frame. Categorical variable should be transformed into dummy variables.
response	If TRUE, when predicting binary outcomes, class labels (0/1) are returned using a 0.5 threshold. If FALSE, predicted probabilities are returned.
g_computation	If TRUE, prediction uses only information on G, making it the counterfactual mode: hold the fitted model fixed, vary G, and read off what the model implies. It is the only mode in which Z may be omitted, and it is also the only one that returns <code>pred.z</code> . Supplied Z and Y are ignored (with a printed notice) for "early", "parallel", and "serial", so results are unchanged by passing them.
verbose	A flag indicates whether detailed information is printed in console. Default is FALSE. Applies consistently to all three model types (early, parallel, serial).

Value

A list containing:

<code>inclusion.p</code>	Posterior inclusion probabilities for latent clusters (a matrix for "early"; a list by layer for "parallel" and "serial"). Columns are ordered by cluster, matching the row order of the model's <code>res_Mu</code> and <code>res_Beta</code> .
<code>pred.x</code>	Predicted latent-cluster labels (a numeric vector for "early"; a list by layer for "parallel" and "serial"), obtained as the maximum a posteriori column of <code>inclusion.p</code> . Labels run 1, ..., K, agreeing with Eq 21 and with the cluster names used by <code>summary()</code> and the mu and beta row names. Versions before 3.1.0 returned 0, ..., K - 1 here; code that compensated by adding one must drop that adjustment.
<code>pred.y</code>	Predicted outcome values. For binary outcomes, this is class labels when <code>response = TRUE</code> and probabilities when <code>response = FALSE</code> .

pred.z Predicted omics means under g-computation mode (g_computation = TRUE); NULL otherwise.

Supplying Y makes the cluster prediction supervised: the outcome enters the posterior alongside G and Z, as it does during fitting. Omitting it predicts clusters from G and Z alone, which is what is wanted when the outcome is unavailable or must not inform the assignment.

Examples

```
# prepare data (a small subset keeps the example quick)
G <- sim_data$G[1:150, ]
Z <- sim_data$Z[1:150, ]
Y_normal <- sim_data$Y_normal[1:150, , drop = FALSE]

# fit lucid model
fit1 <- estimate_lucid(G = G, Z = Z, Y = Y_normal, lucid_model = "early", K = 2,
                      family = "normal", max_itr = 20, max_tot_itr = 50)

# prediction on training set (lucid_model is auto-detected from fit1's class)
pred1 <- predict_lucid(model = fit1, G = G, Z = Z, Y = Y_normal)
pred2 <- predict_lucid(model = fit1, G = G, Z = Z)

# g-computation style prediction using only G
pred_g <- predict_lucid(model = fit1, G = G, Z = NULL, g_computation = TRUE)
```

```
print.sumlucid_early    Print the output of LUCID in a nicer table
```

Description

Print the output of LUCID in a nicer table

Usage

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

Arguments

x An object returned by summary

... Other parameters to be passed to print.sumlucid_serial

Value

Prints a structured model summary, including model specification, missing-data profile, feature-selection overview, model fit statistics, regularization settings, and detailed parameter estimates. If boot.se is provided in summary(), bootstrap CI tables are shown for sections (1) Y, (2) Z, and (3) E.

Examples

```
# use simulated data (a small subset keeps the example quick)
G <- sim_data$G[1:150, , drop = FALSE]
Z <- sim_data$Z[1:150, , drop = FALSE]
Y_normal <- sim_data$Y_normal[1:150]

# fit lucid model
fit1 <- estimate_lucid(G = G, Z = Z, Y = Y_normal, lucid_model = "early", family = "normal", K = 2,
seed = 1008, max_itr = 20, max_tot_itr = 50)

# conduct bootstrap resampling
boot1 <- suppressWarnings(
  boot_lucid(G = G, Z = Z, Y = Y_normal, lucid_model = "early", model = fit1, R = 2)
)

# print the summary of the lucid model in a table
temp <- summary(fit1)
print(temp)
```

```
print.sumlucid_parallel
```

Print the output of LUCID in a nicer table

Description

Print the output of LUCID in a nicer table

Usage

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

Arguments

x	An object returned by summary
...	Other parameters to be passed to print.sumlucid_parallel

Value

x, invisibly. Called for its side effect: printing a structured parallel-model summary – per-layer missing-data profile, overall and per-layer feature-selection overview, model fit statistics, regularization settings, and parameter estimates for sections (1) Y, (2) Z and (3) E. If boot.se was provided to summary(), bootstrap confidence limits are shown for each of those three sections.

```
print.sumlucid_serial
```

Print the output of LUCID in a nicer table

Description

Print the output of LUCID in a nicer table

Usage

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

Arguments

`x` An object returned by `summary`

`...` Other parameters to be passed to `print.sumlucid_serial`

Value

`x`, invisibly. Called for its side effect: printing the serial model's summary stage by stage – per-stage missing-data profile, feature-selection overview, model fit statistics and parameter estimates, with the outcome section attached to the final stage. If `boot.se` was supplied to `summary()`, bootstrap confidence limits are shown alongside the estimates.

```
safe_impute
```

Single-value imputation that tolerates fully missing columns

Description

Fills missing entries feature by feature with a single summary of that feature's observed values. This is a starting point for the EM algorithm, not a substitute for it: LUCID's own E-step imputes missing omics values under the fitted cluster model (an internal EM detail, not part of the public API), and single-value filling here only has to be finite and roughly located.

Unlike a bare `mean(x, na.rm = TRUE)`, a feature with *no* observed value does not yield NaN: it falls back to the mean over the whole matrix and warns. Note that this fallback uses the mean whichever method was requested, since a median or limit of detection is not defined for a feature with nothing observed.

Usage

```
safe_impute(Z, method = c("mean", "median", "lod"))
```

Arguments

Z	A numeric matrix with missing values coded NA.
method	The summary used to fill a feature: "mean" The feature's observed mean. "median" The feature's observed median; preferable for a skewed feature, where the mean is pulled towards the tail. "lod" The feature's observed minimum divided by $\sqrt{2}$, the standard substitution for values below an assay's limit of detection. This is appropriate only for data on the original measurement scale, where the minimum stands in for the detection limit; on centred or standardised data the observed minimum is negative and dividing it by $\sqrt{2}$ moves the filled value <i>up</i> , which is not what the convention intends.

Value

A matrix of the same dimensions as Z with no missing values, unless every value of Z is missing, in which case Z is returned unchanged with a warning.

See Also

[check_imputation_quality](#) to check the result, and the `init_impute` argument of [estimate_lucid](#) for the imputation LUCID applies internally.

Examples

```
Z <- matrix(rnorm(100), nrow = 10)
Z[2:4, 2] <- NA
colMeans(is.na(safe_impute(Z, method = "median")))
```

simulated_HELIX_data *A simulated HELIX dataset for LUCID*

Description

The Human Early-Life Exposome (HELIX) project is a multi-center research project that aims to characterize early-life environmental exposures and associate these with omics biomarkers and child health outcomes (Vrijheid, 2014. doi: 10.1289/ehp.1307204). This is a simulated subset of the HELIX data released for the Exposome Data Challenge 2021 (held by ISGlobal), used to illustrate the LUCID model on three omics layers.

The three omics layers share the same 420 subjects in the same row order, so they can be passed together as the Z list of a parallel or serial fit, or used one at a time for early integration.

Usage

```
simulated_HELIX_data
```

Format

A list of 4 elements, each with 420 observations:

phenotype A 420 by 6 data frame holding the exposure, the outcome and the covariates: `id`; `hs_hg_m_scaled`, the scaled maternal exposure to in-utero mercury, used as `G`; `ck18_scaled`, a scaled continuous indicator of metabolic-dysfunction-associated fatty liver disease (MAFLD), used as `Y`; and the covariates `hs_child_age_yrs_None`, `e3_sex_None` and `h_fish_preg_Ter`.

methylome A 420 by 10 matrix of methylomics features.

transcriptome A 420 by 10 matrix of transcriptomics features.

miRNA A 420 by 10 matrix of miRNA features.

sim_data	<i>A simulated dataset for LUCID</i>
----------	--------------------------------------

Description

An example dataset used to illustrate the LUCID model, simulated under two latent clusters. The exposures are associated with the latent cluster, which in turn affects PFAS concentration and liver injury in children; the clusters are also characterised by differential metabolite levels. Because the generating cluster membership is retained in `X`, the data can be used to check recovered clusters against the truth, up to the arbitrary labelling of clusters.

Usage

```
sim_data
```

Format

A list of 6 elements, each with 2000 observations:

G A 2000 by 10 matrix of exposures.

Z A 2000 by 10 matrix of metabolites.

Y_normal A 2000 by 1 matrix; continuous outcome, PFAS concentration in children.

Y_binary A 2000 by 1 matrix; binary outcome, liver injury status, coded 0 and 1.

Covariate A 2000 by 2 matrix of continuous covariates, usable as either `CoG` or `CoY`.

X An integer vector of length 2000 giving the latent cluster each observation was generated from.

summary.lucid_parallel

Summarize results of the parallel LUCID model

Description

Summarize results of the parallel LUCID model

Usage

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

Arguments

object	A LUCID model fitted by estimate_lucid
...	Additional argument boot.se, which can be an object returned by boot_lucid to display bootstrap CIs in print output.

Value

A list of class sumlucid_parallel, with the same components as the early-model summary (see [summary_lucid](#)) but resolved per omics layer: feature_selection reports both the overall retained set and the per-layer sets, parameters holds mu by layer and beta by layer alongside the single gamma, and missing_data is a list by layer. BIC and loglik remain scalars for the joint model. Returned invisibly when printed.

summary.lucid_serial *Summarize results of the serial LUCID model*

Description

Summarize results of the serial LUCID model

Usage

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

Arguments

object	A LUCID model fitted by estimate_lucid
...	Additional arguments. boot.se accepts an object returned by boot_lucid , whose confidence limits are then shown alongside the point estimates. auto_print = FALSE suppresses printing.

Value

A list of class `sumlucid_serial` with components:

BIC, loglik Assembled over the whole serial model, and repeated inside `model_fit`.

model_info The outcome family, `n_observations`, `n_stages`, `stage_type` (whether each stage is an "early" or a "parallel" sub-model) and `stage_K`, the clusters per stage.

regularization The penalties in force, or NULL if none.

missing_data `n_stages` and a per-stage breakdown.

stage_summary The heart of the object: one summary per stage, each with the same shape as the corresponding early or parallel summary. Outcome parameters appear on the final stage, the only one the outcome enters.

transition How consecutive stages connect: labels names the previous stage's clusters as they enter the next stage's design, and `prev_stage_type` records that stage's type. The first element is empty, the first stage having no predecessor.

boot.se The `boot.se` argument as supplied, or NULL.

summary.list A legacy alias of `stage_summary`, retained for downstream code written before the rename. Prefer `stage_summary`.

Returned invisibly when printed.

summary_lucid

Summarize results of the early LUCID model

Description

Assembles the reported quantities for a fitted LUCID model and, by default, prints them. The same components are returned invisibly as a list of class `sumlucid_early`, so they can be extracted programmatically rather than parsed from the printed output.

Two conventions are worth noting when reading the output. Outcome effects are printed as an intercept – cluster 1's level – followed by explicit contrasts of each remaining cluster against it, so the second row is a between-cluster difference and not that cluster's own mean. And the parameter tables are restricted to the features the model retained, so their dimensions match the fit rather than the original input.

Usage

```
summary_lucid(object, ...)
```

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

Arguments

object A LUCID model fitted by `estimate_lucid` or `lucid`.

... Additional arguments. `boot.se` accepts an object returned by `boot_lucid`, whose confidence limits are then shown alongside the point estimates. `auto_print = FALSE` suppresses printing and returns the summary list only.

Value

A list of class `sumlucid_early` with components:

BIC, loglik The Bayesian information criterion and the observed-data log-likelihood at the estimates. Also repeated inside `model_fit`, alongside `n_parameters`, the effective parameter count the BIC charges (Eq 13, reduced per Eq 18 when a penalty deselected variables).

model_info The outcome family, the number of clusters `K`, `n_observations`, and `n_features`, which counts *retained* exposures and omics features.

feature_selection Which exposures and omics features survived, and how many were dropped.

regularization The penalties in force, `Rho_G`, `Rho_Z_Mu` and `Rho_Z_Cov`.

parameters Estimates restricted to the retained features: `beta` (exposure-to-cluster, intercept column always kept along with any covariate columns), `mu` (cluster-specific omics means) and `gamma` (cluster-to-outcome, in both absolute and reference-coded form).

missing_data The fit's `missing_summary`; see [estimate_lucid](#).

boot.se The `boot.se` argument as supplied, or `NULL`.

When `boot.se` is supplied, every printed bootstrap CI table (G-to-X, cluster-to-Y, and cluster-specific omics means) gains a `sig` column: "*" where the normal-theory confidence interval excludes 0, "" otherwise.

See Also

[boot_lucid](#) for the confidence limits, and [predict_lucid](#) for cluster and outcome prediction.

Examples

```
# use simulated data (a small subset keeps the example quick)
G <- sim_data$G[1:150, , drop = FALSE]
Z <- sim_data$Z[1:150, , drop = FALSE]
Y_normal <- sim_data$Y_normal[1:150]

# fit lucid model
fit1 <- estimate_lucid(G = G, Z = Z, Y = Y_normal, lucid_model = "early", family = "normal", K = 2,
seed = 1008, max_itr = 20, max_tot_itr = 50)

# conduct bootstrap resampling
boot1 <- suppressWarnings(
  boot_lucid(G = G, Z = Z, Y = Y_normal, lucid_model = "early", model = fit1, R = 3)
)

# summarize lucid model
summary(fit1)

# summarize lucid model with bootstrap CIs
summary(fit1, boot.se = boot1)
```

tune_lucid

*Wrapper for LUCID Model and Penalty Tuning***Description**

Fit a grid of LUCID models over candidate numbers of latent clusters K and (optionally) L1 penalties Rho_G , Rho_Z , and $\text{Rho}_{Z\text{-Cov}}$. The input format for K differs by `lucid_model`. For "early", use an integer vector (for example, 2:4). For "parallel", use a list of vectors/integers, one per layer (for example, `list(2:3, 2:3, 2)`). For "serial", use a nested list as required by the serial model.

Usage

```
tune_lucid(
  G,
  Z,
  Y,
  CoG = NULL,
  CoY = NULL,
  family = c("normal", "binary"),
  K,
  lucid_model = c("early", "parallel", "serial"),
  Rho_G = 0,
  Rho_Z_Mu = 0,
  Rho_Z_Cov = 0,
  verbose_tune = FALSE,
  ...
)
```

Arguments

G	Exposures, a numeric vector, matrix, or data frame. Categorical variables should be transformed into dummy variables.
Z	Omics data. If "early", an N by M matrix. If "parallel", a list of matrices (same N). If "serial", a list matching the serial model structure.
Y	Outcome, a numeric vector. Binary outcomes should be coded as 0/1.
CoG	Optional covariates for the G-to-X model.
CoY	Optional covariates for the X-to-Y model.
family	Outcome family: "normal" or "binary".
K	Candidate latent-cluster values in model-specific format.
lucid_model	LUCID model type: "early", "parallel", or "serial".
Rho_G	Scalar or vector penalty for exposure coefficients in the G-to-X model. CoG covariates are included unpenalized. Vector tuning is supported for "early" and "parallel". For "serial", only scalar inputs are supported.
Rho_Z_Mu	Scalar or vector penalty for cluster-specific Z means. Vector tuning is supported for "early" and "parallel". For "serial", only scalar inputs are supported.

Rho_Z_Cov	Scalar or vector penalty for cluster-specific Z covariance matrices. Vector tuning is supported for "early" and "parallel". For "serial", only scalar inputs are supported.
verbose_tune	Logical; print tuning progress if TRUE.
...	Additional arguments passed to estimate_lucid .

Value

A list holding the tuning table, every fitted candidate, and the selected model. The element names differ by `lucid_model`:

- "early": `tune_list`, `res_model`, `best_model`
- "parallel" and "serial": `tune_K`, `model_list`, `model_opt`

The tuning table has one row per grid point, in the order the candidates were fitted, and the fitted-model list is aligned with it by position. Its columns are the grid coordinates – K (one column per layer or stage for "parallel" and "serial") together with `Rho_G`, `Rho_Z_Mu` and `Rho_Z_Cov` – followed by BIC.

Selection is on BIC, minimised over the rows. For "early" and "parallel" this is the penalized BIC of Eq 18: the full parameter count is reduced by one for each variable deselected by the penalty, so a sparser fit is not charged for coefficients it has driven to zero. Penalties are not tuned for "serial" – only scalar penalties are accepted there – so a serial grid varies K alone. A candidate whose EM algorithm failed records NA and is skipped; if every candidate fails, an error is raised rather than a model returned. Ties are broken by taking the first minimising row.

Note that the returned optimum is the penalized fit itself. It is [lucid](#), not this function, that refits the selected variables without a penalty – so estimates taken straight from `best_model` or `model_opt` here are shrunk towards zero.

Examples

```
## Not run:
G <- sim_data$G
Z <- sim_data$Z
Y <- sim_data$Y_normal
tune_early <- tune_lucid(G = G, Z = Z, Y = Y, lucid_model = "early", K = 2:3)
tune_rho <- tune_lucid(
  G = G, Z = Z, Y = Y, lucid_model = "early", K = 2,
  Rho_G = c(0, 0.1), Rho_Z_Mu = c(0, 5), Rho_Z_Cov = c(0, 0.1)
)

## End(Not run)
```

Index

* datasets

sim_data, [29](#)

simulated_HELIX_data, [28](#)

analyze_missing_pattern, [2](#), [7](#), [8](#)

boot_lucid, [3](#), [30–32](#)

check_imputation_quality, [5](#), [28](#)

check_na, [3](#), [7](#)

estimate_lucid, [8](#), [13](#), [17](#), [18](#), [23](#), [28](#), [30–32](#),
[34](#)

get_cluster_assignment, [12](#)

get_selected_G, [13](#)

get_selected_Z, [14](#)

get_top_omics_features, [15](#)

lucid, [16](#), [18](#), [31](#), [34](#)

plot.early_lucid, [18](#), [20](#)

plot.lucid_parallel, [19](#), [19](#)

plot.lucid_serial, [19](#), [20](#)

plot_cluster_omic_profile, [21](#)

predict_lucid, [23](#), [32](#)

print.sumlucid_early, [25](#)

print.sumlucid_parallel, [26](#)

print.sumlucid_serial, [27](#)

safe_impute, [6](#), [27](#)

sankeyNetwork, [19](#)

sim_data, [29](#)

simulated_HELIX_data, [28](#)

summary.early_lucid(summary_lucid), [31](#)

summary.lucid_parallel, [30](#)

summary.lucid_serial, [30](#)

summary_lucid, [30](#), [31](#)

tune_lucid, [33](#)