# Package: MetVAE (via r-universe)

November 23, 2026

**Type** Package

**Title** Sparse Correlation Networks for Untargeted Metabolomics Data with a Variational Autoencoder

**Version** 0.99.0

**Date** 2026-11-20

**Description** Estimates a sparse correlation network among metabolites from untargeted metabolomics abundance data. Zeros are treated as values below the detection limit and are imputed by a variational autoencoder, measured covariates are removed on the centered log-ratio scale, and the correlation estimate is sparsified either by Fisher z-tests with a multiplicity correction or by the sparse estimation of correlation (SEC) estimator with a cross-validated penalty. Results can be exported as GraphML networks.

**License** MIT + file LICENSE

**Depends** R (>= 4.5.0)

**Imports** igraph, Matrix, methods, stats, utils, SummarizedExperiment, withr, torch (>= 0.16.0)

**Suggests** BiocStyle, knitr, rmarkdown, testthat (>= 3.0.0)

**VignetteBuilder** knitr

**SystemRequirements** LibTorch (https://pytorch.org/), downloaded once by running torch::install_torch() after the R package torch is installed

**biocViews** Software, Metabolomics, MassSpectrometry, Network, NetworkInference, NeuralNetwork, DimensionReduction, Preprocessing, Regression

**URL** https://github.com/MATRIX-Research-Lab/MetVAE

**BugReports** https://github.com/MATRIX-Research-Lab/MetVAE/issues

**Encoding** UTF-8

**Roxygen** list(markdown = FALSE)

**Config/testthat/edition** 3

**Config/roxygen2/version** 8.1.1

**Config/pak/sysreqs** libglpk-dev libxml2-dev zlib1g-dev

**Repository** https://biocstaging.r-universe.dev

**Date/Publication** 2026-11-23 15:48:19 UTC

**RemoteUrl** https://github.com/BiocStaging/MetVAE

**RemoteRef** HEAD

**RemoteSha** 9832a7f1a28f5c98cefff3799a654799a2e0fbd0

# Contents

confound . . . . . 2
export_graphml . . . . . 3
get_corr . . . . . 4
impute_zeros . . . . . 5
loadings . . . . . 6
metvae . . . . . 7
metvae-methods . . . . . 9
metvae_sim_data . . . . . 10
sparse_by_p . . . . . 11
sparse_by_sec . . . . . 12

**Index** **15**

---

`confound` *Covariate coefficients and effects*

---

## Description

`confound_coef` returns the estimated covariate coefficients on the centered log-ratio (CLR) scale, and `confound_es` returns the covariate effect of every sample and feature, i.e. the scaled covariate matrix times the coefficients. Continuous covariates are centered and scaled, so a coefficient is the change in CLR abundance per standard deviation of the covariate.

## Usage

```
confound_coef(fit)

confound_es(fit)
```

## Arguments

| | |
|---|---|
| `fit` | An object returned by `metvae`. |

## Value

confound_coef: a features x covariates matrix, or NULL when the model has no covariates. confound_es: a samples x features matrix, or NULL.

## See Also

metvae

## Examples

```
fits <- readRDS(system.file("extdata", "metvae_example_fits.rds",
                            package = "MetVAE"))
head(confound_coef(fits$example))
confound_es(fits$example)[1:3, 1:4]
```

---

# export_graphml *Export correlation networks as GraphML*

---

## Description

Builds one undirected network per cutoff from a sparse correlation matrix: the nodes are features and an edge joins two features whose correlation has absolute value at least the cutoff. The networks are returned as igraph objects and optionally written as GraphML files, which can be opened in Cytoscape.

## Usage

```
export_graphml(
  x,
  cutoffs = 0.7,
  output_dir = NULL,
  file_prefix = "correlation_graph_cutoff"
)
```

## Arguments

| | |
|---|---|
| x | A features x features correlation matrix (matrix, data.frame, or Matrix), or a result of sparse_by_p or sparse_by_sec, whose sparse_estimate is used. |
| cutoffs | Positive cutoffs of the absolute correlation. Default 0.7. |
| output_dir | Directory for the GraphML files, created if absent; NULL (default) writes no files. |
| file_prefix | Prefix of the file names; the cutoff and ".graphml" are appended. Default "correlation_graph_cutoff". |

## Value

A named list of `igraph` graphs, one per cutoff with at least one edge, named "Correlation_cutoff" followed by the cutoff. Every edge has the attributes weight, correlation, and EdgeScore (all equal to the correlation) and EdgeType and id (the graph name).

## See Also

`sparse_by_p`, `sparse_by_sec`

## Examples

```
fits <- readRDS(system.file("extdata", "metvae_example_fits.rds",
                            package = "MetVAE"))
res <- sparse_by_p(fits$example)
graphs <- export_graphml(res, cutoffs = c(0.3, 0.5))
graphs
```

---

# get_corr *Estimate the correlation matrix by multiple imputation*

---

## Description

Imputes the zeros `num_sim` times with the trained autoencoder, computes the correlation matrix of each completed data set from the variance of the log ratios, sets entries with absolute value below `threshold` to zero, and averages the matrices over the imputations.

## Usage

```
get_corr(fit, num_sim = 100L, threshold = 0.2, seed = NULL)
```

## Arguments

| | |
|---|---|
| `fit` | An object returned by `metvae`. |
| `num_sim` | Number of imputations. Default 100. |
| `threshold` | Entries of each imputation's correlation matrix with absolute value below this value are set to zero before averaging. Default 0.2. |
| `seed` | Integer seed; imputation s uses the seed `seed + s - 1`. NULL (default) uses the seed of `fit`. |

## Details

Results are identical for a given seed on one machine with one version of the package torch. The number of threads used by torch, set with `torch::torch_set_num_threads()`, can change the last digits.

## Value

`fit` with the component `corr`: a list with `impute_log_data` (samples x features matrix, mean of the imputed log abundances), `estimate` (features x features sparse matrix of class `dgCMatrix`, mean correlation), and the settings `num_sim`, `threshold`, and `seed`. Assign the result: `fit <- get_corr(fit)`.

## See Also

`metvae`, `sparse_by_p`, `sparse_by_sec`

## Examples

```
if (requireNamespace("torch", quietly = TRUE) &&
    torch::torch_is_installed()) {
    sim <- metvae_sim_data(n = 40, d = 10, cor_pairs = 2, mu = 10:12,
                           seed = 1)
    y <- sim$y
    y[y < apply(y, 2, stats::quantile, 0.2)[col(y)]] <- 0
    fit <- metvae(y, max_epochs = 20, verbose = FALSE)
    fit <- get_corr(fit, num_sim = 5)
    fit$corr$estimate[1:4, 1:4]
}
```

---

# impute_zeros *Impute the zeros*

---

## Description

Completes the centered log-ratio (CLR) data of a fit once: the zeros are given random initial values from the lower tail of the estimated feature distributions and then replaced by their reconstruction by the trained autoencoder. Observed entries are unchanged.

## Usage

```
impute_zeros(fit, seed = NULL)
```

## Arguments

| | |
|---|---|
| `fit` | An object returned by `metvae`. |
| `seed` | Integer seed of the random draws. NULL (default) uses the global torch random number generator. |

## Value

A samples x features matrix of completed CLR values, after removal of the covariate effects.

## See Also

metvae, get_corr

## Examples

```
if (requireNamespace("torch", quietly = TRUE) &&
    torch::torch_is_installed()) {
    sim <- metvae_sim_data(n = 40, d = 10, cor_pairs = 2, mu = 10:12,
                           seed = 1)
    y <- sim$y
    y[y < apply(y, 2, stats::quantile, 0.2)[col(y)]] <- 0
    fit <- metvae(y, max_epochs = 20, verbose = FALSE)
    impute_zeros(fit, seed = 1)[1:3, 1:4]
}
```

---

# loadings *Decoder loadings and co-occurrence*

---

## Description

clr_loading returns the weights of the linear decoder, which give the loading of every feature on every latent variable on the centered log-ratio scale. cooccurrence returns the squared Euclidean distance between the loading vectors of every pair of features, multiplied by (n - 1) / n for n samples; small values indicate features that vary together.

## Usage

```
clr_loading(fit)

cooccurrence(fit)
```

## Arguments

| | |
|---|---|
| fit | An object returned by metvae. |

## Value

clr_loading: a features x latent variables matrix with columns latent_0, latent_1, .... cooccurrence: a features x features matrix.

## See Also

metvae

## Examples

```
fits <- readRDS(system.file("extdata", "metvae_example_fits.rds",
                            package = "MetVAE"))
dim(clr_loading(fits$example))
cooccurrence(fits$example)[1:3, 1:3]
```

---

`metvae` *Fit the MetVAE model*

---

## Description

Preprocesses an abundance table and trains the variational autoencoder of MetVAE. Features whose proportion of zeros exceeds `feature_zero_threshold` are removed, the data are transformed to the centered log-ratio (CLR) scale, the feature means, standard deviations, and covariate coefficients are estimated by a left-censored normal model in which zeros are values below the detection limit, the covariate effects are removed, and the autoencoder is trained on the result. The correlations are estimated afterwards by `get_corr`.

## Usage

```
metvae(
  x,
  meta = NULL,
  assay_name = NULL,
  features_as_rows = FALSE,
  continuous_covariate_keys = NULL,
  categorical_covariate_keys = NULL,
  feature_zero_threshold = 0.3,
  sample_zero_threshold = NULL,
  latent_dim = NULL,
  hidden_dims = NULL,
  activation = c("relu", "tanh", "gelu", "silu", "none"),
  batch_size = 128L,
  max_epochs = 1000L,
  learning_rate = 0.001,
  max_grad_norm = 1,
  shuffle = TRUE,
  seed = 0L,
  verbose = TRUE
)
```

## Arguments

| | |
|---|---|
| `x` | A `SummarizedExperiment` with features in rows and samples in columns, or a numeric matrix or data.frame with samples in rows and features in columns (see `features_as_rows`). Values are non-negative abundances; zeros mark values below the detection limit and missing values are treated as zeros. |

`meta` A data.frame of sample metadata with the sample names as row names, or NULL. Every sample of `x` must be present. Must be NULL when `x` is a `SummarizedExperiment`, whose `colData` is used.

`assay_name` Name of the assay of a `SummarizedExperiment`; NULL uses the first assay.

`features_as_rows` Logical; TRUE when the rows of a matrix or data.frame `x` are features. Default FALSE.

`continuous_covariate_keys` Character vector of numeric metadata columns to adjust for, or NULL. They are centered and scaled.

`categorical_covariate_keys` Character vector of categorical metadata columns to adjust for, or NULL. Each is coded by indicator variables with the first level as reference (factor levels as given, other values sorted).

`feature_zero_threshold` Features with a proportion of zeros greater than this value are removed. NULL keeps every feature. Default 0.3.

`sample_zero_threshold` Samples with a proportion of zeros greater than this value are removed after the feature filter. NULL (default) keeps every sample.

`latent_dim` Number of latent variables. NULL (default) uses the smaller of the numbers of samples and features.

`hidden_dims` Integer vector of hidden layer sizes of the encoder, or NULL (default) for a linear encoder.

`activation` Activation between hidden layers: "relu" (default), "tanh", "gelu", "silu", or "none".

`batch_size` Number of samples per minibatch. Default 128.

`max_epochs` Number of training epochs. Default 1000.

`learning_rate` Initial learning rate of the AdamW optimizer; a cosine schedule with warm restarts varies it between this value and half of it. Default 1e-3.

`max_grad_norm` Maximum 2-norm of the gradient, or NULL for no clipping. Default 1.

`shuffle` Logical; TRUE (default) permutes the samples every epoch.

`seed` Integer seed of the model initialization, the batch order, and the default seed of `get_corr`. Default 0.

`verbose` Logical; TRUE (default) reports the preprocessing steps and shows a progress bar in interactive sessions.

## Value

An object of class `"metvae"`: a list holding the sample, feature, and covariate names, the preprocessed CLR data (`clr_data`, samples x features, NA at the zeros), the per-sample offsets (`shift`), the zero counts, the estimated CLR means (`clr_mean`), standard deviations (`clr_sd`) and covariate coefficients (`clr_coef`, covariates x features, or NULL), the model weights, and the training loss of each epoch (`train_loss`). The object contains only R objects and can be saved with `saveRDS()`.

## See Also

get_corr, sparse_by_p, sparse_by_sec

## Examples

```
if (requireNamespace("torch", quietly = TRUE) &&
    torch::torch_is_installed()) {
    sim <- metvae_sim_data(n = 40, d = 10, cor_pairs = 2, mu = 10:12,
                           seed = 1)
    fit <- metvae(sim$y, max_epochs = 20, verbose = FALSE)
    fit
}
```

---

# metvae-methods *Print and summarize a MetVAE fit*

---

## Description

`print` reports the data dimensions before and after filtering, the proportion of zeros, the covariates, the model size, the training, and whether correlations have been estimated. `summary` returns these quantities as a list.

## Usage

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

## S3 method for class 'summary.metvae'
print(x, ...)

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

## Arguments

| | |
|---|---|
| `...` | Unused. |
| `x`, `object` | An object returned by `metvae`. |

## Value

`print` returns `x` invisibly. `summary` returns an object of class `"summary.metvae"`, a list with the components n_samples, n_features, n_input_samples, n_input_features, zero_fraction, covariates, latent_dim, hidden_dims, epochs, final_loss, and n_edges (number of nonzero off-diagonal pairs of the correlation estimate, or NA).

## Examples

```
fits <- readRDS(system.file("extdata", "metvae_example_fits.rds",
                            package = "MetVAE"))
fits$example
summary(fits$example)$n_edges
```

---

`metvae_sim_data` *Simulate metabolomics abundance data*

---

## Description

Simulates log-normal abundances of d features in n samples. The first `2 * cor_pairs` features form disjoint consecutive pairs (1, 2), (3, 4), ... with correlations drawn from +/-0.5, +/-0.6, and +/-0.7; the other features are independent. The mean of each log abundance is drawn from `mu`. When covariates are supplied, every correlated feature and a proportion `da_prop` of the other features receive covariate effects drawn from -2, -1, 1, 2.

## Usage

```
metvae_sim_data(
  n,
  d,
  cor_pairs = 0L,
  mu,
  sigma = 1,
  x = NULL,
  cont_list = NULL,
  cat_list = NULL,
  da_prop = 0.1,
  seed = NULL
)
```

## Arguments

| | |
|---|---|
| `n` | Number of samples. |
| `d` | Number of features. |
| `cor_pairs` | Number of correlated pairs, at most `d %/% 2`. Default 0. |
| `mu` | Candidate means of the log abundances. |
| `sigma` | Standard deviation of the log abundances. Default 1. |
| `x` | A data.frame of covariates with n rows, or NULL. |
| `cont_list` | Names of the continuous covariates in `x`; they are centered and scaled. |
| `cat_list` | Names of the categorical covariates in `x`; they are coded by indicators with the first level dropped. |
| `da_prop` | Proportion of uncorrelated features with covariate effects. Default 0.1. |
| `seed` | Integer seed, or NULL to use the current random number generator state. The global state is restored after a seeded call. |

## Value

A list with `y` (n x d abundance matrix with samples s0, s1, ... and features f0, f1, ...), `cor_matrix` (true correlation matrix of the log abundances), `x` (covariate matrix used, or NULL), and `beta` (d x covariates effect matrix, or NULL).

## Examples

```
sim <- metvae_sim_data(n = 50, d = 20, cor_pairs = 4, mu = 10:14,
                       seed = 1)
dim(sim$y)
which(sim$cor_matrix[1:4, 1:4] != 0, arr.ind = TRUE)
```

---

`sparse_by_p` *Sparsify the correlation matrix by Fisher z-tests*

---

## Description

Tests every off-diagonal correlation of the estimate from `get_corr` with the Fisher z-transformation, adjusts the p-values for multiplicity, and sets the correlations whose adjusted p-value exceeds `cutoff` to zero.

## Usage

```
sparse_by_p(
  fit,
  p_adj_method = "fdr_bh",
  cutoff = 0.05,
  pq = c("dense", "edges", "none")
)
```

## Arguments

| | |
|---|---|
| `fit` | An object returned by `get_corr`. |
| `p_adj_method` | Multiplicity adjustment, with the names and formulas of statsmodels: "fdr_bh" (Benjamini-Hochberg, default), "fdr_by" (Benjamini-Yekutieli), "bonferroni", "sidak", "holm", "holm-sidak", "simes-hochberg", "hommel", "fdr_tsbh" and "fdr_tsbky" (two-stage procedures at level 0.05). |
| `cutoff` | Correlations with an adjusted p-value greater than this value are set to zero. Default 0.05. |
| `pq` | Form of the p-values and adjusted p-values: "dense" (default) returns features x features matrices; "edges" returns a data.frame with one row per feature pair with a nonzero estimate, which needs much less memory for many features; "none" omits them. |

## Details

The test statistic of a pair with correlation r is `atanh(r) * sqrt(n - 3)`, where n is the number of samples. Pairs with r = 0 have p-value 1 and are included in the multiplicity adjustment.

## Value

A list with `estimate` (sparse features x features matrix of class `dgCMatrix`; the estimate of `get_corr` with entries restricted to [-1 + 1e-7, 1 - 1e-7] and unit diagonal), `p_value` and `q_value` (unadjusted and adjusted p-values, see `pq`; zero on the diagonal), and `sparse_estimate` (sparse matrix of the retained correlations).

## See Also

`get_corr`, `sparse_by_sec`, `export_graphml`

## Examples

```
fits <- readRDS(system.file("extdata", "metvae_example_fits.rds",
                            package = "MetVAE"))
res <- sparse_by_p(fits$example, cutoff = 0.05)
res$sparse_estimate[1:4, 1:4]
```

---

# sparse_by_sec *Sparsify the correlation matrix by SEC*

---

## Description

Computes the sparse estimate of the correlation matrix of Cui, Leng and Sun (2016), which minimizes `0.5 * ||R - Rn||_F^2 + rho * sum(W * |R|)` over correlation matrices R with eigenvalues at least `epsilon`, where Rn is the estimate from `get_corr` and `W = 1 / |Rn|`. The penalty is `rho` when given, and otherwise chosen by K-fold cross-validation over `rho = c * sqrt(log(p) / n)` for `c` in `c_grid`.

## Usage

```
sparse_by_sec(
  fit,
  rho = NULL,
  c_grid = seq_len(10L),
  n_splits = 5L,
  refine = TRUE,
  refine_points = 10L,
  threshold = 0.1,
  epsilon = 1e-05,
  tol = 0.001,
  max_iter = 1000L,
```

```
  restart = 50L,
  line_search_apg = TRUE,
  delta = NULL,
  c_delta = 0.1,
  seed = 0L
)
```

## Arguments

| | |
|---|---|
| fit | An object returned by get_corr. |
| rho | Penalty. NULL (default) selects it by cross-validation. |
| c_grid | Values of c for the cross-validation. Default 1, ..., 10. |
| n_splits | Number of folds. Default 5. |
| refine | Logical; TRUE (default) evaluates a finer grid between the neighbors of the best value of c. |
| refine_points | Number of points of the finer grid. Default 10. |
| threshold | Off-diagonal entries of the estimate with absolute value below this value are set to zero. Default 0.1. |
| epsilon | Lower bound on the eigenvalues. Default 1e-5. |
| tol | Convergence tolerance of the relative gradient residual. Default 1e-3. |
| max_iter | Maximum number of iterations. Default 1000. |
| restart | Period of the momentum restart; NULL or 0 disables it. Default 50. |
| line_search_apg | Logical; TRUE (default) adapts the step size. |
| delta | Entries of Rn with absolute value below delta are fixed at zero; NULL (default) uses c_delta * sqrt(log(p) / n). |
| c_delta | Scale of the default delta. Default 0.1. |
| seed | Integer seed of the fold assignment. Default 0. |

## Details

The columns of Rn are centered and scaled before the iterations and the result is symmetrized, as in the Python package metvae. In Cui, Leng and Sun (2016) the standardization applies to the data variables.

The implementation is adapted from the MATLAB code of Cui, Leng and Sun (2016) and is distributed with the permission of Chenlei Leng.

## Value

A list with estimate (the estimate of get_corr), sparse_estimate (sparse features x features matrix of class dgCMatrix), best_rho (the penalty used), and scores_by_rho (data.frame with columns c, rho, and the mean validation error score, ordered by rho; NULL when rho is given).

## References

Cui Y, Leng C, Sun D. Sparse estimation of high-dimensional correlation matrices. Computational Statistics and Data Analysis 93 (2016) 390-403. doi:10.1016/j.csda.2014.10.001

## See Also

get_corr, sparse_by_p

## Examples

```
if (requireNamespace("torch", quietly = TRUE) &&
    torch::torch_is_installed()) {
    fits <- readRDS(system.file("extdata", "metvae_example_fits.rds",
                                package = "MetVAE"))
    res <- sparse_by_sec(fits$example, c_grid = c(1, 2, 3),
                         refine = FALSE)
    res$best_rho
}
```

# Index

clr_loading *(*loadings*)*, 6
confound, 2
confound_coef *(*confound*)*, 2
confound_es *(*confound*)*, 2
cooccurrence *(*loadings*)*, 6

export_graphml, 3, *12*

get_corr, 4, *6–9*, *11–14*

impute_zeros, 5

loadings, 6

metvae, *2–6*, 7, *9*
metvae-methods, 9
metvae_sim_data, 10

print.metvae *(*metvae-methods*)*, 9
print.summary.metvae *(*metvae-methods*)*, 9

sparse_by_p, *3–5*, *9*, 11, *14*
sparse_by_sec, *3–5*, *9*, 12, *12*
summary.metvae *(*metvae-methods*)*, 9