Package ‘MetaDICT’

February 2, 2026

Title Microbiome data integration method via shared dictionary learning

Version 1.1.0

Description MetaDICT is a method for the integration of microbiome data. This method is designed to remove batch effects and preserve biological variation while integrating heterogeneous datasets. MetaDICT can better avoid overcorrection when unobserved confounding variables are present.

License Artistic-2.0

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

Depends R (>= 4.2.0)

Imports stats, RANN, igraph, vegan, edgeR, ecodist, ggplot2, viridis, ggpubr, ape, cluster, matrixStats

BugReports `https://github.com/BoYuan07/MetaDICT/issues`

URL `https://github.com/BoYuan07/MetaDICT`

biocViews Microbiome, BatchEffect, Sequencing, Clustering, Software

Suggests BiocStyle, knitr, rmarkdown, DT, ggraph, tidyverse, testthat (>= 3.0.0)

Config/testthat/edition 3

VignetteBuilder knitr

git_url https://git.bioconductor.org/packages/MetaDICT

git_branch devel

git_last_commit b0ced9b

git_last_commit_date 2025-10-29

Repository Bioconductor 3.23

Date/Publication 2026-02-01

Author Bo Yuan [aut, cre] (ORCID: <`https://orcid.org/0009-0008-5428-4447`>), Shulei Wang [aut]

Maintainer Bo Yuan <`boyuan5@illinois.edu`>

Contents

`community_detection` *Taxa/Sample Community detection.*

Description

A `k`-nearest neighbor graph is constructed based on Euclidean distance. Then community detection method is applied to identify communities. Various values of `k` within a specific range are tried and the one that yields the highest average Silhouette score is selected.

Usage

```
community_detection(
  X,
  max_k = 10,
  method = "Louvain",
  resolution = 1,
  min_k = 2
)
```

Arguments

`X`	Input data. Rows represent clustering objects, and columns represent features.
`max_k`	The largest number of connected neighbors.
`method`	The community detection method to use. Options include `"Louvain"` and `"Walktrap"`.
`resolution`	The resolution parameter for the Louvain algorithm.
`min_k`	The smallest number of connected neighbors.

Value

A `list` with the following components:

`cluster`	– The estimated cluster labels.
`graph`	– The `k`-nearest neighbor graph.

Examples

```
data(exampleData)
O = exampleData$O
meta = exampleData$meta
dist_mat = exampleData$dist_mat
metadict_res = MetaDICT(O, meta, distance_matrix = dist_mat)
D = metadict_res$D
D_filter = D[,1:20]
taxa_c = community_detection(D_filter, max_k = 5)
```

`data_check` *Data Check*

Description

Check the format of inputs.

Usage

```
data_check(
  count,
  meta,
  covariates = "all",
  distance_matrix = NULL,
  tree = NULL,
  taxonomy = NULL,
  tax_level = NULL,
  verbose = TRUE
)
```

Arguments

`count`	The integrated count table of taxa by samples. The `count` parameter should be provided as either a `matrix` or a `data.frame`.
`meta`	The integrated meta table `meta` contains sample information and batch id.
`covariates`	The covariates used in data integration. Default is all.
`distance_matrix`	The distance matrix that measures sequence dissimilarity.
`tree`	The phylogenetic tree (optional if distance matrix or taxonomy is provided).
`taxonomy`	The taxonomy table (optional if distance matrix or phylogenetic tree is provided).
`tax_level`	The taxonomic level of count table.
`verbose`	Logical; whether to print progress messages. Default is TRUE.

Value

a `list` contains count list, meta table list, sequencing distance matrix and parameters.

`exampleData` *Example microbiome dataset bundle*

Description

A list containing example OTU table, sample metadata, taxonomy, tree, and sequence distance matrix.

Usage

```
data(exampleData)
```

Format

A named list with 5 elements: `O`, `dist_mat`, `meta`, `taxonomy`, `tree`.

`exampleData_transfer` *Example microbiome dataset bundle for transfer learning*

Description

A list containing example OTU table, sample metadata.

Usage

```
data(exampleData_transfer)
```

Format

A named list file with 2 elements: `new_data`, `new_meta`.

MetaDICT — *Microbiome data integration method via shared dictionary learning.*

Description

A method for microbiome data integration. This method is designed to remove batch effects and preserve biological variation while integrating heterogeneous datasets. MetaDICT can better avoid overcorrection when unobserved confounding variables are present.

Usage

```
MetaDICT(
  count,
  meta,
  covariates = "all",
  tree = NULL,
  taxonomy = NULL,
  distance_matrix = NULL,
  tax_level = NULL,
  customize_parameter = FALSE,
  alpha = 0.1,
  beta = 0.01,
  normalization = "uq",
  max_iter = 10000,
  imputation = FALSE,
  verbose = TRUE,
  optim_trace = FALSE
)
```

Arguments

`count`	The integrated count table (taxa-by-sample matrix). Should be provided as either a `matrix` or a `data.frame`.
`meta`	The integrated meta table containing sample information and batch IDs. The data must include a column named 'batch' containing all batch IDs. The row names of the meta should match the sample names in the count table.
`covariates`	The covariates used in data integration. Default is `"all"`.
`tree`	The phylogenetic tree (optional if a distance matrix or taxonomy is provided).
`taxonomy`	The taxonomy table (optional if a distance matrix or phylogenetic tree is provided). The row names of the taxonomy table should match the taxa names in the count table.
`distance_matrix`	A `matrix` measuring the dissimilarity of taxa. Default is `NULL`, in which case MetaDICT generates a distance matrix based on phylogenetic and taxonomic information.

`tax_level`	The taxonomic level of the count table.
`customize_parameter`	A logical variable. Set to `TRUE` if the `alpha` and `beta` parameters are customized. If `FALSE`, MetaDICT determines these parameters based on the number of covariates.
`alpha`	A parameter controlling the rank of the final corrected count table. A larger `alpha` leads to a lower-rank shared dictionary.
`beta`	A parameter controlling the smoothness of the estimated measurement efficiency. A larger `beta` results in more similar measurement efficiencies across taxa.
`normalization`	The normalization method. Options are `"Upper quantile"`, `"RSim"` or `"TSS"`. Set to `NULL` if normalization is not needed.
`max_iter`	The maximum number of iterations for the optimization process. Default is `10000`.
`imputation`	A logical variable. Whether to allow MetaDICT to perform imputation based on dictionary learning results. Default is `FALSE`.
`verbose`	A logical variable. Whether to generate verbose output. Default is `TRUE`.
`optim_trace`	A logical variable. Whether to print optimization steps. Default is `FALSE`.

Details

MetaDICT is a two-step approach. It initially estimates the batch effects by covariate balancing, then refines the estimation via shared dictionary learning.

Value

A `list` with the following components:

`count`	(`data.frame`) – The corrected count table. Rows represent taxa, and columns represent samples.
`D`	(`matrix`) – The estimated shared dictionary.
`R`	(`matrix`) – The estimated sample representation.
`w`	(`matrix`) – The estimated measurement efficiency. Rows represent datasets, and columns represent taxa.
`meta`	(`data.frame`) – The meta table used in the covariate balancing step.
`dist_mat`	(`matrix`) – The distance matrix measuring taxa dissimilarity.

Examples

```
data(exampleData)
O = exampleData$O
meta = exampleData$meta
dist_mat = exampleData$dist_mat
metadict_res = MetaDICT(O, meta, distance_matrix = dist_mat)
```

`metadict_add_new_data` *Batch correction for new datasets using existing dictionary.*

Description

This function adds new studies to an integrated dataset using a pre-learned dictionary. The corrected data can be directly used with machine learning models trained on the previously integrated dataset, enabling seamless application without retraining.

Usage

```
metadict_add_new_data(
  newdata,
  newmeta,
  integrated_result,
  customize_parameter = FALSE,
  beta = 0.01,
  normalization = "uq",
  max_iter = 10000,
  imputation = FALSE,
  verbose = TRUE,
  optim_trace = FALSE
)
```

Arguments

`newdata`	The integrated count table of new studies. Rows represent taxa, and columns represent samples. Should be provided as either a `matrix` or a `data.frame`.
`newmeta`	The integrated meta table (`meta`) for the new studies, containing sample information and batch IDs.
`integrated_result`	The output list from a previous MetaDICT integration task.
`customize_parameter`	A logical variable. Set to `TRUE` if the `beta` parameter is customized. If `FALSE`, MetaDICT determines `beta` based on the number of covariates.
`beta`	A parameter controlling the smoothness of the estimated measurement efficiency. A larger `beta` results in more similar measurement efficiency across taxa.
`normalization`	The normalization method. Options are `"Upper quantile"`, `"RSim"` or `"TSS"`. Set to `NULL` if normalization is not needed. This should be the same as in the previous integration task.
`max_iter`	The maximum number of iterations for the optimization process. Default is `10000`.
`imputation`	A logical variable. Whether to allow MetaDICT to perform imputation based on dictionary learning results. Default is `FALSE`.

`verbose`	A logical variable. Whether to generate verbose output. Default is `TRUE`.
`optim_trace`	A logical variable. Whether to print optimization steps. Default is `FALSE`.

Details

This function estimates measurement efficiency and debiased representations for new studies while keeping the dictionary unchanged.

Value

A `list` with the following components:

`count`	(`data.frame`) – The corrected count table. Rows represent taxa, and columns represent samples.
`D`	(`matrix`) – The estimated shared dictionary.
`R`	(`matrix`) – The estimated sample representation.
`w`	(`matrix`) – The estimated measurement efficiency. Rows represent datasets, and columns represent taxa.
`meta`	(`data.frame`) – The meta table used in the covariate balancing step.
`dist_mat`	(`matrix`) – The distance matrix measuring taxa dissimilarity.

Examples

```
data(exampleData)
O = exampleData$O
meta = exampleData$meta
dist_mat = exampleData$dist_mat
metadict_res = MetaDICT(O, meta, distance_matrix = dist_mat)
data("exampleData_transfer")
new_data = exampleData_transfer$new_data
new_meta = exampleData_transfer$new_meta
new_data_res = metadict_add_new_data(new_data, new_meta, metadict_res)
```

`pcoa_plot_continuous` *PCoA plots for continuous variables.*

Description

PCoA plots for continuous variables.

Usage

```
pcoa_plot_continuous(
  X,
  covariate,
  title,
  R2 = TRUE,
  dissimilarity = "Bray-Curtis",
  point_size = 1
)
```

Arguments

`X`	Abundance matrix. Rows represent taxa, and columns represent samples.
`covariate`	A discrete sample covariate.
`title`	The title of the graph.
`R2`	A logical variable. Whether to display the R^2 statistic in the subtitle. Default is `TRUE`.
`dissimilarity`	The dissimilarity type to use. Options include: • `"Bray-Curtis"` for Bray-Curtis dissimilarity. • `"Euclidean"` for generalized UniFrac dissimilarity.
`point_size`	The size of the points in the plot. Default is `1`.

Value

a PCoA plot.

Examples

```
data(exampleData)
O = exampleData$O
Y = runif(ncol(O))
pcoa_plot_continuous(O,Y,"Y")
```

`pcoa_plot_discrete` *PCoA plots for discrete variables.*

Description

PCoA plots for discrete variables.

Usage

```
pcoa_plot_discrete(
  X,
  covariate,
  title,
  R2 = TRUE,
  dissimilarity = "Bray-Curtis",
  colorset = "Set1",
  point_size = 1
)
```

Arguments

`X`	Abundance matrix. Rows represent taxa, and columns represent samples.
`covariate`	A discrete sample covariate.
`title`	The title of the graph.
`R2`	A logical variable. Whether to display the R² statistic in the subtitle. Default is `TRUE`.
`dissimilarity`	The dissimilarity type. Options include: • `"Bray-Curtis"` for Bray-Curtis dissimilarity. • `"Euclidean"` for generalized UniFrac dissimilarity.
`colorset`	The color set for visualization. Default is `"Set1"`.
`point_size`	The size of the points in the plot. Default is `1`.

Value

a PCoA plot.

Examples

```
data(exampleData)
O = exampleData$O
meta = exampleData$meta
batchid = meta$batch
pcoa_plot_discrete(O,batchid,"Batch")
```

`plot_singular_values` *Generate Singular Value Plots for Each Dataset.*

Description

This function produces singular value plots for each input dataset to assess the validity of the low-rank assumption. A rapid decay in the singular values indicates that the dataset can be effectively approximated by matrix factorization.

Usage

```
plot_singular_values(count, meta)
```

Arguments

`count`	The integrated count table of taxa by samples. The `count` parameter should be provided as either a `matrix` or a `data.frame`.
`meta`	The integrated meta table `meta` contains a column named “batch” which stores batch id.

Value

A list of ggplot objects displaying the singular values for each dataset.

Examples

```
data(exampleData)
O = exampleData$O
meta = exampleData$meta
plot_singular_values(O, meta)
```

Index