

# Package: NeighborFinder (via r-universe)

June 30, 2026

**Title** Find Neighbor Species of a Bacteria of Interest in the Human Gut Microbiota

**Version** 1.0.1

**Description** Implementation of the local approach described in Sola et al., 2026 <[doi:10.64898/2025.12.05.692507](https://doi.org/10.64898/2025.12.05.692507)> to identify companion species of a bacteria of interest. From several abundance tables of metagenomic data, 'NeighborFinder' suggests a shortlist of companion species based on the integration of results. A visualization via a network is proposed.

**License** MIT + file LICENSE

**Depends** R (>= 3.5.0)

**Imports** dplyr, GGally, glmnet, glue, igraph, magrittr, Matrix, mvtnorm, network, purrr, rlang, sna, stats, stringr, tibble, tidyr, utils

**Suggests** knitr, rmarkdown, testthat

**VignetteBuilder** knitr

**Config/fusen/version** 0.7.2

**Config/roxygen2/version** 8.0.0

**Encoding** UTF-8

**LazyData** true

**RoxygenNote** 7.3.3

**NeedsCompilation** no

**Author** Mathilde Sola [aut, cre] (ORCID: <<https://orcid.org/0009-0009-0436-5078>>), Mahendra Mariadassou [aut] (ORCID: <<https://orcid.org/0000-0003-2986-354X>>), Magali Berland [aut] (ORCID: <<https://orcid.org/0000-0002-6762-5350>>)

**Maintainer** Mathilde Sola <[mathilde.sola@inrae.fr](mailto:mathilde.sola@inrae.fr)>

**Repository** <https://cran.r-universe.dev>

**Date/Publication** 2026-06-30 12:20:32 UTC

**RemoteUrl** <https://github.com/cran/NeighborFinder>

**RemoteRef** HEAD

**RemoteSha** d2527c165e8fc295be264993463694d12c5d1524

## Contents

apply_NeighborFinder	2
apply_NF_simple	4
choose_params_values	5
compute_precision	6
compute_recall	7
cvglm_to_coeffs_by_object	7
data	9
final_step	9
find_all_module_neighbors	10
find_module_neighbors	11
get_count_table	12
graph_step	14
graphs	15
identify_module	16
intersections_network	17
intersections_table	19
mclr	20
metadata	21
module_to_node	21
new_synth_data	22
norm_data	25
prev_for_selected_nodes	26
result_example	27
simulate_by_prevalence	28
simulate_from_ecdf	29
taxo	31
test_filter	31
truth_by_prevalence	32
visualize_network	34
<b>Index</b>	<b>36</b>

---

apply\_NeighborFinder    *Apply NeighborFinder on raw data*

---

### Description

Apply NeighborFinder on raw data

### Usage

```
apply_NeighborFinder(
  data_with_annotation,
  object_of_interest,
  col_module_id,
  annotation_level,
```

```

    prev_level = 0.3,
    filtering_top = 20,
    .seed = NULL,
    ...
)

```

## Arguments

<code>data_with_annotation</code>	Dataframe. The abundance table merged with the module names. Required format: modules are the rows and samples are the columns. The first column must be the modules name (e.g. species), the second is the module ID (e.g. msp), and each subsequent column is a sample
<code>object_of_interest</code>	String. The name of the bacteria or species of interest or a key word in the functional module definition
<code>col_module_id</code>	String. The name of the column with the module names in <code>annotation_table</code>
<code>annotation_level</code>	String. The name of the column with the level to be studied. Examples: species, genus, level_1
<code>prev_level</code>	Numeric. The prevalence to be studied. Required format is decimal: 0.20 for 20% of prevalence
<code>filtering_top</code>	Numeric. The filtering top percentage to be studied. Required format is: 10 for top 10%
<code>.seed</code>	Integer. Top level RNG seed to control the generation of RNG seed for the inner loop or NULL if reproducibility is not required.
<code>...</code>	Additional arguments passed on to <code>apply_NF_simple()</code>

## Value

Dataframe. Returns results after using `apply_NeighborFinder()`: for each module ID from 'object\_of\_interest', the names of their neighbors and the corresponding coefficients calculated by `cv.glmnet()`

## Examples

```

data(data)
res_CRC_JPN <- apply_NeighborFinder(
  data$CRC_JPN[, 1:100],
  object_of_interest = "Escherichia coli",
  col_module_id = "msp_id",
  annotation_level = "species",
  .seed = 123
)

```

---

<code>apply_NF_simple</code>	<i>Apply NeighborFinder simplest version on raw data</i>
------------------------------	--

---

### Description

Apply NeighborFinder simplest version on raw data

### Usage

```
apply_NF_simple(
  data_with_annotation,
  object_of_interest,
  col_module_id,
  annotation_level,
  prev_level = 0.3,
  filtering_top = 20,
  seed = NULL,
  ...
)
```

### Arguments

<code>data_with_annotation</code>	Dataframe. The abundance table merged with the module names. Required format: modules are the rows and samples are the columns. The first column must be the modules name (e.g. species), the second is the module ID (e.g. msp), and each subsequent column is a sample
<code>object_of_interest</code>	String. The name of the bacteria or species of interest or a key word in the functional module definition
<code>col_module_id</code>	String. The name of the column with the module names in <code>annotation_table</code>
<code>annotation_level</code>	String. The name of the column with the level to be studied. Examples: species, genus, level_1
<code>prev_level</code>	Numeric. The prevalence to be studied. Required format is decimal: 0.20 for 20% of prevalence
<code>filtering_top</code>	Numeric. The filtering top percentage to be studied. Required format is: 10 for top 10%
<code>seed</code>	Numeric. The seed number, ensuring reproducibility
<code>...</code>	Additional arguments passed on to <code>cvglm_to_coeffs_by_object()</code>

### Value

Dataframe. Returns results after using `apply_NeighborFinder()`: for each module ID from 'object\_of\_interest', the names of their neighbors and the corresponding coefficients calculated by `cv.glmnet()`

**Examples**

```

data(data)
res_CRC_JPN <- apply_NF_simple(
  data$CRC_JPN[, 1:100],
  object_of_interest = "Escherichia coli",
  col_module_id = "msp_id",
  annotation_level = "species",
  seed = 20242025
)

```

---

choose\_params\_values *Render a table to give an indication of the values to choose for the prevalence level and the top filtering percentage*

---

**Description**

Render a table to give an indication of the values to choose for the prevalence level and the top filtering percentage

**Usage**

```

choose_params_values(
  data_with_annotation,
  object_of_interest,
  sample_size,
  prev_list = c(0.2, 0.3, 0.4),
  filtering_list = c(10, 20, 30),
  graph_file = NULL,
  col_module_id,
  annotation_level,
  seed = NULL
)

```

**Arguments**

**data\_with\_annotation** Dataframe. The abundance table merged with the module names. Required format: modules are the rows and samples are the columns. The first column must be the modules name (e.g. species), the second is the module ID (e.g. msp), and each subsequent column is a sample

**object\_of\_interest** String. The name of the bacteria or species of interest or a key word in the functional module definition

**sample\_size** Numeric. Number of samples in each dataset.

**prev\_list** List of numeric. The prevalences to be studied. Required format is decimal: 0.20 for 20% of prevalence

filtering_list	List of numeric. The filtering top percentages to be studied. Required format is: 10 for the top 10%
graph_file	Dataframe. The object generated by graph_step() function
col_module_id	String. The name of the column with the module names in annotation_table
annotation_level	String. The name of the column with the level to be studied. Examples: species, genus, level_1
seed	Numeric. The seed number, ensuring reproducibility

**Value**

Dataframe. Returns F1 rates before and after using apply\_NeighborFinder()

**Examples**

```

data(data)
data(graphs)
choose_params_values(
  data_with_annotation = data$CRC_JPN,
  object_of_interest = "Escherichia coli",
  sample_size = 100,
  prev_list = c(0.20, 0.30),
  filtering_list = c(10, 20),
  graph_file = graphs$CRC_JPN,
  col_module_id = "msp_id",
  annotation_level = "species",
  seed = 123
)

```

---

compute_precision	<i>Compute precision rate</i>
-------------------	-------------------------------

---

**Description**

Compute precision rate

**Usage**

```
compute_precision(true, detected)
```

**Arguments**

true	List. The one of true neighbors
detected	List. The one of detected neighbors

**Value**

Numeric. Returns the precision rate

**Examples**

```
compute_precision(c("a"), c("a", "b", "c"))
compute_precision(c("a", "b"), c("a", "c"))
```

---

compute_recall	<i>Compute recall rate</i>
----------------	----------------------------

---

**Description**

Compute recall rate

**Usage**

```
compute_recall(true, detected)
```

**Arguments**

true	List. The one of true neighbors
detected	List. The one of detected neighbors

**Value**

Numeric. Returns the recall rate

**Examples**

```
compute_recall(c("a"), c("a", "b", "c"))
compute_recall(c("a", "b"), c("a", "c"))
```

---

cvglm_to_coeffs_by_object	<i>Apply cv.glmnet() for a list of module IDs and for each prevalence level</i>
---------------------------	---

---

**Description**

Apply cv.glmnet() for a list of module IDs and for each prevalence level

**Usage**

```
cvglm_to_coeffs_by_object(
  list_dfs,
  test_module = identify_module(),
  seed = NULL,
  ...
)
```

**Arguments**

<code>list_dfs</code>	List of dataframe. A normalized dataframe
<code>test_module</code>	List of string. The module IDs
<code>seed</code>	Numeric. The seed number, ensuring reproducibility
<code>...</code>	Additional arguments passed on to <a href="#">find_all_module_neighbors()</a>

**Value**

Dataframe. Returns the module ID, its detected neighbor and the corresponding coefficient

**Examples**

```
data(data)
data(metadata)
# Simple example
normed_JPN <- norm_data(
  data$CRC_JPN,
  col_module_id = "msp_id",
  annotation_level = "species",
  prev_list = c(0.25, 0.30)
)
neighbors_JPN <- cvglm_to_coeffs_by_object(
  list_dfs = normed_JPN,
  test_module = c("msp_0030", "msp_0345"),
  seed = 20242025
)
# Example with covariate
# normed_CHN <- norm_data(
#   data$CRC_CHN,
#   col_module_id = "msp_id",
#   annotation_level = "species",
#   prev_list = c(0.25, 0.30)
# )
# neighbors_CHN <- cvglm_to_coeffs_by_object(
#   list_dfs = normed_CHN,
#   test_module = c("msp_0030", "msp_0345"),
#   seed = 20242025,
#   covar = ~study_accession,
#   meta_df = metadata$CRC_CHN,
#   sample_col = "secondary_sample_accession"
# )
```

---

data	<i>data</i>
------	-------------

---

### Description

#' @format A list of dataframes corresponding to abundance tables merges with taxonomic information.

**CRC\_JPN** dataframe with only Japanese patients diagnosed with colorectal cancer

**CRC\_CHN** dataframe with only Chinese patients diagnosed with colorectal cancer

**CRC\_EUR** dataframe with only European patients diagnosed with colorectal cancer

**CRC\_JPN\_CHN\_EUR** dataframe with patients diagnosed with colorectal cancer from the 3 previous countries

### Usage

data

### Format

An object of class `list` of length 4.

### Source

<https://entrepot.recherche.data.gouv.fr/dataset.xhtml?persistentId=doi:10.57745/7IVO3E>

---

final_step	<i>Gather lists of neighbors of true ones from the graph and detected ones from cv.glmnet()</i>
------------	---

---

### Description

Gather lists of neighbors of true ones from the graph and detected ones from `cv.glmnet()`

### Usage

```
final_step(df_truth, df_glm, robustness_step = NULL)
```

### Arguments

`df_truth` Dataframe. The one resulting from `truth_by_prevalence()`

`df_glm` Dataframe. The one resulting from `cvglm_to_coeffs_by_object()`

`robustness_step`

Boolean. When TRUE, `filtering_top` will be different from 100%, when FALSE the results from the naïve method are looked at

**Value**

Dataframe. Returns for each level of prevalence and module ID, the list of true and/or detected neighbors and the corresponding list of coefficients

**Examples**

```
# Dataframe with true neighbors
df_true <- list(
  tibble::tibble(
    node1 = c("msp_1", "msp_1", "msp_2", "msp_3"),
    node2 = c("msp_55", "msp_20", "msp_3", "msp_18"),
    prev1 = c(0.28, 0.28, 0.96, 0.75),
    prev2 = c(0.76, 0.25, 0.75, 0.60)
  ),
  tibble::tibble(
    node1 = c("msp_2", "msp_3"),
    node2 = c("msp_3", "msp_18"),
    prev1 = c(0.96, 0.75),
    prev2 = c(0.75, 0.60)
  )
) %>%
  rlang::set_names(c("0.20", "0.30"))

# Dataframe with detected neighbors
df_detected <- list(
  tibble::tibble(
    prev_level = c("0.20", "0.30", "0.30", "0.30"),
    node1 = c("msp_2", "msp_2", "msp_3", "msp_3"),
    node2 = c("msp_3", "msp_3", "msp_18", "msp_8"),
    coef = c(0.406, -0.025, 0.160, 0.005),
    filtering_top = c(100, 100, 100, 100)
  ),
  tibble::tibble()
) %>%
  rlang::set_names(c("0.20", "0.30"))
# Use final_step() to gather both
neighbors <- final_step(df_true, df_detected, robustness_step = FALSE)
```

---

find\_all\_module\_neighbors

*Apply cv.glmnet() for a list of module IDs*

---

**Description**

Apply cv.glmnet() for a list of module IDs

**Usage**

```
find_all_module_neighbors(df, test_module, seed = NULL, ...)
```

**Arguments**

df	Dataframe. A normalized dataframe
test_module	List of string. The module IDs
seed	Numeric. The seed number, ensuring reproducibility
...	Additional arguments passed on to <code>find_module_neighbors()</code>

**Value**

Dataframe. Returns the module ID, its detected neighbor and the corresponding coefficient

**Examples**

```

data(data)
data(metadata)
# Simple example
x <- norm_data(data$CRC_JPN, 0.30, annotation_level = "species")[[1]]
neighbors_JPN <- find_all_module_neighbors(
  df = x,
  test_module = c("msp_0030", "msp_0345"),
  seed = 20242025
)
# Example with covariate
# x <- norm_data(data$CRC_CHN, 0.30, annotation_level = "species")[[1]]
# neighbors_CHN <- find_all_module_neighbors(
#   df = x,
#   test_module = c("msp_0030", "msp_0345"),
#   seed = 20242025,
#   covar = ~study_accession,
#   meta_df = metadata$CRC_CHN,
#   sample_col = "secondary_sample_accession"
# )

```

---

`find_module_neighbors` *Apply cv.glmnet() for a given module ID*

---

**Description**

Apply `cv.glmnet()` for a given module ID

**Usage**

```

find_module_neighbors(
  df,
  module,
  seed = NULL,
  covar = NULL,
  meta_df = NULL,
  sample_col = NULL
)

```

**Arguments**

df	Dataframe. A normalized dataframe
module	String. The module ID name
seed	Numeric. The seed number, ensuring reproducibility
covar	String or formula. Formula or the name of the column of the covariate in the metadata table. Note that "study_accession" is equivalent to ~study_accession
meta_df	Dataframe. The dataframe giving metadata information
sample_col	String. The name of the column in metadata indicating the sample names, it should be consistent with the colnames of 'df'

**Value**

Dataframe. Returns the module ID, its detected neighbor and the corresponding coefficient

**Examples**

```

data(data)
data(metadata)
# Simple example
x <- norm_data(data$CRC_JPN, 0.30)[[1]]
neighbors_JPN <- find_module_neighbors(
  df = x,
  module = "msp_0030",
  seed = 20242025
)
# Example with covariate
# x <- norm_data(data$CRC_CHN, 0.30)[[1]]
# neighbors_CHN <- find_module_neighbors(
#   df = x,
#   module = "msp_0030",
#   seed = 20242025,
#   covar = ~study_accession,
#   meta_df = metadata$CRC_CHN,
#   sample_col = "secondary_sample_accession"
# )

```

---

get\_count\_table

*Conversion to count table function with prevalence filter*


---

**Description**

Conversion to count table function with prevalence filter

**Usage**

```
get_count_table(  
  abund.path = NULL,  
  abund.table = NULL,  
  sample.id = NULL,  
  prev.min,  
  verbatim = TRUE,  
  msp = NULL  
)
```

**Arguments**

abund.path	String. Path to the abundance table
abund.table	Dataframe. Abundance table, it should have the bacterial species names as first column
sample.id	String vector. IDs of samples to keep in the final table
prev.min	Numeric. The value is between 0 and 1 and corresponds to the minimal prevalence threshold of bacterial species to keep in the final table
verbatim	Boolean. Controls verbosity
msp	String vector. It indicates bacterial species names, if they are not specified in the abundance table first column

**Value**

A list containing

data:	the final count table (tibble)
prevalences:	a tibble gathering the prevalence of each bacterial species

**Source**

This function is adapted from the same name function in OneNet package (version 0.3.1), which is licensed under the MIT License. Original copyright (c) 2021-2024 INRAE.

The MIT License text for the original package is as follows:

---

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR

IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

---

## Examples

```

tiny_data <- data.frame(
  msp_name = c("msp_1", "msp_2", "msp_3", "msp_4"),
  SAMPLE1 = c(0, 1.328425e-06, 0, 1.527688e-07),
  SAMPLE2 = c(1.251707e-07, 1.251707e-07, 3.985320e-07, 0),
  SAMPLE3 = c(0, 0, 4.926046e-09, 5.626392e-06),
  SAMPLE4 = c(0, 0, 2.98320e-05, 0)
)
# Applying a prevalence filter of 30% on the new count_table
count_table <- get_count_table(
  abund.table = tiny_data,
  sample.id = colnames(tiny_data),
  prev.min = 0.3
)

```

---

graph\_step

*Generate a graph with a "cluster-like" structure, only needed for simulation purposes*

---

## Description

Generate a graph with a "cluster-like" structure, only needed for simulation purposes

## Usage

```

graph_step(
  data_with_annotation,
  col_module_id,
  annotation_level,
  seed = 10010,
  data_type = "shotgun",
  ...
)

```

## Arguments

data\_with\_annotation

Dataframe. The abundance table merged with the module names. Required format: modules are the rows and samples are the columns. The first column must be the modules name (e.g. species), the second is the module ID (e.g. msp), and each subsequent column is a sample

col\_module\_id String. The name of the column with the module names in the annotation table

annotation\_level String. The name of the column with the level to be studied. Examples: species, genus, level\_1

seed Numeric. Seed number for data generation (new\_synth\_data)

data\_type String. Enables the treatment of 16S data with "16S", default value is "shotgun"

... Additional arguments passed on to the generator\_graph() routine of [new\\_synth\\_data\(\)](#)

### Value

Dataframe. The dataframe is composed of 0 and 1 corresponding to the existence of edges on the graph.

### Examples

```

tiny_data <- data.frame(
  species = c(
    "One bacteria",
    "One bacterium L",
    "One bacterium G",
    "Two bact",
    "Three bact A",
    "Three bact B"
  ),
  msp_name = c("msp_1", "msp_2", "msp_3", "msp_4", "msp_5", "msp_6"),
  SAMPLE1 = c(0, 1.328425e-06, 0, 1.527688e-07, 0, 0),
  SAMPLE2 = c(1.251707e-07, 0, 3.985320e-07, 0, 1.33607e-04, 0.8675e-03),
  SAMPLE3 = c(0, 0, 4.926046e-09, 5.626392e-06, 0, 0.662e-03),
  SAMPLE4 = c(0, 0, 2.98320e-05, 0, 1.275e-04, 0),
  SAMPLE5 = c(0.0976, 0.9862, 2.98320e-03, 0, 3.9754e-03, 0),
  SAMPLE6 = c(0.26417e-06, 0, 1.0077e-05, 3.983320e-08, 0, 0)
)

tiny_graph <- graph_step(
  tiny_data,
  col_module_id = "msp_name",
  annotation_level = "species",
  seed = 20242025
) %>%
  suppressWarnings()

```

---

graphs

*graphs*

---

### Description

graphs

**Usage**

graphs

**Format**

A list of dataframes corresponding to graphs based on synthetic data.

**CRC\_JPN** graph corresponding to data with only Japanese patients diagnosed with colorectal cancer

**CRC\_JPN\_CHN\_EUR** graph corresponding to data gathering Japanese, Chinese and European patients diagnosed with colorectal cancer

---

identify_module	<i>List the modules corresponding to a given object of interest</i>
-----------------	---

---

**Description**

List the modules corresponding to a given object of interest

**Usage**

```
identify_module(
  object_of_interest,
  annotation_table,
  col_module_id,
  annotation_level = "species"
)
```

**Arguments**

**object\_of\_interest** String. The name of the bacteria or species of interest or a key word in the functional module definition

**annotation\_table** Dataframe. The dataframe gathering the taxonomic or functional module correspondence information

**col\_module\_id** String. The name of the column with the module names in the annotation table

**annotation\_level** String. The name of the column with the level to be studied. Examples: species, genus, level\_1. Default value is set to the species level

**Value**

List of string. The module names of the corresponding object of interest

**Examples**

```
df_taxo <- data.frame(
  msp_name = c("msp_1", "msp_2", "msp_3", "msp_4"),
  genus = c("One", "One", "One", "Two"),
  species = c("One bacteria", "One bacterium L", "One bacterium G", "Two bact")
)
identify_module(
  object_of_interest = "bacterium",
  annotation_table = df_taxo,
  col_module_id = "msp_name",
  annotation_level = "species"
)
identify_module(
  object_of_interest = "One",
  annotation_table = df_taxo,
  col_module_id = "msp_name",
  annotation_level = "species"
)
```

---

intersections\_network *Display the intersection network from 2 or more datasets*

---

**Description**

Display the intersection network from 2 or more datasets

**Usage**

```
intersections_network(
  res_list,
  threshold,
  annotation_table,
  col_module_id,
  annotation_level,
  object_of_interest,
  annotation_option = FALSE,
  node_size = 12,
  label_size = 4,
  edge_label_size = 2,
  object_color = "cadetblue2",
  seed = NULL
)
```

**Arguments**

res_list	List of dataframes. The results from apply_NeighborFinder() on several datasets
threshold	Numeric. Integer corresponding to the minimum number of datasets in which you want neighbors to have been found

<code>annotation_table</code>	Dataframe. The dataframe gathering the taxonomic or functional module correspondence information
<code>col_module_id</code>	String. The name of the column with the module names in <code>annotation_table</code>
<code>annotation_level</code>	String. The name of the column with the level to be studied. Examples: species, genus, level_1
<code>object_of_interest</code>	String. The name of the bacteria or species of interest or a key word in the functional module definition
<code>annotation_option</code>	Boolean. Default value is False. If True: labels on nodes become module names instead of module IDs
<code>node_size</code>	Numeric. The parameter to adjust size of nodes
<code>label_size</code>	Numeric. The parameter to adjust size of labels
<code>edge_label_size</code>	Numeric. The parameter to adjust the size of edge labels
<code>object_color</code>	String. The name of the color to differentiate the nodes corresponding to 'object_of_interest' from the other module IDs
<code>seed</code>	Numeric. The seed number, ensuring reproducibility

**Value**

Network. Visualization of NeighborFinder results from several datasets. Edge color encodes the mean coefficient sign: green if positive, red if negative.

**Examples**

```

data(taxo)
data(result_example)
intersections_network(
  res_list = list(
    result_example$res_CRC_JPN,
    result_example$res_CRC_CHN,
    result_example$res_CRC_EUR
  ),
  taxo,
  threshold = 2,
  "Escherichia coli",
  col_module_id = "msp_id",
  annotation_level = "species",
  label_size = 7,
  edge_label_size = 4,
  node_size = 15,
  annotation_option = TRUE,
  seed = 3
)

```

---

intersections\_table    *Display the intersection table summarizing the results from 2 or more datasets*

---

### Description

Display the intersection table summarizing the results from 2 or more datasets

### Usage

```
intersections_table(
  res_list,
  threshold,
  annotation_table,
  col_module_id,
  annotation_level,
  object_of_interest
)
```

### Arguments

**res\_list**            List of dataframes. The results from apply\_NeighborFinder() on several datasets

**threshold**          Numeric. Integer corresponding to the minimum number of datasets in which you want neighbors to have been found

**annotation\_table**    Dataframe. The dataframe gathering the taxonomic or functional module correspondence information

**col\_module\_id**      String. The name of the column with the module names in annotation\_table

**annotation\_level**    String. The name of the column with the level to be studied. Examples: species, genus, level\_1

**object\_of\_interest**    String. The name of the bacteria or species of interest or a key word in the functional module definition

### Value

Dataframe. Table gathering the intersection of NeighborFinder results from several datasets. The column 'datasets' indicates the datasets in which the same neighbor has been found, the column 'intersections' indicates the number of datasets in which the same neighbor has been found

### Examples

```
data(taxo)
data(result_example)
intersections_table(
  res_list = list(
```

```

    result_example$res_CRC_JPN,
    result_example$res_CRC_CHN,
    result_example$res_CRC_EUR
  ),
  threshold = 2,
  taxo,
  col_module_id = "msp_id",
  annotation_level = "species",
  "Escherichia coli"
)

```

---

mclr	<i>Modified central log ratio (mclr) transformation extracted from the SPRING package</i>
------	---

---

### Description

Modified central log ratio (mclr) transformation extracted from the SPRING package

### Usage

```
mclr(dat, base = exp(1), tol = 1e-16, eps = NULL, atleast = 1)
```

### Arguments

dat	raw count data or compositional data (n by p) does not matter.
base	exp(1) for natural log
tol	tolerance for checking zeros
eps	epsilon in eq (2) of the paper "Yoon, Gaynanova, Müller (2019), Frontiers in Genetics". positive shifts to all non-zero compositions. Refer to the paper for more details. eps = absolute value of minimum of log ratio counts plus c.
atleast	default value is 1. Constant c which ensures all nonzero values to be strictly positive. default is 1.

### Details

This function implements the mclr normalization introduced in Yoon G, Gaynanova I and Müller CL (2019) Microbial Networks in SPRING - Semi-parametric Rank-Based Correlation and Partial Correlation Estimation for Quantitative Microbiome Data. Front. Genet. 10:516. [doi:10.3389/fgene.2019.00516](https://doi.org/10.3389/fgene.2019.00516)

### Value

mclr returns a data matrix of the same dimension with input data matrix.

---

metadata	<i>metadata</i>
----------	-----------------

---

**Description**

#' @format A list of dataframes corresponding to metadata, giving more information on each patient.

**CRC\_JPN** dataframe with only Japanese patients diagnosed with colorectal cancer

**CRC\_CHN** dataframe with only Chinese patients diagnosed with colorectal cancer

**CRC\_EUR** dataframe with only European patients diagnosed with colorectal cancer

**CRC\_JPN\_CHN\_EUR** dataframe with patients diagnosed with colorectal cancer from the 3 previous countries

**Usage**

```
metadata
```

**Format**

An object of class `list` of length 4.

**Source**

<https://entrepot.recherche.data.gouv.fr/dataset.xhtml?persistentId=doi:10.57745/7IVO3E>

---

module_to_node	<i>Correspondence between the module ID (msp or functional module) and its name (bacteria or function)</i>
----------------	--

---

**Description**

Correspondence between the module ID (msp or functional module) and its name (bacteria or function)

**Usage**

```
module_to_node(
  module,
  annotation_table,
  col_module_id = "msp_name",
  annotation_level
)
```

**Arguments**

module	String. The name of the biological object (msp or functional module), can be a single one or a list
annotation_table	Dataframe. The dataframe gathering the taxonomic or functional module correspondence information
col_module_id	String. The name of the column with the module names in the annotation table
annotation_level	String. The name of the column with the level to be studied. Examples: species, genus, level_1

**Value**

Dictionary. The name of the module, can be a single one or a list

**Examples**

```
df_taxo <- data.frame(
  msp_name = c("msp_1", "msp_2", "msp_3", "msp_4"),
  genus = c("A", "B", "C", "D"),
  species = c("A a", "B a", "C c", "D b")
)
# Correspondence for one specific msp
module_to_node(
  "msp_1",
  annotation_table = df_taxo,
  col_module_id = "msp_name",
  annotation_level = "species"
)
# or for several msps
module_to_node(
  c("msp_1", "msp_3", "msp_4"),
  annotation_table = df_taxo,
  col_module_id = "msp_name",
  annotation_level = "species"
)
# and if one msp is repeated
module_to_node(
  c("msp_1", "msp_1", "msp_2"),
  annotation_table = df_taxo,
  col_module_id = "msp_name",
  annotation_level = "genus"
)
```

---

new\_synth\_data

*Simulate data from some empirical count dataset with a "cluster-like" structure*

---

**Description**

Simulate data from some empirical count dataset with a "cluster-like" structure

**Usage**

```
new_synth_data(
  real_data,
  graph_type = "cluster",
  must_connect = TRUE,
  graph = NULL,
  n = 300,
  seed = NULL,
  r = 50,
  dens = 4,
  k = 3,
  verbatim = TRUE,
  signed = FALSE
)
```

**Arguments**

real_data	Matrix. Empirical count table
graph_type	String. Structure type for the conditional dependency structure. Here only "cluster" was kept, see EMtree package for more options
must_connect	Boolean. TRUE to force the output graph to be connected
graph	Boolean. Optional graph to be used, must have rownames and colnames and reference all features from real_data
n	Numeric. Number of samples to simulate
seed	Numeric. Seed number for data generation (rmvnorm)
r	Numeric. For cluster structure, controls the within/between ratio connection probability
dens	Numeric. Graph density (for cluster graphs) or edges probability (for erdős-renyi graphs)
k	Numeric. For cluster structure, number of groups
verbatim	Boolean. Controls verbosity
signed	Boolean. TRUE for simulating both positive and negative partial correlations. Default is to FALSE, which implies only negative partial correlations

**Value**

List. Containing the simulated discrete counts, the corresponding true partial correlation matrix from the latent Gaussian layer of the model and the original graph structure that was used

## Source

This function is adapted from the same name function in OneNet package (version 0.3.1), which is licensed under the MIT License. Original copyright (c) 2021-2024 INRAE.

The MIT License text for the original package is as follows:

---

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

---

## Examples

```

tiny_data <- data.frame(
  species = c(
    "One bacteria",
    "One bacterium L",
    "One bacterium G",
    "Two bact",
    "Three bact A",
    "Three bact B"
  ),
  msp_name = c("msp_1", "msp_2", "msp_3", "msp_4", "msp_5", "msp_6"),
  SAMPLE1 = c(0, 1.328425e-06, 0, 1.527688e-07, 0, 0),
  SAMPLE2 = c(1.251707e-07, 0, 3.985320e-07, 0, 1.33607e-04, 0.8675e-03),
  SAMPLE3 = c(0, 0, 4.926046e-09, 5.626392e-06, 0, 0.662e-03),
  SAMPLE4 = c(0, 0, 2.98320e-05, 0, 1.275e-04, 0),
  SAMPLE5 = c(0.0976, 0.9862, 2.98320e-03, 0, 3.9754e-03, 0),
  SAMPLE6 = c(0.26417e-06, 0, 1.0077e-05, 3.983320e-08, 0, 0)
)

count_table <- get_count_table(
  abund.table = tiny_data %>% dplyr::select(-species),
  sample.id = colnames(tiny_data),
  prev.min = 0.1
)

```

```

tiny_graph <- graph_step(
  tiny_data,
  col_module_id = "msp_name",
  annotation_level = "species",
  seed = 20242025
) %>%
  suppressWarnings()
sim_data <- new_synth_data(
  count_table$data,
  n = 50,
  graph = as.matrix(tiny_graph %>% dplyr::select(-species)),
  verbatim = FALSE,
  seed = 20242025
) %>%
  suppressWarnings()

```

---

norm\_data

*Normalize data and filters it by prevalence level*


---

## Description

Normalize data and filters it by prevalence level

## Usage

```

norm_data(
  data_with_annotation,
  col_module_id,
  prev_list = c(0.3),
  annotation_level,
  data_type = "shotgun"
)

```

## Arguments

data_with_annotation	Dataframe. The abundance table merged with the module names. Required format: modules are the rows and samples are the columns. The first column must be the modules name (e.g. species), the second is the module ID (e.g. msp), and each subsequent column is a sample
col_module_id	String. The name of the column with the module names in annotation_table
prev_list	Vector of numeric. The prevalences to be studied. Required format is decimal: 0.20 for 20% of prevalence
annotation_level	String. Annotation level to aggregate the taxa
data_type	String. Enables the treatment of 16S data with "16S", default value is "shotgun"

**Value**

List of dataframes. Each element of the list corresponds to a normalized 'data\_with\_annotation', by level of prevalence

**Examples**

```

tiny_data <- data.frame(
  species = c("One bacteria", "One bacterium L", "One bacterium G", "Two bact"),
  msp_name = c("msp_1", "msp_2", "msp_3", "msp_4"),
  SAMPLE1 = c(0, 1.328425e-06, 0, 1.527688e-07),
  SAMPLE2 = c(1.251707e-07, 1.251707e-07, 3.985320e-07, 0),
  SAMPLE3 = c(0, 0, 4.926046e-09, 5.626392e-06),
  SAMPLE4 = c(0, 0, 2.98320e-05, 0)
)

tiny_normed <- norm_data(
  tiny_data,
  col_module_id = "msp_name",
  annotation_level = "species",
  prev_list = c(0.20, 0.30)
)

```

---

```
prev_for_selected_nodes
```

*Extract edges in graph involving any module in object\_of\_interest set*

---

**Description**

Extract edges in graph involving any module in object\_of\_interest set

**Usage**

```

prev_for_selected_nodes(
  data_with_annotation,
  graph_file,
  col_module_id,
  annotation_level,
  object_of_interest = NULL
)

```

**Arguments**

`data_with_annotation` Dataframe. The abundance table merged with the module names. Required format: modules are the rows and samples are the columns. The first column must be the modules name (e.g. species), the second is the module ID (e.g. msp), and each subsequent column is a sample

`graph_file` Dataframe. The object generated by graph\_step() function

col\_module\_id String. The name of the column with the module names in annotation\_table

annotation\_level String. The name of the column with the level to be studied. Examples: species, genus, level\_1

object\_of\_interest String. The name of the bacteria or species of interest or a key word in the functional module definition

### Value

Dataframe. The dataframe of edges in the graph involving modules corresponding to object\_of\_interest and their corresponding prevalences.

### Examples

```

tiny_data <- data.frame(
  species = c("One bacteria", "One bacterium L", "One bacterium G", "Two bact"),
  msp_name = c("msp_1", "msp_2", "msp_3", "msp_4"),
  SAMPLE1 = c(0, 1.328425e-06, 0, 1.527688e-07),
  SAMPLE2 = c(1.251707e-07, 1.251707e-07, 3.985320e-07, 0),
  SAMPLE3 = c(0, 0, 4.926046e-09, 5.626392e-06),
  SAMPLE4 = c(0, 0, 2.98320e-05, 0)
)
tiny_graph <- graph_step(
  tiny_data,
  col_module_id = "msp_name",
  annotation_level = "species",
  seed = 20242025
) %>%
  suppressWarnings()

tiny_truth <- prev_for_selected_nodes(
  tiny_data,
  tiny_graph,
  col_module_id = "msp_name",
  annotation_level = "species",
  object_of_interest = "bacterium"
)

```

---

result\_example

*result\_example*

---

### Description

result\_example

### Usage

result\_example

**Format**

A list of dataframes corresponding to apply\_NeighborFinder() results.

**res\_CRC\_JPN** edge table of Escherichia coli neighbors with only Japanese patients diagnosed with colorectal cancer

**res\_CRC\_CHN** edge table of Escherichia coli neighbors with only Chinese patients diagnosed with colorectal cancer

**res\_CRC\_EUR** edge table of Escherichia coli neighbors with only European patients diagnosed with colorectal cancer

---

simulate\_by\_prevalence

*List the simulated count tables by level of prevalence*

---

**Description**

List the simulated count tables by level of prevalence

**Usage**

```
simulate_by_prevalence(
  data_with_annotation,
  prev_list,
  graph_file = NULL,
  col_module_id,
  annotation_level,
  sample_size = 500,
  seed = NULL,
  verbatim = FALSE,
  data_type = "shotgun"
)
```

**Arguments**

data_with_annotation	Dataframe. The abundance table merged with the module names. Required format: modules are the rows and samples are the columns. The first column must be the modules name (e.g. species), the second is the module ID (e.g. msp), and each subsequent column is a sample
prev_list	List of numeric. The prevalences to be studied. Required format is decimal: 0.20 for 20% of prevalence.
graph_file	Dataframe. The object generated by graph_step() function
col_module_id	String. The name of the column with the module names in data_with_annotation
annotation_level	String. The name of the column with the level to be studied. Examples: species, genus, level_1

sample_size	Numeric. The size to be considered, the value of 500 is recommended
seed	Numeric. The seed number, ensuring reproducibility
verbatim	Boolean. Controls verbosity
data_type	String. Enables the treatment of 16S data with "16S", default value is "shotgun"

### Value

List of dataframes. Each element of the list corresponds to a level of prevalence and is a simulated abundance table

### Examples

```

tiny_data <- data.frame(
  species = c("One bacteria", "One bacterium L", "One bacterium G", "Two bact"),
  msp_name = c("msp_1", "msp_2", "msp_3", "msp_4"),
  SAMPLE1 = c(0, 1.328425e-06, 0, 1.527688e-07),
  SAMPLE2 = c(1.251707e-07, 1.251707e-07, 3.985320e-07, 0),
  SAMPLE3 = c(0, 0, 4.926046e-09, 5.626392e-06),
  SAMPLE4 = c(0, 0, 2.98320e-05, 0)
)

tiny_graph <- graph_step(
  tiny_data,
  col_module_id = "msp_name",
  annotation_level = "species",
  seed = 20242025
) %>%
  suppressWarnings()

tiny_sims <- simulate_by_prevalence(
  tiny_data,
  prev_list = c(0.20, 0.30),
  graph_file = tiny_graph,
  col_module_id = "msp_name",
  annotation_level = "species",
  sample_size = 500,
  seed = 20242025
)

```

---

simulate_from_ecdf	<i>Simulate data Generates synthetic count data based on empirical cumulative distribution (ecdf) of real count data</i>
--------------------	--

---

### Description

Simulate data Generates synthetic count data based on empirical cumulative distribution (ecdf) of real count data

**Usage**

```
simulate_from_ecdf(real_data, Sigma, n, seed = NULL, verbatim = FALSE)
```

**Arguments**

<code>real_data</code>	Matrix. Contains real count data of size $n$ by $p$
<code>Sigma</code>	Matrix. Covariance structure of size $p$ by $p$
<code>n</code>	Numeric. Number of samples
<code>seed</code>	Numeric. Seed number for data generation
<code>verbatim</code>	Boolean. If TRUE: iteration and index calculation for each step are printed out

**Value**

Matrix. The vector from the upper triangular part of `A.mat`

**Source**

This function is adapted from the same name function in OneNet package (version 0.3.1), which is licensed under the MIT License. Original copyright (c) 2021-2024 INRAE.

The MIT License text for the original package is as follows:

---

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

---

---

taxo	<i>taxo</i>
------	-------------

---

**Description**

#' @format A dataframe with 2537 msp (rows) and 4 columns:

**msp\_id** string  
**species** string  
**genus** string  
**catalogue** string indicating gut and/or oral

**Usage**

taxo

**Format**

An object of class `data.frame` with 2537 rows and 4 columns.

**Source**

<https://entrepot.recherche.data.gouv.fr/dataset.xhtml?persistentId=doi:10.57745/7IVO3E>

---

test_filter	<i>Render a table gathering precision and recall rates before and after filtering on coefficient values</i>
-------------	---

---

**Description**

Render a table gathering precision and recall rates before and after filtering on coefficient values

**Usage**

```
test_filter(df_before, df_after, prevs = NULL)
```

**Arguments**

**df\_before** Dataframe. The one of true neighbors  
**df\_after** Dataframe. The one of detected neighbors  
**prevs** List of numeric. The prevalences to be studied. Required format is decimal: 0.20 for 20% of prevalence

**Value**

Dataframe. Returns the precision and recall rates before and after the modification

**Examples**

```

# Dataframe with true neighbors
list_true <- list(
  tibble::tibble(
    node1 = c("msp_1", "msp_1", "msp_2", "msp_3"),
    node2 = c("msp_55", "msp_20", "msp_3", "msp_18"),
    prev1 = c(0.28, 0.28, 0.96, 0.75),
    prev2 = c(0.76, 0.25, 0.75, 0.60)
  ),
  tibble::tibble(
    node1 = c("msp_2", "msp_3"),
    node2 = c("msp_3", "msp_18"),
    prev1 = c(0.96, 0.75),
    prev2 = c(0.75, 0.60)
  )
) %>%
  rlang::set_names(c("0.20", "0.30"))

# Dataframes with detected neighbors
list_detected <- list(
  tibble::tibble(
    prev_level = c("0.20", "0.30", "0.30", "0.30"),
    node1 = c("msp_2", "msp_2", "msp_3", "msp_3"),
    node2 = c("msp_3", "msp_3", "msp_18", "msp_8"),
    coef = c(0.406, -0.025, 0.160, 0.005),
    filtering_top = c(100, 100, 100, 100)
  ),
  tibble::tibble()
) %>%
  rlang::set_names(c("0.20", "0.30"))
list_detected2 <- list(
  tibble::tibble(
    prev_level = c("0.20", "0.20"),
    node1 = c("msp_2", "msp_3"),
    node2 = c("msp_3", "msp_18"),
    coef = c(0.160, 0.005),
    filtering_top = c(100, 100)
  ),
  tibble::tibble()
) %>%
  rlang::set_names(c("0.20", "0.30"))
# Use final_step() to gather both
neighbors <- final_step(list_true, list_detected, robustness_step = FALSE)
neighbors2 <- final_step(list_true, list_detected2, robustness_step = FALSE) %>%
  dplyr::mutate(filtering_top = 10)
# Calculate scores
scores <- test_filter(neighbors, neighbors2)

```

**Description**

Give true neighbors by level of prevalence

**Usage**

```
truth_by_prevalence(edge_table, prev_list)
```

**Arguments**

edge_table	Dataframe. The result of prev_for_selected_nodes()
prev_list	List of numeric. The prevalences to be studied. Required format is decimal: 0.20 for 20% of prevalence

**Value**

List of dataframes. Each element of the list corresponds to a dataframe of true edges by level of prevalence

**Examples**

```

tiny_data <- data.frame(
  species = c("One bacteria", "One bacterium L", "One bacterium G", "Two bact"),
  msp_name = c("msp_1", "msp_2", "msp_3", "msp_4"),
  SAMPLE1 = c(0, 1.328425e-06, 0, 1.527688e-07),
  SAMPLE2 = c(1.251707e-07, 1.251707e-07, 3.985320e-07, 0),
  SAMPLE3 = c(0, 0, 4.926046e-09, 5.626392e-06),
  SAMPLE4 = c(0, 0, 2.98320e-05, 0)
)

tiny_graph <- graph_step(
  tiny_data,
  col_module_id = "msp_name",
  annotation_level = "species",
  seed = 20242025
) %>%
  suppressWarnings()

tiny_truth <- prev_for_selected_nodes(
  tiny_data,
  tiny_graph,
  col_module_id = "msp_name",
  annotation_level = "species",
  object_of_interest = "bacterium"
)

tiny_true_edges <- truth_by_prevalence(tiny_truth, c(0.20, 0.30))

```

---

visualize_network	<i>Display network after applying NeighborFinder</i>
-------------------	--

---

## Description

Display network after applying NeighborFinder

## Usage

```
visualize_network(
  res_NeighborFinder,
  annotation_table,
  col_module_id,
  annotation_level,
  object_of_interest,
  annotation_option = FALSE,
  node_size = 12,
  label_size = 4,
  object_color = "cadetblue2",
  seed = NULL
)
```

## Arguments

<code>res_NeighborFinder</code>	Dataframe. The result from <code>apply_NeighborFinder()</code>
<code>annotation_table</code>	Dataframe. The dataframe gathering the taxonomic or functional module correspondence information
<code>col_module_id</code>	String. The name of the column with the module names in <code>annotation_table</code>
<code>annotation_level</code>	String. The name of the column with the level to be studied. Examples: <code>species</code> , <code>genus</code> , <code>level_1</code>
<code>object_of_interest</code>	String. The name of the bacteria or species of interest or a key word in the functional module definition
<code>annotation_option</code>	Boolean. Default value is <code>False</code> . If <code>True</code> : labels on nodes become module names instead of module IDs
<code>node_size</code>	Numeric. The parameter to adjust size of nodes
<code>label_size</code>	Numeric. The parameter to adjust size of labels
<code>object_color</code>	String. The name of the color to differentiate the nodes corresponding to <code>'object_of_interest'</code> from the other module IDs
<code>seed</code>	Numeric. The seed number, ensuring reproducibility

**Value**

Network. Visualization of NeighborFinder results. Edge color encodes coefficient sign: green if positive, red if negative; edge width encodes magnitude.

**Examples**

```
data(taxo)
visualize_network(
  result_example$res_CRC_JPN,
  taxo,
  object_of_interest = "Escherichia coli",
  col_module_id = "msp_id",
  annotation_level = "species",
  label_size = 5
)
# #With species names instead of msp names
# visualize_network(
#   result_example$res_CRC_JPN,
#   taxo,
#   object_of_interest = "Escherichia coli",
#   col_module_id = "msp_id",
#   annotation_level = "species",
#   label_size = 5,
#   annotation_option = TRUE,
#   seed = 2
# )
# #With esthetic changes
# visualize_network(
#   result_example$res_CRC_JPN,
#   taxo,
#   object_of_interest = "Escherichia coli",
#   col_module_id = "msp_id",
#   annotation_level = "species",
#   annotation_option = TRUE,
#   node_size = 15,
#   label_size = 6,
#   object_color = "orange",
#   seed = 2
# )
```

# Index

## \* datasets

- data, [9](#)
- graphs, [15](#)
- metadata, [21](#)
- result\_example, [27](#)
- taxo, [31](#)

apply\_NeighborFinder, [2](#)  
apply\_NF\_simple, [4](#)  
apply\_NF\_simple(), [3](#)

choose\_params\_values, [5](#)  
compute\_precision, [6](#)  
compute\_recall, [7](#)  
cvglm\_to\_coeffs\_by\_object, [7](#)  
cvglm\_to\_coeffs\_by\_object(), [4](#)

data, [9](#)

final\_step, [9](#)  
find\_all\_module\_neighbors, [10](#)  
find\_all\_module\_neighbors(), [8](#)  
find\_module\_neighbors, [11](#)  
find\_module\_neighbors(), [11](#)

get\_count\_table, [12](#)  
graph\_step, [14](#)  
graphs, [15](#)

identify\_module, [16](#)  
intersections\_network, [17](#)  
intersections\_table, [19](#)

mclr, [20](#)  
metadata, [21](#)  
module\_to\_node, [21](#)

new\_synth\_data, [22](#)  
new\_synth\_data(), [15](#)  
norm\_data, [25](#)

prev\_for\_selected\_nodes, [26](#)

result\_example, [27](#)

simulate\_by\_prevalence, [28](#)  
simulate\_from\_ecdf, [29](#)

taxo, [31](#)  
test\_filter, [31](#)  
truth\_by\_prevalence, [32](#)

visualize\_network, [34](#)