Package ‘ulrb’
June 18, 2024

Type Package
Title Unsupervised Learning Based Definition of Microbial Rare Biosphere
Version 0.1.5
Maintainer Francisco Pascoal <fpascoal1996@gmail.com>
Date 2024-06-18
Description A tool to define rare biosphere. ‘ulrb’ solves the problem of the
definition of rarity by replacing arbitrary thresholds with an unsupervised machine
learning algorithm (partitioning around medoids, or k-medoids). This algorithm
works for any type of microbiome data, provided there is a species abundance table.
For validation of this method to different species abundance tables see Pascoal et al, 2024 (in peer-
review).
This method also works for non-microbiome data.
License GPL (>= 3)
Encoding UTF-8
LazyData true
Depends R (>= 2.10)
Suggests knitr, rmarkdown, stringr, testthat (>= 3.0.0), vegan
Config/testthat/edition 3
RoxygenNote 7.3.1
Imports cluster, dplyr, ggplot2, purrr, rlang, stats, tidyr,
clusterSim, gridExtra
VignetteBuilder knitr
URL https://pascoalf.github.io/ulrb/
BugReports https://github.com/pascoalf/ulrb/issues
NeedsCompilation no
Author Francisco Pascoal [aut, cre] (<https://orcid.org/0000-0003-2315-358X>),
Paula Branco [aut] (<https://orcid.org/0000-0002-9917-3694>),
Luís Torgo [aut] (<https://orcid.org/0000-0002-6892-8871>),
Rodrigo Costa [aut] (<https://orcid.org/0000-0002-5932-4101>),
Catarina Magalhães [aut] (<https://orcid.org/0000-0001-9576-2398>)
# ulrb-package

**ulrb**: *Unsupervised Learning Based Definition of Microbial Rare Biosphere*

## Description

The R package *ulrb* stands for Unsupervised Machine Learning definition of the Rare Biosphere. As the name suggests, it applies unsupervised learning principles to define the rare biosphere.

More specifically, the partitioning around medoids (k-medoids) algorithm is used to divide phylogenetic units (ASVs, OTUs, Species, ...) within a microbial community (usually, a sample) into clusters. The clusters are then ordered based on a user-defined classification vector. By default, our method classifies all phylogenetic units in one of these: “rare”, “undetermined” or “abundant”. In alternative, we provide functions to help the user decide the number of clusters and we also provide a fully automated option. Besides clustering, we have functions to help you evaluate the clustering quality (e.g. silhouette scores).

For detailed theory behind our reasoning for this definition of the microbial rare biosphere, results and applications, see our paper Pascoal et al., 2023 (in preparation). For more details on the R functions used and data wrangling please see the package documentation.

## Details

```r
Contents

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ulrb-package</td>
<td>2</td>
</tr>
<tr>
<td>check_avgSil</td>
<td>4</td>
</tr>
<tr>
<td>check_CH</td>
<td>6</td>
</tr>
<tr>
<td>check_DB</td>
<td>9</td>
</tr>
<tr>
<td>define_rb</td>
<td>11</td>
</tr>
<tr>
<td>evaluate_k</td>
<td>15</td>
</tr>
<tr>
<td>evaluate_sample_k</td>
<td>17</td>
</tr>
<tr>
<td>nice</td>
<td>18</td>
</tr>
<tr>
<td>nice_env</td>
<td>19</td>
</tr>
<tr>
<td>nice_raw</td>
<td>20</td>
</tr>
<tr>
<td>nice_tidy</td>
<td>22</td>
</tr>
<tr>
<td>plot_ulrb</td>
<td>23</td>
</tr>
<tr>
<td>plot_ulrb_clustering</td>
<td>25</td>
</tr>
<tr>
<td>plot_ulrb_silhouette</td>
<td>27</td>
</tr>
<tr>
<td>prepare_tidy_data</td>
<td>29</td>
</tr>
<tr>
<td>suggest_k</td>
<td>30</td>
</tr>
</tbody>
</table>
```

Index

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
</table>

---

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ulrb-package</td>
<td>2</td>
</tr>
<tr>
<td>ulrb: <em>Unsupervised Learning Based Definition of Microbial Rare Biosphere</em></td>
<td></td>
</tr>
</tbody>
</table>
Author(s)

Francisco Pascoal <fpascoal1996@gmail.com>, Paula Branco <paobranco@gmail.com>, Luis Torgo, Rodrigo Costa <rodrigoscosta@tecnico.ulisboa.pt>, Catarina Magalhães <catarinamagalhaes1972@gmail.com>

Maintainer: Francisco Pascoal

References


Examples

library(ulrb)
# nice is an OTU table in wide format
head(nice)

# first, we tidy the "nice" OTU table
sample_names <- c("ERR2044662", "ERR2044663", "ERR2044664", "ERR2044665", "ERR2044666", "ERR2044667", "ERR2044668", "ERR2044669", "ERR2044670")

# If data is in wide format, with samples in cols
nice_tidy <- prepare_tidy_data(nice, sample_names = sample_names, samples_in = "cols")

# second, we apply ulrb algorithm in automatic setting
nice_classification_results <- define_rb(nice_tidy)

# third, we plot microbial community and the quality of k-medoids clustering
plot_ulrb(nice_classification_results, taxa_col = "OTU", plot_all = TRUE)

# In case you want to inspect the result of a particular sample, do:
plot_ulrb(nice_classification_results, taxa_col = "OTU", sample_id = "ERR2044662")
check_avgSil  

Check average Silhouette score index

Description
Calculates average Silhouette score for a given sample.

Usage
```r
check_avgSil(
  data,
  sample_id = NULL,
  samples_col = "Sample",
  abundance_col = "Abundance",
  range = 3:10,
  with_plot = FALSE,
  ...
)
```

Arguments
- **data**: A tibble with, at least, a column for Abundance and Sample. Additional columns are allowed.
- **sample_id**: String with name of the sample to apply this function.
- **samples_col**: String with name of column with sample names.
- **abundance_col**: String with name of column with abundance values.
- **range**: The range of values of k to test, default is from 3 to 10.
- **with_plot**: If FALSE (default) returns a vector, but if TRUE will return a plot with the scores.
- **...**: Extra arguments.

Details
The average Silhouette score index provides a sense of cluster definition and separation. It varies between -1 (complete cluster overlap) and 1 (no cluster overlap), the closest to 1, the better. Thus, **the k value with highest average Silhouette score is the best k**. This is the standard metric used by the **ulrb** package for automation of the decision of k, in functions `suggest_k()` and `define_rb()`.

Note: The average Silhouette score is different from the common calculation of the Silhouette index, which provides a score for each observation in a clustering result. Just like the name says, we are taking the average of all silhouette scores obtained in a clustering result. In this way we can have a single, comparable value for each k we test.

Data input
This function takes a data.frame with a column for samples and a column for abundance (minimum), but can take any number of other columns. It will then filter the specific sample that you want to
analyze. You can also pre-filter for your specific sample, but you still need to provide the sample ID (sample_id) and the table always needs a column for Sample and another for Abundance (indicate how you name them with the arguments samples_col and abundance_col).

**Output options**

The default option returns a vector with CH scores for each k. This is a simple output that can then be used for other analysis. However, we also provide the option to show a plot (set with_plot = TRUE) with the CH score for each k.

Note that this function does not plot the classical Silhouette plot of a clustering result. To do that particular plot, use the function `plot_ulrb_silhouette()` instead.

**Explanation of average Silhouette score**

To calculate the Silhouette score for a single observation, let:

- $a$ be the mean distance between an observation and all other observations from the same cluster; and
- $b$ be the mean distance between all observations in a cluster and the centroid of the nearest cluster.

The silhouette score (Sil), is given by:

$$ SIL = \frac{(b - a)}{\max(a, b)} $$

Once you have the Silhouette score for all observations in a clustering result, just take the simple mean and get the average Silhouette score.

**Silhouette score intuition**

From the above formula, $SIL = \frac{(b - a)}{\max(a, b)}$, it is clear that, for a given observation:

- if $a > b$, the Silhouette score approaches 1; this means that the distance between an observation and its own cluster is larger than the distance to the nearest different cluster. This is the distance that must be maximized so that all points in a cluster are more similar with each other, than they are with other clusters.

- if $a = b$, then the Silhouette score is 0; this means that the distance between the observation and its own cluster is equivalent to distance between the nearest different cluster.

- if $a < b$, then the Silhouette score approaches -1; in this situation, an observation is nearer the nearest different cluster, than it is to its own cluster. Thus, a negative score indicates that the observation is not in the correct cluster.

**average Silhouette score intuition**

If we take the average of the Silhouette score obtained for each observation in a clustering result, then we have the ability to compare the overall success of that clustering with another clustering. Thus, if we compare the average Silhouette score across different k values, i.e. different number of clusters, we can select the k with highest average Silhouette score.

**Value**

Vector with average Silhouette score index for each pre-specified k.
check_CH

Check Calinski-Harabasz index

Description

Calculates Calinski-Harabasz pseudo F-statistic (CH) for a given sample

Usage

check_CH(
  data,
  sample_id,
  samples_col = "Sample",
  abundance_col = "Abundance",
  range = 3:10,
  with_plot = FALSE,
  ...
)

Arguments

data A tibble with, at least, a column for Abundance and Sample. Additional columns are allowed.
sample_id String with name of the sample to apply this function.
samples_col String with name of column with sample names.
abundance_col String with name of column with abundance values.

References


See Also

define_rb(), suggest_k(), cluster::pam(), cluster::silhouette()

Examples

library(dplyr)
# Just scores
check_avgSil(nice_tidy, sample_id = "ERR2044662")

# To change range
check_avgSil(nice_tidy, sample_id = "ERR2044662", range = 4:11)

# To see a simple plot
check_avgSil(nice_tidy, sample_id = "ERR2044662", range = 4:11, with_plot=TRUE)
Details

CH is an index used to decide the number of clusters in a clustering algorithm. This function, `check_CH()`, calculates the CH index for every `k` in a pre-specified range of values. Thus providing a score for each number of clusters tested (`k`). The default range of cluster values (`k`) is `range = 3:10` (see why this is in Pascoal et al., 2024, in peer review). However, this function may calculate the CH index for all possible `k`'s.

Note that CH index is not an absolute value that indicates the quality of a single clustering. Instead, it allows the comparison of clustering results. Thus, if you have several clusterings, the best one will be the one with higher CH index.

Data input

This function takes a data frame with a column for samples and a column for abundance (minimum), but can take any number of other columns. It will then filter the specific sample that you want to analyze. You can also pre-filter for your specific sample, but you still need to provide the sample ID (sample_id) and the table always needs a column for Sample and another for Abundance (indicate how you name them with the arguments `samples_col` and `abundance_col`).

Output options

The default option returns a vector with CH scores for each `k`. This is a simple output that can then be used for other analysis. However, we also provide the option to show a plot (set `with_plot = TRUE`) with the CH score for each `k`.

Explanation of Calinski-Harabasz index

The CH index is a variance ratio criterion, it measures both separation and density of the clusters. The higher, the better, because it means that the points within the same cluster are close to each other; and the different clusters are well separated.

You can see CH index as:

\[ CH = \frac{\text{inter cluster dispersion}}{\text{intra cluster dispersion}} \]

To calculate inter-cluster:

Let \( k \) be the number of clusters and \( BGSS \) be the Between-group sum of squares, inter-cluster dispersion is

\[ \frac{BGSS}{(k - 1)} \]

To calculate BGSS:

Let \( n_k \) be the number of observations in a cluster, \( C \) be the centroid of the dataset (barycenter) and \( C_k \) the centroid of a cluster,
Thus, the BGSS multiplies the distance between the cluster centroid and the centroid of the whole dataset, by all observations in a given cluster, for all clusters.

To calculate intra-cluster dispersion:
Let $WGSS$ be the Within Group Sum of Squares and $N$ be the total number of observations in the dataset.

\[
WGSS = \left( \frac{N - 1}{N} \right)
\]

Let $X_{ij}$ be i’th observation of a cluster and $n_k$ be the number of observations in a cluster.

\[
WGSS = \sum_{k=1}^{k} \sum_{i=1}^{n_k} |X_{ij} - C_k|
\]

Thus, $WGSS$ measures the distance between observations and their cluster center; if divided by the total number of observations, then gives a sense of intra-dispersion.

Finally, the CH index can be given by:

\[
CH = \frac{\sum_{k=1}^{k} n_k * |C_k - C|^2}{(N - k) (k - 1)}
\]

Value

Vector or plot with Calinski-Harabasz index for each pre-specified $k$.

References


Pascoal et al., 2024 (in peer review)

See Also

clusterSim::index.G1

Examples

library(dplyr)
# Just scores
check_CH(nice_tidy, sample_id = "ERR2044662")
# To change range
check_DB

check_CH(nice_tidy, sample_id = "ERR2044662", range = 4:11)

# To see a simple plot
check_CH(nice_tidy, sample_id = "ERR2044662", range = 4:11, with_plot=TRUE)

---

check_DB  

**Check Davies-Bouldin Index**

**Description**

Calculates Davies-Bouldin (DB) Index for a given sample.

**Usage**

```
check_DB(
  data,
  sample_id,
  samples_col = "Sample",
  abundance_col = "Abundance",
  range = 3:10,
  with_plot = FALSE,
  ...
)
```

**Arguments**

- `data` A tibble with, at least, a column for Abundance and Sample. Additional columns are allowed.
- `sample_id` String with name of the sample to apply this function.
- `samples_col` String with name of column with sample names.
- `abundance_col` String with name of column with abundance values.
- `range` The range of values of k to test, default is from 3 to 10.
- `with_plot` If FALSE (default) returns a vector, but if TRUE will return a plot with the scores.
- `...` Extra arguments.

**Details**

DB is an index used to decide the number of clusters in a clustering algorithm. This function, `check_DB()`, calculates the DB index for every k in a pre-specified range of values. Thus providing a score for each number of clusters tested (k). The default range of cluster values (k) is `range = 3:10` (see why this is in Pascoal et al., 2023). However, this function may calculate the DB index for all possible k’s.
Note that DB index is not an absolute value that indicates the quality of a single clustering. Instead, it allows the comparison of clustering results. Thus, if you have several clusterings, the best one will be the one with higher DB index.

**Data input**

This function takes a data.frame with a column for samples and a column for abundance (minimum), but can take any number of other columns. It will then filter the specific sample that you want to analyze. You can also pre-filter for your specific sample, but you still need to provide the sample ID (sample_id) and the table always needs a column for Sample and another for Abundance (indicate how you name them with the arguments samples_col and abundance_col).

**Output options**

The default option returns a vector with DB scores for each k. This is a simple output that can then be used for other analysis. However, we also provide the option to show a plot (set with_plot = TRUE) with the DB score for each k.

**Explanation of Davies-Bouldin index**

The DB index (Davies and Bouldin, 1979) is an averaged measure of cluster similarity to the closest cluster. This provides a sense of how separated the clusters are.

Lower DB scores are better, because they represent more distinct clusters. Higher values of DB indicate overlapping clusters.

Let $N$ be the number of clusters and $R_i$ the similarity between the $i$'th cluster and the cluster most similar to it. The DB index is calculated as the mean similarity between each cluster and the most similar cluster,

$$DB = \frac{1}{N} \sum_{i=1}^{N} R_i$$

Thus, $R_i$ is the maximum similarity among all possible combinations of $R_{ij}$, with $i \neq j$.

To get $R_{ij}$, let $S_i$ be the intra-cluster dispersion of $i$, $S_j$ be the intra-cluster dispersion of cluster $j$ and $M_{ij}$ be the distance between clusters $i$ and $j$.

The similarity between any two clusters, $i$ and $j$, is:

$$R_{ij} = \frac{S_i + S_j}{M_{ij}}$$

The distance between any two clusters, $M_{ij}$, is measured as the distance between the centroids of both clusters, $|C_i - C_j|$.

The dispersion of clusters, $S_i$, provides a sense of intra-dispersion of a given cluster.

To calculate $S_i$, let $T_i$ and $T_j$ be the number of observations in $i$ and $j$, respectively; let $X_j$ be the value for j’th observation (again, $i \neq j$).

$$S_i = \sqrt{\frac{1}{T_i} \sum_{j=1}^{T_i} |X_j - C_i|}$$

Note that this is the case for euclidean distances.
Value

A vector or plot with Davies-Bouldin index for each pre-specified k in a given sample.

References


See Also

clusterSim::index.DB()

Examples

library(dplyr)
# Just scores
check_DB(nice_tidy, sample_id = "ERR2044662")

# To change range
check_DB(nice_tidy, sample_id = "ERR2044662", range = 4:11)

# To see a simple plot
check_DB(nice_tidy, sample_id = "ERR2044662", range = 4:11, with_plot=TRUE)
define_rb

Arguments

data       A tibble with, at least, a column for Abundance and Sample. Additional columns are allowed.
classification_vector  A vector of strings with the names for each cluster, from lower to higher abundance. Default is c("Rare", "Undetermined", "Abundance").
samples_col   String with name of column with sample names.
abundance_col String with name of column with abundance values.
simplified    Can be TRUE/FALSE. Default (FALSE) provides an additional column with detailed pam() results and Silhouette scores. If TRUE, only the Classification result is added to the original input data.
automatic     By default (FALSE), will assume a classification into "Rare", "Undetermined" or "Abundant". If TRUE, then it will automatically select the number of classifications (or k), based on the index argument.
index         Index used to select best k. Can be one of: "Average Silhouette Score", "Davies-Bouldin" or "Calinski-Harabasz".
...           Extra arguments.

Details

Overview

Function to cluster species abundance with partition around medoids algorithm (Kaufman and Roussieu. 1991). By default, we propose the division into three clusters (k = 3), which can have the convenient description of: "rare", "undetermined" and "abundant". The phylogenetic units from the cluster with lowest median abundance is considered to be the "rare biosphere".

The classification vector

The classification vector (argument classification_vector) represents the different clusters to be used, by ascending order of median abundance. To change the number of clusters, change the number of elements in the classification vector, but order matters! Depending on the number of clusters used, you can change the meaning that best applies to your research.

For example, you can use a classification vector with the designations: "very rare", "rare", "abundant" and "very abundant"; which would apply a k = 4 underneath. It is possible to use any number of clusters, as long as they are within 2 and the maximum possible k.

The maximum possible k is the number of different abundance scores observed in a single sample. Note, however, that we do not recommend any clustering for k > 10 and we also don’t recommend k = 2 (we explain in more detail in Pascoal et al., 2023; and in the vignette vignette("explore-classifications").

Automatic selection of the number of clusters

To automatically decide the number of clusters (i.e., the value of k), it is possible to do so with the argument automatic=TRUE. For details on complete automation of define_rb(), please see the documentation for suggest_k(). Briefly, the k with best average Silhouette score is selected from a range of k values between 3 and 10. It is possible to decide k based on other indices ("Davies-Bouldin" or "Calinski-Harabasz").

If you want a more fine grained analysis of k values, we provide several functions:
**Verify clustering results**

If half of the taxa of any cluster got a Silhouette score below 0.5 in any sample, then a warning is provided. The warning provides the number of times this issue occurred. You can inspect other alternatives to reduce the occurrences of a bad clustering, but it is possible that, in some situations, you just can’t find an optimal clustering.

The detailed output gives you access to all of the clustering results:

- `pam_object` is a list with the original results from the k-medoids clustering, see `cluster::pam()` documentation.
- `Level` is an integer indicating the specific cluster attributed by the `cluster::pam()` function for each observation. Its order is random.
- `Silhouette_scores` provides the Silhouette score obtained for each observation, i.e. a score for each taxa.
- `Cluster_median_abundance` provides the median taxa abundance of each cluster.
- `median_Silhouette` provides the median Silhouette score obtained for each cluster.
- `Evaluation` indicates if the silhouette score obtained for a given observation is below the median Silhouette of its cluster and sample.

You can make your own plots and analysis, but we also provide another function, `plot_ulrb()`, which illustrates the results obtained.

**Partition around medoids (pam)**

To calculate k-medoids, we used the partition around medoids (pam) algorithm, which was described in Chapter 2 of "Finding Groups in Data: An Introduction to Cluster Analysis." (Kaufman and Rousseeuw, 1991) and implemented by the cluster package with the `cluster::pam()` function.

Briefly, the pam algorithm is divided into two main phases: **build** and **swap**.

The first phase (**build**) selects k observations as cluster representatives. The first observation selected as representative is the one that minimizes the sum of the dissimilarities to the remaining observations. The second, third and so on repeat the same process, until k clusters have been formed.

The **build** steps are:

1 - Propose a centroid with observation, $i$, which has not been selected as a centroid yet
2 - Calculate the distance between another observation, $j$, and its most similar observation, $D_j$; and calculate the difference with the proposed centroid, $i$, i.e., $d(j, i)$
3 - If $d(j, i) > 0$, then calculate its contribution to the centroid:

   $$\max(D_j - d(j, i), 0)$$

4 - Calculate the total gain obtained by $i$,

   $$\sum_j C_{ji}$$
5 - From all possible centroids, select the one that maximizes the previous total gain obtained,

$$\max_i \sum_j C_{ji}$$

6 - Repeat until k observations have been selected as cluster representatives.

The purpose of the next phase, swap, is to improve the representatives for the clusters. The principle is to swap the cluster representative between all possibilities and calculate the value sum of dissimilarities between each observation and the closest centroid. The swapping continues until no more improvement is possible, i.e., when the minimum sum of dissimilarities of the clusters is reached.

Notes:
Understand that ulrb package considers each sample as an independent community of phylogenetic units, which means clustering is also independent across different samples. Thus, be aware that you will have clustering results and metrics for each single sample, which is why we also provide some functions to analyze results across any number of samples (see: plot_ulrb() for clustering results and evaluate_k() for k selection).

Value
The input data.frame with extra columns containing the classification and additional metrics (if detailed = TRUE).

References

See Also
suggest_k(), evaluate_k(), plot_ulrb(), cluster::pam()

Examples
library(dplyr)
# Sample ID's
sample_names <- c("ERR2044662", "ERR2044663", "ERR2044664", 
"ERR2044665", "ERR2044666", "ERR2044667", 
"ERR2044668", "ERR2044669", "ERR2044670")

# If data is in wide format, with samples in cols
nice_tidy <- prepare_tidy_data(nice, 
  sample_names = sample_names, 
  samples_in = "cols")

# Straightforward with tidy format
define_rb(nice_tidy)

# Closer look
classified_table <- define_rb(nice_tidy)
classified_table %>%
  select(Sample, Abundance, Classification) %>%
  head()

# Automatic decision, instead of a predefined definition
define_rb(nice_tidy, automatic = TRUE) %>% select(Sample, Abundance, Classification)

# Automatic decision, using Davies-Bouldin index,
# instead of average Silhouette score (default)
define_rb(nice_tidy, automatic = TRUE, index = "Davies-Bouldin") %>%
  select(Sample, Abundance, Classification)

# User defined classifications
user_classifications <- c("very rare",
"rare",
"undetermined",
"abundant",
"very abundant")
define_rb(nice_tidy, classification_vector = user_classifications) %>%
  select(Sample, Abundance, Classification)

# Easy to incorporate in big pipes
# Remove Archaea
# Remove taxa below 10 reads
# Classify according to a different set of classifications
nice_tidy %>%
  filter(Domain != "sk__Archaea") %>%
  filter(Abundance > 10) %>%
  define_rb(classification_vector = c("very rare",
"rare",
"abundant",
"very abundant")) %>%
  select(Sample, Abundance, Classification)

# An example that summarises results
nice_tidy %>%
  filter(Domain != "sk__Archaea") %>%
  filter(Abundance > 10) %>%
  define_rb(classification_vector = c("very rare",
"rare",
"abundant",
"very abundant")) %>%
  select(Sample, Abundance, Classification) %>%
  group_by(Sample, Classification) %>%
  summarise(totalAbundance = sum(Abundance))

evaluate_k

Evaluate k from all samples in a dataset
**Description**

This function extends `evaluate_sample_k()` for any number of samples in a dataset.

**Usage**

```r
evaluate_k(
  data,
  range = 3:10,
  samples_col = "Sample",
  abundance_col = "Abundance",
  with_plot = FALSE,
  ...
)
```

**Arguments**

- `data`: a data.frame with, at least, the classification, abundance and sample information for each phylogenetic unit.
- `range`: The range of values of k to test, default is from 3 to 10.
- `samples_col`: String with name of column with sample names.
- `abundance_col`: string with name of column with abundance values. Default is "Abundance".
- `with_plot`: If FALSE (default) returns a vector, but if TRUE will return a plot with the scores.
- `...`: Extra arguments.

**Details**

The plot option (with_plot = TRUE) provides centrality metrics for all samples used.

For more details on indices calculation, please see the documentation for `evaluate_sample_k()`, `check_DB()`, `check_CH()` and `check_avgSil()`.

**Value**

A nested data.frame (or a plot) with three indices for each k and for each sample.

**See Also**

`evaluate_sample_k()`, `check_DB()`, `check_CH()`, `check_avgSil()`, `suggest_k()`

**Examples**

```r
library(dplyr)

#` evaluate_k(nice_tidy)

evaluate_k(nice_tidy, range = 4:11, with_plot = TRUE)
```
evaluate_sample_k

Description

This function calculates three indices (Davies-Bouldin, Calinski-Harabasz and average Silhouette score) for each k. Calculations are made for a single sample and for a default range of k that goes from 3 to 10.

Usage

```
evaluate_sample_k(
  data,
  sample_id,
  samples_col = "Sample",
  abundance_col = "Abundance",
  range = 3:10,
  with_plot = FALSE,
  ...
)
```

Arguments

data: a data.frame with, at least, the classification, abundance and sample information for each phylogenetic unit.
sample_id: String with name of the sample to apply this function.
samples_col: String with name of column with sample names.
abundance_col: string with name of column with abundance values. Default is "Abundance".
range: The range of values of k to test, default is from 3 to 10.
with_plot: If FALSE (default) returns a vector, but if TRUE will return a plot with the scores.
...
Extra arguments.

Details

Note: To get the indices for all samples, use `evaluate_k()` instead.

Data input

This function takes a data.frame with a column for samples and a column for abundance (minimum), but can take any number of other columns. It will then filter the specific sample that you want to analyze. You can also pre-filter for your specific sample, but you still need to provide the sample ID (`sample_id`) and the table always needs a column for Sample and another for Abundance (indicate how you name them with the arguments `samples_col` and `abundance_col`).
Output options
The default option returns a data.frame with Davies-Bouldin, Calinsky-Harabasz and average Silhouette scores for each k. This is a simple output that can then be used for other analysis. However, we also provide the option to show a plot (set with_plot = TRUE).

Three indices are calculated by this function:

- Davies-Bouldin with check_DB();
- Calinsky-Harabasz with check_DB();
- average Silhouette score check_avgSil().

Value
A data.frame (or plot) with several indices for each number of clusters.

See Also
check_CH(), check_DB(), check_avgSil(), suggest_k(), evaluate_k()

Examples
library(dplyr)
#
evaluate_sample_k(nice_tidy, sample_id = "ERR2044662")

# To change range
evaluate_sample_k(nice_tidy, sample_id = "ERR2044662", range = 4:11)

# To make simple plot
evaluate_sample_k(nice_tidy, sample_id = "ERR2044662", range = 4:11, with_plot = TRUE)

nice V4-V5 16S rRNA gene amplicons, clean OTU table (N-ICE, 2015)

Description
Table in "wide" format with abundance and taxonomic classification of each OTU.

Usage
nice
Format

nice:
A data frame with 524 rows and 17 columns:

<table>
<thead>
<tr>
<th>ERR2044662, ERR2044663, ERR2044664, ERR2044665, ERR2044666, ERR2044667, ERR2044668, ERR2044669 and ERR2044670</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sample ID</td>
</tr>
<tr>
<td>OTU ID</td>
</tr>
<tr>
<td>Domain</td>
</tr>
<tr>
<td>Phylum</td>
</tr>
<tr>
<td>Class</td>
</tr>
<tr>
<td>Order</td>
</tr>
<tr>
<td>Family</td>
</tr>
<tr>
<td>Genus</td>
</tr>
<tr>
<td>Species</td>
</tr>
</tbody>
</table>

Details

This OTU table was cleaned so that it only includes samples from 16S rRNA amplicon sequencing and no eukaryotes (similarly to Pascoal et al., 2022). Additionally, we added a column with a ID for each OTU.

For details on raw data, see nice_raw

Source

https://www.ebi.ac.uk/metagenomics/studies/MGYS00001922#analysis

References


See Also

nice_tidy(), nice_raw, nice_env

nice_env Metadata of samples from OTU tables (N-ICE, 2015)

Description

This dataset provides information on the samples used for the N-ICE dataset.

Usage

nice_env
Format

nice_tidy:
A data frame with 4716 rows and 10 columns:

Sample  Sample ID used in the original study
ENA_ID   Sample ID equivalent to Sample in the OTU table
Month    Month of sampling
Region   Ocean region of sampling event
Water.mass Water mass of sampling event
Latitude Longitude of sampling event

Details

Based on de Sousa et al., 2019.

Source


References


See Also

nice, nice_raw, nice_tidy

Description

This is the "raw" data for the N-ICE dataset.

Usage

nice_raw
Format

```
nice_raw:
A data frame with 524 rows and 17 columns:

ERR2044662, ERR2044663, ERR2044664, ERR2044665, ERR2044666, ERR2044667, ERR2044668, ERR2044669 and ERR2044670
```

**Sample ID**

**OTU ID**

**Domain** Domain level classification of OTU

**Phylum** Phylum level classification of OTU

**Class** Class level classification of OTU

**Order** Order level classification of OTU

**Family** Family level classification of OTU

**Genus** Genus level classification of OTU

**Species** Species level classification of OTU

Details

The original sequencing results are available at European Nucleotide Archive (accession number: PRJEB15043). Those reads were processed into OTUs by MGnify platform (Study: MGYS00001922). The later study accession provides the table used in here.

This table contains the taxonomy and an abundance score for each taxonomic lineage, which we will refer to as "OTU" (Operational OTU) for simplicity sake.

For details on the sampling campaign in the Arctic ocean, sequencing protocols and bioinformatic processing, please see ref (de Sousa et al., 2019).

Source

https://www.ebi.ac.uk/metagenomics/studies/MGYS00001922#analysis

References


See Also

```
nice(), nice_tidy, nice_env
```
nice_tidy  

V4-V5 16S rRNA gene amplicons, clean OTU table in tidy/long format 
(N-ICE, 2015)

Description

Original OTU table (nice) in "long" format.

Usage

nice_tidy

Format

nice_tidy:
A data frame with 4716 rows and 10 columns:

Sample  Sample ID
Abundance Abundance
OTU  OTU ID
Domain  Domain level classification of OTU
Phylum  Domain level classification of OTU
Class  Domain level classification of OTU
Order  Domain level classification of OTU
Family  Domain level classification of OTU
Genus  Domain level classification of OTU
Species  Domain level classification of OTU ...

Details

A new column (Sample) includes the sample identifiers and a new column (Abundance) includes the abundance for each OTU. For details on OTU table processing see help pages for nice and nice_raw.

Some details on N-ICE dataset:

This dataset resulted from the Norwegian Young Sea Ice expedition (N-ICE) in 2015 (Granskog et al., 2018). The sample processing and DNA sequencing were described in de Sousa et al., 2019, the bioinformatic processing was performed by the M Gnify platform (v5) (Mitchell et al., 2020).

Since the purpose of this dataset if for creating examples and testing the package, we did not apply strict quality control to the final OTU table. Thus, we didn't remove singletons, etc. However, we did remove any non-prokarotic OTUs and organelles, if any (Pascoal et al., 2022).

Source

https://www.ebi.ac.uk/metagenomics/studies/MGYS00001922#analysis
References

- Mitchell, A. L., Almeida, A., Beracochea, M., Boland, M., Burgin, J., Cochrane, G., Cru- 
  soe, M. R., Kale, V., Potter, S. C., Richardson, L. J., Sakharova, E., Scheremetjew, M., Ko-
  the microbiome analysis resource in 2020. Nucleic Acids Research, 48(D1), D570–D578.

  Processes in a Thinner Arctic Sea Ice Regime: The Norwegian Young Sea ICE (N-ICE2015) 

  and Composition of Pelagic Prokaryotic and Protist Communities in a Thin Arctic Sea-Ice 

  of Rarity in the Arctic Ocean from the Perspective of Multiple Methodologies. Microbial 
  Ecology, 84(1), 59–72.

See Also

nice(), nice_raw, nice_env

---

plot_ulrb

Plot ulrb clustering results and silhouette scores

Description

Function to help access clustering results from ulrb.

Usage

plot_ulrb(
  data,
  sample_id = NULL,
  taxa_col,
  plot_all = FALSE,
  silhouette_score = "Silhouette_scores",
  classification_col = "Classification",
  abundance_col = "Abundance",
  log_scaled = FALSE,
  colors = c("#009E73", "#F0E442", "#CC79A7"),
  ...
)
Arguments

data
- a data.frame with, at least, the classification, abundance and sample information for each phylogenetic unit.
sample_id
- string with name of selected sample.
taxa_col
- string with name of column with phylogenetic units. Usually OTU or ASV.
plot_all
- If TRUE, will make a plot for all samples with mean and standard deviation. If FALSE (default), then the plot will illustrate a single sample, that you have to specify in sample_id argument.
silhouette_score
- string with column name with silhouette score values. Default is "Silhouette_scores"
classification_col
- string with name of column with classification for each row. Default value is "Classification".
abundance_col
- string with name of column with abundance values. Default is "Abundance".
log_scaled
- if TRUE then abundance scores will be shown in Log10 scale. Default to FALSE.
colors
- vector with colors. Should have the same length as the number of classifications.

Details

This function combined plot_ulrb_clustering() and plot_ulrb_silhouette(). The plots can be done for a single sample or for all samples.

The results from the main function of ulrb package, define_rb(), will include the classification of each species (OTU, ASVs, etc) and the silhouette score obtained for each observation. Thus, to access the clustering results, there are two main plots to check:

- the rank abundance curve obtained after ulrb classification;
- and the silhouette plot.

Interpretation of Silhouette plot

Based on chapter 2 of "Finding Groups in Data: An Introduction to Cluster Analysis." (Kaufman and Rousseeuw, 1991); a possible (subjective) interpretation of the clustering structure based on the Silhouette plot is:

- 0.71-1.00 (A strong structure has been found);
- 0.51-0.70 (A reasonable structure has been found);
- 0.26-0.50 (The structure is weak and could be artificial);
- <0.26 (No structure has been found).

Value

A grid of ggplot objects with clustering results and silhouette plot obtained from define_rb().
**plot_ulrb_clustering**

See Also

`define_rb()`, `check_avgSil()`, `plot_ulrb_clustering()`, `plot_ulrb_silhouette()`

**Examples**

classified_species <- define_rb(nice_tidy)

# Default parameters for a single sample ERR2044669
plot_ulrb(classified_species,
          sample_id = "ERR2044669",
          taxa_col = "OTU",
          abundance_col = "Abundance")

# All samples in a dataset
plot_ulrb(classified_species,
          taxa_col = "OTU",
          abundance_col = "Abundance",
          plot_all = TRUE)

# All samples with a log scale
plot_ulrb(classified_species,
          taxa_col = "OTU",
          abundance_col = "Abundance",
          plot_all = TRUE,
          log_scaled = TRUE)

---

**plot_ulrb_clustering**  
*Plot Rank Abundance Curve of classification results*

**Description**

Plots the clustering results from `define_rb()`.

**Usage**

```r
plot_ulrb_clustering(
  data,  
  sample_id = NULL,  
  taxa_col,  
  plot_all = FALSE,  
  samples_col = "Sample",  
  classification_col = "Classification",  
  abundance_col = "Abundance",  
  log_scaled = FALSE,  
  colors = c("#009E73", "#F0E442", "#CC79A7"),  
  ...
)
```
**Arguments**

- **data**
  a data.frame with, at least, the classification, abundance and sample information for each phylogenetic unit.

- **sample_id**
  string with name of selected sample.

- **taxa_col**
  string with name of column with phylogenetic units. Usually OTU or ASV.

- **plot_all**
  If TRUE, will make a plot for all samples with mean and standard deviation. If FALSE (default), then the plot will illustrate a single sample, that you have to specify in sample_id argument.

- **samples_col**
  name of column with sample ID’s.

- **classification_col**
  string with name of column with classification for each row. Default value is "Classification".

- **abundance_col**
  string with name of column with abundance values. Default is "Abundance".

- **log_scaled**
  if TRUE then abundance scores will be shown in Log10 scale. Default to FALSE.

- **colors**
  vector with colors. Should have the same length as the number of classifications.

- **...**
  other arguments.

**Details**

This works as a sanity check of the results obtained by the unsupervised learning method used to classify species. This is specially important if you used an automatic number of clusters.

The function works for either a single sample (that you specify with sample_id argument), or it can apply a centrality metric for species across all your samples (plot_all = TRUE).

**Value**

A ggplot object with clustering results from `define_rb()`.

**See Also**

`define_rb()`, `plot_ulrb()`, `plot_ulrb_silhouette()`

**Examples**

```r
classified_species <- define_rb(nice_tidy)

# Standard plot for a single sample
plot_ulrb_clustering(classified_species,
                     sample_id = "ERR2044669",
                     taxa_col = "OTU",
                     abundance_col = "Abundance")

# All samples in a dataset
plot_ulrb_clustering(classified_species,
                     taxa_col = "OTU",
                     abundance_col = "Abundance",
                     plot_all = TRUE)
```
# All samples with a log scale
plot_ulrb_clustering(classified_species,
  taxa_col = "OTU",
  abundance_col = "Abundance",
  plot_all = TRUE,
  log_scaled = TRUE)

plot_ulrb_silhouette  

Plot silhouette scores from clustering results

Description

Plots the Silhouette scores from the clustering results of define_rb().

Usage

plot_ulrb_silhouette(
  data,
  sample_id = NULL,
  taxa_col,
  samples_col = "Sample",
  plot_all = FALSE,
  classification_col = "Classification",
  silhouette_score = "Silhouette_scores",
  colors = c("#009E73", "#F0E442", "#CC79A7"),
  log_scaled = FALSE,
  ...
)

Arguments

data ... 
sample_id string with name of selected sample.
taxa_col string with name of column with phylogenetic units. Usually OTU or ASV.
samples_col name of column with sample ID's.
plot_all If TRUE, will make a plot for all samples with mean and standard deviation. If FALSE (default), then the plot will illustrate a single sample, that you have to specify in sample_id argument.
classification_col string with name of column with classification for each row. Default value is "Classification".
silhouette_score string with column name with silhouette score values. Default is "Silhouette_scores"
colors vector with colors. Should have the same length as the number of classifications.
log_scaled if TRUE then abundance scores will be shown in Log10 scale. Default to FALSE.

... other arguments.

Details

This works as a sanity check of the results obtained by the unsupervised learning method used to classify species. This is specially important if you used an automatic number of clusters.

The function works for either a single sample (that you specify with sample_id argument), or it can apply a centrality metric for species across all your samples (plot_all = TRUE).

For more details on Silhouette score, see check_avgSil() and cluster::silhouette().

Interpretation of Silhouette plot

Based on chapter 2 of "Finding Groups in Data: An Introduction to Cluster Analysis." (Kaufman and Rousseeuw, 1991); a possible (subjective) interpretation of the clustering structure based on the Silhouette plot is:

• 0.71-1.00 (A strong structure has been found);
• 0.51-0.70 (A reasonable structure has been found);
• 0.26-0.50 (The structure is weak and could be artificial);
• < 0.26 (No structure has been found).

Value

A ggplot object of Silhouette plot obtained from the selected sample.

See Also

define_rb(), check_avgSil(), plot_ulrb_clustering(), plot_ulrb(), cluster::silhouette(),
cluster::pam()

Examples

classified_species <- define_rb(nice_tidy)

# Standard plot for a single sample
plot_ulrb_silhouette(classified_species,
  sample_id = "ERR2044669",
  taxa_col = "OTU",
  abundance_col = "Abundance")

# All samples in a dataset
plot_ulrb_silhouette(classified_species,
  taxa_col = "OTU",
  abundance_col = "Abundance",
  plot_all = TRUE)

# All samples with a log scale
plot_ulrb_silhouette(classified_species,
  taxa_col = "OTU",
  abundance_col = "Abundance",
  log_scaled = TRUE)
### Description

Function to transform common abundance table formats into a "long" format.

### Usage

```r
prepare_tidy_data(data, sample_names, samples_in = "cols", ...)
```

### Arguments

- `data`：a data.frame in "wide" format, with samples in either columns or rows. This data.frame should not include any data besides abundance values per sample, per taxonomic unit. Additional data (e.g. taxonomy details) should be added afterwards.

- `sample_names`：a vector with the name of all samples.

- `samples_in`：a vector specifying the location of the samples. It can either be "cols" (default) if samples are in columns, or "rows" if samples are in rows.

- `...`：additional arguments

### Details

This function guarantees that the abundance table includes one column with sample ID’s and one column with abundance.

**Common species table formats**

There are two common formats for abundance tables:

- samples as rows and phylogenetic units as columns;
- phylogenetic units as rows and samples as columns.

However, both formats are not tidy, because they include several columns with the same variable. They are in a "wide format" instead of a "long format".

This function re-organizes samples and phylogenetic units so that there is a single column with the samples ID’s and another with the abundance scores; Extra columns are allowed.

### Value

An abundance table in long format, compatible with dplyr pipes and `ultrb` package functions.
suggest_k

Suggest k

Description

Tool to help decide how many clusters to use for partition around medoids algorithm.

Usage

suggest_k(
  data,
  range = 3:10,
  samples_col = "Sample",
  abundance_col = "Abundance",
  index = "Average Silhouette Score",
  detailed = FALSE,
  ...)

Examples

library(dplyr)
#
sample_names <- c("ERR2044662", "ERR2044663", "ERR2044664",
                  "ERR2044665", "ERR2044666", "ERR2044667",
                  "ERR2044668", "ERR2044669", "ERR2044670")

# Example for samples in cols and with additional data available
prepare_tidy_data(nice, sample_names = sample_names, samples_in = "cols")

# Example for samples in rows
# Select columns with samples from nice
nice_rows <- nice %>% select(all_of(sample_names))

# Change columns to rows
nice_rows <- nice_rows %>% t() %>% as.data.frame()

# Turn colnames into phylogenetic units ID
colnames(nice_rows) <- paste0("OTU_", seq_along(colnames(nice_rows)))

prepare_tidy_data(nice_rows, sample_names = sample_names, samples_in = "rows")
suggest_k

**Arguments**

- `data`  
  a data.frame with, at least, the classification, abundance and sample information for each phylogenetic unit.
- `range`  
  The range of values of k to test, default is from 3 to 10.
- `samples_col`  
  String with name of column with sample names.
- `abundance_col`  
  string with name of column with abundance values. Default is "Abundance".
- `index`  
  Index used to select best k. Can be one of: "Average Silhouette Score", "Davies-Bouldin" or "Calinski-Harabasz".
- `detailed`  
  If False (default) returns an integer with best overall k. If TRUE, returns a list with full details.
- `...`  
  Extra arguments.

**Details**

The best k is selected for each sample, based on the selected index. If different k's are obtained for different samples (probable) then we calculate the mean value of k and return it as an integer. Alternatively, we can return a more detailed result in the form of a list.

**Note:** this function is used within `define_rb()`, with default parameters, for the optional automatic selection of k.

**Detailed option**

If `detailed = TRUE`, then the output is a list with information to help decide for k. More specifically, the list will include:

- A data.frame summarizing what information each index provides and how to interpret the value.
- A brief summary indicating the number of samples in the dataset and the range of k values used.
- A data.frame with the best k for each sample, based on each index.

**Automatic k selection**

If `detailed = FALSE`, this function will provide a single integer with the best k. The `default` decision is based on the maximum average Silhouette score obtained for the values of k between 3 and 10. To better understand why the average Silhouette score and this range of k’s were selected, we refer to Pascoal et al., 2024 (in peer-review) and to vignette("explore-classifications").

Alternatively, this function can also provide the best k, as an integer, based on another index (Davies-Bouldin and Calinski-Harabasz) and can compare the entire of possible k’s.

**Value**

Integer indicating best k from selected index. Optionally, can return a list with details.

**See Also**

evaluate_k(), evaluate_sample_k(), check_DB(), check_CH(), check_avgSil(), cluster::pam()
Examples

# Get the best k with default parameters
suggest_k(nice_tidy)

# Get detailed results to decide for yourself
suggest_k(nice_tidy, detailed = TRUE, range = 2:7)

# Get best k, based on Davies-Bouldin index
suggest_k(nice_tidy, detailed = FALSE, index = "Davies-Bouldin")
Index

* datasets
  nice, 18
  nice Env, 19
  nice raw, 20
  nice tidy, 22

* package
  ulrb-package, 2

check_avgSil, 4
check_avgSil(), 13, 16, 18, 25, 28, 31
check_CH, 6
check_CH(), 7, 13, 16, 18, 31
check_DB, 9
check_DB(), 9, 13, 16, 18, 31
cluster::pam(), 6, 13, 14, 28, 31
cluster::silhouette(), 6, 28
clusterSim::index.DB(), 11
clusterSim::index.G1, 8

define_rb, 11
define_rb(), 4, 6, 12-28, 30, 31

evaluate_k, 15
evaluate_k(), 13, 14, 17, 18, 31
evaluate_sample_k, 17
evaluate_sample_k(), 16, 31

nice, 18, 20, 22
nice(), 21, 23
nice Env, 19, 19, 21, 23
nice raw, 19, 20, 20, 22, 23
nice tidy, 20, 21, 22
nice tidy(), 19

plot_ulrb, 23
plot_ulrb(), 13, 14, 26, 28
plot_ulrb_clustering, 25
plot_ulrb_clustering(), 24, 25, 28
plot_ulrb_silhouette, 27
plot_ulrb_silhouette(), 5, 24-26
prepare_tidy_data, 29

suggest_k, 30
suggest_k(), 4, 6, 12, 14, 16, 18
ulrb-package, 2