Package 'mappeR'

October 16, 2024

Type Package

Title Construct and Visualize TDA Mapper Graphs

Description Topological data analysis (TDA) is a method of data analysis that uses techniques from topology to analyze high-dimensional data. Here we implement Mapper, an algorithm from this area developed by Singh, Mémoli and Carlsson (2007) which generalizes the concept of a Reeb graph <https://en.wikipedia.org/wiki/Reeb_graph>. The output graph is able to be visualized in R using 'igraph' or using a free network analysis software called 'Cytoscape', available for download from at <https://cytoscape.org/>.

License MIT + file LICENSE

URL https://github.com/Uiowa-Applied-Topology/mappeR

BugReports https://github.com/Uiowa-Applied-Topology/mappeR/issues

Version 1.1.0

Encoding UTF-8

Imports fastcluster, grDevices, igraph, stats, utils

Suggests testthat (>= 3.0.0)

Config/testthat/edition 3

RoxygenNote 7.3.2

NeedsCompilation no

Author George Clare Kennedy [aut, cre]

Maintainer George Clare Kennedy <george-clarekennedy@uiowa.edu>

Repository CRAN

Date/Publication 2024-10-16 05:30:02 UTC

Contents

check_in_interval 2
compute_tightness 3
convert_to_clusters 4

create_1D_mapper_object 4
create_balls 5
create_ball_mapper_object 6
create_bins 7
create_clusterball_mapper_object 7
create_mapper_object 8
create_single_bin 9
create_width_balanced_cover 10
cut_dendrogram 11
eccentricity_filter 11
get_bin_vector 12
get_clustered_data 12
get_clusters 13
get_cluster_sizes 13
get_cluster_tightness_vector 14
get_edgelist_from_overlaps 14
get_edge_weights 15
get_overlaps 15
get_single_linkage_clusters 16
get_tallest_branch 16
is_in_ball 17
mapper_object_to_igraph 17
next_triangular 18
process_dendrograms 18
run_cluster_machine 19
run_mapper 19
run_slink 20
subset_dists 20

Index **21**

`check_in_interval` *Get a tester function for an interval.*

Description

Get a tester function for an interval.

Usage

```
check_in_interval(endpoints)
```

Arguments

`endpoints`	A vector of interval endpoints, namely a left and a right. Must be in order.

Value

A function that eats a data point and outputs TRUE if the datapoint is in the interval and FALSE if not.

`compute_tightness` *Compute dispersion of a single cluster*

Description

Compute dispersion of a single cluster

Usage

```
compute_tightness(dists, cluster)
```

Arguments

`dists`	A distance matrix for points in the cluster.
`cluster`	A list containing named vectors, whose names are data point names and whose values are cluster labels

Value

A real number in $(0, \infty)$ representing a measure of dispersion of a cluster. This method finds the medoid of the input data set, the point with the smallest sum of distances to all other points, and returns that sum divided by the largest distance from the medoid to another point. Formally, we say the tightness τ of a cluster C is given by

$$\tau(C) = \frac{1}{\max\limits_{x_i \in C, i \neq j} \text{dist}(x_i, x_j)} \sum_i \text{dist}(x_i, x_j)$$

where

$$x_j = \arg\min_{x_j \in C} \sum_{x_i \in C, i \neq j} \text{dist}(x_i, x_j)$$

A smaller value indicates a tighter cluster based on this metric.

convert_to_clusters *The easiest clustering method*

Description

The easiest clustering method

Usage

```
convert_to_clusters(bins)
```

Arguments

bins	A list of bins, each containing names of data from some data frame.

Value

A named vector whose names are data point names and whose values are cluster labels

create_1D_mapper_object *Run 1D mapper*

Description

Run mapper using a one-dimensional filter, a cover of intervals, and a clustering algorithm.

Usage

```
create_1D_mapper_object(
  data,
  dists,
  filtered_data,
  cover,
  clustering_method = "single"
)
```

Arguments

data	A data frame.
dists	A distance matrix for the data frame.
filtered_data	The result of a function applied to the data frame; there should be one row per observation in the original data frame.
cover	A 2D array of interval left and right endpoints.
clustering_method	Your favorite clustering algorithm.

Value

A list of two data frames, one with node data containing bin membership, data points per cluster, and cluster dispersion, and one with edge data containing sources, targets, and weights representing overlap strength.

Examples

```
data = data.frame(x = sapply(1:100, function(x) cos(x)), y = sapply(1:100, function(x) sin(x)))
projx = data$x

num_bins = 10
percent_overlap = 25

cover = create_width_balanced_cover(min(projx), max(projx), num_bins, percent_overlap)

create_1D_mapper_object(data, dist(data), projx, cover, "single")
```

---

create_balls *Make a cover of balls*

Description

Make a cover of balls

Usage

```
create_balls(data, dists, eps)
```

Arguments

data	A data frame.
dists	A distance matrix for the data frame.
eps	A positive real number.

Value

A list of vectors of data point names, one list element per ball. The output is such that every data point is contained in a ball of radius ε, and no ball center is contained in more than one ball. The centers are datapoints themselves.

Examples

```
num_points = 5000

P.data = data.frame(
  x = sapply(1:num_points, function(x)
    sin(x) * 10) + rnorm(num_points, 0, 0.1),
  y = sapply(1:num_points, function(x)
```

```
    cos(x) ^ 2 * sin(x) * 10) + rnorm(num_points, 0, 0.1),
  z = sapply(1:num_points, function(x)
    10 * sin(x) ^ 2 * cos(x)) + rnorm(num_points, 0, 0.1)
)

P.dist = dist(P.data)
balls = create_balls(data = P.data, dists = P.dist, eps = .25)
```

`create_ball_mapper_object` *Run mapper using a trivial filter, a cover of balls, and no clustering algorithm.*

Description

Run mapper using an ε-net cover (greedily generated) and the 2D inclusion function as a filter.

Usage

```
create_ball_mapper_object(data, dists, eps)
```

Arguments

`data`	A data frame.
`dists`	A distance matrix for the data frame.
`eps`	A positive real number for your desired ball radius.

Value

A list of two data frames, one with node data containing ball size, data points per ball, ball tightness, and one with edge data containing sources, targets, and weights representing overlap strength.

Examples

```
data = data.frame(x = sapply(1:100, function(x) cos(x)), y = sapply(1:100, function(x) sin(x)))
eps = .5

create_ball_mapper_object(data, dist(data), eps)
```

`create_bins` *Create bins of data*

Description

Create bins of data

Usage

```
create_bins(data, filtered_data, cover_element_tests)
```

Arguments

`data`	A data frame.
`filtered_data`	The result of a function applied to the data frame; there should be one row per observation in the original data frame.
`cover_element_tests`	A list of membership test functions for a list of cover elements. Each member of `cover_element_tests` should be able to identify (return `TRUE` or `FALSE`) if a single input data point is a member of the cover element it represents.

Value

A list of bins, each containing a vector of the names of the data inside it.

`create_clusterball_mapper_object` *Run clusterball mapper*

Description

Run ball mapper, but additionally cluster within the balls. Can use two different distance matrices to accomplish this.

Usage

```
create_clusterball_mapper_object(data, dist1, dist2, eps, clustering_method)
```

Arguments

`data`	A data frame.
`dist1`	A distance matrix for the data frame; this will be used to ball the data.
`dist2`	Another distance matrix for the data frame; this will be used to cluster the data after balling.
`eps`	A positive real number for your desired ball radius.
`clustering_method`	Your favorite clustering algorithm.

Value

A list of two dataframes, one with node data containing bin membership, datapoints per cluster, and cluster dispersion, and one with edge data containing sources, targets, and weights representing overlap strength.

Examples

```
data = data.frame(x = sapply(1:100, function(x) cos(x)), y = sapply(1:100, function(x) sin(x)))
data.dists = dist(data)
eps = 1

create_clusterball_mapper_object(data, data.dists, data.dists, eps, "single")
```

---

create_mapper_object *Create a mapper object*

Description

Run the mapper algorithm on a data frame.

Usage

```
create_mapper_object(
  data,
  dists,
  filtered_data,
  cover_element_tests,
  method = "none"
)
```

Arguments

data	A data frame.
dists	A distance matrix for the data frame.
filtered_data	The result of a function applied to the data frame; there should be one row per observation in the original data frame.
cover_element_tests	A list of membership test functions for a list of cover elements. Each member of cover_element_tests should be able to identify (return TRUE or FALSE) if a single input data point is a member of the cover element it represents.
method	The desired clustering method to use. e.g., "single"

Value

A list of two dataframes, one with node data containing bin membership, datapoints per cluster, and cluster dispersion, and one with edge data containing sources, targets, and weights representing overlap strength.

Examples

```
data = data.frame(x = sapply(1:100, function(x) cos(x)), y = sapply(1:100, function(x) sin(x)))
projx = data$x

num_bins = 10
percent_overlap = 25
xcover = create_width_balanced_cover(min(projx), max(projx), num_bins, percent_overlap)

check_in_interval <- function(endpoints) {
 return(function(x) (endpoints[1] - x <= 0) & (endpoints[2] - x >= 0))
}

# each of the "cover" elements will really be a function that checks if a data point lives in it
xcovercheck = apply(xcover, 1, check_in_interval)

# build the mapper object
xmapper = create_mapper_object(
  data = data,
  dists = dist(data),
  filtered_data = projx,
  cover_element_tests = xcovercheck,
  method = "single"
)
```

`create_single_bin` *Create a bin of data*

Description

Create a bin of data

Usage

```
create_single_bin(data, filtered_data, cover_element_test)
```

Arguments

`data`	A data frame.
`filtered_data`	The result of a function applied to the data frame; there should be one row per observation in the original data frame.
`cover_element_test`	A membership test function for a cover element. It should identify (return `TRUE` or `FALSE`) if a single input data point, is a member of the cover element it represents.

Value

A vector of names of points from the data frame, representing a bin.

`create_width_balanced_cover` *Generate an overlapping cover of an interval*

Description

This is a function that generates a cover of an interval $[a, b]$ with some number of (possibly) overlapping, evenly spaced, identical width subintervals.

Usage

```
create_width_balanced_cover(min_val, max_val, num_bins, percent_overlap)
```

Arguments

`min_val`	The left endpoint a. A real number.
`max_val`	The right endpoint b. A real number.
`num_bins`	The number of cover intervals with which to cover the interval. A positive integer.
`percent_overlap`	How much overlap desired between the cover intervals (the percent of the intersection of each interval with its immediate neighbor relative to its length, e.g., $[0, 2]$ and $[1, 3]$ would have 50% overlap). A real number between 0 and 100, inclusive.

Value

A 2D numeric array.

- left_ends - The left endpoints of the cover intervals.
- right_ends - The right endpoints of the cover intervals.

Examples

```
create_width_balanced_cover(min_val=0, max_val=100, num_bins=10, percent_overlap=15)
create_width_balanced_cover(-11.5, 10.33, 100, 2)
```

cut_dendrogram *Cut a dendrogram*

Description

Cut a dendrogram

Usage

```
cut_dendrogram(dend, threshold)
```

Arguments

dend	A single dendrogram.
threshold	A mininum tallest branch value.

Value

A named vector whose names are data point names and whose values are cluster labels. The number of clusters is determined to be 1 if the tallest branch of the dendrogram is less than the threshold, or if the index of dispersion (standard deviation squared divided by mean) of the branch heights is too low. Otherwise, we cut at the longest branch of the dendrogram to determine the number of clusters.

eccentricity_filter *Compute eccentricity of data points*

Description

Compute eccentricity of data points

Usage

```
eccentricity_filter(dists)
```

Arguments

dists	A distance matrix associated to a data frame.

Value

A vector of centrality measures, calcuated per data point as the sum of its distances to every other data point, divided by the number of points.

Examples

```
num_points = 5000

P.data = data.frame(
  x = sapply(1:num_points, function(x)
    sin(x) * 10) + rnorm(num_points, 0, 0.1),
  y = sapply(1:num_points, function(x)
    cos(x) ^ 2 * sin(x) * 10) + rnorm(num_points, 0, 0.1),
  z = sapply(1:num_points, function(x)
    10 * sin(x) ^ 2 * cos(x)) + rnorm(num_points, 0, 0.1)
)

P.dist = dist(P.data)
eccentricity = eccentricity_filter(P.dist)
```

---

get_bin_vector *Recover bins*

Description

Recover bins

Usage

```
get_bin_vector(binclust_data)
```

Arguments

binclust_data	A list of bins, each containing named vectors whose names are those of data points and whose values are cluster ids.

Value

A vector of integers equal in length to the number of clusters, whose members identify which bin that cluster belongs to.

---

get_clustered_data *Get data within a cluster*

Description

Get data within a cluster

Usage

```
get_clustered_data(binclust_data)
```

Arguments

binclust_data	A list of bins, each containing named vectors whose names are those of data points and whose values are cluster ids

Value

A list of strings, each a comma separated list of the toString values of the data point names.

---

get_clusters *Initate the clustering process*

Description

This function processes the binned data and global distance matrix to return freshly clustered data.

Usage

```
get_clusters(bins, dists, method)
```

Arguments

bins	A list containing "bins" of vectors of names of data points.
dists	A distance matrix containing pairwise distances between named data points.
method	A clue!

Value

A list containing named vectors (one per bin), whose names are data point names and whose values are cluster labels

---

get_cluster_sizes *Compute cluster sizes*

Description

Compute cluster sizes

Usage

```
get_cluster_sizes(binclust_data)
```

Arguments

binclust_data	A list of bins, each containing named vectors whose names are those of data points and whose values are cluster

Value

A vector of integers representing the lengths of the clusters in the input data.

`get_cluster_tightness_vector`
Compute dispersion measures of a list of clusters

Description

Compute dispersion measures of a list of clusters

Usage

```
get_cluster_tightness_vector(dists, binclust_data)
```

Arguments

`dists`	A distance matrix for the data points inside the input clusters
`binclust_data`	A list of bins, each containing named vectors whose names are those of data points and whose values are cluster ids

Value

A vector of real numbers in $(0, \infty)$ representing a measure of dispersion of a cluster, calculated according to `compute_tightness()`

`get_edgelist_from_overlaps`
Obtain edge list from cluster intersections

Description

Obtain edge list from cluster intersections

Usage

```
get_edgelist_from_overlaps(overlaps, num_vertices)
```

Arguments

`overlaps`	A named list of edges, whose elements contain the names of clusters in the overlap represented by that edge; output of `get_overlaps()`.
`num_vertices`	The number of vertices in the graph.

Value

A 2D array representing the edge list of a graph.

get_edge_weights *Calculate edge weights*

Description

Calculate edge weights

Usage

```
get_edge_weights(overlap_lengths, cluster_sizes, edges)
```

Arguments

overlap_lengths	A named vector of cluster overlap lengths, obtained by calling `length()` on the output from `[get_overlaps()]`.
cluster_sizes	A vector of cluster sizes.
edges	A 2D array of source and target nodes, representing an edge list. Should be ordered consistently with the `overlap_lengths` parameter.

Value

A vector of real numbers representing cluster overlap strength. This is calculated per edge by dividing the number of data points in the overlap by the number of points in the cluster on either end, and taking the maximum value.

get_overlaps *Get cluster overlaps*

Description

Get cluster overlaps

Usage

```
get_overlaps(binclust_data)
```

Arguments

binclust_data	A list of bins, each containing named vectors whose names are those of data points and whose values are cluster ids.

Value

A named list of edges, whose elements contain the names of clusters in the overlap represented by that edge.

get_single_linkage_clusters *Perform single linkage clustering and process dendrograms*

Description

Perform single linkage clustering and process dendrograms

Usage

```
get_single_linkage_clusters(dist_mats)
```

Arguments

dist_mats	A list of distance matrices to be used for clustering.

Value

A list containing named vectors (one per dendrogram), whose names are data point names and whose values are cluster labels

get_tallest_branch *Find the tallest branch of a dendrogram*

Description

Find the tallest branch of a dendrogram

Usage

```
get_tallest_branch(dend)
```

Arguments

dend	A single dendrogram.

Value

The height of the tallest branch (longest time between merge heights) of the input dendrogram.

is_in_ball *Get a tester function for a ball.*

Description

Get a tester function for a ball.

Usage

```
is_in_ball(ball)
```

Arguments

ball	A list of data points.

Value

A function that eats a data point and returns TRUE or FALSE depending if the point is in the ball or not.

mapper_object_to_igraph *make igraph*

Description

make igraph

Usage

```
mapper_object_to_igraph(mapperobject)
```

Arguments

mapperobject	mapper object generated by mappeR

Value

an igraph object

Examples

```
data = data.frame(x = sapply(1:100, function(x) cos(x)), y = sapply(1:100, function(x) sin(x)))

projy = data$y

cover = create_width_balanced_cover(min(projy), max(projy), 10, 25)

mapperobj = create_1D_mapper_object(data, dist(data), data$y, cover, "single")

mapper_object_to_igraph(mapperobj)
```

---

next_triangular *Find which triangular number you're on*

Description

Find which triangular number you're on

Usage

```
next_triangular(x)
```

Arguments

x	A positive integer.

Value

The index of the next greatest or equal triangular number to x.

---

process_dendrograms *Cut many dendrograms*

Description

Cut many dendrograms

Usage

```
process_dendrograms(dends)
```

Arguments

dends	A list of dendrograms to be cut.

Value

A list of named vectors (one per dendrogram) whose names are data point names and whose values are cluster labels. This function determines a global minimum threshold based on the longest branches in all the input dendrograms, and uses that as a heuristic to gauge if the best number of clusters is 1, or the value obtained by cutting the longest branch.

---

run_cluster_machine *Ship data off to the clustering goblins*

Description

This function tells the computer to look away for a second, so the goblins come and cluster your data while it's not watching.

Usage

```
run_cluster_machine(dist_mats, method)
```

Arguments

dist_mats	A list of distance matrices of each bin that is to be clustered.
method	A string that suggests how the goblins will handle the data.

Value

A list containing named vectors (one per bin), whose names are data point names and whose values are cluster labels (within each bin)

---

run_mapper *Construct mapper graph from data*

Description

Construct mapper graph from data

Usage

```
run_mapper(binclust_data, dists, binning = TRUE)
```

Arguments

binclust_data	A list of bins, each containing named vectors whose names are those of data points and whose values are cluster ids
dists	A distance matrix for the data that has been binned and clustered.
binning	Whether the output dataframe should sort vertices into "bins" or not. Should be true if using clustering, leave false otherwise

Value

A list of two dataframes, one with node data containing bin membership, datapoints per cluster, and cluster dispersion, and one with edge data containing sources, targets, and weights representing overlap strength.

---

run_slink *Perform single linkage clustering*

Description

Perform single linkage clustering

Usage

```
run_slink(dist)
```

Arguments

dist	A distance matrix.

Value

A dendrogram generated by `fastcluster`.

---

subset_dists *Subset a distance matrix*

Description

Subset a distance matrix

Usage

```
subset_dists(bin, dists)
```

Arguments

bin	A list of names of data points.
dists	A distance matrix for data points in the bin, possibly including extra points.

Value

A distance matrix for only the data points in the input bin.

Index

check_in_interval, 2
compute_tightness, 3
compute_tightness(), *14*
convert_to_clusters, 4
create_1D_mapper_object, 4
create_ball_mapper_object, 6
create_balls, 5
create_bins, 7
create_clusterball_mapper_object, 7
create_mapper_object, 8
create_single_bin, 9
create_width_balanced_cover, 10
cut_dendrogram, 11

eccentricity_filter, 11

get_bin_vector, 12
get_cluster_sizes, 13
get_cluster_tightness_vector, 14
get_clustered_data, 12
get_clusters, 13
get_edge_weights, 15
get_edgelist_from_overlaps, 14
get_overlaps, 15
get_overlaps(), *14*
get_single_linkage_clusters, 16
get_tallest_branch, 16

is_in_ball, 17

length(), *15*

mapper_object_to_igraph, 17

next_triangular, 18

process_dendrograms, 18

run_cluster_machine, 19
run_mapper, 19
run_slink, 20

subset_dists, 20