Package 'mappeR'

June 28, 2025

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>.

License MIT + file LICENSE

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

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

Version 2.2.0

Encoding UTF-8

Imports fastcluster, 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 2025-06-28 18:10:02 UTC

Contents

`create_1D_mapper_object` *One-Dimensional Mapper*

Description

Run Mapper using a one-dimensional filter, a cover of the codomain of intervals, and a clusterer.

Usage

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

Arguments

`data`	A data frame.
`dists`	A distance matrix associated to the data frame. Can be a `dist` object or `matrix`.
`filtered_data`	The result of a function applied to the data frame; there should be one filter value per observation in the original data frame. These values need to be named, and the names of these values must match the names of the original data set.
`cover`	An $n \times 2$ `matrix` of interval left and right endpoints; rows should be intervals and columns left and right endpoints (in that order).
`clusterer`	A function which accepts a list of distance matrices as input, and returns the results of clustering done on each distance matrix; that is, it should return a list of named vectors, whose name are the names of data points and whose values are cluster assignments (integers). If this value is omitted, then trivial clustering will be done.

Value

A `list` of two data frames, `nodes` and `edges`, which contain information about the Mapper graph constructed from the given parameters.

The node data frame consists of:

- `id`: vertex ID
- `cluster_size`: number of data points in vertex

- `mean_dist_to_medoid`: mean distance to medoid of vertex
- `data`: names of data points in cluster
- `patch`: level set ID

The edge data frame contains consists of:

- `source`: vertex ID of edge source
- `target`: vertex ID of edge target
- `weight`: Jaccard index of edge; this is the size of the intersection between the vertices divided by the union
- `overlap_data`: names of data points in overlap
- `overlap_size`: number of data points overlap

Examples

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

# Project to horizontal axis as lens
projx = data$x
names(projx) = row.names(data)

# Create a one-dimensional cover
num_bins = 10
percent_overlap = 25
cover = create_width_balanced_cover(min(projx), max(projx), num_bins, percent_overlap)

# Build Mapper object
create_1D_mapper_object(data, dist(data), projx, cover)
```

`create_balls` *Greedy Baller*

Description

Make a greedy epsilon net of a data set.

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 themselves are data points.

Examples

```
# Create a data set from 5000 points sampled from a parametric curve, plus some noise
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)

# Ball it up
balls = create_balls(data = P.data, dists = P.dist, eps = .25)
```

---

create_ball_mapper_object

Ball Mapper

Description

Run Mapper using the identity function as a lens and an ε-net cover, greedily generated using a distance matrix.

Usage

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

Arguments

`data`	A data frame.
`dists`	A distance matrix for the data frame. Can be a `dist` object or a `matrix`.
`eps`	A positive real number for the desired ball radius.

Value

A `list` of two data frames, `nodes` and `edges`, which contain information about the Mapper graph constructed from the given parameters.

The node data frame consists of:

- `id`: vertex ID

- `cluster_size`: number of data points in vertex
- `mean_dist_to_medoid`: mean distance to medoid of vertex
- `data`: names of data points in cluster

The edge data frame contains consists of:

- `source`: vertex ID of edge source
- `target`: vertex ID of edge target
- `weight`: Jaccard index of edge; this is the size of the intersection between the vertices divided by the union
- `overlap_data`: names of data points in overlap
- `overlap_size`: number of data points overlap

Examples

```
# Create noisy cirle data set
data = data.frame(x = sapply(1:100, function(x) cos(x)), y = sapply(1:100, function(x) sin(x)))

# Set ball radius
eps = .5

# Create Mapper object
create_ball_mapper_object(data, dist(data), eps)
```

`create_clusterball_mapper_object` *ClusterBall Mapper*

Description

Run Ball Mapper, but non-trivially cluster within the balls. You can use two different distance matrices to for the balling and clustering.

Usage

```
create_clusterball_mapper_object(
  data,
  dist1,
  dist2,
  eps,
  clusterer = local_hierarchical_clusterer("single")
)
```

Arguments

data	A data frame.
dist1	A distance matrix for the data frame; this will be used to ball the data. It can be a dist object or a matrix.
dist2	Another distance matrix for the data frame; this will be used to cluster the data after balling. It can be a dist object or a matrix.
eps	A positive real number for the desired ball radius.
clusterer	A function which accepts a list of distance matrices as input, and returns the results of clustering done on each distance matrix; that is, it should return a list of named vectors, whose name are the names of data points and whose values are cluster assignments (integers). If this value is omitted, then single-linkage clustering will be done (and a cutting height will be decided for you).

Value

A list of two data frames, nodes and edges, which contain information about the Mapper graph constructed from the given parameters.

The node data frame consists of:

- id: vertex ID
- cluster_size: number of data points in vertex
- mean_dist_to_medoid: mean distance to medoid of vertex
- data: names of data points in cluster
- patch: level set ID

The edge data frame contains consists of:

- source: vertex ID of edge source
- target: vertex ID of edge target
- weight: Jaccard index of edge; this is the size of the intersection between the vertices divided by the union
- overlap_data: names of data points in overlap
- overlap_size: number of data points overlap

Examples

```
# Create noisy circle data set
data = data.frame(x = sapply(1:100, function(x) cos(x)), y = sapply(1:100, function(x) sin(x)))
data.dists = dist(data)

# Set ball radius
eps = 1

# Do single-linkage clustering in the balls to produce Mapper graph
create_clusterball_mapper_object(data, data.dists, data.dists, eps)
```

create_mapper_object *Mapper*

Description

Run the Mapper algorithm on a data frame.

Usage

```
create_mapper_object(
  data,
  dists,
  filtered_data,
  cover_element_tests,
  clusterer = NULL
)
```

Arguments

data	A data frame.
dists	A distance matrix for the data frame. Can be a dist object or matrix.
filtered_data	The result of a function applied to the data frame; there should be one filter value per observation in the original data frame. These values need to be named, and the names of these values must match the names of the original data set.
cover_element_tests	A list of membership test functions for a set of cover elements. In other words, each element of cover_element_tests is a function that returns TRUE or FALSE when given a filter value.
clusterer	A function which accepts a list of distance matrices as input, and returns the results of clustering done on each distance matrix; that is, it should return a list of named vectors, whose names are the names of data points and whose values are cluster assignments (integers). If this value is omitted, then trivial clustering will be done.

Value

A list of two data frames, one with node data and one with edge data. The node data includes:

- id: vertex ID
- cluster_size: number of data points in cluster
- mean_dist_to_medoid: mean distance to medoid of cluster
- data: names of data points in cluster
- patch: level set ID

The edge data includes:

- `source`: vertex ID of edge source
- `target`: vertex ID of edge target
- `weight`: Jaccard index of edge; intersection divided by union
- `overlap_data`: names of data points in overlap
- `overlap_size`: number of data points overlap

Examples

```
# Create noisy data around a circle
data = data.frame(x = sapply(1:100, function(x) cos(x)), y = sapply(1:100, function(x) sin(x)))

# Apply lens function to data
projx = data$x
names(projx) = row.names(data)

# Build a width-balanced cover with 10 intervals and 25 percent overlap
num_bins = 10
percent_overlap = 25
xcover = create_width_balanced_cover(min(projx), max(projx), num_bins, percent_overlap)

# Write a function which can check if a data point lives in an interval of our lens function
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
)
```

`create_width_balanced_cover` *Width-Balanced Cover Maker*

Description

Generate a cover of an interval $[a, b]$ with uniformly sized and spaced 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

```
# Cover `[0, 100]` in 10 patches with 15% overlap
create_width_balanced_cover(min_val=0, max_val=100, num_bins=10, percent_overlap=15)

# Cover `[-11.5, 10.33]` in 100 patches with 2% overlap
create_width_balanced_cover(-11.5, 10.33, 100, 2)
```

`eccentricity_filter` *Eccentric Lens*

Description

Compute eccentricity of data points.

Usage

```
eccentricity_filter(dists)
```

Arguments

`dists`	A distance matrix associated to a data frame. Can be a `dist` object or `matrix`.

Value

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

Examples

```
# Generate some noisy data along a 2D curve
num_points = 100
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)
)
P.dist = dist(P.data)

# Apply eccentricity filter
eccentricity = eccentricity_filter(P.dist)
```

---

global_hierarchical_clusterer

Global Longevity Clusterer

Description

Create a dude to perform hierarchical clustering in a global context using the hclust package.

Usage

```
global_hierarchical_clusterer(method, dists, cut_height = -1)
```

Arguments

method	A string to pass to hclust to tell it what kind of clustering to do.
dists	The global distance matrix on which to run clustering to determine a global cutting height.
cut_height	The cutting height at which you want all dendrograms to be cut. If this is not specified then the clusterer will use a cut height 5% above the merge point preceding the tallest branch in the global dendrogram.

Details

This clusterer cuts all dendrograms it is given at a uniform cutting height, defaulting to a heuristic if necessary.

Value

A function that inputs a list of distance matrices and returns a list containing one vector per matrix, whose element names are data point names and whose values are cluster labels (relative to each matrix).

Examples

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

dists = dist(data)

num_bins = 10
percent_overlap = 25

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

create_1D_mapper_object(data, dists, projx, cover, global_hierarchical_clusterer("mcquitty", dists))
```

`local_hierarchical_clusterer` *Local Longevity Clusterer*

Description

Create a dude to perform hierarchical clustering in a local context using the hclust package.

Usage

```
local_hierarchical_clusterer(method)
```

Arguments

`method`	A string to pass to hclust to tell it what kind of clustering to do.

Details

This clusterer determines cutting heights for dendrograms by cutting them individually, 5% above the merge point with the longest unbroken gap until the next merge point.

Value

A function that inputs a list of distance matrices and returns a list containing one vector per matrix, whose element names are data point names and whose values are cluster labels (within each patch).

Examples

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

dists = dist(data)

num_bins = 10
```

```
percent_overlap = 25

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

create_1D_mapper_object(data, dists, projx, cover, local_hierarchical_clusterer("mcquitty"))
```

Index