

# Package ‘Mercator’

May 7, 2026

**Version** 1.1.7

**Date** 2025-05-27

**Title** Clustering and Visualizing Distance Matrices

**Description** Defines the classes used to explore, cluster and visualize distance matrices, especially those arising from binary data. See Abrams and colleagues, 2021, <[doi:10.1093/bioinformatics/btab037](https://doi.org/10.1093/bioinformatics/btab037)>.

**Depends** R (>= 3.5), Thresher (>= 1.1)

**Imports** methods, stats, utils, graphics, grDevices, KernSmooth, cluster, Rtsne, ClassDiscovery, Polychrome, dendextend, igraph, flexmix, umap, kohonen

**VignetteBuilder** knitr

**Suggests** knitr, rmarkdown

**License** Apache License (== 2.0)

**biocViews** Clustering

**URL** <http://oompa.r-forge.r-project.org/>

**NeedsCompilation** no

**Author** Kevin R. Coombes [aut, cre],  
Caitlin E. Coombes [aut]

**Maintainer** Kevin R. Coombes <[krc@silicovore.com](mailto:krc@silicovore.com)>

**Repository** CRAN

**Date/Publication** 2025-05-27 18:40:02 UTC

## Contents

binaryDistance . . . . .	2
BinaryMatrix-class . . . . .	3
downsample . . . . .	5
Mercator-class . . . . .	6
mercator-data . . . . .	9
removeDuplicates . . . . .	10
setClusters . . . . .	11
threshLGF . . . . .	13

---

binaryDistance	<i>Binary Distance Definitions Across Methods</i>
----------------	---

---

### Description

The `binaryDistance` function defines various similarity or distance measures between binary vectors, which represent the first step in the algorithm underlying the Mercator visualizations.

### Usage

```
binaryDistance(X, metric)
```

### Arguments

<code>X</code>	An object of class <code>matrix</code> .
<code>metric</code>	An object of class <code>character</code> limited to the names of 10 selected distance metrics: <code>jaccard</code> , <code>sokalMichener</code> , <code>hamming</code> , <code>russellRao</code> , <code>pearson</code> , <code>goodmanKruskal</code> , <code>manhattan</code> , <code>canberra</code> , <code>binary</code> , or <code>euclid</code> .

### Details

Similarity or difference between binary vectors can be calculated using a variety of distance measures. In the main reference (below), Choi and colleagues reviewed 76 different measures of similarity of distance between binary vectors. They also produced a hierarchical clustering of these measures, based on the correlation between their distance values on multiple simulated data sets. For metrics that are highly similar, we chose a single representative.

Cluster 1, represented by the `jaccard` distance, contains Dice & Sorenson, Ochiai, Kulczynski, Bray & Curtis, Baroni-Urbani & Buser, and Jaccard.

Cluster 2, represented by the `sokalMichener` distance, contains Sokal & Sneath, Gilbert & Wells, Gower & Legendre, Pearson & Heron, Hamming, and Sokal & Michener. Also within this cluster are 4 distances represented independently within this function: `hamming`, `manhattan`, `canberra`, and euclidean distances

Cluster 3, represented by the `russellRao` distance, contains Driver & Kroeber, Forbes, Fossum, and Russell & Rao.

The remaining metrics are more isolated, without strong clustering. We considered a few examples, including the Pearson distance (`pearson`) and the Goodman & Kruskal distance (`goodmanKruskal`). The binary distance is also included.

### Value

Returns an object of class `dist` corresponding to the distance `metric` provided.

**Note**

Although the distance metrics provided in the `binaryDistance` function are explicitly offered for use on matrices of binary vectors, some metrics may return useful distances when applied to non-binary matrices.

**Author(s)**

Kevin R. Coombes <krc@silicovore.com>, Caitlin E. Coombes

**References**

Choi SS, Cha SH, Tappert CC, A Survey of Binary Similarity and Distance Measures. *Systemics, Cybernetics, and Informatics*. 2010; 8(1):43-48.

**See Also**

This set includes all of the metrics from the `dist` function.

**Examples**

```
my.matrix <- matrix(rbinom(50*100, 1, 0.15), ncol=50)
my.dist <- binaryDistance(my.matrix, "jaccard")
```

---

BinaryMatrix-class      *Class "BinaryMatrix"*

---

**Description**

The `BinaryMatrix` object class underlies the `threshLGF` and `Mercator` methods and visualizations of the `Mercator` package. The `BinaryMatrix` function returns a new object of `BinaryMatrix` class.

**Usage**

```
BinaryMatrix(binmat, columnInfo, rowInfo)
```

**Arguments**

<code>binmat</code>	A binary matrix of class <code>numeric</code> or <code>integer</code> with values 0 and 1.
<code>columnInfo</code>	A <code>data.frame</code> of at least one column containing column names. If no <code>colnames</code> are specified, the column names associated with the data matrix will be used.
<code>rowInfo</code>	A <code>dataframe</code> of at least one column containing row names. If no <code>rownames</code> are specified, the row names associated with the data matrix will be used.

**Value**

The `BinaryMatrix` function returns a new object of `binaryMatrix` class.

## Objects from the Class

Objects should be defined using the `BinaryMatrix` constructor. In the simplest case, you simply pass in the binary data matrix that you want to visualize, and the `BinaryMatrix` is constructed using the matrix's existing column and row names.

## Slots

`binmat`: Object of class `matrix`; the binary data used for visualization.

`columnInfo`: Object of class `data.frame`; names and definitions of columns.

`rowInfo`: Object of class `data.frame`; names and definitions of rows.

`info`: Object of class `list`; identifies `$notUsed` and `$redundant` features.

`history`: Object of class "character"; returns a history of manipulations by Mercator functions to the `BinaryMatrix` object, including "Newly created," "Subsetted," "Transposed," "Duplicate features removed," and "Threshed."

## Methods

`[]`: Subsetting by `[]` returns a subsetted binary matrix, including subsetted row and column names. Calling `@history` will return the history "Subsetted."

`dim`: returns the dimensions of the `@binmat` component of the `BinaryMatrix` object.

`print`: Shows the first ten rows and columns of the `@binmat` component.

`show`: Shows the first ten rows and columns of the `@binmat` component.

`summary`: For a given `BinaryMatrix`, returns object class, dimensions of the `@binmat` component, and `@history`.

`t`: Transposes the `@binmat` and its associated `rowInfo` and `columnInfo`. Calling `@history` will return the history "Subsetted."

## Note

Attempting to construct or manipulate a `BinaryMatrix` containing NAs, missing values, or columns containing exclusively 0 values may introduce error.

## Author(s)

Kevin R. Coombes <krc@silicovore.com>, Caitlin E. Coombes

## See Also

The [removeDuplicateFeatures](#) function can be used to remove duplicate columns from the `BinaryMatrix` class before threshing or visualization. The [threshLGF](#) can be used to identify and remove uninformative features before visualization or further analysis.

**Examples**

```

my.matrix <- matrix(rbinom(50*100, 1, 0.15), ncol=50)
my.rows <- as.data.frame(paste("R", 1:100, sep=""))
my.cols <- as.data.frame(paste("C", 1:50, sep=""))
my.binmat <- BinaryMatrix(my.matrix, my.cols, my.rows)
summary(my.binmat)
my.binmat <- my.binmat[1:50, 1:30]
my.binmat <- t(my.binmat)
dim(my.binmat)
my.binmat@history
my.binmat

```

downsample

*Downsampling a Distance Visualization to Facilitate iGraph***Description**

The `downsample` function implements a structured reduction of data points with a parent Mercator distance visualization to improve visualization and computational time, especially for the implementation of the `iGraph` visualization.

**Usage**

```
downsample(target, distanceMat, cutoff)
```

**Arguments**

<code>target</code>	An integer number of points to which the user wishes to reduce the parent Mercator object.
<code>distanceMat</code>	An object of class <code>matrix</code> containing the distance matrix component of the parent Mercator object.
<code>cutoff</code>	An inclusion cutoff for selected points based on the local density within the parent data.

**Details**

`Mercator` can be used to visualize complex networks using `iGraph`. To improve clarity of the visualization and computational time, we implement the `downsample` function to reduce the number of data points to be linked and visualized. The conceptual grounding for `downsample` draws on Peng Qiu's implementation of the SPADE clustering algorithm for mass cytometry data. The `downsample` function under-samples the densest regions of the data space to make it more likely that rarer clusters will still be adequately sampled.

**Value**

`downsample` returns an object of class `Mercator` containing a structured subset of items from the parent Mercator object.

**Author(s)**

Kevin R. Coombes <krc@silicovore.com>, Caitlin E. Coombes

**References**

Qiu, P., et. al. (2011). Extracting a cellular hierarchy from high-dimensional cytometry data with SPADE. *Nature biotechnology*, 29(10), 886.

**Examples**

```
#Form a BinaryMatrix
data("iris")
my.data <- as.matrix(iris[,c(1:4)])
my.rows <- as.data.frame(c(1:length(my.data[,1])))
my.binmat <- BinaryMatrix(my.data, , my.rows)
my.binmat <- t(my.binmat)
summary(my.binmat)

# Form and plot Mercator object
# Set K to the known number of species in the dataset
my.vis <- Mercator(my.binmat, "euclid", "tsne", K=3)
summary(my.vis)
plot(my.vis, view = "tsne", main="t-SNE plot of all data points")

#Downsample the Mercator object
M <- as.matrix(my.vis@distance)
set.seed(21340)
DS <- downsample(50, M, 0.1)
red.vis <- my.vis[DS]

#Visualize the down sampled t-SNE plot
plot(red.vis, view = "tsne", main="Down sampled t-SNE Plot")
```

---

Mercator-class

*The Mercator Distance Visualization Object*

---

**Description**

The Mercator object represents a distance matrix together with clustering assignments and a set of visualizations. It implements four visualizations for clusters of large-scale, multi-dimensional data: hierarchical clustering, multi-dimensional scaling, t-Stochastic Neighbor Embedding (t-SNE), and iGraph. The default Mercator constructor applies one of ten metrics of [binaryDistance](#) to an object of the [BinaryMatrix](#) class.

**Usage**

```
Mercator(X, metric, method, K, ...)
addVisualization(DV, method, ...)
getClusters(DV)
```

**Arguments**

X	Either a <a href="#">BinaryMatrix</a> or a <a href="#">dist</a> object.
metric	A <a href="#">binaryDistance</a> currently limited to the names of 10 selected distance metrics: jaccard, sokalMichener, hamming, russellRao, pearson, goodmanKruskal, manhattan, canberra, binary, or euclid.
method	A visualization method, currently limited to hclust, mds, tsne, graph, umap, and som.
K	An integer specifying the number of desired clusters.
DV	A distance visualization produced as the output of the Mercator function.
...	Additional arguments passed on to the functions that implement different methods for <code>addVisualization</code> (possibly passed here through the Mercator function). These include <ul style="list-style-type: none"> <li>any Any arguments to the <a href="#">cmdscale</a> function for an mds visualization. Most commonly, this is likely to include <code>k</code>, the number of dimensions to compute.</li> <li>any Any arguments to the <a href="#">Rtsne</a> function for a tsne visualization. Common examples include <code>dims</code> and <code>perplexity</code>.</li> <li>any Any arguments to the <a href="#">som</a> function to construct or view a self-organizing map. Most commonly, this is likely to include <code>somgrid</code>, to specify the size and layout of the grid, or <code>type</code> to specify which data to show in the plot.</li> <li>Q A quantile cutoff for the creation of the IGraph visualization. By default, the value is set at the 10th percentile.</li> </ul>

**Value**

The `Mercator` function constructs and returns a distance visualization object of the `Mercator` class, including a distance matrix calculated on a given metric and given visualizations. It is also possible (though not advisable) to construct a `Mercator` object directly using the new function. Default clustering in `Mercator` is now performed on the distance matrix using hierarchical clustering (`hclust`) with the `wardD2` linkage method.

The `addVisualizations` function can be used to add additional visualizations to an existing `Mercator` object.

The `getClusters` function returns a vector of cluster assignments.

**Slots**

`metric`: Object of class "character"; the name of the [binaryDistance](#) applied to create this object.

`distance`: Object of class "dist"; the distance matrix used and represented by this object.

`view`: Object of class "list"; contains the results of calculations to generate each visualize the object.

`clusters`: A numeric vector of cluster assignments.

`symbols`: A numeric vector of valid plotting characters, as used by `par(pch)`.

`palette`: A character vector of color names.

## Methods

**plot(x, view = NULL, ask = NULL, ...):** Produce a plot of one or more visualizations within a Mercator object. The default view, when omitted, is the first one contained in the object. You can request multiple views at once; if the current plot layout doesn't have enough space in an interactive session, the ask parameters determines whether the system will ask you before advancing to the next plot. When plotting a graph view, you can use an optional layout parameter to select a specific layout by name.

**barplot(height, main = "", sub = NULL, border = NA, space = 0, ...)** Produce a (colored) barplot of the silhouette widths for elements clustered in this class. Arguments are as described in the base function [barplot](#).

**scatter(object, view = NULL, ask = NULL, colramp = NULL, ...):** Produce a smooth scatter plot of one or more visualizations within a Mercator object. The default view, when omitted, is the first one contained in the object. You can request multiple views at once; if the current plot layout doesn't have enough space in an interactive session, the ask parameters determines whether the system will ask you before advancing to the next plot. When plotting a graph view, you can use an optional layout parameter to select a specific layout by name. Arguments are otherwise the same as the [smoothScatter](#) function, except that the default color ramp is `topo.colors`.

**hist(x, breaks=123, ...):** Produce a histogram of distances calculated in the dissimilarity matrix generated in the Mercator object.

**summary(object, ...):** Returns the chosen distance metric, dimensions of the distance matrix, and available, calculated visualizations in this object.

**dim(x):** Returns the dimensions of the distance matrix of this object.

**[:** Subsets the distance matrix of this object.

## Author(s)

Kevin R. Coombes <krc@silicovore.com>, Caitlin E. Coombes

## See Also

[silhouette](#), [smoothScatter](#), [topo.colors](#), [som](#), [umap](#).

## Examples

```
# Form a BinaryMatrix
data("iris")
my.data <- as.matrix(iris[,c(1:4)])
my.rows <- as.data.frame(c(1:length(my.data[,1])))
my.binmat <- BinaryMatrix(my.data, , my.rows)
my.binmat <- t(my.binmat)
summary(my.binmat)

# Form a Mercator object
# Set K to the known number of species in the dataset
my.vis <- Mercator(my.binmat, "euclid", "hclust", K=3)
summary(my.vis)
hist(my.vis)
```

```
barplot(my.vis)
my.vis <- addVisualization(my.vis, "mds")
plot(my.vis, view = "hclust")
plot(my.vis, view = "mds")

scatter(my.vis, view = "mds")

# change the color palette
slot(my.vis, "palette") <- c("purple", "red", "orange", "green")
scatter(my.vis, view = "mds")

# Recover cluster identities
# What species comprise cluster 1?
my.clust <- getClusters(my.vis)
my.species <- iris$Species[my.clust == 1]
my.species
```

---

mercator-data

*CML Cytogenetic Data*

---

## Description

These data sets contain binary versions of subsets of cytogenetic karyotype data from patients with chronic myelogenous leukemia (CML).

## Usage

```
data("lgfFeatures")
data("CML500")
data("CML1000")
data("fakedata") # includes "fakeclin"
```

## Format

**lgfFeatures** A data matrix with 2748 rows and 6 columns listing the cytogenetic bands produced as output of the CytoGPS algorithm that converts text-based karyotypes into a binary Loss-Gain-Fusion (LGF) model. The columns include the Label (the Type and Band, joined by an underscore), Type (Loss, Gain, or Fusion), Band (standard name of the cytogenetic band), Chr (chromosome), Arm (the chromosome arm, of the form #p or #q), and Index (an integer that can be used for sorting or indexing).

**CML500** A [BinaryMatrix](#) object with 770 rows (subset of LGF features) and 511 columns (patients). The patients were selected using the [downsample](#) function from the full set of more than 3000 CML karyotypes. The rows were selected by removing redundant and non-informative features when considering the full data set.

**CML1000** A [BinaryMatrix](#) object with 770 rows (subset of LGF features) and 1057 columns (patients). The patients were selected using the [downsample](#) function from the full set of more than 3000 CML karyotypes. The rows were selected by removing redundant and non-informative features when considering the full data set.

fakedata A matrix with 776 rows ("features") and 300 columns ("samples") containing synthetic continuous data.

fakeclin A data frame with 300 rows ("samples") and 4 columns of synthetic clinical data related to the fakedata.

### Author(s)

Kevin R. Coombes <krc@silicovore.com>, Caitlin E. Coombes

### Source

The cytogenetic data were obtained from the public Mitelman Database of Chromosomal Aberrations and Gene Fusions in Cancer on 4 April 2019. The database is currently located at <https://cgap.nci.nih.gov/Chromosomes> as part of the Cancer Genome Anatomy Project (CGAP). The CGAP web site is expected to close on 1 October 2019 at which point the Mitelman database will move to an as-yet-undisclosed location. The data were then converted from text-based karyotypes into binary vectors using CytoGPS <http://cytogps.org/>.

### References

Abrams ZB, Zhang L, Abruzzo LV, Heerema NA, Li S, Dillon T, Rodriguez R, Coombes KR, Payne PRO. CytoGPS: A Web-Enabled Karyotype Analysis Tool for Cytogenetics. *Bioinformatics*. 2019 Jul 2. pii: btz520. doi: 10.1093/bioinformatics/btz520. [Epub ahead of print]

---

removeDuplicates

*Remove Duplicate Features or Samples from a Binary Matrix Object*

---

### Description

The removeDuplicates function removes duplicate columns from a binaryMatrix object in the Mercator package.

### Usage

```
removeDuplicates(object)
removeDuplicateFeatures(object)
```

### Arguments

object            An object of class binaryMatrix.

## Details

In some analyses, it may be desirable to remove duplicate features to collapse a group of identical, related events to a single feature, to prevent overweighting when clustering. Historically, this function was called `removeDuplicateFeatures`. That name is still retained for backwards compatibility, but it may be deprecated in future versions in favor of `removeDuplicates`. In the same way, for some clustering applications, it may be useful to remove duplicate samples, or those that have an identical feature set.

Removal of duplicates is not required for performance of the `binaryMatrix` or `Mercator` objects and associated functions.

The `history` slot of the `binaryMatrix` object documents the removal of duplicates.

Future versions of this package may include functionality to store the identities of any duplicates that were removed.

## Value

Returns an object of class `binaryMatrix` with duplicate columns removed.

## Note

Transposing the `binaryMatrix` can allow the `removeDuplicates` function to be applied to both features and observations, if desired.

Features containing exclusively 0s or 1s may interfere with performance of `removeDuplicates`.

## Author(s)

Kevin R. Coombes <krc@silicovore.com>, Caitlin E. Coombes

## Examples

```
my.matrix <- matrix(rbinom(50*100, 1, 0.15), ncol=50)
my.matrix <- cbind(my.matrix, my.matrix[, 1:5]) # add duplicates
dimnames(my.matrix) <- list(paste("R", 1:100, sep=''),
                           paste("C", 1:55, sep=''))
my.binmat <- BinaryMatrix(my.matrix)
dim(my.binmat)
my.binmat <- removeDuplicates(my.binmat)
dim(my.binmat)
```

## Description

Cluster assignments from unsupervised analyses typically assign arbitrary integers to the classes. When comparing the results of different algorithms or different distance metrics, it is helpful to match the integers in order to use colors and symbols that are as consistent as possible. These functions help achieve that goal.

## Usage

```
setClusters(DV, clusters)
recolor(DV, clusters)
remapColors(fix, vary)
recluster(DV, K)
```

## Arguments

DV	An object of the Mercator class.
clusters	An integer vector specifying the cluster membership of each element.
fix	An object of the Mercator class, used as the source of color and cluster assignments.
vary	An object of the Mercator class, used as the target of color and cluster assignments.
K	An integer, the number of clusters.

## Details

In the most general sense, clustering can be viewed as a function from the space of "objects" of interest into a space of "class labels". In less mathematical terms, this simply means that each object gets assigned an (arbitrary) class label. This is all well-and-good until you try to compare the results of running two different clustering algorithms that use different labels (or even worse, use the same labels – typically the integers  $1, 2, \dots, K$  – with different meanings). When that happens, you need a way to decide which labels from the different sets are closest to meaning the "same thing".

The functions `setClusters` and `remapColors` solve this problem in the context of Mercator objects. They accomplish this task using the greedy algorithm implemented and described in the `remap` function in the `Thresher` package.

## Value

Both `setClusters` and `remapColors` return an object of class `Mercator` in which only the cluster labels and associated color and symbol representations (but not the distance metric used, the number of clusters, the cluster assignments, nor the views) have been updated.

The `recolor` function is currently an alias for `setClusters`. However, using `recolor` is deprecated, and the alias will likely be removed in the next version of the package.

## Author(s)

Kevin R. Coombes <krc@silicovore.com>

## See Also

[remap](#)

## Examples

```

#Form a BinaryMatrix
data("iris")
my.data <- t(as.matrix(iris[,c(1:4)]))
colnames(my.data) <- 1:ncol(my.data)
my.binmat <- BinaryMatrix(my.data)

# Form a Mercator object; Set K to the number of known species
my.vis <- Mercator(my.binmat, "euclid", "tsne", K=3)
table(getClusters(my.vis), iris$Species)
summary(my.vis)

# Recolor the Mercator object with known species
DS <- recolor(my.vis, as.numeric(iris$Species))
table(getClusters(DS), iris$Species)

# Use a different metric
my.vis2 <- Mercator(my.binmat, "manhattan", "tsne", K=3)
table(getClusters(my.vis2), iris$Species)
table(Pearson = getClusters(my.vis2), Euclid = getClusters(my.vis))

# remap colors so the two methods match as well as possible
my.vis2 <- remapColors(my.vis, my.vis2)
table(Pearson = getClusters(my.vis2), Euclid = getClusters(my.vis))

# recluster with K=4
my.vis3 <- recluster(my.vis, K = 4)

# view the results
opar <- par(mfrow=c(1,2))
plot(my.vis, view = "tsne", main="t-SNE plot, Euclid")
plot(my.vis2, view = "tsne", main="t-SNE plot, Pearson")
par(opar)

```

---

threshLGF

*Threshing and Reaping the BinaryMatrix*

---

## Description

The threshLGF function produces an object of class `ThreshedBinaryMatrix` from threshing on an object of class `BinaryMatrix`.

The function `threshLGF` and the `ThreshedBinaryMatrix` object can be used to access the functionality of the `Thresher` R-package within Mercator.

## Usage

```
threshLGF(object, cutoff = 0)
```

**Arguments**

object	An object of class <code>BinaryMatrix</code>
cutoff	The value of <code>delta</code> set to demarcate an uninformative feature. Features with a value greater than the cutoff will be kept.

**Details**

The `Thresher` R-package provides a variety of functionalities for data filtering and the identification of and reduction to "informative" features. It performs clustering using a combination of outlier detection, principal component analysis, and von Mises Fisher mixture models. By identifying significant features, `Thresher` performs feature reduction through the identification and removal of noninformative features and the nonbiased calculation of the number of groups (K) for down-stream use.

**Value**

`threshLGF` returns an object of class `ThreshedBinaryMatrix`. The `ThreshedBinaryMatrix` object retains all the functionality, slots, and methods of the `BinaryMatrix` object class with added features. After threshing, the `ThreshedBinaryMatrix` records the history, "Threshed."

**Additional Slots**

`thresher`: Returns the functions of the `Thresher` object class of the `Thresher` R-package.

`reaper`: Returns the functions of the `Reaper` object class of the `Thresher` R-package.

**Note**

The `Thresher` R-package applies the Auer-Gervini statistic for principal component analysis, outlier detection, and identification of uninformative features on a `matrix` of class `integer` or `numeric`.

An initial `delta` of 0.3 is recommended.

**Author(s)**

Kevin R. Coombes <krc@silicovore.com>, Caitlin E. Coombes

**References**

Wang, M., Abrams, Z. B., Kornblau, S. M., & Coombes, K. R. (2018). `Thresher`: determining the number of clusters while removing outliers. *BMC bioinformatics*, 19(1), 9.

**See Also**

The `threshLGF` function creates a new object of class `ThreshedBinaryMatrix` from an object of class `BinaryMatrix`.

**Examples**

```
#Create a BinaryMatrix
set.seed(52134)
my.matrix <- matrix(rbinom(50*100, 1, 0.15), ncol=50)
my.rows <- as.data.frame(paste("R", 1:100, sep=""))
my.cols <- as.data.frame(paste("C", 1:50, sep=""))
my.binmat <- BinaryMatrix(my.matrix, my.cols, my.rows)
summary(my.binmat)

#Identify delta cutoff and thresh
my.binmat <- threshLGF(my.binmat)
Delta <- my.binmat@thresher@delta
sort(Delta)
hist(Delta, breaks=15, main="", xlab="Weight")
abline(v=0.3, col='red')
my.binmat <- threshLGF(my.binmat, cutoff = 0.3)
summary(my.binmat)

#Principal Component Analysis
my.binmat@reaper@pcdim
my.binmat@reaper@nGroups
plot(my.binmat@reaper@ag)
abline(h=1, col="red")
screplot(my.binmat@reaper)
abline(v=6, col="forestgreen", lwd=2)
```

# Index

- \* **cluster**
  - binaryDistance, 2
  - BinaryMatrix-class, 3
  - downsample, 5
  - Mercator-class, 6
  - removeDuplicates, 10
  - setClusters, 11
  - threshLGF, 13
- \* **datasets**
  - mercator-data, 9
- [, BinaryMatrix, ANY, ANY, ANY-method (BinaryMatrix-class), 3
- [, Mercator, ANY, ANY, ANY-method (Mercator-class), 6
- addVisualization (Mercator-class), 6
- barplot, 8
- barplot, Mercator-method (Mercator-class), 6
- binaryDistance, 2, 6, 7
- BinaryMatrix, 6, 7, 9, 13, 14
- BinaryMatrix (BinaryMatrix-class), 3
- BinaryMatrix-class, 3
- cmdscale, 7
- CML1000 (mercator-data), 9
- CML500 (mercator-data), 9
- dim, 4
- dim, BinaryMatrix-method (BinaryMatrix-class), 3
- dim, Mercator-method (Mercator-class), 6
- dist, 2, 3, 7
- downsample, 5, 9
- fakeclin (mercator-data), 9
- fakedata (mercator-data), 9
- getClusters (Mercator-class), 6
- goodmanKruskalDistance (binaryDistance), 2
- goodmanKruskalSimilarity (binaryDistance), 2
- hammingDistance (binaryDistance), 2
- hist, BinaryMatrix-method (BinaryMatrix-class), 3
- hist, Mercator-method (Mercator-class), 6
- jaccardDistance (binaryDistance), 2
- jaccardSimilarity (binaryDistance), 2
- lgfFeatures (mercator-data), 9
- Mercator (Mercator-class), 6
- Mercator-class, 6
- mercator-data, 9
- pearsonDistance (binaryDistance), 2
- pearsonSimilarity (binaryDistance), 2
- plot, Mercator, missing-method (Mercator-class), 6
- print, 4
- print, BinaryMatrix-method (BinaryMatrix-class), 3
- recluster (setClusters), 11
- recolor (setClusters), 11
- remap, 12
- remapColors (setClusters), 11
- removeDuplicateFeatures, 4
- removeDuplicateFeatures (removeDuplicates), 10
- removeDuplicates, 10
- Rtsne, 7
- russellRaoSimilarity (binaryDistance), 2
- russelRaoDistance (binaryDistance), 2
- scatter, Mercator-method (Mercator-class), 6

setClusters, [11](#)  
show, [4](#)  
show, BinaryMatrix-method  
    (BinaryMatrix-class), [3](#)  
silhouette, [8](#)  
smoothScatter, [8](#)  
sokalMichenerDistance (binaryDistance),  
    [2](#)  
sokalMichenerSimilarity  
    (binaryDistance), [2](#)  
som, [7](#), [8](#)  
summary, [4](#)  
summary, BinaryMatrix-method  
    (BinaryMatrix-class), [3](#)  
summary, Mercator-method  
    (Mercator-class), [6](#)  
  
t, [4](#)  
t, BinaryMatrix-method  
    (BinaryMatrix-class), [3](#)  
ThreshedBinaryMatrix, [13](#)  
ThreshedBinaryMatrix (threshLGF), [13](#)  
ThreshedBinaryMatrix-class (threshLGF),  
    [13](#)  
Thresher, [12–14](#)  
threshLGF, [4](#), [13](#), [13](#)  
topo.colors, [8](#)  
  
umap, [8](#)  
  
validBinaryMatrix (BinaryMatrix-class),  
    [3](#)