Package ‘Thresher’

December 17, 2019

Version 1.1.3

Date 2019-12-07

Title Threshing and Reaping for Principal Components

Author Kevin R. Coombes

Maintainer Kevin R. Coombes <krc@silicovore.com>

Description Defines the classes used to identify
outliers (threshing) and compute the number of significant principal
components and number of clusters (reaping) in a joint application
of PCA and hierarchical clustering. See Wang et al., 2018,
<doi:10.1186/s12859-017-1998-9>.

Depends R (>= 3.1), ClassDiscovery, PCDimension

Imports methods, stats, graphics, grDevices, MASS, colorspace, movMF,
ade4, oompaBase

Suggests NbClust

License Apache License (== 2.0)

biocViews Clustering

URL http://oompa.r-forge.r-project.org/

NeedsCompilation no

Repository CRAN

Date/Publication 2019-12-17 19:00:03 UTC

R topics documented:

`generics` *Generic Methods in the Thresher package*

Description

New generic functions for threshing and reaping datasets.

Usage

```
## S4 method for signature 'ANY'
getColors(object, ...)
## S4 method for signature 'ANY'
getSplit(object, ...)
## S4 method for signature 'ANY'
getStyles(object, ...)
## S4 method for signature 'ANY'
scatter(object, ...)
## S4 method for signature 'ANY'
heat(object, ...)
## S4 method for signature 'ANY'
makeFigures(object, DIR=NULL, ...)
```

Arguments

`object`	an object of any class, but typically `Thresher`, `Reaper`, or `SimThresher`.
`DIR`	a character string containing the name of an existing directory.
`...`	extra arguments for generic routines.

Details

The methods `getColors`, `getSplit`. and `getStyles` are used to extract the implicit color assignments used in standard plots of objects of the `Thresher` or `Reaper` classes.

The `heat` and `scatter` methods represent generic heat maps and scatter plots, respectively.

The `makeFigures` method is to generate a standard suite of figures for an object. If the `DIR` argument is not NULL, then the figures will be written to the indicated directory in PNG format. Otherwise, the figures will be displayed interactivey on screen, waiting for user input to show each plot.

Value

The form of the value returned by these functions may change depending on the class of its argument. See the documentation of the particular methods for details of what is produced by that method.

The `getColors` method should return a vector of colors with length equal to the number of columns in a data set.

The `getSplit` method should return a vector of colors with length equal to the number of rows in a data set.

The `heat`, `scatter`, and `makeFigures` methods are called for their side effects of producing plots.

Author(s)

Kevin R. Coombes <`krc@silicovore.com`>

See Also

`Thresher`, `Reaper`, and `SimThresher`.

`matchLabels` *Match Arbitrary Class Assignments Across Methods*

Description

These functions provide a set of tools to find the best match between the labels used by two different algorithms to cluster the same set of samples.

Usage

```
labelMatcher(tab, verbose = FALSE)
matchLabels(tab)
countAgreement(tab)
labelAccuracy(data, labels, linkage="ward.D2")
bestMetric(data, labels)
remap(fix, vary)
```

Arguments

`tab`	A contingency table, represented as a square `matrix` or `table` as an R object. Both dimensions represent an assignment of class labels, with each row and column representing one of the labels. Entries should be non-negative integer counts of the number of objects having the labels represented by the row and column.
`verbose`	A logical value; should the routine print something out periodically so you know it's still working?
`data`	A matrix whose columns represent objects to be clustered and whose rows represent the anonymous features used to perform the clustering.
`labels`	A factor (or character vector) of class labels for the objects in the `data` matrix.
`linkage`	A linkage rule accepted by the `hclust` function.
`fix`	A vector of cluater assignments.
`vary`	A vector of cluater assignments.

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 $1, 2, \ldots, K$ – 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".

That's where this set of functions comes in. The core algorithm is implemented in the function `labelMatcher`, which works on a contingency table whose entries N_{ij} are the number of samples with row-label = i and column-label = j. To find the best match, one computes (heuristically) the values F_{ij} that describe the fraction of all entries in row i and column j represented by N_{ij}. Perfectly matched labels would consist of a row i and a column j where N_{ij} is the only nonzero entry in its row and column, so $F_{ij} = 1$. The largest value for F_{ij} (with ties broken simply by which entry is closer to the upper-left corner of the matrix) defines the best match. The matched row and column are then removed from the matrix and the process repeats recursively.

We apply this method to determine which distance metric, when used in hierarchical clustering, best matches a "gold standard" set of class labels. (These may not really be gold, of course; they can also be a set of labels determined by k-means or another clustering algorithm.) The idea is to cluster the samples using a variety of different metrics, and select the one whose label assignments best macth the standard.

Value

The `labelMatcher` function returns a list of two vectors of the same length. These contain the matched label-indices, in the order they were matched by the algorithm.

The `matchLabels` function is a user-friendly front-end to the `labelmatcher` function. It returns a matrix, with the rows and columns reordered so the labels match.

The `countAgreement` function returns an integer, the number of samples with the "same" labels, computed by summing the diagonal of the reordered matrix produced by `matchLabels`.

The `labelAccuracy` function returns a vector indexed by the set of nine distance metrics hard-coded in the function. Each entry is the fraction of samples whose hierarchical clusters match the prespecified `labels`.

The `bestMetric` function is a user-friendly front-end to the `labelAccuracy` function. It returns the name of the distance metric whose hierarchical clusters best match the prespecified `labels`.

The `remap` function takes two sets of integer cluster assignments and returns a new set of labels for the target that best matches the source.

Note

The `labelAccuracy` function should probably allow the user to supply a list of distance metrics instead of relying on the hard-coded list internally.

Author(s)

Kevin R. Coombes <krc@silicovore.com>

See Also

Hierarchical clustering is implemented in the `hclust` function. We use the extended set of distance metrics provided by the `distanceMatrix` function from the ClassDiscovery package. This set includes all of the metrics from the `dist` funciton.

Examples

```
factor1 <- sample(c("A", "B", "C"), 30, replace=TRUE)
factor2 <- rep(c("X", "Y", "Z"), each=10)
tab <- table(factor1, factor2)
matchLabels(tab)
labelMatcher(tab)
R <- remap(factor1, factor2)
table(R, factor2) # remapping
table(R, factor1) # cross-comparison
```

`Reaper-class` *Class* `"Reaper"`

Description

The `Reaper` class implements the second step in the algorithm to combine outlier detection with cliustering. The first step, implemented in the Thresher-class, performs principal components analysis an computes the PC dimension. Features with short loading vectors are identified as outliers. Remaining features are clustering, based on the directions of the loading vectors, using mixtures of von Mises-Fisher distributions.

Usage

```
Reaper(thresher, useLoadings = FALSE, cutoff = 0.3,
       metric = NULL, linkage="ward.D2",
       maxSampleGroups = 0, ...)
```

Arguments

`thresher`	A `Thresher` object.
`useLoadings`	A logical value; should model-based clustering using von Mises-Fisher distributions be performed in the principal component space?
`cutoff`	A real number; what length loading vector should be used to separate outliers from significant contributers.
`metric`	A character string containing the name of a clustering metric recognized by either `dist` or `distanceMatrix`.
`linkage`	A character string containing the name of a linkage rule recognized by `hclust`.
`maxSampleGroups`	An integer; the maximum number of sample groups to be indicated by color in plots of the object.
`...`	Additional arguments to be passed to the `Thresher` function.

Details

Using the dimension computed when constructing the `Thresher` object, we computed the lengths of the loading vectors associated to features in the data set. Features whose length is less than a specified `cutoff` are identified as outliers and removed. (Based on extensive simulations, the default cutoff is taken to be 0.3.) We then refit the Thresher model on the remaining features, which should, in theory, leave the PC dimension, `D`, unchanged. We then rescale the remaining loading vectors to unit length, so they can be viewed as points on a hypersphere. In order to cluster points on a hypersphere, we use a model based on a mixture of von Mises-Fisher distributions. We fit mixtures for every integer in the range $D \leq N \leq 2D + 1$; this range accounts for the possibility that each axis has both positively and negatively correlated features. The extra $+1$ handles the degenerate case when $D = 0$. The best fit is determined using the Bayes Information Criterion (BIC). The final step is to compute a `SignalSet`; see the description of that class for more details.

Value

The `Reaper` function returns an object of the Reaper class.

Objects from the Class

Objects should be defined using the `Reaper` constructor. In the simplest case, you simply pass in a previously computed `Thresher` object.

Slots

`useLoadings`**:** Logical; should model-based clustering be performed in PC space?

`keep`**:** Logical vector: which of the features (columns) should be retained as meaningful signal instead of being removed as outliers?

`nGroups`**:** Object of class `"number or miss"`; the optimal number of groups/clusters found by the algorithm. If all of the fits fail, this is NA.

`fit`**:** Object of class `"fit or miss"`; the best mixture model fit. Can be an NA if something goes wrong when trying to fit mixture models.

`allfits`**:** Object of class `"list"`; a list, each of whose entries should be the results of fitting a mixture model with a different number of components.

`bic`**:** Object of class `"number or miss"`; the optimal valus of the Bayes Information Criterion; can be NA if all attempts to fit models fail.

`metric`**:** A character string; the preferred distance metric for hierarchical clustering. If not specified by the user, then this is computed using the `bestMetric` function.

`signalSet`**:** Object of class `SignalSet`

`maxSampleGroups`**:** An integer; the maximum number of sample groups to be distinguished by color in plots of the object.

Extends

Class `"Thresher"`, directly.

Methods

- **makeFigures** `signature(object = "Reaper")`: This is a convenience function to produce a standard set of figures. In addition tot he plots preodcued for`Thresher` object, this function also produces heatmaps where sample clustering depends on either the continuous or binary signal sets. If the `DIR` argument is non-null, it is treated as the name of an existing directory where the figures are stored as PNG files. Otherwise, the figures are displayed interactively, one at a time, in a window on screen.
- **getColors** `signature(object = "Reaper")`: Returns the vector of colors assigned to the clustered columns in the data set.
- **getSplit** `signature(object = "Reaper")`: Returns the vector of colors assigned to the clustered rows in the data set.

Author(s)

Kevin R. Coombes <krc@silicovore.com>, Min Wang.

References

Wang M, Abrams ZB, Kornblau SM, Coombes KR. Thresher: determining the number of clusters while removing outliers. BMC Bioinformatics, 2018; 19(1):1-9. doi://10.1186/s12859-017-1998-9.

Wang M, Kornblau SM, Coombes KR. Decomposing the Apoptosis Pathway Into Biologically Interpretable Principal Components. bioRxiv, 2017. doi://10.1101/237883.

Banerjee A, Dhillon IS, Ghosh J, Sra S. Clustering on the unit hypersphere using von Mises-Fisher distributions. Journal of Machine Learning Research, 2005; 6:1345–1382.

Kurt Hornik and Bettina Gr\"un. movMF: An R Package for Fitting Mixtures of von Mises-Fisher Distributions. Journal of Statistical Software, 2014; 58(10):1–31.

See Also

`PCDimension`, `SignalSet`.

Examples

```
# Simulate  a data set with some structure
set.seed(250264)
sigma1 <- matrix(0, ncol=16, nrow=16)
sigma1[1:7, 1:7] <- 0.7
sigma1[8:14, 8:14] <- 0.3
diag(sigma1) <- 1
st <- SimThresher(sigma1, nSample=300)
# Threshing is completed; now we can reap
reap <- Reaper(st)
screeplot(reap, col='pink', lcol='red')
scatter(reap)
plot(reap)
heat(reap)
```

`SignalSet-class` *Class* `"SignalSet"`

Description

We use the term "(continuous) signal" to refer to a weighted sum (by default, the mean) of gene-features. By dichotomizing a continuous signals, we obtain a "binary signal". The `SignalSet` class represents the set of continuous and binary signals obtained after clustering the features in a data set.

Objects from the Class

Objects can be created by calls of the form `new("SignalSet",...)`. However, users are styrongly discouraged from contructing a `SignalSet` manually. They are only used in the code internal to the construction of a `Reaper` object.

Slots

- `members`**:** Object of class `"list"`. Each member of the list is a character vector enumerating the features defining each signal.
- `continuous`**:** A matrix where the number of columns equals the length of the `members` list; each column contains the mean expression of the (assumed standardized) corresponding features.
- `binary`**:** A matrix where the number of columns equals the length of the `members` list; each column contains expression values dichotmoized to 0 or 1 by splitting the conmtinuous siognal at zero.
- `continuousClusters`**:** Object of class `"hclust"` obtained by clustering samples based on the continuous signals.
- `binaryClusters`**:** Object of class `"hclust"` obtained by clustering samples based on the binary signals.

Methods

No methods defined with class "SignalSet" in the signature.

Note

The length of `members` and thus the number of signals may be smaller than expected from the number of clusters found by `Reaper`. The implementation of the `SignalSet` tries to determine if two signals are pointing in opposite directions, which could happen if they are postively and negatively correlated sets. This behavior is likely to change in the future.

Author(s)

Kevin R. Coombes <krc@silicovore.com>

See Also

`Reaper`

Examples

```
# Simulate  a data set with some structure
set.seed(250264)
sigma1 <- matrix(0, ncol=16, nrow=16)
sigma1[1:7, 1:7] <- 0.7
sigma1[8:14, 8:14] <- 0.3
diag(sigma1) <- 1
st <- SimThresher(sigma1, nSample=300)
# Threshing is completed; now we can reap
reap <- Reaper(st)
# now extract the signal set
ss <- reap@signalSet
dim(ss@continuous)
dim(ss@binary)
table(ss@binary[,1], ss@binary[,2])
plot(ss@continuousClusters)
```

SimThresher-class *Class* "SimThresher"

Description

The SimThresher class is used to simulate Thresher objects.

Usage

```
SimThresher(ss, nSample, nm = deparse(substitute(ss)), rho = NULL,
             agfun = agDimTwiceMean, ...)
```

Arguments

ss	A covariance matrix.
nSample	An integer; the number of samples to simulate.
nm	A character string; the name of this object.
rho	A numeric vector; the correlation between different variables. If NULL, then these are obtained from the covariance matrix.
agfun	A function used by the AuerGervini function to determine the number of principal components.
...	Parameters to be passed to the Thresher constructor.

Details

Basically, given a number of samples and a covariance matrix, simulate a data matrix of the appropriate size and multivariate normal structure by assuming that all of the means are zero. After simulating the data, we apply the Thresher algorithm. The result is an object that combines the simulation parameters, simulated data, and fitted model.

Value

The `SimThresher` function returns an object of the `SimThresher` class.

Objects from the Class

Objects should be created using the `SimThresher` constructor.

Slots

`nSample`**:** An integer; the number of simulated samples.

`covariance`**:** A covariance matrix.

`rho`**:** A vector of correlation coefficients; essentially the unique values in the upper triangular part of the covariance matrix.

Extends

Class `Thresher`, directly.

Methods

image `signature(x = "SimThresher")`: Produces an image of the covariance matrix.

makeFigures `signature(object = "SimThresher")`: This is a convenience function to produce a standard set of figures. In addition tot he plots preodcued for`Thresher` object, this function also produces an image of te covariance matrix used in the simulations. If the `DIR` argument is non-null, it is treated as the name of an existing directory where the figures are stored as PNG files. Otherwise, the figures are displayed interactively, one at a time, in a window on screen.

Author(s)

Kevin R. Coombes <krc@silicovore.com>, Min Wang.

References

Wang M, Abrams ZB, Kornblau SM, Coombes KR. Thresher: determining the number of clusters while removing outliers. BMC Bioinformatics, 2018; 19(1):1-9. doi://10.1186/s12859-017-1998-9.

See Also

`Thresher`, `Reaper`.

Examples

```
set.seed(250264)
rho <- 0.5
nProtein <- 16
splinter <- sample((nProtein/2) + (-3:3), 1)
sigma1 <- matrix(rho, ncol=nProtein, nrow=nProtein)
diag(sigma1) <- 1
st <- SimThresher(sigma1, nSample=300)
```

```
image(st, col=redgreen(64), zlim=c(-1,1))
screeplot(st, col='pink', lcol='red')
plot(st)
scatter(st)
heat(st)
```

`Thresher class unions` *Class Unions*

Description

The implemetation of the `Reaper` class makes use of two class unions in order to deal with cases when we are unable to fit a model.

The `number or miss` class union can either hold a numeric vector or a logical vector.

The `fit or miss` class union can hold either a fitted `movMF` object or a logical vector.

Objects from the Class

Virtual Class: No objects may be created directly from either of these class-unions.

Methods

No methods are defined with class "number or miss" or "fit or miss" in the signature.

Author(s)

Kevin R. Coombes <krc@silicovore.com>

`Thresher-class` *Class* `"Thresher"`

Description

The `Thresher` class represents the first step of an algorithm that combines outlier detection with clustering. The object combines the results of hierarchical clustering and principal components analysis (with a computation of its dimension) on the same data set.

Usage

```
Thresher(data, nm = deparse(substitute(data)),
         metric = "pearson", linkage="ward.D2",
         method = c("auer.gervini", "broken.stick"),
         scale = TRUE, agfun = agDimTwiceMean)
```

Arguments

`data`	A data matrix.
`nm`	A character string; the name of this object.
`metric`	A character string containing the name of a clustering metric recognized by either `dist` or `distanceMatrix`.
`linkage`	A character string containing the name of a linkage rule recognized by `hclust`.
`method`	A character string describing the algorthim used from the `PCDimension` package to compute the number of significant components.
`scale`	A logical value; should the data be scaled before use?
`agfun`	A function that will be accepted by the `AuerGervini` function in the `PCDimension` package.

Details

`Thresher` operates on a data matrix that is assumed to be organized with rows equal to samples and columns equal to features (like genes or proteins). The algorithm begins by centering and (by default, though this can be overridden with the `scale` parameter) standardizes the data columns. It then performs a principal components analysis, and uses the Auer-Gervini method, as automated in the `PCDimension` package, to determine the number, `D`, of statistically significant principal components. For each column-feature, it computes and remembers the length of its loading vector in D-dimensional space. (In case the Auer-Gervini method finds that $D = 0$, the length is instead computed using $D = 1$.) These loading-lengths will be used later to identify and remove features that act as outliers and do not contribute to clustering the samples. Finally, `Thresher` computes and saves the results of hierarchically clustering the features in the data set, using the specified distance `metric` and `linkage` rule.

Value

The `Thresher` function constructs and returns an object of the `Thresher` class.

Objects from the Class

Objects should be defined using the `Thresher` constructor. In the simplest case, you simply pass in the data matrix that you want to cluster using the Thresher algorithm.

Slots

`name`**:** Object of class `"character"`; the name of this object.

`data`**:** Object of class `"matrix"`; the data that was used for clustering.

`spca`**:** Object of class `"SamplePCA"`; represents the results of performing a principal components analysis on the original `data`.

`loadings`**:** Object of class `"matrix"`; the matrix of loading vectors from the principal components analysis.

`gc`**:** Object of class `"hclust"`; the result of performing hierarchical clustering on the data columns.

`pcdim`**:** Object of class `"numeric"`; the number of significant principal components.

`delta`**:** Object of class `"numeric"`; the lengths of the loading vectors in the principal component space of dimension equal to `pcdim`.

`ag`**:** Object of class `"AuerGervini"`; represents the result of running the automated Auer-Gervini algorithm to detemine the number of principal components.

`agfun`**:** A function, which is used as the default method for computing the principal component dimension from the Auer-Gervini plot.

Methods

screeplot `signature(x = "Thresher")`: Produce a scree plot of the PCA part of the Thresher object.

scatter `signature(object = "Thresher")`: Produce a scatter plot of the first two principal components.

plot `signature(x = "Thresher",y = "missing")`: In two dimensions, plot the loading vectors of the PCA part of the object.

heat `signature(object = "Thresher")`: Produce a heatmap of the data set.

makeFigures `signature(object = "Thresher")`: This is a convenience function to produce a standard set of figures for a `Thresher` object. These are (1) a scree plot, (2) a plot of teh Auer-Gervini slot, (3) a scatter plot of the firtst trwo principal components, (4) one or more plots of the loading vectors, depending on the PCV dimension, and (5) a heat map. If the `DIR` argument is non-null, it is treated as the name of an existing directory where the figures are stored as PNG files. Otherwise, the figures are displayed interactively, one at a time, in a window on screen.

getColors `signature(object = "Thresher")`: Returns the vector of colors assigned to the clustered columns in the data set.

getSplit `signature(object = "Thresher")`: Returns the vector of colors assigned to the clustered rows in the data set.

getStyles `signature(object = "Thresher")`: I refuse to document this, since I am not convinced that it should actually exist.

Author(s)

Kevin R. Coombes <krc@silicovore.com>, Min Wang.

References

Wang M, Abrams ZB, Kornblau SM, Coombes KR. Thresher: determining the number of clusters while removing outliers. BMC Bioinformatics, 2018; 19(1):1-9. doi://10.1186/s12859-017-1998-9.

Wang M, Kornblau SM, Coombes KR. Decomposing the Apoptosis Pathway Into Biologically Interpretable Principal Components. bioRxiv, 2017. doi://10.1101/237883.

See Also

`Thresher`, `Reaper-class`, `AuerGervini-class`

Examples

```
set.seed(3928270)
ranData <- matrix(rnorm(100*12), ncol=12)
colnames(ranData) <- paste("G", 1:12, sep='')
thresh <- Thresher(ranData) # fit the model
screeplot(thresh)           # check the scree plot; suggests dim = 4
plot(thresh@ag, list(thresh@agfun)) # Auer-Gervini object; dim = 0
scatter(thresh)             # PCA scatter plot  (rows = samples)
plot(thresh)                # PCA loadings plot (cols = features)
heat(thresh)                # ubiquitous 2-way heatmap
```

`Thresher-data` *Thresher and Reaper Simulated Data*

Description

This data set contains five related simulated data sets, along with the Thresher and Reaper objects obtained by clustering the data sets and removing outliers.

Usage

```
data(savedSims)
```

Format

1. `sigma`: A list of length five; each entry is a covariance matrix used to simulate data.
2. `savedSims`: A list of length five; each entry is a `SimThresher` object obtained by simulating data from one of the covariance matrices and running the `Thresher` algorithm.
3. `savedReap`: A list of length five; each entry is a `Reaper` object obtained by applying the `Reaper` function.

Source

The simulated data sets were prepared by running the script `Examples/makeSims.R` that is installed along with the `Thresher` package. The five covariance matrices vary in the number of correlated subgroups (one or two) and whether they include both positively and negatively correlated variables, or just positively correlated ones. Each data set also includes two uncorrelated "noise" variables that should automatically be removed by the Reper-Thresher algorithms.

References

Wang M, Abrams ZB, Kornblau SM, Coombes KR. Thresher: determining the number of clusters while removing outliers. BMC Bioinformatics, 2018; 19(1):1-9. doi://10.1186/s12859-017-1998-9.

`unitize` *Convert a Vector to Unit Length*

Description

Rescales each column of a matrix to produce vectors of length one.

Usage

```
unitize(mat)
```

Arguments

`mat`	A matrix of real numbers.

Details

No details beyond the simple description are requires; it is implemented exactly the way you would suspect.

Value

A matrix of the same size as the input matrix.

Author(s)

Kevin R. Coombes <krc@silicovore.com>

Examples

```
simmat <- matrix(rnorm(12), 3, 4)
U <- unitize(simmat)
apply(U^2, 2, sum)  # check unit length
simmat/U            # view normalization factors
```

Index