Package ‘Thresher’

July 6, 2017

Version 0.11.0

Date 2016-05-16

Title Threshing and Reaping for Principal Components

Author Kevin R. Coombes

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

Description The Thresher package 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.

Depends R (>= 3.1), ClassDiscovery, PCDimension

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

License Apache License (== 2.0)

biocViews Clustering

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

NeedsCompilation no

R topics documented:

`Thresher-package` *What the package does (short line) Threshing and Reaping for Principal Components*

Description

The Thresher package 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.

Details

Package:	Thresher
Type:	Package
Version:	1.0
Date:	2014-11-20
License:	What license is it under?
Depends:	methods

Standard usage is to apply the `Thresher` function to a data set in order to estimate the principal component dimension and identifty outliers. You then apply the `Reaper` function to actually remove the outliers and asssign the remaining objects to clusters.

Author(s)

Kevin R. Coombes and Min Wang

Maintainer: <krc@silicovore.com>

References

Submitted to some jounral.

See Also

`PCDimension`

`fit or miss-class` *Class* `"fit or miss"`

Objects from the Class

A virtual Class: No objects may be created from it.

Methods

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

Examples

```
showClass("fit or miss")
```

`getColors-methods` ~~ *Methods for Function* `getColors` ~~

Description

~~ Methods for function `getColors` ~~

Methods

`signature(object = "Reaper")`

`signature(object = "Thresher")`

`getSplit-methods` ~~ *Methods for Function* `getSplit` ~~

Description

~~ Methods for function `getSplit` ~~

Methods

`signature(object = "Reaper")`

`signature(object = "Thresher")`

`heat-methods` ~~ *Methods for Function* `heat` ~~

Description

~~ Methods for function `heat` ~~

Methods

`signature(object = "Thresher")`

image-methods *~~ Methods for Function* image *~~*

Description

~~ Methods for function image ~~

Methods

`signature(x = "SimThresher")`

makeFigures-methods *~~ Methods for Function* makeFigures *~~*

Description

~~ Methods for function makeFigures ~~

Methods

`signature(object = "Reaper")`

`signature(object = "SimThresher")`

`signature(object = "Thresher")`

matchLabels *Match Arbitrary Class Assignments Across Methods*

Usage

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

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.

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 mattrix) 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`.

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 clusteriung 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)
```

`movMF-class` *Class* `"movMF"`

Objects from the Class

A virtual Class: No objects may be created from it.

Slots

`.S3Class`**:** Object of class `"character"` ~~

Extends

Class `"oldClass"`, directly. Class `"fit or miss"`, directly.

Methods

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

Examples

```
showClass("movMF")
```

`number or miss-class` *Class* `"number or miss"`

Objects from the Class

A virtual Class: No objects may be created from it.

Methods

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

Examples

```
showClass("number or miss")
```

plot-methods *~~ Methods for Function* plot ~~

Description

~~ Methods for function plot ~~

Methods

signature(x = "AuerGervini", y = "missing")

signature(x = "Thresher", y = "missing")

Reaper *Reaper*

Usage

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

Arguments

thresher

useLoadings

cutoff

metric

linkage

maxSampleGroups

verbose

...

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (thresher, useLoadings = FALSE, cutoff = 0.3, metric = NULL,
    maxSampleGroups = 0, verbose = TRUE, ...)
{
    if (verbose)
        cat(thresher@name, "\n", file = stderr())
    keep <- thresher@delta > cutoff
    m <- ifelse(is.null(metric), "pearson", metric)
    cleaned <- Thresher(thresher@data[, keep], paste(thresher@name,
        "cleaned", sep = "."), metric = m, ...)
    tab <- 0
    counter <- 0
    while (any(tab == 0) & counter < 5) {
        counter <- counter + 1
        fits <- .fitModels(cleaned, useLoadings)
        if (length(fits) == 0)
            next
        bic <- sapply(fits, BIC)
        woo <- which(bic == min(bic))
        ng <- as.integer(sub("NC=", "", names(woo)))
        fit <- fits[[woo]]
        gassign <- factor(predict(fit), levels = 1:ng)
        tab <- table(gassign)
    }
    if (length(fits) == 0) {
        bic <- ng <- fit <- NA
        metric <- "no fit"
        sigset <- new("SignalSet")
    }
    else {
        if (is.null(metric)) {
            pp <- factor(paste("G", predict(fit), sep = ""))
            metric <- bestMetric(cleaned@data, pp)
            cleaned@gc <- hclust(distanceMatrix(cleaned@data,
                metric, p = 1), "ward")
        }
        if (any(tab == 0)) {
            sigset <- new("SignalSet")
        }
        else {
            sigset <- .findSignals(cleaned, fit, ng)
        }
    }
    new("Reaper", cleaned, useLoadings = useLoadings, keep = keep,
        nGroups = ng, fit = fit, allfits = fits, bic = bic, metric = metric,
        signalSet = sigset, maxSampleGroups = maxSampleGroups)
  }
```

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

Objects from the Class

Objects can be created by calls of the form `new("Reaper", ...)`.

Slots

`useLoadings`: Object of class `"logical"` ~~

`keep`: Object of class `"logical"` ~~

`nGroups`: Object of class `"number or miss"` ~~

`fit`: Object of class `"fit or miss"` ~~

`allfits`: Object of class `"list"` ~~

`bic`: Object of class `"number or miss"` ~~

`metric`: Object of class `"character"` ~~

`signalSet`: Object of class `"SignalSet"` ~~

`maxSampleGroups`: Object of class `"numeric"` ~~

`name`: Object of class `"character"` ~~

`data`: Object of class `"matrix"` ~~

`spca`: Object of class `"SamplePCA"` ~~

`loadings`: Object of class `"matrix"` ~~

`gc`: Object of class `"hclust"` ~~

`pcdim`: Object of class `"numeric"` ~~

`delta`: Object of class `"numeric"` ~~

`ag`: Object of class `"AuerGervini"` ~~

Extends

Class `"Thresher"`, directly.

Methods

getColors `signature(object = "Reaper")`: ...

getSplit `signature(object = "Reaper")`: ...

makeFigures `signature(object = "Reaper")`: ...

Examples

```
showClass("Reaper")
```

`samplePalette` *Color Palettes Used By Thresher Classes*

Usage

```
data(samplePalette)
data(thresherPalette)
```

Format

The format is: chr [1:20] "gray" "blue" "red" "purple" "green" "cyan" ... The format is: chr [1:19] "#FF0000FF" "#0000FFFF" "#00BA38" "#AA00FFFF" ...

Examples

```
data(samplePalette)
data(thresherpalette)
```

`scatter-methods` *~~ Methods for Function* `scatter` *~~*

Description

~~ Methods for function `scatter` ~~

Methods

`signature(object = "Thresher")`

`screeplot-methods` *~~ Methods for Function* `screeplot` *~~*

Description

~~ Methods for function `screeplot` ~~

Methods

`signature(x = "Thresher")`

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

Objects from the Class

Objects can be created by calls of the form `new("SignalSet", ...)`.

Slots

`members`: Object of class `"list"` ~~

`continuous`: Object of class `"matrix"` ~~

`binary`: Object of class `"matrix"` ~~

`continuousClusters`: Object of class `"hclust"` ~~

`binaryClusters`: Object of class `"hclust"` ~~

Methods

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

Examples

```
showClass("SignalSet")
```

`SimThresher` *simt*

Usage

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

Arguments

`ss`

`nSample`

`nm`

`rho`

`...`

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (ss, nSample, nm = deparse(substitute(ss)), rho = NULL,
    ...)
{
    if (is.null(rho)) {
        rho <- sort(unique(abs(ss[upper.tri(ss)])))[-1]
    }
    require(MASS)
    nFeature <- ncol(ss)
    mu <- rep(0, nFeature)
    simdata <- mvrnorm(nSample, mu, ss)
    colnames(simdata) <- paste("Pr", 1:ncol(simdata), sep = "")
    new("SimThresher", Thresher(simdata, nm, ...), nSample = nSample,
        covariance = ss, rho = rho)
  }
```

`SimThresher-class` *Class* `"SimThresher"`

Objects from the Class

Objects can be created by calls of the form `new("SimThresher", ...)`.

Slots

`nSample`: Object of class "`numeric`" ~~

`covariance`: Object of class "`matrix`" ~~

`rho`: Object of class "`numeric`" ~~

`name`: Object of class "`character`" ~~

`data`: Object of class "`matrix`" ~~

`spca`: Object of class "`SamplePCA`" ~~

`loadings`: Object of class "`matrix`" ~~

`gc`: Object of class "`hclust`" ~~

`pcdim`: Object of class "`numeric`" ~~

`delta`: Object of class "`numeric`" ~~

`ag`: Object of class "`AuerGervini`" ~~

Extends

Class "`Thresher`", directly.

Methods

image `signature(x = "SimThresher")`: ...

makeFigures `signature(object = "SimThresher")`: ...

Examples

```
showClass("SimThresher")
```

---

summary-methods ~~ *Methods for Function* `summary` ~~

Description

~~ Methods for function `summary` ~~

Methods

`signature(object = "AuerGervini")`

`Thresher` *Thresher*

Usage

```
Thresher(data, nm = deparse(substitute(data)), FUZZ = 0.005,
          metric = "pearson", linkage="ward.D2",
          method = c("broken.stick", "auer.gervini"))
```

Arguments

`data`

`nm`

`FUZZ`

`metric`

`linkage`

`method`

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (data, nm = deparse(substitute(data)), FUZZ = 0.005,
    metric = "pearson", method = c("broken.stick", "auer.gervini"))
{
    std <- scale(data)
    spca <- SamplePCA(t(std))
    ag <- AuerGervini(spca)
    method <- match.arg(method)
    pcdim <- switch(method, broken.stick = bsDimension(spca),
        auer.gervini = agDimension(ag))
    deltaDim <- max(1, pcdim)
    lambda <- sqrt(spca@variances)
    loadings <- sweep(spca@components, 2, lambda, "*")
    delta <- sqrt(apply(loadings[, 1:deltaDim, drop = FALSE]^2,
        1, sum))
    gc <- hclust(distanceMatrix(std, metric), "ward")
    new("Thresher", name = nm, data = data, spca = spca, loadings = loadings,
        gc = gc, pcdim = pcdim, delta = delta, ag = ag)
  }
```

Thresher-class *Class* "Thresher"

Objects from the Class

Objects can be created by calls of the form `new("Thresher", ...)`.

Slots

`name`: Object of class `"character"` ~~

`data`: Object of class `"matrix"` ~~

`spca`: Object of class `"SamplePCA"` ~~

`loadings`: Object of class `"matrix"` ~~

`gc`: Object of class `"hclust"` ~~

`pcdim`: Object of class `"numeric"` ~~

`delta`: Object of class `"numeric"` ~~

`ag`: Object of class `"AuerGervini"` ~~

Methods

getColors `signature(object = "Thresher")`: ...

getSplit `signature(object = "Thresher")`: ...

heat `signature(object = "Thresher")`: ...

makeFigures `signature(object = "Thresher")`: ...

plot `signature(x = "Thresher", y = "missing")`: ...

scatter `signature(object = "Thresher")`: ...

screeplot `signature(x = "Thresher")`: ...

Examples

```
showClass("Thresher")
```

unitize *Convert a Vector to Unit Length*

Usage

```
unitize(mat)
```

Arguments

`mat`

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (mat)
{
    enorm <- sqrt(apply(mat^2, 2, sum))
    sweep(mat, 2, enorm, "/")
  }
```

Index