Package ‘hilbertSimilarity’

October 13, 2022

Type Package

Title Hilbert Similarity Index for High Dimensional Data

Version 0.4.3

Date 2019-11-11

Description Quantifying similarity between high-dimensional single cell samples is challenging, and usually requires some simplifying hypothesis to be made. By transforming the high dimensional space into a high dimensional grid, the number of cells in each sub-space of the grid is characteristic of a given sample. Using a Hilbert curve each sample can be visualized as a simple density plot, and the distance between samples can be calculated from the distribution of cells using the Jensen-Shannon distance. Bins that correspond to significant differences between samples can identified using a simple bootstrap procedure.

LinkingTo Rcpp

Imports Rcpp, entropy

Suggests knitr, rmarkdown, ggplot2, dplyr, tidyr, reshape2, bodenmiller, abind

License CC BY-NC-SA 4.0

LazyData TRUE

Encoding UTF-8

URL `http://github.com/yannabraham/hilbertSimilarity`

BugReports `http://github.com/yannabraham/hilbertSimilarity/issues`

VignetteBuilder knitr

RoxygenNote 6.1.1

NeedsCompilation yes

Author Yann Abraham [aut, cre],
Marilisa Neri [aut],
John Skilling [ctb]

Maintainer Yann Abraham <`yann.abraham@gmail.com`>

Repository CRAN

Date/Publication 2019-11-11 23:50:02 UTC

R topics documented:

`add.cut` *Add New Cut Thresholds*

Description

Add new manual cuts to the `cuts` matrix generated using `make.cut`

Usage

```
add.cut(cuts, new.cuts, cut.id = "manual", update = FALSE)
```

Arguments

`cuts`	a list of cuts generated using `make.cut`
`new.cuts`	a list of new cut thresholds to be added to `cuts`
`cut.id`	string identifying the new cuts
`update`	if FALSE (the default) adding a `cut.id` that already exists in `cuts` will return an error

Details

The matrix can be cut using either the fixed cuts (`type='fixed'`), or the combined cuts (`type='combined'`) where the limits have been adjusted to match local minima and maxima.

Value

an updated `cuts` matrix with an extra set of thresholds named `cut.id`.

Author(s)

Yann Abraham

Examples

```
# generate a random 3D matrix with 2 peaks
mat <- rbind(matrix(rnorm(300),ncol=3),
             matrix(rnorm(300,5,1),ncol=3))
dimnames(mat)[[2]] <- LETTERS[1:3]
# estimate the Hilbert order
hilbert.order(mat)
# generate 2 bins with a minimum bin size of 5
cuts <- make.cut(mat,n=3,count.lim=5)
show.cut(cuts)
# Generate the cuts
cut.mat <- do.cut(mat,cuts,type='fixed')
head(cut.mat)
```

`andrewsProjection` *Use Andrews plots to visualize the Hilbert curve*

Description

Use a Fourier series to project the Hilbert curve, based on the number of points per Hilbert index. See Wikipedia - Andrews plot for a description of the method.

Usage

```
andrewsProjection(x, breaks = 30)
```

Arguments

`x`	a matrix of counts, where rows correspond to samples and columns to Hilbert index
`breaks`	the number of points used to display the Andrews curve

Details

The Andrews curve corresponds to a projection of each item to $(1/2^{0}.5, sin(t), cos(t), sin(2t), cos(2t), ...)$ where *t* (the Andrews index) varies between $-\pi$ and π.

Value

a list with 2 items:

- freq : a matrix with `breaks` rows and `ncol(x)` columns containing the Andrews vector for projection
- i : a vector with `breaks` elements corresponding to the Andrews indices

Author(s)

Yann Abraham

Examples

```
# generate a random matrix
ncols <- 5
mat <- matrix(rnorm(ncols*1000),ncol=ncols)
dimnames(mat)[[2]] <- LETTERS[seq(ncols)]

# generate categories
conditions <- sample(letters[1:3],nrow(mat),replace = TRUE)
# generate 4 bins with a minimum bin size of 5
horder <- 4
cuts <- make.cut(mat,n=horder+1,count.lim=5)
# Generate the cuts and compute the Hilbert index
cut.mat <- do.cut(mat,cuts,type='fixed')
hc <- do.hilbert(cut.mat,horder)
# compute hilbert index per condition
condition.mat <- table(conditions,hc)
condition.pc <- apply(condition.mat,1,function(x) x/sum(x))
condition.pc <- t(condition.pc)
# project the matrix to the Andrews curve
av <- andrewsProjection(condition.pc)
proj <- condition.pc %*% t(av$freq)

plot(range(av$i),
     range(proj),
     type='n',
     xlab='Andrews index',
     ylab='Projection')
for(i in seq(nrow(proj))) {
    lines(av$i,
          proj[i,],
          col=i)
}
legend('bottomleft',
       legend=letters[1:3],
       col=seq(1,3),
       pch=16,
       bty='n')
```

`do.cut` *Apply Cuts to the Reference Matrix*

Description

Apply cuts generated using the `make.cut` function to the reference matrix

Usage

```
do.cut(mat, cuts, type = "combined")
```

Arguments

`mat`	the matrix to cut
`cuts`	a list of cuts generated using `make.cut`
`type`	the type of cuts to use (use `combined` by default)

Details

The matrix can be cut using either the fixed cuts (`type='fixed'`), or the combined cuts (`type='combined'`) where the limits have been adjusted to match local minima and maxima. Returned values correspond to the bin defined between the first and second threshold of the specified `cuts`, then between the second and third threshold, and so on. The values will range between 0 (the first bin) and `n-1` where `n` is the number of values in the specified `cuts`.

Value

a matrix of the same dimensionality as `mat` where values correspond to bins defined by the `type` thresholds defined `cuts`.

Author(s)

Yann Abraham

Examples

```
# generate a random 3D matrix with 2 peaks
mat <- rbind(matrix(rnorm(300),ncol=3),
             matrix(rnorm(300,5,1),ncol=3))
dimnames(mat)[[2]] <- LETTERS[1:3]
# estimate the Hilbert order
hilbert.order(mat)
# generate 2 bins with a minimum bin size of 5
cuts <- make.cut(mat,n=3,count.lim=5)
show.cut(cuts)
# Generate the cuts
cut.mat <- do.cut(mat,cuts,type='fixed')
head(cut.mat)
```

`do.hilbert` *Generate the Hilbert Index from a Cut Reference Matrix*

Description

Generate the Hilbert Index corresponding to the sub-spaces defined by the coordinates generated *via* `do.cut`

Usage

```
do.hilbert(mat, horder)
```

Arguments

`mat`	the cut reference matrix
`horder`	the Hilbert order, *i.e.* the number of bins in each dimension

Details

For each line in `mat`, the function will compute the corresponding Hilbert index. Each index corresponds to a specific sub-cube of the original high-dimensional space, and consecutive hilbert index correspond to adjacent sub-cubes

Value

a vector of indices, one for each line in `mat`

Author(s)

Marilisa Neri

Yann Abraham

John Skilling (for the original `C` function)

Examples

```
# generate a random 3D matrix
mat <- matrix(rnorm(300),ncol=3)
dimnames(mat)[[2]] <- LETTERS[1:3]
# generate 2 bins with a minimum bin size of 5
cuts <- make.cut(mat,n=3,count.lim=5)
show.cut(cuts)
# Generate the cuts
cut.mat <- do.cut(mat,cuts,type='fixed')
head(cut.mat)
# generate the Hilber index
hc <- do.hilbert(cut.mat,2)
plot(table(hc),type='l')
```

`hilbert.order` *Estimate the Hilbert order for a given matrix*

Description

Estimate the Hilbert order, or the number of bins in each dimension, so that if the matrix was random every row in the matrix would correspond to a single bin.

Usage

```
hilbert.order(mat)
```

Arguments

`mat`	the matrix for which to estimate the Hilbert order

Details

Assuming the matrix is fully random, there is no need to generate more voxels (the combination of bins over all dimensions) than there are rows in the matrix. The number can be derived from the following formula:

$$c^d < N$$

where c is the number of bins, d is the number of dimensions and N is the total number of cells in the dataset. c can be computed easily using the following formula:

$$c = \lfloor \sqrt[d]{N}$$

The number of cuts for `do.cut` is the number of bins plus 1.

Value

the suggested number of bins to use for the specified `mat`.

Author(s)

Yann Abraham

Examples

```
# generate a random 3D matrix with 2 peaks
mat <- rbind(matrix(rnorm(300),ncol=3),
             matrix(rnorm(300,5,1),ncol=3))
dimnames(mat)[[2]] <- LETTERS[1:3]
# estimate the Hilbert order
hilbert.order(mat)
# generate 2 bins with a minimum bin size of 5
cuts <- make.cut(mat,n=3,count.lim=5)
show.cut(cuts)
# Generate the cuts
```

```
cut.mat <- do.cut(mat,cuts,type='fixed')
head(cut.mat)
```

`hilbertMapping` *Map High Dimensional Coordinates to Hilbert Index and back*

Description

`hilbertMapping` will compute the Hilbert index for each row of a matrix of integer coordinates corresponding to sub-cubes in a high dimensional space.

Arguments

`x`	a matrix of a matrix of integer coordinates (see `do.hilbert`)
`bits`	the hilbert order, *i.e.* the number of cuts in each dimension

Details

Functions: TransposetoAxes AxestoTranspose Purpose: Transform in-place between Hilbert transpose and geometrical axes Example: b=5 bits for each of n=3 coordinates. 15-bit Hilbert integer = A B C D E F G H I J K L M N O is stored as its Transpose X[0] = A D G J M X[2]| X[1] = B E H K N <——-> | /X[1] X[2] = C F I L O axes |/ high low 0—— X[0] Axes are stored conventionally as b-bit integers. Author: John Skilling 20 Apr 2001 to 11 Oct 2003

The source code includes the correction suggested in the following StackOverflow discussion.

Value

a vector of hilbert index, one for each line in `x`

Author(s)

Marilisa Neri

Yann Abraham

John Skilling

`hilbertProjection` *Project a Cut Reference Matrix to a Different Space through an Hilbert Index*

Description

Starting from a Hilbert Index generated in a high dimensional space, returns a set of coordinates in a new (lower) dimensional space

Usage

```
hilbertProjection(hc, target = 2)
```

Arguments

`hc`	the hilbert index returned by `do.hilbert`
`target`	the number of dimensions in the target space (defaults to 2)

Details

Based on the maximum index and the targeted number of dimensions the number of target bins is computed and used to generate a reference matrix and a reference index. The reference matrix is returned, ordered by the reference index.

Value

a matrix with `target` columns, corresponding to the projection of each Hilbert index to `target` dimensions

Author(s)

Marilisa Neri

Yann Abraham

John Skilling (for the original `C` function)

Examples

```
# generate a random matrix
ncols <- 5
mat <- matrix(rnorm(ncols*5000),ncol=ncols)
dimnames(mat)[[2]] <- LETTERS[seq(ncols)]

# generate 4 bins with a minimum bin size of 5
horder <- 4
cuts <- make.cut(mat,n=horder+1,count.lim=5)

# Generate the cuts and compute the Hilbert index
cut.mat <- do.cut(mat,cuts,type='fixed')
```

```
hc <- do.hilbert(cut.mat,horder)
chc <- table(hc)
idx <- as.numeric(names(chc))

# project the matrix to 2 dimensions
proj <- hilbertProjection(hc)

# visualize the result
img <- matrix(0,ncol=max(proj[,2])+1,nrow = max(proj[,1])+1)
img[proj[idx,]+1] <- chc
image(img)
```

`hilbertSimilarity` *Hilbert Similarity Index for High Dimensional Data*

Description

This package provides a method to compute similarity between single cell samples in high dimensional space. After dividing the space into voxels, each sample is summarized as a number of cells per voxel. Voxels are ordered using a Hilbert curve, so that each sample can be represented as a 1-dimensional density plot. the distance between 2 samples corresponds to the Jensen Shannon distance between the 2 probability vectors.

Examples

```
# generate 3 samples over 5 dimensions
# sample 1 and 2 are similar, sample 3 has an extra population
# set the seed for reproducible examples
set.seed(1234)
my.samples <- lapply(LETTERS[1:3],function(j) {
    # each sample has a different number of events
    n <- floor(runif(1,0.5,0.8)*10000)
    # matrix is random normal over 5 dimensions
    cur.mat <- matrix(rnorm(5*n),ncol=5)
    # rescale cur.mat to a [0,3] interval
    cur.mat <- 3*(cur.mat-min(cur.mat))/diff(range(cur.mat))
    dimnames(cur.mat)[[2]] <- LETTERS[(length(LETTERS)-4):length(LETTERS)]
    if(j=='C') {
      # select 30% of the points
      cur.rws <- sample(n,round(n*0.3,0))
      # select 2 columns at random
      cur.cls <- sample(ncol(cur.mat),2)
      # create an artificial sub population
      cur.mat[cur.rws,cur.cls] <- 4*cur.mat[cur.rws,cur.cls]
    }
    return(cur.mat)
  }
)
names(my.samples) <- LETTERS[1:3]
```

```
# check the population size
lapply(my.samples,nrow)

# assemble a sample matrix
my.samples.mat <- do.call('rbind',my.samples)
my.samples.id <- lapply(names(my.samples),
                        function(cur.spl) rep(cur.spl,nrow(my.samples[[cur.spl]])))
my.samples.id <- unlist(my.samples.id)

# Estimate the maximum required Hilbert order
hilbert.order(my.samples.mat)

# Estimate the cut positions
my.cuts <- make.cut(my.samples.mat,n=5,count.lim=5)

# Visualize the cuts
show.cut(my.cuts)

# Cut the matrix & compute the hilbert index
my.samples.cut <- do.cut(my.samples.mat,my.cuts,type='combined')
system.time(my.samples.index <- do.hilbert(my.samples.cut,horder=4))

# Visualize samples as density plots
my.samples.dens <- density(my.samples.index)
my.samples.dens$y <- (my.samples.dens$y-min(my.samples.dens$y))/diff(range(my.samples.dens$y))

plot(my.samples.dens,col='grey3',lty=2)
ksink <- lapply(names(my.samples),function(cur.spl) {
    cat(cur.spl,'\n')
    cur.dens <- density(my.samples.index[my.samples.id==cur.spl],
                        bw=my.samples.dens$bw)
    cur.dens$y <- (cur.dens$y-min(cur.dens$y))/diff(range(cur.dens$y))
    lines(cur.dens$x,
          cur.dens$y,
          col=match(cur.spl,names(my.samples))+1)
  }
)
legend('topright',
       legend=names(my.samples),
       co=seq(length(my.samples))+1,
       pch=16,
       bty='n' )

# assemble a contingency table
my.samples.table <- table(my.samples.index,my.samples.id)
dim(my.samples.table)

heatmap(log10(my.samples.table+0.00001),
        col=colorRampPalette(c('white',blues9))(24),
        Rowv=NA,Colv=NA,
        scale='none')

# compute the Jensen-Shannon distance
```

```
my.samples.dist <- js.dist(t(my.samples.table))
my.samples.clust <- hclust(my.samples.dist)

plot(my.samples.clust)
```

`js.dist` *Compute the Jensen-Shannon Distance between 2 sets of Hilbert Index*

Description

The Jensen-Shannon distance is a method to measure the distance between discrete probability distributions. To measure the distance between 2 high-dimensional datasets, we cut the space into sub-cubes, then count the number of events per cube. The resulting probability distributions can be compared using the Jensen-Shannon distance.

Usage

```
js.dist(mat, pc = 1e-04)
```

Arguments

`mat`	a matrix of counts, where rows correspond to samples and columns to Hilbert index
`pc`	a pseudo-count that is added to all samples to avoid divide-by-zero errors

Value

a S3 distance object

Author(s)

Yann Abraham

Examples

```
# generate 3 samples over 5 dimensions
# sample 1 and 2 are similar, sample 3 has an extra population
# set the seed for reproducible examples
set.seed(1234)
my.samples <- lapply(LETTERS[1:3],function(j) {
    # each sample has a different number of events
    n <- floor(runif(1,0.5,0.8)*10000)
    # matrix is random normal over 5 dimensions
    cur.mat <- matrix(rnorm(5*n),ncol=5)
    # rescale cur.mat to a [0,3] interval
    cur.mat <- 3*(cur.mat-min(cur.mat))/diff(range(cur.mat))
    dimnames(cur.mat)[[2]] <- LETTERS[(length(LETTERS)-4):length(LETTERS)]
    if(j=='C') {
      # select 30% of the points
```

```
      cur.rws <- sample(n,round(n*0.3,0))
      # select 2 columns at random
      cur.cls <- sample(ncol(cur.mat),2)
      # create an artificial sub population
      cur.mat[cur.rws,cur.cls] <- 4*cur.mat[cur.rws,cur.cls]
    }
    return(cur.mat)
  }
)
names(my.samples) <- LETTERS[1:3]

# check the population size
lapply(my.samples,nrow)

# assemble a sample matrix
my.samples.mat <- do.call('rbind',my.samples)
my.samples.id <- lapply(names(my.samples),
                        function(cur.spl) rep(cur.spl,nrow(my.samples[[cur.spl]])))
my.samples.id <- unlist(my.samples.id)

# Estimate the maximum required Hilbert order
hilbert.order(my.samples.mat)

# Estimate the cut positions
my.cuts <- make.cut(my.samples.mat,n=5,count.lim=5)

# Visualize the cuts
show.cut(my.cuts)

# Cut the matrix & compute the hilbert index
my.samples.cut <- do.cut(my.samples.mat,my.cuts,type='combined')
system.time(my.samples.index <- do.hilbert(my.samples.cut,horder=4))

# Visualize samples as density plots
my.samples.dens <- density(my.samples.index)
my.samples.dens$y <- (my.samples.dens$y-min(my.samples.dens$y))/diff(range(my.samples.dens$y))

plot(my.samples.dens,col='grey3',lty=2)
ksink <- lapply(names(my.samples),function(cur.spl) {
    cat(cur.spl,'\n')
    cur.dens <- density(my.samples.index[my.samples.id==cur.spl],
                        bw=my.samples.dens$bw)
    cur.dens$y <- (cur.dens$y-min(cur.dens$y))/diff(range(cur.dens$y))
    lines(cur.dens$x,
          cur.dens$y,
          col=match(cur.spl,names(my.samples))+1)
  }
)
legend('topright',
       legend=names(my.samples),
       co=seq(length(my.samples))+1,
       pch=16,
       bty='n' )
```

```
# assemble a contingency table
my.samples.table <- table(my.samples.index,my.samples.id)
dim(my.samples.table)

heatmap(log10(my.samples.table+0.00001),
        col=colorRampPalette(c('white',blues9))(24),
        Rowv=NA,Colv=NA,
        scale='none')

# compute the Jensen-Shannon distance
my.samples.dist <- js.dist(t(my.samples.table))
my.samples.clust <- hclust(my.samples.dist)

plot(my.samples.clust)
```

`localMaxima` *Find Local Maxima in a vector*

Description

Given a density object, find the position of local maxima (inflection points)

Usage

```
localMaxima(x)
```

Arguments

`x`	a vector of density values, as generated through a call to `density`

Value

a vector of index corresponding to local maxima

Author(s)

Tommy `http://stackoverflow.com/questions/6836409/finding-local-maxima-and-minima`

Examples

```
x <- c(rnorm(100),rnorm(100,3))
dx <- density(x)
plot(dx)
abline(v=dx$x[localMaxima(dx$y)],col=2,lty=2)
```

`localMinima` *Find Local Minima in a vector*

Description

Given a density object, find the position of local minima (inflection points)

Usage

```
localMinima(x)
```

Arguments

`x`	a vector of density values, as generated through a call to `density`

Value

a vector of index corresponding to local minima

Author(s)

Tommy `http://stackoverflow.com/questions/6836409/finding-local-maxima-and-minima`

Examples

```
x <- c(rnorm(100),rnorm(100,3))
dx <- density(x)
plot(dx)
abline(v=dx$x[localMinima(dx$y)],col=2,lty=2)
```

`make.cut` *Generate Cutting Points for a Multidimensional Matrix*

Description

For a given column `cur.ch` that belongs to a matrix, and a given number of cuts `n`, compute `n-1` bins using either fixed of combined limits

Usage

```
make.cut(mat, n = 5, count.lim = 40)
```

Arguments

`mat`	the matrix to cut
`n`	the number of cuts to generate (defaults to 5)
`count.lim`	the minimum number of counts to consider for density (defaults to 40)

Details

the fixed limits correspond to 5 equally spaced values over the range of the column. the combined limits take the local minima and maxima determined using the `localMinima` and `localMaxima` functions, to adjust the limits using the following algorithm:

- define `d` as half the distance between 2 fixed limits
- merge local minima and local maxima that are closer than `d`
- if any fixed limit is closer to a local minima than `d`, move the fixed limit to the local minima; move the limits that are not been moved yet, and that are before and after the moved limit so that they are evenly spread; repeat until no fixed limit can be moved
- if some limits have been moved to a local minima, **remove** limits that are closer than `d` to a local maxima; move the limits that are not been moved yet, and that are before and after the deleted limit so that they are evenly spread; repeat until no fixed limit can be moved
- if no limits has been moved to a local minima, move limits that are closer than `d` to a local maxima; move the limits that are not been moved yet, and that are before and after the moved limit so that they are evenly spread; repeat until no fixed limit can be moved

The function returns a list of lists, one for each column in `mat`, consisting of

- `cur.dens` the density used to describe the data
- `cur.hist` the histogram used to describe the data
- `fixed` the fixed, evenly spaced cuts
- `minima` the local minima detected in the data
- `maxima` the local maxima detected in the data
- `combined` the cuts defined using a combination of fixed positions, local minima and local maxima

Value

a list of of cuts for each column in `mat`, see *details*

Author(s)

Yann Abraham

Examples

```
# generate a random 3D matrix with 2 peaks
mat <- rbind(matrix(rnorm(300),ncol=3),
             matrix(rnorm(300,5,1),ncol=3))
dimnames(mat)[[2]] <- LETTERS[1:3]
# estimate the Hilbert order
hilbert.order(mat)
# generate 2 bins with a minimum bin size of 5
cuts <- make.cut(mat,n=3,count.lim=5)
show.cut(cuts)
# Generate the cuts
cut.mat <- do.cut(mat,cuts,type='fixed')
head(cut.mat)
```

`show.cut` *Plot the cuts generated through make.cut*

Description

Visualize the cuts in relation with the distribution of the data for each dimension in the original matrix

Usage

```
show.cut(cuts, type = "all", local = FALSE)
```

Arguments

`cuts`	the output of the `make.cut`.
`type`	which cuts to show. This must be one of "all", "fixed" or "combined". Any unambiguous substring can be given.
`local`	defaults to `FALSE`; if `TRUE`, shows the local minima and maxima as a rug plot.

Details

"fixed" will show `n` equally spaced cuts (see `make.cut` for the definition of `n`). "combined" will show the cuts after adjustment for local minima and maxima. "all" will show both. Setting `local` to `TRUE` will enable the visualization of local minima and maxima detected by the algorithm in each dimension.

Value

the function returns an invisible 'NULL'.

Author(s)

Yann Abraham

Examples

```
# generate a random 3D matrix with 2 peaks
mat <- rbind(matrix(rnorm(300),ncol=3),
             matrix(rnorm(300,5,1),ncol=3))
dimnames(mat)[[2]] <- LETTERS[1:3]
# estimate the Hilbert order
hilbert.order(mat)
# generate 2 bins with a minimum bin size of 5
cuts <- make.cut(mat,n=3,count.lim=5)
show.cut(cuts)
# Generate the cuts
cut.mat <- do.cut(mat,cuts,type='fixed')
head(cut.mat)
```

Index