Package ‘MutExMatSorting’

January 13, 2021

Type Package

Title Sort rows and columns of a binary matrix in a way that the patterns of non-null entries have a minimal overlap across rows

Version 0.1.0

Author Alessandro Vinceti

Maintainer Alessandro Vinceti <`alessandro.vinceti@fht.org`>

Description This package implements an heuristic algorithm that takes in input a sparse binary matrix and sorts its rows and columns in a way that the patterns of non-null entries have a minimal overlap across rows. This highlights possible mutual exclusive trends among these patterns.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

Depends pheatmap

Suggests knitr, rmarkdown

VignetteBuilder knitr

R topics documented:

`MExMaS.findBestInClass` *Find gene with the highest exclusive coverage*

Description

This function finds the gene (i.e. row) with the highest exclusive coverage. The exclusive coverage for a gene g is defined as the number of uncovered samples in which this gene is mutated minus the number of samples in which at least another uncovered gene is mutated.

Usage

```
MExMaS.findBestInClass(patterns)
```

Arguments

`patterns`	numeric binary matrix of the values to be sorted, after removing samples with no entries.

Examples

```
#Generating a random binary matrix with row and column names
r <- 100
c <- 100
dens<-0.10
mutPatterns <- matrix(0, r, c,dimnames = list(paste('row',1:r,sep=''),paste('col',1:c,sep='')))
mutPatterns[sample(r*c,round(r*c*dens))]<-1

#Removing samples with no entries
idNull<-which(colSums(mutPatterns)==0)
nullCol<-matrix(c(mutPatterns[,idNull]),nrow(mutPatterns),
          length(idNull),dimnames = list(rownames(mutPatterns),colnames(mutPatterns)[idNull]))

idNonNull<-which(colSums(mutPatterns)>0)
patterns<-matrix(c(mutPatterns[,idNonNull]),
                  nrow(mutPatterns),length(idNonNull),
                 dimnames=list(rownames(mutPatterns),colnames(mutPatterns)[idNonNull]))

#Find gene that maximise exclusive coverage
BS<-MExMaS.findBestInClass(patterns)
BS
```

`MExMaS.HeuristicMutExSorting`

Minimal overlap sorting

Description

This function implements an heuristic algroritm that takes in input a sparse binary matrix and sorts its rows and column in a way that the patterns of non null entries have a minimal overalp across rows.

Usage

```
MExMaS.HeuristicMutExSorting(mutPatterns)
```

Arguments

`mutPatterns`	numeric binary matrix of the values to be sorted.

Examples

```
library(pheatmap)

#Generating a random binary matrix with row and column names
r <- 100
c <- 100
dens<-0.10
mutPatterns <- matrix(0, r, c,dimnames = list(paste('row',1:r,sep=''),paste('col',1:c,sep='')))
```

```
mutPatterns[sample(r*c,round(r*c*dens))]<-1

#Executing mutual exclusivity sorting
sortedMat<-MExMaS.HeuristicMutExSorting(mutPatterns)

#visualising original matrix
pheatmap(mutPatterns,cluster_rows = FALSE,cluster_cols = FALSE,legend = FALSE,
 show_colnames = FALSE,show_rownames = FALSE,main='Original Matrix',col=c('white','blue'))
#visualising original matrix
pheatmap(sortedMat,cluster_rows = FALSE,cluster_cols = FALSE,legend = FALSE,
  show_colnames = FALSE,show_rownames = FALSE,main='Sorted Matrix',col=c('white','blue'))
```

`MExMaS.rearrangeMatrix` *Rearrange matrix columns to minimise row-wise entry overlap*

Description

This function rearranges the binary matrix columns in order to minimise row-wise entry overlap based on exclusive coverage.

Usage

```
MExMaS.rearrangeMatrix(patterns,GENES)
```

Arguments

`patterns`	numeric binary matrix of the values to be sorted, after removing samples with no entries.
`GENES`	character vector containing rownames ordered according to exclusive coverage.

Examples

```
#Generating a random binary matrix with row and column names
r <- 100
c <- 100
dens<-0.10
mutPatterns <- matrix(0, r, c,dimnames = list(paste('row',1:r,sep=''),paste('col',1:c,sep='')))
mutPatterns[sample(r*c,round(r*c*dens))]<-1

#Rowwise sorting
nsamples<-ncol(mutPatterns)

coveredGenes<-NA
uncoveredGenes<-rownames(mutPatterns)

idNull<-which(colSums(mutPatterns)==0)
nullCol<-matrix(c(mutPatterns[,idNull]),nrow(mutPatterns),
          length(idNull),dimnames = list(rownames(mutPatterns),colnames(mutPatterns)[idNull]))

idNonNull<-which(colSums(mutPatterns)>0)
mutPatterns<-matrix(c(mutPatterns[,idNonNull]),
                    nrow(mutPatterns),length(idNonNull),
                 dimnames=list(rownames(mutPatterns),colnames(mutPatterns)[idNonNull]))

coveredSamples<-NA
uncoveredSamples<-colnames(mutPatterns)
BS<-NA

while(length(uncoveredGenes)>0 & length(uncoveredSamples)>0){

  patterns<-matrix(c(mutPatterns[uncoveredGenes,uncoveredSamples]),
                   nrow = length(uncoveredGenes),
                   ncol = length(uncoveredSamples),
                   dimnames = list(uncoveredGenes,uncoveredSamples))

  if(length(uncoveredGenes)>1){
    bestInClass<-MExMaS.findBestInClass(patterns)
  }else{
    bestInClass<-uncoveredGenes
  }

  if(is.na(BS[1])){
    BS<-bestInClass
  }else{
    BS<-c(BS,bestInClass)
  }

  if(is.na(coveredGenes[1])){
    coveredGenes<-bestInClass
  }else{
    coveredGenes<-c(coveredGenes,bestInClass)
  }

  uncoveredGenes<-setdiff(uncoveredGenes,coveredGenes)
  toCheck<-matrix(c(patterns[bestInClass,uncoveredSamples]),
    nrow = 1,ncol=ncol(patterns),
    dimnames = list(bestInClass,uncoveredSamples))

  if (length(coveredGenes)==1){
    coveredSamples<-names(which(colSums(toCheck)>0))
  }else{
    coveredSamples<-c(coveredSamples,names(which(colSums(toCheck)>0)))
  }

  uncoveredSamples<-setdiff(uncoveredSamples,coveredSamples)

}

GENES<-c(BS,uncoveredGenes)

## Columnwise sorting
CID<-MExMaS.rearrangeMatrix(mutPatterns,GENES)
CID
```