Package ‘MCbiclust’

April 26, 2017

Type Package

Title Massive correlating biclusters for gene expression data and associated methods

Version 1.0.0

Date 2017-15-03

Author Robert Bentham

Maintainer Robert Bentham <robert.bentham.11@ucl.ac.uk>

Description Custom made algorithm and associated methods for finding, visualising and analysing biclusters in large gene expression data sets. Algorithm is based on with a supplied gene set of size n, finding the maximum strength correlation matrix containing m samples from the data set.

Depends R (>= 3.4)

Imports BiocParallel, graphics, utils, stats, AnnotationDbi, GO.db, org.Hs.eg.db, GGally, ggplot2, scales, cluster

Suggests gplots, knitr, rmarkdown, BiocStyle, gProfileR, MASS, dplyr, pander, devtools, testthat

License GPL-2

biocViews Clustering, Microarray, StatisticalMethod, Software, RNASeq, GeneExpression

RoxygenNote 5.0.1

VignetteBuilder knitr

LazyData true

NeedsCompilation no

R topics documented:

CCLE_samples ........................................... 2
CCLE_small .............................................. 3
CorScoreCalc ........................................... 3
CVEval .................................................. 4
CVPlot .................................................. 5
FindSeed ................................................. 7
GOEnrichmentAnalysis .................................... 8
HclustGenesHiCor ........................................ 9
CCLE_samples

Description
A dataset containing clinical information for the CCLE samples.

Usage
CCLE_samples

Format
A data frame with 967 rows and 14 variables:
- CCLE.name: Sample name identifier.
- Cell.line.primary.name: Cell line name.
- Cell.line.aliases: Any known aliases of cell line.
- Gender: Gender of patient cell line derived from.
- Site.Primary: Primary site cell line derived from.
- Histology: Histology of tumour cell line derived from.
- Hist.Subtype1: Histology subtype of tumour cell line derived from.
- Notes: Additional notes.
- Source: Source of the cell line.
- Expression.arrays: Expression array used.
- SNP.arrays: SNP array used.
- Oncomap: Oncomap mutation array used.
- Hybrid.Capture.Sequencing: Hybrid capture sequencing used.
- Name: Sample name identifier

Value
NA

Source
http://www.broadinstitute.org/ccle/data/browseData Filename: CCLE_sample_info_file_2012-04-06.txt
CCLE_small

Subset of expression levels of CCLE data

Description
A dataset containing the gene-centric RMA-normalized mRNA expression data for nearly 1000 genes and 500 samples taken as a random subset of the complete CCLE data. 1000 genes were selected randomly such that 500 were mitochondrial and 500 non-mitochondrial.

Usage
CCLE_small

Format
A data frame with 1000 rows and 500 variables:

- MKN74_STOMACH: mRNA expression on sample MKN74_STOMACH
- OC316_OVARY: mRNA expression on sample OC316_OVARY
- ...

@source http://www.broadinstitute.org/ccle/data/browseData Filename: CCLE_Expression_Entrez_2012-04-06.gct.gz

Value
NA

CorScoreCalc Calculate correlation score

Description
The standard method to calculate the correlation score used to judge biclusters in MChiclust

Usage
CorScoreCalc(gene.expr.matrix, sample.vec)

Arguments

gene.expr.matrix Gene expression matrix with genes as rows and samples as columns

sample.vec Vector of samples

Value
The correlation score
Examples

data(CCLE_small)
data(Mitochondrial_genes)

mito.loc <- which(row.names(CCLE_small) %in% Mitochondrial_genes)
CCLE.mito <- CCLE_small[mito.loc,]

random.seed <- sample(seq(length = dim(CCLE.mito)[2]), 10)
CCLE.seed <- FindSeed(gem = CCLE.mito,
  seed.size = 10,
  iterations = 100,
  messages = 100)

CorScoreCalc(CCLE.mito, random.seed)
CorScoreCalc(CCLE.mito, CCLE.seed)

CCLE.hicor.genes <- as.numeric(HclustGenesHiCor(CCLE.mito,
  CCLE.seed,
  cuts = 8))

CorScoreCalc(CCLE.mito[CCLE.hicor.genes,], CCLE.seed)

CVEval

Method for the calculation of a correlation vector

Description

Upon identifying a bicluster seed with FindSeed, one of the next steps is to identify which genes not in your chosen gene set are also highly correlated to the bicluster found. This is done by CVEval, and the output is known as the correlation vector.

Usage

CVEval(gem.part, gem.all, seed, splits)

Arguments

gem.part Part of gene expression matrix only containing gene set of interest with genes as rows and samples as columns
gem.all All of gene expression matrix
seed Seed of highly correlating samples
splits Number of cuts from hierarchical clustering

Details

CVEval uses hierarchical clustering to select the genes most representative of the bicluster and then uses the average expression of these genes across the sample seed and calculates the correlation of every gene measured across the sample seed to this average expression value.

The correlation vector is the output of this calculation.
CVPlot

Value

Correlation vector

Examples

data(CCLE_small)
data(Mitochondrial_genes)

mito.loc <- (row.names(CCLE_small) %in% Mitochondrial_genes)
CCLE.mito <- CCLE_small[mito.loc,]

set.seed(102)
CCLE.seed <- FindSeed(gem = CCLE.mito,
seed.size = 10,
iterations = 100,
messages = 1000)

CCLE.sort <- SampleSort(gem = CCLE.mito,seed = CCLE.seed,sort.length = 11)

# Full ordering are in Vignette_sort in sysdata.rda
CCLE.samp.sort <- MCbiclust:::Vignette_sort[[1]]

CCLE.pc1 <- PC1VecFun(top.gem = CCLE.mito,
seed.sort = CCLE.samp.sort,
n = 10)

CCLE.cor.vec <- CVEval(gem.part = CCLE.mito,
gem.all = CCLE_small,
seed = CCLE.seed,
splits = 10)

CCLE.bic <- ThresholdBic(cor.vec = CCLE.cor.vec,sort.order = CCLE.samp.sort,
pc1 = as.numeric(CCLE.pc1))

CCLE.pc1 <- PC1Align(gem = CCLE_small, pc1 = CCLE.pc1,
cor.vec = CCLE.cor.vec ,
sort.order = CCLE.samp.sort,
bic = CCLE.bic)

CCLE.fork <- ForkClassifier(CCLE.pc1, samp.num = length(CCLE.bic[[2]]))

CVPlot  Make correlation vector plot

Description

A function to visualise the differences between different found biclusters. Output is a matrix of plots. Each correlation vector is plotted against each other across the entire measured gene set in the lower diagonal plots, and a chosen gene set (e.g. mitochondrial) in the upper diagonal plots. The diagonal plots themselves show the density plots of the entire measured and chosen gene set. There are addition options to set the transparancy of the data points and names of the correlation vectors.
Usage

CVPlot(cv.df, geneset.loc, geneset.name, alpha1 = 0.005, alpha2 = 0.1, cnames = NULL)

Arguments

cv.df  A dataframe containing the correlation vectors of one or more patterns.
geneset.loc  A gene set of interest (e.g. mitochondrial) to be plotted separately from rest of genes.
geneset.name  Name of geneset (e.g. mitochondrial genes)
alpha1  Transparency level of non-gene set genes
alpha2  Transparency level of gene set genes
cnames  Character vector containing names for the correlation vector

Value

A plot of the correlation vectors

Examples

data(CCLE_small)
data(Mitochondrial_genes)

mito.loc <- which(row.names(CCLE_small) %in% Mitochondrial_genes)
CCLE.mito <- CCLE_small[mito.loc,]

CCLE.seed <- list()
CCLE.cor.vec <- list()

for(i in 1:3){
  set.seed(i)
  CCLE.seed[[i]] <- FindSeed(gem = CCLE.mito,
                           seed.size = 10,
                           iterations = 100,
                           messages = 100))

  for(i in 1:3){
    CCLE.cor.vec[[i]] <- CVEval(gem.part = CCLE.mito,
                                 gem.all = CCLE_small,
                                 seed = CCLE.seed[[i]],
                                 splits = 10))

  CCLE.cor.df <- (as.data.frame(CCLE.cor.vec))
CVPlot(cv.df = CCLE.cor.df, geneset.loc = mito.loc, geneset.name = "Mitochondrial", alpha1 = 0.5)
FindSeed

Find highly correlated seed of samples for gene expression matrix

Description

FindSeed() is the key function in MChiclust. It takes a gene expression matrix and by a stochastic method greedily searches for a seed of samples that maximizes the correlation score of the chosen gene set.

Usage

FindSeed(gem, seed.size, iterations, initial.seed = NULL, messages = 100)

Arguments

- **gem**: Gene expression matrix with genes as rows and samples as columns
- **seed.size**: Size of sample seed
- **iterations**: Number of iterations
- **initial.seed**: Initial seed used, if NULL randomly chosen
- **messages**: Frequency of progress messages

Details

Additional options allow for the search to start at a chosen seed, for instance if an improvement to a known seed is desired. The result of FindSeed() is dependent on the number of iterations, with above 1000 usually providing a good seed, and above 10000 an optimum seed.

Value

Highly correlated seed

Examples

data(CCLE_small)
data(Mitochondrial_genes)
mito.loc <- which(row.names(CCLE_small) %in% Mitochondrial_genes)
CCLE.mito <- CCLE_small[mito.loc,]
random.seed <- sample(seq(length = dim(CCLE.mito)[2]), 10)
CCLE.seed <- FindSeed(gem = CCLE.mito,
seed.size = 10,
iterations = 100,
messages = 100)

CorScoreCalc(CCLE.mito, random.seed)
CorScoreCalc(CCLE.mito, CCLE.seed)

CCLE.hicor.genes <- as.numeric(HclustGenesHiCor(CCLE.mito,
CCLE.seed,
cuts = 8))
GOEnrichmentAnalysis

Calculate gene set enrichment of correlation vector using Mann-Whitney test

Description

The Mann-Whitney test is typically used due to the values of the correlation vector, not being normally distributed. GOEnrichmentAnalysis provides an interface with the GO database annotation to find the most significant GO terms.

Usage

GOEnrichmentAnalysis(gene.names, gene.values, sig.rate)

Arguments

gene.names Names of the genes in standard gene name format.
gene.values Values associated with the genes, e.g. the correlation vector output of CVEval.
sig.rate Level of significance required after multiple hypothesis adjustment.

Value

Data frame of the significant gene sets, with GOID, GO Term, number of genes, number of genes in GO Term also in gene set, adjusted p-value, average value of correlation vector in gene set and phenotype describing whether average value of correlation vector is above or below the total average.

Examples

data(CCLE_small)
data(Mitochondrial_genes)

mito.loc <- (row.names(CCLE_small) %in% Mitochondrial_genes)
CCLE.mito <- CCLE_small[mito.loc,]

set.seed(101)
CCLE.seed <- FindSeed(gem = CCLE.mito,
seed.size = 10,
iterations = 100,
messages = 100)

CCLE.cor.vec <- CVEval(gem.part = CCLE.mito,
gem.all = CCLE_small,
seed = CCLE.seed, splits = 10)

# Significant GO terms can be calculated as follows:
# GEA <- GOEnrichmentAnalysis(gene.names = row.names(CCLE_small),
# gene.values = CCLE.cor.vec,
# sig.rate = 0.05)
**HclustGenesHiCor**

**Find the most highly correlated genes using hierarchical clustering**

**Description**

Upon finding an initial bicluster with `FindSeed()` not all the genes in the chosen geneset will be highly correlated to the bicluster. `HclustGenesHiCor()` uses the output of `FindSeed()` and hierarchical clustering to only select the genes that are most highly correlated to the bicluster. This is achieved by cutting the dendogram produced from the clustering into a set number of groups and then only selecting the groups that are most highly correlated to the bicluster.

**Usage**

```
HclustGenesHiCor(gem, seed, cuts)
```

**Arguments**

- **gem** Gene expression matrix with genes as rows and samples as columns
- **seed** Seed of highly correlating samples
- **cuts** Number of groups to cut dendogram into

**Value**

Numeric vector of most highly correlated genes

**Examples**

```r
data(CCLE_small)
data(Mitochondrial_genes)
mito.loc <- which(row.names(CCLE_small) %in% Mitochondrial_genes)
CCLE.mito <- CCLE_small[mito.loc,]
random.seed <- sample(seq(length = dim(CCLE.mito)[2]), 10)
CCLE.seed <- FindSeed(gem = CCLE.mito,
  seed.size = 10,
  iterations = 100,
  messages = 100)

CorScoreCalc(CCLE.mito, random.seed)
CorScoreCalc(CCLE.mito, CCLE.seed)

CCLE.hicor.genes <- as.numeric(HclustGenesHiCor(CCLE.mito,
  CCLE.seed,
  cuts = 8))

CorScoreCalc(CCLE.mito[CCLE.hicor.genes,, CCLE.seed]
```

MCbiclust: Massively Correlated biclustering

Description

MCbiclust is an R package for running massively correlating biclustering analysis. MCbiclust aims to find large scale biclusters with selected features being highly correlated with each other over a subset of samples.

Details

The package was originally designed in order to solve a problem in bioinformatics: to find biclusters representing different modes of regulation of mitochondria gene expression in disease states such as breast cancer. The same methods however, can be used on any gene expression data set to find biclusters of interest.

To learn more about MCbiclust, start with the vignette: browseVignettes(package = "MCbiclust")

Mitochondrial_genes

List of known mitochondrial genes

Description

A dataset from MitoCarta1.0 containing the 1023 mitochondrial genes Available from the broad institute: http://www.broadinstitute.org/scientific-community/science/programs/metabolic-disease-program/publications/mitocarta/mitocarta-in-0

Usage

Mitochondrial_genes

Format

A Character vector of the HGNC approved gene names

Value

NA

Source

https://www.broadinstitute.org/publications/broad807s
PC1VecFun

Calculate PC1 vector of found pattern

Description
The correlations found between the chosen geneset in a subset of samples can be summarised by looking at the first principal component (PC1) using principal component analysis (PCA).

Usage
PC1VecFun(top.gem, seed.sort, n)

Arguments
- top.gem: Gene expression matrix containing only highly correlating genes
- seed.sort: Ordering of samples according to strength of correlation
- n: Number of samples to use in calculation of PC1

Details
PC1VecFun() takes a gene expression matrix and the sample ordering and fits a PC1 value to all the samples based on a PCA analysis done on the first n samples.

Value
PC1 value for each sample

Examples
```r
data(CCLE_small)
data(Mitochondrial_genes)
mito.loc <- (row.names(CCLE_small) %in% Mitochondrial_genes)
CCLE.mito <- CCLE_small[mito.loc,]
set.seed(102)
CCLE.seed <- FindSeed(gem = CCLE.mito,
  seed.size = 10,
  iterations = 100,
  messages = 1000)
CCLE.sort <- SampleSort(gem = CCLE.mito,seed = CCLE.seed,sort.length = 11)
# Full ordering are in Vignette_sort in sysdata.rda
CCLE.samp.sort <- MCbiclust:::Vignette_sort[[1]]
CCLE.pc1 <- PC1VecFun(top.gem = CCLE.mito,
  seed.sort = CCLE.samp.sort,
  n = 10)
CCLE.cor.vec <- CVEval(gem.part = CCLE.mito,
  gem.all = CCLE_small,
  seed = CCLE.seed,
```
CCLE.bic <- ThresholdBic(cor.vec = CCLE.cor.vec, sort.order = CCLE.samp.sort, pc1 = as.numeric(CCLE.pc1))

CCLE.pc1 <- PC1Align(gem = CCLE_small, pc1 = CCLE.pc1, cor.vec = CCLE.cor.vec, sort.order = CCLE.samp.sort, bic = CCLE.bic)

CCLE.fork <- ForkClassifier(CCLE.pc1, samp.num = length(CCLE.bic[[2]]))

SampleSort

Methods for ordering samples

Description
After finding an initial bicluster with FindSeed() the next step is to extend the bicluster by ordering the remaining samples by how they preserve the correlation found.

Usage
SampleSort(gem, seed, num.cores = NULL, sort.length = NULL)

MultiSampleSortPrep(gem, av.corvec, top.genes.num, groups, initial.seeds)

Arguments
- **gem**: Gene expression matrix with genes as rows and samples as columns
- **seed**: Sample seed of highly correlating genes
- **num.cores**: Number of cores used in parallel evaluation
- **sort.length**: Number of samples to be sorted
- **av.corvec**: List of average correlation vector
- **top.genes.num**: Number of the top genes in correlation vector to use for sorting samples
- **groups**: List showing what runs belong to which correlation vector group
- **initial.seeds**: List of sample seeds from all runs

Details
SampleSort() is the basic function that achieves this, it takes the gene expression matrix, seed of samples, and also has options for the number of cores to run the method on and the number of samples to sort.

MultiSampleSortPrep() is a preparation function for SampleSort() when MCbiclust has been run multiple times and returns a list of gene expression matrices and seeds for each ‘distinct’ bicluster found.

Value
Order of samples by strength to correlation pattern
Examples

data(CCLE_small)
data(Mitochondrial_genes)

mito.loc <- (row.names(CCLE_small) %in% Mitochondrial_genes)
CCLE.mito <- CCLE_small[mito.loc,]

set.seed(102)
CCLE.seed <- FindSeed(gem = CCLE.mito,
  seed.size = 10,
  iterations = 100,
  messages = 1000)

CCLE.sort <- SampleSort(gem = CCLE.mito,seed = CCLE.seed,sort.length = 11)

# Full ordering are in Vignette_sort in sysdata.rda
CCLE.samp.sort <- MCbiclust:::Vignette_sort[[1]]

CCLE.pc1 <- PC1VecFun(top.gem = CCLE.mito,
  seed.sort = CCLE.samp.sort,
  n = 10)

CCLE.cor.vec <- CVEval(gem.part = CCLE.mito,
  gem.all = CCLE_small,
  seed = CCLE.seed,
  splits = 10)

CCLE.bic <- ThresholdBic(cor.vec = CCLE.cor.vec,sort.order = CCLE.samp.sort,
  pc1 = as.numeric(CCLE.pc1))

CCLE.pc1 <- PC1Align(gem = CCLE_small, pc1 = CCLE.pc1,
  cor.vec = CCLE.cor.vec ,
  sort.order = CCLE.samp.sort,
  bic =CCLE.bic)

CCLE.fork <- ForkClassifier(CCLE.pc1, samp.num = length(CCLE.bic[[2]]))
Arguments

- `cor.vec.mat`: Correlation matrix of the correlation vectors (CVs)
- `max.clusters`: Maximum number of clusters to divide CVs into
- `plots`: True or False for whether to show silhouette plots
- `seed1`: Value used to set random seed
- `rand.vec`: True or False for whether to add random correlation vector used for comparison

Value

The distinct clusters of correlation vectors

Examples

```r
data(CCLE_small)
data(Mitochondrial_genes)

mito.loc <- row.names(CCLE_small) %in% Mitochondrial_genes
CCLE.mito <- CCLE_small[mito.loc,]

CCLE.seed <- list()
CCLE.cor.vec <- list()

for(i in 1:5){
  set.seed(i)
  CCLE.seed[[i]] <- FindSeed(gem = CCLE.mito,
      seed.size = 10,
      iterations = 100,
      messages = 100)
}

for(i in 1:5){
  CCLE.cor.vec[[i]] <- CVEval(gem.part = CCLE.mito,
      gem.all = CCLE_small,
      seed = CCLE.seed[[i]],
      splits = 10))
}

CCLE.cor.mat <- as.matrix(as.data.frame(CCLE.cor.vec))

CCLE.clust.groups <- SilhouetteClustGroups(cor.vec.mat = CCLE.cor.mat,
    plots = TRUE,
    max.clusters = 10)

av.corvec.fun <- function(x) rowMeans(CCLE.cor.mat[,x])
CCLE.average.corvec <- lapply(X = CCLE.clust.groups,
    FUN = av.corvec.fun)

multi.sort.prep <- MultiSampleSortPrep(gem = CCLE_small,
    av.corvec = CCLE.average.corvec,
    top.genes.num = 750,
    groups = CCLE.clust.groups,
    initial.seeds = CCLE.seed)

multi.sort <- list()
for(i in seq_len(length(CCLE.clust.groups))){
  multi.sort[[i]] <- SampleSort(multi.sort.prep[[1]][[i]],
      seed = multi.sort.prep[[2]][[i]],
Description

A bicluster is the fundamental result found using MCbiclust. These three functions are essential for the precise definition of these biclusters.

Usage

ThresholdBic(cor.vec, sort.order, pc1, samp.sig = 0)

PC1Align(gem, pc1, cor.vec, sort.order, bic)

ForkClassifier(pc1, samp.num)

Arguments

cor.vec      Correlation vector (output of CVEval()).
sort.order   Order of samples (output of SampleSort()).
pc1          PC1 values for samples (output of PC1VecFun).
samp.sig     Value between 0 and 1 determining number of samples in bicluster
gem          Gene expression matrix containing genes as rows and samples as columns.
bic          bicluster (output of ThresholdBic())
samp.num     Number of samples in the bicluster

Details

ThresholdBic() takes as its main inputs the correlation vector (output of CVEval()), sample ordering (output of SampleSort()), PC1 vector (output of PC1VecFun) and returns a list of the genes and samples which belong to the bicluster according to a certain level of significance.

PC1Align() is a function used once the bicluster has been found to ensure that the upper fork samples (those with higher PC1 values) correspond to those samples that have genes with positive correlation vector values up-regulated.

ForkClassifier() is a function used to classify which samples are in the upper or lower fork.

Value

Defined bicluster
Examples

data(CCLE_small)
data(Mitochondrial_genes)

mito.loc <- (row.names(CCLE_small) %in% Mitochondrial_genes)
CCLE.mito <- CCLE_small[mito.loc,]

set.seed(102)
CCLE.seed <- FindSeed(gem = CCLE.mito,
  seed.size = 10,
  iterations = 100,
  messages = 1000)

CCLE.sort <- SampleSort(gem = CCLE.mito,seed = CCLE.seed,sort.length = 11)

# Full ordering are in Vignette_sort in sysdata.rda
CCLE.samp.sort <- MCbiclust:::Vignette_sort[[1]]

CCLE.pc1 <- PC1VecFun(top.gem = CCLE.mito,
  seed.sort = CCLE.samp.sort,
  n = 10)

CCLE.cor.vec <- CVEval(gem.part = CCLE.mito,
  gem.all = CCLE_small,
  seed = CCLE.seed,
  splits = 10)

CCLE.bic <- ThresholdBic(cor.vec = CCLE.cor.vec,sort.order = CCLE.samp.sort,
  pc1 = as.numeric(CCLE.pc1))

CCLE.pc1 <- PC1Align(gem = CCLE_small, pc1 = CCLE.pc1,
  cor.vec = CCLE.cor.vec ,
  sort.order = CCLE.samp.sort,
  bic =CCLE.bic)

CCLE.fork <- ForkClassifier(CCLE.pc1, samp.num = length(CCLE.bic[[2]]))
Index

*Topic datasets
  CCLE_samples, 2
  CCLE_small, 3
  Mitochondrial_genes, 10

CCLE_samples, 2
CCLE_small, 3
CorScoreCalc, 3
CVEval, 4
CVPlot, 5

FindSeed, 7
ForkClassifier (ThresholdBic), 15

GOEnrichmentAnalysis, 8
HclustGenesHiCor, 9

MCbiclust, 10
MCbiclust-package (MCbiclust), 10
Mitochondrial_genes, 10
MultiSampleSortPrep (SampleSort), 12

PC1Align (ThresholdBic), 15
PC1VecFun, 11

SampleSort, 12
SilhouetteClustGroups, 13

ThresholdBic, 15