Package ‘dcanr’

December 15, 2025

Title Differential co-expression/association network analysis

Version 1.26.0

Description This package implements methods and an evaluation framework to infer differential co-expression/association networks. Various methods are implemented and can be evaluated using simulated datasets. Inference of differential co-expression networks can allow identification of networks that are altered between two conditions (e.g., health and disease).

biocViews NetworkInference, GraphAndNetwork, DifferentialExpression, Network

Depends R (>= 3.6.0)

License GPL-3

Encoding UTF-8

LazyData false

Imports igraph, foreach, plyr, stringr, reshape2, methods, Matrix, graphics, stats, RColorBrewer, circlize, doRNG

Suggests EBcoexpress, testthat, EBarrays, GeneNet, mclust, minqa, SummarizedExperiment, Biobase, knitr, rmarkdown, BiocStyle, edgeR

RoxygenNote 7.2.0

Collate 'LDGM.R' 'dcZscore.R' 'dcanr.R' 'statistical_tests.R' 'performance_metrics.R' 'multtest_adjust.R' 'evaluation_functions.R' 'inference_methods.R' 'inference_methods_generic.R' 'network_inference.R' 'sim102.R' 'simulation_accessors.R' 'simulation_plot.R'

VignetteBuilder knitr

Enhances parallel, doSNOW, doParallel

URL https://davislaboratory.github.io/dcanr/, https://github.com/DavisLaboratory/dcanr

BugReports https://github.com/DavisLaboratory/dcanr/issues

git_url https://git.bioconductor.org/packages/dcanr

git_branch RELEASE_3_22

git_last_commit 848b497

git_last_commit_date 2025-10-29

Repository Bioconductor 3.22

Date/Publication 2025-12-15

Author Dharmesh D. Bhuva [aut, cre] (ORCID: <https://orcid.org/0000-0002-6398-9157>)

Maintainer Dharmesh D. Bhuva <bhuva.d@wehi.edu.au>

Contents

dcanr-package 2
cor.pairs 4
dcAdjust 4
dcEvaluate 5
dcMethods 6
dcNetwork 7
dcPipeline 8
dcScore 10
dcTest 11
dcZscore 13
getSimData 14
mi.ap 15
perfMethods 16
performanceMeasure 16
plotSimNetwork 17
sim102 18

Index **20**

`dcanr-package` *dcanr: Differential co-expression/association network analysis*

Description

Methods and an evaluation framework for the inference of differential co-expression/association networks.

Details

There are three categories of functions available

1. Differential co-expression methods (DC) - These functions are used to perform a differential co-expression analysis on experimental data with binary conditions.
2. Functions to evaluate DC methods - These functions are used to evaluate methods implemented in the package and novel methods on simulated data. Expression data is simulated for 2 conditions, wild-type and knock-down of given genes.
3. By-products of implementations

Differential co-expression methods (DC)

- dcMethods
- dcScore
- dcTest
- dcAdjust
- dcNetwork

Functions to evaluate DC methods

Accessors of simulated data:

- getConditionNames
- getSimData
- getTrueNetwork
- plotSimNetwork

Functions for evaluating inference methods

- dcPipeline
- dcEvaluate

By-products of implementations

These are functions used in the package but have further uses in general:

- cor.pairs - a faster implementation of pairwise correlation computation
- mi.ap - pairwise computation of mutual information MI with data discretisation performed using adaptive partitioning
- perfMethods - available performance metrics
- performanceMeasure - performance measures of prediction algorithms. Predictions have to be binary

Author(s)

Maintainer: Dharmesh D. Bhuva <bhuva.d@wehi.edu.au> (ORCID)

See Also

Useful links:

- https://davislaboratory.github.io/dcanr/
- https://github.com/DavisLaboratory/dcanr
- Report bugs at https://github.com/DavisLaboratory/dcanr/issues

cor.pairs *Fast pairwise correlation estimation*

Description

Fast estimation of pairwise correlation coefficients.

Usage

```
cor.pairs(emat, cor.method = c("pearson", "spearman"))
```

Arguments

emat	a numeric matrix
cor.method	a character, specifying the method to use for estimation. Possible values are 'pearson' (default) and 'spearman'

Value

a numeric matrix with estimated correlation coefficients

Examples

```
x <- matrix(rnorm(200), 100, 2)
cor.pairs(x)
cor.pairs(x, cor.method = 'spearman')
```

dcAdjust *Adjust for multiple testing in differential association analysis*

Description

Adjust for multiple hypothesis testing after performing statistical tests using `dcTest`. This can be performed using a method provided by the users. `p.adjust` is used by default.

Usage

```
dcAdjust(dcpvals, f = stats::p.adjust, ...)
```

Arguments

dcpvals	a matrix, the result of the `dcTest` function. The results should be passed as produced by the function and not modified in intermediate steps
f	a function, the function to be used for adjustment. `p.adjust` from the `stats` package is the default with the specific adjustment method 'fdr' used. The range of available methods can be accessed using `p.adjust.methods`. Custom functions should accept a numeric vector of p-values as the first argument
...	additional parameters to the adjustment function such as `method`

Details

Ensure that the p-value matrix passed to this function is the one produced by `dcTest`. Any modification to the result matrix will result in failure of the function.

This method applies the adjustment method only to one triangle of the matrix to ensure adjustment is not performed for duplicated tests (symmetric matrix). As results from the DiffCoEx and EBcoexpress do not produce p-values, this method does not change anything thereby returning the original matrix.

Value

a matrix, of adjusted p-values (or scores in the case of DiffCoEx and EBcoexpress) representing significance of differential associations.

See Also

`dcTest` `p.adjust`

Examples

```
x <- matrix(rnorm(60), 2, 30)
cond <- rep(1:2, 15)
zscores <- dcScore(x, cond)
pvals <- dcTest(zscores, emat = x, condition = cond)
dcAdjust(pvals, p.adjust, method = 'fdr')
```

`dcEvaluate` *Evaluate performance of DC methods on simulations*

Description

Quantify the performance of a differential co-expression pipeline on simulated data.

Usage

```
dcEvaluate(
  simulation,
  dclist,
  truth.type = c("association", "influence", "direct"),
  perf.method = "f.measure",
  combine = TRUE,
  ...
)
```

Arguments

`simulation`	a list, storing data and results generated from simulations
`dclist`	a list of igraphs, produced using `dcPipeline`
`truth.type`	a character, specifying which level of the true network to retrieve: 'association' (default), 'influence' or 'direct'

perf.method	a character, specifying the method to use. Available methods can be accessed using perfMethods
combine	a logical, indicating whether differential networks from independent knock-outs should be treated as a single inference or independent inferences (defaults to TRUE)
...	additional parameters to be passed on to the performance metric method (see performanceMeasure)

Value

a numeric, representing the performance metric. A single value if combine = TRUE and a named vector otherwise.

See Also

dcPipeline, performanceMeasure, perfMethods

Examples

```
data(sim102)

#run a standard pipeline
resStd <- dcPipeline(sim102, dc.func = 'zscore')
dcEvaluate(sim102, resStd)
dcEvaluate(sim102, resStd, combine = FALSE)
```

---

dcMethods *Get names of differential co-expression methods*

Description

Returns a list of differential co-expression methods

Usage

```
dcMethods()
```

Value

names of methods implemented

Examples

```
dcMethods()
```

dcNetwork *Generate a differential network from a DC analysis*

Description

Threshold the results from a differential co-expression analysis and create a differential network.

Usage

```
dcNetwork(dcscores, dcpvals = NULL, thresh = NULL, ...)
```

Arguments

dcscores	a matrix, the result of the dcScore function. The results should be passed as produced by the function and not modified in intermediate steps
dcpvals	a matrix or NULL, raw or adjusted p-values resulting from dcTest or dcAdjust respectively. Should be left NULL only if method is EBcoexpress or DiffCoEx
thresh	a numeric, threshold to apply. If NULL, defaults to 0.1 for methods that generate a p-value, 0.9 for posterior probabilities from EBcoexpress and 0.1 on the absolute score from DiffCoEx
...	see details

Details

No extra arguments required for this function. The ellipsis are used to allow flexibility in pipelines.

Value

an igraph object, representing the differential network. Scores are added as edge attributes with the name 'score'

See Also

dcScore, dcTest, dcAdjust

Examples

```
#create data
set.seed(360)
x <- matrix(rnorm(120), 4, 30)
cond <- rep(1:2, 15)

#perform analysis - z-score
zscores <- dcScore(x, cond)
pvals <- dcTest(zscores, emat = x, condition = cond)
pvals <- dcAdjust(pvals, p.adjust, method = 'fdr')
ig <- dcNetwork(zscores, pvals, 0.1)

#perform analysis - DiffCoEx
dcscores <- dcScore(x, cond, dc.method = 'diffcoex')
ig <- dcNetwork(dcscores, thresh = 0.001)
```

```
#plot the resulting differential co-expression network
igraph::plot.igraph(ig)
```

---

dcPipeline *Run a DC pipeline on a simulation*

Description

Run a differential co-expression pipeline on data from a simulation experiment. A default pipeline can be used which consists of methods in the package or custom pipelines can be provided.

Usage

```
dcPipeline(
  simulation,
  dc.func = "zscore",
  precomputed = FALSE,
  continuous = FALSE,
  cond.args = list(),
  ...
)
```

Arguments

`simulation`	a list, storing data and results generated from simulations
`dc.func`	a function or character. Character represents one of the method names from `dcMethods` which is run with the default settings. A function can be used to provide custom processing pipelines (see details)
`precomputed`	a logical, indicating whether the precomputed inference should be used or a new one computed (default FALSE)
`continuous`	a logical, indicating whether binary or continuous conditions should be used (default FALSE). No methods implemented currently use continuous conditions. This is to allow custom methods that require continuous conditions
`cond.args`	a list, containing condition-specific arguments for the DC inference pipeline. See details
`...`	additional parameters to `dc.func`

Details

If `dc.func` is a character, the existing methods in the package will be run with their default parameters. The pipeline is as such: dcScore -> dcTest -> dcAdjust -> dcNetwork, resulting in a igraph object. Parameters to the independent processing steps can also be provided to this function as shown in the examples.

If `precomputed` is TRUE while `dc.func` is a character, pre-computed results will be used. These can then be evaluated using `dcEvaluate`.

Custom pipelines need to be coded into a function which can then be provided instead of a character. Functions must have the following structure:

```
function(emat, condition, ...)
```

They must return either an igraph object or an adjacency matrix stored in a base R 'matrix' or the S4 'Matrix' class, containing all genes in the expression matrix 'emat'. See examples for how the in-built functions are combined into a pipeline.

If the pipeline (in-built or custom) requires condition-specific parameters to run, cond.args can be used to pass these. For instance, LDGM requires lambda OR the number of edges in the target network to be specified for each inference/condition. For the latter case and with 3 different conditions, this can be done by setting `cond.args = list('ldgm.ntarget' = c(100, 140, 200))`. Non-specific arguments should be passed directly to the `dcPipeline` function call.

Value

a list of igraphs, representing the differential network for each independent condition (knock-out).

See Also

`plot.igraph`, `dcScore`, `dcTest`, `dcAdjust`, `dcNetwork`, `dcMethods`

Examples

```
data(sim102)

#run a standard pipeline
resStd <- dcPipeline(sim102, dc.func = 'zscore')

#run a standard pipeline and specify params
resParam <- dcPipeline(sim102, dc.func = 'zscore', cor.method = 'pearson')

#run a standard pipeline and specify condition-specific params
resParam <- dcPipeline(
  sim102,
  dc.func = 'diffcoex',
  #arguments for the conditions ADR1 knockdown and UME6 knockdown resp.
  cond.args = list(diffcoex.beta = c(6, 20))
)

#retrieve pre-computed results
resPrecomputed <- dcPipeline(sim102, dc.func = 'zscore', precomputed = TRUE)

#run a custom pipeline
analysisInbuilt <- function(emat, condition, dc.method = 'zscore', ...) {
  #compute scores
  score = dcScore(emat, condition, dc.method, ...)
  #perform statistical test
  pvals = dcTest(score, emat, condition, ...)
  #adjust tests for multiple testing
  adjp = dcAdjust(pvals, ...)
  #threshold and generate network
  dcnet = dcNetwork(score, adjp, ...)

  return(dcnet)
}
resCustom <- dcPipeline(sim102, dc.func = analysisInbuilt)

plot(resCustom[[1]])
```

`dcScore` *Compute scores from differential association analysis*

Description

Implementations and wrappers for existing implementations for methods inferring differential associations/co-expression. This method requires a matrix of expression and a binary condition to compute the differential association scores for all pairs of features (genes). Applications are not limited to analysis of gene expression data and may be used for differential associations in general.

Usage

```
dcScore(emat, condition, dc.method, ...)

## S4 method for signature 'matrix'
dcScore(emat, condition, dc.method = "zscore", ...)

## S4 method for signature 'Matrix'
dcScore(emat, condition, dc.method = "zscore", ...)

## S4 method for signature 'data.frame'
dcScore(emat, condition, dc.method = "zscore", ...)

## S4 method for signature 'ExpressionSet'
dcScore(emat, condition, dc.method = "zscore", ...)

## S4 method for signature 'SummarizedExperiment'
dcScore(emat, condition, dc.method = "zscore", ...)

## S4 method for signature 'DGEList'
dcScore(emat, condition, dc.method = "zscore", ...)
```

Arguments

`emat`	a matrix, Matrix, data.frame, ExpressionSet, SummarizedExperiment or DGEList
`condition`	a numeric, (with 1's and 2's representing a binary condition), a factor with 2 levels or a character representing 2 conditions
`dc.method`	a character, representing the method to use. Use `dcMethods()` to get a list of methods
`...`	possible arguments are `cor.method`, `diffcoex.beta`, `ebcoexpress.useBWMC`, `ebcoexpress.plot`, `ldgm.lambda`, `ldgm.ntarget` and `ldgm.iter`. See details

Details

When using data from sequencing experiments, make sure appropriate filtering for low counts and data transformation has been performed. Not doing so will affect estimation of correlation coefficients which most methods rely on.

Additional method specific parameters can be supplied to the function. `cor.method` can be set to either 'pearson' (default) or 'spearman' to determine the method to use for estimating correlations. These are the two measures currently supported in the package. We recommend using the 'spearman' correlation when dealing with sequencing data.

The beta parameter in the DiffCoEx method can be specified using `diffcoex.beta` (defaults to 6). This enable soft thresholding of correlations similar to WGCNA.

EBcoexpress specific parameters include `ebcoexpress.useBWMC` (defaults to `TRUE`) representing whether to use the bi-weight mid-correlation coefficient or not, and `ebcoexpress.plot` which plots the diagnostic plots if set to `TRUE` (defaults to `FALSE`).

LDGM specific parameters include `ldgm.lambda`, `ldgm.ntarget` and `ldgm.iter`. `ldgm.lambda` specifies the L1 regularisation parameter to use when fitting the model. This can be tuned and specified by the user. Alternatively, this can be tuned such that the resulting network has a specified number of edges. In this case, `ldgm.ntarget` should be specified instead. `ldgm.iter` is the maximum number of iterations to perform when tuning `ldgm.lambda` using `ldgm.ntarget` (defaults to 50).

EBcoexpress and GGM-based are implemented by providing interfaces to, or using functions from the `EBcoexpress`, `GeneNet`, and `COSINE` packages respectively. If using any of these methods, please **cite** the appropriate packages and the appropriate methodology articles.

Value

a matrix, of scores/statistics representing differential associations; p-values will be returned if FTGI is used and posterior probabilities if EBcoexpress is used.

See Also

`dcMethods`

Examples

```
x <- matrix(rnorm(60), 2, 30)
cond <- rep(1:2, 15)
dcScore(x, cond) #defaults to zscore
dcScore(x, cond, dc.method = 'diffcoex')
```

`dcTest` *Statistical test for differential association analysis*

Description

Perform statistical tests for scores generated using `dcScore`. Selects appropriate tests for the different methods used in computing scores. The exact test is selected based on the scoring method used and cannot be manually specified. Available tests include the z-test and permutation tests. Parallel computation supported for the permutation test.

Usage

```
dcTest(dcscores, emat, condition, ...)
```

Arguments

dcscores	a matrix, the result of the dcScore function. The results should be passed as produced by the function and not modified in intermediate steps
emat	a matrix, data.frame, ExpressionSet, SummarizedExperiment or DGEList. This should be the one passed to dcScore
condition	a numeric, (with 1's and 2's representing a binary condition), a factor with 2 levels or a character representing 2 conditions. This should be the one passed to dcScore
...	see details

Details

Ensure that the score matrix passed to this function is the one produced by dcScore. Any modification to the result matrix will cause this function to fail. This is intended as the test need to be performed on the entire score matrix, not subsets.

The appropriate test is chosen automatically based on the scoring method used. A z-test is performed for the z-score method while no tests are performed for DiffCoEx, EBcoexpress and FTGI. Permutation tests are performed for the remainder of methods by permutation sample labels. Statistics from a permutation are pooled such that statistics from all scores are used to evaluate a single observed score.

Additional method specific parameters can be supplied to the function when performing permutation tests. B specifies the number of permutations to be performed and defaults to 20.

If a cluster exists, computation in a permutation test will be performed in parallel (see examples).

Value

a matrix, of p-values (or scores in the case of DiffCoEx and EBcoexpress) representing significance of differential associations. DiffCoEx will return scores as the publication specifies direct thresholding of scores and EBcoexpress returns posterior probabilities.

See Also

dcMethods, dcScore

Examples

```
x <- matrix(rnorm(60), 2, 30)
cond <- rep(1:2, 15)
scores <- dcScore(x, cond, dc.method = 'mindy')
dcTest(scores, emat = x, condition = cond)

## Not run:
#running in parallel
num_cores = 2
cl <- parallel::makeCluster(num_cores)
doSNOW::registerDoSNOW(cl) #or doParallel
set.seed(36) #for reproducibility
dcTest(scores, emat = x, condition = cond, B = 100)
parallel::stopCluster(cl)

## End(Not run)
```

dcZscore — *DC analysis using the z-score method*

Description

This function packs the entire DC analysis pipeline using the z-score method. It simplifies the implementation of the analysis and increases the flexibility of the analysis (not just limited to all pairwise comparisons).

Usage

```
dcZscore(
  emat,
  condition,
  from = NULL,
  to = NULL,
  fdrthresh = 0.1,
  cor.method = c("spearman", "pearson")
)
```

Arguments

`emat`	a matrix, Matrix, data.frame, ExpressionSet, SummarizedExperiment or DGEList
`condition`	a numeric, (with 1's and 2's representing a binary condition), a factor with 2 levels or a character representing 2 conditions
`from`	a character vector, with the names of nodes from which comparisons need to be performed.
`to`	a character vector, with the names of nodes to which comparisons need to be performed.
`fdrthresh`	a numeric, specifying the FDR cutoff to apply to the inferred network.
`cor.method`	a character, either 'spearman' (default) or 'pearson' specifying the correlation computation method to use.

Value

an igraph object, containing the differential coexpression network.

Examples

```
x <- matrix(rnorm(60), 10, 30)
rownames(x) = 1:10
cond <- rep(1:2, 15)
dcZscore(x, cond)
dcZscore(x, cond, to = 1:2)
```

getSimData — *Get data and conditions from a given knock-down (KD)*

Description

Retrieves the simulated expression matrix and sample classification for a specific knock-down experiment.

Usage

```
getSimData(simulation, cond.name = NULL, full = FALSE)

getConditionNames(simulation)

getTrueNetwork(
  simulation,
  cond.name = NULL,
  truth.type = c("association", "influence", "direct"),
  full = FALSE
)
```

Arguments

`simulation`	a list, storing data and results generated from simulations
`cond.name`	a character, indicating the knock-down to use to derive conditions. Multiple knock-downs (KDs) are performed per simulation. If `NULL`, the first KD is chosen
`full`	a logical, indicating whether genes associated with the condition should be excluded. Defaults to `FALSE` and is recommended
`truth.type`	a character, specifying which level of the true network to retrieve: 'association' (default), 'influence' or 'direct'

Details

Genes discarded when `full` is `FALSE` are those that are solely dependent on the condition. These genes are discarded from the analysis to focus on those that are differentially co-expressed, not coordinately co-expressed.

The names of all genes knocked-out can be retrieved using `getConditionNames`.

The direct, influence and association networks represent different levels of true differential networks. The direct network contains differential regulatory interactions present in the original network. The influence network includes upstream interactions and the association network includes non-causative differential interactions.

Value

a list, containing `emat`, a matrix representing the expression data, `condition`, a numeric containing the classification of samples, and , `condition_c`, a numeric containing the expression levels of the KD gene (continuous condition) for `getSimData`; the names of all genes that are KD for `getConditionNames`; and an adjacency matrix for `getTrueNetwork`.

Functions

- `getSimData`: get the expression matrix and sample classification
- `getConditionNames`: get names of the conditions (KDs)
- `getTrueNetwork`: get the true differential network

See Also

`dcScore`

Examples

```
data(sim102)
KDs <- getConditionNames(sim102)

#get simulated data
simdata <- getSimData(sim102, KDs[2])
cond <- simdata$condition
emat <- simdata$emat
zscores <- dcScore(emat, cond)

#get the true network to evaluate against
truenet <- getTrueNetwork(sim102, KDs[2], truth.type = 'association')
```

`mi.ap` *Mutual information using adaptive partitioning*

Description

Computes the mutual information between all pairs of variables in the matrix (along the columns). Variables are discretised using the adaptive partitioning algorithm

Usage

```
mi.ap(mat)
```

Arguments

`mat`	a numeric matrix

Value

matrix of pairwise mutual information estimates

Examples

```
x <- matrix(rnorm(200), 100, 2)
mi.ap(x)
```

perfMethods — *Get names of performance metric methods*

Description

Returns a list of performance metrics

Usage

```
perfMethods()
```

Value

names of methods implemented

Examples

```
perfMethods()
```

performanceMeasure — *Performance metrics to evaluate classification*

Description

Quantify the performance of a classification algorithm. Predictions and truth both have to be binary.

Usage

```
performanceMeasure(pred, obs, perf.method = "f.measure", ...)
```

Arguments

`pred`	a logical or numeric, where 0 and FALSE represent control, and, 1 and TRUE represent cases
`obs`	a logical or numeric, where 0 and FALSE represent control, and, 1 and TRUE represent cases
`perf.method`	a character, specifying the method to use. Available methods can be accessed using `perfMethods`
`...`	additional parameters to methods. see details

Details

The F-measure requires the beta parameter which can be specified using `f.beta` which defaults to 1 thereby computing the F1-measure.

Value

a numeric, representing the performance

See Also

perfMethods

Examples

```
pred <- sample(0:1, 100, replace = TRUE, prob = c(0.75, 0.25))
obs <- sample(0:1, 100, replace = TRUE, prob = c(0.75, 0.25))

#compute the F1 and F2 scores
f1 <- performanceMeasure(pred, obs)
f2 <- performanceMeasure(pred, obs, f.beta = 2)
```

---

plotSimNetwork — *Plot source and true differential networks from simulations*

Description

Plots either the source network or the true differential network for all KDs performed in the simulation. KD nodes are coloured with their resulting differential networks coloured accordingly.

Usage

```
plotSimNetwork(
  simulation,
  what = c("source", "direct", "influence", "association"),
  ...
)
```

Arguments

simulation	a list, storing data and results generated from simulations
what	a character, indicating which network to retrieve, 'source' (default), 'direct', 'influence' or 'association'
...	additional parameters to plot.igraph

Details

The direct, influence and association networks represent different levels of true differential networks. The direct network contains differential regulatory interactions present in the original network. The influence network includes upstream interactions and the association network includes non-causative differential interactions.

Value

a plot of the network

See Also

plot.igraph

Examples

```
data(sim102)
plotSimNetwork(sim102)
plotSimNetwork(sim102, what = 'direct')
plotSimNetwork(sim102, what = 'influence')
plotSimNetwork(sim102, what = 'association')
```

`sim102` *Simulated expression data with knock-outs*

Description

A dataset containing simulated expression dataset. Data is simulated using a dynamical systems model from a network sampled from the S. Cerevisiae regulatory network. The dataset is a list containing the results from the simulation, and other information generated subsequently.

Usage

```
sim102
```

Format

A named list with 14 elements:

simitr a numeric, indicating the iteration of the simulation (a total of 1000 were performed and 812 converged)

scores an S4 Matrix, containing vectorised inference scores of applying the methods implemented in the package. These are precomputed predictions

inputmodels a named list, storing the parameters used to sample the initial values of input genes. Proportions, means and variances of each gene is stored for each gene

staticnet an igraph object, storing the initial regulatory network (150 node network)

infnet an igraph object, representing the true differential network as determined using sensitivity analysis of the model

netlayout a matrix (150 x 2), storing the (x, y) positions of nodes for laying out the graph

infdens a numeric, network density of the true differential association network

numinput a numeric, the number of input genes in the regulatory network. These are genes that have no regulators therefore need to be pre-defined

numbimodal a numeric, the number of input genes that are knocked-down therefore have a bi-modal distribution

numtfs a numeric, the number of genes in the network that regulate any other gene (are TFs)

numcotargets a numeric, the number of genes that are co-regulated, i.e. regulated by more than one TF

data an S4 Matrix, the expression data with samples along the columns and genes along the rows. Condition classification (KD vs WT) are stored as attributes of this object

triplets a data frame, consisting of gene triplets representing TF- Target associations conditioned on the gene knocked-down. Triplets are annotated for being in either the direct, influence and association networks

sensmat an S4 Matrix, sensitivities of genes to TFs based on perturbation analysis of the simulation model

Source

LINK TO PAPERRRR

Index

∗ **classif**
dcanr-package, 2
∗ **datasets**
sim102, 18
∗ **graphs**
dcanr-package, 2
∗ **internal**
dcanr-package, 2

cor.pairs, *3*, 4

dcAdjust, *3*, 4, *7*, *9*
dcanr *(dcanr-package)*, 2
dcanr-package, 2
dcEvaluate, *3*, 5
dcMethods, *3*, 6, *9*, *11*, *12*
dcNetwork, *3*, 7, *9*
dcPipeline, *3*, *6*, 8
dcScore, *3*, *7*, *9*, 10, *12*, *15*
dcScore,data.frame-method *(dcScore)*, 10
dcScore,DGEList-method *(dcScore)*, 10
dcScore,ExpressionSet-method *(dcScore)*, 10
dcScore,Matrix-method *(dcScore)*, 10
dcScore,matrix-method *(dcScore)*, 10
dcScore,SummarizedExperiment-method *(dcScore)*, 10
dcTest, *3*, *5*, *7*, *9*, 11
dcZscore, 13

getConditionNames, *3*
getConditionNames *(getSimData)*, 14
getSimData, *3*, 14
getTrueNetwork, *3*
getTrueNetwork *(getSimData)*, 14

mi.ap, *3*, 15

p.adjust, *5*
perfMethods, *3*, *6*, 16, *17*
performanceMeasure, *3*, *6*, 16
plot.igraph, *9*, *17*
plotSimNetwork, *3*, 17

sim102, 18