Package 'DNEA'

November 9, 2026

Title Differential Network Enrichment Analysis for Biological Data

Version 1.2.0

Description The DNEA R package is the latest implementation of the Differential Network Enrichment Analysis algorithm and is the successor to the Filigree Java-application described in Iyer et al. (2020). The package is designed to take as input an m x n expression matrix for some -omics modality (ie. metabolomics, lipidomics, proteomics, etc.) and jointly estimate the biological network associations of each condition using the DNEA algorithm described in Ma et al. (2019). This approach provides a framework for data-driven enrichment analysis across two experimental conditions that utilizes the underlying correlation structure of the data to determine feature-feature interactions.

License MIT + file LICENSE

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

output BiocStyle::html_document vignette: >
%\VignetteIndexEntry{Vignette Title}
%\VignetteEngine{knitr::rmarkdown} %\VignetteEncoding{UTF-8}

Imports BiocParallel, dplyr, gdata, glasso, igraph (>= 2.0.3), janitor, Matrix, methods, netgsa, stats, stringr, utils, SummarizedExperiment

Collate 'JSEM-internals.R' 'aggregate-features.R' 'all-classes.R' 'all-generics.R' 'all-methods.R' 'clustering-internals.R' 'initiator.R' 'start-here.R' 'utilities-internals.R' 'utilities-exported.R' 'primary.R'

Depends R (>= 4.2)

LazyData false

Suggests BiocStyle, ggplot2, Hmisc, kableExtra, knitr, pheatmap, rmarkdown, testthat (>= 3.0.0), withr, airway

Enhances massdataset

URL https://github.com/Karnovsky-Lab/DNEA

BugReports https://github.com/Karnovsky-Lab/DNEA/issues

biocViews Metabolomics, Proteomics, Lipidomics, DifferentialExpression, NetworkEnrichment, Network, Clustering, DataImport

Config/testthat/edition 3

VignetteBuilder knitr

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

git_branch RELEASE_3_23

git_last_commit e3a9713

git_last_commit_date 2026-04-28

Repository Bioconductor 3.23

Date/Publication 2026-11-08

Author Christopher Patsalis [cre, aut] (ORCID: <https://orcid.org/0009-0003-4585-0017>), Gayatri Iyer [aut], Alla Karnovsky [fnd] (NIH_GRANT: 1U01CA235487), George Michailidis [fnd] (NIH_GRANT: 1U01CA235487)

Maintainer Christopher Patsalis <chrispatsalis@gmail.com>

Contents

DNEA-package . . . 3
addExpressionData . . . 4
adjacencyGraph . . . 6
adjacencyMatrix . . . 7
aggregateFeatures . . . 8
BICscores . . . 11
BICtune . . . 12
CCsummary . . . 15
clusterNet . . . 16
collapsed_DNEA-class . . . 17
consensusClusteringResults-class . . . 19
createDNEAobject . . . 19
datasetSummary . . . 21
diagnostics . . . 22
DNEA-class . . . 24
DNEAinputSummary-class . . . 26
dnw . . . 27
edgeList . . . 28

expressionData . . . 29
featureNames . . . 30
filterNetworks . . . 31
getNetworkFiles . . . 32
getNetworks . . . 33
includeMetadata . . . 35
lambdas2Test . . . 37
massDataset2DNEA . . . 38
metab_data . . . 40
metaData . . . 41
netGSAresults . . . 42
networkGroupIDs . . . 43
networkGroups . . . 44
nodeList . . . 45
numFeatures . . . 46
numSamples . . . 47
optimizedLambda . . . 48
plotNetworks . . . 49
projectName . . . 51
runNetGSA . . . 52
sampleNames . . . 53
selectionProbabilities . . . 54
selectionResults . . . 55
stabilitySelection . . . 56
subnetworkMembership . . . 59
sumExp2DNEA . . . 60
T1Dmeta . . . 61
TEDDY . . . 62

Index **64**

`DNEA-package` *DNEA: Differential Network Enrichment Analysis for Biological Data*

Description

The DNEA R package is the latest implementation of the Differential Network Enrichment Analysis algorithm and is the successor to the Filigree Java-application described in Iyer et al. (2020). The package is designed to take as input an m x n expression matrix for some -omics modality (ie. metabolomics, lipidomics, proteomics, etc.) and jointly estimate the biological network associations of each condition using the DNEA algorithm described in Ma et al. (2019). This approach provides a framework for data-driven enrichment analysis across two experimental conditions that utilizes the underlying correlation structure of the data to determine feature-feature interactions.

Primary Components

The main workflow contains the following functions:

1. createDNEAobject
2. BICtune
3. stabilitySelection
4. getNetworks
5. clusterNet
6. runNetGSA

A more descriptive workflow can be viewed in the package vignette. This can be accessed by running `vignette("DNEA")` in the console.

Author(s)

Maintainer: Christopher Patsalis <chrispatsalis@gmail.com> (ORCID)

Authors:

- Gayatri Iyer <griyer@umich.edu>

Other contributors:

- Alla Karnovsky <akarnovs@med.umich.edu> (1U01CA235487) [funder]
- George Michailidis <gmichail@ufl.edu> (1U01CA235487) [funder]

See Also

Useful links:

- https://github.com/Karnovsky-Lab/DNEA
- Report bugs at https://github.com/Karnovsky-Lab/DNEA/issues

---

addExpressionData *Include custom normalized data in the DNEA object*

Description

This function allows the user to input custom-normalized data into the DNEA object for use in DNEA analysis.

Usage

```
addExpressionData(object, dat, assay_name)
```

Arguments

object	A DNEA object.
dat	A list of *m x n* numeric matrices of custom-normalized expression data, one matrix for each experimental condition. The list elements should be labeled for their respective condition. These should match the labels returned by networkGroups.
assay_name	A character string corresponding to the name the new data will be stored under in the assays slot of the DNEA.

Value

A DNEA object with the added expression data in the @assays slot

Author(s)

Christopher Patsalis

See Also

createDNEAobject,DNEA,

Examples

```
#load example data
#load example data
data(TEDDY)
data(T1Dmeta)

#make sure metadata and expression data are in same order
T1Dmeta <- T1Dmeta[colnames(TEDDY),]

#create group labels
group_labels <- T1Dmeta$group
names(group_labels) <- rownames(T1Dmeta)

#initiate DNEA object
dnw <- createDNEAobject(project_name = "test", expression_data = TEDDY,
                        group_labels = group_labels)

#transpose TEDDY data
TEDDY <- t(log(TEDDY))

#make sure metadata and expression data are in same order
T1Dmeta <- T1Dmeta[rownames(TEDDY),]

dat <- list()
for(cond in networkGroups(dnw)){
  dat[[cond]] <- TEDDY[names(group_labels)[group_labels == cond],]
}


#log-transform and median center the expression data without scaling
```

```
newdat <- list()
for(cond in networkGroups(dnw)){

  group_dat <- dat[[cond]]
  for(i in seq(1, ncol(group_dat))){
   metab_median=median(group_dat[, i], na.rm=TRUE)
    metab_range=range(group_dat[, i], na.rm=TRUE)
    scale_factor=max(abs(metab_range - metab_median))
    group_dat[, i] <- (group_dat[, i] - metab_median) / scale_factor

    rm(metab_median, metab_range, scale_factor)
  }

  group_dat <- t(group_dat)
  newdat <- append(newdat, list(group_dat))

  rm(i, group_dat)
}

#add names
names(newdat) <- names(dat)

#add data
dnw <- addExpressionData(object=dnw,
                         dat=newdat,
                         assay_name="median_scaled_data")
```

---

adjacencyGraph *Retrieve the adjacency graph for the case, control, or joint network*

Description

The function returns the adjacency graph made for the case, control, or joint network constructed via consensus clustering using clusterNet.

Usage

```
adjacencyGraph(x, graph)

## S4 method for signature 'DNEA'
adjacencyGraph(x, graph)

## S4 method for signature 'consensusClusteringResults'
adjacencyGraph(x, graph)
```

Arguments

x	A DNEA

graph	A character string indicating which of the adjacency graphs to return. Values can be "joint_graph" for the whole graph object, or one of the group values returned by networkGroups.

Value

An igraph graph object corresponding to the specified adjacency graph.

Author(s)

Christopher Patsalis

See Also

clusterNet

Examples

```
#dnw is a \code{\link{DNEA}} object with the results
#generated for the example data accessed by running
#data(TEDDY) in the console. The workflow for this data
#can be found in the vignette accessed by running
#browseVignettes("DNEA") in the console.
data("dnw")

adjacencyGraph(dnw, graph="DM:case")
```

---

adjacencyMatrix *Retrieve the weighted or unweighted adjacency matrix*

Description

The function takes as input a DNEA object and returns the weighted or un-weighted adjacency matrix for each group network constructed via the getNetworks function.

Usage

```
adjacencyMatrix(x, weighted)

## S4 method for signature 'DNEA'
adjacencyMatrix(x, weighted = FALSE)
```

Arguments

x	A DNEA object.
weighted	TRUE/FALSE indicating whether the weighted unweighted adjacency matrix should be returned.

Value

A matrix corresponding to the adjacency matrix specified.

Author(s)

Christopher Patsalis

See Also

getNetworks

Examples

```
#dnw is a \code{\link{DNEA}} object with the results
#generated for the example data accessed by running
#data(TEDDY) in the console. The workflow for this data
#can be found in the vignette accessed by running
#browseVignettes("DNEA") in the console.
data("dnw")

adjacencyMatrix(dnw, weighted=TRUE)
```

---

aggregateFeatures *Aggregate correlated features into a single feature class*

Description

This function takes as input a DNEA object and aggregates highly correlated features within the non-normalized, non-transformed data using one of three methods:

1. **correlation-based**
2. **knowledge-based**
3. **hybrid**

More info about the different approaches can be found in the ***Details*** section below. Highly correlated groups of features are aggregated by taking the mean expression of all features in the group, respectively.

NOTE: This method was developed using non-normalized, non-transformed data. Since the mean expression value is used for each group, normalized data may alter erroneously alter the aggregated expression values

Usage

```
aggregateFeatures(
  object,
  method = c("correlation", "knowledge", "hybrid"),
  correlation_threshold = NULL,
  feature_groups = NULL,
  assay = "input_data"
)
```

Arguments

object	A DNEA object.
method	A character string that dictates the collapsing method to use. The available methods are: "correlation", "knowledge", or "hybrid".
correlation_threshold	A threshold wherein features correlated above the supplied value are aggregated into one feature class. This parameter is only necessary for the correlation and hybrid methods.
feature_groups	A data frame containing group information for the algorithm indicated by the "knowledge" and "hybrid" methods.
assay	A character string indicating which expression assay to use for analysis. The default is the non-transformed input data that is stored as the"input_data" assay. It is highly recommended that this setting is used.

Details

Due to the computational complexity of the DNEA algorithm, the processing time for a given data set increases dramatically as the number of features increases. The ability to process each replicate performed in stabilitySelection in parallel helps circumvent this issue, however, an analysis may still be constrained by the resources available (i.e. a limited number of cpu cores or memory). Aggregating related features into a single feature class is another method by which the user can reduce the complexity of the analysis, and as a result decrease the necessary resources.

In a related scenario, you may also have many highly-correlated features of the same class of compounds (i.e. fatty acids, carnitines, etc.), and network analysis at the resolution of these individual features is not important. Aggregating features would decrease the computing time without losing critical information to the analysis (Please see the ***Details*** section of createDNEAobject for more information about the motivation behind aggregating highly correlated features).

Ultimately, this function allows the user to reduce the complexity of the data set and reduce the computational power necessary for the analysis and/or improve the quality of the results. The most appropriate method to use when aggregating data is dependent on the data set and prior information known about the features. The following text explains more about each method and the best use cases:

1. **correlation-based -** The user specifies a correlation threshold wherein features with a higher pearson correlation value than the threshold are aggregated into one group. This approach is useful when the user does not have any particular class definitions for the features.

2. **knowledge-based -** The user specifies feature classes based on *a priori* information (i.e. all of the carnitine's in a data set are specified as one class) and the features within a class are aggregated into one feature. This approach is best in experiments where the data set contains many highly similar compounds, like fatty acids, carnitines, ceramides, etc.
3. **hybrid -** The user specifies both a correlation threshold like in the correlation-based approach and feature classes based on *a priori* information similar to the knowledge-based approach. The features within each user-specified class that have a higher pearson correlation than the provided threshold are aggregated into one class This approach is best in experiments where the data set contains many compounds of a similar class, but the user is unsure how correlated the features of said class will be. This method prevents poorly correlated or uncorrelated features from being aggregated into a single feature.

Value

A `collapsed_DNEA` object.

Author(s)

Christopher Patsalis

See Also

`createDNEAobject`, `stabilitySelection`

Examples

```
#load example data
data(TEDDY)
data(T1Dmeta)

#make sure metadata and expression data are in same order
T1Dmeta <- T1Dmeta[colnames(TEDDY),]

#create group labels
group_labels <- T1Dmeta$group
names(group_labels) <- rownames(T1Dmeta)

#initiate DNEA object
dnw <- createDNEAobject(project_name = "test", expression_data = TEDDY,
                            group_labels = group_labels)

#simulate group labels
TEDDY_groups <- data.frame(features=rownames(expressionData(x=dnw,
                           assay="input_data")),
                           groups=rownames(expressionData(x=dnw,
                                          assay="input_data")),
                           row.names=rownames(expressionData(x=dnw,
                                             assay="input_data")))

TEDDY_groups$groups[TEDDY_groups$groups %in% c("isoleucine",
                                               "leucine",
```

```
                                                  "valine")] <- "BCAAs"
TEDDY_groups$groups[grep("acid", TEDDY_groups$groups)] <- "fatty_acids"


collapsed_TEDDY <- aggregateFeatures(object=dnw,
                                     method="hybrid",
                                     correlation_threshold=0.7,
                                     feature_groups=TEDDY_groups)
```

`BICscores` *Access the BIC scores for each lambda value evaluated*

Description

The function takes as input a DNEA object and returns the BIC values for each lambda tested during hyper parameter optimization performed via BICtune.

Usage

```
BICscores(x)

BICscores(x) <- value

## S4 method for signature 'DNEA'
BICscores(x)

## S4 replacement method for signature 'DNEA'
BICscores(x) <- value
```

Arguments

x	A DNEA object.
value	a list of two lists that consist of the likelihood and BIC scores for each tested lambda value.

Value

The optimized lambda hyperparameter.

Author(s)

Christopher Patsalis

See Also

BICtune

Examples

```
#dnw is a DNEA with the results generated for the example data
#accessed by running data(TEDDY) in the console. The workflow
#for this data can be found in the vignette accessed by
#running browseVignettes("DNEA") in the console.
data(dnw)

BICscores(dnw)
```

`BICtune` *Optimize the lambda regularization parameter for the glasso-based network models using Bayesian-information Criterion*

Description

This function will calculate the Bayesian information criterion (BIC) and likelihood for a range of lambda values that are automatically generated (*please see* **Details** *for more info*) or that are user-specified. The lambda value with the minimum BIC score is the optimal lambda value for the data set and is stored in the DNEA object for use in stability selection using `stabilitySelection`.

Usage

```
BICtune(
  object,
  lambda_values,
  interval = 0.001,
  informed = TRUE,
  assay,
  eps_threshold = 1e-06,
  eta_value = 0.1,
  BPPARAM = bpparam(),
  BPOPTIONS = bpoptions()
)

## S4 method for signature 'DNEA'
BICtune(
  object,
  lambda_values,
  interval = 0.001,
  informed = TRUE,
  assay,
  eps_threshold = 1e-06,
  eta_value = 0.1,
  BPPARAM = bpparam(),
  BPOPTIONS = bpoptions()
)
```

```
## S4 method for signature 'matrix'
BICtune(
  object,
  lambda_values,
  interval = 0.001,
  informed = TRUE,
  eps_threshold = 1e-06,
  eta_value = 0.1,
  BPPARAM = bpparam(),
  BPOPTIONS = bpoptions()
)
```

Arguments

object	A DNEA object. See createDNEAobject
lambda_values	**OPTIONAL -** A list of values to test while optimizing the lambda parameter. If not provided, a set of lambda values are chosen based on the theoretical value for the asymptotically valid lambda. More information about this can be found in the details section.
interval	A numeric value indicating the specificity by which to optimize lambda. The default value is 1e-3, which indicates lambda will be optimized to 3 decimal places. The value should be between 0 and 0.1.
informed	TRUE/FALSE indicating whether the asymptotic properties of lambda for large data sets should be utilized to tune the parameter. This reduces the necessary number of computations for optimization.
assay	A character string indicating which expression assay to use for analysis. The default is the "log_scaled_data" assay that is created during createDNEAobject.
eps_threshold	A significance cut-off for thresholding network edges. The default value is 1e-06. This value generally should not change.
eta_value	A tuning parameter that that ensures that the empirical covariance matrix of the data is positive definite so that we can calculate its inverse. The default value is 0.01.
BPPARAM	A BiocParallel object.
BPOPTIONS	a list of options for BiocParallel created using the bpoptions function.

Details

There are several ways to optimize the lambda parameter for a glasso model - We utilize Bayesian-information criterion (BIC) to optimize the lambda parameter in DNEA because it is a more balanced method and less computationally expensive. We can reduce the total number of values that need to be tested in optimization by carefully selecting values around the asymptotically valid lambda for data sets with many samples and many features following the equation:

$$\lambda = \sqrt{\ln(num.features)/num.samples}$$

For smaller data sets, the asymptotically valid lambda is described by modifying the previous equation to include an unknown constant, c, that needs to be determined mathematically. Therefore, to

optimize lambda we modify the previous equation as follows:

$$\lambda = c\sqrt{\ln(num.features)/num.samples}$$

where c takes on values between 0 and the theoretical maximum of C in intervals of 0.02. C is then estimated and a new range is tested to the specificity of the "interval" input. More information regarding the optimization method deployed here can be found in the Guo et al. (2011) paper referenced below.

Value

A DNEA object containing the BIC and likelihood scores for every lambda value tested, as well as the optimized lambda value

Author(s)

Christopher Patsalis

References

Guo J, Levina E, Michailidis G, Zhu J. Joint estimation of multiple graphical models. Biometrika. 2011 Mar;98(1):1-15. doi: 10.1093/biomet/asq060. Epub 2011 Feb 9. PMID: 23049124; PMCID: PMC3412604. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3412604/

See Also

optimizedLambda, bpparam, bpoptions glasso

Examples

```
#import BiocParallel package
library(BiocParallel)

#load example data
data(TEDDY)
data(T1Dmeta)

#make sure metadata and expression data are in same order
TEDDY <- TEDDY[seq(50), ]
T1Dmeta <- T1Dmeta[colnames(TEDDY),]

#create group labels
group_labels <- T1Dmeta$group
names(group_labels) <- rownames(T1Dmeta)

#initiate DNEA object
dnw <- createDNEAobject(project_name = "test", expression_data = TEDDY,
                            group_labels = group_labels)

#optimize lambda parameter
dnw <- BICtune(object=dnw,
```

```
                informed=TRUE,
                interval=0.01)
```

CCsummary *Retrieves the summary results of consensus clustering*

Description

The function takes as input a DNEA object and returns a summary of the results of consensus clustering stored in the consensus_clustering slot as a consensusClusteringResults object.

Usage

```
CCsummary(x)

## S4 method for signature 'DNEA'
CCsummary(x)
```

Arguments

x	A DNEA object.

Value

A data frame summary of the consensus clustering results from DNEA.

Author(s)

Christopher Patsalis

See Also

clusterNet

Examples

```
#dnw is a \code{\link{DNEA}} object with the results
#generated for the example data accessed by running
#data(TEDDY) in the console. The workflow for this data
#can be found in the vignette accessed by running
#browseVignettes("DNEA") in the console.
data("dnw")

CCsummary(dnw)
```

clusterNet *Identify metabolic modules within the biological networks using a consensus clustering approach*

Description

This function clusters the jointly estimated adjacency matrices constructed using `getNetworks` via the consensus clustering approach described in Ma et al. (*Please see the **Details** section for more information*) to identify metabolic modules, aka sub networks, present in the larger networks. Only sub networks with consensus that meets or exceeds tau are identified as real.

Usage

```
clusterNet(object, tau = 0.5, max_iterations = 5, verbose = TRUE)
```

Arguments

`object`	A `DNEA` object.
`tau`	The % agreement among the clustering algorithms for a node to be included in a sub network.
`max_iterations`	The maximum number of replicates of consensus clustering to be performed if consensus is not reached.
`verbose`	TRUE/FALSE whether a progress bar should be displayed in the console.

Details

Seven clustering algorithms from the `igraph` package are utilized in this consensus clustering approach:

1. `cluster_edge_betweenness`
2. `cluster_fast_greedy`
3. `cluster_infomap`
4. `cluster_label_prop`
5. `cluster_louvain`
6. `cluster_walktrap`
7. `cluster_leading_eigen`

For each iteration, node membership in a respective cluster is compared across the algorithms, and only the nodes with tau % agreement for a given cluster are kept. A new adjacency graph is then created and clustering is performed again. This occurs iteratively until consensus on is reached stable sub networks or the specified "max_iterations" is reached *(Please see references for more details)*.

Value

A DNEA object containing sub network determinations for the nodes within the input network. A summary of the consensus clustering results can be viewed using CCsummary. Sub network membership for each node can be found in the "membership" column of the node list, which can be accessed using nodeList.

Author(s)

Christopher Patsalis

References

Ma J, Karnovsky A, Afshinnia F, Wigginton J, Rader DJ, Natarajan L, Sharma K, Porter AC, Rahman M, He J, Hamm L, Shafi T, Gipson D, Gadegbeku C, Feldman H, Michailidis G, Pennathur S. Differential network enrichment analysis reveals novel lipid pathways in chronic kidney disease. Bioinformatics. 2019 Sep 15;35(18):3441-3452. doi: 10.1093/bioinformatics/btz114. PMID: 30887029; PMCID: PMC6748777. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6748777/

See Also

plotNetworks

Examples

```
#dnw is a \code{\link[=DNEA-class]{DNEA}} object with the results
#generated for the example data accessed by running
#data(TEDDY) in the console. The workflow for this data
#can be found in the vignette accessed by running
#browseVignettes("DNEA") in the console.
data(dnw)

#identify metabolic modules via consensus clustering
dnw <- clusterNet(object=dnw, tau=0.5, max_iterations=5)

#we can also plot the subnetworks
plotNetworks(object=dnw, type="sub_networks", subtype=1)
```

---

collapsed_DNEA-class *collapsed_DNEA*

Description

An s4 class to represent the DNEA workflow, including collapsing features. This class inherits from the DNEA class.

Value

A collapsed_DNEA object

Slots

`original_experiment` The DNEA object input to aggregateFeatures.

`feature_membership` A data frame containing all of the features from the original input data and their corresponding group membership in the new aggregated data.

Author(s)

Christopher Patsalis

See Also

aggregateFeatures, createDNEAobject

Examples

```
#load example data
data(TEDDY)
data(T1Dmeta)

#make sure metadata and expression data are in same order
T1Dmeta <- T1Dmeta[colnames(TEDDY),]

#create group labels
group_labels <- T1Dmeta$group
names(group_labels) <- rownames(T1Dmeta)

#initiate DNEA object
dnw <- createDNEAobject(project_name = "test", expression_data = TEDDY,
                            group_labels = group_labels)

#simulate group labels
TEDDY_groups <- data.frame(features=rownames(expressionData(x=dnw,
                           assay="input_data")),
                           groups=rownames(expressionData(x=dnw,
                                          assay="input_data")),
                           row.names=rownames(expressionData(x=dnw,
                                             assay="input_data")))

TEDDY_groups$groups[TEDDY_groups$groups %in% c("isoleucine",
                                               "leucine",
                                               "valine")] <- "BCAAs"
TEDDY_groups$groups[grep("acid", TEDDY_groups$groups)] <- "fatty_acids"


collapsed_TEDDY <- aggregateFeatures(object=dnw,
                                     method="hybrid",
                                     correlation_threshold=0.7,
```

```
                                    feature_groups=TEDDY_groups)
```

`consensusClusteringResults-class` *consensusClusteringResults*

Description

An s4 class to represent the results from consensus clustering within DNEA

Value

A consensusClusteringResults object

Slots

`summary` a data frame containing the sub networks as rows and summary information as columns. The columns include: number_of_nodes, number_of_edges, number_of_DE_nodes, and number_of_DE_edges.

`subnetwork_membership` A data frame with the same number of rows as features in the data, and a column indicating which sub network a given feature belongs to, if any.

`adjacency_graph` The resulting adjacency graph from `igraph` created after consensus clustering.

Author(s)

Christopher Patsalis

See Also

`clusterNet`

`createDNEAobject` *Initialize a DNEA object*

Description

This function takes as input a matrix of non-normalized, non-transformed expression data and the case/control group labels in order to initiate a DNEA object. Differential expression analysis is performed using a student's T-test and Benjamini-Hochberg for multiple-testing corrections. Diagnostic testing is done on the input data by checking the minimum eigen value and condition number of the expression data for each experimental condition. To initialize a *DNEA* from a `SummarizedExperiment-class`, or a mass_dataset-class from the massdataset package, please see the `sumExp2DNEA` and `massDataset2DNEA` documentation, respectively.

IMPORTANT::
Special attention should be given to the diagnostic criteria that is output. The minimum eigen value and condition number are calculated for the whole data set as well as for each condition to determine mathematic stability of the data set and subsequent results from a GGM model. More information about interpretation can be found in the ***Details*** section below.

Usage

```
createDNEAobject(
  project_name,
  expression_data,
  scaled_expression_data,
  group_labels,
  assay
)
```

Arguments

`project_name`	A character string name for the experiment.
`expression_data`	A numeric *m x n* matrix or data frame of un-transformed, un-scaled expression data. The sample names should be column names and the feature names should be row names.
`scaled_expression_data`	A list of numeric *m x n* matrices or data frames of transformed and/or scaled expression data. The sample names should be column names and the feature names should be row names. Each set of expression data should be aproximately normal.
`group_labels`	A factor vector of experimental group labels named with the corresponding sample name.
`assay`	A character string indicating which assay to use for diagnostics and differential expression analysis NOTE: The function always defaults to using log transformed data for differential expression analysis if provided.

Details

Diagnostics Motivation:
Negative or zero eigenvalues in a data set can represent instability in that portion of the matrix, thereby invalidating parametric statistical methods and creating unreliable results. In this function, the minimum eigenvalue of the data set is calculated by first creating a pearson correlation matrix of the data. Instability may then occur for a number of reasons, but one common cause is highly correlated features (in the positive and negative direction).

Regularization often takes care of this problem by arbitrarily selecting one of the variables in a highly correlated group and removing the rest. We have developed DNEA to be very robust in situations where ***p » n*** by optimizing the model via several regularization steps (*please see* `BICtune` *and* `stabilitySelection`) that may handle such problems without intervention, however, the user can also pre-emptively collapse highly-correlated features into a single group via `aggregateFeatures`.

Benefits of Feature Aggregation:

When your dataset contains highly correlated features, we recommend aggregating features into related groups - such as highly-correlated features of a given class of molecules (ie. many fatty acids, carnitines, etc.) - because the user then has more control over which variables are included in the model. Without collapsing, the model regularization may result in one of the features within a class being included and some or all of the remaining features being removed. By collapsing first, you retain the signal from all of the features in the collapsed group and also have information pertaining to which features are highly correlated and will therefore have similar feature-feature associations.

Value

A DNEA object.

Author(s)

Christopher Patsalis

See Also

BICtune, stabilitySelection, sumExp2DNEA, massDataset2DNEA

Examples

```
#import example data
data(TEDDY)
data(T1Dmeta)

#create group labels
group_labels <- factor(T1Dmeta$group,
                       levels=c("DM:control", "DM:case"))
names(group_labels) <- rownames(T1Dmeta)


#initiate DNEA object
DNEA <- createDNEAobject(expression_data=TEDDY,
                         project_name="TEDDYmetabolomics",
                         group_labels=group_labels)
```

---

datasetSummary *Access the dataset_summary slot of a DNEA object*

Description

This function prints to console the number of samples, number of features, and diagnostic values of the input data stored in the dataset_summary slot of the DNEA.

Usage

```
datasetSummary(x)

## S4 method for signature 'DNEA'
datasetSummary(x)
```

Arguments

x	A DNEA object.

Value

The numbers of samples/features and diagnostic values of the input data stored in the dataset_summary slot of the DNEA.

Author(s)

Christopher Patsalis

See Also

createDNEAobject,aggregateFeatures

Examples

```
#load example data
data(TEDDY)
data(T1Dmeta)

#make sure metadata and expression data are in same order
T1Dmeta <- T1Dmeta[colnames(TEDDY),]

#create group labels
group_labels <- T1Dmeta$group
names(group_labels) <- rownames(T1Dmeta)

#initiate DNEA object
dnw <- createDNEAobject(project_name = "test", expression_data = TEDDY,
                            group_labels = group_labels)

datasetSummary(dnw)
```

---

diagnostics *Retrieve the diagnostic values for the input expression data*

Description

This function retrieves the diagnostic values calculated for the input expression data by the createDNEAobject function.

Usage

```
diagnostics(x)

## S4 method for signature 'DNEA'
diagnostics(x)

## S4 method for signature 'DNEAinputSummary'
diagnostics(x)
```

Arguments

x	DNEA or DNEAinputSummary object.

Value

Returns the diagnostic values for the input expression data.

Author(s)

Christopher Patsalis

See Also

createDNEAobject, aggregateFeatures

Examples

```
#load example data
data(TEDDY)
data(T1Dmeta)

#make sure metadata and expression data are in same order
T1Dmeta <- T1Dmeta[colnames(TEDDY),]

#create group labels
group_labels <- T1Dmeta$group
names(group_labels) <- rownames(T1Dmeta)

#initiate DNEA object
dnw <- createDNEAobject(project_name = "test", expression_data = TEDDY,
                            group_labels = group_labels)

diagnostics(dnw)
```

DNEA-class *DNEA object*

Description

An s4 class to represent the DNEA workflow

Usage

```
## S4 method for signature 'DNEA'
show(object)
```

Arguments

object	A DNEA object.

Value

A DNEA object

A summary of the information stored in a DNEA object.

Functions

- show(DNEA): This function will display a summary of the information stored within a DNEA object.

Slots

project_name A character string name for the experiment.

assays A list of matrices, "input_data" being the original non-normalized, non-transformed data, "log_input_data" is the input data log transformed, and "log-scaled_input" is the input data log-transformed and auto-scaled. The row names between the input assays must be identical (the expression data can be accessed via the expressionData function). Any other assay input into the DNEA object can be accessed by supplying its name to the assay parameter.

metadata A list of information about the data, including a data frame for sample metadata (the row names must match the sample order of the stored expression data), a data frame for feature metadata (the row names must match the feature order of the stored expression data), a two-level factor corresponding to the two groups in the data, and a character vector the same length as the number of samples corresponding to the group membership for each sample (the user may add additional metadata via the includeMetadata function).

dataset_summary A DNEAinputSummary object (can view data via datasetSummary and diagnostics)

node_list A data frame containing all of the features in the data set as rows as well as the differential expression analysis results (can view the node list via nodeList).

edge_list A data frame containing the network edges identified via getNetworks (can view the edge list via edgeList).

`hyperparameter` A list of results obtained from `BICtune` containing a numeric vector of the lambda values tested during optimization, the resulting Bayesian-information criterion and likelihood scores for each lambda value, and the optimized lambda for analysis (the optimized lambda can be accessed or changed via the `optimizedLambda` function).

`adjacency_matrix` A list of adjacency matrices, one for each experimental condition, jointly estimated via `getNetworks`.

`stable_networks` A list of the selection results and selection probabilities, one for each experimental condition, for every possible feature-feature edge.

`consensus_clustering` A `consensusClusteringResults` object containing the results from consensus clustering obtained via the `clusterNet` function.

`netGSA` a data frame containing the results from enrichment analysis performed via `runNetGSA` and the `NetGSA` algorithm. Each row is the results for a given sub network tested for enrichment.

Author(s)

Christopher Patsalis

See Also

`expressionData`,`includeMetadata`, `nodeList`,`edgeList`, `datasetSummary`,`diagnostics`, `BICtune`,`getNetworks`, `stabilitySelection`,`clusterNet`, `runNetGSA`,`selectionProbabilities`, `selectionResults`,`createDNEAobject`,

`createDNEAobject`, `aggregateFeatures`

Examples

```
#load example data
data(TEDDY)
data(T1Dmeta)

#make sure metadata and expression data are in same order
T1Dmeta <- T1Dmeta[colnames(TEDDY),]

#create group labels
group_labels <- T1Dmeta$group
names(group_labels) <- rownames(T1Dmeta)

#initiate DNEA object
dnw <- createDNEAobject(project_name = "test", expression_data = TEDDY,
                            group_labels = group_labels)

dnw
```

DNEAinputSummary-class *DNEAinputSummary*

Description

An s4 class to represent the results from diagnostic testing on the input data to a DNEA.

Usage

```
## S4 method for signature 'DNEAinputSummary'
show(object)
```

Arguments

object	A DNEAinputSummary object

Value

A DNEAinputSummary object

A summary of the input data to createDNEAobject.

Functions

- show(DNEAinputSummary): This function will display the number of samples, number of features, and diagnostics values of the input data set to a DNEA object.

Slots

num_samples a single-value numeric vector corresponding to the number of samples in the data set.

num_features a single-value numeric vector corresponding to the number of features in the data set

diagnostic_values a 3x3 data frame with the diagnostic values calculated via createDNEAobject.

Author(s)

Christopher Patsalis

See Also

createDNEAobject

createDNEAobject, aggregateFeatures

Examples

```
#load example data
data(TEDDY)
data(T1Dmeta)

#make sure metadata and expression data are in same order
T1Dmeta <- T1Dmeta[colnames(TEDDY),]

#create group labels
group_labels <- T1Dmeta$group
names(group_labels) <- rownames(T1Dmeta)

#initiate DNEA object
dnw <- createDNEAobject(project_name = "test", expression_data = TEDDY,
                            group_labels = group_labels)

datasetSummary(dnw)
```

---

dnw *Example results for DNEA*

Description

"dnw" is a DNEA object containing the results for the full DNEA workflow on the TEDDY example data. The exact workflow to produce these results can be replicated by following the package vignette accessed by entering browseVignettes("DNEA") in the console. 1000 replicates were performed during stability selection *with* the subsampling protocol. The lambda value used during joint estimation was aproximated as

$$\lambda = \sqrt{\ln(num.features)/num.samples}$$

Usage

```
data("dnw")
```

Format

A DNEA results object after completing a DNEA experiment.

Value

A DNEA object containing the results of a DNEA experiment.

Source

The data the results of the full DNEA workflow performed using the TEDDY example data, as described above.

edgeList *Access the edge list*

Description

The function takes as input a DNEA object and returns the edge list created by the getNetworks function.

Usage

```
edgeList(x)

edgeList(x) <- value

## S4 method for signature 'DNEA'
edgeList(x)

## S4 replacement method for signature 'DNEA'
edgeList(x) <- value
```

Arguments

x	a DNEA object.
value	a data frame of edges in the network.

Value

A data frame corresponding to the edge list determined by DNEA.

Author(s)

Christopher Patsalis

See Also

getNetworks,filterNetworks, getNetworkFiles

Examples

```
#dnw is a \code{\link{DNEA}} object with the results
#generated for the example data accessed by running
#data(TEDDY) in the console. The workflow for this data
#can be found in the vignette accessed by running
#browseVignettes("DNEA") in the console.
data("dnw")

edgeList(dnw)
```

expressionData *Access expression data within a DNEA object,*

Description

This function accesses the expression data stored in the assays slot of the DNEA object. The output is an *n x m* matrix with one row for each sample and one column for each feature in the data.

Usage

```
expressionData(x, assay)

## S4 method for signature 'DNEA'
expressionData(x, assay = names(assays(x)))
```

Arguments

x	A DNEA object.
assay	A character string corresponding to the data to retrieve: "input_data" retrieves the data as it was input, "log_input_data" retrieves the input data after log transforming, and "log_scaled_data" retrieves a list of matrices corresponding to the log-scaled data for each experimental condition, respectively. Any other externally transformed data that is stored in the DNEA object can be accessed by providing its name to the assay parameter.

Value

The expression matrix specified by the user.

Author(s)

Christopher Patsalis

See Also

createDNEAobject,aggregateFeatures

Examples

```
#load example data
data(TEDDY)
data(T1Dmeta)

#make sure metadata and expression data are in same order
T1Dmeta <- T1Dmeta[colnames(TEDDY),]

#create group labels
group_labels <- T1Dmeta$group
```

```
names(group_labels) <- rownames(T1Dmeta)

#initiate DNEA object
dnw <- createDNEAobject(project_name = "test", expression_data = TEDDY,
                            group_labels = group_labels)

expressionData(x=dnw, assay="input_data")
expressionData(x=dnw, assay="log_input_data")
expressionData(x=dnw, assay="log_scaled_data")
```

`featureNames` *Retrieve the feature names from the metadata slot.*

Description

This function accesses the feature names stored in the metadata slot of the DNEA object.

Usage

```
featureNames(x, original = FALSE)

## S4 method for signature 'DNEA'
featureNames(x, original = FALSE)
```

Arguments

x	A DNEA object.
original	"TRUE" returns the original feature names and "FALSE" returns the feature names that have been modified to avoid errors as a result of special characters using make_clean_names.

Value

A character vector of feature names.

Author(s)

Christopher Patsalis

See Also

createDNEAobject

Examples

```
#load example data
data(TEDDY)
data(T1Dmeta)

#make sure metadata and expression data are in same order
T1Dmeta <- T1Dmeta[colnames(TEDDY),]

#create group labels
group_labels <- T1Dmeta$group
names(group_labels) <- rownames(T1Dmeta)

#initiate DNEA object
dnw <- createDNEAobject(project_name = "test", expression_data = TEDDY,
                            group_labels = group_labels)

featureNames(dnw, original=TRUE)
```

`filterNetworks` *Filter the adjacency matrices to only the edges that meet the filter conditions*

Description

This function takes as input a DNEA object and allows the user to filter the network edges by one of two methods:

1. **Partial Correlation** - The networks can be filtered to only include edges greater than or equal to a specified partial correlation (pcor) value.
2. **Top *X*% of edges** - The networks can be filtered to only include the strongest *X*% of edges determined by their partial correlation values.

Filtering is performed on the case and control adjacency matrices separately.

Usage

```
filterNetworks(data, pcor, top_percent_edges)

## S4 method for signature 'DNEA'
filterNetworks(data, pcor, top_percent_edges)

## S4 method for signature 'list'
filterNetworks(data, pcor, top_percent_edges)
```

Arguments

data	A DNEA object.
pcor	A partial correlation value of which to threshold the adjacency matrices. Edges with pcor values <= to this value will be removed.
top_percent_edges	A value between 0-1 that corresponds to the top x% edges to keep in each network, respectively (i.e. top_percent_edges = 0.1 will keep only the top 10% strongest edges in the networks).

Value

The input object after filtering the egdes in the network according to the specified parameters.

Author(s)

Christopher Patsalis

See Also

getNetworks,adjacencyMatrix

Examples

```
#dnw is a \code{\link[=DNEA-class]{DNEA}} object with the results
#generated for the example data accessed by running
#data(TEDDY) in the console. The workflow for this data
#can be found in the vignette accessed by running
#browseVignettes("DNEA") in the console.
data(dnw)

#filter the networks by a correlation threshold of 0.166
dnw <- filterNetworks(dnw, pcor=0.166)

#filter networks for the top 40% strongest correlations
dnw <- filterNetworks(dnw, top_percent_edges=0.4)
```

---

getNetworkFiles *Save network information to .csv files*

Description

This function will save the node and edge information as .csv files to the specified directory. The files are already formatted for input into Cytoscape.

Usage

```
getNetworkFiles(object, file_path = getwd())
```

Arguments

object	A DNEA object.
file_path	The file path to save the node and edge lists to. If **NULL**, the files will be saved to the working directory.

Value

Two .csv files, one for the node list and one for the edge list, saved to the specified file path

Author(s)

Christopher Patsalis

See Also

edgeList,nodeList

Examples

```
#dnw is a \code{\link[=DNEA-class]{DNEA}} object with the results
#generated for the example data accessed by running
#data(TEDDY) in the console. The workflow for this data
#can be found in the vignette accessed by running
#browseVignettes("DNEA") in the console.
data(dnw)

#filepath wherein to save the networks files
filepath <- tempdir()
#save node and edge list for input to cytoscape
getNetworkFiles(dnw, file_path=filepath)
```

---

getNetworks *Construct the GLASSO-based biological Networks*

Description

This function constructs a biological network for each experimental condition using the joint estimation method described in Ma et al. (2019) (*please see references below*). If stabilitySelection was performed previously, the selection probabilities will be used to for model optimization when constructing the networks (please see the ***Details*** section of stabilitySelection for more information).

Usage

```
getNetworks(
  object,
  lambda_values,
  aprox = FALSE,
  informed = TRUE,
  interval = 0.001,
  assay,
  eps_threshold = 1e-06,
  eta_value = 0.1,
  optimal_lambdas,
  BPPARAM = bpparam(),
  BPOPTIONS = bpoptions()
)
```

Arguments

object	A DNEA object.
lambda_values	**OPTIONAL** A list of values to test while optimizing the lambda parameter. If not provided, a set of lambda values are chosen based on the theoretical value for the asymptotically valid lambda. More information about this can be found in the details section of BICtune.
aprox	TRUE/FALSE indicating whether BICtune should be used to optimize the lambda value for each condition. If aprox==FALSE, sqrt(log(# features)/#samples) is used to approximate lambda.
informed	TRUE/FALSE indicating whether the asymptotic properties of lambda for large data sets should be utilized to tune the parameter. This reduces the necessary number of computations for optimization.
interval	A numeric value indicating the specifity by which to optimize lambda. The default value is 1e-3, which indicates lambda will be optimized to 3 decimal places. The value should be between 0 and 0.1.
assay	A character string indicating which expression assay to use for analysis. The default is the "log_scaled_data" assay that is created during createDNEAobject.
eps_threshold	A numeric value between 0 and 1 by which to threshold the partial correlation values for edge identification. Edges with an absolute partial correlation value below this threshold will be zero'd out from the adjacency matrix.
eta_value	A tuning parameter that that ensures that the empirical covariance matrix of the data is positive definite so that we can calculate its inverse. The default value is 0.01.
optimal_lambdas	**OPTIONAL -** The lambda value to be used in analysis. If not provided, the lambda value is determined based on the input of the "aprox" parameter.
BPPARAM	A BiocParallel object.
BPOPTIONS	a list of options for BiocParallel created using the bpoptions function.

Value

A DNEA object after populating the adjaceny_matrix and edge_list slots with the corresponding adjacency_matrix for each sample condition as well as the network edge list.

Author(s)

Christopher Patsalis

References

Ma J, Karnovsky A, Afshinnia F, Wigginton J, Rader DJ, Natarajan L, Sharma K, Porter AC, Rahman M, He J, Hamm L, Shafi T, Gipson D, Gadegbeku C, Feldman H, Michailidis G, Pennathur S. Differential network enrichment analysis reveals novel lipid pathways in chronic kidney disease. Bioinformatics. 2019 Sep 15;35(18):3441-3452. doi: 10.1093/bioinformatics/btz114. PMID: 30887029; PMCID: PMC6748777. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6748777/

Iyer GR, Wigginton J, Duren W, LaBarre JL, Brandenburg M, Burant C, Michailidis G, Karnovsky A. Application of Differential Network Enrichment Analysis for Deciphering Metabolic Alterations. Metabolites. 2020 Nov 24;10(12):479. doi: 10.3390/metabo10120479. PMID: 33255384; PMCID: PMC7761243. https://pubmed.ncbi.nlm.nih.gov/33255384/

See Also

edgeList,adjacencyMatrix

Examples

```
#dnw is a \code{\link[=DNEA-class]{DNEA}} object with the results
#generated for the example data accessed by running
#data(TEDDY) in the console. The workflow for this data
#can be found in the vignette accessed by running
#browseVignettes("DNEA") in the console.
data(dnw)

#construct the networks
dnw <- getNetworks(object=dnw, aprox = TRUE)

#now we can plot the group networks
plotNetworks(object=dnw, type="group_networks")
```

---

includeMetadata *Add additional metadata to the DNEA object*

Description

This function will take additional metadata and add it to the specified data frame in the metadata slot. ***NOTE:*** The row names of the new metadata must match the order of the input sample names or feature names, respectively.

Usage

```
includeMetadata(object, type = c("samples", "features"), metadata)
```

Arguments

object	A DNEA object.
type	A character string corresponding to the type of metadata being included. Can be either "samples" or "features"
metadata	a data frame containing metadata to add. The row names should be either the sample names or feature names, respectively

Value

A DNEA object with the specified additions.

Author(s)

Christopher Patsalis

See Also

featureNames,sampleNames, metaData

Examples

```
#load example data
data(TEDDY)
data(T1Dmeta)

#make sure metadata and expression data are in same order
T1Dmeta <- T1Dmeta[colnames(TEDDY),]

#create group labels
group_labels <- T1Dmeta$group
names(group_labels) <- rownames(T1Dmeta)

#initiate DNEA object
dnw <- createDNEAobject(project_name = "test", expression_data = TEDDY,
                            group_labels = group_labels)

#make sure metadata has same sample order as DNEA object
T1Dmeta <- T1Dmeta[sampleNames(dnw), ]

#add new metadata to DNEA object
dnw <- includeMetadata(object=dnw, type="samples", metadata=T1Dmeta)
```

lambdas2Test *Access the lambda values tested during hyper parameter optimization*

Description

The function takes as input a DNEA object and returns the lambda values that were testing during hyper parameter optimization performed via BICtune.

Usage

```
lambdas2Test(x)

## S4 method for signature 'DNEA'
lambdas2Test(x)
```

Arguments

x	A DNEA or collapsed_DNEA object.

Value

The lambda values to evaluate in optimization.

Author(s)

Christopher Patsalis

See Also

BICtune

Examples

```
#load example data
data(TEDDY)
data(T1Dmeta)

#make sure metadata and expression data are in same order
T1Dmeta <- T1Dmeta[colnames(TEDDY),]

#create group labels
group_labels <- T1Dmeta$group
names(group_labels) <- rownames(T1Dmeta)

#initiate DNEA object
dnw <- createDNEAobject(project_name = "test", expression_data = TEDDY,
                            group_labels = group_labels)

lambdas2Test(dnw)
```

massDataset2DNEA *Initialize a DNEA object from a mass_dataset object*

Description

This function takes as input a mass_dataset-class object from the massdataset package to initiate a DNEA object. Differential expression analysis is performed using a student's T-test and Benjamini-Hochberg for multiple-testing corrections. Diagnostic testing is done on the input data by checking the minimum eigen value and condition number of the expression data for each experimental condition. **NOTE: the massdataset package from the tidymass software suite must be installed to use this function. Please see https://massdataset.tidymass.org/ for more installation instructions**

IMPORTANT::

Special attention should be given to the diagnostic criteria that is output. The minimum eigen value and condition number are calculated for the whole data set as well as for each condition to determine mathematic stability of the data set and subsequent results from a GGM model. More information about interpretation can be found in the ***Details*** section below.

Usage

```
massDataset2DNEA(project_name, object, group_label_col, scaled_input = FALSE)
```

Arguments

project_name	A character string name for the experiment.
object	a mass_dataset object.
group_label_col	A character string corresponding to the column in the sample metadata stored in the mass_dataset object to use as the group labels.
scaled_input	A TRUE/FALSE indicating whether the input data is already normalized

Details

Diagnostics Motivation:

Negative or zero eigenvalues in a data set can represent instability in that portion of the matrix, thereby invalidating parametric statistical methods and creating unreliable results. In this function, the minimum eigenvalue of the data set is calculated by first creating a pearson correlation matrix of the data. Instability may then occur for a number of reasons, but one common cause is highly correlated features (in the positive and negative direction).

Regularization often takes care of this problem by arbitrarily selecting one of the variables in a highly correlated group and removing the rest. We have developed DNEA to be very robust in situations where $\boldsymbol{p \gg n}$ by optimizing the model via several regularization steps (*please see* BICtune *and* stabilitySelection) that may handle such problems without intervention, however, the user can also pre-emptively collapse highly-correlated features into a single group via aggregateFeatures.

Benefits of Feature Aggregation:

When your dataset contains highly correlated features, we recommend aggregating features into related groups - such as highly-correlated features of a given class of molecules (ie. many fatty acids, carnitines, etc.) - because the user then has more control over which variables are included in the model. Without collapsing, the model regularization may result in one of the features within a class being included and some or all of the remaining features being removed. By collapsing first, you retain the signal from all of the features in the collapsed group and also have information pertaining to which features are highly correlated and will therefore have similar feature-feature associations.

Value

A DNEA object.

Author(s)

Christopher Patsalis

See Also

BICtune, stabilitySelection, createDNEAobject

Examples

```
#load data
data(TEDDY)
data(T1Dmeta)
data(metab_data)

#make sure metadata and expression data are in same order
T1Dmeta <- T1Dmeta[colnames(TEDDY),]
T1Dmeta <- T1Dmeta[, c(6,7,7)]
colnames(T1Dmeta) <- c("sample_id", "group", "class")

metab_data <- metab_data[rownames(TEDDY), ]

sample_info_note = data.frame(name = c("sample_id", "group", "class"),
                              meaning = c("sample", "group", "class"))
variable_info_note = data.frame(name = c("variable_id", "mz", "rt"),
                                meaning = c("variable_id", "mz", "rt"))
if (require(massdataset)) {
#create mass_dataset object from TEDDY
object <- massdataset::create_mass_dataset(expression_data = data.frame(TEDDY),
                                           sample_info = T1Dmeta,
                                           variable_info = metab_data,
                                           sample_info_note = sample_info_note,
                                           variable_info_note = variable_info_note)

DNEA <- massDataset2DNEA(project_name = "mass_dataset",
                             object = object,
                             group_label_col = "group")
}
```

metab_data *Feature meta data for the The Environmental Determinants of Diabetes in the Young (TEDDY) clinical trial*

Description

This is a data frame containing metadata for the metabolites in the corresponding TEDDY example data from "The Environmental Determinants of Diabetes in the Young" clinical trial.

Usage

```
data("metab_data")
```

Format

A data frame with 134 rows and 3 columns. Each row corresponds to a metabolite, and each column corresponds to:

variable_id The metabolite name

mz The mass/charge ratio for a given metabolite

rt The retention time for a given metabolite

Value

A data frame containing the metabolite metadata for the TEDDY metabolomics study

Source

The raw data can be downloaded from the Metabolomics workbench under study ID **ST001386**: https://www.metabolomicsworkbench.org/data/DRCCStudySummary.php?Mode=SetupRawDataDownload&StudyID=ST001386

References

Lee HS, Burkhardt BR, McLeod W, Smith S, Eberhard C, Lynch K, Hadley D, Rewers M, Simell O, She JX, Hagopian B, Lernmark A, Akolkar B, Ziegler AG, Krischer JP; TEDDY study group. Biomarker discovery study design for type 1 diabetes in The Environmental Determinants of Diabetes in the Young (TEDDY) study. Diabetes Metab Res Rev. 2014 Jul;30(5):424-34. doi: 10.1002/dmrr.2510. PMID: 24339168; PMCID: PMC4058423. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4058423/

metaData *Retrieve metadata stored in a DNEA*

Description

This function retrieves the specified metadata stored in the metadata slot of the DNEA object.

Usage

```
metaData(x, type)

## S4 method for signature 'DNEA'
metaData(x, type = c("samples", "features"))
```

Arguments

x	A DNEA object.
type	A character string indicating the type of metadata to access. Can be "sample" or "feature".

Value

A data frame of the indicated metadata

Author(s)

Christopher Patsalis

See Also

includeMetadata

createDNEAobject, includeMetadata

includeMetadata

Examples

```
#load example data
data(TEDDY)
data(T1Dmeta)

#make sure metadata and expression data are in same order
T1Dmeta <- T1Dmeta[colnames(TEDDY),]

#create group labels
group_labels <- T1Dmeta$group
names(group_labels) <- rownames(T1Dmeta)

#initiate DNEA
```

```
dnw <- createDNEAobject(project_name = "test", expression_data = TEDDY,
                            group_labels = group_labels)

metaData(dnw, type = "sample")
```

`netGSAresults` *Access the netGSA slot of a DNEA object*

Description

The function takes as input a DNEA object and returns a summary of the enrichment analysis results stored in the netGSA slot.

Usage

```
netGSAresults(x)

## S4 method for signature 'DNEA'
netGSAresults(x)
```

Arguments

x	A DNEA object.

Value

A data frame of the results from runNetGSA.

Author(s)

Christopher Patsalis

See Also

runNetGSA, netgsa::NetGSA()

Examples

```
#dnw is a \code{\link{DNEA}} object with the results
#generated for the example data accessed by running
#data(TEDDY) in the console. The workflow for this data
#can be found in the vignette accessed by running
#browseVignettes("DNEA") in the console.
data("dnw")

netGSAresults(dnw)
```

networkGroupIDs *Access and set the experimental group labels*

Description

This function accesses the experimental group labels for each sample stored in the metadata slot of a DNEA object.

Usage

```
networkGroupIDs(x)

networkGroupIDs(x) <- value

## S4 method for signature 'DNEA'
networkGroupIDs(x)
```

Arguments

x	A DNEA object.
value	a character string name corresponding to a column name of the sample metadata data frame.

Value

A vector of the unique condition labels.

Author(s)

Christopher Patsalis

See Also

includeMetadata

Examples

```
#load example data
data(TEDDY)
data(T1Dmeta)

#make sure metadata and expression data are in same order
T1Dmeta <- T1Dmeta[colnames(TEDDY),]

#create group labels
group_labels <- T1Dmeta$group
names(group_labels) <- rownames(T1Dmeta)
```

```
#initiate DNEA object
dnw <- createDNEAobject(project_name = "test", expression_data = TEDDY,
                            group_labels = group_labels)

networkGroupIDs(dnw)
```

---

networkGroups *Retrieve the unique group values of the experimental condition*

Description

This function takes in a DNEA object and returns the unique group labels of the experimental condition in the data set.

Usage

```
networkGroups(x)

## S4 method for signature 'DNEA'
networkGroups(x)
```

Arguments

x	A DNEA or collapsed_DNEA

Value

A vector of the condition values.

Author(s)

Christopher Patsalis

See Also

networkGroupIDs, createDNEAobject

Examples

```
#load example data
data(TEDDY)
data(T1Dmeta)

#make sure metadata and expression data are in same order
T1Dmeta <- T1Dmeta[colnames(TEDDY),]

#create group labels
group_labels <- T1Dmeta$group
names(group_labels) <- rownames(T1Dmeta)
```

```
#initiate DNEA object
dnw <- createDNEAobject(project_name = "test", expression_data = TEDDY,
                            group_labels = group_labels)

networkGroups(dnw)
```

---

nodeList *Access the node list*

Description

The function takes as input a DNEA object and returns the node list created from the createDNEAobject function.

Usage

```
nodeList(x)

nodeList(x) <- value

## S4 method for signature 'DNEA'
nodeList(x)

## S4 replacement method for signature 'DNEA'
nodeList(x) <- value
```

Arguments

x	a DNEA object.
value	a data frame of nodes in the network.

Value

A data frame corresponding to the node list determined by DNEA.

Author(s)

Christopher Patsalis

See Also

createDNEAobject,clusterNet, getNetworkFiles

Examples

```
#load example data
data(TEDDY)
data(T1Dmeta)

#make sure metadata and expression data are in same order
T1Dmeta <- T1Dmeta[colnames(TEDDY),]

#create group labels
group_labels <- T1Dmeta$group
names(group_labels) <- rownames(T1Dmeta)

#initiate DNEA object
dnw <- createDNEAobject(project_name = "test", expression_data = TEDDY,
                            group_labels = group_labels)

nodeList(dnw)
```

`numFeatures` *Retrieve the total number of features in the dataset*

Description

This function prints to console the total number of features in the data set

Usage

```
numFeatures(x)

## S4 method for signature 'DNEA'
numFeatures(x)

## S4 method for signature 'DNEAinputSummary'
numFeatures(x)
```

Arguments

x	A DNEA object.

Value

The number of features in the data set.

Author(s)

Christopher Patsalis

See Also

createDNEAobject

Examples

```
#load example data
data(TEDDY)
data(T1Dmeta)

#make sure metadata and expression data are in same order
T1Dmeta <- T1Dmeta[colnames(TEDDY),]

#create group labels
group_labels <- T1Dmeta$group
names(group_labels) <- rownames(T1Dmeta)

#initiate DNEA object
dnw <- createDNEAobject(project_name = "test", expression_data = TEDDY,
                            group_labels = group_labels)

numFeatures(dnw)
```

---

numSamples *Retrieves the total number of samples in the dataset*

Description

This function prints to console the total number of samples in the data set.

Usage

```
numSamples(x)

## S4 method for signature 'DNEA'
numSamples(x)

## S4 method for signature 'DNEAinputSummary'
numSamples(x)
```

Arguments

x	A DNEA object.

Value

The number of samples in the data set.

Author(s)

Christopher Patsalis

See Also

createDNEAobject

Examples

```
#load example data
data(TEDDY)
data(T1Dmeta)

#make sure metadata and expression data are in same order
T1Dmeta <- T1Dmeta[colnames(TEDDY),]

#create group labels
group_labels <- T1Dmeta$group
names(group_labels) <- rownames(T1Dmeta)

#initiate DNEA object
dnw <- createDNEAobject(project_name = "test", expression_data = TEDDY,
                            group_labels = group_labels)

numSamples(dnw)
```

---

optimizedLambda *Access the lambda value used in analysis*

Description

The function takes as input a DNEA object and returns the hyper parameter (lambda) that is currently being used for the analysis. The user may also provide a single-value numeric vector to change the lambda value for analysis.

Usage

```
optimizedLambda(x)

optimizedLambda(x) <- value

## S4 method for signature 'DNEA'
optimizedLambda(x)

## S4 replacement method for signature 'DNEA'
optimizedLambda(x) <- value
```

Arguments

x	A DNEA object.
value	a single-value numeric vector corresponding to the lambda value to use in analysis.

Value

The optimized lambda hyperparameter.

Author(s)

Christopher Patsalis

See Also

BICtune

Examples

```
#load example data
data(TEDDY)
data(T1Dmeta)

#make sure metadata and expression data are in same order
T1Dmeta <- T1Dmeta[colnames(TEDDY),]

#create group labels
group_labels <- T1Dmeta$group
names(group_labels) <- rownames(T1Dmeta)

#initiate DNEA object
dnw <- createDNEAobject(project_name = "test", expression_data = TEDDY,
                            group_labels = group_labels)

optimizedLambda(dnw) <- 0.15
optimizedLambda(dnw)
```

---

plotNetworks *Visualize the biological networks*

Description

This function plots the total network, condition networks, or sub networks as specified by the user. Purple nodes are differential features, green indicates edges specific to group 1, and red indicates edges specific to group 2.

Usage

```
plotNetworks(
  object,
  type = c("group_networks", "sub_networks"),
  subtype = "All",
  layout_func,
  main = "",
  node_size = 15,
  edge_width = 1,
  label_size = 1,
  label_font = 1
)
```

Arguments

object	A DNEA object.
type	There are two possible arguments to **type**: *"group_networks"* specifies the whole network or condition networks. *"sub_networks"* specifies that one of the sub networks should be plotted. *Additional input via the* **subtype** *parameter is required.*
subtype	There are several possible arguments to **subtype**. If *type == "group_networks"*, **subtype** can be *"All"* to plot the whole network (ie. both conditions in the data returned by networkGroups), or one of the condition network names to plot the network corresponding to that condition. If *type == "sub_networks"*, **subtype** should be a single-value numeric vector corresponding to the sub network to plot.
layout_func	The layout in which to plot the specified network. Please see plot.igraph for more information.
main	A character string to use as the plot title.
node_size	The size of the nodes in the plot. The default is 15. Please see *vertex.size* parameter in tkplot for more details.
edge_width	The width of the edges in the plot. The default is 1. Please see *width* parameter in tkplot for more details.
label_size	The size of the node labels in the plot. The default is 1. Please see *label.size* in tkplot for more details.
label_font	Specifies the font type to use in the plot. 1 is normal font, 2 is bold-type, 3 is italic-type, 4 is bold- and italic-type. Please see the *label.font* parameter in plot.igraph for more details.

Value

A plot of the specified network

Author(s)

Christopher Patsalis

See Also

getNetworks,clusterNet, networkGroups

Examples

```
#dnw is a \code{\link[=DNEA-class]{DNEA}} object with the results
#generated for the example data accessed by running
#data(TEDDY) in the console. The workflow for this data
#can be found in the vignette accessed by running
#browseVignettes("DNEA") in the console.
data(dnw)

#plot the networks
plotNetworks(object=dnw, type="group_networks", subtype="All")
plotNetworks(object=dnw, type="sub_networks", subtype=1)
```

---

projectName *Return the name of the current experiment*

Description

This function returns the name of the DNEA experiment.

Usage

```
projectName(x)

## S4 method for signature 'DNEA'
projectName(x)
```

Arguments

x	A DNEA object.

Value

The name of the DNEA experiment.

Author(s)

Christopher Patsalis

See Also

createDNEAobject

Examples

```
#load example data
data(TEDDY)
data(T1Dmeta)

#make sure metadata and expression data are in same order
T1Dmeta <- T1Dmeta[colnames(TEDDY),]

#create group labels
group_labels <- T1Dmeta$group
names(group_labels) <- rownames(T1Dmeta)

#initiate DNEA object
dnw <- createDNEAobject(project_name = "test", expression_data = TEDDY,
                            group_labels = group_labels)

projectName(dnw)
```

`runNetGSA` *Identify metabolic modules that are enriched across experimental conditions using NetGSA*

Description

This function performs pathway enrichment analysis on the metabolic modules identified via `clusterNet` using the `netgsa::NetGSA()` algorithm.

Usage

```
runNetGSA(object, min_size = 5, assay, pathways)
```

Arguments

`object`	A `DNEA`.
`min_size`	The minimum size of a given metabolic module for to be tested for enrichment across the experimental condition.
`assay`	A character string indicating which expression assay to use for analysis. The default is the "log_input-data" assay that is created during `createDNEAobject`.
`pathways`	An adjacency matrix indicating feature inclusion for a given pathway. Features should be columns (with corresponding column names) and pathways should be rows(with corresponding row names). 1 indicates a feature is included in a given pathway and 0 indicates that it is not. Please see `NetGSA` for more information.

Value

A `DNEA` object after populating the @netGSA slot. A summary of the NetGSA results can be viewed using `netGSAresults`.

Author(s)

Christopher Patsalis

References

Hellstern M, Ma J, Yue K, Shojaie A. netgsa: Fast computation and interactive visualization for topology-based pathway enrichment analysis. PLoS Comput Biol. 2021 Jun 11;17(6):e1008979. doi: 10.1371/journal.pcbi.1008979. PMID: 34115744; PMCID: PMC8221786 https://pubmed.ncbi.nlm.nih.gov/34115744/

See Also

netGSAresults clusterNet NetGSA

Examples

```
#dnw is a \code{\link[=DNEA-class]{DNEA}} object with the results
#generated for the example data accessed by running
#data(TEDDY) in the console. The workflow for this data
#can be found in the vignette accessed by running
#browseVignettes("DNEA") in the console.
data(dnw)

#perform pathway enrichment analysis using netGSA
dnw <- runNetGSA(object=dnw, min_size=5)

#view the results
netGSAresults(dnw)
```

---

sampleNames *Retrieve the sample names from the metadata slot.*

Description

This function accesses the sample names stored in the metadata slot of the DNEA object.

Usage

```
sampleNames(x, original = FALSE)

## S4 method for signature 'DNEA'
sampleNames(x, original = FALSE)
```

Arguments

x	A DNEA object.
original	"TRUE" returns the original sample names and "FALSE" returns the sample names that have been modified to avoid errors as a result of special characters using make_clean_names.

Value

A character vector of sample names.

Author(s)

Christopher Patsalis

See Also

createDNEAobject

Examples

```
#load example data
data(TEDDY)
data(T1Dmeta)

#make sure metadata and expression data are in same order
T1Dmeta <- T1Dmeta[colnames(TEDDY),]

#create group labels
group_labels <- T1Dmeta$group
names(group_labels) <- rownames(T1Dmeta)

#initiate DNEA object
dnw <- createDNEAobject(project_name = "test", expression_data = TEDDY,
                            group_labels = group_labels)

sampleNames(dnw)
```

---

selectionProbabilities

Access and set the edge selection probabilities from stabilitySelection()

Description

The function takes as input a DNEA object and returns an *m x n* matrix of selection probabilities for every possible network edge calculated during stabilitySelection.

Usage

```
selectionProbabilities(x)

## S4 method for signature 'DNEA'
selectionProbabilities(x)
```

Arguments

x	A DNEA object.

Value

A DNEA object after filling the selection_probabilities section of the stable_networks slot.

Author(s)

Christopher Patsalis

See Also

stabilitySelection,selectionResults

Examples

```
#dnw is a \code{\link{DNEA}} object with the results generated
#for the example data accessed by running data(TEDDY) in the
#console. The workflow for this data can be found in the
#vignette accessed by running browseVignettes("DNEA")
#in the console.
data("dnw")

selectionProbabilities(dnw)
```

---

selectionResults *Access and set the edge selection results from stabilitySelection()*

Description

The function takes as input a DNEA object and returns an *m x n* matrix of selection results for every possible network edge calculated during stabilitySelection.

Usage

```
selectionResults(x)

## S4 method for signature 'DNEA'
selectionResults(x)
```

Arguments

x	A DNEA object.

Value

A DNEA object after filling the selection_results section of the stable_networks slot.

Author(s)

Christopher Patsalis

See Also

stabilitySelection,selectionProbabilities

Examples

```
#dnw is a DNEA with the results generated for the example data
#accessed by running data(TEDDY) in the console. The workflow
#for this data can be found in the vignette accessed by
#running browseVignettes("DNEA") in the console.
data(dnw)

selectionResults(dnw)
```

---

stabilitySelection *Stability selection calculates selection probabilities for every possible feature-feature interaction within the input data*

Description

This function randomly samples the input data and fits a glasso model with the sampled data for ***nreps*** number of replicates. The resulting adjacency matrices are summed together and selection probabilities for each feature-feature interaction are calculated. Stability selection is particularly useful for smaller data sets. A large number of replicates should be performed (the default is 1000). The exact method deployed varies slightly whether or not additional sub-sampling of the data is performed. More information can be found in the ***Details*** section.

Usage

```
stabilitySelection(
  object,
  subSample = FALSE,
  nreps = 500,
  optimal_lambda,
  assay,
  BPPARAM = bpparam(),
  BPOPTIONS = bpoptions()
)
```

Arguments

object	A DNEA object.
subSample	TRUE/FALSE indicating whether the number of samples are unevenly split by condition and subsampling should be performed when randomly sampling to even out the groups.
nreps	The total number of replicates to perform in stability selection. The default is 1000.
optimal_lambda	**OPTIONAL -** The optimal lambda value to be used in the model. This parameter is only necessary if BICtune is not performed prior.
assay	A character string indicating which expression assay to use for analysis. The default is the "log_scaled_data" assay that is created during createDNEAobject.
BPPARAM	a BiocParallel object.
BPOPTIONS	a list of options for BiocParallel created using the bpoptions function.

Details

Stability selection provides an additional approach by which to regularize the network model and create more robust results, particularly when $\boldsymbol{p \gg n}$. Stability selection works by randomly sampling (without replacement) the input data many times and fitting a glasso model to each subset of sampled data. The unwieghted adjacency matrix from each model is summed together (A feature-feature interaction is considered present if the partial correlation value is above 1e-5), and the probability of an edge being selected in a random subset of the data is calculated by dividing the number of times an edge was selected in the replicates over the total number of replicates. This results in a selection probability for every possible feature-feature interaction that is used to modify the regularization parameter via the following equation:

$$\rho = \lambda * (1/(0.0001 + selection.probability))$$

However, when the sample groups are very unbalanced, randomly sampling strongly favors the larger group, resulting in over representation. In order to combat this, setting subSample=TRUE modifies the random sample by sub-sampling the experimental groups to even out the sample numbers. In this method, 90% of the smaller group is randomly sampled without replacement, and an additional 10% is randomly sampled without replacement from the entire group to preserve the variance. The larger group is randomly sampled to have 1.3 times the number of samples present in the smaller group. This method ensures that each group is equally represented in stability selection.

The principles of stability selection remain similar with both methods, however, there are a few small differences. Stability selection *without* additional sub-sampling randomly samples 50% of each group (without replacement) and fits a model for both halves of the sampled data. Since nearly all of the data for the smaller group is used *with* additional sub-sampling, only one model is fit per replicate when subSample=TRUE. This means that at the default value of nreps=500, 1000 randomly sampled models are fit in total *without* sub-sampling, but 500 randomly sampled models are fit in total *with* sub-sampling. More details about the stability approach deployed in this function can be found in Ma et al. (2019) referenced below.

Value

A DNEA object after populating the stable_networks slot of the object. It contains the selection results from stability selection as well as the calculated selection probabilities.

Author(s)

Christopher Patsalis

References

Ma J, Karnovsky A, Afshinnia F, Wigginton J, Rader DJ, Natarajan L, Sharma K, Porter AC, Rahman M, He J, Hamm L, Shafi T, Gipson D, Gadegbeku C, Feldman H, Michailidis G, Pennathur S. Differential network enrichment analysis reveals novel lipid pathways in chronic kidney disease. Bioinformatics. 2019 Sep 15;35(18):3441-3452. doi: 10.1093/bioinformatics/btz114. PMID: 30887029; PMCID: PMC6748777. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6748777/

Nicolai, M., & Peter, B. (2010). Stability selection. Journal of the Royal Statistical Society: Series B (Statistical Methodology), 72(4), 417-473. https://stat.ethz.ch/Manuscripts/buhlmann/stability.pdf

See Also

selectionProbabilities, selectionResults, bpparam, bpoptions glasso

Examples

```
#import BiocParallel package
library(BiocParallel)

#load example data
data(TEDDY)
data(T1Dmeta)

#make sure metadata and expression data are in same order
TEDDY <- TEDDY[seq(50), ]
T1Dmeta <- T1Dmeta[colnames(TEDDY),]

#create group labels
group_labels <- T1Dmeta$group
names(group_labels) <- rownames(T1Dmeta)

#initiate DNEA object
dnw <- createDNEAobject(project_name = "test", expression_data = TEDDY,
                            group_labels = group_labels)

#optimize lambda parameter
dnw <- BICtune(object=dnw,
               informed=TRUE,
               interval=0.01)

# perform stability selection
```

```
dnw <- stabilitySelection(object=dnw,
                          subSample=FALSE,
                          nreps=4,
                          BPPARAM=bpparam())
```

---

subnetworkMembership *Retrieve the subnetwork membership for each feature*

Description

The function takes as input a DNEA object and returns the results of consensus clustering determined via clusterNet.

Usage

```
subnetworkMembership(x)

## S4 method for signature 'DNEA'
subnetworkMembership(x)

## S4 method for signature 'consensusClusteringResults'
subnetworkMembership(x)
```

Arguments

x	A DNEA object.

Value

A data frame that corresponds to the results of consensus clustering.

Author(s)

Christopher Patsalis

See Also

clusterNet

Examples

```
#dnw is a \code{\link{DNEA}} object with the results
#generated for the example data accessed by running
#data(TEDDY) in the console. The workflow for this data
#can be found in the vignette accessed by running
#browseVignettes("DNEA") in the console.
data("dnw")

subnetworkMembership(dnw)
```

sumExp2DNEA *Initialize a DNEA from SummarizedExperiment*

Description

This function takes as input a SummarizedExperiment-class object non-transformed in order to initiate a DNEA object. Differential expression analysis is performed using a student's T-test and Benjamini-Hochberg for multiple-testing corrections. Diagnostic testing is done on the input data by checking the minimum eigen value and condition number of the expression data for each experimental condition.

IMPORTANT::

Special attention should be given to the diagnostic criteria that is output. The minimum eigen value and condition number are calculated for the whole data set as well as for each condition to determine mathematic stability of the data set and subsequent results from a GGM model. More information about interpretation can be found in the ***Details*** section below.

Usage

```
sumExp2DNEA(project_name, object, scaled_expression_assay, group_label_col)
```

Arguments

project_name	A character string name for the experiment.
object	a SummarizedExperiment object object.
scaled_expression_assay	A character string corresponding to the assay in the summarizedExperiment object to use for analysis. Defaults to log-scaling the count data if not provided.
group_label_col	A character string corresponding to the column in the sample metadata stored in the SummarizedExperiment object to use as the group labels.

Details

Diagnostics Motivation:

Negative or zero eigenvalues in a data set can represent instability in that portion of the matrix, thereby invalidating parametric statistical methods and creating unreliable results. In this function, the minimum eigenvalue of the data set is calculated by first creating a pearson correlation matrix of the data. Instability may then occur for a number of reasons, but one common cause is highly correlated features (in the positive and negative direction).

Regularization often takes care of this problem by arbitrarily selecting one of the variables in a highly correlated group and removing the rest. We have developed DNEA to be very robust in situations where $\boldsymbol{p \gg n}$ by optimizing the model via several regularization steps (*please see* BICtune *and* stabilitySelection) that may handle such problems without intervention, however, the user can also pre-emptively collapse highly-correlated features into a single group via aggregateFeatures.

Benefits of Feature Aggregation:

When your dataset contains highly correlated features, we recommend aggregating features into related groups - such as highly-correlated features of a given class of molecules (ie. many fatty acids, carnitines, etc.) - because the user then has more control over which variables are included in the model. Without collapsing, the model regularization may result in one of the features within a class being included and some or all of the remaining features being removed. By collapsing first, you retain the signal from all of the features in the collapsed group and also have information pertaining to which features are highly correlated and will therefore have similar feature-feature associations.

Value

A DNEA object.

Author(s)

Christopher Patsalis

See Also

BICtune, stabilitySelection, createDNEAobject

Examples

```
#load example data from airway package
library(airway)
data(airway)

airway <- airway[1:50,]
airway <- airway[rowSums(SummarizedExperiment::assays(airway)$counts) > 5, ]
DNEA <- sumExp2DNEA(project_name = "airway",
                        object = airway,
                        group_label_col = "dex")
```

T1Dmeta *Sample meta data for the The Environmental Determinants of Diabetes in the Young (TEDDY) clinical trial*

Description

This is a data frame containing metadata for the samples in the corresponding TEDDY example data from "The Environmental Determinants of Diabetes in the Young" clinical trial.

Usage

```
data("T1Dmeta")
```

Format

A data frame with 322 rows and 7 columns. Each row corresponds to a sample, and each column corresponds to:

subject The individual patient

Endpoint1 The age of the case subject in days when this sample was collected

Endpoint2 The age of the control subject in days when this sample was collected

Age The age of the subject in days when this sample was collected

Sex The sex of the subject

sample The name of this sample

group A variable indicating whether or not this sample is part of the T1D case or T1D control group

Value

A data frame containing the sample metadata for the TEDDY metabolomics study

Source

The raw data can be downloaded from the Metabolomics workbench under study ID **ST001386**: https://www.metabolomicsworkbench.org/data/DRCCStudySummary.php?Mode=SetupRawDataDownload&StudyID=ST001386

References

Lee HS, Burkhardt BR, McLeod W, Smith S, Eberhard C, Lynch K, Hadley D, Rewers M, Simell O, She JX, Hagopian B, Lernmark A, Akolkar B, Ziegler AG, Krischer JP; TEDDY study group. Biomarker discovery study design for type 1 diabetes in The Environmental Determinants of Diabetes in the Young (TEDDY) study. Diabetes Metab Res Rev. 2014 Jul;30(5):424-34. doi: 10.1002/dmrr.2510. PMID: 24339168; PMCID: PMC4058423. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4058423/

TEDDY *Example expresion data set from The Environmental Determinants of Diabetes in the Young (TEDDY) clinical trial*

Description

This data is an *m x n* expression matrix corresponding to a curated list of metabolites from "The Environmental Determinants of Diabetes in the Young" clinical trial. The data was downloaded from

Usage

```
data("TEDDY")
```

Format

A matrix with 134 rows and 322 columns. Each row corresponds to a unique metabolite, and each column corresponds to a sample

Value

An *m x n* expression matrix of metabolomics data from the TEDDY dataset

Source

This data is available at the NIH Common Fund's National Metabolomics Data Repository (NMDR) website, the Metabolomics Workbench, where it has been assigned Project ID PR000950 and Study ID ST001386. The data can be accessed directly via it's Project DOI: 10.21228/M8WM4P. This work is supported by NIH grant, U2C- DK119886.

References

Lee HS, Burkhardt BR, McLeod W, Smith S, Eberhard C, Lynch K, Hadley D, Rewers M, Simell O, She JX, Hagopian B, Lernmark A, Akolkar B, Ziegler AG, Krischer JP; TEDDY study group. Biomarker discovery study design for type 1 diabetes in The Environmental Determinants of Diabetes in the Young (TEDDY) study. Diabetes Metab Res Rev. 2014 Jul;30(5):424-34. doi: 10.1002/dmrr.2510. PMID: 24339168; PMCID: PMC4058423

Index

∗ **datasets**
dnw, 27
metab_data, 40
T1Dmeta, 61
TEDDY, 62

addExpressionData, 4
adjacencyGraph, 6
adjacencyGraph,consensusClusteringResults-method *(adjacencyGraph)*, 6
adjacencyGraph,DNEA-method *(adjacencyGraph)*, 6
adjacencyMatrix, 7, *32*, *35*
adjacencyMatrix,DNEA-method *(adjacencyMatrix)*, 7
aggregateFeatures, 8, *18*, *20*, *22, 23*, *25, 26*, *29*, *38*, *60*

BICscores, 11
BICscores,DNEA-method *(BICscores)*, 11
BICscores<- *(BICscores)*, 11
BICscores<-,DNEA-method *(BICscores)*, 11
BICtune, *4*, *11*, 12, *20, 21*, *25*, *34*, *37–39*, *49*, *57*, *60, 61*
BICtune,DNEA-method *(BICtune)*, 12
BICtune,matrix-method *(BICtune)*, 12
BiocParallel, *13*, *34*
bpoptions, *13, 14*, *34*, *57, 58*
bpparam, *14*, *58*

CCsummary, 15, *17*
CCsummary,DNEA-method *(CCsummary)*, 15
cluster_edge_betweenness, *16*
cluster_fast_greedy, *16*
cluster_infomap, *16*
cluster_label_prop, *16*
cluster_leading_eigen, *16*
cluster_louvain, *16*
cluster_walktrap, *16*
clusterNet, *4*, *6, 7*, *15*, 16, *19*, *25*, *45*, *51–53*, *59*
collapsed_DNEA, *37*, *44*
collapsed_DNEA *(collapsed_DNEA-class)*, 17
collapsed_DNEA-class, 17
consensusClusteringResults, *15*
consensusClusteringResults *(consensusClusteringResults-class)*, 19
consensusClusteringResults-class, 19
createDNEAobject, *4, 5*, *9, 10*, *13*, *18*, 19, *22, 23*, *25, 26*, *29, 30*, *34*, *39*, *41*, *44, 45*, *47, 48*, *51, 52*, *54*, *57*, *61*

datasetSummary, 21, *24, 25*
datasetSummary,DNEA-method *(datasetSummary)*, 21
diagnostics, 22, *24, 25*
diagnostics,DNEA-method *(diagnostics)*, 22
diagnostics,DNEAinputSummary-method *(diagnostics)*, 22
DNEA, *4–9*, *11*, *13–18*, *21–24*, *26–39*, *41–61*
DNEA *(DNEA-package)*, 3
DNEA-class, 24
DNEA-package, 3
DNEAinputSummary *(DNEAinputSummary-class)*, 26
DNEAinputSummary-class, 26
dnw, 27

edgeList, *24, 25*, 28, *33*, *35*
edgeList,DNEA-method *(edgeList)*, 28
edgeList<- *(edgeList)*, 28
edgeList<-,DNEA-method *(edgeList)*, 28
expressionData, *24, 25*, 29
expressionData,DNEA-method *(expressionData)*, 29

featureNames, 30, *36*

featureNames,DNEA-method
(featureNames), 30
filterNetworks, *28*, 31
filterNetworks,DNEA-method
(filterNetworks), 31
filterNetworks,list-method
(filterNetworks), 31

getNetworkFiles, *28*, 32, *45*
getNetworks, *4*, *7, 8*, *16*, *24, 25*, *28*, *32*, 33, *51*
glasso, *14*, *58*

igraph, *7*, *16*, *19*
includeMetadata, *24, 25*, 35, *41*, *43*

lambdas2Test, 37
lambdas2Test,DNEA-method
(lambdas2Test), 37

make_clean_names, *30*, *54*
massDataset2DNEA, *19*, *21*, 38
metab_data, 40
metaData, *36*, 41
metaData,DNEA-method (metaData), 41

NetGSA, *25*, *52, 53*
netgsa::NetGSA(), *42*, *52*
netGSAresults, 42, *52, 53*
netGSAresults,DNEA-method
(netGSAresults), 42
networkGroupIDs, 43, *44*
networkGroupIDs,DNEA-method
(networkGroupIDs), 43
networkGroupIDs<- (networkGroupIDs), 43
networkGroups, *5*, *7*, 44, *50, 51*
networkGroups,DNEA-method
(networkGroups), 44
nodeList, *17*, *24, 25*, *33*, 45
nodeList,DNEA-method (nodeList), 45
nodeList<- (nodeList), 45
nodeList<-,DNEA-method (nodeList), 45
numFeatures, 46
numFeatures,DNEA-method (numFeatures),
46
numFeatures,DNEAinputSummary-method
(numFeatures), 46
numSamples, 47
numSamples,DNEA-method (numSamples), 47
numSamples,DNEAinputSummary-method
(numSamples), 47

optimizedLambda, *14*, *25*, 48
optimizedLambda,DNEA-method
(optimizedLambda), 48
optimizedLambda<- (optimizedLambda), 48
optimizedLambda<-,DNEA-method
(optimizedLambda), 48

plot.igraph, *50*
plotNetworks, *17*, 49
projectName, 51
projectName,DNEA-method (projectName),
51

runNetGSA, *4*, *25*, *42*, 52

sampleNames, *36*, 53
sampleNames,DNEA-method (sampleNames),
53
selectionProbabilities, *25*, 54, *56*, *58*
selectionProbabilities,DNEA-method
(selectionProbabilities), 54
selectionResults, *25*, *55*, 55, *58*
selectionResults,DNEA-method
(selectionResults), 55
show,DNEA-method (DNEA-class), 24
show,DNEAinputSummary-method
(DNEAinputSummary-class), 26
stabilitySelection, *4*, *9, 10*, *12*, *20, 21*, *25*,
33, *38, 39*, *54–56*, 56, *60, 61*
subnetworkMembership, 59
subnetworkMembership,consensusClusteringResults-method
(subnetworkMembership), 59
subnetworkMembership,DNEA-method
(subnetworkMembership), 59
sumExp2DNEA, *19*, *21*, 60

T1Dmeta, 61
TEDDY, *27*, *40*, *61*, 62
tkplot, *50*