

# Package ‘glmSparseNet’

May 2, 2023

**Type** Package

**Title** Network Centrality Metrics for Elastic-Net Regularized Models

**Version** 1.18.0

**Description** glmSparseNet is an R-package that generalizes sparse regression models when the features (e.g. genes) have a graph structure (e.g. protein-protein interactions), by including network-based regularizers. glmSparseNet uses the glmnet R-package, by including centrality measures of the network as penalty weights in the regularization. The current version implements regularization based on node degree, i.e. the strength and/or number of its associated edges, either by promoting hubs in the solution or orphan genes in the solution. All the glmnet distribution families are supported, namely ``gaussian``, ``poisson``, ``binomial``, ``multinomial``, ``cox``, and ``mgaussian``.

**License** GPL-3

**Encoding** UTF-8

**LazyData** false

**NeedsCompilation** no

**biocViews** Software, StatisticalMethod, DimensionReduction, Regression, Classification, Survival, Network, GraphAndNetwork

**Depends** R (>= 4.1), Matrix, MultiAssayExperiment, glmnet

**Imports** SummarizedExperiment, biomaRt, futile.logger, futile.options, forcats, utils, dplyr, glue, readr, digest, httr, ggplot2, survminer, reshape2, stringr, parallel, methods

**Suggests** testthat, knitr, rmarkdown, survival, survcomp, pROC, VennDiagram, BiocStyle, curatedTCGADData, TCGAutils

**VignetteBuilder** knitr

**RoxygenNote** 7.1.2

**BugReports** <https://www.github.com/sysbiomed/glmSparseNet/issues>

**URL** <https://www.github.com/sysbiomed/glmSparseNet>

**git\_url** <https://git.bioconductor.org/packages/glmSparseNet>

**git\_branch** RELEASE\_3\_17

**git\_last\_commit** fa0321e

**git\_last\_commit\_date** 2023-04-25

**Date/Publication** 2023-05-02

**Author** André Veríssimo [aut, cre],  
 Susana Vinga [aut],  
 Eunice Carrasquinha [ctb],  
 Marta Lopes [ctb]

**Maintainer** André Veríssimo <andre.verissimo@tecnico.ulisboa.pt>

## R topics documented:

.calcPenalty . . . . .	3
.degreeGeneric . . . . .	4
.glmSparseNetPrivate . . . . .	5
.networkGenericParallel . . . . .	6
.networkWorker . . . . .	7
balanced.cv.folds . . . . .	7
base.dir . . . . .	8
biomart.load . . . . .	8
build.function.digest . . . . .	9
buildLambda . . . . .	10
buildStringNetwork . . . . .	11
cache.compression . . . . .	12
calculate.combined.score . . . . .	12
calculate.result . . . . .	13
create.directory.for.cache . . . . .	14
curl.workaround . . . . .	14
cv.glmDegree . . . . .	15
cv.glmHub . . . . .	16
cv.glmOrphan . . . . .	17
cv.glmSparseNet . . . . .	18
degreeCor . . . . .	19
degreeCov . . . . .	20
digest.cache . . . . .	21
downloadFileLocal . . . . .	22
ensemblGeneNames . . . . .	23
geneNames . . . . .	23
glmDegree . . . . .	24
glmHub . . . . .	25
glmOrphan . . . . .	25
glmSparseNet . . . . .	26
glmSparseNet.options . . . . .	28
hallmarks . . . . .	28
heuristicScale . . . . .	29
hubHeuristic . . . . .	30

<code>.calcPenalty</code>	3
<code>my.colors</code>	30
<code>my.symbols</code>	31
<code>networkCorParallel</code>	31
<code>networkCovParallel</code>	32
<code>networkOptions</code>	33
<code>orphanHeuristic</code>	34
<code>protein2EnsemblGeneNames</code>	34
<code>run.cache</code>	35
<code>run.cache,function-method</code>	36
<code>save.run.cache</code>	37
<code>separate2GroupsCox</code>	38
<code>show.message</code>	40
<code>string.network.700.cache</code>	40
<code>stringDBhomoSapiens</code>	41
<code>tempdir.cache</code>	41
<code>write.readme</code>	42
<b>Index</b>	<b>43</b>

---

<code>.calcPenalty</code>	<i>Calculate penalty based on data</i>
---------------------------	--

---

## Description

Internal method to calculate the network using data-dependant methods

## Usage

```
.calcPenalty(xdata, penalty.type, network.options = networkOptions())
```

## Arguments

<code>xdata</code>	input data
<code>penalty.type</code>	which method to use
<code>network.options</code>	options to be used

## Value

vector with penalty weights

**Examples**

```
xdata <- matrix(rnorm(1000), ncol = 200)
glmSparseNet:::calcPenalty(xdata, 'none')
glmSparseNet:::calcPenalty(xdata, 'correlation',
                           networkOptions(cutoff = .6))
glmSparseNet:::calcPenalty(xdata, 'correlation')
glmSparseNet:::calcPenalty(xdata, 'covariance',
                           networkOptions(cutoff = .6))
glmSparseNet:::calcPenalty(xdata, 'covariance')
```

---

*.degreeGeneric**Generic function to calculate degree based on data*

---

**Description**

The assumption to use this function is that the network represented by a matrix is symmetric and without any connection the node and itself.

**Usage**

```
.degreeGeneric(
  fun = stats::cor,
  fun.prefix = "operator",
  xdata,
  cutoff = 0,
  consider.unweighted = FALSE,
  chunks = 1000,
  force.recalc.degree = FALSE,
  force.recalc.network = FALSE,
  n.cores = 1,
  ...
)
```

**Arguments**

<code>fun</code>	function that will calculate the edge weight between 2 nodes
<code>fun.prefix</code>	used to store low-level information on network as it can become to large to be stored in memory
<code>xdata</code>	calculate correlation matrix on each column
<code>cutoff</code>	positive value that determines a cutoff value
<code>consider.unweighted</code>	consider all edges as 1 if they are greater than 0
<code>chunks</code>	calculate function at batches of this value (default is 1000)
<code>force.recalc.degree</code>	force recalculation of penalty weights (but not the network), instead of going to cache

force.recalc.network      force recalculation of network and penalty weights, instead of going to cache  
n.cores                      number of cores to be used  
...                            extra parameters for fun

**Value**

a vector of the degrees

---

.glmSparseNetPrivate    *Calculate GLM model with network-based regularization*

---

**Description**

Calculate GLM model with network-based regularization

**Usage**

```
.glmSparseNetPrivate(
  fun,
  xdata,
  ydata,
  network,
  experiment.name = NULL,
  network.options = networkOptions(),
  ...
)
```

**Arguments**

fun                          function to be called (glmnet or cv.glmnet)  
xdata                        input data, can be a matrix or MultiAssayExperiment  
ydata                        response data compatible with glmnet  
network                     type of network, see below  
experiment.name            when xdata is a MultiAssayExperiment object this parameter is required  
network.options            options to calculate network  
...                          parameters that glmnet accepts

**Value**

an object just as glmnet network parameter accepts:  
\* string to calculate network based on data (correlation, covariance) \* matrix representing the network \* vector with already calculated penalty weights (can also be used directly with glmnet)

---

`.networkGenericParallel`*Calculate the upper triu of the matrix*

---

**Description**

Calculate the upper triu of the matrix

**Usage**

```
.networkGenericParallel(  
  fun,  
  fun.prefix,  
  xdata,  
  build.output = "matrix",  
  n.cores = 1,  
  force.recalc.network = FALSE,  
  show.message = FALSE,  
  ...  
)
```

**Arguments**

<code>fun</code>	function that will calculate the edge weight between 2 nodes
<code>fun.prefix</code>	used to store low-level information on network as it can become to large to be stored in memory
<code>xdata</code>	base data to calculate network
<code>build.output</code>	if output returns a 'matrix', 'vector' of the upper triu without the diagonal or NULL with any other argument
<code>n.cores</code>	number of cores to be used
<code>force.recalc.network</code>	force recalculation, instead of going to cache
<code>show.message</code>	shows cache operation messages
<code>...</code>	extra parameters for fun

**Value**

depends on `build.output` parameter

---

.networkWorker	<i>Worker to calculate edge weight for each pair of ix.i node and following</i>
----------------	---

---

**Description**

Note that it assumes it does not calculate for index below and equal to ix.i

**Usage**

```
.networkWorker(fun, xdata, ix.i, ...)
```

**Arguments**

fun	function to be used, can be cor, cov or any other defined function
xdata	original data to calculate the function over
ix.i	starting index, this can be used to save ony upper triu
...	extra parameters for fun

**Value**

a vector with size 'ncol(xdata) - ix.i'

---

balanced.cv.folds	<i>Create balanced folds for cross validation</i>
-------------------	---

---

**Description**

Create balanced folds for cross validation

**Usage**

```
balanced.cv.folds(..., n folds = 10)
```

**Arguments**

...	vectors representing data
n folds	number of folds to be created

**Value**

list with given input, n folds and result. The result is a list matching the input with foldid attributed to each position.

### Examples

```
glmSparseNet::balanced.cv.folds(seq(10), seq(11, 15), nfolds = 2)
# will give a warning
glmSparseNet::balanced.cv.folds(seq(10), seq(11, 13), nfolds = 10)
glmSparseNet::balanced.cv.folds(seq(100), seq(101, 133), nfolds = 10)
```

---

base.dir	<i>change base.dir for run.cache</i>
----------	--------------------------------------

---

### Description

change base.dir for run.cache

### Usage

```
base.dir(path = NULL)
```

### Arguments

path                   to base directory where cache is saved

### Value

the new path

### Examples

```
glmSparseNet::base.dir('/tmp/cache')
```

---

biomart.load	<i>Common call to biomaRt to avoid repetitive code</i>
--------------	--

---

### Description

Common call to biomaRt to avoid repetitive code

### Usage

```
biomart.load(attributes, filters, values, use.cache, verbose)
```



**Arguments**

attributes	Attributes you want to retrieve. A possible list of attributes can be retrieved using the function <code>biomaRt::listAttributes</code> .
filters	Filters (one or more) that should be used in the query. A possible list of filters can be retrieved using the function <code>biomaRt::listFilters</code> .
values	Values of the filter, e.g. vector of affy IDs. If multiple filters are specified then the argument should be a list of vectors of which the position of each vector corresponds to the position of the filters in the filters argument
use.cache	Boolean indicating if <code>biomaRt</code> cache should be used
verbose	When using <code>biomaRt</code> in webservice mode and setting <code>verbose</code> to <code>TRUE</code> , the XML query to the webservice will be printed.

**Value**

data.frame with attributes as columns and values translated to them

**See Also**

`geneNames`  
`ensemblGeneNames`  
`protein2EnsemblGeneNames`  
`biomaRt::getBM()`  
`biomaRt::useEnsembl()`

**Examples**

```
glmSparseNet::biomart.load(  
  attributes = c("external_gene_name", "ensembl_gene_id"),  
  filters = "external_gene_name",  
  values = c('MOB1A', 'RFLNB', 'SPIC', 'TP53'),  
  use.cache = TRUE,  
  verbose = FALSE  
)
```

---

`build.function.digest` *Build digest of function from the actual code*

---

**Description**

Build digest of function from the actual code

**Usage**

```
build.function.digest(fun)
```

**Arguments**

fun                    function call name

**Value**

a digest

**Examples**

```
glmSparseNet:::build.function.digest(sum)
glmSparseNet:::build.function.digest(c)
```

---

buildLambda

*Auxiliary function to generate suitable lambda parameters*

---

**Description**

Auxiliary function to generate suitable lambda parameters

**Usage**

```
buildLambda(
  lambda.largest = NULL,
  xdata = NULL,
  ydata = NULL,
  family = NULL,
  orders.of.magnitude.smaller = 3,
  lambda.per.order.magnitude = 150
)
```

**Arguments**

lambda.largest    numeric value for largest number of lambda to consider (usually with a target of 1 selected variable)

xdata             X parameter for glmnet function

ydata             Y parameter for glmnet function

family            family parameter to glmnet function

orders.of.magnitude.smaller  
                  minimum value for lambda ( $\text{lambda.largest} / 10^{\text{orders.of.magnitude.smaller}}$ )

lambda.per.order.magnitude  
                  how many lambdas to create for each order of magnitude

**Value**

a numeric vector with suitable lambdas

**Examples**

```
buildLambda(5.4)
```

---

```
buildStringNetwork      Build gene network from peptide ids
```

---

**Description**

This can reduce the dimension of the original network, as there may not be a mapping between peptide and gene id

**Usage**

```
buildStringNetwork(string.tbl, use.names = "protein")
```

**Arguments**

<code>string.tbl</code>	matrix with colnames and rownames as ensembl peptide id (same order)
<code>use.names</code>	default is to use protein names ('protein'), other options are 'ensembl' for ensembl gene id or 'external' for external gene names

**Value**

a new matrix with gene ids instead of peptide ids. The size of matrix can be different as there may not be a mapping or a peptide mapping can have multiple genes.

**See Also**

```
stringDBhomoSapiens
```

**Examples**

```
all.interactions.700 <- stringDBhomoSapiens(score_threshold = 700)
string.network       <- buildStringNetwork(all.interactions.700,
                                           use.names = 'external')

# number of edges
sum(string.network != 0)
```

---

cache.compression	<i>change cache.compression for run.cache</i>
-------------------	---

---

**Description**

change cache.compression for run.cache

**Usage**

```
cache.compression(compression = NULL)
```

**Arguments**

compression      see compression parameter in save function

**Value**

the new compression

**Examples**

```
glmSparseNet:::cache.compression('bzip2')
```

---

```
calculate.combined.score
```

*Calculate combined score for STRINGdb interactions*

---

**Description**

Please note that all the interactions have duplicates as it's a two way interaction ( $\text{score}(\text{ProteinA-Protein}) == \text{score}(\text{ProteinB, PorteinA})$ )

**Usage**

```
calculate.combined.score(all.interactions, score_threshold, remove.text)
```

**Arguments**

all.interactions  
                   table with score of all interactions

score\_threshold  
                   threshold to keep interactions

remove.text      remove text-based interactions

**Details**

To better understand how the score is calculated, please see: <https://string-db.org/help/faq/#how-are-the-scores-computed>

**Value**

table with combined score

---

calculate.result	<i>Calculate/load result and save if necessary</i>
------------------	--

---

**Description**

This is where the actual work is done

**Usage**

```
calculate.result(path, compression, force.recalc, show.message, fun, ...)
```

**Arguments**

path	path to save cache
compression	compression used in save
force.recalc	force to recalculate cache
show.message	boolean to show messages
fun	function to be called
...	arguments to said function ,

**Value**

result of fun(...)

**Examples**

```
glmSparseNet::calculate.result(
  file.path(tempdir(), 'calculate.result.Rdata'),
  'gzip',
  FALSE,
  TRUE,
  sum,
  1, 2, 3
)
```

---

```
create.directory.for.cache
```

*Create directories for cache*

---

### Description

Create directories for cache

### Usage

```
create.directory.for.cache(base.dir, parent.path)
```

### Arguments

base.dir	tentative base dir to create.
parent.path	first 4 characters of digest that will become parent directory for the actual cache file (this reduces number of files per folder)

### Value

a list of updated base.dir and parent.dir

### Examples

```
glmSparseNet::create.directory.for.cache(tempdir(), 'abcd')

glmSparseNet::create.directory.for.cache(
  file.path(getwd(), 'run-cache'), 'abcd'
)
```

---

```
curl.workaround
```

*Workaround for bug with curl when fetching specific ensembl mirror*

---

### Description

Should be solved in issue #39, will test to remove it.

### Usage

```
curl.workaround(expr)
```

### Arguments

expr	expression
------	------------

**Value**

result of expression

**Examples**

```
glmSparseNet::curl.workaround({
  biomaRt::useEnsembl(
    biomart = "genes",
    dataset = 'hsapiens_gene_ensembl')
})
```

---

cv.glmDegree

*GLMNET cross-validation model penalizing nodes with small degree*


---

**Description**

This function overrides the ‘trans.fun’ options in ‘network.options’ with the inverse of a degree described in Verissimo et al. (2015) that penalizes nodes with small degree.

**Usage**

```
cv.glmDegree(xdata, ydata, network, network.options = networkOptions(), ...)
```

**Arguments**

xdata	input data, can be a matrix or MultiAssayExperiment
ydata	response data compatible with glmnet
network	type of network, see below
network.options	options to calculate network
...	parameters that glmnet accepts

**Value**

see cv.glmSparseNet

**See Also**

glmNetSparse

**Examples**

```
xdata <- matrix(rnorm(100), ncol = 5)
cv.glmDegree(xdata, rnorm(nrow(xdata)), 'correlation',
  family = 'gaussian',
  nfolds = 5,
  network.options = networkOptions(min.degree = .2))
```

---

`cv.glmHub`*GLMNET cross-validation model penalizing nodes with small degree*

---

### Description

This function overrides the ‘trans.fun’ options in ‘network.options’ with an heuristic described in Verissimo et al. that penalizes nodes with small degree.

### Usage

```
cv.glmHub(xdata, ydata, network, network.options = networkOptions(), ...)
```

### Arguments

<code>xdata</code>	input data, can be a matrix or MultiAssayExperiment
<code>ydata</code>	response data compatible with glmnet
<code>network</code>	type of network, see below
<code>network.options</code>	options to calculate network
<code>...</code>	parameters that glmnet accepts

### Value

see `cv.glmSparseNet`

### See Also

`glmNetSparse`

### Examples

```
xdata <- matrix(rnorm(100), ncol = 5)
cv.glmHub(xdata, rnorm(nrow(xdata)), 'correlation',
          family = 'gaussian',
          nfolds = 5,
          network.options = networkOptions(min.degree = .2))
```



---

cv.glmOrphan	<i>GLMNET cross-validation model penalizing nodes with high degree</i>
--------------	--

---

## Description

This function overrides the ‘trans.fun’ options in ‘network.options’ with an heuristic described in Verissimo et al. that penalizes nodes with high degree.

## Usage

```
cv.glmOrphan(xdata, ydata, network, network.options = networkOptions(), ...)
```

## Arguments

xdata	input data, can be a matrix or MultiAssayExperiment
ydata	response data compatible with glmnet
network	type of network, see below
network.options	options to calculate network
...	parameters that glmnet accepts

## Value

see cv.glmSparseNet

## See Also

glmNetSparse

## Examples

```
xdata <- matrix(rnorm(100), ncol = 5)
cv.glmOrphan(xdata, rnorm(nrow(xdata)), 'correlation',
             family = 'gaussian',
             nfolds = 5,
             network.options = networkOptions(min.degree = .2))
```

---

cv.glmSparseNet	<i>Calculate cross validating GLM model with network-based regularization</i>
-----------------	---

---

### Description

network parameter accepts:

### Usage

```
cv.glmSparseNet(
  xdata,
  ydata,
  network,
  network.options = networkOptions(),
  experiment.name = NULL,
  ...
)
```

### Arguments

xdata	input data, can be a matrix or MultiAssayExperiment
ydata	response data compatible with glmnet
network	type of network, see below
network.options	options to calculate network
experiment.name	Name of experiment in MultiAssayExperiment
...	parameters that cv.glmnet accepts

### Details

\* string to calculate network based on data (correlation, covariance) \* matrix representing the network \* vector with already calculated penalty weights (can also be used directly glmnet)

### Value

an object just as cv.glmnet

### Examples

```
# Gaussian model
xdata <- matrix(rnorm(500), ncol = 5)
cv.glmSparseNet(xdata, rnorm(nrow(xdata)), 'correlation',
  family = 'gaussian')
cv.glmSparseNet(xdata, rnorm(nrow(xdata)), 'covariance',
```

```

        family = 'gaussian')

#
#
# Using MultiAssayExperiment with survival model

#
# load data
data('miniACC', package="MultiAssayExperiment")
xdata <- miniACC

#
# build valid data with days of last follow up or to event
event.ix <- which(!is.na(xdata$days_to_death))
cens.ix <- which(!is.na(xdata$days_to_last_followup))
xdata$surv_event_time <- array(NA, nrow(colData(xdata)))
xdata$surv_event_time[event.ix] <- xdata$days_to_death[event.ix]
xdata$surv_event_time[cens.ix] <- xdata$days_to_last_followup[cens.ix]

#
# Keep only valid individuals
valid.ix <- as.vector(!is.na(xdata$surv_event_time) &
                     !is.na(xdata$vital_status) &
                     xdata$surv_event_time > 0)
xdata.valid <- xdata[, rownames(colData(xdata))[valid.ix]]
ydata.valid <- colData(xdata.valid)[,c('surv_event_time', 'vital_status')]
colnames(ydata.valid) <- c('time', 'status')

#
cv.glmSparseNet(xdata.valid,
                ydata.valid,
                nfolds      = 5,
                family      = 'cox',
                network      = 'correlation',
                experiment.name = 'RNASeq2GeneNorm')

```

---

degreeCor

*Calculate the degree of the correlation network based on xdata*


---

## Description

Calculate the degree of the correlation network based on xdata

## Usage

```

degreeCor(
  xdata,
  cutoff = 0,

```

```

    consider.unweighted = FALSE,
    force.recalc.degree = FALSE,
    force.recalc.network = FALSE,
    n.cores = 1,
    ...
)

```

### Arguments

xdata	calculate correlation matrix on each column
cutoff	positive value that determines a cutoff value
consider.unweighted	consider all edges as 1 if they are greater than 0
force.recalc.degree	force recalculation of penalty weights (but not the network), instead of going to cache
force.recalc.network	force recalculation of network and penalty weights, instead of going to cache
n.cores	number of cores to be used
...	extra parameters for cor function

### Value

a vector of the degrees

### Examples

```

n.col <- 6
xdata <- matrix(rnorm(n.col * 4), ncol = n.col)
degreeCor(xdata)
degreeCor(xdata, cutoff = .5)
degreeCor(xdata, cutoff = .5, consider.unweighted = TRUE)

```

---

degreeCov

*Calculate the degree of the covariance network based on xdata*

---

### Description

Calculate the degree of the covariance network based on xdata

**Usage**

```

degreeCov(
  xdata,
  cutoff = 0,
  consider.unweighted = FALSE,
  force.recalc.degree = FALSE,
  force.recalc.network = FALSE,
  n.cores = 1,
  ...
)

```

**Arguments**

xdata	calculate correlation matrix on each column
cutoff	positive value that determines a cutoff value
consider.unweighted	consider all edges as 1 if they are greater than 0
force.recalc.degree	force recalculation of penalty weights (but not the network), instead of going to cache
force.recalc.network	force recalculation of network and penalty weights, instead of going to cache
n.cores	number of cores to be used
...	extra parameters for cov function

**Value**

a vector of the degrees

**Examples**

```

n.col <- 6
xdata <- matrix(rnorm(n.col * 4), ncol = n.col)
degreeCov(xdata)
degreeCov(xdata, cutoff = .5)
degreeCov(xdata, cutoff = .5, consider.unweighted = TRUE)

```

---

digest.cache	<i>Default digest method</i>
--------------	------------------------------

---

**Description**

Sets a default caching algorithm to use with run.cache

**Usage**

```
digest.cache(val)
```

**Arguments**

val                    object to calculate hash over

**Value**

a hash of the sha256

**Examples**

```
glmSparseNet:::digest.cache(c(1,2,3,4,5))  
glmSparseNet:::digest.cache('some example')
```

---

downloadFileLocal        *Download files to local temporary path*

---

**Description**

In case of new call it uses the temporary cache instead of downloading again.

**Usage**

```
downloadFileLocal(urlStr, oD = tempdir())
```

**Arguments**

urlStr                url of file to download  
oD                    temporary directory to store file

**Details**

Inspired by STRINGdb Bioconductor package, but using curl as file may be too big to handle.

**Value**

path to file

**Examples**

```
glmSparseNet:::downloadFileLocal(  
  'https://string-db.org/api/tsv-no-header/version')
```

---

ensemblGeneNames	<i>Retrieve ensembl gene names from biomaRt</i>
------------------	---

---

**Description**

Retrieve ensembl gene names from biomaRt

**Usage**

```
ensemblGeneNames(gene.id, use.cache = TRUE, verbose = FALSE)
```

**Arguments**

gene.id	character vector with gene names
use.cache	Boolean indicating if biomaRt cache should be used
verbose	When using biomaRt in webservice mode and setting verbose to TRUE, the XML query to the webservice will be printed.

**Value**

a dataframe with external gene names, ensembl\_id

**Examples**

```
ensemblGeneNames(c('MOB1A', 'RFLNB', 'SPIC', 'TP53'))
```

---

geneNames	<i>Retrieve gene names from biomaRt</i>
-----------	---

---

**Description**

Retrieve gene names from biomaRt

**Usage**

```
geneNames(ensembl.genes, use.cache = TRUE, verbose = FALSE)
```

**Arguments**

ensembl.genes	character vector with gene names in ensembl_id format
use.cache	Boolean indicating if biomaRt cache should be used
verbose	When using biomaRt in webservice mode and setting verbose to TRUE, the XML query to the webservice will be printed.

**Value**

a dataframe with external gene names, `ensembl_id`

**Examples**

```
geneNames(c('ENSG00000114978', 'ENSG00000166211', 'ENSG00000183688'))
```

---

glmDegree

*GLMNET model penalizing nodes with small degree*

---

**Description**

This function overrides the ‘`trans.fun`’ options in ‘`network.options`’ with the inverse of a degree described in Verissimo et al. (2015) that penalizes nodes with small degree.

**Usage**

```
glmDegree(xdata, ydata, network, network.options = networkOptions(), ...)
```

**Arguments**

<code>xdata</code>	input data, can be a matrix or <code>MultiAssayExperiment</code>
<code>ydata</code>	response data compatible with <code>glmnet</code>
<code>network</code>	type of network, see below
<code>network.options</code>	options to calculate network
<code>...</code>	parameters that <code>glmnet</code> accepts

**Value**

see `glmNetSparse`

**See Also**

`glmNetSparse`

**Examples**

```
xdata <- matrix(rnorm(100), ncol = 5)
glmDegree(xdata, rnorm(nrow(xdata)), 'correlation',
          family = 'gaussian',
          network.options = networkOptions(min.degree = .2))
```



---

glmHub	<i>GLMNET model penalizing nodes with small degree</i>
--------	--

---

**Description**

This function overrides the ‘trans.fun’ options in ‘network.options’ with an heuristic described in Veríssimo et al. that penalizes nodes with small degree.

**Usage**

```
glmHub(xdata, ydata, network, network.options = networkOptions(), ...)
```

**Arguments**

xdata	input data, can be a matrix or MultiAssayExperiment
ydata	response data compatible with glmnet
network	type of network, see below
network.options	options to calculate network
...	parameters that glmnet accepts

**Value**

see glmNetSparse

**See Also**

glmNetSparse

**Examples**

```
xdata <- matrix(rnorm(100), ncol = 5)
glmHub(xdata, rnorm(nrow(xdata)), 'correlation', family = 'gaussian',
       network.options = networkOptions(min.degree = .2))
```

---

glmOrphan	<i>GLMNET model penalizing nodes with high degree</i>
-----------	---

---

**Description**

This function overrides the ‘trans.fun’ options in ‘network.options’ with an heuristic described in Veríssimo et al. that penalizes nodes with high degree.

**Usage**

```
glmOrphan(xdata, ydata, network, network.options = networkOptions(), ...)
```

**Arguments**

xdata	input data, can be a matrix or MultiAssayExperiment
ydata	response data compatible with glmnet
network	type of network, see below
network.options	options to calculate network
...	parameters that glmnet accepts

**Value**

see glmNetSparse

**See Also**

glmNetSparse

**Examples**

```
xdata <- matrix(rnorm(100), ncol = 5)
glmOrphan(xdata, rnorm(nrow(xdata)), 'correlation', family = 'gaussian',
          network.options = networkOptions(min.degree = .2))
```

---

glmSparseNet

*Calculate GLM model with network-based regularization*

---

**Description**

network parameter accepts:

**Usage**

```
glmSparseNet(
  xdata,
  ydata,
  network,
  network.options = networkOptions(),
  experiment.name = NULL,
  ...
)
```

**Arguments**

xdata	input data, can be a matrix or MultiAssayExperiment
ydata	response data compatible with glmnet
network	type of network, see below
network.options	options to calculate network
experiment.name	name of experiment to use as input in MultiAssayExperiment object (only if xdata is an object of this class)
...	parameters that glmnet accepts

**Details**

\* string to calculate network based on data (correlation, covariance) \* matrix representing the network \* vector with already calculated penalty weights (can also be used directly with glmnet)

**Value**

an object just as glmnet

**Examples**

```
xdata <- matrix(rnorm(100), ncol = 20)
glmSparseNet(xdata, rnorm(nrow(xdata)), 'correlation', family = 'gaussian')
glmSparseNet(xdata, rnorm(nrow(xdata)), 'covariance', family = 'gaussian')

#
#
# Using MultiAssayExperiment
# load data
data('miniACC', package="MultiAssayExperiment")
xdata <- miniACC
# TODO asking out x individuals missing values
# build valid data with days of last follow up or to event
event.ix <- which(!is.na(xdata$days_to_death))
cens.ix <- which(!is.na(xdata$days_to_last_followup))
xdata$surv_event_time <- array(NA, nrow(colData(xdata)))
xdata$surv_event_time[event.ix] <- xdata$days_to_death[event.ix]
xdata$surv_event_time[cens.ix] <- xdata$days_to_last_followup[cens.ix]
# Keep only valid individuals
valid.ix <- as.vector(!is.na(xdata$surv_event_time) &
                     !is.na(xdata$vital_status) &
                     xdata$surv_event_time > 0)
xdata.valid <- xdata[, rownames(colData(xdata))[valid.ix]]
ydata.valid <- colData(xdata.valid)[,c('surv_event_time', 'vital_status')]
colnames(ydata.valid) <- c('time', 'status')
glmSparseNet(xdata.valid,
             ydata.valid,
             family = 'cox',
             network = 'correlation',
```

```
experiment.name = 'RNASeq2GeneNorm')
```

---

```
glmSparseNet.options
```

*Constants for 'glmSparseNet' package*


---

### Description

Log level constants and the logger options.

### Usage

```
glmSparseNet.options(..., simplify = FALSE, update = list())
```

### Arguments

...	TODO
simplify	TODO
update	pair list of update to options

### Details

The logging configuration is managed by 'glmSparseNet.options', a function generated by Option-Manager within 'futile.options'.

### Value

futile.options::OptionsManager object

### See Also

futile.options

---

```
hallmarks
```

*Retrieve hallmarks of cancer count for genes*


---

### Description

Retrieve hallmarks of cancer count for genes

### Usage

```
hallmarks(
  genes,
  metric = "count",
  hierarchy = "full",
  generate.plot = TRUE,
  show.message = FALSE
)
```

**Arguments**

genes	gene names
metric	see below
hierarchy	see below
generate.plot	flag to indicate if return object has a ggplot2 object
show.message	flag to indicate if run.cache method shows messages

**Value**

data.frame with choosen metric and hierarchy It also returns a vector with genes that do not have any hallmarks.

See <http://chat.lionproject.net/api> for more details on the metric and hallmarks parameters

To standardize the colors in the gradient you can use `scale_fill_gradientn(limits=c(0,1), colours=topo.colors(3))` to limit between 0 and 1 for cprob and -1 and 1 for npmi

**Examples**

```
hallmarks(c('MOB1A', 'RFLNB', 'SPIC'))

hallmarks(c('MOB1A', 'RFLNB', 'SPIC'), metric = 'cprob')
```

---

heuristicScale	<i>Heuristic function to use in high dimensions</i>
----------------	---

---

**Description**

Heuristic function to use in high dimensions

**Usage**

```
heuristicScale(x, sub.exp10 = -1, exp.mult = -1, sub.exp = -1)
```

**Arguments**

x	vector of values to scale
sub.exp10	value to subtract to base 10 exponential, for example: '10 <sup>0 - sub.exp10</sup> = 1 - sub.exp10'
exp.mult	parameter to multiply exponential, i.e. to have a negative exponential or positive
sub.exp	value to subtract for exponential, for example if x = 0, 'exp(0) - sub.exp = 1 - sub.exp'

**Value**

a vector of scaled values

**Examples**

```
heuristicScale(rnorm(1:10))
```

---

hubHeuristic	<i>Heuristic function to penalize nodes with low degree</i>
--------------	---

---

**Description**

Heuristic function to penalize nodes with low degree

**Usage**

```
hubHeuristic(x)
```

**Arguments**

x                    single value of vector

**Value**

transformed

**Examples**

```
hubHeuristic(rnorm(1:10))
```

---

my.colors	<i>Custom pallete of colors</i>
-----------	---------------------------------

---

**Description**

Custom pallete of colors

**Usage**

```
my.colors(ix = NULL)
```

**Arguments**

ix                    index for a color

**Value**

a color

**Examples**

```
my.colors()  
my.colors(5)
```

---

my.symbols	<i>Custom palette of symbols in plots</i>
------------	---

---

**Description**

Custom palette of symbols in plots

**Usage**

```
my.symbols(ix = NULL)
```

**Arguments**

ix                    index for symbol

**Value**

a symbol

**Examples**

```
my.symbols()  
my.symbols(2)
```

---

networkCorParallel	<i>Calculates the correlation network</i>
--------------------	---

---

**Description**

Calculates the correlation network

**Usage**

```
networkCorParallel(  
  xdata,  
  build.output = "matrix",  
  n.cores = 1,  
  force.recalc.network = FALSE,  
  show.message = FALSE,  
  ...  
)
```

**Arguments**

xdata	base data to calculate network
build.output	if output returns a 'matrix', 'vector' of the upper triu without the diagonal or NULL with any other argument
n.cores	number of cores to be used
force.recalc.network	force recalculation, instead of going to cache
show.message	shows cache operation messages
...	extra parameters for fun

**Value**

depends on build.output parameter

**Examples**

```
n.col <- 6
xdata <- matrix(rnorm(n.col * 4), ncol = n.col)
networkCorParallel(xdata)
```

---

networkCovParallel     *Calculates the covariance network*

---

**Description**

Calculates the covariance network

**Usage**

```
networkCovParallel(
  xdata,
  build.output = "matrix",
  n.cores = 1,
  force.recalc.network = FALSE,
  show.message = FALSE,
  ...
)
```

**Arguments**

xdata	base data to calculate network
build.output	if output returns a 'matrix', 'vector' of the upper triu without the diagonal or NULL with any other argument
n.cores	number of cores to be used
force.recalc.network	force recalculation, instead of going to cache
show.message	shows cache operation messages
...	extra parameters for fun



**Value**

depends on build.output parameter

**Examples**

```
n.col <- 6
xdata <- matrix(rnorm(n.col * 4), ncol = n.col)
networkCovParallel(xdata)
```

---

networkOptions	<i>Setup network options</i>
----------------	------------------------------

---

**Description**

Setup network options, such as using weighted or unweighted degree, which centrality measure to use

**Usage**

```
networkOptions(
  method = "pearson",
  unweighted = TRUE,
  cutoff = 0,
  centrality = "degree",
  min.degree = 0,
  n.cores = 1,
  trans.fun = function(x) { x }
)
```

**Arguments**

method	in case of correlation and covariance, which method to use
unweighted	calculate degree using unweighted network
cutoff	cutoff value in network edges to trim the network
centrality	centrality measure to use, currently only supports degree
min.degree	minimum value that individual penalty weight can take
n.cores	number of cores to use, default to 1
	The trans.fun argument takes a function definition that will apply a transformation to the penalty vector calculated from the degree. This transformation allows to change how the penalty is applied.
trans.fun	see below

**Value**

a list of options

**See Also**

glmOrphan glmDegree

**Examples**

```
networkOptions(unweighted = FALSE)
```

---

orphanHeuristic      *Heuristic function to penalize nodes with high degree*

---

**Description**

Heuristic function to penalize nodes with high degree

**Usage**

```
orphanHeuristic(x)
```

**Arguments**

x                    single value of vector

**Value**

transformed

**Examples**

```
orphanHeuristic(rnorm(1:10))
```

---

protein2EnsemblGeneNames  
*Retrieve ensembl gene ids from proteins*

---

**Description**

Retrieve ensembl gene ids from proteins

**Usage**

```
protein2EnsemblGeneNames(ensembl.proteins, use.cache = TRUE, verbose = FALSE)
```

**Arguments**

ensembl.proteins      character vector with gene names in ensembl\_peptide\_id format

use.cache              Boolean indicating if biomaRt cache should be used

verbose                When using biomaRt in webservice mode and setting verbose to TRUE, the XML query to the webservice will be printed.

**Value**

a dataframe with external gene names, ensembl\_peptide\_id

**Examples**

```
protein2EnsemblGeneNames(c(
  'ENSP00000235382',
  'ENSP00000233944',
  'ENSP00000216911'
))
```

---

run.cache

*Run function and save cache*

---

**Description**

This method saves the function that's being called

**Usage**

```
run.cache(
  fun,
  ...,
  seed = NULL,
  base.dir = NULL,
  cache.prefix = "generic_cache",
  cache.digest = list(),
  show.message = NULL,
  force.recalc = FALSE,
  add.to.hash = NULL
)
```

**Arguments**

fun                    function call name

...                    parameters for function call

seed                   when function call is random, this allows to set seed beforehand

base.dir               directory where data is stored

cache.prefix	prefix for file name to be generated from parameters (...)
cache.digest	cache of the digest for one or more of the parameters
show.message	show message that data is being retrieved from cache
force.recalc	force the recalculation of the values
add.to.hash	something to add to the filename generation

**Value**

the result of fun(...)

**Examples**

```
# [optional] save cache in a temporary directory
#
glmSparseNet::base.dir(tempdir())
glmSparseNet::run.cache(c, 1, 2, 3, 4)
#
# next three should use the same cache
# note, the middle call should be a little faster as digest is not
# calculated
# for the first argument
glmSparseNet::run.cache(c, 1, 2, 3, 4)
glmSparseNet::run.cache(c, a=1, 2, c= 3, 4)

# Using a local folder
# glmSparseNet::run.cache(c, 1, 2, 3, 4, base.dir = "runcache")
```

---

run.cache,function-method

*Run function and save cache*

---

**Description**

Run function and save cache

**Usage**

```
## S4 method for signature ``function``
run.cache(
  fun,
  ...,
  seed = NULL,
  base.dir = NULL,
  cache.prefix = "generic_cache",
  cache.digest = list(),
  show.message = NULL,
```

```

    force.recalc = FALSE,
    add.to.hash = NULL
  )

```

### Arguments

fun	function call name
...	parameters for function call
seed	when function call is random, this allows to set seed beforehand
base.dir	directory where data is stored
cache.prefix	prefix for file name to be generated from parameters (...)
cache.digest	cache of the digest for one or more of the parameters
show.message	show message that data is being retrieved from cache
force.recalc	force the recalculation of the values
add.to.hash	something to add to the filename generation

### Value

the result of fun(...)

### Examples

```

# [optional] save cache in a temporary directory
#
glmSparseNet::base.dir(tempdir())
glmSparseNet::run.cache(c, 1, 2, 3, 4)
#
# next three should use the same cache
# note, the middle call should be a little faster as digest is not
# calculated
# for the first argument
glmSparseNet::run.cache(c, 1, 2, 3, 4)
glmSparseNet::run.cache(c, a=1, 2, c= 3, 4)

# Using a local folder
# glmSparseNet::run.cache(c, 1, 2, 3, 4, base.dir = "runcache")

```

---

save.run.cache

*Saving the cache*

---

### Description

Saving the cache

**Usage**

```
save.run.cache(result, path, compression, show.message)
```

**Arguments**

result	main result to save
path	path to the file to save
compression	compression method to be used
show.message	TRUE to show messages, FALSE otherwise

**Value**

result of save operation

**Examples**

```
glmSparseNet::save.run.cache(
  35, file.path(tempdir(), 'save.run.cache.Rdata'), FALSE, TRUE
)
```

---

separate2GroupsCox      *Separate data in High and Low risk groups (based on Cox model)*

---

**Description**

Draws multiple kaplan meyer survival curves (or just 1) and calculates logrank test

**Usage**

```
separate2GroupsCox(
  chosen.btas,
  xdata,
  ydata,
  probs = c(0.5, 0.5),
  no.plot = FALSE,
  plot.title = "SurvivalCurves",
  xlim = NULL,
  ylim = NULL,
  expand.yzero = FALSE,
  legend.outside = FALSE,
  stop.when.overlap = TRUE,
  ...
)
```



`show.message`*Show messages option in run.cache*

---

**Description**

Show messages option in run.cache

**Usage**

```
show.message(show.message = NULL)
```

**Arguments**

`show.message` boolean indicating to show messages or not

**Value**

the show.message option

**Examples**

```
glmSparseNet::show.message(FALSE)
```

---

`string.network.700.cache`*Cache of protein-protein network, as it takes some time to retrieve and process this will facilitate the vignette building*

---

**Description**

It was filtered with combined\_scores and individual scores below 700 without text-based scores

**Usage**

```
data('string.network.700.cache', package = 'glmSparseNet')
```

**Format**

An object of class dgCMatix with 11033 rows and 11033 columns.

**References**

<https://string-db.org/>



---

stringDBhomoSapiens     *Download protein-protein interactions from STRING DB*

---

**Description**

Download protein-protein interactions from STRING DB

**Usage**

```
stringDBhomoSapiens(version = "11.0", score_threshold = 0, remove.text = TRUE)
```

**Arguments**

version	version of the database to use
score_threshold	remove scores below threshold
remove.text	remove text mining-based scores

**Value**

a data.frame with rows representing an interaction between two proteins, and columns the count of scores above the given score\_threshold

**Examples**

```
stringDBhomoSapiens(score_threshold = 800)
```

---

tempdir.cache     *Temporary directory for runCache*

---

**Description**

Temporary directory for runCache

**Usage**

```
tempdir.cache()
```

**Value**

a path to a temporary directory used by runCache

---

write.readme	<i>Write a file in run-cache directory to explain the origin</i>
--------------	--

---

**Description**

Write a file in run-cache directory to explain the origin

**Usage**

```
write.readme(base.dir)
```

**Arguments**

base.dir      directory where to build this file

**Value**

the path to the file it has written

**Examples**

```
glmSparseNet::write.readme(tempdir())
```

# Index

## \* data

- string.network.700.cache, 40
- .calcPenalty, 3
- .degreeGeneric, 4
- .glmSparseNetPrivate, 5
- .networkGenericParallel, 6
- .networkWorker, 7
  
- balanced.cv.folds, 7
- base.dir, 8
- biomart.load, 8
- build.function.digest, 9
- buildLambda, 10
- buildStringNetwork, 11
  
- cache.compression, 12
- calculate.combined.score, 12
- calculate.result, 13
- create.directory.for.cache, 14
- curl.workaround, 14
- cv.glmDegree, 15
- cv.glmHub, 16
- cv.glmOrphan, 17
- cv.glmSparseNet, 18
  
- degreeCor, 19
- degreeCov, 20
- digest.cache, 21
- downloadFileLocal, 22
  
- ensemblGeneNames, 23
  
- geneNames, 23
- glmDegree, 24
- glmHub, 25
- glmOrphan, 25
- glmSparseNet, 26
- glmSparseNet.options, 28
  
- hallmarks, 28
- heuristicScale, 29

- hubHeuristic, 30
  
- my.colors, 30
- my.symbols, 31
  
- networkCorParallel, 31
- networkCovParallel, 32
- networkOptions, 33
  
- orphanHeuristic, 34
  
- protein2EnsemblGeneNames, 34
  
- run.cache, 35
- run.cache,function-method, 36
  
- save.run.cache, 37
- separate2GroupsCox, 38
- show.message, 40
- string.network.700.cache, 40
- stringDBhomoSapiens, 41
  
- tempdir.cache, 41
  
- write.readme, 42