Package ‘Corbi’

December 10, 2020

Version 0.6-1
Title Collection of Rudimentary Bioinformatics Tools
Description Provides a bundle of basic and fundamental bioinformatics tools, such as network querying and alignment, subnetwork extraction and search, network biomarker identification.
ByteCompile TRUE
Depends R (>= 3.0.2)
Imports Matrix, MASS, stats, CRF, igraph
Suggests knitr, rmarkdown, BiocParallel, matrixcalc, mpmi, fitdistrplus
VignetteBuilder knitr
License GPL (>= 2)

BugReports https://github.com/wulingyun/Corbi/issues
URL https://github.com/wulingyun/Corbi
RoxygenNote 7.1.1
Encoding UTF-8
Author Ling-Yun Wu [aut, cre], Qiang Huang [aut], Duanchen Sun [aut]
Maintainer Ling-Yun Wu <wulingyun@gmail.com>
Repository CRAN
Repository/R-Forge/Project corbi
Repository/R-Forge/Revision 43
Repository/R-Forge/DateTimeStamp 2020-12-10 03:17:17
Date/Publication 2020-12-10 09:00:17 UTC
NeedsCompilation yes
Corbi-package

Description

This package provides a bundle of basic and fundamental bioinformatics tools.
Details

These bioinformatics tools are developed by WuLab at Academy of Mathematics and Systems Science, Chinese Academy of Sciences.

Network querying and alignment:

- **net_query** Network querying method based on conditional random fields
- **net_query_batch** Batch processing version of **net_query**
- **net_align** Network alignment method based on conditional random fields

Subnetwork extraction and search:

- **get_subnets** Enumerate all subnetworks of limited size
- **extend_subnets** Extend subnetworks from smaller subnetworks
- **best_subnets** Search best subnetworks that maximize given objective function

Biomarker identification:

- **markrank** Biomarker identification and prioritization by integrating gene expression with biomolecular network

Differential expression analysis:

- **netDEG** Sample specific differential expression analysis

Data normalization:

- **URG_getFactor** Gene expression data normalization by the uniform ratio graph method

References


The best subnetworks

**Description**

Search best subnetworks that maximize given objective functions.

**Usage**

```r
best_subnets(
  func,
  net.matrix,
  max.size = 10,
  exhaust.size = 5,
  max.top = 10000
)
```

**Arguments**

- `func`: The objective function to maximize
- `net.matrix`: The adjacent matrix of network
- `max.size`: The maximal size of subnetworks
- `exhaust.size`: The maximal size of subnetworks that use exhaustive searching strategy
- `max.top`: The maximal number of top candidates kept for evaluation of next size, used in heuristic searching strategy

**Details**

Enumerate and search the best subnetworks that maximize given objective function. If the size of subnetworks <= `exhaust.size`, exact exhaustive searching is applied, otherwise, heuristic searching algorithm is used.

**Value**

A list with the following two components:

- `subnets`: The list of top subnetworks in different sizes
- `obj.values`: The list of objective values of corresponding subnetworks

**See Also**

- `get_subnets`, `extend_subnets`
### Examples

```r
library(Corbi)
net <- matrix(FALSE, nrow=10, ncol=10)
net[sample.int(100, 20)] <- TRUE
net <- net | t(net)
func <- function(subnet) max(subnet) - min(subnet)
result <- best_subnets(func, net, 5)
```

### column

**Extract a column from a matrix**

#### Description

Extract a specified column from a sparse matrix rapidly

#### Usage

```r
column(m, i)
```

#### Arguments

- `m`: The matrix
- `i`: The column index

#### Details

This function implements faster column extraction algorithm for the `CsparseMatrix` class in the package `Matrix`.

#### Value

This function will return the specified column as a vector of corresponding type.

### extend_subnets

**Extend subnetworks from smaller subnetworks**

#### Description

Extend subnetworks by pairwise overlapping two sets of smaller subnetworks.

#### Usage

```r
extend_subnets(subnet1, subnet2, size = 0)
```
Arguments

subnet1  The matrix representing the first set of subnetworks
subnet2  The matrix representing the second set of subnetworks
size     The desired size of extended subnetworks

Details

Enumerate all possible subnetworks of desired size by pairwise overlapping two sets of subnetworks of size $s_1$ and $s_2$. The desired size should be between $\max(s_1, s_2)+1$ and $s_1+s_2-1$. Invalid desired size will be replaced by the minimum allowed value $\max(s_1, s_2)+1$.

Value

A matrix represents the extended subnetworks, in which each row represents a subnetwork.

Examples

```r
library(Corbi)
net <- matrix(FALSE, nrow=10, ncol=10)
net[sample.int(100, 20)] <- TRUE
net <- net | t(net)
subnets <- get_subnets(net, 3)
subnets[[4]] <- extend_subnets(subnets[[3]], subnets[[2]], 4)
```

get_adjusted_deg_diff  Calculate adjusted degree differences for given network

Description

Calculate the adjusted degree differences for all genes in the given network.

Usage

```r
get_adjusted_deg_diff(net, log.expr.val, scale.degree = FALSE, p = 0.5)
```

Arguments

net          The binary adjacent matrix of differential expression ratio network.
log.expr.val Numeric vector containing the logarithmic scale gene expression values.
scale.degree Logical variable indicating whether the degree values are scaled according to the dropout rate.
p            The parameter for calculating the adjusted degree differences.
get_diff_ratio_net

Value

This function will return a list with the following components:

- **diff**: A numeric vector containing the adjusted degree differences of all genes.
- **degree**: A list containing the raw degree differences and sums of all genes.

---

get_diff_ratio_net  Construct differential expression ratio network

---

Description

Construct the differential expression ratio network for a single sample.

Usage

```r
get_diff_ratio_net(
  ref.ratio.dist,
  expr.val,
  log.expr = FALSE,
  scale.degree = FALSE
)
```

Arguments

- **ref.ratio.dist**: The expression ratio distribution profile returned by `get_ratio_distribution` or `get_ratio_distribution2`.
- **expr.val**: Numeric vector of gene expression values in the sample.
- **log.expr**: Logical variable indicating whether the input expression vector is in logarithmic scale.
- **scale.degree**: Logical variable indicating whether the degree values are scaled according to the dropout rate.

Value

This function will return a list with the following components:

- **net**: The binary adjacent matrix of differential expression ratio network.
- **diff**: A numeric vector containing the adjusted degree differences of all genes.
- **degree**: A list containing the raw degree differences and sums of all genes.
Description

Calculate the lower and upper quantiles of expression ratios for each pair of genes, and estimate the parameters of negative binomial distribution from reference expression data.

Usage

```r
get_ratio_distribution(
  ref.expr.matrix,
  p.edge = 0.1,
  log.expr = FALSE,
  scale.degree = FALSE,
  use.parallel = FALSE
)
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ref.expr.matrix</td>
<td>The reference expression matrix. Each row represents a gene and each column represents a sample.</td>
</tr>
<tr>
<td>p.edge</td>
<td>The expected probability of edges in the expression ratio network for a normal sample.</td>
</tr>
<tr>
<td>log.expr</td>
<td>Logical variable indicating whether the input expression matrix is in logarithmic scale.</td>
</tr>
<tr>
<td>scale.degree</td>
<td>Logical variable indicating whether the degree values are scaled according to the dropout rate.</td>
</tr>
<tr>
<td>use.parallel</td>
<td>Logical variable indicating to use the BiocParallel package to accelerate computation.</td>
</tr>
</tbody>
</table>

Value

This function will return a list with the following components:

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LB</td>
<td>A numeric matrix with element [i,j] represents the lower quantile of expression ratios for gene pairs (i,j).</td>
</tr>
<tr>
<td>NB</td>
<td>A numeric vector with two elements: size and mu, which are the estimated parameters of negative binomial distribution.</td>
</tr>
<tr>
<td>p.edge</td>
<td>The used input parameter p.edge.</td>
</tr>
</tbody>
</table>

get_ratio_distribution

Calculate expression ratio distribution
get_ratio_distribution2

*Calculate expression ratio distribution*

**Description**

Calculate the lower and upper quantiles of expression ratios after trimming the extreme values, and estimate the parameters of negative binomial distribution from reference expression data.

**Usage**

```r
get_ratio_distribution2(
  ref.expr.matrix,
  p.edge = 0.1,
  p.trim = 0.3,
  log.expr = FALSE,
  scale.degree = FALSE,
  use.parallel = FALSE
)
```

**Arguments**

- `ref.expr.matrix`: The reference expression matrix. Each row represents a gene and each column represents a sample.
- `p.edge`: The expected probability of edges in the expression ratio network for a normal sample.
- `p.trim`: The percentage of lower or upper extreme values to be trimmed from the expression ratios for each pair of genes.
- `log.expr`: Logical variable indicating whether the input expression matrix is in logarithmic scale.
- `scale.degree`: Logical variable indicating whether the degree values are scaled according to the dropout rate.
- `use.parallel`: Logical variable indicating to use the BiocParallel package to accelerate computation.

**Value**

This function will return a list with the following components:

- `LB`: A numeric matrix with element [i,j] represents the lower quantile of trimmed expression ratios for gene pairs (i, j).
- `NB`: A numeric vector with two elements: `size` and `mu`, which are the estimated parameters of negative binomial distribution.
- `p.edge`: The used input parameter `p.edge`.
- `p.trim`: The used input parameter `p.trim`.
**get_ratio_variance**

*Calculate expression ratio variances*

**Description**

Calculate the variances of expression ratios for each pair of genes.

**Usage**

```r
get_ratio_variance(expr.matrix, log.expr = FALSE)
```

**Arguments**

- `expr.matrix`: The expression matrix. Each row represents a gene and each column represents a sample.
- `log.expr`: Logical variable indicating whether the input expression matrix is in logarithmic scale.

**Value**

This function will return a numeric matrix with element \([i,j]\) represents the variance of expression ratios for gene pairs \((i, j)\).

**get_shortest_distances**

*Calculate shortest distances of unweighted network*

**Description**

Calculate all pairs of shortest distances of unweighted network

**Usage**

```r
get_shortest_distances(    net.matrix,    source.nodes = rep_len(TRUE, dim(net.matrix)[1]))
```

**Arguments**

- `net.matrix`: Logical adjacency matrix of given unweighted network
- `source.nodes`: Logical vector to indicate the source nodes that need to calculate the shortest distances
get_subnets

Details
This function calculates all pairs of shortest distances of unweighted network by using breadth-first-search (BFS) algorithm.

Value
This function will return the shortest distance matrix, where the element $[i,j]$ is the shortest distance between node $i$ and $j$. Value -1 means unreachable. If source.nodes[i] equals FALSE, the shortest distance from $i$ to other nodes will not be calculated and the row $i$ will be all -1.

get_subnets  All subnetworks of limited size

Description
Enumerate all subnetworks of size $\leq \max\_size$ from given network.

Usage
get_subnets(net.matrix, max.size = 2)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>net.matrix</td>
<td>The adjacent matrix of network</td>
</tr>
<tr>
<td>max.size</td>
<td>The maximal size of subnetworks</td>
</tr>
</tbody>
</table>

Value
A list of generated subnetworks, with element $i$ corresponds the subnetworks of size $i$. Each element is a matrix, in which each row represents a subnetwork.

Examples

```R
library(Corbi)
net <- matrix(FALSE, nrow=10, ncol=10)
net[sample.int(100, 20)] <- TRUE
net <- net | t(net)
subnets <- get_subnets(net, 3)
```
### kappa_score  
*Cohen's kappa score*

**Description**

Calculate Cohen’s kappa score for two vectors.

**Usage**

```r
kappa_score(x1, x2)
```

**Arguments**

- `x1`: The first logical vector
- `x2`: The second logical vector

**Details**

This function calculates Cohen’s kappa score for two logical vectors.

**Value**

The Cohen’s kappa score

---

### make_DEG_data  
*Simulate differentially expressed gene data (Gaussian)*

**Description**

Generate differentially expressed gene (DEG) data from Gaussian distribution.

**Usage**

```r
make_DEG_data(
  n.genes,
  n.samples.A,
  n.samples.B,
  exp.mean = 8,
  exp.sd = 2,
  alpha = 0.2,
  size.factor.sd = 0.1,
  ...
)
```
Arguments

- **n.genes**: The total number of genes in the simulated data.
- **n.samples.A**: The number of samples in the group A.
- **n.samples.B**: The number of samples in the group B.
- **exp.mean**: The mean of log-normal distribution that determines gene-specific expression mean.
- **exp.sd**: The standard deviation of log-normal distribution that determines gene-specific expression means.
- **alpha**: The dispersion ratio of gene-specific expression standard deviation to mean.
- **size.factor.sd**: The standard deviation of size factors for samples.
- **...**: The parameters passed to function `make_DEG_pattern`.

Details

The expression values of each gene are assumed following a Gaussian distribution with gene-specific mean, which follows a log-normal distribution. The size factor for each sample follows a Gaussian distribution with zero mean and specific standard deviation. The heterogeneity of gene expression data is simulated by using the function `make_DEG_pattern`.

Value

This function will return a list with the following components:

- **DEG**: The matrix of simulated DEG pattern, which is generated by `make_DEG_pattern`.
- **countsA**: The expression matrix of group A. Each row represents a gene and each column represents a sample.
- **countsB**: The expression matrix of group B. Each row represents a gene and each column represents a sample.

---

**make_DEG_data2**  
*Simulate differentially expressed gene data (Negative binomial)*

Description

Generate differentially expressed gene (DEG) data from negative binomial distribution.

Usage

```r
make_DEG_data2(
  n.genes,
  n.samples.A,
  n.samples.B,
  exp.mean = 8,
  exp.sd = 2,
```
dispersion = NULL, size.factor.sd = 0.1, ... )

**Arguments**

- `n.genes` The total number of genes in the simulated data.
- `n.samples.A` The number of samples in the group A.
- `n.samples.B` The number of samples in the group B.
- `exp.mean` The mean of log-normal distribution that determines gene-specific expression mean.
- `exp.sd` The standard deviation of log-normal distribution that determines gene-specific expression mean.
- `dispersion` The dispersion parameter for negative binomial distribution. The default values are determined by the expression mean.
- `size.factor.sd` The standard deviation of size factors for samples.
- `...` The parameters passed to function `make_DEG_pattern`.

**Details**

The expression values of each gene are assumed following a negative binomial distribution with gene-specific mean, which follows a log-normal distribution. The size factor for each sample follows a Gaussian distribution with zero mean and specific standard deviation. The heterogeneity of gene expression data is simulated by using the function `make_DEG_pattern`.

**Value**

This function will return a list with the following components:

- `DEG` The matrix of simulated DEG pattern, which is generated by `make_DEG_pattern`.
- `countsA` The expression matrix of group A. Each row represents a gene and each column represents a sample.
- `countsB` The expression matrix of group B. Each row represents a gene and each column represents a sample.

---

**Description**

Generate complicated differentially expressed gene (DEG) pattern to simulate varied degree of heterogeneity.
Usage

```r
make_DEG_pattern(
  n.genes,
  n.samples,
  fold.change = 2,
  gene.rate = 0.3,
  sample.rate = 1,
  active.rate = 1,
  up.rate = 0.5
)
```

Arguments

- `n.genes`: The total number of genes in the simulated data.
- `n.samples`: The total number of samples in the simulated data.
- `fold.change`: The fold change level of DEGs.
- `gene.rate`: The proportion of DEGs to all genes.
- `sample.rate`: The proportion of abnormal samples to all samples.
- `active.rate`: The probability that a DEG is truly differentially expressed in an abnormal sample.
- `up.rate`: The proportion of up-regulated DEGs to all DEGs.

Details

The heterogeneity of gene expression pattern is mainly controlled by two parameters: `sample.rate` and `active.rate`. If both parameters are equal to 1, the gene expression pattern will be homogeneous, otherwise heterogeneous.

Value

This function will return a list with the following components:

- `FC`: The matrix of simulated fold changes. Each row represents a gene and each column represents a sample.
- `gene`: The vector of gene status: 1 for up-regulated, -1 for down-regulated, and 0 for normal genes.
- `sample`: The vector of sample status: 1 for abnormal, and 0 for normal samples.
MarkRank is a novel proposed network-based model, which can identify the cooperative biomarkers for heterogeneous complex disease diagnoses.

Usage

```r
markrank(
    dataset,
    label,
    adj_matrix,
    alpha = 0.8,
    lambda = 0.2,
    eps = 1e-10,
    E_value = NULL,
    trace = TRUE,
    d = Inf,
    Given_NET2 = NULL
)
```

Arguments

- **dataset**: The microarray expression matrix of related disease. Each row represents a sample and each column represents a gene.
- **label**: The 0-1 binary phenotype vector of dataset samples. The size of label must accord with the sample number in dataset.
- **adj_matrix**: The 0-1 binary adjacent matrix of a connected biological network. Here the node set should be the same order as the gene set in expression matrix.
- **alpha**: The convex combination coefficient of network effect and prior information vector $E_value$. The range of alpha is in $[0, 1]$. A larger alpha will lay more emphasis on the network information. The default value is 0.8.
- **lambda**: In the random walk-based iteration, matrix $A_1$ reflects the structure information of the biological network, whereas matrix $A_2$ reflects the cooperative effect of gene combinations. Parameter lambda is the convex combination coefficient of two network effects. The range of lambda is in $[0, 1]$. A larger lambda will lay more emphasis on the $A_1$. The default value is 0.2.
- **eps**: The stop criteria for the iterative solution method. The default value is 1e-10.
- **E_value**: A vector containing the prior information about the importance of nodes. Default is the absolute Pearson correlation coefficient (PCC).
- **trace**: Localal variable indicated whether tracing information on the progress of the gene cooperation network construction is produced.
Threshold for simplifying the G_2 computation. Only the gene pairs whose shortest distances in PPI network are less than $d$ participate in the G_2 computation. The default value is Inf.

Given_NET2 Whether a computed cooperation network is given for tuning parameter. See Details for a more specific description.

Details

MarkRank is a network-based biomarker identification method to prioritize disease genes by integrating multi-source information including the biological network, e.g. protein-protein interaction (PPI) network, the prior information about related diseases, and the discriminative power of cooperative gene combinations. MarkRank shows that explicit modeling of gene cooperative effects can greatly improve biomarker identification for complex disease, especially for diseases with high heterogeneity.

MarkRank algorithm contains mainly two steps: 1) The construction of gene cooperation network $G_2$ and 2) a random walk based iteration procedure. The following descriptions will help the users to using markrank more convenient:

1) As for the construction of the gene cooperation network, we suggest the user to set trace=TRUE to output the G_2 computation process. The G_2 construction step finished if the output number is identical to the gene number of the input expression matrix. The parameter $d$ introduced the structure information of used biological network to facilitate the construction of G_2, only the gene pairs whose shortest distances in network are less than $d$ participate the G_2 computation. We suggest $d$=Inf, the default value, to fully use the information of expression matrix. If the user given a preset $d$, the distance matrix of input network dis will be returned.

2) MarkRank uses a random-walk based iteration procedure to score each gene. The detailed formula is:

$$\text{score} = \alpha [(\lambda A_1 + (1-\lambda) A_2) * \text{score} + (1-\alpha) E_{\text{value}}]$$

The users could set an appropriate parameter settings in their practical application. Our suggested value is $\alpha$=0.8 and $\lambda$=0.2. The model input parameter combinations and iteration steps will be returned in output components initial_pars and steps, respectively. Because the iteration step is separate with the cooperation network construction, the user can use the parameter Given_NET2 to tune the model parameters. In detail, the user could set

$$\text{Given_NET2} = \text{result$\text{NET2}$}$$

in markrank input to avoid the repeated computation of G_2, where the object result is the returned variable of markrank function.

3) The final MarkRank score for each gene is in output score. The users could sort this result and use the top ranked genes for further analysis.

Value

This function will return a list with the following components:

- score The vector of final MarkRank scores for each gene.
- steps The final iteration steps in random walk based scoring procedure.
- NET2 The weighted adjacent matrix of gene cooperation network.
- initial_pars The initial/input parameter values used in MarkRank.
dis The pairwise distance matrix of input network. This variable will be Null if input d=Inf.

References


netDEG: Differentially expressed gene identification method

Description

Perform netDEG for two group samples.

Usage

```r
netDEG(
  ref.expr.matrix,
  expr.matrix,
  p.edge = 0.1,
  summarize = c("gene", "sample"),
  summarize.method = c("sumlog", "sumlog"),
  summarize.shrink = c(Inf, Inf),
  log.expr = FALSE,
  zero.as.dropout = TRUE,
  scale.degree = TRUE,
  use.parallel = FALSE
)
```

Arguments

- `ref.expr.matrix` The reference expression matrix. Each row represents a gene and each column represents a sample.
- `expr.matrix` The test expression matrix. Each row represents a gene and each column represents a sample.
- `p.edge` The expected probability of edges in the expression ratio network for a normal sample.
- `summarize` Character vector indicating how to summarize the results. Available methods are c("gene","sample").
- `summarize.method` Character vector indicating the methods used to summarize the results. See `p_combine`.
- `summarize.shrink` Character vector indicating the methods used to summarize the results. See `p_combine`.
- `log.expr` Logical. If TRUE, log-transform the expression data. Default is FALSE.
- `zero.as.dropout` Logical. If TRUE, treat zeros as dropout. Default is TRUE.
- `scale.degree` Logical. If TRUE, scale the degree of each node. Default is TRUE.
- `use.parallel` Logical. If TRUE, use parallel computation. Default is FALSE.
netDEG_pvalue

summarize.shrink
- Numeric vector indicating the shrink parameter to summarize the results. See p_combine.

log.expr
- Logical variable indicating whether the input expression matrix is in logarithmic scale.

zero.as.dropout
- Logical variable indicating whether the zero expressions are regarded as dropouts.

scale.degree
- Logical variable indicating whether the degree values are scaled according to the dropout rate.

use.parallel
- Logical variable indicating to use the BiocParallel package to accelerate computation.

Value
- This function will return a list with the following components:

  up
  - A numeric matrix with same dimension as expr.matrix, containing the p-values of up-regulation test.

  down
  - A numeric matrix with same dimension as expr.matrix, containing the p-values of down-regulation test.

  twoside
  - A numeric matrix with same dimension as expr.matrix, containing the p-values of twoside test.

  rev
  - A list containing the reverse comparison results, containing three components: up, down, and twoside. Available if the gene method is specified in summarize argument.

  gene
  - A list containing the gene-wise summaried results, containing three components: up, down, and twoside. Available if the gene method is specified in summarize argument.

  sample
  - A list containing the sample-wise summaried results, containing three components: up, down, and twoside. Available if the sample method is specified in summarize argument.

---

netDEG_pvalue  Calculate netDEG p-values

Description
- Perform the single or two side tests and calculate the p-values.

Usage
netDEG_pvalue(ref.ratio.dist, expr.val, log.expr = FALSE, scale.degree = FALSE)
Arguments

- ref.ratio.dist: The expression ratio distribution profile returned by `get_ratio_distribution` or `get_ratio_distribution2`.
- expr.val: Numeric vector of gene expression values in the sample.
- log.expr: Logical variable indicating whether the input expression vector is in logarithmic scale.
- scale.degree: Logical variable indicating whether the degree values are scaled according to the dropout rate.

Value

This function will return a list with the following components:

- up: A numeric vector containing the p-values of up-regulation test.
- down: A numeric vector containing the p-values of down-regulation test.
- twoside: A numeric vector containing the p-values of twoside test.

Description

Find the maximal matching subnetworks from a target network for a query network based on the conditional random fields (CRF) model.

Usage

```r
net_align(
  query.net, target.net, node.sim, query.type = 4, delta.d = 1e-10, delta.c = 0.5, delta.e = 1, delta.s = 1, output = "result.txt"
)
```

Arguments

- query.net: The input file name of the query network.
- target.net: The input file name of the target network.
- node.sim: The input file name of the node similarity scores between the query network and the target network.
query.type  The querying network type: 1 - general, 2 - chain, 3 - tree, 4 - heuristic.
delta.d    The parameter delta.d is a parameter for deletions.
delta.c    The parameter delta.c is a parameter for consecutive deletions.
delta.e    The parameter delta.e is a parameter for single deletion.
delta.s    The parameter delta.s is a parameter for insertions.
output     The suffix of output file name. The output contains two files in the working directory. One is the matching nodes and edges between query network and target network, the other is the unique matching node pairs.

Details

This is an approach for network alignment problem based on conditional random field (CRF) model which uses the node similarity and structure information equally. This method is based on our network querying method net_query. This method uses an iterative strategy to get the one-to-one map between the query network and target network.

More details can be seen in net_query.

References


Examples

```r
## Not run:
library(Corbi)

## An example: "querynet.txt", "targetnet.txt", "nodesim.txt" are
## three input files in the working directory
net_align("querynet.txt", "targetnet.txt", "nodesim.txt")

## End(Not run)
```

---

**net_query**

Network querying method based on conditional random fields

**Description**

Find the best matching subnetworks from a large target network for small query networks based on the conditional random fields (CRF) model.
Usage

```r
net_query(
  query.net, 
  target.net, 
  node.sim, 
  query.type = 4, 
  delta.d = 1e-10, 
  delta.c = 0.5, 
  delta.e = 1, 
  delta.s = 1, 
  output = "result.txt"
)
```

```r
net_query_batch(
  query.nets, 
  target.net, 
  node.sim, 
  query.type = 4, 
  delta.d = 1e-10, 
  delta.c = 0.5, 
  delta.e = 1, 
  delta.s = 1, 
  output = "result.txt"
)
```

Arguments

- `query.net`: The input file name of the query network.
- `target.net`: The input file name of the target network.
- `node.sim`: The input file name of the node similarity scores between the query network and the target network.
- `query.type`: The querying network type: 1 - general, 2 - chain, 3 - tree, 4 - heuristic.
- `delta.d`: The parameter delta.d is a parameter for deletions.
- `delta.c`: The parameter delta.c is a parameter for consecutive deletions.
- `delta.e`: The parameter delta.e is a parameter for single deletion.
- `delta.s`: The parameter delta.s is a parameter for insertions.
- `output`: The suffix of output file name.
- `query.nets`: The vector of input file names of the query networks.

Details

This is an approach for network querying problem based on conditional random field (CRF) model which can handle both undirected and directed networks, acyclic and cyclic networks, and any number of insertions/deletions.

When querying several networks in the same target network, `net_query_batch` will save much time.
• query.net: The query network file is written as follows:
  v1 v2 v3 v4 v5
  v3 v4
  ...
  where v1, v2, v3, v4, v5 ... are the nodes’ names and each line indicates there are edges
  between the first node and other nodes in the line. For example, the first line denotes 4 edges:
  (v1, v2), (v1, v3), (v1, v4), and (v1, v5).
• target.net: The format of this file is the same as the query network file.
• node.sim: This similarity file’s format is as follows:
  v1 V1 s1
  v1 V2 s2
  ...
  v1 is the node from the query network, V1 is the node from the target network, s1 is the
  similarity score between the node v1 and V1, and so on.
• query.type: If query.type = 1, the loopy belief propagation (LBP) algorithm will be applied,
  which is an approximate algorithm for a general graph with loops. If the query is a chain or
  tree, there are exact algorithms. Set query.type = 2 when the query is a chain, and query.type
  = 3 when the query is a tree. The heuristic algorithm will be used when query.type = 4, which
  will try the exact algorithm (junction tree algorithm) first and resort to LBP algorithm when
  the exact algorithm failed. The default value is 4.
• delta.d: The smaller delta.d is, the heavier penalty for deletions.
• delta.c: The smaller delta.c is, the heavier penalty for consecutive deletions.
• delta.e: The smaller delta.e is, the heavier penalty for single deletion.
• delta.s: The larger delta.s indicates heavier penalty for insertions.

References

Qiang Huang, Ling-Yun Wu, and Xiang-Sun Zhang. An Efficient Network Querying Method Based

Examples

## Not run:
library(Corbi)

## An example: "querynet.txt", "targetnet.txt", "nodesim.txt" are
## three input files in the working directory
net_query("querynet.txt", "targetnet.txt", "nodesim.txt”, query.type=3)

## End(Not run)

## Not run:
## Batch example
net_query_batch(c("querynet.txt", "querynet2.txt"),
  "targetnet.txt", "nodesim.txt", query.type=3)
nnzero

The number of non-zero values of a submatrix

Description
Retuen the number of non-zero values of the specified submatrix of a given sparse matrix rapidly

Usage
nnzero(m, rows = 1:dim(m)[1], cols = 1:dim(m)[2])

Arguments
m
The matrix
rows
The integer vector of row index(es) or logical vector indicated the selected rows
cols
The integer vector of column index(es) or logical vector indicated the selected cols

Details
This function implements faster calculation algorithm for the CsparseMatrix and RsparseMatrix class in the package Matrix.

Value
This function will return the number of non-zero values in the specified submatrix.

pmultihyper

The Multivariate Hypergeometric Distribution

Description
The distribution function for the weighted sums of multivariate hypergeometric distribution

Usage
pmultihyper(x, k, m, w)
**pmultinom**

**Arguments**

- **x**: The quantile of weighted sum.
- **k**: The total number of balls drawn from the urn.
- **m**: Integer non-negative vector of length N, containing the number of balls of each color in the urn. N is the number of colors.
- **w**: Numeric non-negative vector of length N, specifying the weight of balls of each color.

**Details**

This function gives the distribution function for the weighted sums of multivariate hypergeometric distribution by recursively calling the hypergeometric distribution density function `dhyper`.

**Value**

This function will return the probability of $P(X \leq x)$.

**See Also**

`dhyper`

---

### pmultinom

**Description**

The distribution function for the weighted sums of multinomial distribution

**Usage**

`pmultinom(x, k, m, w)`

**Arguments**

- **x**: The quantile of weighted sum.
- **k**: The total number of balls drawn from the urn.
- **m**: Numeric non-negative vector of length N, specifying the probability for drawing the ball of each color; is internally normalized to sum 1. Infinite and missing values are not allowed. N is the number of colors.
- **w**: Numeric non-negative vector of length N, specifying the weight of balls of each color.

**Details**

This function gives the distribution function for the weighted sums of multinomial distribution by recursively calling the binomial distribution density function `dbinom`.
Value

This function will return the probability of \( P(X \leq x) \).

See Also

dbinom, dmultinom, rmultinom

p_combine

Calculate combined p-value

Description

Combine the statistical significance results from several independent tests by using one of several methods.

Usage

p_combine(p, method = "sumlog", shrink = Inf)

Arguments

- **p**: the numeric vector containing the p-values need to combine.
- **method**: the method use to combine the p-values, can be "sumlog" (Fisher’s method), "sumz" (Stouffer’s method).
- **shrink**: the number of p-values used in calculation, which are uniform selected from original p-value vector.

Value

This function will return a list with the following components:

- **p**: The combined p-value.
- **v**: The value of statistic.
  
  Use "sumlog" method:

  - **chisq**: The value of chi-squared statistic.
  
  - **df**: The degrees of freedom of chi-squared distribution.
  
  Use "sumz" method:

- **z**: The value of sum z statistic.
**read_net**

*Read network information from text file*

**Description**

Read the network information from a text file with specific format.

**Usage**

```r
read_net(file)
```

**Arguments**

- `file` The name of text file

**Details**

This function reads the network information from a text file with specific format: each line contains two strings separated by spaces, which correspond to the names of two end points of one edge in the network.

**Value**

A list with the following components:

- `size` The number of network nodes
- `node` The vector of network node names
- `matrix` The logical adjacency matrix

**See Also**

- `write_net`

---

**rmultihyper**

*The Multivariate Hypergeometric Distribution*

**Description**

Generate random variables for the multivariate hypergeometric distribution

**Usage**

```r
rmultihyper(n, k, m)
```
Arguments

n  The number of observations.
k  The total number of balls drawn from the urn.
m  The integer vector containing the number of balls of each color in the urn. Length of vector is the number of colors.

Details

This function generates random variables for the multivariate hypergeometric distribution by iteratively calling hypergeometric random variable generator rhyper.

Value

This function will return a matrix of length(m) rows and n columns, and each column contains the number of balls of each color drawn from the urn.

See Also

rhyper

simulate_dropout  Simulate dropout expression data

Description

Generate the expression data with desired dropout rate

Usage

simulate_dropout(counts, dropout.rate = 0, dropout.rate.sd = 0.1)

Arguments

counts  expression matrix where each row is a gene and each column is a sample.
dropout.rate  the desired average dropout rate of all samples.
dropout.rate.sd  the desired standard deviation of dropout rate among samples.

Details

The dropout event is modelled by a logistic distribution such that the low expression genes have higher probability of dropout. The expression value of genes in a sample are randomly set to zero with probabilities associated with their true expression values until the desired dropout rate for that sample is meet.
simulate_dropout2

Value

This function will return a list with the following components:

- `counts` The modified expression matrix with the same dimension as input `counts`.
- `original.counts` The original input expression matrix.
- `dropout` The binary matrix indicating where the dropout events happen.

References


---

simulate_dropout2  Simulate dropout expression data

Description

Generate the expression data with desired dropout rate range

Usage

`simulate_dropout2(counts, min.rate = 0, max.rate = 0.8)`

Arguments

- `counts` expression matrix where each row is a gene and each column is a sample.
- `min.rate` the minimum dropout rate of all samples.
- `max.rate` the maximum dropout rate of all samples.

Details

The dropout event is modelled by a logistic distribution such that the low expression genes have higher probability of dropout. The expression value of genes in a sample are randomly set to zero with probabilities associated with their true expression values until the desired dropout rate for that sample is meet.

Value

This function will return a list with the following components:

- `counts` The modified expression matrix with the same dimension as input `counts`.
- `original.counts` The original input expression matrix.
- `dropout` The binary matrix indicating where the dropout events happen.
References


simulate_sample_groups

Simulate sample groups from given samples with labels

Description

Generate sample groups with desired labels and sizes from given sample labels.

Usage

simulate_sample_groups(labels, groups, sizes, replace = FALSE)

Arguments

labels a vector containing the label of each sample in the pool.
groups a vector containing the desired label of samples in each group. The label must be available in the sample pool provided by labels.
sizes integer vector indicating the desired number of samples in each group. The length must be either one or the same as groups.
replace logical variable indicating whether sampling is with replacement.

Value

This function will return a list with the same length as groups. Each component is a vector containing the indexes of samples that are sampled for the corresponding group.

submatrix

Extract a submatrix from a matrix

Description

Extract a specified submatrix from a sparse matrix rapidly

Usage

submatrix(m, rows, cols)
**URG_getFactor**

**Arguments**

- \( m \) The matrix
- rows The integer vectors of row index(es)
- cols The integer vectors of column index(es)

**Details**

This function implements faster submatrix extraction algorithm for the `CsparseMatrix` class in the package **Matrix**.

**Value**

This function will return the specified submatrix as a matrix of corresponding type.

---

**Description**

Calculate the normalization factor for each sample by using URG (uniform ratio graph) method.

**Usage**

```r
URG_getFactor(expr.matrix, p.edge = 0.25, p.gene = 0.4, log.expr = FALSE)
```

**Arguments**

- expr.matrix The expression matrix. Each row represents a gene and each column represents a sample.
- p.edge The percentage of gene pairs that are selected into the uniform ratio graph.
- p.gene The maximal percentage of genes that are selected as the stable genes.
- log.expr Logical variable indicating whether the input expression matrix is in logarithmic scale.

**Value**

This function will return a numeric vector with each element [i] represents the normalization factor of sample (i).

**References**


**See Also**

- `URG_normalize`
**URG_normalize**  
*Normalize using given factors*

**Description**

Normalize the expression matrix by using the given factor for each sample.

**Usage**

```r
URG_normalize(expr.matrix, factor, log.expr = FALSE)
```

**Arguments**

- `expr.matrix`: The expression matrix. Each row represents a gene and each column represents a sample.
- `factor`: The numeric vector of normalization factors.
- `log.expr`: Logical variable indicating whether the input expression matrix is in logarithmic scale.

**Value**

This function will return a numeric matrix with the same dimension of `expr.matrix`.

**See Also**

- `URG_getFactor`

**write_net**  
*Write network information to text file*

**Description**

Write the network information to a text file with specific format.

**Usage**

```r
write_net(net, file)
```

**Arguments**

- `net`: A list as returned by `read_net`
- `file`: The name of text file
write_net

Details

This function writes the network information to a text file with specific format: each line contains two strings separated by spaces, which correspond to the names of two end points of one edge in the network.

See Also

read_net
Index

* package
  Corbi-package, 2

best_subnets, 3, 4

column, 5
Corbi (Corbi-package), 2
Corbi-package, 2
CsparseMatrix, 5, 24, 31

dbinom, 25, 26
dhyper, 25
dmultinom, 26

extend_subnets, 3, 5

get_adjusted_deg_diff, 6
get_diff_ratio_net, 7
get_ratio_distribution, 8
get_ratio_distribution2, 9
get_ratio_variance, 10
get_shortest_distances, 10
get_subnets, 3, 11

kappa_score, 12

make_DEG_data, 12
make_DEG_data2, 13
make_DEG_pattern, 13, 14, 14
markrank, 3, 16

net_align, 3, 20
net_query, 3, 21, 21
net_query_batch, 3, 22
net_query_batch (net_query), 21
netDEG, 3, 18
netDEG_pvalue, 19
nnzero, 24

p_combine, 26
pmultihyper, 24

pmultinom, 25
read_net, 27, 27, 33
rhyper, 28
rmultihyper, 27
rmultinom, 26
RsparseMatrix, 24

simulate_dropout, 28
simulate_dropout2, 29
simulate_sample_groups, 30
submatrix, 30

URG_getFactor, 3, 31, 32
URG_normalize, 31, 32

write_net, 27, 32