Package ‘inferCSN’

April 17, 2024

Type Package
Title Inferring Cell-Specific Gene Regulatory Network
Version 1.0.3
Date 2024-4-16
Maintainer Meng Xu <mengxu98@qq.com>
Description A method for inferring cell-specific gene regulatory network from single-cell sequencing data.
License MIT + file LICENSE
URL https://mengxu98.github.io/inferCSN/
BugReports https://github.com/mengxu98/inferCSN/issues
Depends R (>= 3.3.0)
Imports ComplexHeatmap, doParallel, dplyr, foreach, ggnetwork, ggplot2, ggraph, Matrix, methods, parallel, patchwork, progress, purrr, Rcpp, stats, utils
Suggests circlize, gtools, igraph, precrec, pROC, testthat (>= 3.0.0), tidygraph
LinkingTo Rcpp, RcppArmadillo
Config/Needs/website mengxu98/mxtemplate
Config/testthat/edition 3
Encoding UTF-8
LazyData true
RoxygenNote 7.3.1
Language en-US
NeedsCompilation yes
Author Meng Xu [aut, cre] (<https://orcid.org/0000-0002-8300-1054>)
Repository CRAN
Date/Publication 2024-04-17 08:50:02 UTC
inferCSN-package

R topics documented:

inferCSN-package .................................................. 2
acc.calculate .......................................................... 3
auc.calculate .......................................................... 3
calculate.gene.rank .................................................. 4
check.parameters .................................................... 5
coeff.SRM_fit ........................................................ 6
contrast.networks ..................................................... 7
crossweight ............................................................ 8
crossweight_params .................................................. 9
dynamic.networks ..................................................... 10
example_ground_truth .............................................. 11
example_matrix ....................................................... 11
example_meta_data .................................................. 11
filter_sort_matrix ................................................... 11
inferCSN ............................................................... 12
model.fit ............................................................... 14
net.format ............................................................. 16
network.heatmap ..................................................... 17
normalization ........................................................ 19
predict.SRM_fit ....................................................... 20
prepare.performance.data .......................................... 21
print.SRM_fit ........................................................ 21
single.network ....................................................... 22
sparse.regression .................................................... 23
table.to.matrix ........................................................ 24
weight_filter .......................................................... 25

Index ................................................................. 27

inferCSN-package inferCSN: Inferring Cell-Specific Gene Regulatory Network

Description

A method for inferring cell-specific gene regulatory network from single-cell sequencing data.

Author(s)

Meng xu (Maintainer), <mengxu98@qq.com>

Source

https://github.com/mengxu98/inferCSN
### acc.calculate

**ACC calculate**

**Description**
ACC calculate

**Usage**
```r
acc.calculate(weight_table, ground_truth)
```

**Arguments**
- `weight_table`: The weight data table of network
- `ground_truth`: Ground truth for calculate AUC

**Value**
ACC value

**Examples**
```r
library(inferCSN)
data("example_matrix")
data("example_ground_truth")
weight_table <- inferCSN(example_matrix)
acc.calculate(weight_table, example_ground_truth)
```

---

### auc.calculate

**AUC value calculate**

**Description**
AUC value calculate
calculate.gene.rank

Usage

calculate.gene.rank(
    weight_table, 
    regulators = NULL, 
    targets = NULL, 
    directed = FALSE
)

Arguments

weight_table The weight data table of network
regulators Ground truth for calculate AUC
targets If true, draw and print figure of AUC
line_color The color of line in the figure
line_width The width of line in the figure

directed

Value

AUC values and figure

Examples

library(inferCSN)
data("example_matrix")
data("example_ground_truth")
weight_table <- inferCSN(example_matrix)
auc.calculate(weight_table, example_ground_truth, plot = TRUE)

calculate.gene.rank

Calculate and rank TFs in network

Description

Calculate and rank TFs in network

Usage

calculate.gene.rank(
    weight_table, 
    regulators = NULL, 
    targets = NULL, 
    directed = FALSE
)
check.parameters

Arguments

weight_table  The weight data table of network.
regulators  Regulators list.
targets  Targets list.
directed  If network is directed or not.

Value

A data.table with three columns

Examples

library(inferCSN)
data("example_matrix")
weight_table <- inferCSN(example_matrix)
head(calculate.gene.rank(weight_table))
head(calculate.gene.rank(weight_table, regulators = "g1"))
Arguments

- **matrix**: An expression matrix, cells by genes.
- **penalty**: The type of regularization. This can take either one of the following choices: "L0" and "L0L2". For high-dimensional and sparse data, such as single-cell sequencing data, "L0L2" is more effective.
- **algorithm**: The type of algorithm used to minimize the objective function. Currently "CD" and "CDPSI" are supported. The CDPSI algorithm may yield better results, but it also increases running time.
- **cross_validation**: Check whether cross validation is used.
- **seed**: The seed used in randomly shuffling the data for cross-validation.
- **n_folds**: The number of folds for cross-validation.
- **k_folds**: The number of folds for sample split.
- **r_threshold**: Threshold of R^2.
- **regulators**: Regulator genes.
- **targets**: Target genes.
- **regulators_num**: The number of non-zero coef, this value will affect the final performance. The maximum support size at which to terminate the regularization path. Recommend setting this to a small fraction of min(n,p) (e.g. 0.05 * min(n,p)) as L0 regularization typically selects a small portion of non-zeros.
- **verbose**: Print detailed information.
- **cores**: CPU cores.

Value

No return value, called for check input parameters

do.coef.SRM_fit

Extracts a specific solution in the regularization path

Description

Extracts a specific solution in the regularization path

Usage

```r
## S3 method for class 'SRM_fit'
do.coef(object, lambda = NULL, gamma = NULL, supportSize = NULL, ...)
```

```r
## S3 method for class 'SRM_fit_CV'
do.coef(object, lambda = NULL, gamma = NULL, ...)
```
**Arguments**

- object: The output of `model.fit` or `inferCSN.cvfit`
- lambda: The value of lambda at which to extract the solution
- gamma: The value of gamma at which to extract the solution
- supportSize: The number of non-zeros each solution extracted will contain
- ...: Other parameters

**Value**

- Return the specific solution
- Return the specific solution

---

**Description**

contrast.networks

**Usage**

```r
contrast.networks(
  weight_table,
  degree_value = 0,
  weight_value = 0,
  legend_position = "bottom"
)
```

**Arguments**

- weight_table: The weight data table of network.
- degree_value: degree_value
- weight_value: weight_value
- legend_position: The position of legend.

**Value**

- Return a ggplot2 object

**Examples**

```r
library(inferCSN)
data("example_matrix")
weight_table <- inferCSN(example_matrix)
contrast.networks(weight_table[1:50, ])
```
Perform crossweighting

Usage

crossweight(
    weight_table,
    matrix,
    meta_data = NULL,
    lag = floor(ncol(matrix)/5),
    min = ceiling(ncol(matrix)/50),
    max = floor(ncol(matrix)/12),
    symmetric_filter = FALSE,
    filter_thresh = 0
)

Arguments

weight_table  GRN dataframe, the result of running reconstructargetRN or reconstructargetRN_GENIE3
matrix         genes-by-cells expression matrix
meta_data      result of running findDynGenes
lag            lag window on which to run cross-correlation. Cross-correlation computed from -lag to +lag.
min            minimum of weighting window. Edges with offsets (or absolute offsets if symmetric_filter=TRUE) less than min will not be negatively weighted.
max            maximum of weighting window. Edges with offsets (or absolute offsets if symmetric_filter=TRUE) greater than max will have weights set to 0.
symmetric_filter whether or not to employ a symmetric weight scheme. If true, absolute offset is used in place of offset.
filter_thresh  after crossweighting, edges with weights less than filter_thresh will be set to 0.

Value

weight_table with offset and weighted_score added

Examples

## Not run:
library(inferCSN)
data("example_matrix")
weight_table <- inferCSN(example_matrix, verbose = TRUE)
crossweight_params <- crossweight(
  weight_table, 
  matrix = t(example_matrix)
)

p1 <- network.heatmap(weight_table)
p2 <- network.heatmap(weight_table_new[, 1:3])
p1 + p2

## End(Not run)

crossweight_params estimates min and max values for crossweighting for now assumes uniform cell density across pseudotime/only considers early time this needs to be refined if it’s to be useful...

Description

estimates min and max values for crossweighting for now assumes uniform cell density across pseudotime/only considers early time this needs to be refined if it’s to be useful...

Usage

crossweight_params(
  matrix, 
  meta_data, 
  pseudotime_min = 0.005, 
  pseudotime_max = 0.01 
)

Arguments

matrix matrix
meta_data meta_data
pseudotime_min pseudotime_min
pseudotime_max pseudotime_max

Value

Params list
dynamic.networks  

*Plot of dynamic networks*

**Description**

Plot of dynamic networks

**Usage**

```r
dynamic.networks(
  weight_table,
  regulators = NULL,
  targets = NULL,
  legend_position = "right"
)
```

**Arguments**

- `weight_table`: The weight data table of network.
- `regulators`: Regulators list.
- `targets`: Targets list.
- `legend_position`: The position of legend.

**Value**

A list of ggplot2 objects

**Examples**

```r
library(inferCSN)
data("example_matrix")
weight_table <- inferCSN(example_matrix)
dynamic.networks(
  weight_table,
  regulators = weight_table[1, 1]
)
dynamic.networks(
  weight_table,
  targets = weight_table[1, 1]
)
dynamic.networks(
  weight_table,
  regulators = weight_table[1, 1],
  targets = weight_table[1, 2]
)
```
**example_ground_truth**  
*Example ground truth data*

**Description**  
The data used for calculate the evaluating indicator.

---

**example_matrix**  
*Example matrix data*

**Description**  
The matrix used for reconstruct gene regulatory network.

---

**example_meta_data**  
*Example meta data*

**Description**  
The data contains cells and pseudotime information.

---

**filter_sort_matrix**  
*Filter and sort matrix*

**Description**  
Filter and sort matrix

**Usage**

`filter_sort_matrix(weight_matrix, regulators = NULL, targets = NULL)`

**Arguments**

- `weight_matrix`: The matrix of network weight.
- `regulators`: Regulators list.
- `targets`: Targets list.

**Value**

Filtered and sorted matrix
Examples

```r
library(inferCSN)
data("example_matrix")
weight_table <- inferCSN(example_matrix)
weight_matrix <- table.to.matrix(weight_table)
filter_sort_matrix(weight_matrix)[1:6, 1:6]

filter_sort_matrix(
  weight_matrix ,
  regulators = c("g1", "g2"),
  targets = c("g3", "g4")
)
```

inferCSN

**Inferring Cell-Specific Gene Regulatory Network**

### Description
Inferring Cell-Specific Gene Regulatory Network

### Usage

```r
inferCSN(object, ...)

## S4 method for signature 'matrix'
inferCSN(
  object,
  penalty = "L0",
  algorithm = "CD",
  cross_validation = FALSE,
  seed = 1,
  n_folds = 10,
  k_folds = NULL,
  r_threshold = 0,
  regulators = NULL,
  targets = NULL,
  regulators_num = NULL,
  verbose = FALSE,
  cores = 1,
  ...
)

## S4 method for signature 'data.frame'
inferCSN(
  object,
  penalty = "L0",
  algorithm = "CD",
```
cross_validation = FALSE,
seed = 1,
n_folds = 10,
k_folds = NULL,
r_threshold = 0,
regulators = NULL,
targets = NULL,
regulators_num = NULL,
verbose = FALSE,
cores = 1,
...
)

Arguments

object Input object
...
Arguments for other methods
penalty The type of regularization. This can take either one of the following choices: "L0" and "L0L2". For high-dimensional and sparse data, such as single-cell sequencing data, "L0L2" is more effective.
algorithm The type of algorithm used to minimize the objective function. Currently "CD" and "CDPSI" are supported. The CDPSI algorithm may yield better results, but it also increases running time.
cross_validation Check whether cross validation is used.
seed The seed used in randomly shuffling the data for cross-validation.
n_folds The number of folds for cross-validation.
k_folds The number of folds for sample split.
r_threshold Threshold of R^2.
regulators Regulator genes.
targets Target genes.
regulators_num The number of non-zero coef, this value will affect the final performance. The maximum support size at which to terminate the regularization path. Recommend setting this to a small fraction of min(n,p) (e.g. 0.05 * min(n,p)) as L0 regularization typically selects a small portion of non-zeros.
verbose Print detailed information.
cores CPU cores.

Value

A data table of gene-gene regulatory relationship
model.fit

Examples

```r
library(inferCSN)
data("example_matrix")
weight_table <- inferCSN(example_matrix, verbose = TRUE)
head(weight_table)

weight_table <- inferCSN(example_matrix, verbose = TRUE, cores = 2)
head(weight_table)
```

---

model.fit  
*Fit a sparse regression model*

Description

Computes the regularization path for the specified loss function and penalty function

Usage

```r
model.fit(
  x,
  y,
  penalty = "L0",
  algorithm = "CD",
  regulators_num = NULL,
  cross_validation = FALSE,
  n_folds = 10,
  seed = 1,
  loss = "SquaredError",
  nLambda = 100,
  nGamma = 5,
  gammaMax = 10,
  gammaMin = 1e-04,
  partialSort = TRUE,
  maxIters = 200,
  rtol = 1e-06,
  atol = 1e-09,
  activeSet = TRUE,
  activeSetNum = 3,
  maxSwaps = 100,
  scaleDownFactor = 0.8,
  screenSize = 1000,
  autoLambda = NULL,
  lambdaGrid = list(),
  excludeFirstK = 0,
  intercept = TRUE,
  lows = -Inf,
  highs = Inf
)
```
Arguments

- **x**: The data matrix
- **y**: The response vector
- **penalty**: The type of regularization. This can take either one of the following choices: "L0" and "L0L2". For high-dimensional and sparse data, such as single-cell sequencing data, "L0L2" is more effective.
- **algorithm**: The type of algorithm used to minimize the objective function. Currently "CD" and "CDPSI" are supported. The CDPSI algorithm may yield better results, but it also increases running time.
- **regulators_num**: The number of non-zero coefficients, this value will affect the final performance. The maximum support size at which to terminate the regularization path. Recommend setting this to a small fraction of min(n,p) (e.g. 0.05 * min(n,p)) as L0 regularization typically selects a small portion of non-zeros.
- **cross_validation**: Check whether cross validation is used.
- **n_folds**: The number of folds for cross-validation.
- **seed**: The seed used in randomly shuffling the data for cross-validation.
- **loss**: The loss function
- **nLambda**: The number of Lambda values to select
- **nGamma**: The number of Gamma values to select
- **gammaMax**: The maximum value of Gamma when using the L0L2 penalty
- **gammaMin**: The minimum value of Gamma when using the L0L2 penalty
- **partialSort**: If TRUE, partial sorting will be used for sorting the coordinates to do greedy cycling. Otherwise, full sorting is used.
- **maxIters**: The maximum number of iterations (full cycles) for CD per grid point
- **rtol**: The relative tolerance which decides when to terminate optimization (based on the relative change in the objective between iterations)
- **atol**: The absolute tolerance which decides when to terminate optimization (based on the absolute L2 norm of the residuals)
- **activeSet**: If TRUE, performs active set updates
- **activeSetNum**: The number of consecutive times a support should appear before declaring support stabilization
- **maxSwaps**: The maximum number of swaps used by CDPSI for each grid point
- **scaleDownFactor**: This parameter decides how close the selected Lambda values are
- **screenSize**: The number of coordinates to cycle over when performing initial correlation screening
- **autoLambda**: Ignored parameter. Kept for backwards compatibility
- **lambdaGrid**: A grid of Lambda values to use in computing the regularization path
- **excludeFirstK**: This parameter takes non-negative integers
- **intercept**: If FALSE, no intercept term is included in the model
- **lows**: Lower bounds for coefficients
- **highs**: Upper bounds for coefficients
Value
An S3 object describing the regularization path

Examples
library(inferCSN)
data("example_matrix")
fit <- model.fit(
  example_matrix[, -1],
  example_matrix[, 1]
)
head(coef(fit))

net.format

Description
Format weight table

Usage
net.format(weight_table, regulators = NULL, targets = NULL, abs_weight = TRUE)

Arguments
weight_table The weight data table of network.
regulators Regulators list.
targets Targets list.
abs_weight Logical value, whether to perform absolute value on weights, default set to 'TRUE'; and when set 'abs_weight' to 'TRUE', the output of weight table will create a new column named 'Interaction'.

Value
Format weight table

Examples
library(inferCSN)
data("example_matrix")
weight_table <- inferCSN(example_matrix)

net.format(
  weight_table,
  regulators = c("g1")
)
net.format
network.heatmap

weight_table, regulators = c("g1"), abs_weight = FALSE

net.format(
  weight_table, targets = c("g3")
)
net.format(
  weight_table, regulators = c("g1", "g3"), targets = c("g3", "g5")
)

network.heatmap # The heatmap of network

Description

The heatmap of network

Usage

network.heatmap(
  weight_table, regulators = NULL, targets = NULL, switch_matrix = TRUE,
  show_names = FALSE, heatmap_size = 5, heatmap_height = NULL,
  heatmap_width = NULL, heatmap_title = NULL, heatmap_color = c("#1966ad", "white", 
  "#bb141a"), border_color = "gray", rect_color = NA, anno_width = 1,
  anno_height = 1, row_anno_type = NULL, column_anno_type = NULL,
  legend_name = "Weight", row_title = "Regulators"
)

Arguments

weight_table The weight data table of network.
network.heatmap

regulators Regulators list.
targets Targets list.
switch_matrix Logical value, default set to ‘TRUE’, whether to weight data table to matrix.
show_names Logical value, default set to ‘FALSE’, whether to show names of row and column.
heatmap_size Default set to 5. The size of heatmap.
heatmap_height The height of heatmap.
heatmap_width The width of heatmap.
heatmap_title The title of heatmap.
heatmap_color Colors of heatmap.
border_color Default set to ‘gray’. Color of heatmap border.
rect_color Default set to ‘NA’. Color of heatmap rect.
anno_width Width of annotation.
anno_height Height of annotation.
row_anno_type Default set to ‘NULL’. c("boxplot", "barplot", "histogram", "density", "lines", "points", "horizon")
column_anno_type Default set to ‘NULL’. c("boxplot", "barplot", "histogram", "density", "lines", "points")
legend_name The name of legend.
row_title The title of row.

Value

Return a heatmap

Examples

library(inferCSN)
data("example_matrix")
data("example_ground_truth")
weight_table <- inferCSN(example_matrix)

p1 <- network.heatmap(
  example_ground_truth[, 1:3],
  heatmap_title = "Ground truth",
  legend_name = "Ground truth"
)
p2 <- network.heatmap(
  weight_table,
  heatmap_title = "inferCSN",
  legend_name = "inferCSN"
)
ComplexHeatmap::draw(p1 + p2)

p3 <- network.heatmap(
Normalization

Description

Normalization

Usage

normalization(x, method = "max_min")
predict.SRM_fit

Arguments

  x         A numeric vector.
  method    Method for normalization.

Value

Normalized vector

---

predict.SRM_fit  Predict Response

Description

Predicts response for a given sample

Usage

```r
## S3 method for class 'SRM_fit'
predict(object, newx, lambda = NULL, gamma = NULL, ...)

## S3 method for class 'SRM_fit.CV'
predict(object, newx, lambda = NULL, gamma = NULL, ...)
```

Arguments

  object     The output of model.fit
  newx       A matrix on which predictions are made. The matrix should have p columns
  lambda     The value of lambda to use for prediction. A summary of the lambdas in the
              regularization path can be obtained using print(fit)
  gamma      The value of gamma to use for prediction. A summary of the gammas in the
              regularization path can be obtained using print(fit)
  ...        Other parameters

Details

If both lambda and gamma are not supplied, then a matrix of predictions for all the solutions in the
regularization path is returned. If lambda is supplied but gamma is not, the smallest value of gamma
is used. In case of logistic regression, probability values are returned

Value

Return predict value
Return the predict value
**prepare.performance.data**

**Description**

prepare.performance.data

**Usage**

prepare.performance.data(weight_table, ground_truth)

**Arguments**

- **weight_table**: The weight data table of network
- **ground_truth**: Ground truth for calculate AUC

**Value**

Formatted data

**print.SRM_fit**

*Prints a summary of model.fit*

**Description**

Prints a summary of model.fit

**Usage**

```r
## S3 method for class 'SRM_fit'
print(x, ...)
```

```r
## S3 method for class 'SRM_fit.CV'
print(x, ...)
```

**Arguments**

- **x**: The output of model.fit or inferCSN.cvfit
- **...**: Other parameters

**Value**

Return information of model.fit

Return information of model.fit
single.network  
Construct network for single gene

Description
Construct network for single gene

Usage
single.network(
  matrix,  
  regulators,  
  target,  
  cross_validation = FALSE,  
  seed = 1,  
  penalty = "L0",  
  algorithm = "CD",  
  regulators_num = NULL,  
  n_folds = 10,  
  k_folds = NULL,  
  r_threshold = 0,  
  verbose = FALSE  
)

Arguments
matrix  
An expression matrix, cells by genes.

regulators  
Regulator genes.

target  
Target genes.

cross_validation  
Check whether cross validation is used.

seed  
The seed used in randomly shuffling the data for cross-validation.

penalty  
The type of regularization. This can take either one of the following choices: 
"L0" and "L0L2". For high-dimensional and sparse data, such as single-cell 
sequencing data, "L0L2" is more effective.

algorithm  
The type of algorithm used to minimize the objective function. Currently "CD" 
and "CDPSI" are supported. The CDPSI algorithm may yield better results, but 
it also increases running time.

regulators_num  
The number of non-zero coef, this value will affect the final performance. The 
maximum support size at which to terminate the regularization path. Recommend 
setting this to a small fraction of min(n,p) (e.g. 0.05 * min(n,p)) as L0 
regularization typically selects a small portion of non-zeros.

n_folds  
The number of folds for cross-validation.

k_folds  
The number of folds for sample split.

r_threshold  
Threshold of R^2.

verbose  
Print detailed information.
Value

The weight data table of sub-network

Examples

library(inferCSN)
data("example_matrix")
single_network <- single.network(  
  example_matrix,  
  regulators = colnames(example_matrix),  
  target = "g1"  
)
head(single_network)

single.network(  
  example_matrix,  
  regulators = "g1",  
  target = "g2"  
)

sparse.regression

Sparse regression model

Description

Sparse regression model

Usage

sparse.regression(  
  x,  
  y,  
  cross_validation = FALSE,  
  seed = 1,  
  penalty = "L0",  
  algorithm = "CD",  
  regulators_num = NULL,  
  n_folds = 10,  
  k_folds = NULL,  
  r_threshold = 0,  
  verbose = FALSE  
)

Arguments

x The data matrix
y The response vector
cross_validation
   Check whether cross validation is used.
seed
   The seed used in randomly shuffling the data for cross-validation.
penalty
   The type of regularization. This can take either one of the following choices:
   "L0" and "L0L2". For high-dimensional and sparse data, such as single-cell
   sequencing data, "L0L2" is more effective.
algorithm
   The type of algorithm used to minimize the objective function. Currently "CD"
   and "CDPSI" are supported. The CDPSI algorithm may yield better results, but
   it also increases running time.
regulators_num
   The number of non-zero coef, this value will affect the final performance. The
   maximum support size at which to terminate the regularization path. Recommend
   setting this to a small fraction of min(n,p) (e.g. 0.05 * min(n,p)) as L0
   regularization typically selects a small portion of non-zeros.
n_folds
   The number of folds for cross-validation.
k_folds
   The number of folds for sample split.
r_threshold
   Threshold of R^2.
verbose
   Print detailed information.

Value
   Coefficients

Examples

library(inferCSN)
data("example_matrix")
coefficients <- sparse.regression(  
  example_matrix[, -1],
  example_matrix[, 1]
)
coefficients

---

### table.to.matrix

#### Switch weight table to matrix

#### Description

Switch weight table to matrix

#### Usage

```r
table.to.matrix(weight_table, regulators = NULL, targets = NULL)
```
weight_filter

Arguments

weight_table The weight data table of network.
regulators Regulators list.
targets Targets list.

Value

Weight matrix

Examples

library(inferCSN)
data("example_matrix")
weight_table <- inferCSN(example_matrix)
head(weight_table)

table.to.matrix(weight_table)[1:6, 1:6]

table.to.matrix(
  weight_table,
  regulators = c("g1", "g2"),
  targets = c("g3", "g4")
)

-------------------

weight_filter weight_filter

-------------------

Description

weight_filter

Usage

weight_filter(weight_table, method = "max")

Arguments

weight_table weight_table
method method

Value

Filtered weight table
Examples

```r
library(inferCSN)
data("example_matrix")
data("example_ground_truth")
weight_table <- inferCSN(example_matrix, verbose = TRUE)
weight_table_new <- weight_filter(weight_table)

network.heatmap(
    example_ground_truth[, 1:3],
    heatmap_title = "Ground truth",
    show_names = TRUE,
    rect_color = "gray90"
)

network.heatmap(
    weight_table,
    heatmap_title = "Raw",
    show_names = TRUE,
    rect_color = "gray90"
)

network.heatmap(
    weight_table_new,
    heatmap_title = "Filtered",
    show_names = TRUE,
    rect_color = "gray90"
)

auc.calculate(
    weight_table,
    example_ground_truth,
    plot = TRUE
)

auc.calculate(
    weight_table_new,
    example_ground_truth,
    plot = TRUE
)
```
Index

acc.calculate, 3
auc.calculate, 3
calculate.gene.rank, 4
check.parameters, 5
coef.SRM_fit, 6
coef.SRM_fit_CV (coef.SRM_fit), 6
contrast.networks, 7
crossweight, 8
crossweight_params, 9
dynamic.networks, 10
eexample_ground_truth, 11
eexample_matrix, 11
example_meta_data, 11
filter_sort_matrix, 11
inferCSN, 12
inferCSN, data.frame-method (inferCSN), 12
inferCSN, matrix-method (inferCSN), 12
inferCSN-package, 2
model.fit, 14
net.format, 16
network.heatmap, 17
normalization, 19
predict.SRM_fit, 20
predict.SRM_fit_CV (predict.SRM_fit), 20
prepare.performance.data, 21
print.SRM_fit, 21
print.SRM_fit_CV (print.SRM_fit), 21
single.network, 22
sparse.regression, 23
table.to.matrix, 24
weight_filter, 25