Package 'inferCSN'

August 24, 2024

Type Package

Title Inferring Cell-Specific Gene Regulatory Network

Version 1.0.8

Date 2024-8-10

Maintainer Meng Xu <mengxu98@qq.com>

Description
An R package for inferring cell-type specific gene regulatory network from single-cell RNA data.

License MIT + file LICENSE

URL https://mengxu98.github.io/inferCSN/

BugReports https://github.com/mengxu98/inferCSN/issues

Depends R (>= 4.1.0)

Imports cli, dplyr, doParallel, foreach, ggnetwork, ggplot2, ggraph, Matrix, methods, parallel, patchwork, pbapply, purrr, Rcpp, RcppArmadillo, RcppParallel, stats, utils

Suggests ComplexHeatmap, circlize, gtools, gganimate, ggExtra, ggpointdensity, ggpubr, igraph, network, plotly, precrec, pROC, testthat (>= 3.0.0), tidygraph, RColorBrewer, Rtsne, RTransferEntropy, uwot, viridis

LinkingTo Rcpp, RcppArmadillo, RcppParallel

Config/Needs/website mengxu98/mxtemplate

Config/testthat/edition 3

Encoding UTF-8

LazyData true

RoxygenNote 7.3.2

Language en-US

NeedsCompilation yes

Author Meng Xu [aut, cre] (<https://orcid.org/0000-0002-8300-1054>)

Repository CRAN

Date/Publication 2024-08-24 05:30:02 UTC

Contents

`inferCSN-package` **inferCSN**: **infer***ring* **C***ell-***S***pecific gene regulatory* **N***etwork*

Description

An R package for **infer**ring **C**ell-**S**pecific gene regulatory **N**etwork from single-cell RNA data

Author(s)

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

Source

https://github.com/mengxu98/inferCSN

See Also

Useful links:

- https://mengxu98.github.io/inferCSN/
- Report bugs at https://github.com/mengxu98/inferCSN/issues

`as_matrix` *Convert dgCMatrix into a dense matrix*

Description

Convert dgCMatrix into a dense matrix

Usage

```
as_matrix(x, parallel = FALSE, sparse = FALSE)
```

Arguments

`x`	A matrix.
`parallel`	Logical value, default is `FALSE`. Setting to parallelize the computation with `setThreadOptions`.
`sparse`	Logical value, default is `FALSE`, whether to output a sparse matrix.

Examples

```
dims_i <- 2000
dims_j <- 2000
sparse_matrix <- Matrix::sparseMatrix(
  i = sample(1:dims_i, 500),
  j = sample(1:dims_j, 500),
  x = rnorm(500),
  dims = c(dims_i, dims_j),
  dimnames = list(
    paste0("a", rep(1:dims_i)),
    paste0("b", rep(1:dims_j))
  )
)

system.time(as.matrix(sparse_matrix))
system.time(as_matrix(sparse_matrix))
system.time(as_matrix(sparse_matrix, parallel = TRUE))

identical(
```

```
  as.matrix(sparse_matrix),
  as_matrix(sparse_matrix)
)

identical(
  as.matrix(sparse_matrix),
  as_matrix(sparse_matrix, parallel = TRUE)
)

identical(
  sparse_matrix,
  as_matrix(as.matrix(sparse_matrix), sparse = TRUE)
)

## Not run:
network_table_1 <- inferCSN(
  as_matrix(example_matrix, sparse = TRUE)
)
network_table_2 <- inferCSN(
  as(example_matrix, "sparseMatrix")
)

plot_scatter(
  data.frame(
    network_table_1$weight,
    network_table_2$weight
  ),
  legend_position = "none"
)

## End(Not run)
```

`calculate_acc` *Calculate accuracy value*

Description

Calculate accuracy value

Usage

```
calculate_acc(network_table, ground_truth)
```

Arguments

`network_table`	The weight data table of network
`ground_truth`	Ground truth for calculate AUC

Value

ACC value

Examples

```
data("example_matrix")
data("example_ground_truth")
network_table <- inferCSN(example_matrix)
calculate_acc(network_table, example_ground_truth)
```

`calculate_auc` *Calculate AUPRC and AUROC values*

Description

Calculate AUPRC and AUROC values

Usage

```
calculate_auc(
  network_table,
  ground_truth,
  plot = FALSE,
  line_color = "#1563cc",
  line_width = 1
)
```

Arguments

`network_table`	The weight data table of network
`ground_truth`	Ground truth for calculate AUC
`plot`	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

Value

AUC values and figure

Examples

```
data("example_matrix")
data("example_ground_truth")
network_table <- inferCSN(example_matrix)
calculate_auc(network_table, example_ground_truth, plot = TRUE)
```

calculate_gene_rank *Rank TFs and genes in network*

Description

Rank TFs and genes in network

Usage

```
calculate_gene_rank(
  network_table,
  regulators = NULL,
  targets = NULL,
  directed = FALSE
)
```

Arguments

network_table	The weight data table of network.
regulators	Regulators list.
targets	Targets list.
directed	Whether the network is directed.

Value

A table of gene rank.

Examples

```
data("example_matrix")
network_table <- inferCSN(example_matrix)
head(calculate_gene_rank(network_table))
head(calculate_gene_rank(network_table, regulators = "g1"))
head(calculate_gene_rank(network_table, targets = "g1"))
```

check_sparsity *Check sparsity of matrix*

Description

Check sparsity of matrix

Usage

```
check_sparsity(x)
```

Arguments

x	A matrix.

Value

Sparsity of matrix

---

coef.srm *Extracts a specific solution in the regularization path*

Description

Extracts a specific solution in the regularization path

Usage

```
## S3 method for class 'srm'
coef(object, lambda = NULL, gamma = NULL, regulators_num = NULL, ...)

## S3 method for class 'srm_cv'
coef(object, lambda = NULL, gamma = NULL, ...)
```

Arguments

object	The output of fit_sparse_regression.
lambda	The value of lambda at which to extract the solution.
gamma	The value of gamma at which to extract the solution.
regulators_num	The number of non-zore 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.
...	Other parameters

Value

Return the specific solution

---

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(network_matrix, regulators = NULL, targets = NULL)
```

Arguments

network_matrix	The matrix of network weight.
regulators	Regulators list.
targets	Targets list.

Value

Filtered and sorted matrix

Examples

```
data("example_matrix")
network_table <- inferCSN(example_matrix)
network_matrix <- table_to_matrix(network_table)
filter_sort_matrix(network_matrix)[1:6, 1:6]

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

`fit_sparse_regression` *Fit a sparse regression model*

Description

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

Usage

```
fit_sparse_regression(
  x,
  y,
  penalty = "L0",
  algorithm = "CD",
  regulators_num = ncol(x),
  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 matrix of regulators.
y	The vector of target.

penalty	The type of regularization, default is L0. This can take either one of the following choices: L0, L0L1, and L0L2. For high-dimensional and sparse data, L0L2 is more effective.
algorithm	The type of algorithm used to minimize the objective function, default is CD. 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-zore 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	Logical value, default is FALSE, whether to use cross-validation.
n_folds	The number of folds for cross-validation, default is 10.
seed	The random seed for cross-validation, default is 1.
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. For the L0L1 penalty this is automatically selected.
gammaMin	The minimum value of Gamma when using the L0L2 penalty. For the L0L1 penalty, the minimum value of gamma in the grid is set to gammaMin * gammaMax. Note that this should be a strictly positive quantity.
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.
...	Parameters for other methods.

Value

An S3 object describing the regularization path

References

Hazimeh, Hussein et al. "L0Learn: A Scalable Package for Sparse Learning using L0 Regularization." J. Mach. Learn. Res. 24 (2022): 205:1-205:8.

Hazimeh, Hussein and Rahul Mazumder. "Fast Best Subset Selection: Coordinate Descent and Local Combinatorial Optimization Algorithms." Oper. Res. 68 (2018): 1517-1537.

https://github.com/hazimehh/L0Learn/blob/master/R/fit.R

Examples

```
data("example_matrix")
fit <- fit_sparse_regression(
  example_matrix[, -1],
  example_matrix[, 1]
)
head(coef(fit))
```

`inferCSN` **infer***ring* **C***ell-***S***pecific gene regulatory* **N***etwork*

Description

inferring **C**ell-**S**pecific gene regulatory **N**etwork

Usage

```
inferCSN(
  object,
  penalty = "L0",
  algorithm = "CD",
  cross_validation = FALSE,
  seed = 1,
  n_folds = 10,
  percent_samples = 1,
  r_threshold = 0,
  regulators = NULL,
  targets = NULL,
  regulators_num = NULL,
  cores = 1,
  verbose = TRUE,
  ...
)

## S4 method for signature 'matrix'
```

```
inferCSN(
  object,
  penalty = "L0",
  algorithm = "CD",
  cross_validation = FALSE,
  seed = 1,
  n_folds = 10,
  percent_samples = 1,
  r_threshold = 0,
  regulators = NULL,
  targets = NULL,
  regulators_num = NULL,
  cores = 1,
  verbose = TRUE,
  ...
)

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

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

```
  cores = 1,
  verbose = TRUE,
  ...
)
```

Arguments

object	The input data for inferCSN.
penalty	The type of regularization, default is L0. This can take either one of the following choices: L0, L0L1, and L0L2. For high-dimensional and sparse data, L0L2 is more effective.
algorithm	The type of algorithm used to minimize the objective function, default is CD. Currently CD and CDPSI are supported. The CDPSI algorithm may yield better results, but it also increases running time.
cross_validation	Logical value, default is FALSE, whether to use cross-validation.
seed	The random seed for cross-validation, default is 1.
n_folds	The number of folds for cross-validation, default is 10.
percent_samples	The percent of all samples used for sparse_regression, default is 1.
r_threshold	Threshold of R^2 or correlation coefficient, default is 0.
regulators	The regulator genes for which to infer the regulatory network.
targets	The target genes for which to infer the regulatory network.
regulators_num	The number of non-zore 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.
cores	The number of cores to use for parallelization with foreach, default is 1.
verbose	Logical value, default is TRUE, whether to print progress messages.
...	Parameters for other methods.

Value

A data table of regulator-target regulatory relationships

Examples

```
data("example_matrix")
network_table_1 <- inferCSN(
  example_matrix
)
head(network_table_1)

network_table_2 <- inferCSN(
  example_matrix,
  cores = 2
```

```
)

identical(
  network_table_1,
  network_table_2
)

inferCSN(
  example_matrix,
  regulators = c("g1", "g2"),
  targets = c("g3", "g4")
)
inferCSN(
  example_matrix,
  regulators = c("g1", "g2"),
  targets = c("g3", "g0")
)
inferCSN(
  example_matrix,
  regulators = c("g1", "g0"),
  targets = c("g2", "g3")
)
inferCSN(
  example_matrix,
  regulators = c("g1"),
  targets = c("g2")
)
## Not run:
data("example_matrix")
network_table <- inferCSN(example_matrix)
head(network_table)

network_table_sparse_1 <- inferCSN(
  as(example_matrix, "sparseMatrix")
)
head(network_table_sparse_1)

network_table_sparse_2 <- inferCSN(
  as(example_matrix, "sparseMatrix"),
  cores = 2
)
identical(
  network_table,
  network_table_sparse_1
)

identical(
  network_table_sparse_1,
  network_table_sparse_2
)

plot_scatter(
  data.frame(
```

```
    network_table$weight,
    network_table_sparse_1$weight
  ),
  legend_position = "none"
)

plot_weight_distribution(
  network_table
) + plot_weight_distribution(
  network_table_sparse_1
)

## End(Not run)
```

---

log_message *Print diagnostic message*

Description

Print diagnostic message

Usage

```
log_message(..., verbose = TRUE, message_type = "info", cli_model = TRUE)
```

Arguments

...	Text to print.
verbose	Logical value, default is TRUE. Whether to print the message.
message_type	Type of message, default is info. Could be choose one of info, warning, and error.
cli_model	Logical value, default is TRUE. Whether to use the cli package to print the message. Add because the message is printed by message, the message could be suppressed by suppressMessages.

Examples

```
log_message("Hello, ", "world!")
suppressMessages(log_message("Hello, ", "world!"))
log_message("Hello, world!", verbose = FALSE)
log_message("Hello, world!", verbose = TRUE, message_type = "warning")
```

network_format *Format network table*

Description

Format network table

Usage

```
network_format(
  network_table,
  regulators = NULL,
  targets = NULL,
  abs_weight = TRUE
)
```

Arguments

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

Value

Formated network table

Examples

```
data("example_matrix")
network_table <- inferCSN(example_matrix)

network_format(
  network_table,
  regulators = c("g1")
)

network_format(
  network_table,
  regulators = c("g1"),
  abs_weight = FALSE
)

network_format(
  network_table,
  targets = c("g3")
```

```
)

network_format(
  network_table,
  regulators = c("g1", "g3"),
  targets = c("g3", "g5")
)
```

`network_sift` *Sifting network*

Description

Sifting network

Usage

```
network_sift(
  network_table,
  matrix = NULL,
  meta_data = NULL,
  pseudotime_column = NULL,
  method = c("entropy", "max"),
  entropy_method = c("Shannon", "Renyi"),
  effective_entropy = FALSE,
  shuffles = 100,
  entropy_nboot = 300,
  lag_value = 1,
  entropy_p_value = 0.05,
  cores = 1,
  verbose = TRUE
)
```

Arguments

`network_table`	The weight data table of network.
`matrix`	The expression matrix.
`meta_data`	The meta data for cells or samples.
`pseudotime_column`	The column of pseudotime.
`method`	The method used for filter edges. Could be choose `entropy` or `max`.
`entropy_method`	If setting `method` to `entropy`, could be choose `Shannon` or `Renyi` to compute entropy.
`effective_entropy`	Default is `FALSE`. Logical value, using effective entropy to filter weights or not.

shuffles	Default is 100. The number of shuffles used to calculate the effective transfer entropy.
entropy_nboot	Default is 300. The number of bootstrap replications for each direction of the estimated transfer entropy.
lag_value	Default is 1. Markov order of gene expression values, i.e. the number of lagged values affecting the current value of gene expression values.
entropy_p_value	P value used to filter edges by entropy.
cores	The number of cores to use for parallelization with foreach, default is 1.
verbose	Logical value, default is TRUE, whether to print progress messages.

Value

Sifted network table

Examples

```
## Not run:
data("example_matrix")
data("example_meta_data")
data("example_ground_truth")
network_table <- inferCSN(example_matrix)
network_table_sifted <- network_sift(network_table)
network_table_sifted_entropy <- network_sift(
  network_table,
  matrix = example_matrix,
  meta_data = example_meta_data,
  pseudotime_column = "pseudotime",
  lag_value = 2,
  shuffles = 0,
  entropy_nboot = 0
)

plot_network_heatmap(
  example_ground_truth[, 1:3],
  heatmap_title = "Ground truth",
  show_names = TRUE,
  rect_color = "gray70"
)
plot_network_heatmap(
  network_table,
  heatmap_title = "Raw",
  show_names = TRUE,
  rect_color = "gray70"
)
plot_network_heatmap(
  network_table_sifted,
  heatmap_title = "Filtered",
  show_names = TRUE,
  rect_color = "gray70"
```

```
)
plot_network_heatmap(
  network_table_sifted_entropy,
  heatmap_title = "Filtered by entropy",
  show_names = TRUE,
  rect_color = "gray70"
)

calculate_auc(
  network_table,
  example_ground_truth,
  plot = TRUE
)
calculate_auc(
  network_table_sifted,
  example_ground_truth,
  plot = TRUE
)
calculate_auc(
  network_table_sifted_entropy,
  example_ground_truth,
  plot = TRUE
)

## End(Not run)
```

`normalization` *Normalize numeric vector*

Description

Normalize numeric vector

Usage

```
normalization(x, method = "max_min", na_rm = TRUE)
```

Arguments

`x`	Input numeric vector.
`method`	Method used for normalization.
`na_rm`	Whether to remove `NA` values, and if setting TRUE, using `0` instead.

Value

Normalized numeric vector

Examples

```
nums <- c(runif(2), NA, -runif(2))
nums
normalization(nums, method = "max_min")
normalization(nums, method = "maximum")
normalization(nums, method = "sum")
normalization(nums, method = "softmax")
normalization(nums, method = "z_score")
normalization(nums, method = "mad")
normalization(nums, method = "unit_vector")
normalization(nums, method = "unit_vector", na_rm = FALSE)
```

---

parallelize_fun *Parallelize a function*

Description

Parallelize a function

Usage

```
parallelize_fun(x, fun, cores = 1, export_fun = NULL, verbose = TRUE)
```

Arguments

x	A vector or list to apply over.
fun	The function to be applied to each element.
cores	The number of cores to use for parallelization with foreach, default is 1.
export_fun	The functions to export the function to workers.
verbose	Logical value, default is TRUE, whether to print progress messages.

Value

A list of computed results

plot_contrast_networks *Plot contrast networks*

Description

Plot contrast networks

Usage

```
plot_contrast_networks(
  network_table,
  degree_value = 0,
  weight_value = 0,
  legend_position = "bottom"
)
```

Arguments

network_table	The weight data table of network.
degree_value	Degree value to filter nodes.
weight_value	Weight value to filter edges.
legend_position	The position of legend.

Value

A ggplot2 object

Examples

```
data("example_matrix")
network_table <- inferCSN(example_matrix)
plot_contrast_networks(network_table[1:50, ])
```

plot_dynamic_networks *Plot dynamic networks*

Description

Plot dynamic networks

Usage

```
plot_dynamic_networks(
  network_table,
  celltypes_order,
  ntop = 10,
  title = NULL,
  theme_type = "theme_void",
  plot_type = "ggplot",
  layout = "fruchtermanreingold",
  nrow = 2,
  figure_save = FALSE,
  figure_name = NULL,
  figure_width = 6,
  figure_height = 6,
  seed = 1
)
```

Arguments

network_table	The weight data table of network.
celltypes_order	The order of cell types.
ntop	The number of top genes to plot.
title	The title of figure.
theme_type	Default is theme_void, the theme of figure, could be theme_void, theme_blank or theme_facet.
plot_type	Default is "ggplot", the type of figure, could be ggplot, animate or ggplotly.
layout	Default is "fruchtermanreingold", the layout of figure, could be fruchtermanreingold or kamadakawai.
nrow	The number of rows of figure.
figure_save	Default is FALSE, Logical value, whether to save the figure file.
figure_name	The name of figure file.
figure_width	The width of figure.
figure_height	The height of figure.
seed	Default is 1, the seed random use to plot network.

Value

A dynamic figure object

Examples

```
data("example_matrix")
network <- inferCSN(example_matrix)[1:100, ]
network$celltype <- c(
```

```
  rep("cluster1", 20),
  rep("cluster2", 20),
  rep("cluster3", 20),
  rep("cluster5", 20),
  rep("cluster6", 20)
)

celltypes_order <- c(
  "cluster5", "cluster3",
  "cluster2", "cluster1",
  "cluster6"
)

plot_dynamic_networks(
  network,
  celltypes_order = celltypes_order
)

plot_dynamic_networks(
  network,
  celltypes_order = celltypes_order[1:3]
)

plot_dynamic_networks(
  network,
  celltypes_order = celltypes_order,
  plot_type = "ggplotly"
)

## Not run:
# If setting `plot_type = "animate"` to plot and save `gif` figure,
# please install `gifski` package first.
plot_dynamic_networks(
  network,
  celltypes_order = celltypes_order,
  plot_type = "animate"
)

## End(Not run)
```

`plot_embedding` *Plot embedding*

Description

Plot embedding

Usage

```
plot_embedding(
```

```
  matrix,
  labels = NULL,
  method = "pca",
  colors = RColorBrewer::brewer.pal(length(unique(labels)), "Set1"),
  seed = 1,
  point_size = 1,
  cores = 1
)
```

Arguments

matrix	Input matrix.
labels	Input labels.
method	Method to use for dimensionality reduction.
colors	Colors to use for the plot.
seed	Seed for the random number generator.
point_size	Size of the points.
cores	Set the number of threads when setting method to umap and Rtsne.

Value

An embedding plot

Examples

```
data("example_matrix")
samples_use <- 1:200
plot_data <- example_matrix[samples_use, ]
labels <- sample(
  c("A", "B", "C", "D", "E"),
  nrow(plot_data),
  replace = TRUE
)

plot_embedding(
  plot_data,
  labels,
  method = "pca",
  point_size = 2
)

plot_embedding(
  plot_data,
  labels,
  method = "tsne",
  point_size = 2
)
```

plot_network_heatmap *Plot network heatmap*

Description

Plot network heatmap

Usage

```
plot_network_heatmap(
  network_table,
  regulators = NULL,
  targets = NULL,
  switch_matrix = TRUE,
  show_names = FALSE,
  heatmap_size_lock = TRUE,
  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

network_table	The weight data table of network.
regulators	Regulators list.
targets	Targets list.
switch_matrix	Logical value, default is TRUE, whether to weight data table to matrix.
show_names	Logical value, default is FALSE, whether to show names of row and column.
heatmap_size_lock	Lock the size of heatmap.
heatmap_size	Default is 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 is gray. Color of heatmap border.
rect_color	Default is NA. Color of heatmap rect.
anno_width	Width of annotation.
anno_height	Height of annotation.
row_anno_type	Default is NULL, could add a annotation plot to row, choose one of boxplot, barplot, histogram, density, lines, points, and horizon.
column_anno_type	Default is NULL, could add a annotation plot to column, choose one of boxplot, barplot, histogram, density, lines, and points.
legend_name	The name of legend.
row_title	The title of row.

Value

A heatmap

Examples

```
data("example_matrix")
data("example_ground_truth")
network_table <- inferCSN(example_matrix)

p1 <- plot_network_heatmap(
  example_ground_truth[, 1:3],
  heatmap_title = "Ground truth",
  legend_name = "Ground truth"
)
p2 <- plot_network_heatmap(
  network_table,
  heatmap_title = "inferCSN",
  legend_name = "inferCSN"
)
ComplexHeatmap::draw(p1 + p2)

p3 <- plot_network_heatmap(
  network_table,
  heatmap_title = "inferCSN",
  legend_name = "Weight1",
  heatmap_color = c("#20a485", "#410054", "#fee81f")
)
p4 <- plot_network_heatmap(
  network_table,
  heatmap_title = "inferCSN",
  legend_name = "Weight2",
  heatmap_color = c("#20a485", "white", "#fee81f")
)
ComplexHeatmap::draw(p3 + p4)
```

```
plot_network_heatmap(
  network_table,
  show_names = TRUE,
  rect_color = "gray90",
  row_anno_type = "density",
  column_anno_type = "barplot"
)

plot_network_heatmap(
  network_table,
  regulators = c("g1", "g2"),
  show_names = TRUE
)

plot_network_heatmap(
  network_table,
  targets = c("g1", "g2"),
  row_anno_type = "boxplot",
  column_anno_type = "histogram",
  show_names = TRUE
)

plot_network_heatmap(
  network_table,
  regulators = c("g1", "g3", "g5"),
  targets = c("g3", "g6", "g9"),
  show_names = TRUE
)
```

`plot_scatter` *Plot expression data in a scatter plot*

Description

Plot expression data in a scatter plot

Usage

```
plot_scatter(
  data,
  smoothing_method = "lm",
  group_colors = RColorBrewer::brewer.pal(9, "Set1"),
  title_color = "black",
  title = NULL,
  col_title = NULL,
  row_title = NULL,
  legend_title = NULL,
  legend_position = "bottom",
  margins = "both",
```

```
  marginal_type = NULL,
  margins_size = 10,
  compute_correlation = TRUE,
  compute_correlation_method = "pearson",
  keep_aspect_ratio = TRUE,
  facet = FALSE,
  se = FALSE,
  pointdensity = TRUE
)
```

Arguments

data	Input data.
smoothing_method	Method for smoothing curve, lm or loess.
group_colors	Colors for different groups.
title_color	Color for the title.
title	Main title for the plot.
col_title	Title for the x-axis.
row_title	Title for the y-axis.
legend_title	Title for the legend.
legend_position	The position of legend.
margins	The position of marginal figure ("both", "x", "y").
marginal_type	The type of marginal figure (density, histogram, boxplot, violin, densigram).
margins_size	The size of marginal figure, note the bigger size the smaller figure.
compute_correlation	Whether to compute and print correlation on the figure.
compute_correlation_method	Method to compute correlation (pearson or spearman).
keep_aspect_ratio	Logical value, whether to set aspect ratio to 1:1.
facet	Faceting variable. If setting TRUE, all settings about margins will be inalidation.
se	Display confidence interval around smooth.
pointdensity	Plot point density when only provide 1 cluster.

Value

ggplot object

Examples

```
data("example_matrix")
test_data <- data.frame(
  example_matrix[1:200, c(1, 7)],
  c = c(
    rep("c1", 40),
    rep("c2", 40),
    rep("c3", 40),
    rep("c4", 40),
    rep("c5", 40)
  )
)

p1 <- plot_scatter(
  test_data
)
p2 <- plot_scatter(
  test_data,
  marginal_type = "boxplot"
)
p1 + p2

p3 <- plot_scatter(
  test_data,
  facet = TRUE
)
p3

p4 <- plot_scatter(
  test_data[, 1:2],
  marginal_type = "histogram"
)
p4
```

---

plot_static_networks *Plot dynamic networks*

Description

Plot dynamic networks

Usage

```
plot_static_networks(
  network_table,
  regulators = NULL,
  targets = NULL,
  legend_position = "right"
)
```

Arguments

network_table	The weight data table of network.
regulators	Regulators list.
targets	Targets list.
legend_position	The position of legend.

Value

A ggplot2 object

Examples

```
data("example_matrix")
network_table <- inferCSN(example_matrix)
plot_static_networks(
  network_table,
  regulators = network_table[1, 1]
)
plot_static_networks(
  network_table,
  targets = network_table[1, 1]
)
plot_static_networks(
  network_table,
  regulators = network_table[1, 1],
  targets = network_table[1, 2]
)
```

---

plot_weight_distribution

Plot weight distribution

Description

Plot weight distribution

Usage

```
plot_weight_distribution(
  network_table,
  binwidth = 0.01,
  show_border = FALSE,
  border_color = "black",
  alpha = 1,
  theme = "viridis",
  theme_begin = 0,
```

```
  theme_end = 0.5,
  theme_direction = -1,
  legend_position = "right"
)
```

Arguments

network_table	The weight data table of network.
binwidth	Width of the bins.
show_border	Logical value, whether to show border of the bins.
border_color	Color of the border.
alpha	Alpha value of the bins.
theme	Theme of the bins.
theme_begin	Begin value of the theme.
theme_end	End value of the theme.
theme_direction	Direction of the theme.
legend_position	The position of legend.

Value

ggplot object

Examples

```
data("example_matrix")
network_table <- inferCSN(example_matrix)
plot_weight_distribution(network_table)
```

---

predict.srm *Predicts response for a given sample*

Description

Predicts response for a given sample

Usage

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

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

Arguments

object	The output of fit_sparse_regression.
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.srm.
gamma	The value of gamma to use for prediction. A summary of the gammas in the regularization path can be obtained using print.srm.
...	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

---

print.srm *Prints a summary of* fit_sparse_regression

Description

Prints a summary of fit_sparse_regression

Usage

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

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

Arguments

x	The output of fit_sparse_regression.
...	Other parameters

Value

Return information of fit_sparse_regression

r_square R^2 *(coefficient of determination)*

Description

R^2 (coefficient of determination)

Usage

```
r_square(y_true, y_pred)
```

Arguments

y_true	A numeric vector with ground truth values.
y_pred	A numeric vector with predicted values.

single_network *Construct network for single target gene*

Description

Construct network for single target gene

Usage

```
single_network(
  matrix,
  regulators,
  target,
  cross_validation = FALSE,
  seed = 1,
  penalty = "L0",
  algorithm = "CD",
  regulators_num = (ncol(matrix) - 1),
  n_folds = 10,
  percent_samples = 1,
  r_threshold = 0,
  verbose = FALSE,
  ...
)
```

Arguments

matrix	An expression matrix.
regulators	The regulator genes for which to infer the regulatory network.
target	The target gene.
cross_validation	Logical value, default is FALSE, whether to use cross-validation.
seed	The random seed for cross-validation, default is 1.
penalty	The type of regularization, default is L0. This can take either one of the following choices: L0, L0L1, and L0L2. For high-dimensional and sparse data, L0L2 is more effective.
algorithm	The type of algorithm used to minimize the objective function, default is CD. 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-zore 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.
n_folds	The number of folds for cross-validation, default is 10.
percent_samples	The percent of all samples used for sparse_regression, default is 1.
r_threshold	Threshold of R^2 or correlation coefficient, default is 0.
verbose	Logical value, default is TRUE, whether to print progress messages.
...	Parameters for other methods.

Value

The weight data table of sub-network

Examples

```
data("example_matrix")
head(
  single_network(
    example_matrix,
    regulators = colnames(example_matrix),
    target = "g1"
  )
)

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 = ncol(x),
  n_folds = 10,
  percent_samples = 1,
  r_threshold = 0,
  computation_method = "cor",
  verbose = TRUE,
  ...
)
```

Arguments

`x`	The matrix of regulators.
`y`	The vector of target.
`cross_validation`	Logical value, default is `FALSE`, whether to use cross-validation.
`seed`	The random seed for cross-validation, default is `1`.
`penalty`	The type of regularization, default is `L0`. This can take either one of the following choices: `L0`, `L0L1`, and `L0L2`. For high-dimensional and sparse data, `L0L2` is more effective.
`algorithm`	The type of algorithm used to minimize the objective function, default is `CD`. 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-zore 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.
`n_folds`	The number of folds for cross-validation, default is `10`.
`percent_samples`	The percent of all samples used for `sparse_regression`, default is `1`.

r_threshold	Threshold of R^2 or correlation coefficient, default is 0.
computation_method	The method used to compute r.
verbose	Logical value, default is TRUE, whether to print progress messages.
...	Parameters for other methods.

Value

Coefficients

Examples

```
data("example_matrix")
sparse_regression(
  example_matrix[, -1],
  example_matrix[, 1]
)
```

---

table_to_matrix *Switch network table to matrix*

Description

Switch network table to matrix

Usage

```
table_to_matrix(network_table, regulators = NULL, targets = NULL)
```

Arguments

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

Value

Weight matrix

Examples

```
data("example_matrix")
network_table <- inferCSN(example_matrix)
head(network_table)

table_to_matrix(network_table)[1:6, 1:6]

table_to_matrix(
```

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

Index