Package ‘inferCSN’

June 26, 2024

Type Package
Title Inferring Cell-Specific Gene Regulatory Network
Version 1.0.5
Date 2024-6-26
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, ggraph, igraph, Matrix, methods, parallel, patchwork, pbapply, purrr, Rcpp, RcppArmadillo, stats, utils
Suggests circlize, gtools, gganimate, ggExtra, ggdensity, ggpubr, igraph, network, plotly, precrec, pROC, testthat (>= 3.0.0), tidygraph, RColorBrewer, RTransferEntropy, viridis
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 2024-06-26 12:10:02 UTC
inferCSN-package

Description

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

Author(s)

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

Source

https://github.com/mengxu98/inferCSN

See Also

Useful links:

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

---

acc.calculate  

ACC calculate

Description

ACC calculate

Usage

acc.calculate(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)
acc.calculate(network_table, example_ground_truth)
as_matrix  
Attempts to turn a dgCMatrix into a dense matrix

Description
Attempts to turn a dgCMatrix into a dense matrix

Usage
as_matrix(x)

Arguments

x  A matrix.

Examples
sparse_matrix <- Matrix::sparseMatrix(
i = sample(1:200, 50),
j = sample(1:200, 50),
x = rnorm(50),
dims = c(200, 200),
dimnames = list(
paste0("a", rep(1:200)),
paste0("b", rep(1:200))
)
)

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

auc.calculate  
AUC value calculate

Description
AUC value calculate

Usage

auc.calculate(
  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

```r
data("example_matrix")
data("example_ground_truth")
network_table <- inferCSN(example_matrix)
auc.calculate(network_table, example_ground_truth, plot = TRUE)
```

---

**calculate.gene.rank**  
*Calculate and rank TFs in network*

Description

Calculate and rank TFs in network.

Usage

```r
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: If network is directed or not.

Value

A data.table with three columns.
Examples

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

---

check.parameters Check input parameters

Description

Check input parameters

Usage

check.parameters(
    matrix,
    penalty,
    algorithm,
    cross_validation,
    seed,
    n_folds,
    percent_samples,
    r_threshold,
    regulators,
    targets,
    regulators_num,
    verbose,
    cores,
    ...
)

Arguments

matrix An expression matrix, cells by genes
penalty The type of regularization. This can take either one of the following choices: $L_0$ and $L_0L_2$. For high-dimensional and sparse data, such as single-cell sequencing data, $L_0L_2$ is more effective.
algorithim 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.
% coef.SRM_fit

percent_samples
The percent of all samples used for sparse.regression. Default set to 1.

r_threshold
Threshold of $R^2$ or correlation coefficient.

regulators
A character vector with the regulators to consider for CSN inference.

targets
A character vector with the targets to consider for CSN inference.

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. \( 0.05 \times \min(n,p) \)) as L0 regularization typically selects a small portion of non-zeros.

verbose
Print detailed information.
cores
Number of CPU cores used. Setting to parallelize the computation with foreach.

... Parameters for other methods.

Value
Not return value, called for check input parameters

description

coef.SRM_fit

| Extracts a specific solution in the regularization path |

Description
Extracts a specific solution in the regularization path

Usage

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

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

Arguments

- object The output of \code{model.fit} or \code{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
Example ground truth data

Description
The data used for calculate the evaluating indicator.

Example matrix data

Description
The matrix used for reconstruct gene regulatory network.

Example meta data

Description
The data contains cells and pseudotime information.

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

library(inferCSN)
data("example_matrix")
network_table <- inferCSN(example_matrix)
weight_matrix <- table.to.matrix(network_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

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 = FALSE,
  ...
)

## 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 = FALSE,
...
)

## 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 = FALSE,
  ...
)

Arguments

object          The input data for inferCSN.
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.
percent_samples  The percent of all samples used for sparse.regression. Default set to 1.
r_threshold     Threshold of $R^2$ or correlation coefficient.
regulators      A character vector with the regulators to consider for CSN inference.
targets         A character vector with the targets to consider for CSN inference.
model.fit

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.
cores  Number of CPU cores used. Setting to parallelize the computation with foreach.
verbose  Print detailed information.
...
Parameters for other methods.

Value
A data table of gene-gene regulatory relationship

Examples

data("example_matrix")
network_table <- inferCSN(example_matrix, verbose = TRUE)
head(network_table)

network_table <- inferCSN(example_matrix, cores = 2)
head(network_table)

model.fit  Fit a sparse regression model

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

Usage
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,
model.fit

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)
network.heatmap

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>atol</td>
<td>The absolute tolerance which decides when to terminate optimization (based on the absolute L2 norm of the residuals)</td>
</tr>
<tr>
<td>activeSet</td>
<td>If TRUE, performs active set updates</td>
</tr>
<tr>
<td>activeSetNum</td>
<td>The number of consecutive times a support should appear before declaring support stabilization</td>
</tr>
<tr>
<td>maxSwaps</td>
<td>The maximum number of swaps used by CDPSI for each grid point</td>
</tr>
<tr>
<td>scaleDownFactor</td>
<td>This parameter decides how close the selected Lambda values are</td>
</tr>
<tr>
<td>screenSize</td>
<td>The number of coordinates to cycle over when performing initial correlation screening</td>
</tr>
<tr>
<td>autoLambda</td>
<td>Ignored parameter. Kept for backwards compatibility</td>
</tr>
<tr>
<td>lambdaGrid</td>
<td>A grid of Lambda values to use in computing the regularization path</td>
</tr>
<tr>
<td>excludeFirstK</td>
<td>This parameter takes non-negative integers</td>
</tr>
<tr>
<td>intercept</td>
<td>If FALSE, no intercept term is included in the model</td>
</tr>
<tr>
<td>lows</td>
<td>Lower bounds for coefficients</td>
</tr>
<tr>
<td>highs</td>
<td>Upper bounds for coefficients</td>
</tr>
<tr>
<td>...</td>
<td>Parameters for other methods.</td>
</tr>
</tbody>
</table>

Value

An S3 object describing the regularization path

Examples

```r
data("example_matrix")
fit <- model.fit(
  example_matrix[, -1],
  example_matrix[, 1]
)  
head(coef(fit))
```

Description

The heatmap of network
Usage

```r
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 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_lock` Lock the size of heatmap.
- `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

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

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

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

network.heatmap(
  network_table,
  show_names = TRUE,
  rect_color = "gray90",
  row_anno_type = "density",
  column_anno_type = "barplot"
)

network.heatmap(
  network_table,
  show_names = TRUE,
  rect_color = "gray90",
  row_anno_type = "density",
  column_anno_type = "barplot"
)
```
network_format

Description
Format weight 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, 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

```r
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")
)
```

Description

network_sift

Usage

```r
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, 
  history_length = 1, 
  entropy_p_value = 0.05, 
  cores = 1, 
  verbose = TRUE
)
```
Arguments

- `network_table` network_table
- `matrix` The expression matrix.
- `meta_data` The meta data for cells or samples.
- `pseudotime_column` The column of pseudotime.
- `method` 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` Logical value, using effective entropy to filter weights or not.
- `shuffles` The number of shuffles used to calculate the effective transfer entropy. Default is 'shuffles' = 100.
- `entropy_nboot` entropy_nboot
- `history_length` history_length
- `entropy_p_value` P value.
- `cores` Number of CPU cores used. Setting to parallelize the computation with `foreach`.
- `verbose` Print detailed information.

Value

Filtered network table

Examples

```r
data("example_matrix")
data("example_ground_truth")
network_table <- inferCSN(example_matrix)
network_table_filtered <- network_sift(network_table)
data("example_meta_data")
network_table_filtered_entropy <- network_sift(
  network_table, 
  matrix = example_matrix, 
  meta_data = example_meta_data, 
  pseudotime_column = "pseudotime", 
  history_length = 2, 
  shuffles = 0, 
  entropy_nboot = 0 
)

network.heatmap(
  example_ground_truth[, 1:3], 
  heatmap_title = "Ground truth", 
  show_names = TRUE, 
  rect_color = "gray70" 
)
```
network.heatmap(
    network_table,
    heatmap_title = "Raw",
    show_names = TRUE,
    rect_color = "gray70"
)

network.heatmap(
    network_table_filtered,
    heatmap_title = "Filtered",
    show_names = TRUE,
    rect_color = "gray70"
)

network.heatmap(
    network_table_filtered_entropy,
    heatmap_title = "Filtered by entropy",
    show_names = TRUE,
    rect_color = "gray70"
)

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

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

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

---

**normalization**

**Description**

normalization

**Usage**

normalization(x, method = "max_min")

**Arguments**

- **x** A numeric vector.
- **method** Method for normalization.
**parallelize_fun**  
*Apply function over a List or Vector*

**Description**  
Apply function over a List or Vector

**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`: cores.
- `export_fun`: export_fun.
- `verbose`: Logical. Whether to print progress bar. Only works in sequential mode.

**Value**  
A list.

---

**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
- weight_value: weight_value
- legend_position: The position of legend.

Value

Return a ggplot2 object

Examples

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

Description

plot_dynamic_networks

Usage

```r
plot_dynamic_networks(
  network_table,  # The weight data table of network.
  celltypes_order,
  ntop = 10,
  width = 6,
  height = 6,
  seed = 2024,
  theme_type = "theme_void",
  plot_type = "ggplot",
  layout = "fruchtermanreingold",
  nrow = 2,
  title = NULL,
  figure_save = FALSE,
  figure_name = NULL
)
```
plot_dynamic_networks

Arguments

- `network_table`: ggplot object
- `celltypes_order`: celltypes_order
- `ntop`: ntop
- `width`: width
- `height`: height
- `seed`: seed
- `theme_type`: theme_type
- `plot_type`: plot_type
- `layout`: layout
- `nrow`: nrow
- `title`: The title of figure.
- `figure_save`: figure_save
- `figure_name`: figure_name

Value

- ggplot object

Examples

```r
data("example_matrix")
network <- inferCSN(example_matrix)[1:100, ]
network$celltype <- c(
  rep("cluster5", 20),
  rep("cluster1", 20),
  rep("cluster3", 20),
  rep("cluster2", 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]
)
```
## 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_dynamic_networks(
  network,
  celltypes_order = celltypes_order,
  plot_type = "ggplotly"
)

---

**plot_scatter**  
**plot_scatter**

**Description**

`plot_scatter`

**Usage**

```r
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 = FALSE,
  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", "density").
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 invalidation.
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,
keep_aspect_ratio = TRUE
)
p2 <- plot_scatter(
test_data,
marginal_type = "boxplot",
keep_aspect_ratio = TRUE
)
p1 + p2

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

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

---

plot_static_networks  
Plot of dynamic networks

Description
Plot of 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.
predict.SRM_fit

Value
A list of ggplot2 objects

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]
)
```

predict.SRM_fit  Predict Response

Description
Predicts response for a given sample

Usage
```
## 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

Description

prepare.performance.data

Usage

prepare.performance.data(network_table, ground_truth)

Arguments

network_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

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

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

\( x \quad \text{The output of model.fit or inferCSN.cvfit} \\
\ldots \quad \text{Other parameters} \\

Value

Return information of model.fit
Return information of model.fit

\[ \text{rse} \quad \text{Relative Squared Error} \]

Description

Relative Squared Error

Usage

\[ \text{rse}(y_{\text{true}}, y_{\text{pred}}) \]

Arguments

\( y_{\text{true}} \quad \text{A numeric vector with ground truth values.} \\
\text{y\_pred} \quad \text{A numeric vector with predicted values.} \)

\[ \text{r\_square} \quad \text{R}^2 \ (\text{coefficient of determination}) \]

Description

\( R^2 \) (coefficient of determination)

Usage

\[ \text{r\_square}(y_{\text{true}}, y_{\text{pred}}) \]

Arguments

\( y_{\text{true}} \quad \text{A numeric vector with ground truth values.} \\
\text{y\_pred} \quad \text{A numeric vector with predicted values.} \)
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 = (ncol(matrix) - 1),
  n_folds = 10,
  percent_samples = 1,
  r_threshold = 0,
  verbose = FALSE,
  ...
)

Arguments

matrix An expression matrix, cells by genes.
regulators A character vector with the regulators to consider for CSN inference.
target Target gene.
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 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.
percent_samples

The percent of all samples used for `sparse.regression`. Default set to 1.

r_threshold

Threshold of $R^2$ or correlation coefficient.

verbose

Print detailed information.

...

Parameters for other methods.

Value

The weight data table of sub-network

Examples

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

```r
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 = 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 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.

- **percent_samples**  
  The percent of all samples used for `sparse.regression`. Default set to 1.

- **r_threshold**  
  Threshold of $R^2$ or correlation coefficient.

- **computation_method**  
  The method used to compute $r$.

- **verbose**  
  Print detailed information.

- **...**  
  Parameters for other methods.

**Value**

Coefficients

**Examples**

```r
data("example_matrix")
sparse.regression(
  example_matrix[, -1],
  example_matrix[, 1]
)
```
### sse

*Sum of Squared Errors*

**Description**

Sum of Squared Errors

**Usage**

```r
sse(y_true, y_pred)
```

**Arguments**

- `y_true`: A numeric vector with ground truth values.
- `y_pred`: A numeric vector with predicted values.

### table.to.matrix

*Switch weight table to matrix*

**Description**

Switch weight table to matrix

**Usage**

```r
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

acc.calculate, 3
as_matrix, 4
auc.calculate, 4

calculate.gene.rank, 5
check.parameters, 6
coeff.SRM_fit, 7
coeff.SRM_fit_CV(coeff.SRM_fit), 7

eexample_ground_truth, 8
eexample_matrix, 8
eexample_meta_data, 8

filter_sort_matrix, 8
foreach, 7, 11, 18

inferCSN, 9
inferCSN.data.frame-method(inferCSN), 9
inferCSN.matrix-method(inferCSN), 9
inferCSN-package, 2

model.fit, 11

network.heatmap, 13
network_format, 16
network_sift, 17
normalization, 19

parallelize_fun, 20
plot_contrast_networks, 20
plot_dynamic_networks, 21
plot_scatter, 23
plot_static_networks, 25
predict.SRM_fit, 26
predict.SRM_fit_CV(predict.SRM_fit), 26
prepare.performance.data, 27
print.SRM_fit, 27
print.SRM_fit_CV(print.SRM_fit), 27

r_square, 28
rse, 28