

# Package ‘inferCSN’

March 27, 2025

**Type** Package

**Title** Inferring Cell-Specific Gene Regulatory Network

**Version** 1.1.6

**Date** 2025-03-27

**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, pbapply, purrr, Rcpp, RcppArmadillo, RcppParallel, stats

**Suggests** ComplexHeatmap, circlize, gtools, ganimate, ggExtra, ggpointdensity, ggpubr, igraph, irlba, network, patchwork, plotly, precrec, pROC, proxy, tidygraph, RANN, 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** 2025-03-27 04:10:05 UTC

## Contents

inferCSN-package . . . . .	3
as_matrix . . . . .	3
calculate_accuracy . . . . .	5
calculate_auc . . . . .	5
calculate_auprc . . . . .	6
calculate_auroc . . . . .	7
calculate_f1 . . . . .	8
calculate_gene_rank . . . . .	9
calculate_ji . . . . .	9
calculate_metrics . . . . .	10
calculate_precision . . . . .	11
calculate_recall . . . . .	12
calculate_si . . . . .	13
check_sparsity . . . . .	14
coef.srm . . . . .	14
example_ground_truth . . . . .	15
example_matrix . . . . .	15
example_meta_data . . . . .	15
filter_sort_matrix . . . . .	15
fit_srm . . . . .	16
inferCSN . . . . .	17
log_message . . . . .	21
matrix_to_table . . . . .	22
meta_cells . . . . .	23
network_format . . . . .	25
network_sift . . . . .	26
normalization . . . . .	28
parallelize_fun . . . . .	29
pearson_correlation . . . . .	30
plot_coefficient . . . . .	30
plot_coefficients . . . . .	31
plot_contrast_networks . . . . .	32
plot_dynamic_networks . . . . .	32
plot_edges_comparison . . . . .	34
plot_embedding . . . . .	35
plot_histogram . . . . .	36
plot_network_heatmap . . . . .	38
plot_scatter . . . . .	40
plot_static_networks . . . . .	42
predict.srm . . . . .	43
print.srm . . . . .	44
r_square . . . . .	45
simulate_sparse_matrix . . . . .	45
single_network . . . . .	46
sparse_cor . . . . .	47
sparse_regression . . . . .	49

<i>inferCSN-package</i>	3
split_indices . . . . .	51
subsampling . . . . .	52
table_to_matrix . . . . .	53
weight_sift . . . . .	54
%ss% . . . . .	55
<b>Index</b>	<b>56</b>

---

inferCSN-package	<b>inferCSN: inferring Cell-Specific gene regulatory Network</b>
------------------	--

---

### Description

An R package for **inferring Cell-Specific gene regulatory Network** 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	<i>Convert sparse matrix into dense matrix</i>
-----------	--

---

### Description

Convert sparse matrix into 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 <a href="#">setThreadOptions</a> .
sparse	Logical value, default is FALSE, whether to output a sparse matrix.

**Examples**

```
sparse_matrix <- simulate_sparse_matrix(  
  2000,  
  2000,  
  density = 0.01  
)  
  
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_0 <- inferCSN(example_matrix)  
  
network_table_1 <- inferCSN(  
  as_matrix(example_matrix, sparse = TRUE)  
)  
network_table_2 <- inferCSN(  
  as(example_matrix, "sparseMatrix")  
)  
  
plot_scatter(  
  data.frame(  
    network_table_0$weight,  
    network_table_1$weight  
  ),  
  legend_position = "none"  
)  
  
plot_scatter(  
  data.frame(  
    network_table_1$weight,  
    network_table_2$weight  
  ),  
  legend_position = "none"  
)
```

```
## End(Not run)
```

---

calculate\_accuracy      *Calculate Accuracy*

---

### Description

Calculates accuracy metric

### Usage

```
calculate_accuracy(network_table, ground_truth)
```

### Arguments

network\_table    A data frame of predicted network structure  
ground\_truth     A data frame of ground truth network

### Value

A list containing the metric

### Examples

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

---

calculate\_auc            *Calculate AUC Metrics*

---

### Description

Calculates AUROC and AUPRC metrics with optional visualization

### Usage

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

**Arguments**

network_table	A data frame of predicted network structure
ground_truth	A data frame of ground truth network
return_plot	Logical value indicating whether to generate plots
line_color	Color for plot lines
line_width	Width for plot lines
tag_levels	Tag levels for plot annotations

**Value**

A list containing metrics and optional plots

**Examples**

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

---

calculate_auprc	<i>Calculate AUPRC Metric</i>
-----------------	-------------------------------

---

**Description**

Calculates AUPRC metric with optional visualization

**Usage**

```
calculate_auprc(
  network_table,
  ground_truth,
  return_plot = FALSE,
  line_color = "#1563cc",
  line_width = 1
)
```

**Arguments**

network_table	A data frame of predicted network structure
ground_truth	A data frame of ground truth network
return_plot	Logical value indicating whether to generate plot
line_color	Color for plot lines
line_width	Width for plot lines

**Value**

A list containing metric and optional plot

**Examples**

```
data("example_matrix")
data("example_ground_truth")
network_table <- inferCSN(example_matrix)
calculate_auprc(
  network_table,
  example_ground_truth,
  return_plot = TRUE
)
```

---

calculate_auroc	<i>Calculate AUROC Metric</i>
-----------------	-------------------------------

---

**Description**

Calculates AUROC metric with optional visualization

**Usage**

```
calculate_auroc(
  network_table,
  ground_truth,
  return_plot = FALSE,
  line_color = "#1563cc",
  line_width = 1
)
```

**Arguments**

network_table	A data frame of predicted network structure
ground_truth	A data frame of ground truth network
return_plot	Logical value indicating whether to generate plot
line_color	Color for plot lines
line_width	Width for plot lines

**Value**

A list containing metric and optional plot

**Examples**

```
data("example_matrix")
data("example_ground_truth")
network_table <- inferCSN(example_matrix)
calculate_auroc(
  network_table,
  example_ground_truth,
  return_plot = TRUE
)
```

---

`calculate_f1`*Calculate F1 Score*

---

**Description**

Calculates F1 score

**Usage**

```
calculate_f1(network_table, ground_truth)
```

**Arguments**

`network_table` A data frame of predicted network structure  
`ground_truth` A data frame of ground truth network

**Value**

A list containing the metric

**Examples**

```
data("example_matrix")
data("example_ground_truth")
network_table <- inferCSN(example_matrix)
calculate_f1(
  network_table,
  example_ground_truth
)
```



---

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"))
```

---

calculate\_ji     *Calculate Jaccard Index*

---

**Description**

Calculates Jaccard Index (JI) metric

**Usage**

```
calculate_ji(network_table, ground_truth)
```

**Arguments**

network\_table A data frame of predicted network structure  
ground\_truth A data frame of ground truth network

**Value**

A list containing the metric

**Examples**

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

---

calculate\_metrics *Calculate Network Prediction Performance Metrics*

---

**Description**

Calculates comprehensive performance metrics for evaluating predicted network structures, including classification performance, precision-recall metrics, and network topology metrics.

**Usage**

```
calculate_metrics(
  network_table,
  ground_truth,
  metric_type = c("all", "auc", "auROC", "auprc", "precision", "recall", "f1",
    "accuracy", "si", "ji"),
  return_plot = FALSE,
  line_color = "#1563cc",
  line_width = 1
)
```

**Arguments**

network\_table A data frame of predicted network structure containing:

- regulator - Source nodes of the network edges
- target - Target nodes of the network edges
- weight - Edge weights representing prediction confidence

ground\_truth A data frame of ground truth network with the same format as network\_table.

metric_type	The type of metric to return, default is all. This can take any of the following choices: <ul style="list-style-type: none"> <li>• all - Returns all available metrics with <i>Performance Metrics</i> plot</li> <li>• auc - Returns both AUROC and AUPRC with their plots</li> <li>• auroc - Area Under ROC Curve with plot</li> <li>• auprc - Area Under Precision-Recall Curve with plot</li> <li>• precision - Proportion of correct predictions among positive predictions</li> <li>• recall - Proportion of actual positives correctly identified</li> <li>• f1 - Harmonic mean of precision and recall</li> <li>• accuracy - Overall classification accuracy</li> <li>• si - Set Intersection, counting correctly predicted edges</li> <li>• ji - Jaccard Index, measuring overlap between predicted and true networks</li> </ul>
return_plot	Logical value, default is FALSE, whether to generate visualization plots
line_color	Color for plot lines, default is #1563cc
line_width	Width for plot lines, default is 1

**Value**

A list containing:

- metrics - A data frame with requested metrics
- plot - A plot object if return\_plot = TRUE (optional)

**Examples**

```
data("example_matrix")
data("example_ground_truth")
network_table <- inferCSN(example_matrix)
calculate_metrics(
  network_table,
  example_ground_truth,
  return_plot = TRUE
)
calculate_metrics(
  network_table,
  example_ground_truth,
  metric_type = "auroc"
)
```

---

calculate\_precision     *Calculate Precision Metric*

---

**Description**

Calculates precision metric

**Usage**

```
calculate_precision(network_table, ground_truth)
```

**Arguments**

`network_table` A data frame of predicted network structure  
`ground_truth` A data frame of ground truth network

**Value**

A list containing the metric

**Examples**

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

---

calculate_recall	<i>Calculate Recall Metric</i>
------------------	--------------------------------

---

**Description**

Calculates recall metric

**Usage**

```
calculate_recall(network_table, ground_truth)
```

**Arguments**

`network_table` A data frame of predicted network structure  
`ground_truth` A data frame of ground truth network

**Value**

A list containing the metric

**Examples**

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

---

`calculate_si`*Calculate Set Intersection*

---

**Description**

Calculates Set Intersection (SI) metric

**Usage**

```
calculate_si(network_table, ground_truth)
```

**Arguments**

`network_table` A data frame of predicted network structure  
`ground_truth` A data frame of ground truth network

**Value**

A list containing the metric

**Examples**

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

---

check_sparsity	<i>Check sparsity of matrix</i>
----------------	---------------------------------

---

**Description**

Check sparsity of matrix

**Usage**

```
check_sparsity(x)
```

**Arguments**

x	A matrix.
---	-----------

**Value**

Sparsity of matrix

---

coef.srm	<i>Extracts a specific solution in the regularization path</i>
----------	--

---

**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 <a href="#">sparse_regression</a> .
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-zero coefficients, this value will affect the final performance. The maximum support size at which to terminate the regularization path.
...	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\_srm

*Sparse regression model***Description**

Sparse regression model

**Usage**

```

fit_srm(
  x,
  y,
  cross_validation = FALSE,
  seed = 1,
  penalty = "L0",
  regulators_num = ncol(x),
  n_folds = 5,
  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.
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.
n_folds	The number of folds for cross-validation, default is 5.
verbose	Logical value, default is TRUE, whether to print progress messages.
...	Parameters for other methods.



**Value**

A list of the sparse regression model. The list has the following components:

model	The sparse regression model.
metrics	A list of metrics.
coefficients	A list of coefficients.

**Examples**

```
data("example_matrix")
fit_srm(
  x = example_matrix[, -1],
  y = example_matrix[, 1]
)
```

---

inferCSN

**inferring Cell-Specific gene regulatory Network**


---

**Description**

**inferring Cell-Specific gene regulatory Network**

**Usage**

```
inferCSN(
  object,
  penalty = "L0",
  cross_validation = FALSE,
  seed = 1,
  n_folds = 5,
  subsampling_method = c("sample", "meta_cells", "pseudobulk"),
  subsampling_ratio = 1,
  r_squared_threshold = 0,
  regulators = NULL,
  targets = NULL,
  cores = 1,
  verbose = TRUE,
  ...
)

## S4 method for signature 'matrix'
inferCSN(
  object,
  penalty = "L0",
  cross_validation = FALSE,
  seed = 1,
  n_folds = 5,
```

```

    subsampling_method = c("sample", "meta_cells", "pseudobulk"),
    subsampling_ratio = 1,
    r_squared_threshold = 0,
    regulators = NULL,
    targets = NULL,
    cores = 1,
    verbose = TRUE,
    ...
)

## S4 method for signature 'sparseMatrix'
inferCSN(
  object,
  penalty = "L0",
  cross_validation = FALSE,
  seed = 1,
  n_folds = 5,
  subsampling_method = c("sample", "meta_cells", "pseudobulk"),
  subsampling_ratio = 1,
  r_squared_threshold = 0,
  regulators = NULL,
  targets = NULL,
  cores = 1,
  verbose = TRUE,
  ...
)

## S4 method for signature 'data.frame'
inferCSN(
  object,
  penalty = "L0",
  cross_validation = FALSE,
  seed = 1,
  n_folds = 5,
  subsampling_method = c("sample", "meta_cells", "pseudobulk"),
  subsampling_ratio = 1,
  r_squared_threshold = 0,
  regulators = NULL,
  targets = 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 follow-

	ing choices: $L_0$ , $L_{0L1}$ , and $L_{0L2}$ . For high-dimensional and sparse data, $L_{0L2}$ is more effective.
<code>cross_validation</code>	Logical value, default is FALSE, whether to use cross-validation.
<code>seed</code>	The random seed for cross-validation, default is 1.
<code>n_folds</code>	The number of folds for cross-validation, default is 5.
<code>subsampling_method</code>	The method to use for subsampling. Options are "sample", "pseudobulk" or "meta_cells".
<code>subsampling_ratio</code>	The percent of all samples used for <code>fit_srm</code> , default is 1.
<code>r_squared_threshold</code>	Threshold of $R^2$ coefficient, default is 0.
<code>regulators</code>	The regulator genes for which to infer the regulatory network.
<code>targets</code>	The target genes for which to infer the regulatory network. Recommend setting this to a small fraction of $\min(n,p)$ (e.g. $0.05 * \min(n,p)$ ) as $L_0$ regularization typically selects a small portion of non-zeros.
<code>cores</code>	The number of cores to use for parallelization with <code>foreach</code> , default is 1.
<code>verbose</code>	Logical value, default is TRUE, whether to print progress messages.
<code>...</code>	Parameters for other methods.

**Value**

A data table of regulator-target regulatory relationships

**Examples**

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

network_table_2 <- inferCSN(
  example_matrix,
  cores = 2
)

head(network_table_1)

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")
)

## Not run:
data("example_ground_truth")
network_table_07 <- inferCSN(
  example_matrix,
  r_squared_threshold = 0.7
)
calculate_metrics(
  network_table_1,
  example_ground_truth,
  return_plot = TRUE
)
calculate_metrics(
  network_table_07,
  example_ground_truth,
  return_plot = TRUE
)

## End(Not run)
## 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	<i>Print diagnostic message</i>
-------------	---------------------------------

---

### Description

Integrate the message printing function with the `cli` package, and the `message` function. The message could be suppressed by `suppressMessages`.

### Usage

```

log_message(
  ...,
  verbose = TRUE,
  message_type = c("info", "success", "warning", "error"),
  cli_model = TRUE
)

```

### Arguments

<code>...</code>	Text to print.
<code>verbose</code>	Logical value, default is TRUE. Whether to print the message.
<code>message_type</code>	Type of message, default is info. Could be choose one of info, success, warning, and error.
<code>cli_model</code>	Logical value, default is TRUE. Whether to use the <code>cli</code> package to print the message.

### Examples

```

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

```

---

matrix_to_table	<i>Switch matrix to network table</i>
-----------------	---------------------------------------

---

**Description**

Switch matrix to network table

**Usage**

```
matrix_to_table(  
  network_matrix,  
  regulators = NULL,  
  targets = NULL,  
  threshold = 0  
)
```

**Arguments**

`network_matrix` The matrix of network weight.  
`regulators` Character vector of regulator names to filter by.  
`targets` Character vector of target names to filter by.  
`threshold` The threshold for filtering weights based on absolute values, defaults to 0.

**Value**

Network table

**Examples**

```
data("example_matrix")  
network_table <- inferCSN(example_matrix)  
network_matrix <- table_to_matrix(network_table)  
network_table_new <- matrix_to_table(network_matrix)  
head(network_table)  
head(network_table_new)  
identical(  
  network_table,  
  network_table_new  
)  
  
matrix_to_table(  
  network_matrix,  
  threshold = 0.8  
)  
  
matrix_to_table(  
  network_matrix,  
  regulators = c("g1", "g2"),
```

```
    targets = c("g3", "g4")
  )
```

---

meta\_cells

*Detection of metacells from single-cell gene expression matrix*

---

## Description

This function detects metacells from a single-cell gene expression matrix using dimensionality reduction and clustering techniques.

## Usage

```
meta_cells(
  matrix,
  genes_use = NULL,
  genes_exclude = NULL,
  var_genes_num = min(1000, nrow(matrix)),
  gamma = 10,
  knn_k = 5,
  do_scale = TRUE,
  pc_num = 25,
  fast_pca = FALSE,
  do_approx = FALSE,
  approx_num = 20000,
  directed = FALSE,
  use_nn2 = TRUE,
  seed = 1,
  cluster_method = "walktrap",
  block_size = 10000,
  weights = NULL,
  do_median_norm = FALSE,
  ...
)
```

## Arguments

matrix	A gene expression matrix where rows represent genes and columns represent cells.
genes_use	Default is NULL. A character vector specifying genes to be used for PCA dimensionality reduction.
genes_exclude	Default is NULL. A character vector specifying genes to be excluded from PCA computation.
var_genes_num	Default is <code>min(1000, nrow(matrix))</code> . Number of most variable genes to select when <code>genes_use</code> is not provided.

gamma	Default is 10. Coarse-graining parameter defining the target ratio of input cells to output metacells (e.g., gamma=10 yields approximately n/10 metacells for n input cells).
knn_k	Default is 5. Number of nearest neighbors for constructing the cell-cell similarity network.
do_scale	Default is TRUE. Whether to standardize (center and scale) gene expression values before PCA.
pc_num	Default is 25. Number of principal components to retain for downstream analysis.
fast_pca	Default is TRUE. Whether to use the faster <code>irlba</code> algorithm instead of standard PCA. Recommended for large datasets.
do_approx	Default is FALSE. Whether to use approximate nearest neighbor search for datasets with >50000 cells to improve computational efficiency.
approx_num	Default is 20000. Number of cells to randomly sample for approximate nearest neighbor computation when <code>do_approx = TRUE</code> .
directed	Default is FALSE. Whether to construct a directed or undirected nearest neighbor graph.
use_nn2	Default is TRUE. Whether to use the faster <code>RANN::nn2</code> algorithm for nearest neighbor search (only applicable with Euclidean distance).
seed	Default is 1. Random seed for reproducibility when subsampling cells in approximate mode.
cluster_method	Default is <code>walktrap</code> . Algorithm for community detection in the cell similarity network. Options: <code>walktrap</code> (recommended) or <code>louvain</code> (gamma parameter ignored).
block_size	Default is 10000. Number of cells to process in each batch when mapping cells to metacells in approximate mode. Adjust based on available memory.
weights	Default is NULL. Numeric vector of cell-specific weights for weighted averaging when computing metacell expression profiles. Length must match number of cells.
do_median_norm	Default is FALSE. Whether to perform median-based normalization of the final metacell expression matrix.
...	Additional arguments passed to internal functions.

### Value

A matrix where rows represent metacells and columns represent genes.

### References

<https://github.com/GfellerLab/SuperCell> <https://github.com/kuijjerlab/SCORPION>



**Examples**

```
data("example_matrix")
meta_cells_matrix <- meta_cells(
  example_matrix
)
dim(meta_cells_matrix)
meta_cells_matrix[1:6, 1:6]
```

---

network_format	<i>Format network table</i>
----------------	-----------------------------

---

**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**

Formatted network table

**Examples**

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

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

```

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

network_format(
  network_table,
  targets = "g3"
)

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

```

---

network_sift	<i>Sifting network</i>
--------------	------------------------

---

## 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 <code>foreach</code> , 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_metrics(
network_table,
example_ground_truth,
return_plot = TRUE
)
calculate_metrics(
network_table_sifted,
example_ground_truth,
return_plot = TRUE
)
calculate_metrics(
network_table_sifted_entropy,
example_ground_truth,
return_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.  
 ...            Parameters for other methods.

**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

---

pearson\_correlation     *Correlation and covariance calculation for sparse matrix*

---

**Description**

Correlation and covariance calculation for sparse matrix

**Usage**

```
pearson_correlation(x, y = NULL)
```

**Arguments**

x	Sparse matrix or character vector.
y	Sparse matrix or character vector.

---

plot\_coefficient     *Plot coefficients*

---

**Description**

Plot coefficients

**Usage**

```
plot_coefficient(
  data,
  style = "continuous",
  positive_color = "#3d67a2",
  negative_color = "#c82926",
  neutral_color = "#cccccc",
  bar_width = 0.7,
  text_size = 3,
  show_values = TRUE
)
```

**Arguments**

data	Input data.
style	Plotting style: "binary", "gradient", or "continuous".
positive_color	Color for positive weights.
negative_color	Color for negative weights.
neutral_color	Color for weights near zero (used in "continuous" style).
bar_width	Width of the bars.
text_size	Size of the text for weight values.
show_values	Logical, whether to show weight values on bars.

**Value**

A ggplot object

**Examples**

```
data("example_matrix")
network_table <- inferCSN(example_matrix, targets = "g1")
plot_coefficient(network_table)
plot_coefficient(network_table, style = "binary")
```

---

plot\_coefficients      *Plot coefficients for multiple targets*

---

**Description**

Plot coefficients for multiple targets

**Usage**

```
plot_coefficients(data, targets = NULL, nrow = NULL, ...)
```

**Arguments**

data	Input data.
targets	Targets to plot.
nrow	Number of rows for the plot.
...	Other arguments passed to <code>plot_coefficient</code> .

**Value**

A list of ggplot objects

**Examples**

```
data("example_matrix")
network_table <- inferCSN(
  example_matrix,
  targets = c("g1", "g2", "g3")
)
plot_coefficients(network_table, show_values = FALSE)
plot_coefficients(network_table, targets = "g1")
```

---

`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\_edges\_comparison *Network Edge Comparison Visualization*

---

### Description

Generates visualizations comparing edges of two networks.

### Usage

```
plot_edges_comparison(
```

```

network_table,
ground_truth,
color_pattern = list(predicted = "gray", ground_truth = "#bb141a", overlap = "#1966ad",
total = "#6C757D")
)

```

### Arguments

**network\_table** A data frame of predicted network structure.

**ground\_truth** A data frame of ground truth network.

**color\_pattern** A list of colors for different categories, with default values:

- `*'predicted'*` - Color for predicted edges (`*'gray'*`)
- `*'ground_truth'*` - Color for ground truth edges (`*'bb141a'*`)
- `*'overlap'*` - Color for overlapping edges (`*'1966ad'*`)
- `*'total'*` - Color for total counts (`*'6C757D'*`)

### Value

A patchwork plot object containing network edge comparison and distribution plots

### Examples

```

data("example_matrix")
data("example_ground_truth")
network_table <- inferCSN(example_matrix)
plot_edges_comparison(
  network_table,
  example_ground_truth
)

```

---

plot_embedding	<i>Plot embedding</i>
----------------	-----------------------

---

### 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 <a href="#">umap</a> and <a href="#">Rtsne</a> .

**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\_histogram

*Plot histogram*

---

**Description**

Plot histogram

**Usage**

```
plot_histogram(  
  data,  
  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**

data	A numeric vector.
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**

A ggplot object

**Examples**

```
data("example_matrix")  
network_table <- inferCSN(example_matrix)  
plot_histogram(network_table[, 3])
```

---

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 = c("boxplot", "barplot", "histogram", "density", "lines", "points",  
    "horizon"),  
  column_anno_type = c("boxplot", "barplot", "histogram", "density", "lines", "points"),  
  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 boxplot, could add a annotation plot to row, choose one of boxplot, barplot, histogram, density, lines, points, and horizon.
column_anno_type	Default is boxplot, 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)
## Not run:
p3 <- plot_network_heatmap(
  network_table,
  legend_name = "Weight1",
  heatmap_color = c("#20a485", "#410054", "#fee81f")
)
p4 <- plot_network_heatmap(
  network_table,
  legend_name = "Weight2",
  heatmap_color = c("#20a485", "white", "#fee81f")
)
ComplexHeatmap::draw(p3 + p4)

## End(Not run)

```

```

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", "g3", "g5"),
  targets = c("g3", "g6", "g9"),
  show_names = TRUE
)
## Not run:
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
)

## End(Not run)

```

---

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 inactivation.
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 = "g1"
)
plot_static_networks(
  network_table,
  targets = "g1"
)
plot_static_networks(
  network_table,
  regulators = "g1",
  targets = "g2"
)
```

---

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 <code>sparse_regression</code> .
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 <a href="#">print.srm</a> .
gamma	The value of gamma to use for prediction. A summary of the gammas in the regularization path can be obtained using <a href="#">print.srm</a> .
...	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	<i>Prints a summary of sparse_regression</i>
-----------	--

---

**Description**

Prints a summary of `sparse_regression`

**Usage**

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

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

**Arguments**

x	The output of <a href="#">sparse_regression</a> .
...	Other parameters

**Value**

Return information of `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.

---

simulate_sparse_matrix	<i>Generate a simulated sparse matrix for single-cell data testing</i>
------------------------	--

---

**Description**

Generate a simulated sparse matrix for single-cell data testing

**Usage**

```
simulate_sparse_matrix(
  nrow,
  ncol,
  density = 0.1,
  distribution_fun = function(n) stats::rpois(n, lambda = 0.5) + 1,
  seed = 1
)
```

**Arguments**

nrow	Number of rows (genes) in the matrix.
ncol	Number of columns (cells) in the matrix.
density	Density of non-zero elements (default: 0.1, representing 90 sparsity).
distribution_fun	Function to generate non-zero values.
seed	Random seed for reproducibility.

**Value**

A sparse matrix of class "dgCMatrix"

**Examples**

```
simulate_sparse_matrix(2000, 500) |>
  check_sparsity()
```

---

single_network	<i>Construct network for single target gene</i>
----------------	---

---

**Description**

Construct network for single target gene

**Usage**

```
single_network(
  matrix,
  regulators,
  target,
  cross_validation = FALSE,
  seed = 1,
  penalty = "L0",
  r_squared_threshold = 0,
  n_folds = 5,
  verbose = TRUE,
  ...
)
```

**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.
r_squared_threshold	Threshold of $R^2$ coefficient, default is 0.
n_folds	The number of folds for cross-validation, default is 5.
verbose	Logical value, default is TRUE, whether to print progress messages.
...	Parameters for other methods.

**Value**

A data frame of the single target gene network. The data frame has three columns:

regulator	The regulator genes.
target	The target gene.
weight	The weight of the regulator gene on the target gene.

**Examples**

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

single_network(
  example_matrix,
  regulators = c("g1", "g2", "g3"),
  target = "g1"
)
single_network(
  example_matrix,
  regulators = c("g1", "g2"),
  target = "g1"
)
```

---

sparse_cor	<i>Safe correlation function which returns a sparse matrix without missing values</i>
------------	---

---

**Description**

Safe correlation function which returns a sparse matrix without missing values

**Usage**

```
sparse_cor(  
  x,  
  y = NULL,  
  method = "pearson",  
  allow_neg = TRUE,  
  remove_na = TRUE,  
  remove_inf = TRUE,  
  ...  
)
```

**Arguments**

x	Sparse matrix or character vector.
y	Sparse matrix or character vector.
method	Method to use for calculating the correlation coefficient.
allow_neg	Logical. Whether to allow negative values or set them to 0.
remove_na	Logical. Whether to replace NA values with 0.
remove_inf	Logical. Whether to replace infinite values with 1.
...	Other arguments passed to <code>cor</code> function.

**Value**

A correlation matrix.

**Examples**

```
m1 <- simulate_sparse_matrix(  
  1000, 1000,  
  density = 0.01  
)  
m2 <- simulate_sparse_matrix(  
  1000, 500,  
  density = 0.01  
)  
  
all.equal(  
  as.matrix(sparse_cor(m1)),  
  cor(as_matrix(m1))  
)  
all.equal(  
  as.matrix(sparse_cor(m1, m2)),  
  cor(as_matrix(m1), as_matrix(m2))  
)  
  
system.time(  
  sparse_cor(m1)  
)  
system.time(  
  cor(as_matrix(m1))  
)
```



```

    cor(as_matrix(m1))
  )
system.time(
  sparse_cor(m1, m2)
)
system.time(
  cor(as_matrix(m1), as_matrix(m2))
)

# add missing values
m1[sample(1:500, 10)] <- NA
m2[sample(1:500, 10)] <- NA

sparse_cor(m1, m2)[1:5, 1:5]

```

---

sparse\_regression      *Fit a sparse regression model*

---

## Description

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

## Usage

```

sparse_regression(
  x,
  y,
  penalty = "L0",
  algorithm = c("CD", "CDPSI"),
  regulators_num = ncol(x),
  cross_validation = FALSE,
  n_folds = 5,
  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,
verbose = TRUE,
...
)

```

### 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-zero coefficients, this value will affect the final performance. The maximum support size at which to terminate the regularization path.
cross_validation	Logical value, default is FALSE, whether to use cross-validation.
n_folds	The number of folds for cross-validation, default is 5.
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.
verbose	Logical value, default is TRUE, whether to print progress messages.
...	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 <- sparse_regression(
  example_matrix[, -1],
  example_matrix[, 1]
)
head(coef(fit))
```

---

split\_indices

*Split indices.*

---

**Description**

An optimised version of split for the special case of splitting row indices into groups.

**Usage**

```
split_indices(group, n = 0L)
```

**Arguments**

group	Integer indices
n	The largest integer (may not appear in index). This is hint: if the largest value of group is bigger than n, the output will silently expand.

**Value**

A list of vectors of indices.

**References**

<https://github.com/hadley/plyr/blob/d57f9377eb5d56107ba3136775f2f0f005f33aa3/src/split-numeric.cpp#L20>

**Examples**

```
split_indices(sample(10, 100, rep = TRUE))
split_indices(sample(10, 100, rep = TRUE), 10)
```

---

subsampling	<i>Subsampling function</i>
-------------	-----------------------------

---

**Description**

This function subsamples a matrix using either random sampling or meta cells method.

**Usage**

```
subsampling(
  matrix,
  subsampling_method = c("sample", "meta_cells", "pseudobulk"),
  subsampling_ratio = 1,
  seed = 1,
  verbose = TRUE,
  ...
)
```

**Arguments**

matrix	The input matrix to be subsampled.
subsampling_method	The method to use for subsampling. Options are "sample", "pseudobulk" or "meta_cells".
subsampling_ratio	The percent of all samples used for <code>fit_srm</code> , default is 1.
seed	The random seed for cross-validation, default is 1.
verbose	Logical value, default is TRUE, whether to print progress messages.
...	Parameters for other methods.

**Value**

The subsampled matrix.

**Examples**

```
data("example_matrix")
data("example_ground_truth")
subsample_matrix <- subsampling(
  example_matrix,
  subsampling_ratio = 0.5
)
subsample_matrix_2 <- subsampling(
  example_matrix,
  subsampling_method = "meta_cells",
  subsampling_ratio = 0.5,
  fast_pca = FALSE
)
subsample_matrix_3 <- subsampling(
  example_matrix,
  subsampling_method = "pseudobulk",
  subsampling_ratio = 0.5
)

calculate_metrics(
  inferCSN(example_matrix),
  example_ground_truth,
  return_plot = TRUE
)
calculate_metrics(
  inferCSN(subsample_matrix),
  example_ground_truth,
  return_plot = TRUE
)
calculate_metrics(
  inferCSN(subsample_matrix_2),
  example_ground_truth,
  return_plot = TRUE
)
calculate_metrics(
  inferCSN(subsample_matrix_3),
  example_ground_truth,
  return_plot = TRUE
)
```

---

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")  
)
```

---

weight\_sift

*Weight sift*

---

**Description**

Remove edges with smaller weights in the reverse direction.

**Usage**

```
weight_sift(table)
```

**Arguments**

```
table          A data frame with three columns: "regulator", "target", and "weight".
```

**Examples**

```
data("example_matrix")  
network_table <- inferCSN(example_matrix)  
weight_sift(network_table) |> head()
```

---

%ss%

*Value selection operator*

---

**Description**

This operator returns the left side if it's not NULL, otherwise it returns the right side.

**Usage**

a %ss% b

**Arguments**

a           The left side value to check

b           The right side value to use if a is NULL

**Examples**

```
NULL %ss% 10
5 %ss% 10
```

# Index

`%ss%`, 55

`as_matrix`, 3

`calculate_accuracy`, 5

`calculate_auc`, 5

`calculate_auprc`, 6

`calculate_auroc`, 7

`calculate_f1`, 8

`calculate_gene_rank`, 9

`calculate_ji`, 9

`calculate_metrics`, 10

`calculate_precision`, 11

`calculate_recall`, 12

`calculate_si`, 13

`check_sparsity`, 14

`coef.srm`, 14

`coef.srm_cv (coef.srm)`, 14

`cor`, 48

`example_ground_truth`, 15

`example_matrix`, 15

`example_meta_data`, 15

`filter_sort_matrix`, 15

`fit_srm`, 16, 19, 52

`foreach`, 19, 27, 29

`inferCSN`, 17

`inferCSN, data.frame-method (inferCSN)`, 17

`inferCSN, matrix-method (inferCSN)`, 17

`inferCSN, sparseMatrix-method (inferCSN)`, 17

`inferCSN-package`, 3

`irlba`, 24

`log_message`, 21

`matrix_to_table`, 22

`message`, 21

`meta_cells`, 23

`network_format`, 25

`network_sift`, 26

`normalization`, 28

`parallelize_fun`, 29

`pearson_correlation`, 30

`plot_coefficient`, 30, 31

`plot_coefficients`, 31

`plot_contrast_networks`, 32

`plot_dynamic_networks`, 32

`plot_edges_comparison`, 34

`plot_embedding`, 35

`plot_histogram`, 36

`plot_network_heatmap`, 38

`plot_scatter`, 40

`plot_static_networks`, 42

`predict.srm`, 43

`predict.srm_cv (predict.srm)`, 43

`print.srm`, 44, 44

`print.srm_cv (print.srm)`, 44

`r_square`, 45

`Rtsne`, 36

`setThreadOptions`, 3

`simulate_sparse_matrix`, 45

`single_network`, 46

`sparse_cor`, 47

`sparse_regression`, 14, 44, 49

`split_indices`, 51

`subsampling`, 52

`suppressMessages`, 21

`table_to_matrix`, 53

`umap`, 36

`weight_sift`, 54