Package 'scGraphVerse'

May 1, 2026

Title scGraphVerse: A Gene Network Analysis Package

Version 1.3.0

Description A package for inferring, comparing, and visualizing gene networks from single-cell RNA sequencing data. It integrates multiple methods (GENIE3, GRNBoost2, ZILGM, PCzinb, and JRF) for robust network inference, supports consensus building across methods or datasets, and provides tools for evaluating regulatory structure and community similarity. GRNBoost2 requires Python package 'arboreto' which can be installed using init_py(install_missing = TRUE). This package includes adapted functions from ZILGM (Park et al., 2021), JRF (Petralia et al., 2015), and learn2count (Nguyen et al. 2023) packages with proper attribution under GPL-2 license.

License GPL-3 + file LICENSE

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

biocViews GeneRegulation, NetworkInference, SingleCell, RNASeq, Visualization, Software, GraphAndNetwork, GeneSetEnrichment, NetworkEnrichment, Pathways, Sequencing, Reactome, Network, KEGG

URL https://ngsFC.github.io/scGraphVerse

BugReports https://github.com/ngsFC/scGraphVerse/issues

Depends R (>= 4.5.0)

Imports BiocBaseUtils, BiocParallel (>= 1.30.0), doParallel, doRNG, GENIE3, Matrix, MultiAssayExperiment, SingleCellExperiment, SummarizedExperiment, distributions3, dplyr, grDevices, graphics, httr, igraph, jsonlite, methods, parallel, reticulate, tidyr, glmnet, MASS, utils, stats, S4Vectors, graph, mpath

Suggests AnnotationDbi, BiocStyle, clusterProfiler, DOSE, enrichplot, fmsb, ggplot2, ggraph, gridExtra, INetTool, org.Hs.eg.db, org.Mm.eg.db, patchwork, pROC, RColorBrewer, ReactomePA, rentrez, robin, scales, Seurat, STRINGdb, testthat (>= 3.0.0), knitr, rmarkdown, tidyverse, magick, celldex, SingleR, TENxPBMCData, scater, GenomeInfoDb, GenomicRanges,

VignetteBuilder knitr

LazyData false

SystemRequirements Python (>= 3.6) and arboreto Python package for GRNBoost2 method. Use init_py(install_missing = TRUE) for automated installation.

git_url https://git.bioconductor.org/packages/scGraphVerse

git_branch devel

git_last_commit 9153b37

git_last_commit_date 2026-04-28

Repository Bioconductor 3.24

Date/Publication 2026-04-30

Author Francesco Cecere [aut, cre] (ORCID: <https://orcid.org/0000-0002-0329-0870>), Annamaria Carissimo [aut], Daniela De Canditiis [aut], Claudia Angelini [aut, fnd]

Maintainer Francesco Cecere <francesco.cecerengs@gmail.com>

Contents

build_network_se . . . 3
classify_edges . . . 4
community_path . . . 5
community_similarity . . . 7
compare_consensus . . . 9
compute_community_metrics . . . 10
compute_topology_metrics . . . 11
create_consensus . . . 13
create_mae . . . 15
cutoff_adjacency . . . 16
download_Atlas . . . 17
earlyj . . . 18
edge_mining . . . 19
generate_adjacency . . . 20
infer_networks . . . 21
init_py . . . 24
nb.loglik . . . 25
nb.loglik.dispersion . . . 26
nb.loglik.regression . . . 26
nb.loglik.regression.gradient . . . 27
nb.OptimizeDispersion . . . 28
nb.regression.parseModel . . . 28
PCzinb . . . 29
plotg . . . 30
plotROC . . . 31
plot_community_comparison . . . 33
plot_network_comparison . . . 34
pscores . . . 36
selgene . . . 37
stringdb_adjacency . . . 38
symmetrize . . . 40

toy_adj_matrix 41
toy_counts 41
zinb.regression.parseModel 42
zinb0.noT 43
zinb1.noT 43
zinbOptimizeDispersion 44
zinb_simdata 44

Index **47**

`build_network_se` *Create a SummarizedExperiment for Network Storage*

Description

Constructs a SummarizedExperiment container for multiple gene regulatory network adjacency matrices with shared gene space (p×p matrices).

Usage

```
build_network_se(
  networks,
  networkData = NULL,
  geneData = NULL,
  metadata = list()
)
```

Arguments

`networks`	A list of adjacency matrices (all must have same dimensions)
`networkData`	A DataFrame with metadata for each network
`geneData`	Optional. A DataFrame with gene-level annotations
`metadata`	Optional. List of global metadata

Value

A SummarizedExperiment object where each assay is a p×p network

Examples

```
# Example 1: Building SE from a list of adjacency matrices
data("toy_adj_matrix")

# Create a list of network matrices
net_list <- list(
    network1 = toy_adj_matrix,
    network2 = toy_adj_matrix
)

# Build SummarizedExperiment
network_se <- build_network_se(net_list)
network_se
```

```
# Example 2: Using with inferred networks
data("toy_counts")

networks <- infer_networks(
    count_matrices_list = toy_counts,
    method = "GENIE3",
    nCores = 1
)

# generate_adjacency() internally uses build_network_se()
wadj_se <- generate_adjacency(networks)
wadj_se
```

`classify_edges` *Classify Edges as TP, FP, or FN*

Description

Compares a consensus network to a reference network and classifies edges as True Positives (TP), False Positives (FP), or False Negatives (FN).

Usage

```
classify_edges(consensus_matrix, reference_matrix = NULL, use_stringdb = TRUE)
```

Arguments

`consensus_matrix`	A SummarizedExperiment object representing the consensus network.
`reference_matrix`	A SummarizedExperiment object representing the reference (ground truth) network. If `NULL`, STRINGdb is used to generate a high-confidence physical network.
`use_stringdb`	Logical. If TRUE and `reference_matrix` is NULL, queries STRINGdb for reference network. Default: TRUE.

Details

If `reference_matrix` is NULL and `use_stringdb = TRUE`, this function queries STRINGdb to generate a human high-confidence (score > 900) physical interaction network.

Value

A list containing:

- `tp_edges`: Character vector of True Positive edges
- `fp_edges`: Character vector of False Positive edges
- `fn_edges`: Character vector of False Negative edges
- `consensus_graph`: igraph object of consensus network
- `reference_graph`: igraph object of reference network
- `edge_colors`: Color vector for TP (red) and FN (blue) edges
- `use_stringdb`: Logical indicating if STRINGdb was used

Examples

```
data(toy_counts)
data(toy_adj_matrix)


# Infer networks (toy_counts is already a MultiAssayExperiment)
networks <- infer_networks(
    count_matrices_list = toy_counts,
    method = "GENIE3",
    nCores = 1
)

# Generate and symmetrize adjacency matrices (returns SummarizedExperiment)
wadj_se <- generate_adjacency(networks)
swadj_se <- symmetrize(wadj_se, weight_function = "mean")

# Apply cutoff (returns SummarizedExperiment)
binary_se <- cutoff_adjacency(
    count_matrices = toy_counts,
    weighted_adjm_list = swadj_se,
    n = 1,
    method = "GENIE3",
    quantile_threshold = 0.95,
    nCores = 1
)

# Create consensus (returns SummarizedExperiment)
consensus <- create_consensus(binary_se, method = "union")

# Wrap reference matrix in SummarizedExperiment
ref_se <- build_network_se(list(reference = toy_adj_matrix))

# classify_edges expects SummarizedExperiment objects
edge_class <- classify_edges(consensus, ref_se)
```

`community_path` *Community Detection and Pathway Enrichment Analysis*

Description

Detects gene communities within an adjacency network using one or two community detection methods, and performs pathway enrichment for each detected community.

Usage

```
community_path(
  adj_matrix,
  methods = "louvain",
  pathway_db = "KEGG",
  organism = c("human", "mouse"),
  genes_path = 5,
  plot = TRUE,
  verbose = TRUE,
```

```
  method_params = list(),
  comparison_params = list(),
  BPPARAM = BiocParallel::bpparam()
)
```

Arguments

adj_matrix	A square adjacency matrix or a SummarizedExperiment object containing a single adjacency matrix. Row and column names must correspond to gene symbols.
methods	A character vector of one or two community detection methods supported by **robin**. If two are given, performance is compared and the best is selected. Default: "louvain".
pathway_db	Character string specifying the pathway database to use: "KEGG" or "Reactome". Default: "KEGG".
organism	Character string specifying the organism: "human" or "mouse". Default: "human".
genes_path	Integer. Minimum number of genes per community to run enrichment analysis. Default: 5.
plot	Logical. If TRUE, generates a plot of detected communities. Default: TRUE.
verbose	Logical. If TRUE, shows progress messages. Default: TRUE.
method_params	List of parameters for community detection methods. Common parameters include: • resolution: Resolution parameter for Louvain/Leiden (default: 1) • steps: Number of steps for Walktrap (default: 4) • spins: Number of spins for Spinglass (default: 25) • nb.trials: Number of trials for Infomap (default: 10)
comparison_params	List of parameters for robin comparison: • measure: Stability measure ("vi", "nmi", "split.join", "adjusted.rand"). Default: "vi" • type: Robin construction type ("dependent", "independent"). Default: "independent" • rewire.w.type: Rewiring strategy for weighted graphs. Default: "Rewire"
BPPARAM	BiocParallel backend for parallel processing. Default: BiocParallel::bpparam().

Details

If two methods are provided, the function uses robinCompare and selects the method with higher AUC. Pathway enrichment is done via **clusterProfiler** (KEGG) or via **ReactomePA** (Reactome). Communities smaller than genes_path are excluded.

Value

A list with elements:

- communities: List with best_method and a named vector of community membership per gene.
- pathways: List of enrichment results per community (only for communities meeting size threshold).
- graph: The **igraph** object with community annotations.

Examples

```
data(toy_counts)


# Infer networks (toy_counts is already a MultiAssayExperiment)
networks <- infer_networks(
    count_matrices_list = toy_counts,
    method = "GENIE3",
    nCores = 1
)
head(networks[[1]])

# Generate and symmetrize adjacency matrices (returns SummarizedExperiment)
wadj_se <- generate_adjacency(networks)
swadj_se <- symmetrize(wadj_se, weight_function = "mean")

# Apply cutoff (returns SummarizedExperiment)
binary_se <- cutoff_adjacency(
    count_matrices = toy_counts,
    weighted_adjm_list = swadj_se,
    n = 1,
    method = "GENIE3",
    quantile_threshold = 0.95,
    nCores = 1,
    debug = TRUE
)

# Create consensus (returns SummarizedExperiment)
consensus <- create_consensus(binary_se, method = "union")

# community_path now accepts SummarizedExperiment objects directly
comm_cons <- community_path(consensus)
```

---

community_similarity *Compare Community Assignments and Topological Properties*

Description

Convenience wrapper that computes community assignment metrics, topological properties, and optionally visualizes the comparison. For more control, use the individual functions: compute_community_metrics, compute_topology_metrics, and plot_community_comparison.

Usage

```
community_similarity(control_output, predicted_list, plot = TRUE)
```

Arguments

control_output	A list output from community_path() representing the ground truth network. Must contain a graph (igraph object) and communities$membership.
predicted_list	A list of lists, each output from community_path() representing predicted networks to compare.

`plot` Logical. If TRUE, displays plots immediately. If FALSE, no plots are displayed. Default: TRUE.

Details

This function requires the **igraph** package. If `plot = TRUE`, the **fmsb** package is also required. Community similarity is measured using variation of information (VI), normalized mutual information (NMI), and adjusted Rand index (ARI).

Value

A list containing:

- `community_metrics`: A data frame with VI, NMI, and ARI scores for each prediction.
- `topology_measures`: A data frame with raw topological metrics for each prediction.
- `control_topology`: A list of raw topological metrics for the ground truth network.

See Also

`compute_community_metrics`, `compute_topology_metrics`, `plot_community_comparison`

Examples

```
data(toy_counts)
data(toy_adj_matrix)


# Infer networks (toy_counts is already a MultiAssayExperiment)
networks <- infer_networks(
    count_matrices_list = toy_counts,
    method = "GENIE3",
    nCores = 1
)
head(networks[[1]])

# Generate adjacency matrices
wadj_se <- generate_adjacency(networks)
swadj_se <- symmetrize(wadj_se, weight_function = "mean")

# Apply cutoff
binary_se <- cutoff_adjacency(
    count_matrices = toy_counts,
    weighted_adjm_list = swadj_se,
    n = 1,
    method = "GENIE3",
    quantile_threshold = 0.95,
    nCores = 1,
    debug = TRUE
)
head(binary_se[[1]])

consensus <- create_consensus(binary_se, method = "union")
comm_cons <- community_path(consensus)
comm_truth <- community_path(toy_adj_matrix)

sim_score <- community_similarity(comm_truth, list(comm_cons))
```

compare_consensus *Compare Consensus and Reference Graphs or STRINGdb Networks*

Description

Convenience wrapper that classifies edges and visualizes the comparison between consensus and reference networks. For more control, use the individual functions: `classify_edges` and `plot_network_comparison`.

Usage

```
compare_consensus(
  consensus_matrix,
  reference_matrix = NULL,
  false_plot = FALSE
)
```

Arguments

`consensus_matrix`	A SummarizedExperiment object representing the consensus network.
`reference_matrix`	Optional. A SummarizedExperiment obj representing the reference network. If `NULL`, STRINGdb is queried.
`false_plot`	Logical. If `TRUE`, displays False Positives plot. Default is `FALSE`.

Details

If no `reference_matrix` is provided, STRINGdb is queried to generate a high-confidence physical interaction network.

Value

A `ggplot` object visualizing the comparison.

Note

Requires **ggraph** and **ggplot2**. If `reference_matrix` is NULL, also requires **STRINGdb**. If `false_plot = TRUE`, requires **patchwork**.

See Also

`classify_edges`, `plot_network_comparison`

Examples

```
data(toy_counts)
data(toy_adj_matrix)


# Infer networks (toy_counts is already a MultiAssayExperiment)
networks <- infer_networks(
    count_matrices_list = toy_counts,
```

```
    method = "GENIE3",
    nCores = 1
)
head(networks[[1]])

# Generate adjacency matrices
wadj_se <- generate_adjacency(networks)
swadj_se <- symmetrize(wadj_se, weight_function = "mean")

# Apply cutoff
binary_se <- cutoff_adjacency(
    count_matrices = toy_counts,
    weighted_adjm_list = swadj_se,
    n = 1,
    method = "GENIE3",
    quantile_threshold = 0.95,
    nCores = 1,
    debug = TRUE
)
head(binary_se[[1]])

consensus <- create_consensus(binary_se, method = "union")

# Wrap reference matrix in SummarizedExperiment
ref_se <- build_network_se(list(reference = toy_adj_matrix))

# Compare consensus to reference
compare_consensus(
    consensus,
    reference_matrix = ref_se,
    false_plot = FALSE
)
```

`compute_community_metrics` *Compute Community Assignment Similarity Metrics*

Description

Calculates community assignment similarity between a reference community structure and one or more predicted structures using variation of information (VI), normalized mutual information (NMI), and adjusted Rand index (ARI).

Usage

```
compute_community_metrics(control_output, predicted_list)
```

Arguments

`control_output`	A list output from `community_path()` representing the ground truth network. Must contain `communities$membership`.
`predicted_list`	A list of lists, each output from `community_path()` representing predicted networks to compare.

Details

This function requires the **igraph** package. Lower VI values indicate better similarity (VI = 0 is perfect match). Higher NMI and ARI values indicate better similarity (both range 0-1).

Value

A data frame with columns VI, NMI, and ARI for each prediction. Row names indicate which prediction (e.g., "Predicted_1").

Examples

```
data(toy_counts)
data(toy_adj_matrix)


# Infer networks (toy_counts is already a MultiAssayExperiment)
networks <- infer_networks(
    count_matrices_list = toy_counts,
    method = "GENIE3",
    nCores = 1
)

# Generate adjacency matrices
wadj_se <- generate_adjacency(networks)
swadj_se <- symmetrize(wadj_se, weight_function = "mean")

# Apply cutoff
binary_se <- cutoff_adjacency(
    count_matrices = toy_counts,
    weighted_adjm_list = swadj_se,
    n = 1,
    method = "GENIE3",
    quantile_threshold = 0.95,
    nCores = 1
)

consensus <- create_consensus(binary_se, method = "union")
comm_cons <- community_path(consensus)
comm_truth <- community_path(toy_adj_matrix)

metrics <- compute_community_metrics(comm_truth, list(comm_cons))
```

---

compute_topology_metrics

Compute Network Topological Properties

Description

Calculates topological properties (modularity, number of communities, density, transitivity) for one or more networks.

Usage

```
compute_topology_metrics(control_output, predicted_list)
```

Arguments

control_output	A list output from community_path() representing the ground truth network. Must contain a graph (igraph object) and communities$membership.
predicted_list	A list of lists, each output from community_path() representing predicted networks.

Details

This function requires the **igraph** package. Topological metrics include:

- Modularity: Quality of community division
- Communities: Number of detected communities
- Density: Proportion of actual vs. possible edges
- Transitivity: Clustering coefficient

Value

A list containing:

- topology_measures: A data frame with Modularity,Communities, Density, and Transitivity for each prediction.
- control_topology: A numeric vector of the same metrics for the control network.

Examples

```
data(toy_counts)
data(toy_adj_matrix)


# Infer networks (toy_counts is already a MultiAssayExperiment)
networks <- infer_networks(
    count_matrices_list = toy_counts,
    method = "GENIE3",
    nCores = 1
)

# Generate adjacency matrices
wadj_se <- generate_adjacency(networks)
swadj_se <- symmetrize(wadj_se, weight_function = "mean")

# Apply cutoff
binary_se <- cutoff_adjacency(
    count_matrices = toy_counts,
    weighted_adjm_list = swadj_se,
    n = 1,
    method = "GENIE3",
    quantile_threshold = 0.95,
    nCores = 1
)

consensus <- create_consensus(binary_se, method = "union")
comm_cons <- community_path(consensus)
comm_truth <- community_path(toy_adj_matrix)

topo <- compute_topology_metrics(comm_truth, list(comm_cons))
```

`create_consensus` *Create a Consensus Adjacency Matrix from Multiple Networks*

Description

Builds a consensus adjacency matrix from networks stored in a SummarizedExperiment using one of three methods: `"vote"`, `"union"`, or `"INet"`.

Usage

```
create_consensus(
  adj_matrix_list,
  method = c("vote", "union", "INet"),
  weighted_list = NULL,
  theta = 0.04,
  threshold = 0.5,
  ncores = 1,
  tolerance = 0.1,
  nitermax = 50,
  verbose = FALSE
)
```

Arguments

`adj_matrix_list`	A SummarizedExperiment object containing binary adjacency matrices (square, 0/1) with identical dimensions and matching row/column names, or a list of such matrices.
`method`	Character string specifying the consensus strategy. One of: • `"vote"` (default): An edge is included if supported by at least `threshold` fraction of matrices. • `"union"`: An edge is included if present in any matrix. • `"INet"`: Combines normalized weighted matrices using `consensusNet`.
`weighted_list`	A SummarizedExperiment object containing weighted adjacency matrices (required if `method = "INet"`), or a list of such matrices.
`theta`	Numeric. Tuning parameter passed to `consensusNet` (default: `0.04`).
`threshold`	Numeric between 0 and 1. Threshold for "vote" and "INet" methods. Default is `0.5`.
`ncores`	Integer. Number of CPU cores to use when `method = "INet"`. Default is `1`.
`tolerance`	Numeric. Tolerance for differences between similar graphs in INet method. Default is `0.1`.
`nitermax`	Integer. Maximum number of iterations for INet algorithm. Default is `50`.
`verbose`	Logical. If TRUE, display verbose output for INet method. Default is `FALSE`.

Details

Consensus construction depends on the selected method:

vote Counts the presence of each edge across all matrices and includes edges supported by at least `threshold` × `N` matrices.

union Includes any edge that appears in any matrix.

INet Multiplies binary matrices by corresponding weighted matrices, normalizes the results, and applies `consensusNet` to generate a consensus network.

For "INet", both binary and weighted adjacency matrices must be provided with matching dimensions.

Value

A SummarizedExperiment object with a single assay containing the consensus adjacency matrix (binary or weighted, depending on the method). Metadata includes consensus method and parameters.

Examples

```
data(toy_counts)


# Infer networks (toy_counts is already a MultiAssayExperiment)
networks <- infer_networks(
    count_matrices_list = toy_counts,
    method = "GENIE3",
    nCores = 1
)
head(networks[[1]])

# Generate adjacency matrices
wadj_se <- generate_adjacency(networks)
swadj_se <- symmetrize(wadj_se, weight_function = "mean")

# Apply cutoff
binary_se <- cutoff_adjacency(
    count_matrices = toy_counts,
    weighted_adjm_list = swadj_se,
    n = 1,
    method = "GENIE3",
    quantile_threshold = 0.95,
    nCores = 1,
    debug = TRUE
)
head(binary_se[[1]])

consensus <- create_consensus(binary_se, method = "union")
head(consensus)
```

`create_mae` *Create MultiAssayExperiment from Multiple Single-Cell Datasets*

Description

Converts a list of count matrices, Seurat objects, or SingleCellExperiment objects into a MultiAssayExperiment for integrated network inference.

Usage

```
create_mae(datasets, colData = NULL, ...)
```

Arguments

`datasets`	A named list of datasets. Each element can be: • A matrix (genes × cells) • A Seurat object • A SingleCellExperiment object
`colData`	Optional. A DataFrame with metadata for each experiment. If NULL, automatically generated from list names.
`...`	Additional arguments (currently unused)

Value

A MultiAssayExperiment object with:

- experiments: List of SingleCellExperiment objects
- colData: Metadata for each experiment/condition

Examples

```
# Load the example MAE
data("toy_counts")

# Extract the list of SingleCellExperiment objects
sce_list <- MultiAssayExperiment::experiments(toy_counts)
sce_list <- as.list(sce_list)

# Create a new MAE from the SCE list
mae <- create_mae(sce_list)
mae
```

cutoff_adjacency *Threshold Adjacency Matrices Based on Shuffled Network Quantiles*

Description

Applies a cutoff to weighted adjacency matrices using a percentile estimated from shuffled versions of the original expression matrices. Supports inference methods `"GENIE3"`, `"GRNBoost2"`, and `"JRF"`.

Usage

```
cutoff_adjacency(
  count_matrices,
  weighted_adjm_list,
  n,
  method = c("GENIE3", "GRNBoost2", "JRF"),
  quantile_threshold = 0.99,
  weight_function = "mean",
  nCores = 1,
  grnboost_modules = NULL,
  debug = FALSE
)
```

Arguments

`count_matrices`	A MultiAssayExperiment object containing expression data from multiple experiments or conditions.
`weighted_adjm_list`	A SummarizedExperiment object containing weighted adjacency matrices (one per experiment) to threshold.
`n`	Integer. Number of shuffled replicates generated per original expression matrix.
`method`	Character string. One of `"GENIE3"`, `"GRNBoost2"`, or `"JRF"`.
`quantile_threshold`	Numeric. The quantile used to define the cutoff. Default is `0.99`.
`weight_function`	Character string or function used to symmetrize adjacency matrices (`"mean"`, `"max"`, etc.).
`nCores`	Integer. Number of CPU cores to use for parallelization. Default is the number of workers in the current **BiocParallel** backend. Note: JRF uses C implementation and does not use this parameter.
`grnboost_modules`	Python modules needed for `GRNBoost2` if using reticulate.
`debug`	Logical. If `TRUE`, prints detailed progress messages. Default is `FALSE`.

Details

For each input expression matrix, `n` shuffled versions are generated by randomly permuting each gene's expression across cells. Network inference is performed on the shuffled matrices, and a

cutoff is determined as the specified quantile (`quantile_threshold`) of the resulting edge weights. The original weighted adjacency matrices are then thresholded using these estimated cutoffs.

Parallelization is handled via **BiocParallel**.

The methods are based on:

- **GENIE3**: Random Forest-based inference (Huynh-Thu et al., 2010).
- **GRNBoost2**: Gradient boosting trees using arboreto (Moerman et al., 2019).
- **JRF**: Joint Random Forests across multiple conditions (Petralia et al., 2015).

Value

A SummarizedExperiment object where each assay is a binary (thresholded) adjacency matrix corresponding to an input weighted matrix. Metadata includes cutoff values and method parameters.

Examples

```
data(toy_counts)


# Infer networks (toy_counts is already a MultiAssayExperiment)
networks <- infer_networks(
    count_matrices_list = toy_counts,
    method = "GENIE3",
    nCores = 1
)
head(networks[[1]])

# Generate adjacency matrices
wadj_se <- generate_adjacency(networks)
swadj_se <- symmetrize(wadj_se, weight_function = "mean")

# Apply cutoff
binary_se <- cutoff_adjacency(
    count_matrices = toy_counts,
    weighted_adjm_list = swadj_se,
    n = 1,
    method = "GENIE3",
    quantile_threshold = 0.95,
    nCores = 1,
    debug = TRUE
)
head(binary_se[[1]])
```

---

download_Atlas *Download and Load an RDS File from a URL*

Description

Downloads an RDS file from a specified URL and reads its contents into R. We used it for https://www.singlecellatlas.org

Usage

```
download_Atlas(file_url)
```

Arguments

`file_url`	Character; URL of the RDS file to download.

Details

This function uses **httr** to perform the download. The RDS file is read directly from a raw connection without saving to disk. An internet connection is required.

If the download fails (e.g., invalid URL, server error), an informative error message is returned.

Value

An R object loaded from the downloaded RDS file.

Examples

```
url <- "https://zenodo.org/records/15511027/files/sce_obj.rds?download=1"
atlas_data <- download_Atlas(url)
```

`earlyj` *Modify Cell Names and Combine Datasets*

Description

Extracts expression data from a MultiAssayExperiment object, modifies cell identifiers by appending a unique experiment index (e.g., "-m1", "-m2", etc.), and merges the datasets into a single object.

Usage

```
earlyj(input_list, rowg = TRUE)
```

Arguments

`input_list`	A MultiAssayExperiment object containing expression data from multiple experiments or conditions.
`rowg`	Logical. If `TRUE` (default), genes are assumed to be rows and cells columns. If `FALSE`, matrices are transposed before renaming and combining.

Details

For matrices, this function optionally transposes the input before combining. For `Seurat` and `SingleCellExperiment` objects, only features (genes) common across all input datasets are retained before merging. The cell names are suffixed with "-m1", "-m2", etc., according to their original list position. The result is returned as a MultiAssayExperiment with a single merged experiment.

Value

A MultiAssayExperiment object containing a single merged experiment with modified (unique) cell names.

Examples

```
data(toy_counts)

merged_mae <- earlyj(toy_counts)
merged_mae
```

---

edge_mining *Edge Mining of Gene Interactions Using PubMed*

Description

Query PubMed for literature evidence supporting predicted gene–gene interactions.

Usage

```
edge_mining(
  predicted_list,
  ground_truth,
  delay = 1,
  query_field = "Title/Abstract",
  query_edge_types = c("TP", "FP", "FN"),
  max_retries = 10,
  BPPARAM = BiocParallel::bpparam()
)
```

Arguments

predicted_list	A list of predicted adjacency matrices (row and column names are gene symbols), or a SummarizedExperiment object containing adjacency matrices.
ground_truth	A 0/1 adjacency matrix with row and column names.
delay	Numeric. Seconds to wait between consecutive queries (default = 1).
query_field	Character. PubMed search field. Options: "Title/Abstract" (default), "Title", "Abstract".
query_edge_types	Character vector. Edge types to query: c("TP", "FP", "FN") (default all).
max_retries	Integer. Max retries for PubMed queries (default = 10).
BPPARAM	A BiocParallel parameter object. Default = bpparam().

Details

This function compares predicted adjacency matrices against a ground truth matrix, identifies edge types (TP, FP, FN), and queries PubMed for each gene pair. Returns counts of hits, PMIDs, and query status.

Value

A named list of data.frames. Each data.frame has columns:

gene1 First gene in interaction

gene2 Second gene

edge_type One of "TP", "FP", or "FN"

pubmed_hits Number of PubMed hits

PMIDs Comma-separated PubMed IDs or NA

query_status One of "hits_found", "no_hits", or "error"

Examples

```
data(toy_counts)
data(toy_adj_matrix)


# Infer networks (toy_counts is already a MultiAssayExperiment)
networks <- infer_networks(
    count_matrices_list = toy_counts,
    method = "GENIE3",
    nCores = 1
)
head(networks[[1]])

# Generate adjacency matrices
wadj_se <- generate_adjacency(networks)
swadj_se <- symmetrize(wadj_se, weight_function = "mean")

# Apply cutoff
binary_se <- cutoff_adjacency(
    count_matrices = toy_counts,
    weighted_adjm_list = swadj_se,
    n = 1,
    method = "GENIE3",
    quantile_threshold = 0.95,
    nCores = 1,
    debug = TRUE
)
head(binary_se[[1]])

consensus <- create_consensus(binary_se, method = "union")
head(consensus)
em <- edge_mining(consensus, toy_adj_matrix, query_edge_types = "TP")
```

---

generate_adjacency *Generate Adjacency Matrices from Gene Interaction Tables*

Description

Constructs adjacency matrices from a list of data frames (network edge lists) and returns them in a SummarizedExperiment object.

Usage

```
generate_adjacency(df_list, nCores = 1)
```

Arguments

df_list	A list of data frames. Each data frame must have three columns: **Gene1** Character. First gene in the interaction. **Gene2** Character. Second gene in the interaction. **Weight** Numeric. Weight or strength of the interaction from Gene1 to Gene2.
nCores	Integer. Number of CPU cores to use for parallel processing. Defaults to the number of available workers from the current **BiocParallel** backend.

Details

The function first identifies all unique genes across all data frames to define the matrix dimensions. For each interaction table, it places the corresponding weights at the appropriate gene-pair positions. Parallelization is handled by **BiocParallel** for improved performance on multiple datasets.

Missing weights (NA) are ignored during construction. Only gene pairs matching the global gene list are inserted.

Value

A SummarizedExperiment object where each assay is a square numeric adjacency matrix (p×p genes). Diagonal entries are set to zero (no self-interactions).

Examples

```
data("toy_counts")

# Infer networks (toy_counts is already a MultiAssayExperiment)
networks <- infer_networks(
    count_matrices_list = toy_counts,
    method = "GENIE3",
    nCores = 1
)
head(networks[[1]])

# Generate adjacency matrices
wadj_se <- generate_adjacency(networks) # returns SummarizedExperiment
head(wadj_se[[1]])
```

---

infer_networks *Infer Gene Regulatory Networks from Expression Matrices*

Description

Infers weighted gene regulatory networks (GRNs) from one or more expression matrices using different inference methods: "GENIE3", "GRNBoost2", "ZILGM", "JRF", or "PCzinb".

Usage

```
infer_networks(
  count_matrices_list,
  method = c("GENIE3", "GRNBoost2", "ZILGM", "JRF", "PCzinb"),
  adjm = NULL,
  nCores = 1,
  grnboost_modules = NULL,
  genie3_params = list(),
  grnboost2_params = list(),
  zilgm_params = list(),
  jrf_params = list(),
  pczinb_params = list(),
  verbose = FALSE
)
```

Arguments

`count_matrices_list`
: A MultiAssayExperiment object containing expression data from multiple experiments or conditions.

`method`
: Character string. Inference method to use. One of: `"GENIE3"`, `"GRNBoost2"`, `"ZILGM"`, `"JRF"`, or `"PCzinb"`.

`adjm`
: Optional. Reference adjacency matrix for matching dimensions when using `"ZILGM"` or `"PCzinb"`.

`nCores`
: Integer. Number of CPU cores to use for parallelization. Default: 1.

`grnboost_modules`
: Python modules required for `GRNBoost2` (created via **reticulate**).

`genie3_params`
: List of parameters for GENIE3 method:

- `regulators`: Vector of regulator gene names (default: all)
- `targets`: Vector of target gene names (default: all genes)
- `treeMethod`: "RF" or "ET" (default: "RF")
- `K`: Number of candidate regulators (default: "sqrt")
- `nTrees`: Number of trees per ensemble (default: 1000)
- `seed`: Random seed for reproducibility (default: NULL)

`grnboost2_params`
: List of parameters for GRNBoost2 method:

- `tf_names`: Vector of transcription factor names (default:all)
- `gene_names`: Vector of target gene names (default: all)
- `client_or_address`: Dask client or address (default: NULL)
- `seed`: Random seed for reproducibility (default: NULL)

`zilgm_params`
: List of parameters for ZILGM method:

- `lambda`: Regularization parameter (default: 0.1)
- `alpha`: Elastic net mixing parameter (default: 1)
- `max_iter`: Maximum iterations (default: 100)
- `tol`: Convergence tolerance (default: 1e-4)

`jrf_params`
: List of parameters for JRF method:

- `ntree`: Number of trees (default: 1000)

- `mtry`: Number of variables to sample at each split (default: sqrt(p))

`pczinb_params` List of parameters for PCzinb method:

- `gamma`: Regularization parameter (default: 0.1)
- `beta`: Beta parameter (default: 0.1)
- `max_iter`: Maximum iterations (default: 100)
- `tol`: Convergence tolerance (default: 1e-4)

`verbose` Logical. If TRUE, display messages. Default: FALSE.

Details

Each expression matrix is preprocessed automatically depending on its object type (`Seurat`, `SingleCellExperiment`, or plain matrix).

Parallelization behavior:

- **GENIE3**: No external parallelization; internal `nCores` parameter controls computation.
- **ZILGM**: Uses `nCores` parameter for internal parallelization.
- **GRNBoost2** and **PCzinb**: Parallelized across matrices using **BiocParallel**.
- **JRF**: Joint modeling of all matrices together using optimized C implementation.

Methods are based on:

- **GENIE3**: Random Forest-based inference (Huynh-Thu et al., 2010).
- **GRNBoost2**: Gradient boosting trees using arboreto (Moerman et al., 2019).
- **ZILGM**: Zero-Inflated Graphical Models for scRNA-seq (Zhang et al., 2021).
- **JRF**: Joint Random Forests across multiple conditions (Petralia et al., 2015).
- **PCzinb**: Pairwise correlation under ZINB models (Nguyen et al., 2023).

Value

A list of inferred networks:

- For `"GENIE3"`, `"GRNBoost2"`, `"ZILGM"`, and `"PCzinb"`, a list of inferred network objects (edge lists or adjacency matrices).
- For `"JRF"`, a list of data frames with inferred edge lists for each condition or dataset.

Examples

```
data("toy_counts")

# Infer networks (toy_counts is already a MultiAssayExperiment)
networks <- infer_networks(
    count_matrices_list = toy_counts,
    method = "GENIE3",
    nCores = 1
)
head(networks[[1]])
```

`init_py` *Initialize Python Environment for GRNBoost2*

Description

Sets up the Python environment and lazily loads modules required for running GRNBoost2: `arboreto`, `pandas`, and `numpy`. Automatically installs missing Python packages if requested.

Usage

```
init_py(
  python_path = "/usr/bin/python3",
  required = TRUE,
  install_missing = FALSE,
  install_method = "auto",
  verbose = TRUE
)
```

Arguments

`python_path`	Character string. Path to the Python executable, e.g., `"/usr/bin/python3"`. For optimal GRNBoost2 compatibility, Python 3.8.x is strongly recommended.
`required`	Logical. If `TRUE`, errors if Python is not available or path is invalid. Default: `TRUE`.
`install_missing`	Logical. If `TRUE`, automatically installs missing Python packages. Default: `FALSE`.
`install_method`	Character string. Installation method when `install_missing = TRUE`. Options: "auto", "conda", "pip". Default: "auto".
`verbose`	Logical. If `TRUE`, shows installation progress. Default: `TRUE`.

Details

Uses **reticulate** to bind R to the specified Python interpreter and lazily import modules needed for GRNBoost2. If `install_missing = TRUE`, automatically installs the 'arboreto' package using the specified method if not found.

Value

A list with three Python module objects:

- `arboreto`: GRNBoost2 algorithm module.
- `pandas`: Data handling module.
- `numpy`: Numerical operations module.

Examples

```
# Initialize Python environment (handles missing modules gracefully)
tryCatch(
    {
        modules <- init_py(required = FALSE)
    },
    error = function(e) {
        message("Python environment not available: ", e$message)
    }
)
```

`nb.loglik` *Log-likelihood of the negative binomial model Given a vector of counts, this function computes the sum of the log-probabilities of the counts under a negative binomial (NB) model. The NB distribution is parametrized by two parameters: the mean value and the dispersion of the negative binomial distribution*

Description

Log-likelihood of the negative binomial model Given a vector of counts, this function computes the sum of the log-probabilities of the counts under a negative binomial (NB) model. The NB distribution is parametrized by two parameters: the mean value and the dispersion of the negative binomial distribution

Usage

```
nb.loglik(Y, mu, theta)
```

Arguments

`Y`	the vector of counts
`mu`	the vector of mean parameters of the negative binomial
`theta`	the vector of dispersion parameters of the negative binomial, or a single scalar is also possible if the dispersion parameter is constant. Note that theta is sometimes called inverse dispersion parameter (and phi=1/theta is then called the dispersion parameter). We follow the convention that the variance of the NB variable with mean mu and dispersion theta is mu + mu^2/theta.

Value

the log-likelihood of the model.

nb.loglik.dispersion *Log-likelihood of negative binomial model, for a fixed dispersion parameter*

Description

Given a unique dispersion parameter and a set of counts, together with a corresponding set of mean parameters, this function computes the sum of the log-probabilities of the counts under the NB model. The dispersion parameter is provided to the function through zeta = log(theta), where theta is sometimes called the inverse dispersion parameter.

Usage

```
nb.loglik.dispersion(zeta, Y, mu)
```

Arguments

zeta	a vector, the log of the inverse dispersion parameters of the negative binomial model
Y	a vector of counts
mu	a vector of mean parameters of the negative binomial

Value

the log-likelihood of the model.

nb.loglik.regression *log-likelihood of the NB regression model*

Description

This function computes the log-likelihood of a NB regression model given a vector of counts.

Usage

```
nb.loglik.regression(
  alpha,
  Y,
  A.mu = matrix(nrow = length(Y), ncol = 0),
  C.theta = matrix(0, nrow = length(Y), ncol = 1)
)
```

Arguments

alpha	the vectors of parameters a.mu concatenated
Y	the vector of counts
A.mu	matrix of the model (see Details, default=empty)
C.theta	matrix of the model ($log(\theta)$, default=zero)

Details

The regression model is parametrized as follows:

$$log(\mu) = A_\mu * a_\mu$$

$$log(\theta) = C_\theta$$

where μ, θ are respectively the vector of mean parameters of the NB distribution, and the vector of inverse dispersion parameters. The log-likelihood of a vector of parameters $\alpha = a_\mu$

Value

the log-likelihood.

---

nb.loglik.regression.gradient

Gradient of the log-likelihood of the NB regression model

Description

This function computes the gradient of the log-likelihood of a NB regression model given a vector of counts.

Usage

```
nb.loglik.regression.gradient(
  alpha,
  Y,
  A.mu = matrix(nrow = length(Y), ncol = 0),
  C.theta = matrix(0, nrow = length(Y), ncol = 1)
)
```

Arguments

alpha	the vectors of parameters a.mu concatenated
Y	the vector of counts
A.mu	matrix of the model (see Details, default=empty)
C.theta	matrix of the model (see Details, default=zero)

Details

The regression model is described in nb.loglik.regression.

Value

The gradient of the log-likelihood.

See Also

nb.loglik.regression

nb.OptimizeDispersion *(NB) model. The NB distribution is parametrized by two parameters: the mean value and the dispersion of the negative binomial distribution*

Description

(NB) model. The NB distribution is parametrized by two parameters: the mean value and the dispersion of the negative binomial distribution

Usage

```
nb.OptimizeDispersion(mu, Y, n)
```

Arguments

mu	the vector mean of the negative binomial
Y	the vector of counts
n	the length of the vector to return Note that theta is sometimes called inverse dispersion parameter (and phi=1/theta is then called the dispersion parameter). We follow the convention that the variance of the NB variable with mean mu and dispersion theta is mu + mu^2/theta.

Value

A vector of length n with the optimized dispersion parameter values.

nb.regression.parseModel *Parse ZINB regression model*

Description

Given the parameters of a NB regression model, this function parses the model and computes the vector of log(mu), and the dimensions of the different components of the vector of parameters. See nb.loglik.regression for details of the NB regression model and its parameters.

Usage

```
nb.regression.parseModel(alpha, A.mu)
```

Arguments

alpha	the vectors of parameters c(a.mu) concatenated
A.mu	matrix of the model (default=empty)

Value

A list with slot logMu,

See Also

nb.loglik.regression

PCzinb *Structure learning for count data using PC algorithms*

Description

This function performs structure learning for count data using various PC algorithms adapted for different distributional assumptions including Poisson, Negative Binomial, and Zero-Inflated Negative Binomial models.

Usage

```
PCzinb(
  x,
  method = c("poi", "nb", "zinb0", "zinb1"),
  alpha = NULL,
  maxcard = 2,
  extend = TRUE,
  nCores = 1,
  whichAssay = "processed",
  ...
)
```

Arguments

x	A matrix of count data (n × p), SummarizedExperiment, or SingleCellExperiment object. For matrix input, rows are samples and columns are genes.
method	The algorithm used to estimate the graph: poi (Poisson), nb (Negative Binomial), zinb0 (Zero-Inflated NB with structure only in mu), or zinb1 (Zero-Inflated NB with structure in both mu and pi).
alpha	The significance level of the tests. Default: 2 * pnorm(nrow(x)^0.2, lower.tail = FALSE).
maxcard	The upper bound of the cardinality of the conditional sets K. Default: 2.
extend	If TRUE, considers the union of the tests; if FALSE, considers the intersection. Default: TRUE.
nCores	Number of cores for parallel processing. Default: 1.
whichAssay	The assay to use as input (for SummarizedExperiment or SingleCellExperiment objects). Default: "processed".
...	Additional arguments (currently unused).

Details

PCzinb performs structure learning using PC algorithms for count data. Different methods handle different distributional assumptions:

- poi: Poisson distribution
- nb: Negative Binomial distribution

- `zinb0`: Zero-Inflated NB with structure only in mean parameter
- `zinb1`: Zero-Inflated NB with structure in both mean and zero-inflation parameters

For SummarizedExperiment and SingleCellExperiment inputs, if the specified `whichAssay` is "processed" but not found, the function will use the first assay and issue a warning recommending QPtransform().

Value

- If x is a matrix: the estimated adjacency matrix of the graph
- If x is a SummarizedExperiment: the object with adjacency matrix stored in metadata as `adj_mat`
- If x is a SingleCellExperiment: the object with adjacency matrix stored as rowPair

Examples

```
# Matrix input
mat <- matrix(rpois(50, 5), nrow = 10)
PCzinb(mat, method = "poi")

# SummarizedExperiment input
library(SummarizedExperiment)
se <- SummarizedExperiment(matrix(rpois(50, 5), ncol = 10))
se_result <- PCzinb(se, method = "poi")

# SingleCellExperiment input
library(SingleCellExperiment)
sce <- SingleCellExperiment(matrix(rpois(50, 5), ncol = 10))
sce_result <- PCzinb(sce, method = "poi")
rowPair(sce_result)
```

---

plotg *Visualize Graphs from Adjacency Matrices*

Description

Generates and arranges multiple graph visualizations from a list of adjacency matrices. Each matrix is converted into an undirected **igraph** object and visualized using a force-directed layout via **ggraph**.

Usage

```
plotg(adj_matrix_list)
```

Arguments

`adj_matrix_list`
: A list of square, symmetric adjacency matrices with zeros on the diagonal (no self-loops), or a SummarizedExperiment object containing such matrices as assays. Each matrix represents an undirected graph.

Details

Each adjacency matrix is validated to ensure it is square and symmetric. Disconnected nodes (degree zero) are removed prior to visualization. Graphs are visualized with a force-directed layout using **ggraph**, and multiple plots are arranged into a grid with **gridExtra**.

Each subplot title includes the graph index, number of nodes, and number of edges.

Value

A grid of plots displaying all valid graphs in the input list.

Note

This function requires the following packages: **igraph**, **ggraph**, and **gridExtra**. If any are missing, an informative error will be thrown.

Examples

```
data(toy_counts)


# Infer networks (toy_counts is already a MultiAssayExperiment)
networks <- infer_networks(
    count_matrices_list = toy_counts,
    method = "GENIE3",
    nCores = 1
)
head(networks[[1]])

# Generate adjacency matrices
wadj_se <- generate_adjacency(networks)
swadj_se <- symmetrize(wadj_se, weight_function = "mean")

# Apply cutoff
binary_se <- cutoff_adjacency(
    count_matrices = toy_counts,
    weighted_adjm_list = swadj_se,
    n = 1,
    method = "GENIE3",
    quantile_threshold = 0.95,
    nCores = 1,
    debug = TRUE
)
head(binary_se[[1]])
plotg(binary_se)
```

---

plotROC *Plot ROC Curves for Inferred Networks*

Description

Computes and visualizes Receiver Operating Characteristic (ROC) curves for predicted adjacency matrices stored in a SummarizedExperiment object, compared against a binary ground truth network.

Usage

```
plotROC(predicted_se, ground_truth, plot_title, is_binary = FALSE)
```

Arguments

predicted_se	A SummarizedExperiment object containing predicted adjacency matrices as assays. Each matrix must share dimnames with ground_truth;entries may be binary (0/1) or continuous weights.
ground_truth	A square binary matrix indicating true interactions (1) in the upper triangle. Must match dims and names of each assay in predicted_se.
plot_title	Character string. Title for the ROC plot.
is_binary	Logical. If TRUE, treat matrices as binary predictions. Default FALSE for weighted predictions.

Details

For binary matrices, a single TPR/FPR point is computed per matrix. For weighted ones, a full ROC curve is computed from continuous scores. Diagonals are ignored; symmetry is not enforced.

Value

A list with:
auc: data frame of AUC per matrix.
plot: the ROC plot (via ggplot2).

Examples

```
data(toy_counts)
data(toy_adj_matrix)


# Infer networks (toy_counts is already a MultiAssayExperiment)
networks <- infer_networks(
    count_matrices_list = toy_counts,
    method = "GENIE3",
    nCores = 1
)
head(networks[[1]])

# Generate and symmetrize adjacency matrices (returns SummarizedExperiment)
wadj_se <- generate_adjacency(networks)
swadj_se <- symmetrize(wadj_se, weight_function = "mean")

# plotROC now accepts SummarizedExperiment directly
roc_res <- plotROC(
    swadj_se,
    toy_adj_matrix,
    plot_title = "ROC Curve: GENIE3",
    is_binary = FALSE
)
roc_res$plot
auc_joint <- roc_res$auc
```

plot_community_comparison *Visualize Community and Topology Comparison*

Description

Creates visualization plots for community assignment metrics and topological properties comparison.

Usage

```
plot_community_comparison(
  community_metrics,
  topology_measures,
  control_topology
)
```

Arguments

community_metrics	A data frame with VI, NMI, and ARI scores (output from compute_community_metrics()).
topology_measures	A data frame with Modularity, Communities, Density, and Transitivity (from compute_topology_metrics()).
control_topology	A named numeric vector of control network topology metrics (from compute_topology_metrics())

Details

This function requires the **fmsb** package for radar chart visualization. The radar plot shows normalized community similarity metrics. The bar plots compare raw topological properties between predicted and control networks.

Value

Invisibly returns NULL. Displays a radar plot for community metrics and bar plots for topology comparison.

Examples

```
data(toy_counts)
data(toy_adj_matrix)


# Infer networks (toy_counts is already a MultiAssayExperiment)
networks <- infer_networks(
    count_matrices_list = toy_counts,
    method = "GENIE3",
    nCores = 1
)

# Generate adjacency matrices
```

```
wadj_se <- generate_adjacency(networks)
swadj_se <- symmetrize(wadj_se, weight_function = "mean")

# Apply cutoff
binary_se <- cutoff_adjacency(
    count_matrices = toy_counts,
    weighted_adjm_list = swadj_se,
    n = 1,
    method = "GENIE3",
    quantile_threshold = 0.95,
    nCores = 1
)

consensus <- create_consensus(binary_se, method = "union")
comm_cons <- community_path(consensus)
comm_truth <- community_path(toy_adj_matrix)

comm_metrics <- compute_community_metrics(comm_truth, list(comm_cons))
topo_metrics <- compute_topology_metrics(comm_truth, list(comm_cons))

plot_community_comparison(
    comm_metrics,
    topo_metrics$topology_measures,
    topo_metrics$control_topology
)
```

`plot_network_comparison` *Visualize Network Comparison*

Description

Creates visualization plots comparing consensus and reference networks, showing True Positives (TP), False Negatives (FN), and optionally False Positives (FP) edges.

Usage

```
plot_network_comparison(edge_classification, show_fp = FALSE)
```

Arguments

`edge_classification`	A list output from `classify_edges()` containing edge classifications and graph objects.
`show_fp`	Logical. If TRUE, displays False Positive edges in a separate plot. Default: FALSE.

Details

This function requires the **ggraph** and **ggplot2** packages. If `show_fp = TRUE`, the **patchwork** package is also required.

The plots differentiate:

- TP/CE (True Positives/Confirmed Edges): Red edges present in both
- FN/ME (False Negatives/Missing Edges): Blue edges in reference only
- FP/EE (False Positives/Extra Edges): Edges in consensus only

If STRINGdb was used for reference, labels are CE/ME/EE. Otherwise, TP/FN/FP.

Value

A ggplot object visualizing the comparison. If show_fp = TRUE, a combined plot using patchwork is returned.

Examples

```
data(toy_counts)
data(toy_adj_matrix)


# Infer networks (toy_counts is already a MultiAssayExperiment)
networks <- infer_networks(
    count_matrices_list = toy_counts,
    method = "GENIE3",
    nCores = 1
)

# Generate and symmetrize adjacency matrices (returns SummarizedExperiment)
wadj_se <- generate_adjacency(networks)
swadj_se <- symmetrize(wadj_se, weight_function = "mean")

# Apply cutoff (returns SummarizedExperiment)
binary_se <- cutoff_adjacency(
    count_matrices = toy_counts,
    weighted_adjm_list = swadj_se,
    n = 1,
    method = "GENIE3",
    quantile_threshold = 0.95,
    nCores = 1
)

# Create consensus (returns SummarizedExperiment)
consensus <- create_consensus(binary_se, method = "union")

# Wrap reference matrix in SummarizedExperiment
ref_se <- build_network_se(list(reference = toy_adj_matrix))

# Classify edges
edge_class <- classify_edges(consensus, ref_se)

# Plot comparison
plot_network_comparison(edge_class, show_fp = FALSE)
```

pscores *Compute Performance Scores for Predicted Adjacency Matrices*

Description

Computes classification metrics by comparing predicted adjacency matrices to a ground truth binary network and visualizes the performance via a radar (spider) plot.

Usage

```
pscores(ground_truth, predicted_list, zero_diag = TRUE)
```

Arguments

ground_truth	A square binary adjacency matrix representing the ground truth network. Values must be 0 or 1. Only the upper triangle is used for evaluation.
predicted_list	A list of predicted adjacency matrices to evaluate, or a SummarizedExperiment object containing such matrices as assays. Each matrix must have the same dimensions and row/column names as ground_truth.
zero_diag	Logical. If TRUE (default), sets the diagonal of ground_truth to zero before evaluation, removing self-loops.

Details

For each predicted matrix, the confusion matrix is computed using the upper triangle (non-self edges). Metrics including True Positive Rate (TPR), False Positive Rate (FPR), Precision, F1-score, and Matthews Correlation Coefficient (MCC) are calculated.

A radar plot is automatically generated summarizing the key scores across matrices.

Value

A list with one element:
Statistics: Data frame of evaluation metrics (TP, TN, FP, FN, TPR, FPR, Precision, F1, MCC) for each predicted matrix.

Note

Requires the **fmsb**, **dplyr**, and **tidyr** packages.

Examples

```
data(toy_counts)
data(toy_adj_matrix)


# Infer networks (toy_counts is already a MultiAssayExperiment)
networks <- infer_networks(
    count_matrices_list = toy_counts,
    method = "GENIE3",
    nCores = 1
)
```

```
# Generate adjacency matrices
wadj_se <- generate_adjacency(networks)
swadj_se <- symmetrize(wadj_se, weight_function = "mean")

# Apply cutoff
binary_se <- cutoff_adjacency(
    count_matrices = toy_counts,
    weighted_adjm_list = swadj_se,
    n = 1,
    method = "GENIE3",
    quantile_threshold = 0.95,
    nCores = 1,
    debug = TRUE
)

pscores_data <- pscores(toy_adj_matrix, binary_se)
```

`selgene` *Select Top Expressed Genes from Single-Cell Data*

Description

Identifies and returns the top `n` most highly expressed genes across all cells or within a specific cell type. Supports objects of class Seurat, SingleCellExperiment, or a numeric expression matrix (genes × cells).

Usage

```
selgene(
  object,
  top_n,
  cell_type = NULL,
  cell_type_col = "cell_type",
  assay = NULL,
  remove_mt = FALSE,
  remove_rib = FALSE
)
```

Arguments

`object`	A Seurat object, SingleCellExperiment object, or numeric matrix (genes × cells).
`top_n`	Integer. Number of top expressed genes to return.
`cell_type`	Optional string. If provided, filters the expression matrix to only include cells of this type.
`cell_type_col`	Character. Name of the column in metadata (Seurat `meta.data` or SCE `colData`) containing cell type annotations. Default is `"cell_type"`.
`assay`	Character. For SingleCellExperiment objects only. Name of the assay to use. If `NULL`, defaults to `"logcounts"`.
`remove_mt`	Logical. If `TRUE`, remove mitochondrial genes matching `"^MT-"` (case-insensitive).
`remove_rib`	Logical. If `TRUE`, remove ribosomal genes matching `"^RP[SL]"` (case-insensitive).

Details

The function assumes that log-normalized values are available in the "data" slot (for Seurat objects) or the "logcounts" assay (for SingleCellExperiment). If raw counts are provided as a matrix, no transformation is applied.

Optional filtering is available to exclude mitochondrial genes ("^MT-") and ribosomal genes ("^RP[SL]"), which may otherwise dominate the top expressed genes.

Value

A character vector of the top n most highly expressed gene names.

Details

When using a Seurat object, the function retrieves the log-normalized data from the default assay's "data" slot. For SingleCellExperiment, it uses the specified assay (default is "logcounts"). For matrices, no checks or transformations are applied, and subsetting by cell type is not supported.

Mitochondrial and ribosomal gene removal is based on regular expressions matching gene names. These should follow standard naming conventions (e.g., MT-ND1, RPL13A, RPS6).

See Also

SingleCellExperiment

Examples

```
data(toy_counts)
genes <- selgene(
    object = toy_counts[[1]],
    top_n = 5,
    cell_type = "T_cells",
    cell_type_col = "CELL_TYPE",
    remove_rib = TRUE,
    remove_mt = TRUE,
    assay = "counts"
)
```

---

stringdb_adjacency *Build Adjacency Matrices for Physical Interactions from STRING (POST API)*

Description

Constructs weighted and binary adj matrices for physical protein-protein interactions using a POST request to the STRING database API.

Usage

```
stringdb_adjacency(
  genes,
  species = 9606,
  required_score = 400,
  keep_all_genes = TRUE,
  verbose = TRUE
)
```

Arguments

genes	A character vector of gene symbols or identifiers, e.g., c("TP53", "BRCA1", ...).
species	Integer. NCBI taxonomy ID of the species. Default is 9606 (human).
required_score	Integer in \[0,1000\]. Minimum confidence score for interactions. Default is 400.
keep_all_genes	Logical. If TRUE (default), includes all input genes in the final matrix even if unmapped.
verbose	Logical. If TRUE, displays progress messages. Default is TRUE.

Details

This function:

1. Maps input genes to STRING internal IDs.
2. Uses a POST request to retrieve physical protein-protein interactions from STRING.
3. Builds a weighted adjacency matrix using the STRING combined score.
4. Builds a binary adjacency matrix indicating presence/absence.

Genes not mapped to STRING are optionally retained as zero rows/columns if keep_all_genes = TRUE.

Value

A list containing:

- weighted: A square numeric adjacency matrix with scores as weights.
- binary: A corresponding binary (0/1) adjacency matrix.

Note

Requires packages: **STRINGdb**, **httr**, **jsonlite**.

Examples

```
data(toy_counts)
genes <- selgene(
    object = toy_counts[[1]],
    top_n = 5,
    cell_type = "T_cells",
    cell_type_col = "CELL_TYPE",
    remove_rib = TRUE,
```

```
    remove_mt = TRUE,
    assay = "counts"
)

str_res <- stringdb_adjacency(
    genes = genes,
    species = 9606,
    required_score = 900,
    keep_all_genes = FALSE
)
wadj_truth <- str_res$weighted
toy_adj_matrix <- str_res$binary
```

`symmetrize` *Symmetrize Adjacency Matrices in a SummarizedExperiment*

Description

Symmetrizes each adjacency matrix in a SummarizedExperiment by ensuring entries (i, j) and (j, i) are identical, using a specified combination function.

Usage

```
symmetrize(matrix_list, weight_function = "mean", nCores = 1)
```

Arguments

`matrix_list`	A SummarizedExperiment object containing adjacency matrices to symmetrize.
`weight_function`	Character string or function. Method to combine entries (i, j) and (j, i). Options include `"mean"`, `"max"`, `"min"`, or a user-defined function.
`nCores`	Integer. Number of CPU cores to use for parallel processing. Defaults to the number of available workers in the current **BiocParallel** backend.

Details

For each pair of off-diagonal elements (i, j) and (j, i):

- If one value is zero, the non-zero value is used.
- If both are non-zero, they are combined using the specified `weight_function`.

Diagonal entries are preserved as-is and not modified.

Parallelization is managed via **BiocParallel** for improved performance.

Value

A SummarizedExperiment object with symmetrized adjacency matrices, where for each matrix $A[i, j] = A[j, i]$ for all $i \neq j$.

Examples

```
data("toy_counts")


# Infer networks (toy_counts is already a MultiAssayExperiment)
networks <- infer_networks(
    count_matrices_list = toy_counts,
    method = "GENIE3",
    nCores = 1
)
head(networks[[1]])

# Generate adjacency matrices
wadj_se <- generate_adjacency(networks)
swadj_se <- symmetrize(wadj_se, weight_function = "mean")
```

---

toy_adj_matrix *Toy adjacency matrix for examples*

Description

An adjacency matrix generated using real dataset and STRINGdb for demonstrating functions in the scGraphVerse package.

Usage

```
data(toy_adj_matrix)
```

Format

A matrix of dimension 10x10.

Examples

```
data(toy_adj_matrix)
str(toy_adj_matrix)
```

---

toy_counts *Toy MultiAssayExperiment for Network Inference*

Description

A simulated dataset generated using zinb_simdata() for demonstrating network inference functions in the scGraphVerse package.

Usage

```
data(toy_counts)
```

Format

A MultiAssayExperiment object containing 3 SingleCellExperiment objects (experiments), each with 10 genes x 40 cells. The data was simulated using a known ground truth network (toy_adj_matrix) with zero-inflated negative binomial distributions.

Examples

```
data(toy_counts)
toy_counts

# Access individual experiments
MultiAssayExperiment::experiments(toy_counts)

# Use directly with infer_networks
networks <- infer_networks(
    count_matrices_list = toy_counts,
    method = "GENIE3",
    nCores = 1
)
```

`zinb.regression.parseModel`

Parse ZINB regression model

Description

Given the parameters of a ZINB regression model, this function parses the model and computes the vector of log(mu), logit(pi), and the dimensions of the different components of the vector of parameters.

Usage

```
zinb.regression.parseModel(alpha, A.mu, A.pi)
```

Arguments

`alpha`	the vectors of parameters c(a.mu, a.pi) concatenated
`A.mu`	matrix of the model (default=empty)
`A.pi`	matrix of the model (default=empty)

Value

A list with slots `logMu`, `logitPi`, `dim.alpha` (a vector of length 2 with the dimension of each of the vectors `a.mu`, `a.pi` in `alpha`), and `start.alpha` (a vector of length 2 with the starting indices of the 2 vectors in `alpha`)

zinb0.noT — *Structure learning with zero-inflated negative binomial model (mean only)*

Description

This function estimates the adjacency matrix of a ZINB model given a matrix of counts, using the optim function. Uses BiocParallel for parallelization.

Usage

```
zinb0.noT(X, maxcard, alpha, extend, nCores = 1)
```

Arguments

X	the matrix of counts (n times p).
maxcard	the uper bound of the cardinality of the conditional sets K
alpha	the significant level of the tests
extend	if TRUE it considers the union of the tests, otherwise it considers the intersection.
nCores	number of cores for parallelization

Details

This approach assumes that the structure of the graph only depends on the mean parameter, treating zero inflation as a technical noise effect. We call this model `zinb0`.

Value

the estimated adjacency matrix of the graph.

zinb1.noT — *Structure learning with zero-inflated negative binomial model*

Description

This function estimates the adjacency matrix of a ZINB model given a matrix of counts, using the optim function. Uses BiocParallel for parallelization.

Usage

```
zinb1.noT(X, maxcard, alpha, extend, nCores = 1)
```

Arguments

X	the matrix of counts (n times p).
maxcard	the uper bound of the cardinality of the conditional sets K
alpha	the significant level of the tests
extend	if TRUE it considers the union of the tests, otherwise it considers the intersection.
nCores	number of cores for parallelization

Details

This approach assumes that the structure of the graph depends on both the mean parameter and the zero inflation parameter. We call this model `zinb1`.

Value

the estimated adjacency matrix of the graph.

`zinbOptimizeDispersion` *(ZINB) model. The ZINB distribution is parametrized by three parameters: the mean value and the dispersion of the negative binomial distribution, and the probability of the zero component.*

Description

(ZINB) model. The ZINB distribution is parametrized by three parameters: the mean value and the dispersion of the negative binomial distribution, and the probability of the zero component.

Usage

```
zinbOptimizeDispersion(mu, logitPi, Y, n)
```

Arguments

`mu`	the vector mean of the negative binomial
`logitPi`	the vector of logit of the probabilities of the zero component
`Y`	the vector of counts
`n`	length of the returned vector

Value

A vector of length n with the optimized dispersion parameter values.

`zinb_simdata` *Simulate Zero-Inflated Negative Binomial (ZINB) Count Matrices with Sequencing Depth*

Description

Simulates one or more count matrices following a zero-inflated negative binomial (ZINB) distribution, incorporating gene-gene interaction structures and cell-specific sequencing depth variation.

Usage

```
zinb_simdata(
  n,
  p,
  B,
  mu_range,
  mu_noise,
  theta,
  pi,
  kmat = 1,
  depth_range = NA
)
```

Arguments

n	Integer. Number of cells (samples) in each simulated matrix.
p	Integer. Number of genes (features) in each simulated matrix.
B	A symmetric binary adjacency matrix (0/1) defining gene-gene connectivity. Row and column names correspond to gene names.
mu_range	List of numeric vectors (length 2 each). Range of gene expression means for each simulated matrix.
mu_noise	Numeric vector. Mean of background noise for each matrix.
theta	Numeric vector. Dispersion parameters of the negative binomial distribution for each matrix. Smaller theta implies higher overdispersion.
pi	Numeric vector. Probability of excess zeros (0 < pi < 1) for each matrix.
kmat	Integer. Number of count matrices to simulate. Default is 1.
depth_range	Numeric vector of length 2 or NA. Range of total sequencing depth per cell. If NA, no depth adjustment is performed.

Details

Each simulated matrix:

1. Generates gene expression values based on a ZINB model.
2. Modulates expression using the adjacency matrix B.
3. Applies random sequencing depth scaling if depth_range is provided.

Useful for benchmarking single-cell RNA-seq network inference methods with dropout events and network structure.

Value

A list containing kmat matrices. Each matrix has:

- Rows representing cells (cell_1, ..., cell_n).
- Columns representing genes (rownames(B)).
- Count values following a ZINB distribution.

Examples

```
data(toy_adj_matrix)
nodes <- nrow(toy_adj_matrix)
sims <- zinb_simdata(
    n = 50,
    p = nodes,
    B = toy_adj_matrix,
    mu_range = list(c(1, 4), c(1, 7), c(1, 10)),
    mu_noise = c(1, 3, 5),
    theta = c(1, 0.7, 0.5),
    pi = c(0.2, 0.2, 0.2),
    kmat = 3,
    depth_range = c(0.8 * nodes * 3, 1.2 * nodes * 3)
)
```

Index

∗ **datasets**
toy_adj_matrix, 41
toy_counts, 41
∗ **internal**
nb.loglik, 25
nb.loglik.dispersion, 26
nb.loglik.regression, 26
nb.loglik.regression.gradient, 27
nb.OptimizeDispersion, 28
nb.regression.parseModel, 28
zinb.regression.parseModel, 42
zinb0.noT, 43
zinb1.noT, 43
zinbOptimizeDispersion, 44

build_network_se, 3

classify_edges, 4, *9*
community_path, 5
community_similarity, 7
compare_consensus, 9
compute_community_metrics, *7, 8*, 10
compute_topology_metrics, *7, 8*, 11
consensusNet, *13*
create_consensus, 13
create_mae, 15
cutoff_adjacency, 16

download_Atlas, 17

earlyj, 18
edge_mining, 19

generate_adjacency, 20

infer_networks, 21
init_py, 24

MultiAssayExperiment, *16*, *18*, *22*, *42*

nb.loglik, 25
nb.loglik.dispersion, 26
nb.loglik.regression, 26, *27–29*
nb.loglik.regression.gradient, 27
nb.OptimizeDispersion, 28
nb.regression.parseModel, 28

PCzinb, 29
plot_community_comparison, *7, 8*, 33
plot_network_comparison, *9*, 34
plotg, 30
plotROC, 31
pscores, 36

selgene, 37
SingleCellExperiment, *37, 38*
stringdb_adjacency, 38
SummarizedExperiment, *4*, *6*, *9*, *13, 14*, *16, 17*, *19–21*, *30–32*, *36*, *40*
symmetrize, 40

toy_adj_matrix, 41
toy_counts, 41

zinb.regression.parseModel, 42
zinb0.noT, 43
zinb1.noT, 43
zinb_simdata, 44
zinbOptimizeDispersion, 44