Package: singIST (via r-universe)

November 18, 2026

Title comparative single-cell transcriptomics between disease models and a human condition

Version 1.0.2

Description Provides with toolkits to implement a full singIST analysis with pseudobulked Seurat objects of disease models and human data.

License MIT + file LICENSE

Encoding UTF-8

Roxygen list(markdown = TRUE)

Suggests testthat (>= 3.0.0), BiocStyle, knitr, qpdf, utils, RcppAlgos, glmGamPoi, methods, sp

Config/testthat/edition 3

Imports msigdb, GSEABase, checkmate, stats, asmbPLS, BiocParallel, stringr, FactoMineR, Seurat, SeuratObject, biomaRt, data.table, purrr, SingleCellExperiment, SummarizedExperiment, scran, scuttle, missMDA, S4Vectors

VignetteBuilder knitr

Depends R (>= 4.5.0)

LazyData false

biocViews SingleCell, Classification, Transcriptomics

URL https://github.com/DataScienceRD-Almirall/singIST

BugReports https://github.com/DataScienceRD-Almirall/singIST/issues

Config/roxygen2/version 8.1.0

Config/pak/sysreqs cmake libfreetype6-dev libglpk-dev make libicu-dev libpng-dev libuv1-dev libxml2-dev libssl-dev libx11-dev zlib1g-dev

Repository Bioconductor 3.23

Date/Publication 2026-08-20 21:48:45 UTC

RemoteUrl https://github.com/bioc/singIST

RemoteRef RELEASE_3_23

RemoteSha 065e6d2fce97c7259f0e5e86885875d5f6fd921e

Contents

add_missing_psb_rows . . . 3
asmbPLSDA.cv.kcv . . . 3
asmbPLSDA.cv.loo . . . 5
biological_link_function . . . 6
celltype_mapping . . . 8
celltype_recap . . . 9
check_fit_model . . . 10
check_hyperparameters . . . 10
check_mapping_organism . . . 11
check_pathway . . . 12
check_superpathway . . . 13
check_superpathway_input . . . 14
CIP_GIP . . . 15
CIP_GIP_test . . . 16
clean_mfa_data . . . 17
create_fit_model . . . 18
create_hyperparameters . . . 19
create_mapping_organism . . . 20
create_pathway . . . 21
create_superpathway . . . 22
create_superpathway_input . . . 23
derive_contributions . . . 25
derive_scores . . . 26
diff_expressed . . . 27
fit_mfa_imputer . . . 28
fitOptimal . . . 29
gene_contrib . . . 31
helpers . . . 32
matrixToBlock . . . 43
multiple_check . . . 44
multiple_fitOptimal . . . 45
multiple_singISTrecapitulations . . . 46
orthology_mapping . . . 47
permut_asmbplsda . . . 49
permut_asmbplsda_kcv . . . 50
predict_mfa_imputer . . . 51
pullGeneSet . . . 51
render_multiple_outputs . . . 52
restore_removed_columns . . . 52
Results_comparison_measure . . . 53
setGeneSetsCelltype . . . 54
singIST_treat . . . 55
singISTrecapitulations . . . 56
superpathway_recap . . . 57
update_group_sizes . . . 58
wilcox_CIP_GIP . . . 58

Index **60**

`add_missing_psb_rows` *Ensure all celltype–sample combinations are present in the pseudobulkmatrix*

Description

Ensure all celltype–sample combinations are present in the pseudobulkmatrix

Usage

```
add_missing_psb_rows(mat, celltypes, sample_ids)
```

Arguments

`mat`	A numeric matrix with rownames in the form "celltype_sample".
`celltypes`	Character vector of the celltypes you intend to include (in the exact order of `object@superpathway_info@celltypes`).
`sample_ids`	Character vector of all sample identifiers (in the order of `rownames(object@pseudobulk_lognorm)` split by "_").

Value

A numeric matrix with `length(celltypes) * length(sample_ids)` rows, in the canonical `paste(celltypes, sample_ids, sep = "_")` order, where newly added rows are filled with `NA_real_`.

`asmbPLSDA.cv.kcv` *K-fold × Repeated Cross-Validation for asmbPLS-DA*

Description

Implements stratified K-fold cross-validation with repetitions, mirroring the structure of `asmbPLSDA.cv.loo` but using K k and ncv instead of LOO.

Usage

```
asmbPLSDA.cv.kcv(
  X.matrix,
  Y.matrix,
  PLS_term = 2,
  X.dim,
  quantile.comb.table,
  k = 4,
  ncv = 10,
  outcome.type = c("binary", "multiclass"),
```

```
  Method = NULL,
  measure = "B_accuracy",
  parallel = FALSE,
  expected.measure.increase = 0.005,
  center = TRUE,
  scale = TRUE,
  maxiter = 100
)
```

Arguments

X.matrix	Predictor matrix (n×p)
Y.matrix	Response one-hot matrix (n×q)
PLS_term	Integer: maximum number of PLS components
X.dim	Vector: feature counts per block
quantile.comb.table	Matrix (C×length(X.dim)): quantile combinations
k	Integer: number of CV k (K)
ncv	Integer: number of ncv
outcome.type	"binary" or "multiclass"
Method	Prediction method
measure	"B_accuracy","accuracy","precision","recall","F1"
parallel	Logical: TRUE to parallelize per-fold
expected.measure.increase	Numeric: min performance gain to add PLS
center	Logical: center predictors
scale	Logical: scale predictors
maxiter	Integer: max iterations for asmbPLSDA.fit

Value

A list with:

quantile_table_CV	Matrix (PLS_term × (blocks + metrics)) of optimal quantiles and CV metrics
optimal_nPLS	Integer: selected number of PLS components
splits	List of length (k*ncv) of train/validation splits

Examples

```
# example code
file <- system.file("extdata", "example_superpathway_input.rda",
package = "singIST")
load(file)
data <- example_superpathway_input
matrices <- matrixToBlock(data)
```

```
X.matrix <- matrices$block_predictor
Y.matrix <- matrices$matrix_response
X.dim <- matrices$block_dim
quantile.comb.table <- data$hyperparameters_info$quantile_comb_table
quantile.comb.table <- rbind(quantile.comb.table, c(0.1, 0.2)) # Add 2 cases
outcome.type <- data$hyperparameters_info$outcome_type
asmbPLSDA.cv.kcv(X.matrix, Y.matrix, PLS_term = 1,
X.dim,quantile.comb.table,Method = NULL, measure = "B_accuracy",
parallel = TRUE, outcome.type = outcome.type,
expected.measure.increase = 0.005, center = TRUE, scale = TRUE,
maxiter = 100)
```

`asmbPLSDA.cv.loo` *Leave-one-out Cross-validation*

Description

Leave-one-out Cross-validation

Usage

```
asmbPLSDA.cv.loo(
  X.matrix,
  Y.matrix,
  PLS_term = 1,
  X.dim,
  quantile.comb.table,
  outcome.type = c("binary", "multiclass"),
  Method = NULL,
  measure = "B_accuracy",
  parallel = FALSE,
  expected.measure.increase = 0.005,
  center = TRUE,
  scale = TRUE,
  maxiter = 100
)
```

Arguments

`X.matrix`	Predictor block matrix from `matrixToBlock`
`Y.matrix`	Response matrix from `matrixToBlock`
`PLS_term`	An integer with the number of PLS components to use passed from hyperparameter list
`X.dim`	A list with the observed gene set size for each cell type from `matrixToBlock`
`quantile.comb.table`	A matrix with the quantile comb table passed from hyperparameters list object

outcome.type	A character indicating binary or multiclass passed from hyperparameters list object
Method	A parameter passed from fitOptimal
measure	A parameter passed from fitOptimal
parallel	A parameter passed from fitOptimal
expected.measure.increase	A parameter passed from fitOptimal
center	A parameter passed from fitOptimal
scale	A parameter passed from fitOptimal
maxiter	A parameter passed from fitOptimal

Value

A list containing the optimal quantiles for each PLS component and the optimal number of PLS components.

Examples

```
file <- system.file("extdata", "example_superpathway_input.rda",
package = "singIST")
load(file)
data <- example_superpathway_input
matrices <- matrixToBlock(data)
X.matrix <- matrices$block_predictor
Y.matrix <- matrices$matrix_response
X.dim <- matrices$block_dim
quantile.comb.table <- data$hyperparameters_info$quantile_comb_table
outcome.type <- data$hyperparameters_info$outcome_type
asmbPLSDA.cv.loo(X.matrix, Y.matrix, PLS_term = 1, X.dim,quantile.comb.table,
Method = NULL, measure = "B_accuracy", parallel = TRUE,
outcome.type = outcome.type, expected.measure.increase = 0.005,
center = TRUE, scale = TRUE,maxiter = 100)
```

---

biological_link_function

Biological link function

Description

Maps the organism information in mapping organism list and superpathway fit model list to obtain the "singIST treated samples" with the simulated human. The biological link function involves the cell type mapping, orthology mapping and fold change computation.

Usage

```
biological_link_function(
  object,
  model_object,
  object_gene_identifiers = "external_gene_name",
  model_species = "hsapiens",
  FC_list = NULL,
  orthology_cache = NULL,
  ...
)
```

Arguments

`object`	A mapping organism list with the disease model data
`model_object`	A superpathway fit model list with the fitted model
`object_gene_identifiers`	Annotation of gene identifiers used in `object`. By default `external_gene_name`. If `NULL` orthology_mapping infers the gene identifiers of `object`, note this may add execution time.
`model_species`	Organism for which `model_object` has been trained. By `default` `hsapiens`.
`FC_list`	Optional parameter with list of matrices containing Fold Changes provided by the user. If such list is provided, logFC will not be computed via diff_expressed and the list input will be used instead. The list length must match the number and order of human cell types modelled. Each element of the list must be a `data.frame` whose columns are: "p_value" p-value of test, "avg_log2FC" the log2FC provided, "pct.1" percent of cells where the gene is expressed in base class, "pct.2" percent of cells where the gene is expressed in target class, "p_val_adj" adjusted p-value. Rownames should contain the gene names. Note that log2FC provided should be comparable to log2FC computed in asmbPLS-DA model. log2FC should be reported as descriptive point estimates of mean difference of log2 normalized expression values.
`orthology_cache`	An optional `data.frame` with a precomputed one-to-one orthology mapping, forwarded to orthology_mapping. When provided, no Ensembl BioMart queries are performed, making the workflow reproducible offline. See orthology_mapping for the required format. By default `orthology_cache = NULL`.
`...`	Other parameters to pass onto diff_expressed

Value

A list with; ortholog gene sets as returned by orthology_mapping; a list with the Fold Changes used; singIST treated samples as returned by singIST_treat

Examples

```
file <- system.file("extdata", "example_mapping_organism.rda",
package = "singIST")
```

```
load(file)
data_organism <- example_mapping_organism
file <- system.file("extdata", "example_superpathway_fit_model.rda",
package = "singIST")
load(file)
data_model <- example_superpathway_fit_model
biological_link_function(data_organism, data_model)
```

`celltype_mapping` *Cell type mapping*

Description

For a given mapping organism list it updates the variable `celltype_cluster` so that each element of it is updated accordingly to the mapped cell types as indicated in `object$celltype_mapping`.

Usage

```
celltype_mapping(object)
```

Arguments

`object`	A mapping organism list

Value

A mapping organism list with the `object$counts` slot updated, for the variable `celltype_cluster` with the cell types according to the mapping defined in `object$celltype_mapping`.

Examples

```
file <- system.file("extdata", "example_mapping_organism.rda",
package = "singIST")
load(file)
data <- example_mapping_organism
new_object <- celltype_mapping(data)
new_object$counts$celltype_cluster
```

`celltype_recap` *Derive cell type recapitulation*

Description

Derive cell type recapitulation

Usage

```
celltype_recap(model_object, data_original, data_singIST)
```

Arguments

`model_object`	A superpathway fit model list passed from singISTrecapitulations
`data_original`	A matrix with the cell type contributions as returned by derive_contributions for the non-singIST treated samples, passed from singISTrecapitulations
`data_singIST`	A matrix with the cell type contributions as returned by derive_contributions for the singIST treated samples, passed from singISTrecapitulations

Value

A `data.frame` object with the variables: `pathway` name, `celltype` with the cell type name, `recapitulation` with the cell type recapitulation, and `reference` with the cell type reference recapitulation

Examples

```
file <- system.file("extdata", "example_superpathway_fit_model.rda",
package = "singIST")
load(file)
model <- example_superpathway_fit_model
file <- system.file("extdata", "example_mapping_organism.rda",
package = "singIST")
load(file)
mapped <- example_mapping_organism

singIST_samples <- biological_link_function(mapped, model)$singIST_samples
original <- derive_contributions(model, singIST_samples)
derived <- derive_contributions(model, model$model_fit$predictor_block)
celltype_recap(model, original$celltype_contribution,
derived$celltype_contribution)
```

check_fit_model *Validate superpathway fit model*

Description

Checks that all provided fields for a fitted asmbPLS-DA model meet the expected properties:

- superpathway_input must be a valid superpathway_input object (validated with check_superpathway_input()).
- hyperparameters_fit must be valid hyperparameters (validated with check_hyperparameters()).
- model_fit and model_validation must be lists.

Usage

```
check_fit_model(
  superpathway_input,
  hyperparameters_fit,
  model_fit,
  model_validation
)
```

Arguments

superpathway_input	List. A superpathway object created by create_superpathway().
hyperparameters_fit	List. A hyperparameters object created by create_hyperparameters().
model_fit	List. Fitted model details.
model_validation	List. Validation metrics of the fitted model.

Value

TRUE if all checks pass; otherwise, an error is thrown.

check_hyperparameters *Validate asmbPLS-DA hyperparameters*

Description

Checks that all provided hyperparameters meet the expected properties:

- quantile_comb_table must be a matrix with at least one row.
- outcome_type must be "binary" or "multiclass".
- number_PLS must be an integer >= 0.
- folds_CV and repetition_CV must be integers >= 0 (or NULL).

Usage

```
check_hyperparameters(
  quantile_comb_table,
  outcome_type,
  number_PLS,
  folds_CV,
  repetition_CV
)
```

Arguments

quantile_comb_table	Matrix. Quantile (lambda) sparsity values for CV.
outcome_type	Character. Either "binary" or "multiclass".
number_PLS	Integer. Maximum number of PLS components.
folds_CV	Integer or NULL. Number of folds for CV (default 5).
repetition_CV	Integer or NULL. Number of repetitions for CV (default 10).

Value

TRUE if all checks pass; otherwise, an error is thrown.

Examples

```
quantile_comb_table <- base::as.matrix(RcppAlgos::permuteGeneral(seq(0.05,
0.95, by = 0.50)))
check_hyperparameters(quantile_comb_table, "binary", 3L, 1L, 1L)
```

---

check_mapping_organism

Validate mapping organism input

Description

Validates that all provided fields for a mapping organism meet expected properties:

- organism, target_class, base_class are scalar characters.
- celltype_mapping is a list with non-empty names; each entry is a character vector (length 0 allowed).
- counts is a Seurat or SingleCellExperiment object with metadata columns: 'class' and 'celltype_cluster'.
- target_class and base_class exist in meta$class and are different (after normalization).
- all clusters referenced in celltype_mapping exist in meta$celltype_cluster.

Usage

```
check_mapping_organism(
  organism,
  target_class,
  base_class,
  celltype_mapping,
  counts
)
```

Arguments

organism	Character(1). Scientific Latin name of the organism.
target_class	Character(1). Name of the target class for this organism.
base_class	Character(1). Name of the base class for this organism.
celltype_mapping	List. Mapping of cell types (keys) to cluster names (character vectors).
counts	Seurat or SingleCellExperiment. Contains scRNA-seq counts and metadata.

Value

TRUE if all checks pass; otherwise an informative error is thrown.

Examples

```
# Seurat example
counts <- SeuratObject::pbmc_small
colnames(slot(counts, "meta.data"))[1] <- "donor"
colnames(slot(counts, "meta.data"))[6] <- "class"
colnames(slot(counts, "meta.data"))[7] <- "celltype_cluster"
celltype_mapping <- list("T-cell" = c("T"), "Dendritic Cell" = character(0))
check_mapping_organism("Mus musculus", "g1", "g2", celltype_mapping, counts)
```

---

check_pathway *Validate pathway fields*

Description

Checks that all provided fields for a pathway meet the expected properties.

Usage

```
check_pathway(standard_name, dbsource, collection, subcollection)
```

Arguments

standard_name	Character. Pathway standard name from MsigDB.
dbsource	Character. Database source (KEGG, PID, REACTOME, BIOCARTA, WIKIPATHWAYS).
collection	Character. MsigDB collection (c2 or m2).
subcollection	Character. MsigDB subcollection (CP).

Value

TRUE if all checks pass; otherwise, an error is thrown.

Examples

```
check_pathway(
  standard_name = "KEGG_CYTOKINE_CYTOKINE_RECEPTOR_INTERACTION",
  dbsource = "KEGG",
  collection = "c2",
  subcollection = "CP"
)
```

---

check_superpathway *Validate superpathway gene sets*

Description

Checks that all provided fields for a superpathway meet the expected properties:

- pathway_info must be a valid pathway (validated with check_pathway()).
- celltypes must be a character vector with at least 2 elements.
- gene_sets_celltype must be a list (or NULL) with same length as celltypes.

Usage

```
check_superpathway(pathway_info, celltypes, gene_sets_celltype)
```

Arguments

pathway_info	List. A pathway object created by create_pathway().
celltypes	Character vector. Each element represents a cell type.
gene_sets_celltype	List of character vectors. Each element corresponds to gene sets for each cell type. Can be NULL.

Value

TRUE if all checks pass; otherwise, an error is thrown.

Examples

```
my_pathway <- create_pathway("KEGG_CYTOKINE_CYTOKINE_RECEPTOR_INTERACTION",
"KEGG", "c2", "CP")
check_superpathway(my_pathway, c("T-cell", "Dendritic Cell"), list(c("IL4",
"IL5"), c("IL13")))
```

`check_superpathway_input` *Check superpathway input for asmbPLS-DA*

Description

Checks the validity of the inputs. This version assumes that `superpathway_info` and `hyperparameters_info` are plain lists validated by `check_superpathway()` and `check_hyperparameters()`.

Usage

```
check_superpathway_input(
  superpathway_info,
  hyperparameters_info,
  pseudobulk_lognorm,
  sample_id,
  sample_class,
  base_class,
  target_class
)
```

Arguments

`superpathway_info`	A list representing a superpathway object.
`hyperparameters_info`	A list representing a hyperparameters object.
`pseudobulk_lognorm`	A pseudobulk matrix (rows: "Celltype_Sampleid", cols: genes in "HGNC" format or similar).
`sample_id`	A character vector of sample ids.
`sample_class`	A character vector with the class of each sample.
`base_class`	A character scalar indicating the base class.
`target_class`	A character scalar indicating the target class.

Value

Invisibly returns `TRUE` if all checks pass; otherwise errors.

Examples

```
# ---- Superpathway info (list) ----
my_pathway <- create_pathway("KEGG_CYTOKINE_CYTOKINE_RECEPTOR_INTERACTION",
"KEGG", "c2", "CP")
celltypes <- c("T-cell", "Dendritic Cell")

my_superpathway <- create_superpathway(my_pathway, celltypes, list(c("IL4",
"IL5"), c("IL13")))
# ---- Hyperparameters info (list) ----
quantile_comb_table <- base::as.matrix(
  RcppAlgos::permuteGeneral(seq(0.05, 0.95, by = 0.50)),
  ncol = length(celltypes)
)

my_hyperparameters <- create_hyperparameters(
  quantile_comb_table = quantile_comb_table,
  outcome_type = "binary",
  number_PLS = as.integer(3),
  folds_CV = as.integer(1),
  repetition_CV = as.integer(1)
)

# ---- Pseudobulk + labels ----
sample_id <- c("AD1", "AD2", "HC1", "HC2")
sample_class <- c("AD", "AD", "HC", "HC")
base_class <- "HC"
target_class <- "AD"

pseudobulk_lognorm <- matrix(
  rnorm(length(celltypes) * length(sample_id)),
  nrow = length(celltypes) * length(sample_id),
  ncol = length(celltypes)
)
rownames(pseudobulk_lognorm) <- as.vector(t(outer(
  celltypes, sample_id, function(x, y) paste(x, y, sep = "_")
)))

check_superpathway_input(
  superpathway_info = my_superpathway,
  hyperparameters_info = my_hyperparameters,
  pseudobulk_lognorm = pseudobulk_lognorm,
  sample_id = sample_id,
  sample_class = sample_class,
  base_class = base_class,
  target_class = target_class
)
```

CIP_GIP *Compute Cell Importance Projection (CIP) and Gene Importance Projection (GIP)*

Description

Computes CIP and GIP metrics from a superpathway fit model list for the target class

Usage

```
CIP_GIP(object)
```

Arguments

object	A superpathway fit model list

Value

A list with the CIP and GIP metrics for all cell types. The metrics are computed for the target class.

Examples

```
file <- system.file("extdata", "example_superpathway_fit_model.rda",
package = "singIST")
load(file)
data <- example_superpathway_fit_model
CIP_GIP(data)
```

CIP_GIP_test *Cell and Gene Importance Projections statistical significance*

Description

Computes Cell and Gene Importance Projection observed distribution from fitted asmbPLSDA, and its associated null distributions by permuting the block of predictor matrices. Returns a pvalue of the Mann-Whitney Wilcoxon between the observed and null distribution for each CIP and GIP.

Usage

```
CIP_GIP_test(
  object,
  npermut = 100,
  maxiter = 100,
  type = c("jackknife", "subsampling"),
  nsubsampling = 100,
  ...
)
```

Arguments

object	A superpathway fit model list
npermut	Number of permutations on response block matrices
maxiter	An integer indicating the maximum number of iterations. If NULL the default is 100.
type	Either jackknife or subsampling. If jackknife then the CIP and GIP observed distribution is generated by a jackknife procedure. If subsampling the CIP and GIP observed distribution is generated by subsampling the number of samples without replacement, each subsample is guaranteed to contain at least 2 samples per class. If a LOOCV was performed or one has small sample size it is recommended to select jackknife, otherwise select subsampling.
nsubsampling	Number of subsamples to generate CIP and GIP observed distributions. By default 100.
...	Other parameters to be passed onto wilcox_CIP_GIP

Value

A list containing; observed distributions of CIP and GIP (variability_param); its associated null distributions generated by permutations (NULL_CIP_GIP); the unadjusted pvalue of Mann-Whitney Wilcoxon for CIP distribution (CIP_pvalue); and for GIP distribution (GIP_pvalue).

Examples

```
file <- system.file("extdata", "example_superpathway_fit_model.rda",
package = "singIST")
load(file)
data <- example_superpathway_fit_model
CIP_GIP_test(data, npermut = 3, type = "jackknife")
```

---

clean_mfa_data *Clean a predictor matrix for multiblock MFA*

Description

Converts non-finite values to NA, removes any column that is 100% NA or has zero variance, and returns a logical mask of kept columns.

Usage

```
clean_mfa_data(X)
```

Arguments

X	A numeric matrix or data.frame of predictors (samples × features).

Value

A list with components:

`X_clean` Cleaned matrix (only kept columns).

`keep_cols` Logical vector, `TRUE` for columns kept.

`removed` Integer indices of removed columns.

Examples

```
clean_mfa_data(matrix(matrix(c(0,0,0,NA, 1,2), ncol = 2, nrow = 3)))
```

`create_fit_model` *Create superpathway fit model object*

Description

Creates a simple list representing a fitted asmbPLS-DA model, after validating its components.

Usage

```
create_fit_model(
  superpathway_input,
  hyperparameters_fit,
  model_fit,
  model_validation
)
```

Arguments

`superpathway_input`
: List. A superpathway object created by create_superpathway().

`hyperparameters_fit`
: List. A hyperparameters object created by create_hyperparameters().

`model_fit`
: List. Fitted model details.

`model_validation`
: List. Validation metrics of the fitted model.

Value

A list with elements: superpathway_input, hyperparameters_fit, model_fit, model_validation.

Examples

```
my_pathway <- create_pathway("KEGG_CYTOKINE_CYTOKINE_RECEPTOR_INTERACTION",
"KEGG", "c2", "CP")

celltypes <- c("T-cell", "Dendritic Cell")

my_superpathway <- create_superpathway(my_pathway, celltypes,
list(c("IL4", "IL5"), c("IL13")))

my_hyperparameters <- create_hyperparameters(matrix(1:4, nrow = 2), "binary",
3L, 5L, 10L)

#' # ---- Pseudobulk + labels ----
sample_id <- c("AD1", "AD2", "HC1", "HC2")
sample_class <- c("AD", "AD", "HC", "HC")
base_class <- "HC"
target_class <- "AD"

pseudobulk_lognorm <- matrix(
  rnorm(length(celltypes) * length(sample_id)),
  nrow = length(celltypes) * length(sample_id),
  ncol = length(celltypes)
)
rownames(pseudobulk_lognorm) <- as.vector(t(outer(
  celltypes, sample_id, function(x, y) paste(x, y, sep = "_")
)))

my_superpathway_input <- create_superpathway_input(
    superpathway_info = my_superpathway,
    hyperparameters_info = my_hyperparameters,
    pseudobulk_lognorm = pseudobulk_lognorm,
    sample_id = sample_id,
    sample_class = sample_class,
    base_class = base_class,
    target_class = target_class
    )
my_fit <- create_fit_model(my_superpathway_input, my_hyperparameters,
list(model = "fit"), list(accuracy = 0.95))
```

`create_hyperparameters` *Create asmbPLS-DA hyperparameters object*

Description

Creates a simple list representing hyperparameters for asmbPLS-DA, after validating them.

Usage

```
create_hyperparameters(
  quantile_comb_table,
  outcome_type,
  number_PLS,
  folds_CV = 5L,
  repetition_CV = 10L
)
```

Arguments

quantile_comb_table	Matrix. Quantile (lambda) sparsity values for CV.
outcome_type	Character. Either "binary" or "multiclass".
number_PLS	Integer. Maximum number of PLS components.
folds_CV	Integer or NULL. Number of folds for CV (default 5).
repetition_CV	Integer or NULL. Number of repetitions for CV (default 10).

Value

A list with elements: quantile_comb_table, outcome_type, number_PLS, folds_CV, repetition_CV.

Examples

```
quantile_comb_table <- base::as.matrix(RcppAlgos::permuteGeneral(seq(0.05,
0.95, by = 0.50)))
my_hyperparameters <- create_hyperparameters(quantile_comb_table,"binary",3L,
1L, 1L)
print(my_hyperparameters)
```

---

create_mapping_organism

Create mapping organism object

Description

Creates a simple S3 object (list with class "mapping.organism") after validating its components.

Usage

```
create_mapping_organism(
  organism,
  target_class,
  base_class,
  celltype_mapping,
  counts
)
```

Arguments

organism	Character(1). Scientific Latin name of the organism.
target_class	Character(1). Name of the target class for this organism.
base_class	Character(1). Name of the base class for this organism.
celltype_mapping	List. Mapping of cell types to clusters (character vectors).
counts	Seurat or SingleCellExperiment object with the scRNA-seq LogNormalized counts. This object should contain variables in slot(SeuratObject, meta.data) slot or slot(SingleCellExperimentObject, metadata); class indicating the class the sample belongs to; celltype_cluster indicating the cell type cluster (either character or numeric); donor indicating the sample ID.

Value

A list of class "mapping.organism" with elements:

- organism
- target_class
- base_class
- celltype_mapping
- counts

Examples

```
counts <- SeuratObject::pbmc_small
colnames(slot(counts, "meta.data"))[1] <- "donor"
colnames(slot(counts, "meta.data"))[6] <- "class"
colnames(slot(counts, "meta.data"))[7] <- "celltype_cluster"
celltype_mapping <- list("T-cell" = c("T"), "Dendritic Cell" = character(0))
obj <- create_mapping_organism("Mus musculus", "g1", "g2", celltype_mapping,
counts)
```

---

create_pathway *Create pathway object*

Description

Creates a simple list representing a pathway, after validating its fields.

Usage

```
create_pathway(standard_name, dbsource, collection, subcollection)
```

Arguments

standard_name	Character. Pathway standard name from MsigDB.
dbsource	Character. Database source (KEGG, PID, REACTOME, BIOCARTA, WIKIPATHWAYS).
collection	Character. MsigDB collection (c2 or m2).
subcollection	Character. MsigDB subcollection (CP).

Value

A list with elements: standard_name, dbsource, collection, subcollection.

Examples

```
my_pathway <- create_pathway(
  standard_name = "KEGG_CYTOKINE_CYTOKINE_RECEPTOR_INTERACTION",
  dbsource = "KEGG",
  collection = "c2",
  subcollection = "CP"
)
print(my_pathway)
```

---

create_superpathway *Create superpathway gene sets object*

Description

Creates a simple list representing a superpathway, after validating its fields.

Usage

```
create_superpathway(pathway_info, celltypes, gene_sets_celltype)
```

Arguments

pathway_info	List. A pathway object created by create_pathway().
celltypes	Character vector. Each element represents a cell type.
gene_sets_celltype	List of character vectors. Each element corresponds to gene sets for each cell type. Can be NULL.

Value

A list with elements: pathway_info, celltypes, gene_sets_celltype.

Examples

```
my_pathway <- create_pathway("KEGG_CYTOKINE_CYTOKINE_RECEPTOR_INTERACTION",
"KEGG", "c2", "CP")
my_superpathway <- create_superpathway(my_pathway, c("T-cell",
"Dendritic Cell"), list(c("IL4", "IL5"), c("IL13")))
print(my_superpathway)
```

`create_superpathway_input` *Create superpathway input for asmbPLS-DA*

Description

Creates a plain list containing the fields for superpathway input object and validates the content with `check_superpathway_input`.

Usage

```
create_superpathway_input(
  superpathway_info,
  hyperparameters_info,
  pseudobulk_lognorm,
  sample_id,
  sample_class,
  base_class,
  target_class
)
```

Arguments

`superpathway_info`	A list representing a superpathway object (formerly `superpathway.gene.sets`).
`hyperparameters_info`	A list representing a hyperparameters object (formerly `hyperparameters`).
`pseudobulk_lognorm`	A pseudobulk matrix.
`sample_id`	A character vector of sample ids.
`sample_class`	A character vector with the class of each sample.
`base_class`	A character scalar indicating the base class.
`target_class`	A character scalar indicating the target class.

Value

A list with elements:

- superpathway_info
- hyperparameters_info
- pseudobulk_lognorm
- sample_id
- sample_class
- base_class
- target_class

Examples

```
# ---- Superpathway info (list) ----
my_pathway <- create_pathway("KEGG_CYTOKINE_CYTOKINE_RECEPTOR_INTERACTION",
"KEGG", "c2", "CP")
celltypes <- c("T-cell", "Dendritic Cell")

my_superpathway <- create_superpathway(my_pathway, celltypes, list(c("IL4",
"IL5"), c("IL13")))
# ---- Hyperparameters info (list) ----
quantile_comb_table <- base::as.matrix(
  RcppAlgos::permuteGeneral(seq(0.05, 0.95, by = 0.50)),
  ncol = length(celltypes)
)

my_hyperparameters <- create_hyperparameters(
  quantile_comb_table = quantile_comb_table,
  outcome_type = "binary",
  number_PLS = as.integer(3),
  folds_CV = as.integer(1),
  repetition_CV = as.integer(1)
)

# ---- Pseudobulk + labels ----
sample_id <- c("AD1", "AD2", "HC1", "HC2")
sample_class <- c("AD", "AD", "HC", "HC")
base_class <- "HC"
target_class <- "AD"

pseudobulk_lognorm <- matrix(
  rnorm(length(celltypes) * length(sample_id)),
  nrow = length(celltypes) * length(sample_id),
  ncol = length(celltypes)
)
rownames(pseudobulk_lognorm) <- as.vector(t(outer(
  celltypes, sample_id, function(x, y) paste(x, y, sep = "_")
)))

my_superpathway_input <- create_superpathway_input(
```

```
  superpathway_info = my_superpathway,
  hyperparameters_info = my_hyperparameters,
  pseudobulk_lognorm = pseudobulk_lognorm,
  sample_id = sample_id,
  sample_class = sample_class,
  base_class = base_class,
  target_class = target_class
)
```

`derive_contributions` *Derive superpathway score, cell type contribution and gene contribution*

Description

Computes the superpathway score, its cell type contribution and gene contribution for a block of predictor matrices for its later use to compute recapitulations

Usage

```
derive_contributions(model_object, data)
```

Arguments

`model_object`	A superpathway fit model list with the fitted asmbPLSDA
`data`	A matrix with the block of predictor matrices to compute score and contributions from

Value

A list with the superpathway score for each sample, cell type and gene contributions to the former

Examples

```
file <- system.file("extdata", "example_mapping_organism.rda",
package = "singIST")
load(file)
mapped <- example_mapping_organism
file <- system.file("extdata", "example_superpathway_fit_model.rda",
package = "singIST")
load(file)
model <- example_superpathway_fit_model
singIST_samples <- biological_link_function(mapped,
model)$singIST_samples
derive_contributions(model, singIST_samples)
```

`derive_scores` *Compute predictor scores*

Description

Computes scores from the predictor to later derive the superpathway's score, cell type contribution and gene contributions, for the target class

Usage

```
derive_scores(object, data, sample)
```

Arguments

`object`	A superpathway fit model list passed from derive_contributions
`data`	Block of predictor matrices to compute scores from
`sample`	Current sample from `data` to compute scores

Value

A list containing the needed parameters to compute superpathway's score, cell type contributions and needed scores to compute gene contributions

Examples

```
file <- system.file("extdata", "example_mapping_organism.rda",
package = "singIST")
load(file)
mapped <- example_mapping_organism
file <- system.file("extdata", "example_superpathway_fit_model.rda",
package = "singIST")
load(file)
model <- example_superpathway_fit_model
singIST_samples <- biological_link_function(mapped,
model)$singIST_samples
# Derive the scores for sample 2
derive_scores(model, singIST_samples, 2)
```

`diff_expressed` *Compute differentially expressed genes with FindMarkers/findMarkers and descriptive point estimates of log2FC*

Description

Computes differentially expressed genes with `Seurat::FindMarkers`, for `Seurat` objects, or `scran::findMarkers`, for `SingleCellExperiment` objects, for the conditions indicated. Note that `Seurat::FindMarkers` will compute Wilcoxon Signed Rank Test by default, while `scran::findMarkers` will perform t-test by default instead. The reported logFC values are difference of means of log-normalized expression values with `Seurat::AggregateExpression` or `SingleCellExperiment::aggregateAcrossCells`. This logFC is consistent with the human log2FC computation by asmbPLS-DA.

Usage

```
diff_expressed(
  object,
  condition_1 = c(),
  condition_2 = c(),
  logfc.treshold = 0.25,
  assay = "RNA",
  ...
)
```

Arguments

`object`	A mapping organism list. If a `Seurat` object was provided, then `Idents(object)` assigned to variables with the conditions being tested is expected.
`condition_1`	A vector with the elements of the first factor to perform the hypothesis test. By default the mapped cell types `condition_1 = names(object$celltype_mapping)`
`condition_2`	A vector with the elements of the second factor to perform the hypothesis test with. By default the class of the organism `condition_2 = c(object$target_class), object$base_cl`
`logfc.treshold`	Sets the minimum FindMarkers log-fold change (logFC) cutoff for identifying differentially expressed genes (DEGs). By default `logfc.treshold = 0.25`.
`assay`	Specific assay being used for analysis. By default `assay = RNA`.
`...`	Other parameters to pass onto `Seurat::FindMarkers()` or `scran::findMarkers`.

Value

A list where each element is a data.frame for a cell type containing: `p_val` p-value of test, `avg_log2FC` descriptive point estimate of logFC, `pct.1` percentage of cells where the gene is detected in the base class, `pct.2` percentage of cells where the gene is detected in the target class, `p_val_adj` FDR.

Examples

```
# Set the identities
file <- system.file("extdata", "example_mapping_organism.rda",
package = "singIST")
load(file)
data_organism <- example_mapping_organism
data <- celltype_mapping(data_organism)
data$counts$test <- paste0(data$counts$celltype_cluster, "_",
data$counts$class)
SeuratObject::Idents(data$counts) <- "test"
diff_expressed(data)
```

`fit_mfa_imputer` *Fit a multiblock-MFA imputer on training data*

Description

Runs a full EM-based `imputeMFA()` on the training set then fits a pure MFA to extract the final means and loadings.

Usage

```
fit_mfa_imputer(X_train, group, ncp = 2, method = "Regularized")
```

Arguments

`X_train`	Numeric matrix (train samples × features), may contain NAs.
`group`	Integer vector of block sizes (must sum to `ncol(X_train)`).
`ncp`	Number of MFA components to use for imputation (default 2).
`method`	Method for `imputeMFA()`: `"Regularized"` or `"EM"`.

Value

A list with components:

`imputed` Matrix `X_train` with NAs filled.

`mu` Numeric vector of column means (length = ncol).

`loadings` Numeric matrix of loadings (ncol × ncp).

Examples

```
fit_mfa_imputer(matrix(c(NA,runif(19)), nrow = 5, ncol = 4), c(2,2))
```

`fitOptimal` *Cross validation and fit of asmbPLSDA*

Description

Performs Cross Validation of the provided superpathway input, fits the optimal model and computes its validation metrics. The Cross Validation can either be Leave One-Out Cross Validation (LOOCV) or K-Fold Cross Validation (KCV). A LOOCV is performed if the number of folds was set to 1 or if the number of samples per class is less than 3 for any class. A K-Fold Cross Validation (KCV) is performed if the number of folds is greater or equal than 3 and the number of samples per class is always greater than the number of folds. If the number of samples is low for some of the classes LOOCV is recommended. If KCV is performed, missing values are automatically imputed in the K-CV process. The training set is imputed via missMDA::imputeMFA(), and an FactoMineR::MFA() is trained on the imputed training set from which we extract the mean of each gene and the estimated loadings. We then estimate the validation set by projecting the samples onto MFA space of the training set. Gene whose variance is 0 are excluded from the imputation, if a gene has null variance and full of 0 values, the NA were imputed to 0.

Usage

```
fitOptimal(
  object,
  parallel = FALSE,
  measure = "B_accuracy",
  Method = NULL,
  expected_measure_increase = 0.005,
  maxiter = 100,
  global_significance_full = FALSE,
  CIP.GIP_significance_full = FALSE,
  npermut = 100,
  nbObsPermut = NULL,
  type = "jackknife",
  nsubsampling = 100,
  ...
)
```

Arguments

`object`	A superpathway input list to fit optimal asmbPLSDA.
`parallel`	A boolean indicating whether to parallelize (`TRUE`) for LOOCV on quantile combination or not (`FALSE`). Note this option is only available for LOOCV and not KCV. Default is `FALSE`.
`measure`	Accuracy measure to be used to select optimal asmbPLSDA model. Default is F1 measure. Options are: F1, accuracy, B_accuracy, precision and recall.
`Method`	Decision rule used for prediction. For binary outcome `fixed_cutoff` (default), `Euclidean_distance_X`, and `Mahalanobis_distance_X`. For categorical outcome with more than 2 levels, the methods include `Max_Y` (default), `Euclidean_distance_X`,

Mahalanobis_distance_X, Euclidean_distance_Y, and PCA_Mahalanobis_distance_Y. If NULL the default method is used for the respective outcome binary.

expected_measure_increase
: A double indicating the measure you expect to decrease by percent after including one more PLS component, this will affect the selection of optimal number of PLS components. If NULL the default is 0.005 (0.5%).

maxiter
: An integer indicating the maximum number of iterations. If NULL the default is 100.

global_significance_full
: A boolean indicating whether to return a list with information of each permutation for the global significance test of asmbPLSDA. By default FALSE. Note that if the number of permutations that is set is large, storing this information can be a burden on memory.

CIP.GIP_significance_full
: A boolean indicating whether to return a list with the observed and null distributions of CIP and GIP or only the pvalue and adjusted pvalue. By default FALSE. Note that if the number of permutations that is set is large, storing this information can be a burden on memory.

npermut
: Number of permutations for the tests. By default 100. Parameter passed onto permut_asmbplsda and CIP_GIP_test.

nbObsPermut
: An integer indicating the number of samples to permute in each permutation. By default NULL. If NULL the number of samples to permute at each permutation is randomly chosen (for each permutation). Parameter passed onto permut_asmbplsda.

type
: Either jackknife or subsampling. If jackknife then the CIP and GIP observed distribution is generated by a jackknife procedure. If subsampling the CIP and GIP observed distribution is generated by subsampling the number of samples without replacement, each subsample is guaranteed to contain at least 2 samples per class. If a LOOCV was performed or one has small sample size it is recommended to select jackknife, otherwise select subsampling. Passed onto CIP_GIP_test.

nsubsampling
: Number of subsamples to generate CIP and GIP observed distributions. By default 100. Passed onto CIP_GIP_test.

...
: Other parameters to be passed onto wilcox_CIP_GIP, wilcox test of GIP statistical tests

Value

A superpathway fit model list object with; a superpathway input list object used for CV and model fit; a hyperparameters list object with the hyperparameters used to fit the optimal model (includes optimal quantiles and PLS components from the CV step); a list with the fitted model information including: predictor and response matrices, observed gene sets, from matrixToBlock, and asmbPLSDA output; a list with the validaton metrics of fitted model.

Examples

```
# fitOptimal with jackknife for CIP/GIP statistics and 10 permutations
```

```
# for the global significance test of the optimal model
file <- system.file("extdata", "example_superpathway_input.rda",
package = "singIST")
load(file)
data <- example_superpathway_input
fitOptimal(data, npermut = 10, type = "jackknife")
# fitOptimal with subsampling for CIP/GIP statistics with
# 10 subsamples and 50 permutations for the global significance test of the
# optimal model
fitOptimal(data, npermut = 50, type = "subsampling",
nsubsampling = 10)
```

`gene_contrib` *Derive gene contribution to cell type recapitulation*

Description

Derive gene contribution to cell type recapitulation

Usage

```
gene_contrib(model_object, data_original, data_singIST, cell_reference)
```

Arguments

`model_object`	A superpathway fit model list passed from singISTrecapitulations
`data_original`	A matrix with the gene contributions to superpathway's score as returned by derive_contributions for the non-singIST treated samples, passed from singISTrecapitulations
`data_singIST`	A matrix with the gene contributions to superpathway's score as returned by derive_contributions for the singIST treated samples, passed from singISTrecapitulations
`cell_reference`	A matrix with the cell type recapitulations as returned by celltype_recap

Value

A `data.frame` object with the variables: `pathway` name, `celltype` name, gene name, `contribution` gene contribution to cell type recapitulation

Examples

```
file <- system.file("extdata", "example_superpathway_fit_model.rda",
package = "singIST")
load(file)
model <- example_superpathway_fit_model
file <- system.file("extdata", "example_mapping_organism.rda",
package = "singIST")
load(file)
```

```
mapped <- example_mapping_organism

singIST_samples <- biological_link_function(mapped, model)$singIST_samples
original <- derive_contributions(model, singIST_samples)
derived <- derive_contributions(model, model$model_fit$predictor_block)
# Derive cell type reference
cell <- celltype_recap(model, original$celltype_contribution,
derived$celltype_contribution)
# Compute gene contributions
gene_contrib(model, original$gene_contribution, derived$gene_contribution,
cell)
```

`helpers` *Update block of predictor matrices in matrixToBlock()*

Description

Fill up matrix with the corresponding expression values

`get_measure_index()` returns the index associated to each performance measure

Splits the predictor and response matrices into training and validation sets for leave-one-out cross-validation.

Function to train and validate asmbPLSDA excluding one observation parallelized for each quantile combination provided

Computes the prediction accuracy for different quantile combinations by fitting the asmbPLSDA model and making predictions.

Performs leave-one-out cross-validation (LOO-CV) in parallel.

Computes the performance measure selected between the training LOOCV samples and the validation LOOCV samples for all the quantile combination

For an optimal quantile combination and PLS component it computes its performance metrics between the training and validation sets

Selects the optimal number of PLS according to the performance measure

Iterates over all quantiles to generate the fitted asmbPLSDA for each and its associated predicted values

Creates a structured list to store permutation results.

Performs random permutations of the response matrix.

Calculates correlation, percentage change, and RV coefficient.

Selects sample indices for training and validation.

Fits the asmbPLS-DA model using permuted data.

Computes the p-value for the observed CV error against the null distribution of errors generated from permutation testing.

Calculates the 95% confidence interval for the null distribution of permutation errors.

Perform the jackknife resampling procedure for CIP/GIP calculations.

Perform the subsampling procedure for CIP/GIP calculations.

Generate null distributions of CIP and GIP using permutations.

Permute the X matrix to generate a null distribution.

Compute p-values by applying the Mann-Whitney test.

This helper function performs either Leave-One-Out Cross Validation (LOOCV) or K-Fold Cross Validation (KCV) on the given dataset and returns the optimal hyperparameters based on the specified accuracy measure.

This helper function computes various validation metrics, including global significance, CIP/GIP significance, and adjusted p-values for the fitted model based on cross-validation results.

For a given gene set it identifies the annotation of the genes, it does so if the genes have more than 50% match with a given annotation. Annotation must be either Ensembl, Entrez or Gene Symbols.

Retrieves one to one orthologs between from_species and to_species of orthology_mapping

Applies the biological link function conditions onto a predictor block matrix. The resulting gene expression of the predictor block are the cases defined in the biological link function.

Centers and scales each column of the predictor block matrices. The centering and scaling is according to the centroid and variance estimated in `fit_asmb`.

Given a block and the dimensions of all blocks it returns the indices of the genes belonging to that block within the predictor block matrix

Performs loading deflation for a given predictor block and PLS component

Usage

```
update_block(
  celltype,
  observed_gene_sets,
  block_predictor = block_predictor,
  matrix = matrix
)

get_measure_index(measure)

get_train_val_sets(X.matrix, Y.matrix, validation_index)

quantile_computation(
  j,
  ...,
  results_CV_summary_n,
  F_matrix_validation_bind,
  X.matrix,
  Y.matrix,
  PLS_term = 1,
  X.dim,
  quantile.comb.table,
  outcome.type = c("binary", "multiclass"),
  quantile_table_CV,
```

```
  K,
  n_quantile_comb,
  Method = NULL,
  measure = "B_accuracy",
  expected.measure.increase = 0.005,
  center = TRUE,
  scale = TRUE,
  maxiter = 100
)

evaluate_quantile_combinations(
  j,
  results_CV_summary_n,
  F_matrix_validation_bind,
  E_matrix_training,
  F_matrix_training,
  E_matrix_validation,
  F_matrix_validation,
  quantile_table_CV,
  i,
  X.dim,
  quantile.comb.table,
  outcome.type,
  center,
  scale,
  maxiter,
  Method
)

execute_parallel_cv(
  K,
  results_CV_summary_n,
  F_matrix_validation_bind,
  X.matrix,
  Y.matrix,
  PLS_term,
  X.dim,
  quantile.comb.table,
  outcome.type,
  quantile_table_CV,
  Method,
  measure,
  expected.measure.increase,
  center,
  scale,
  maxiter,
  BPPARAM = BiocParallel::bpparam()
)
```

```
performance_measures(
  n_quantile_comb,
  results_CV_summary_n,
  F_matrix_validation_bind,
  outcome.type,
  measure_selected
)

compute_final_measures(
  K,
  X.matrix,
  Y.matrix,
  i,
  X.dim,
  quantile_table_CV,
  outcome.type,
  center,
  scale,
  maxiter,
  Method
)

select_optimal_PLS(
  PLS_term,
  quantile_table_CV,
  X.dim,
  measure_selected,
  expected.measure.increase
)

execute_sequential_cv(
  K,
  n_quantile_comb,
  results_CV_summary_n,
  F_matrix_validation_bind,
  X.matrix,
  Y.matrix,
  PLS_term,
  X.dim,
  quantile.comb.table,
  outcome.type,
  quantile_table_CV,
  measure,
  expected.measure.increase,
  center,
  scale,
  maxiter,
```

```
  Method
)

initialize_results(npermut, q)

permute_Y_matrix(Y.matrix, nr, nbObsPermut, j)

compute_permutation_stats(res, Y.matrix, Ypermut, j, q, nr)

select_samples(object, nr, Nc)

fit_permuted_model(object, X_train, Y_train, maxiter)

evaluate_performance(
  res,
  Modelpermut,
  X_train,
  X_val,
  Y.matrix,
  s,
  measure,
  j,
  nr,
  Method,
  object
)

compute_pvalue(null_errors, CV_error)

compute_IC95(m)

jackknife_CIP_GIP(object, X.matrix, Y.matrix, K, maxiter, X.dim)

subsampling_CIP_GIP(
  object,
  X.matrix,
  Y.matrix,
  K,
  M,
  nsubsampling,
  maxiter,
  X.dim
)

generate_null_distributions(
  object,
  X.matrix,
  Y.matrix,
```

```
  npermut,
  K,
  X.dim,
  maxiter
)

permute_X_matrix(X.matrix, K, X.dim)

calculate_pvalues(variability, null_dist, test_func, ...)

perform_cv(
  object,
  model_block_matrices,
  nFC,
  measure,
  parallel,
  expected_measure_increase,
  maxiter,
  Method
)

compute_validation_metrics(
  output,
  optimal_hyperparameters,
  model_block_matrices,
  npermut,
  nbObsPermut,
  maxiter,
  global_significance_full,
  CIP.GIP_significance_full,
  type,
  nsubsampling,
  measure,
  Method
)

detect_gene_type(gene_set, mart)

retrieve_one2one_orthologs(
  annotation,
  gene_set,
  mart,
  from_species,
  to_species
)

FCtoExpression(model_object, b, samples, predictor_block, FC)
```

```
center_scale(data, fit_asmb)

get_indices(j, X.dim)

deflate_prediction(data, PLS, delta_cbind, fit_asmb)
```

Arguments

celltype	Cell types modelled
observed_gene_sets	Gene sets observed from your dataset
block_predictor	Block of predictor matrices to update
matrix	To iteratively update with block_predictor values
measure	The accuracy measure used for validation. Default is "F1".
X.matrix	Predictor matrix.
Y.matrix	Response matrix.
validation_index	Index of the validation sample.
j	Block to return indices for
...	Other parameters of test_func
results_CV_summary_n	Passed from asmbPLSDA.cv.loo
F_matrix_validation_bind	Passed from asmbPLSDA.cv.loo
PLS_term	Passed from asmbPLSDA.cv.loo
X.dim	Vector with number of genes of each block
quantile.comb.table	Passed from asmbPLSDA.cv.loo
outcome.type	Passed from asmbPLSDA.cv.loo
quantile_table_CV	Passed from asmbPLSDA.cv.loo
K	Number of samples.
n_quantile_comb	Passed from asmbPLSDA.cv.loo
Method	The decision rule for prediction (e.g., "fixed_cutoff", "Euclidean_distance_X", etc.).
expected.measure.increase	Passed from asmbPLSDA.cv.loo
center	Passed from asmbPLSDA.cv.loo
scale	Passed from asmbPLSDA.cv.loo
maxiter	The maximum number of iterations for validation tests. Default is 100.

E_matrix_training
: Training predictor matrix.

F_matrix_training
: Training response matrix.

E_matrix_validation
: Validation predictor matrix.

F_matrix_validation
: Validation response matrix

i
: Passed from asmbPLSDA.cv.loo

BPPARAM
: A BiocParallel::bpparam() with parallelization options

measure_selected
: Passed from asmbPLSDA.cv.loo

npermut
: The number of permutations for significance testing.

q
: Number of classes.

nr
: Number of samples

nbObsPermut
: The number of samples to permute in each permutation. Default is NULL.

res
: List of results to store statistics

Ypermut
: Permuted response matrix.

object
: A superpathway input list containing the data to be used for the cross-validation.

Nc
: Number of samples to drop at each permutation.

X_train
: Training predictor blocks

Y_train
: Training response matrix.

Modelpermut
: Permuted asmbPLSDA model

X_val
: Validation predictor blocks

s
: Validation samples

null_errors
: A vector of errors from the null distribution (permuted errors).

CV_error
: The observed cross-validation error.

m
: A vector of errors from the null distribution (permuted errors).

M
: Number of classes.

nsubsampling
: The number of subsamples for CIP/GIP testing. Default is 100.

variability
: A list of CIP or GIP values for observed distributions.

null_dist
: A list of CIP or GIP values for null distributions.

test_func
: The test function to use (typically Wilcoxon).

model_block_matrices
: A list containing the model block matrices (predictor and response matrices).

nFC
: The number of folds for K-fold cross-validation. If nFC == 1, LOOCV is performed.

parallel
: A logical value indicating whether parallel computation should be used.

expected_measure_increase
: Expected decrease in measure per additional PLS component. Default is 0.005.

`output` The superpathway fit model list that contains the fitted model and validation information.

`optimal_hyperparameters`
The optimal hyperparameters obtained from cross-validation.

`global_significance_full`
Boolean flag indicating whether to return full global significance results.

`CIP.GIP_significance_full`
Boolean flag indicating whether to return full CIP/GIP significance results.

`type` The procedure type for generating CIP/GIP distributions. Can be "jackknife" or "subsampling".

`gene_set` A parameter passed from orthology_mapping

`mart` A parameter passed from orthology_mapping

`annotation` A parameter passed from orthology_mapping, it indicates the annotation of the gene set provided

`from_species` A parameter passed from orthology_mapping

`to_species` A parameter passed from orthology_mapping

`model_object` A superpathway fit model list

`b` A parameter passed from singIST_treat. The index of current iteration block.

`samples` A parameter passed from singIST_treat. The samples to modify its gene expression from `predictor_block`

`predictor_block`
A parameter passed from singIST_treat. The predictor block of matrices from asmbPLSDA to modify its gene expression.

`FC` A parameter passed from singIST_treat. A `data.frame` with the Fold Changes, for a cell type, of each gene.

`data` Matrix of predictor block to deflate

`fit_asmb` asmbPLSDA fitted model

`PLS` Numeric value indicating the PLS component

`delta_cbind` Gene contributions (loadings) used to deflate the blocks

Value

A list containing the training and validation sets:

`E_matrix_validation`
Validation predictor matrix

`F_matrix_validation`
Validation response matrix

`E_matrix_training`
Training predictor matrix

`F_matrix_training`
Training response matrix

A numeric vector containing predicted values for validation samples.

A list containing updated `results_CV_summary_n` and `F_matrix_validation_bind` matrices.

A vector with the performance measure of each quantile combination

Optimal quantile table for each PLS with all its performance measures

An integer with the optimal number of PLS

A list with the true class of each LOOCV sample and its predicted class for each quantile combination

A list containing initialized data frames for permutation statistics.

A permuted response matrix.

Updated result list with permutation statistics.

A vector of selected sample indices.

The fitted asmbPLS-DA model.

Res list including the performance measure of the permuted model

The computed p-value.

A numeric vector containing the lower and upper bounds of the 95% confidence interval.

A list with the observed CIP and GIP distributions.

A list with the observed CIP and GIP distributions.

A list with the null CIP and GIP distributions.

A permuted X matrix.

A data frame of p-values.

A list containing the optimal hyperparameters and associated quantile table.

The updated `superpathway.fit.model` object with the computed validation metrics.

The identified gene annotation or NULL if it was not identified

A `data.table` object with the Ensembl identifiers of gene set for from_species and to_species with only one to one orthologs

The predictor block matrix updated with the FC translation

The object `data` centered and scaled.

A vector with the indices of the predictor block matrix for the requestes block

The `data` matrix loading deflated

Examples

```
measure <- "F1"
get_measure_index(measure)
X <- matrix(rnorm(100), nrow = 10, ncol = 10)
Y <- matrix(sample(0:1, 10, replace = TRUE), ncol = 1)
result <- get_train_val_sets(X, Y, validation_index = 2)
str(result)
E_train <- matrix(rnorm(100), nrow = 10, ncol = 10)
F_train <- matrix(sample(0:1, 10, replace = TRUE), ncol = 1)
E_valid <- matrix(rnorm(10), nrow = 1, ncol = 10)
```

```
F_valid <- matrix(1, nrow = 1, ncol = 1)
quantile_table <- matrix(runif(2), nrow = 1, ncol = 2)
quantile_table_CV <- matrix(runif(7), nrow = 1, ncol = 7)
results_CV_summary_n <- matrix(0, nrow = 1, ncol = 2)
F_matrix_validation_bind <- matrix(0, nrow = 1, ncol = 2)
result <- evaluate_quantile_combinations(j=1, E_matrix_training = E_train,
                                         F_matrix_training = F_train,
                                         E_matrix_validation = E_valid,
                                         F_matrix_validation = F_valid,
                                         F_matrix_validation_bind =
                                         F_matrix_validation_bind,
                                         results_CV_summary_n =
                                         results_CV_summary_n,
                                         quantile_table_CV=quantile_table_CV,
                                         i = 1, X.dim = c(5,5),
                                         quantile.comb.table =quantile_table,
                                         outcome.type = "binary",
                                         center = TRUE,
                                         scale = TRUE, maxiter = 100,
                                         Method = NULL)
print(result)
set.seed(123)
K <- 5
X <- matrix(rnorm(50), nrow = 5, ncol = 10)
Y <- matrix(sample(0:1, 5, replace = TRUE), ncol = 1)
quantile_comb_table <- matrix(runif(10), nrow = 2, ncol = 10)
results_CV_summary_n <- matrix(0, nrow = 2, ncol = K)
F_matrix_validation_bind <- matrix(0, nrow = 2, ncol = K)
# Parallelization options
library(BiocParallel)
register(SnowParam(workers = 2, exportglobals = FALSE, progressbar = TRUE),
default = TRUE)
output <- execute_parallel_cv(K, results_CV_summary_n,
                              F_matrix_validation_bind, X, Y, PLS_term = 1,
                              X.dim = c(5,5),
                              quantile.comb.table = quantile_comb_table,
                              outcome.type = "binary",
                              quantile_table_CV = quantile_comb_table,
                              measure = "B_accuracy",
                              expected.measure.increase = 0.005,
                              center = TRUE, scale = TRUE, maxiter = 100,
                              Method = NULL)
register(SerialParam(), default = TRUE) # disable parallelization
str(output)
initialize_results(100, 3)
permute_Y_matrix(matrix(rnorm(100), 10, 10), nr = 10, nbObsPermut = 3, j = 2)
res <- initialize_results(100, 3)
compute_permutation_stats(res, matrix(rnorm(100), 10, 10),
matrix(rnorm(100), 10, 10), j = 2, q = 3, nr = 10)
null_errors <- c(0.3, 0.4, 0.35, 0.33)
CV_error <- 0.32
compute_pvalue(null_errors, CV_error)
null_errors <- c(0.3, 0.4, 0.35, 0.33)
```

```
compute_IC95(null_errors)
## Not run:
library(biomaRt)
gene_set <- c("IL13", "IL4", "IL5", "IL21")
mart <- biomaRt::useEnsembl(biomart = "genes",
dataset = "hsapiens_gene_ensembl")
detect_gene_type(gene_set, mart)

## End(Not run)
## This helper requires a live Ensembl connection. For a fully offline
## workflow, call orthology_mapping() with the shipped cache instead
## (see ?orthology_mapping, argument orthology_cache).
## Not run:
annotation <- "external_gene_name"
gene_set <- c("IL13", "IL4", "IL5")
mart <- biomaRt::useEnsembl(biomart = "genes",
dataset = "hsapiens_gene_ensembl")
retrieve_one2one_orthologs(annotation, gene_set, mart, "hsapiens",
"mmusculus")

## End(Not run)
## This helper requires a live Ensembl connection. For a fully offline
## workflow, call orthology_mapping() with the shipped cache instead
## (see ?orthology_mapping, argument orthology_cache).
X.dim <- c(30,40,60)
j <- 2
get_indices(j, X.dim)
```

`matrixToBlock` *Build predictor and response blocks with superpathway input*

Description

Builds the predictor block and matrix response for its fit in asmbPLS-DA

Usage

```
matrixToBlock(object)
```

Arguments

`object`	A superpathway input list object

Value

A list containing the predictor block, response matrix, dimension of each block and observed gene sets with respect to gene_sets_celltype for the original pseudobulk_lognorm matrix, for its use in asmbPLS-DA fit

Examples

```
file <- system.file("extdata", "example_superpathway_input.rda",
package = "singIST")
load(file)
data <- example_superpathway_input
matrixToBlock(data)
```

`multiple_check` *Check if parameter format is consistent*

Description

For the wrapper functions multiple_fitOptimal and multiple_singISTrecapitulations one must pass multiple parameters. To check for the consistency of such parameters we use this function with the logic; if the parameter passed is `NULL` or its length is 1, it is assumed that the desired list of parameters is the repetition of such; if the parameter passed is a vector whose length is the number of objects that the wrappers iterate on, then the function returns a list whose elements are each of the vector elements; otherwise if the parameters are a vector whose length does not match with the number of objects to iterate on then the function stops

Usage

```
multiple_check(parameter, objectLength)
```

Arguments

`parameter`	The parameters passed from either multiple_fitOptimal or multiple_singISTrecapitulations
`objectLength`	The number of objects that the wrapper functions iterate on

Value

A list with the repetition of the parameter

Examples

```
# NULL parameter case
parameter <- NULL
objectLength <- 10
multiple_check(parameter, objectLength)

# Parameter equal for all elements
parameter <- FALSE
objectLength <- 5
multiple_check(parameter, objectLength)

# Parameter differing for all elements
parameter <- c(1, 6, 7, 8, 9)
objectLength <- 5
multiple_check(parameter, objectLength)
```

`multiple_fitOptimal` *Multiple Cross validation and fit of asmbPLSDA*

Description

Use fitOptimal for multiple superpathway input list objects. This wrapper is useful if one wants to assess multiple superpathways for analyses and needs to train its respective optimal models.

Usage

```
multiple_fitOptimal(
  object = list(),
  parallel = c(FALSE),
  measure = c("B_accuracy"),
  expected_measure_increase = c(0.005),
  maxiter = c(100),
  global_significance_full = c(FALSE),
  CIP.GIP_significance_full = c(FALSE),
  npermut = c(100),
  nbObsPermut = c(NULL),
  type = c("jackknife"),
  nsubsampling = c(100),
  Method = c(NULL),
  ...
)
```

Arguments

`object`	A list whose elements are superpathway input lists objects to use fitOptimal
`parallel`	A vector whose elements are `parallel` parameters for each object as requested by fitOptimal
`measure`	A vector whose elements are `measure` parameters for each object as requested by fitOptimal
`expected_measure_increase`	A vector whose elements are `expected_measure_increase` parameters for each object as requested by fitOptimal
`maxiter`	A vector whose elements are `maxiter` parameters for each object as requested by fitOptimal
`global_significance_full`	A vector whose elements are `global_significance_full` parameters for each object as requested by fitOptimal
`CIP.GIP_significance_full`	A vector whose elements are `CIP.GIP_significance_full` parameters for each object as requested by fitOptimal

npermut	A vector whose elements are npermut parameters for each object as requested by fitOptimal
nbObsPermut	A vector whose elements are nbObsPermut parameters for each object as requested by fitOptimal
type	A vector whose elements are type parameters for each object as requested by fitOptimal
nsubsampling	A vector whose elements are nsubsampling parameters for each object as requested by fitOptimal
Method	A vector whose elements are Method parameters for each object as requested by fitOptimal
...	Other parameters to be passed onto fitOptimal

Value

A list of superpathway fit model list

Examples

```
file <- system.file("extdata", "example_superpathway_input.rda",
package = "singIST")
load(file)
data <- example_superpathway_input
models <- list(data, data)
# Example with different options
multiple_model <- multiple_fitOptimal(models, type = c("jackknife",
"subsampling"), nsubsampling = c(NULL, 10), npermut = c(10,15))
```

---

multiple_singISTrecapitulations

Compute singIST recapitulations for multiple superpathways

Description

Use singISTrecapitulations for multiple superpathway fit model list against the same mapping organism list. This wrapper is useful if one wants to assess multiple superpathways against the same mapping organism list.

Usage

```
multiple_singISTrecapitulations(
  object,
  model_object = list(),
  model_species = list("hsapiens"),
  orthology_cache = NULL,
  ...
)
```

Arguments

object	A mapping organism list
model_object	A list whose elements are superpathway fit model list
model_species	A list of characters indicating the organism of each model_object element. By default list("hsapiens") which assumes the same organism across all elements of model_object parameter
orthology_cache	An optional data.frame with a precomputed one-to-one orthology mapping, forwarded to singISTrecapitulations for every superpathway in model_object. When provided, no Ensembl BioMart queries are performed, making the workflow reproducible offline. See orthology_mapping for the required format. By default orthology_cache = NULL.
...	Other parameters to pass onto biological_link_function

Value

A list with the row binded data.frame for each superpathway assessed for the superpathway and cell type recapitulations, and gene contributions to the former.

Examples

```
file <- system.file("extdata", "example_superpathway_input.rda",
package = "singIST")
load(file)
data_model <- example_superpathway_input
models <- list(data_model, data_model)
# Example with different options
multiple_model <- multiple_fitOptimal(models, type = c("jackknife",
"subsampling"), nsubsampling = c(NULL, 10), npermut = c(10,15))
file <- system.file("extdata", "example_mapping_organism.rda",
package = "singIST")
load(file)
data_organism <- example_mapping_organism
multiple_singISTrecapitulations(data_organism, multiple_model,
model_species = list("hsapiens", "hsapiens"))
```

---

orthology_mapping *Orthology mapping*

Description

Performs the one-to-one orthology mapping between the mapped disease model list to the reference (human) organism of the superpathway fit model list.

Usage

```
orthology_mapping(
  object,
  model_object,
  from_species,
  to_species = "mmusculus",
  annotation_to_species = "external_gene_name",
  orthology_cache = NULL
)
```

Arguments

object	A mapping organism list
model_object	A superpathway fit model list
from_species	A character indicating the reference organism for which the parameter model_fit has information from.
to_species	A character indicating the mapped organism for which the parameter object has information from. By default mmusculus.
annotation_to_species	A character indicating the gene identifier annotation used for the to_species. Note this should match with the gene names in object$counts. By default external_gene_name. If NULL the annotation_to_species is inferred with detect_gene_type, note this might take time.
orthology_cache	An optional data.frame with a precomputed one-to-one orthology mapping, used instead of querying Ensembl BioMart. It must contain at least the columns input_gene (gene identifier in the reference organism of model_object, e.g. human gene symbol) and output_gene (gene identifier in the mapped organism, matching the gene names in object$counts). This makes the workflow reproducible offline and robust to Ensembl service interruptions. An example human-to-mouse cache is shipped in system.file("extdata", "example_orthology_mapping.rda", package = "singIST"). By default orthology_cache = NULL and Ensembl BioMart is queried.

Value

A list with the gene sets per cell type with the one-to-one orthology

Examples

```
# Case without stating the gene annotation of the mapping.organisms object
# note this will take longer to execute
file <- system.file("extdata", "example_mapping_organism.rda",
package = "singIST")
load(file)
data_organism <- example_mapping_organism
file <- system.file("extdata", "example_superpathway_fit_model.rda",
package = "singIST")
```

```
load(file)
data_model <- example_superpathway_fit_model
orthology_mapping(data_organism, data_model, "hsapiens",
annotation_to_species = NULL)
# Case assuming the gene annotation of the mapping.organism object is
# by default "external_gene_name" this is faster
orthology_mapping(data_organism, data_model, "hsapiens")
```

`permut_asmbplsda` *Permutation test for asmbPLSDA global significance for LOO*

Description

Performs permutation testing for asmbPLS-DA to evaluate model validity.

Usage

```
permut_asmbplsda(
  object,
  npermut = 100,
  nbObsPermut = NULL,
  Nc = 1,
  CV_error,
  measure = "B_accuracy",
  Method = NULL,
  maxiter = 100
)
```

Arguments

`object`	A superpathway fit model list.
`npermut`	Number of permutations (default: 100).
`nbObsPermut`	Number of samples to permute per iteration (default: NULL).
`Nc`	Number of samples dropped per permutation (default: 1).
`CV_error`	Cross-validation error of the fitted model.
`measure`	Accuracy measure (`"F1"`, `"accuracy"`, `"B_accuracy"`, `"precision"`, `"recall"`, default: `"B_accuracy"`).
`Method`	Decision rule for prediction (default: NULL).
`maxiter`	Maximum iterations (default: 100).

Value

A list with permutation statistics, p-value, and confidence intervals.

Examples

```
file <- system.file("extdata", "example_superpathway_fit_model.rda",
package = "singIST")
load(file)
data <- example_superpathway_fit_model
permut_asmbplsda(data, npermut = 5, Nc = 1,
CV_error = 1)
```

---

permut_asmbplsda_kcv *Permutation test for asmbPLS-DA global significance (LOO or KCV)*

Description

If splits=NULL, runs LOOCV-based permutation. Otherwise treats splits as a list of train/validate splits (e.g. from make_splits_R()) and does a fixed splits K-fold×repeats permutation test.

Usage

```
permut_asmbplsda_kcv(
  object,
  npermut = 100,
  splits = NULL,
  measure = "B_accuracy",
  nbObsPermut = NULL,
  Nc = 1,
  Method = NULL,
  maxiter = 100,
  CV_error = NULL,
  ...
)
```

Arguments

object	A superpathway fit model list
npermut	Number of permutations (default: 100)
splits	Optional list of splits; if NULL uses LOOCV branch
measure	Accuracy measure ("F1", "accuracy", "B_accuracy", "precision", "recall", default: "B_accuracy").
nbObsPermut	Number of samples to permute per iteration (default: NULL).
Nc	Number of samples dropped per permutation (default: 1 if LOOCV).
Method	Decision rule for prediction (default: NULL).
maxiter	Maximum iterations (default: 100).
CV_error	Error obtained from optimal model CV process
...	Other args passed to LOOCV or to evaluate_performance

Value

A list with null distribution, p-value, and (for KCV) splits

`predict_mfa_imputer` *Impute new samples using a fitted MFA imputer*

Description

Projects each new sample into the latent space learned on training, then reconstructs its missing entries.

Usage

```
predict_mfa_imputer(X_new, mu, loadings)
```

Arguments

`X_new`	Numeric matrix (new samples × same features), may contain NAs.
`mu`	Numeric vector of column means (as returned by `fit_mfa_imputer`).
`loadings`	Numeric matrix of loadings (columns = components).

Value

Matrix `X_new` with NAs replaced by reconstructed values.

`pullGeneSet` *Pull Gene Set from MsigDB*

Description

Retrieves the gene set associated with a pathway or superpathway from MsigDB.

Usage

```
pullGeneSet(object, gse = NULL, ...)
```

Arguments

`object`	List. A pathway (from create_pathway()) or superpathway (from create_superpathway()).
`gse`	Gene Set Collection from MsigDB. If NULL, loads default (human, ENTREZ + SYM IDs).
`...`	Additional arguments passed to msigdb::subsetCollection().

Value

A character vector with gene IDs for the specified pathway.

Examples

```
my_pathway <- create_pathway("KEGG_CYTOKINE_CYTOKINE_RECEPTOR_INTERACTION",
"KEGG", "c2", "CP")
pullGeneSet(my_pathway)
```

`render_multiple_outputs` *Render multiple singISTrecapitulation outputs*

Description

Render output of multiple_singISTrecapitulations for multiple disease models and superpathways, the output is friendly for visualizing the results

Usage

```
render_multiple_outputs(objects = list())
```

Arguments

`objects`	A list as retuned by multiple_singISTrecapitulations

Value

A list with the row binded `data.frame` for each superpathway assessed for the superpathway and cell type recapitulations, gene contributions to the former, and fold changes. These row binds are performed for all disease models and superpathways.

`restore_removed_columns` *Restore columns removed during MFA cleaning into the imputed matrix*

Description

After cleaning and imputing a subset of predictors (removing any columns with zero variance or 100% missing), this function re-inserts those removed columns in their original order. For each re-inserted column, original non-missing values are kept and entries that were originally missing are set to zero.

Usage

```
restore_removed_columns(X_imp_clean, X_raw, keep_cols)
```

Arguments

X_imp_clean	Numeric matrix (n_samples × p_cleaned) of imputed values for the columns that were kept.
X_raw	Numeric matrix or data.frame (n_samples × p_oril) of ginathe original predictor data before cleaning.
keep_cols	Logical vector of length p_original, where TRUE indicates the column was kept for imputation and FALSE indicates it was removed.

Value

A numeric matrix (n_samples × p_original) with all original columns in their original order. Columns where keep_cols == TRUE contain the values from X_imp_clean. Columns where keep_cols == FALSE contain the original non-NA values, and any entries that were originally NA are set to zero.

Results_comparison_measure

Compute performance metrics of predicted asmbPLSDA

Description

Compute performance metrics of predicted asmbPLSDA

Usage

```
Results_comparison_measure(
  Y_predict,
  Y_true,
  outcome.type = c("binary", "multiclass")
)
```

Arguments

Y_predict	Predicted matrix from asmbPLSDA
Y_true	True class used to fit asmbPLSDA
outcome.type	Outcome type either "binary" or "multiclass"

Value

A vector with accuracy, balanced accuracy, precision, recall and F1 metric

Examples

```
Results_comparison_measure(c(1,0,1,0,1), c(0,0,1,1,1),
outcome.type = "binary")
```

`setGeneSetsCelltype` *Set gene sets per cell type in a superpathway*

Description

Updates the gene_sets_celltype element of a superpathway object, ensuring validity:

- The number of gene sets must match the number of cell types.

Usage

```
setGeneSetsCelltype(object, value = NULL, ...)
```

Arguments

`object`	List. A superpathway object created by create_superpathway(), or a pathway object created by create_pathway().
`value`	List. A list of genes to incorporate in each cell type slot. By default `NULL`, only use if genes are to be introduced manually.
`...`	other parameters to pass onto pullGeneSet().

Value

The updated superpathway object (list).

Examples

```
my_pathway <- create_pathway(
  standard_name = "KEGG_CYTOKINE_CYTOKINE_RECEPTOR_INTERACTION",
  dbsource = "KEGG",
  collection = "c2",
  subcollection = "CP"
)

my_superpathway <- create_superpathway(my_pathway, c("T-cell",
"Dendritic Cell"), list())

my_superpathway <- setGeneSetsCelltype(my_superpathway, list(c("IL2"),
c("IL4")))
```

singIST_treat *Derive singIST treated samples*

Description

Derive singIST treated samples

Usage

```
singIST_treat(object, model_object, orthologs, logFC)
```

Arguments

object	A mapping organism list passed from biological_link_function.
model_object	A superpathway fit model list passed from biological_link_function
orthologs	A list of data.table objects, as returned by orthology_mapping with the one-to-one orthologs of each gene set per cell type
logFC	A list of data.frame objects, as returned by diff_expressed with the logFC for each gene and cell type.

Value

A list object with the singIST treated samples predictor block matrix and a list of Fold Changes for each cell type used to compute the singIST treated samples.

Examples

```
# Orthology mapping
file <- system.file("extdata", "example_mapping_organism.rda",
package = "singIST")
load(file)
data_organism <- example_mapping_organism
file <- system.file("extdata", "example_superpathway_fit_model.rda",
package = "singIST")
load(file)
data_model <- example_superpathway_fit_model
orthologs <- orthology_mapping(data_organism, data_model, "hsapiens")
# Set the identities
# Cell type mapping
data <- celltype_mapping(data_organism)
data$counts$test <- paste0(data$counts$celltype_cluster,
"_", data$counts$class)
SeuratObject::Idents(data$counts) <- "test"
logFC <- diff_expressed(data)
singIST_treat(data_organism, data_model, orthologs, logFC)
```

`singISTrecapitulations` *Compute singIST recapitulations*

Description

This method provides with all singIST recapitulations; superpathway recapitulations; cell type recapitulations; gene contributions to cell type recapitulations. The procedure encompasses the execution of the biological link function, the derivation of the predictor scores (superpathway, cell type and gene scores), and their use to compute the predicted recapitulations as a fraction of the reference recapitulation.

Usage

```
singISTrecapitulations(object, model_object, orthology_cache = NULL, ...)
```

Arguments

`object`	A mapping organism list for which to calculate the recapitulations against the fitted superpathway model
`model_object`	A superpathway fit model list used to calculate the recapitulations
`orthology_cache`	An optional `data.frame` with a precomputed one-to-one orthology mapping, forwarded to biological_link_function. When provided, no Ensembl BioMart queries are performed, making the workflow reproducible offline. See orthology_mapping for the required format. By default `orthology_cache = NULL`.
`...`	Other parameters to pass onto biological_link_function

Value

A list with; a `data.frame` object with the superpathway recapitulation, containing variables `pathway` name, `recapitulation`, `p_val` with the global significance test of the fitted model as provided in `model_object`, and `target_organism` with the target class of the disease model as provided in `model_object`; a `data.frame` with the cell type recapitulation, containing variables `pathway` name, `celltype`, `recapitulation`, `orthology` with the percentage of observed one-to-one orthology coverage

- if all cell types have the same gene set this value is constant -, and `target_organism`; a `data.frame` object with the gene contributions to cell type recapitulation, containing variables `pathway`, `celltype`, `gene` name, `contribution` indicating the gene contribution to cell type recapitulation, and `target_organism`; `orthologs` a `data.frame` with the one-to-one orthology mapping for each cell type gene set.

Examples

```
file <- system.file("extdata", "example_mapping_organism.rda",
package = "singIST")
load(file)
data_organism <- example_mapping_organism
file <- system.file("extdata", "example_superpathway_fit_model.rda",
package = "singIST")
load(file)
data_model <- example_superpathway_fit_model
singISTrecapitulations(data_organism, data_model)
```

---

superpathway_recap *Derive superpathway recapitulation*

Description

Derive superpathway recapitulation

Usage

```
superpathway_recap(model_object, data_original, data_singIST)
```

Arguments

model_object	A superpathway fit model list passed from singISTrecapitulations
data_original	A matrix with the superpathway's score as returned by derive_contributions for the non-singIST treated samples, passed from singISTrecapitulations
data_singIST	A matrix with the superpathway's score as returned by derive_contributions for the singIST treated samples, passed from singISTrecapitulations

Value

An object `data.frame` with the variables: `pathway` name as indicated in `model_object`, `recapitulation` with the superpathway recapitulation

Examples

```
file <- system.file("extdata", "example_superpathway_fit_model.rda",
package = "singIST")
load(file)
model <- example_superpathway_fit_model
file <- system.file("extdata", "example_mapping_organism.rda",
package = "singIST")
load(file)
mapped <- example_mapping_organism
singIST_samples <- biological_link_function(mapped,
model)$singIST_samples
original <- derive_contributions(model, singIST_samples)
```

```
derived <- derive_contributions(model,model$model_fit$predictor_block)
superpathway_recap(model, original$superpathway_score,
derived$superpathway_score)
```

`update_group_sizes` *Update block-size vector after cleaning*

Description

Given an original vector of block lengths and a mask of kept columns, recomputes the new block lengths.

Usage

```
update_group_sizes(group_orig, keep_cols)
```

Arguments

`group_orig`	Integer vector of original block sizes (sum group_orig = ncol before cleaning).
`keep_cols`	Logical vector of length `sum(group_orig)`, `TRUE` for columns retained.

Value

Integer vector of same length as `group_orig` with updated block sizes.

Examples

```
update_group_sizes(c(2,3), c(TRUE, TRUE, FALSE, TRUE, TRUE))
```

`wilcox_CIP_GIP` *Mann-Whitney Wilcoxon test p-value*

Description

Mann-Whitney Wilcoxon test p-value

Usage

```
wilcox_CIP_GIP(ref_distr, null_distr, ...)
```

Arguments

`ref_distr`	A vector with the reference distribution
`null_distr`	A vector with the null distribution
`...`	Other parameters to be passed onto wilcox.test

Value

A pvalue with the Mann-Whitney Wilcoxon test with the "greater" as the alternative hypothesis

Examples

```
ref_distr <- rnorm(100, mean = 30, sd = 2)
null_distr <- rnorm(100, mean = 0, sd = 1)
wilcox_CIP_GIP(ref_distr, null_distr)
```

Index

add_missing_psb_rows, 3
asmbPLSDA.cv.kcv, 3
asmbPLSDA.cv.loo, 5, *38, 39*

biological_link_function, 6, *47*, *55, 56*

calculate_pvalues *(*helpers*)*, 32
celltype_mapping, 8
celltype_recap, 9, *31*
center_scale *(*helpers*)*, 32
check_fit_model, 10
check_hyperparameters, 10
check_mapping_organism, 11
check_pathway, 12
check_superpathway, 13
check_superpathway_input, 14, *23*
CIP_GIP, 15
CIP_GIP_test, 16, *30*
clean_mfa_data, 17
compute_final_measures *(*helpers*)*, 32
compute_IC95 *(*helpers*)*, 32
compute_permutation_stats *(*helpers*)*, 32
compute_pvalue *(*helpers*)*, 32
compute_validation_metrics *(*helpers*)*, 32
create_fit_model, 18
create_hyperparameters, 19
create_mapping_organism, 20
create_pathway, 21
create_superpathway, 22
create_superpathway_input, 23

deflate_prediction *(*helpers*)*, 32
derive_contributions, *9*, 25, *26*, *31*, *57*
derive_scores, 26
detect_gene_type, *48*
detect_gene_type *(*helpers*)*, 32
diff_expressed, *7*, 27, *55*

evaluate_performance *(*helpers*)*, 32
evaluate_quantile_combinations *(*helpers*)*, 32
execute_parallel_cv *(*helpers*)*, 32
execute_sequential_cv *(*helpers*)*, 32

FCtoExpression *(*helpers*)*, 32
fit_mfa_imputer, 28
fit_permuted_model *(*helpers*)*, 32
fitOptimal, 29, *45, 46*

gene_contrib, 31
generate_null_distributions *(*helpers*)*, 32
get_indices *(*helpers*)*, 32
get_measure_index *(*helpers*)*, 32
get_train_val_sets *(*helpers*)*, 32

helpers, 32

initialize_results *(*helpers*)*, 32

jackknife_CIP_GIP *(*helpers*)*, 32

matrixToBlock, 43
multiple_check, 44
multiple_fitOptimal, *44*, 45
multiple_singISTrecapitulations, *44*, 46, *52*

orthology_mapping, *7*, *33*, *40*, 47, *47*, *55, 56*

perform_cv *(*helpers*)*, 32
performance_measures *(*helpers*)*, 32
permut_asmbplsda, *30*, 49
permut_asmbplsda_kcv, 50
permute_X_matrix *(*helpers*)*, 32
permute_Y_matrix *(*helpers*)*, 32
predict_mfa_imputer, 51
pullGeneSet, 51

quantile_computation *(*helpers*)*, 32

render_multiple_outputs, 52

restore_removed_columns, 52
Results_comparison_measure, 53
retrieve_one2one_orthologs *(*helpers*)*, 32

select_optimal_PLS *(*helpers*)*, 32
select_samples *(*helpers*)*, 32
setGeneSetsCelltype, 54
singIST_treat, *7*, *40*, 55
singISTrecapitulations, *9*, *31*, *46*, *47*, 56, *57*
subsampling_CIP_GIP *(*helpers*)*, 32
superpathway_recap, 57

update_block *(*helpers*)*, 32
update_group_sizes, 58

wilcox_CIP_GIP, *17*, *30*, 58