Package: proteoDeconv (via r-universe)

March 14, 2025

Title Enabling Cell-Type Deconvolution of Proteomics Data

Version 0.1.0.9000

Description Tools for deconvoluting proteomics data to identify and quantify cell types (e.g., immune cell types) in complex biological samples.

License MIT + file LICENSE

URL https://github.com/ComputationalProteomics/proteoDeconv, https://computationalproteomics.github.io/proteoDeconv/

BugReports https://github.com/ComputationalProteomics/proteoDeconv/issues

Depends R (>= 4.1.0)

Imports dplyr, glue, HGNChelper, impute, Matrix, MsCoreUtils, readr, rlang, SimBu, stats, stringr, SummarizedExperiment, tibble, tidyr, tidyselect, utils, uuid, withr

Suggests testthat (>= 3.0.0), BayesDeBulk, EPIC, knitr, rmarkdown, limma, ggplot2

Remotes WangLab-MSSM/BayesDeBulk/BayesDeBulk, GfellerLab/EPIC, omnideconv/SimBu

Config/testthat/edition 3

Encoding UTF-8

SystemRequirements Docker

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

VignetteBuilder knitr

Config/pak/sysreqs libglpk-dev make libicu-dev libpng-dev libxml2-dev libssl-dev python3 libx11-dev

Repository https://computationalproteomics.r-universe.dev

RemoteUrl https://github.com/ComputationalProteomics/proteoDeconv

RemoteRef HEAD

RemoteSha bf2bb1567625dc03af34a2237b26977ede837177

Contents

`convert_to_tpm` *Convert protein abundance data to TPM-like normalization*

Description

Normalizes protein abundance data using a TPM-like approach (Transcripts Per Million), adapting this RNA-seq normalization method for use with proteomics data.

Usage

```
convert_to_tpm(data)
```

Arguments

`data`	A numeric matrix containing protein abundance data with identifiers as row names and samples as columns. Values should be in linear scale, not log-transformed.

Details

This function applies a TPM-like normalization to proteomics data, where each protein abundance value is scaled by the total abundance in the sample and multiplied by 1 million.

If your input data is log-transformed, use `unlog2_data` first to convert it to linear scale before applying this normalization.

Value

A numeric matrix with the same dimensions as the input, with values normalized to a TPM-like scale (sum of each column equals 1 million).

Examples

```
# Create example protein abundance data matrix
prot_mat <- matrix(abs(rnorm(12, mean = 500, sd = 200)), nrow = 4, ncol = 3)
rownames(prot_mat) <- paste0("Protein", 1:4)
colnames(prot_mat) <- paste0("Sample", 1:3)

# View original values and column sums
print(prot_mat)
print(colSums(prot_mat))

# Convert to TPM-like normalization
tpm_mat <- convert_to_tpm(prot_mat)

# Verify that column sums equal 1 million
print(tpm_mat)
print(colSums(tpm_mat))
```

---

create_phenoclasses *Create phenoclasses matrix for cell type deconvolution*

Description

Generates a phenotype classification matrix from sample data for use in cell type deconvolution workflows. This matrix maps samples to their respective cell types.

Usage

```
create_phenoclasses(
  data,
  mapping_rules,
  verbose = FALSE,
  return_format = c("tibble", "matrix")
)
```

Arguments

data	A numeric matrix of pure cell type profiles with genes as row names and samples as columns. Column names should contain identifiers that can be mapped to cell types.

mapping_rules	Either: 1. A named list where names are cell type labels and values are regular expression patterns used to match sample column names (e.g., list("CD8+ T cells" = "CD8", "Monocytes" = "Mono")), OR 2. A character vector with the same length and order as colnames(data), directly specifying the cell type for each sample.
verbose	Logical. If TRUE, displays additional messages during processing.
return_format	A string specifying the return format: "matrix" or "tibble" (default).

Details

The function is particularly useful for preparing reference data for signature matrix generation using CIBERSORTx. Each cell in the output is encoded as:

- 0 for 'Unknown' mappings
- 1 if the sample belongs to the cell type in that row
- 2 if the sample belongs to a different cell type

Value

If return_format is "matrix", a numeric matrix with cell type groups as rows and samples as columns. If return_format is "tibble", a tibble with a "cell_type" column and columns for each sample.

See Also

create_signature_matrix which uses the phenoclasses matrix to generate a cell type signature matrix.

Examples

```
## Not run:
# Example using a named list of regex patterns
pure_samples <- readRDS(system.file("extdata", "pure_samples_matrix.rds",
                                    package = "proteoDeconv"))

mapping_rules <- list(
  "CD8+ T cells" = "CD8",
  "Monocytes" = "Mono"
)

phenoclasses1 <- create_phenoclasses(
  data = pure_samples,
  mapping_rules = mapping_rules,
  verbose = TRUE
)

# Example using a character vector of direct cell type assignments
cell_types <- c("CD8+ T cells", "CD8+ T cells", "Monocytes", "Monocytes", "Unknown")
phenoclasses2 <- create_phenoclasses(
  data = pure_samples,
  mapping_rules = cell_types,
```

```
  verbose = TRUE
)

## End(Not run)
```

`create_signature_matrix` *Create signature matrix using CIBERSORTx*

Description

Generates a signature matrix from reference profiles and cell type classes using the CIBERSORTx Docker image. This signature matrix contains cell type-specific marker proteins that will be used for deconvolution.

Usage

```
create_signature_matrix(
  refsample = NULL,
  phenoclasses = NULL,
  g_min = 200,
  g_max = 400,
  q_value = 0.01,
  replicates = 5,
  sampling = 0.5,
  fraction = 0.75,
  filter = FALSE,
  verbose = FALSE,
  QN = FALSE,
  single_cell = FALSE,
  use_sudo = FALSE,
  ...
)
```

Arguments

`refsample`	A numeric matrix containing reference profiles with genes as row names and samples as columns. This should be preprocessed data from pure cell populations.
`phenoclasses`	A numeric matrix, data frame, or tibble containing cell type classification (0/1/2). If provided as a matrix or base data frame, the cell types are assumed to be in the row names. If provided as a tibble, the cell type identifiers should be included as a column. This is typically created using the create_phenoclasses() function.
`g_min`	Minimum number of genes per cell type in the signature matrix. Default is 200.
`g_max`	Maximum number of genes per cell type in the signature matrix. Default is 400.

q_value	Q-value threshold for differential expression. Default is 0.01.
replicates	Number of replicates to use for building the reference file (only relevant when single_cell=TRUE). Default is 5.
sampling	Fraction of available gene expression profiles selected by random sampling (only relevant when single_cell=TRUE). Default is 0.5.
fraction	Fraction of cells of the same identity showing evidence of expression (only relevant when single_cell=TRUE). Default is 0.75.
filter	Logical indicating whether to remove non-hematopoietic genes. Default is FALSE.
verbose	Logical indicating whether to print detailed output. Default is FALSE.
QN	Logical indicating whether to run quantile normalization. Default is FALSE.
single_cell	Logical indicating whether to create signature from scRNA-Seq data. Default is FALSE.
use_sudo	Logical indicating whether to use sudo for Docker commands. Default is FALSE.
...	Additional arguments passed to the function.

Details

This function uses the CIBERSORTx Docker image to construct a signature matrix based on the input reference profiles and cell type classifications. The CIBERSORTx token and email must be set as environment variables either in the project directory's .Renviron file or in the user's home directory .Renviron file.

Value

A numeric matrix with genes as rows and cell types as columns, representing the expression profile of each cell type. This signature matrix can be used as input for deconvolution algorithms to estimate cell type proportions in mixed samples.

See Also

create_phenoclasses for creating the phenoclasses input and deconvolute for using the signature matrix in deconvolution.

Examples

```
## Not run:
# Load preprocessed pure samples data
pure_samples <- readRDS(system.file("extdata", "pure_samples_matrix.rds",
                                    package = "proteoDeconv"))

# Create phenoclasses for the samples
mapping_rules <- list(
  "CD8+ T cells" = "CD8",
  "Monocytes" = "Mono"
)
phenoclasses <- create_phenoclasses(
  data = pure_samples,
  mapping_rules = mapping_rules
```

```
)

# Create signature matrix
signature_matrix <- create_signature_matrix(
  refsample = pure_samples,
  phenoclasses = phenoclasses,
  g_min = 200,
  g_max = 400,
  q_value = 0.01
)

## End(Not run)
```

`deconvolute` *Deconvolute protein data using multiple algorithms*

Description

This function provides a unified interface for deconvoluting protein or gene expression data using various algorithms, making it easy to switch between different deconvolution methods.

Usage

```
deconvolute(algorithm, data, signature = NULL, ...)
```

Arguments

`algorithm`	A string specifying the deconvolution algorithm to use. Options are "cibersortx", "cibersort", "epic", and "bayesdebulk".
`data`	A numeric matrix containing protein or gene expression data with genes/proteins as row names and samples as columns.
`signature`	A numeric matrix containing the reference signature with genes/proteins as row names and cell types as columns. Required by all supported algorithms.
`...`	Additional arguments passed to the specific deconvolution functions.

Details

The function supports multiple deconvolution algorithms:

- **cibersortx**: Uses Docker-based CIBERSORTx, which requires Docker installation and credentials (see `deconvolute_cibersortx`).
- **cibersort**: Uses the R implementation of CIBERSORT, which requires sourcing the original CIBERSORT.R script (see `deconvolute_cibersort`).
- **epic**: Uses the EPIC algorithm which performs constrained least squares regression (see `deconvolute_epic`).
- **bayesdebulk**: Uses the BayesDeBulk Bayesian deconvolution algorithm (see `deconvolute_bayesdebulk`).

Value

A numeric matrix with samples as rows and cell types as columns, representing the estimated proportion of each cell type in each sample.

See Also

deconvolute_cibersortx for the CIBERSORTx Docker implementation deconvolute_cibersort for the CIBERSORT R implementation deconvolute_epic for the EPIC algorithm deconvolute_bayesdebulk for the BayesDeBulk algorithm

Examples

```
## Not run:
# Load example data
data_file <- system.file("extdata", "mixed_samples_matrix.rds", package = "proteoDeconv")
mixed_samples <- readRDS(data_file)

signature_file <- system.file("extdata", "cd8t_mono_signature_matrix.rds", package = "proteoDeconv")
signature_matrix <- readRDS(signature_file)

# Using EPIC algorithm
epic_results <- deconvolute(
  algorithm = "epic",
  data = mixed_samples,
  signature = signature_matrix,
  with_other_cells = TRUE
)

# Using BayesDeBulk algorithm
bayes_results <- deconvolute(
  algorithm = "bayesdebulk",
  data = mixed_samples,
  signature = signature_matrix,
  n_iter = 1000,
  burn_in = 100
)

## End(Not run)
```

---

deconvolute_bayesdebulk

Deconvolute bulk proteome data using BayesDeBulk

Description

Deconvolutes bulk proteome data using the BayesDeBulk algorithm to estimate cell type proportions in mixed samples based on a signature matrix.

Usage

```
deconvolute_bayesdebulk(
  data,
  signature,
  n_iter = 1000,
  burn_in = 100,
  marker_selection = "limma",
  ...
)
```

Arguments

data	A numeric matrix of bulk proteome data with gene identifiers as row names and samples as columns.
signature	A numeric matrix containing signature marker values with gene identifiers as row names and cell types as columns.
n_iter	Number of iterations for the MCMC sampling; default is 1000.
burn_in	Number of burn-in iterations to discard; default is 100.
marker_selection	The method to use for marker selection: "limma" (default), "simple", or a pre-computed marker matrix.
...	Additional arguments passed to marker selection functions.

Details

This function calculates signature markers using the specified marker selection method and runs the BayesDeBulk deconvolution algorithm. The marker selection process identifies genes that are uniquely expressed in specific cell types, which are then used for the deconvolution.

Value

A matrix containing cell type proportions with samples as rows and cell types as columns.

Examples

```
## Not run:
# Load example data and signature matrix
data_file <- system.file("extdata", "mixed_samples_matrix.rds", package = "proteoDeconv")
mixed_samples <- readRDS(data_file)

signature_file <- system.file("extdata", "cd8t_mono_signature_matrix.rds", package = "proteoDeconv")
signature_matrix <- readRDS(signature_file)

# Run deconvolution
result <- deconvolute_bayesdebulk(mixed_samples, signature_matrix)

## End(Not run)
```

`deconvolute_cibersort` *Deconvolute mixture data using CIBERSORT*

Description

Applies the CIBERSORT algorithm to deconvolute bulk proteome data into constituent cell types using a signature matrix. The function requires the original CIBERSORT.R script to be sourced.

Usage

```
deconvolute_cibersort(
  data,
  signature,
  QN = FALSE,
  absolute = FALSE,
  abs_method = "sig.score",
  ...
)
```

Arguments

`data`	A numeric matrix containing mixture data with genes as row names and samples as columns.
`signature`	A numeric matrix containing signature data with genes as row names and cell types as columns.
`QN`	Logical indicating whether quantile normalization is performed (default FALSE).
`absolute`	Logical indicating whether an absolute score is computed (default FALSE).
`abs_method`	Method for absolute scoring if absolute is TRUE (default "sig.score").
`...`	Additional arguments passed to the CIBERSORT function.

Details

This function requires the original CIBERSORT.R script from the CIBERSORT website (https://cibersortx.stanford.edu/) to be sourced before use. It writes temporary files for the mixture data and signature matrix, calls the CIBERSORT function, and processes the results.

Value

A numeric matrix with samples as rows and cell types as columns, representing the estimated proportion of each cell type in each sample. The returned matrix excludes CIBERSORT's diagnostic columns (RMSE, P-value, Correlation).

See Also

`deconvolute` for a unified interface to multiple deconvolution methods.

Examples

```
## Not run:
# First source the CIBERSORT.R script
source("/path/to/CIBERSORT.R")

# Load example data and signature matrix
data_file <- system.file("extdata", "mixed_samples_matrix.rds", package = "proteoDeconv")
mixed_samples <- readRDS(data_file)

signature_file <- system.file("extdata", "cd8t_mono_signature_matrix.rds", package = "proteoDeconv")
signature_matrix <- readRDS(signature_file)

# Run deconvolution
result <- deconvolute_cibersort(
  data = mixed_samples,
  signature = signature_matrix,
  QN = FALSE,
  absolute = FALSE
)

## End(Not run)
```

`deconvolute_cibersortx` *Deconvolute using CIBERSORTx Docker image*

Description

Performs deconvolution of bulk proteome data into constituent cell types using the CIBERSORTx Docker image. This function handles the interaction with the Docker container and processes the results.

Usage

```
deconvolute_cibersortx(
  data,
  signature,
  perm = 1,
  rmbatch_S_mode = FALSE,
  source_GEPs = NULL,
  rmbatch_B_mode = FALSE,
  QN = FALSE,
  absolute = FALSE,
  abs_method = "sig.score",
  use_sudo = FALSE
)
```

Arguments

data	A numeric matrix containing mixture data with genes as row names and samples as columns.
signature	A numeric matrix containing the signature matrix with genes as row names and cell types as columns.
perm	An integer specifying the number of permutations to be performed. Default is 1.
rmbatch_S_mode	A logical value indicating whether to remove batch effects in source GEPs mode. Default is FALSE.
source_GEPs	A matrix containing the source gene expression profiles. Required if rmbatch_S_mode is TRUE.
rmbatch_B_mode	A logical value indicating whether to remove batch effects in bulk mode. Default is FALSE.
QN	A logical value indicating whether to perform quantile normalization. Default is FALSE.
absolute	A logical value indicating whether to use absolute mode. Default is FALSE.
abs_method	A character string specifying the method to use for absolute mode. Default is "sig.score".
use_sudo	A logical value indicating whether to use sudo for Docker commands. Default is FALSE.

Details

This function requires the CIBERSORTx Docker image to be installed and the CIBERSORTX_EMAIL and CIBERSORTX_TOKEN environment variables to be set. You can get these credentials by registering at the CIBERSORTx website (https://cibersortx.stanford.edu/).

The function creates temporary files for the mixture data and signature matrix, runs the CIBERSORTx Docker container, and processes the results. Note that absolute mode is not currently supported in the Docker version.

Value

A numeric matrix with samples as rows and cell types as columns, representing the estimated proportion of each cell type in each sample.

See Also

deconvolute_cibersort for using the R implementation of CIBERSORT, deconvolute for a unified interface to multiple deconvolution methods.

Examples

```
## Not run:
# Set required environment variables (ideally in .Renviron)
Sys.setenv(CIBERSORTX_EMAIL = "your.email@example.com")
Sys.setenv(CIBERSORTX_TOKEN = "your-token-here")
```

```
# Load example data and signature matrix
data_file <- system.file("extdata", "mixed_samples_matrix.rds", package = "proteoDeconv")
mixed_samples <- readRDS(data_file)

signature_file <- system.file("extdata", "cd8t_mono_signature_matrix.rds", package = "proteoDeconv")
signature_matrix <- readRDS(signature_file)

# Run deconvolution with CIBERSORTx Docker
result <- deconvolute_cibersortx(
  data = mixed_samples,
  signature = signature_matrix,
  perm = 100,
  QN = TRUE
)

## End(Not run)
```

`deconvolute_epic` *Deconvolute bulk data using EPIC*

Description

Runs the EPIC algorithm on preprocessed proteomic data to estimate cell type proportions in mixed samples using a reference signature matrix.

Usage

```
deconvolute_epic(data, signature, with_other_cells = TRUE)
```

Arguments

`data`	A numeric matrix of the bulk proteome data with gene identifiers as row names and samples as columns.
`signature`	A numeric matrix of reference signature profiles with gene identifiers as row names and cell types as columns.
`with_other_cells`	Logical; if TRUE, EPIC will include an "other cells" component in the output to account for cell types not present in the reference. Default is TRUE.

Details

The function normalizes both the input data matrix and signature matrix using the `handle_scaling` function before running the EPIC deconvolution. The EPIC algorithm uses constrained least squares regression to estimate cell type proportions.

Value

A numeric matrix with samples as rows and cell types as columns, representing the estimated proportion of each cell type in each sample.

References

Racle, J. et al. (2017). Simultaneous enumeration of cancer and immune cell types from bulk tumor gene expression data. eLife, 6:e26476.

See Also

handle_scaling for preprocessing inputs before deconvolution, deconvolute for a unified interface to multiple deconvolution methods.

Examples

```
## Not run:
# Load example data and signature matrix
data_file <- system.file("extdata", "mixed_samples_matrix.rds", package = "proteoDeconv")
mixed_samples <- readRDS(data_file)

signature_file <- system.file("extdata", "cd8t_mono_signature_matrix.rds", package = "proteoDeconv")
signature_matrix <- readRDS(signature_file)

# Run EPIC deconvolution
result <- deconvolute_epic(
  data = mixed_samples,
  signature = signature_matrix,
  with_other_cells = TRUE
)

# View first few rows of the result
head(result)

## End(Not run)
```

---

extract_identifiers *Extract primary identifiers from e.g. protein groups*

Description

Extracts the primary identifier from compound row identifiers in a matrix by splitting on a separator and keeping only the first entry.

Usage

```
extract_identifiers(data, separator = ";")
```

Arguments

data	A numeric matrix with compound identifiers (e.g. protein/gene groups) as row names.
separator	A character string used to separate multiple identifiers. Default is ";".

Value

A matrix with simplified identifiers as row names, where each identifier is the first element from the original compound identifier.

Examples

```
# Create matrix with compound identifiers (like protein groups)
mat <- matrix(1:9, nrow = 3, ncol = 3)
rownames(mat) <- c("P04637;P02340", "Q15796;O35182", "P01308;P01315;P01317")
colnames(mat) <- c("Sample1", "Sample2", "Sample3")

# View original matrix
print(mat)

# Extract primary identifiers
result <- extract_identifiers(mat)
print(result)
```

---

handle_duplicates *Handle duplicate identifiers in an expression matrix*

Description

Resolves duplicate row identifiers in an expression matrix using the specified method.

Usage

```
handle_duplicates(data, duplicate_mode = "slice")
```

Arguments

`data`	A numeric matrix containing expression data with identifiers as row names and samples as columns.
`duplicate_mode`	A string specifying the approach to handle duplicates: • `"slice"`: Keep only the row with the maximum median value for each identifier (default) • `"merge"`: Merge duplicate rows by taking the column-wise median of values

Value

A numeric matrix with unique identifiers as row names. The number of rows will be equal to the number of unique identifiers in the input matrix.

Examples

```
# Create example matrix with duplicate identifiers
mat <- matrix(1:12, nrow = 4, ncol = 3)
rownames(mat) <- c("ID1", "ID2", "ID1", "ID3")
colnames(mat) <- c("Sample1", "Sample2", "Sample3")

# View original matrix
print(mat)

# Handle duplicates by keeping rows with maximum median (default)
result1 <- handle_duplicates(mat)
print(result1)

# Handle duplicates by merging rows
result2 <- handle_duplicates(mat, duplicate_mode = "merge")
print(result2)
```

---

handle_missing_values *Handle missing values*

Description

Imputes missing values in an expression matrix using various methods, including simple replacement with the lowest non-zero value or more sophisticated imputation techniques via the MsCoreUtils package.

Usage

```
handle_missing_values(data, imputation_mode = "lowest_value", ...)
```

Arguments

`data`	A numeric matrix containing the data to be processed, with identifiers as row names and samples as columns.
`imputation_mode`	A string specifying the imputation method to use: • `"lowest_value"` (default): Replaces NAs with the lowest non-zero value in the matrix • `"knn"`: k-nearest neighbor imputation • `"zero"`: Replace NAs with zeros • `"MLE"`: Maximum likelihood estimation • `"bpca"`: Bayesian principal component analysis • `"RF"`: Random Forest imputation • `"min"`: Replace NAs with minimum value in each column • `"MinDet"`: Deterministic minimum value imputation

- "MinProb": Probabilistic minimum value imputation
- "QRILC": Quantile regression imputation of left-censored data
- "mixed": Mixed imputation based on feature-wise missingness
- "nbavg": Impute with average of neighbors

... Additional arguments passed to MsCoreUtils::impute_matrix. See the documentation of that function for method-specific parameters.

Value

A matrix with the same dimensions as the input, but with missing values imputed according to the specified method.

See Also

impute_matrix for detailed description of the imputation methods

Examples

```
# Create example matrix with missing values
mat <- matrix(c(1.2, 3.4, NA, 5.6, NA, 7.8, 9.0, 2.1, 4.3), nrow = 3, ncol = 3)
rownames(mat) <- c("Protein1", "Protein2", "Protein3")
colnames(mat) <- c("Sample1", "Sample2", "Sample3")

# View original matrix
print(mat)

# Impute missing values with lowest non-zero value (default)
result1 <- handle_missing_values(mat)
print(result1)

# Impute missing values using k-nearest neighbors
## Not run:
# Requires the 'impute' package
result2 <- handle_missing_values(mat, imputation_mode = "knn")
print(result2)

## End(Not run)
```

---

handle_scaling *Handle scaling of protein abundance data*

Description

Scales protein abundance data by optionally converting from log2 scale to linear scale and/or applying TPM-like normalization for proteomics data comparison.

Usage

```
handle_scaling(data, unlog = FALSE, tpm = FALSE)
```

Arguments

data	A numeric matrix containing protein abundance data with identifiers as row names and samples as columns.
unlog	A logical value indicating whether to unlog the data (convert from log2 scale). Default is FALSE.
tpm	A logical value indicating whether to apply TPM-like normalization (adapting Transcripts Per Million for proteomics). Default is FALSE.

Details

This function combines the functionality of unlog2_data and convert_to_tpm to provide a flexible way to handle common scaling operations for proteomics data.

Value

A numeric matrix with the scaled protein abundance data according to the specified options.

See Also

unlog2_data for just converting from log2 scale, convert_to_tpm for just applying TPM-like normalization

Examples

```
# Create log2-transformed protein abundance matrix
log2_mat <- matrix(rnorm(12, mean = 10, sd = 2), nrow = 4, ncol = 3)
rownames(log2_mat) <- paste0("Protein", 1:4)
colnames(log2_mat) <- paste0("Sample", 1:3)

# Convert from log2 to linear scale only
linear_mat <- handle_scaling(log2_mat, unlog = TRUE, tpm = FALSE)

# Convert from log2 to linear scale and then apply TPM-like normalization
tpm_mat <- handle_scaling(log2_mat, unlog = TRUE, tpm = TRUE)

# Apply TPM-like normalization to already linear protein abundance data
linear_data <- matrix(abs(rnorm(12, mean = 500, sd = 200)), nrow = 4, ncol = 3)
tpm_only <- handle_scaling(linear_data, unlog = FALSE, tpm = TRUE)
```

`map_cell_groups` *Map column names to cell type groups using patterns*

Description

Maps a character vector of column names to predefined cell type categories by matching against patterns, with optional regex support.

Usage

```
map_cell_groups(
  column_names,
  mapping_rules,
  default_group = "Unknown",
  verbose = FALSE,
  use_regex = TRUE
)
```

Arguments

`column_names`	A character vector of column names to be mapped to cell type groups.
`mapping_rules`	A named list where names are the target cell type groups and values are character vectors of patterns that identify columns belonging to each group.
`default_group`	A string that represents the default group name for columns not matching any pattern. Default is "Unknown".
`verbose`	Logical. If TRUE, prints mapping process messages showing which columns were mapped to which groups and which columns remained unmapped. Default is FALSE.
`use_regex`	Logical. If TRUE (default), treats patterns in mapping_rules as regular expressions. If FALSE, performs exact string matching.

Details

This function iterates through the `mapping_rules` list and attempts to match each column name against the patterns for each cell type group. The first matching group in the order of the list will be assigned.

When `use_regex = TRUE`, patterns are treated as regular expressions and matching is case-insensitive. When `use_regex = FALSE`, exact string matching is performed, which is case-sensitive.

Value

A character vector with the same length as `column_names`, containing the mapped cell type group for each column name. Columns that don't match any pattern will be assigned the `default_group`.

Examples

```
# Example column names from a dataset
cols <- c("CD8_T_cell", "T_cell", "B_cell",
          "NK_cell", "Monocyte", "Unknown_cell")

# Define simple mapping rules for cell types
mapping <- list(
  "T_cell" = c("CD8_T_cell", "T_cell"),
  "B_cell" = c("B_cell"),
  "NK_cell" = c("NK_cell"),
  "Monocyte" = c("Monocyte")
)

# Map column names to cell types using exact matching
cell_types <- map_cell_groups(cols, mapping, use_regex = FALSE)
print(data.frame(column = cols, cell_type = cell_types))

# Define mapping rules using regex patterns
mapping_regex <- list(
  "T_cell" = c("CD8.*", "T_cell"),
  "B_cell" = c("B[-_]?cell"),
  "NK_cell" = c("NK[-_]?cell"),
  "Monocyte" = c("Mono.*")
)

# Map using regex patterns (default)
cell_types_regex <- map_cell_groups(cols, mapping_regex)
print(data.frame(column = cols, cell_type = cell_types_regex))
```

---

simulate_data *Simulate artifical mixtures from bulk proteome measurements*

Description

This function simulates bulk proteome data by mixing bulk sample measurements.

Usage

```
simulate_data(
  data,
  cell_types,
  seed = NULL,
  ncells = 100,
  nsamples = 100,
  filter_genes = TRUE,
  scenario = "random",
  whitelist = NULL,
  blacklist = NULL
)
```

Arguments

`data`	A numeric matrix containing protein abundance data with protein identifiers as row names and samples as columns.
`cell_types`	A character vector indicating the cell type associated with each column in the input matrix. Must have the same length as the number of columns in `data`.
`seed`	An integer used as random seed for reproducibility. Default is NULL (no seed).
`ncells`	Integer specifying the number of cells to use for each simulated bulk sample.
`nsamples`	Integer specifying the number of bulk samples to simulate. Default is 100.
`filter_genes`	Logical; whether to filter out proteins/genes with low expression before simulation. Default is TRUE.
`scenario`	String specifying the simulation scenario: • `"random"` (default): Random cell type proportions for each sample • `"even"`: Even proportions of all cell types • See SimBu documentation for additional scenarios
`whitelist`	Optional character vector of cell types to include in the simulation. If provided, only these cell types will be used. Default is NULL (use all).
`blacklist`	Optional character vector of cell types to exclude from the simulation. Default is NULL (exclude none).

Details

This function uses the SimBu package to generate synthetic bulk samples by artifically mixing samples.

Value

A list containing two matrices:

- `simulated_data`: Matrix of simulated bulk protein abundance data (proteins as rows, simulated samples as columns)
- `cell_fractions`: Matrix of cell type fractions used for each simulation (cell types as rows, simulated samples as columns)

Examples

```
# Create example data
cell_data <- matrix(abs(rnorm(1500, mean = 500, sd = 200)), nrow = 100, ncol = 15)
rownames(cell_data) <- paste0("Protein", 1:100)
colnames(cell_data) <- paste0("Cell", 1:15)

# Define cell types
cell_types <- rep(c("T_cell", "B_cell", "Monocyte"), each = 5)

# Run simulation
## Not run:
sim_results <- simulate_data(
```

```
  data = cell_data,
  cell_types = cell_types,
  seed = 42,
  nsamples = 20,
  scenario = "random"
)

dim(sim_results$simulated_data)
dim(sim_results$cell_fractions)

## End(Not run)
```

`unlog2_data` *Convert data from log2 scale to linear scale*

Description

Transforms log2-transformed expression data back to linear scale by calculating 2^x for each value in the input matrix.

Usage

```
unlog2_data(data)
```

Arguments

`data`	A numeric matrix containing log2-transformed data with identifiers as row names and samples as columns.

Details

This function can be used to convert log2-transformed proteomics or gene expression data back to their original scale for downstream analyses that require linear values. It preserves the row and column names of the input matrix.

Value

A numeric matrix with the same dimensions as the input, with values converted to linear scale (2^x).

Examples

```
# Create log2-transformed data matrix
log2_mat <- matrix(rnorm(12, mean = 10, sd = 2), nrow = 4, ncol = 3)
rownames(log2_mat) <- paste0("Protein", 1:4)
colnames(log2_mat) <- paste0("Sample", 1:3)

# View original log2 values
print(log2_mat)
```

```
# Convert to linear scale
linear_mat <- unlog2_data(log2_mat)
print(linear_mat)
```

`update_gene_symbols` *Update gene symbols in a matrix to approved HGNC nomenclature*

Description

This function checks and updates gene symbols in a matrix's row names to ensure they conform to the current approved HGNC (HUGO Gene Nomenclature Committee) standards. Non-standard or outdated gene symbols are replaced with their current official symbols.

Usage

```
update_gene_symbols(data, verbose = FALSE)
```

Arguments

`data`	A numeric matrix containing gene expression or protein abundance data with gene identifiers as row names.
`verbose`	A logical value indicating whether to print detailed messages about the number of approved, non-approved, and unmappable gene symbols. Default is FALSE.

Details

The function uses the `HGNChelper` package to check and standardize gene symbols based on a predefined gene symbols map. This is important for ensuring consistency in gene naming across datasets and avoiding issues with outdated or non-standard gene symbols.

Value

A matrix with updated gene symbols as row names. Rows with unmappable symbols (where no suggested symbol could be found) are removed from the output.

Index