Package 'drugfindR'

May 25, 2026

Title Investigate iLINCS for candidate repurposable drugs

Version 1.1.0

Description This package provides a convenient way to access the LINCS Signatures available in the iLINCS database. These signatures include Consensus Gene Knockdown Signatures, Gene Overexpression signatures and Chemical Perturbagen Signatures. It also provides a way to enter your own transcriptomic signatures and identify concordant and discordant signatures in the LINCS database.

License GPL-3 + file LICENSE

Encoding UTF-8

URL https://github.com/CogDisResLab/drugfindR, https://cogdisreslab.github.io/drugfindR/

BugReports https://github.com/CogDisResLab/drugfindR/issues

LazyData false

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

biocViews FunctionalPrediction, DifferentialExpression, GeneSetEnrichment, SingleCell, Network

Imports tibble, rlang, dplyr, purrr, readr, stringr, stats, lifecycle, S4Vectors, httr2, curl, DFplyr

Depends R (>= 4.5.0)

Suggests AnnotationDbi, BiocStyle, biocthis, codemetar, devtools, here, httptest2, jsonlite, knitr, rmarkdown, testthat (>= 3.0.0), tidyverse, usethis

Config/testthat/edition 3

Config/testthat/parallel true

VignetteBuilder knitr

X-schema.org-applicationCategory Genomics

X-schema.org-keywords LINCS, iLINCS, drug repurposing, drug discovery, transcriptomics, gene expression, gene knockdown, gene overexpression, chemical perturbagen, drugfindR

X-schema.org-isPartOf https://bioconductor.org

Collate 'utilities.R' 'consensusConcordants.R' 'drugfindR-package.R' 'filterSignature.R' 'getConcordants.R' 'getSignature.R' 'prepareSignature.R' 'investigateSignature.R' 'investigateTarget.R'

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

git_branch devel

git_last_commit 8b653fc

git_last_commit_date 2026-04-28

Repository Bioconductor 3.24

Date/Publication 2026-05-25

Author Ali Sajid Imami [aut, cre] (ORCID: <https://orcid.org/0000-0003-3684-3539>), Smita Sahay [aut] (ORCID: <https://orcid.org/0009-0003-4377-8963>), Justin Fortune Creeden [aut] (ORCID: <https://orcid.org/0000-0003-3123-8401>), Robert Erne McCullumsmith [ctb, fnd] (ORCID: <https://orcid.org/0000-0001-6921-7150>)

Maintainer Ali Sajid Imami <Ali.Sajid.Imami@gmail.com>

Contents

`drugfindR-package` *drugfindR: Drug Repurposing using Gene Expression Signatures*

Description

drugfindR is a package that allows users to perform drug repurposing using gene expression signatures. It uses the LINCS L1000 signatures available from iLINCS to find the most similar gene expression signatures to the user's input signature. It then uses the connectivity analysis to find the most similar drugs to the user's input signature.

Details

drugfindR-package

The drugfindR package allows users to perform drug repurposing using gene expression signatures. It uses the LINCS L1000 database to find the most similar gene expression signatures to the user's input signature. It then uses the connectivity analysis to find the most similar drugs to the user's input signature.

References

1. Imami, Ali S., Sinead M. O'donovan, Justin F. Creeden, Xiaojun Wu, Hunter Eby, Cheryl B. McCullumsmith, Kerstin Uvnas-Moberg, Robert E. McCullumsmith, and Elissar Andari. "Oxytocin's Anti-Inflammatory and Proimmune Functions in Covid-19: A Transcriptomic Signature-Based Approach." Physiological Genomics, 2020. https://doi.org/10.1152/physiolgenomics.00095.2020.
2. O'Donovan, Sinead M., Ali Imami, Hunter Eby, Nicholas D. Henkel, Justin Fortune Creeden, Sophie Asah, Xiaolu Zhang, et al. "Identification of Candidate Repurposable Drugs to Combat COVID-19 Using a Signature-Based Approach." Scientific Reports 11, no. 1 (December 24, 2021): 4495. https://doi.org/10.1038/s41598-021-84044-9.
3. Creeden, Justin Fortune, Ali Sajid Imami, Hunter M. Eby, Cassidy Gillman, Kathryn N. Becker, Jim Reigle, Elissar Andari, et al. "Fluoxetine as an Anti-Inflammatory Therapy in SARS-CoV-2 Infection." Biomedicine & Pharmacotherapy 138 (June 1, 2021): 111437. https://doi.org/10.1016/j.biopha.2021.111437.

Author

Ali Sajid Imami

Maintainer

Ali Sajid Imami

Contributors

Smita Sahay Justin Fortune Creeden Sinead M. O'Donovan Hunter Eby

License

GNU Public License 3.0

Acknowledgements

- This work was supported by the National Institutes of Mental Health of the USA.
- This work was heavily supported by the members of the Cognitive Disorders Research Lab.
- This work was made possible by the Fulbright Scholar Program with the Fulbright Masters Grant awarded to Ali Sajid Imami.

Author(s)

Maintainer: Ali Sajid Imami <Ali.Sajid.Imami@gmail.com> (ORCID)

Authors:

- Smita Sahay <Smita.Sahay@rockets.utoledo.edu> (ORCID)
- Justin Fortune Creeden <Justin.Creeden@rockets.utoledo.edu> (ORCID)

Other contributors:

- Robert Erne McCullumsmith <Robert.McCullumsmith@utoledo.edu> (ORCID) [contributor, funder]

See Also

Useful links:

- https://github.com/CogDisResLab/drugfindR
- https://cogdisreslab.github.io/drugfindR/
- Report bugs at https://github.com/CogDisResLab/drugfindR/issues

---

`.applyDirectionFilter` *Apply filtering based on direction and thresholds*

Description

This internal function performs the actual filtering of the signature data based on the specified direction and calculated thresholds. It implements the core filtering logic using dplyr operations.

Usage

```
.applyDirectionFilter(signature, thresholds, direction = "any")
```

Arguments

`signature`	A data.frame-like object containing the signature data. Must have a column named "Value_LogDiffExp" containing log fold-change values.
`thresholds`	A named list containing: • `downThreshold`: Threshold for down-regulated genes • `upThreshold`: Threshold for up-regulated genes
`direction`	Character string specifying the filtering direction. Must be one of: * "up": Keep only up-regulated genes (logFC >= upThreshold) * "down": Keep only down-regulated genes (logFC <= downThreshold) * "any": Keep both up- and down-regulated genes (logFC >= upThreshold OR logFC <= downThreshold)

Details

The filtering logic depends on the direction parameter:

- `"up"`: Retains genes where `Value_LogDiffExp >= upThreshold`
- `"down"`: Retains genes where `Value_LogDiffExp <= downThreshold`
- `"any"`: Retains genes where `Value_LogDiffExp >= upThreshold OR Value_LogDiffExp <= downThreshold`

The function uses `dplyr::filter` with `rlang::.data` for non-standard evaluation, ensuring compatibility with different data frame types and avoiding issues with variable scoping.

Value

A tibble containing the filtered signature data with the same structure as the input but including only rows that meet the filtering criteria.

Examples

```
## Not run:
# Create sample signature data
signature <- data.frame(
    signatureID = rep("TEST", 10),
    Name_GeneSymbol = paste0("GENE", 1:10),
    Value_LogDiffExp = c(-3, -2, -1, -0.5, 0, 0.5, 1, 2, 3, 4)
)

# Define thresholds
thresholds <- list(downThreshold = -1.5, upThreshold = 1.5)

# Filter for up-regulated genes only
up_filtered <- .applyDirectionFilter(signature, "up", thresholds)
# Returns genes with logFC >= 1.5 (GENE8, GENE9, GENE10)

# Filter for down-regulated genes only
down_filtered <- .applyDirectionFilter(signature, "down", thresholds)
# Returns genes with logFC <= -1.5 (GENE1, GENE2)

# Filter for both up- and down-regulated genes
both_filtered <- .applyDirectionFilter(signature, "any", thresholds)
# Returns genes with |logFC| >= 1.5 (GENE1, GENE2, GENE8, GENE9, GENE10)

## End(Not run)
```

`.applySimilarityCutoff` *Apply similarity cutoff filter to concordants data*

Description

This internal function filters concordants data based on absolute similarity values meeting or exceeding the specified cutoff threshold.

Usage

```
.applySimilarityCutoff(concordants, cutoff)
```

Arguments

`concordants`	A dataframe containing concordants data.
`cutoff`	Numeric similarity cutoff value.

Details

This function:

1. Filters based on absolute similarity values
2. Retains both positive and negative similarities above threshold
3. Removes entries below the cutoff threshold

Value

A filtered dataframe containing only entries meeting the similarity cutoff.

Examples

```
## Not run:
testData <- data.frame(similarity = c(0.5, -0.8, 0.2, -0.1))
filtered <- .applySimilarityCutoff(testData, 0.3)
# Returns entries with |similarity| >= 0.3

## End(Not run)
```

.applyTargetRenaming *Apply target column renaming to consensus results*

Description

This internal function applies the standard target column renaming to produce the final consensus concordants output format.

Usage

```
.applyTargetRenaming(concordants)
```

Arguments

concordants	A dataframe containing selected consensus results.

Details

This function:

1. Applies targetRename function to standardize column names
2. Converts internal column names to user-facing consensus format
3. Handles different library types appropriately

Value

A dataframe with renamed columns following consensus output standards.

Examples

```
## Not run:
testData <- data.frame(
    signatureid = "SIG1",
    compound = "A",
    cellline = "A375",
    similarity = 0.8
)
renamed <- .applyTargetRenaming(testData)

## End(Not run)
```

`.calculateAbsoluteThresholds` *Calculate thresholds using absolute threshold values*

Description

This internal function coordinates the calculation of filtering thresholds when absolute threshold values are provided. It dispatches to the appropriate calculation function based on the number of threshold values provided.

Usage

```
.calculateAbsoluteThresholds(threshold)
```

Arguments

`threshold`	A numeric value or vector specifying the absolute threshold(s).
	Can be:
	`- A single value: Dispatched to [.calculateSingleThreshold()]`
	`- A vector of two values: Dispatched to [.calculateDoubleThreshold()]`

Details

This function serves as a dispatcher that:

- Checks the length of the threshold parameter
- Calls the appropriate threshold calculation function
- Throws an error if an invalid number of thresholds is provided

The function ensures that only single values or pairs of values are accepted, maintaining the integrity of the filtering logic.

Value

A named list with two elements:

- `downThreshold`: The threshold for down-regulated genes
- `upThreshold`: The threshold for up-regulated genes

Examples

```
## Not run:
# Single threshold - creates symmetric thresholds
thresholds <- .calculateAbsoluteThresholds(1.0)
# Returns: list(downThreshold = -1.0, upThreshold = 1.0)

# Double threshold - uses provided values
thresholds <- .calculateAbsoluteThresholds(c(-1.5, 2.0))
# Returns: list(downThreshold = -1.5, upThreshold = 2.0)

# Invalid - too many values (will throw error)
# thresholds <- .calculateAbsoluteThresholds(c(1.0, 2.0, 3.0))
```

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

`.calculateDoubleThreshold` *Calculate thresholds from two threshold values*

Description

This internal function handles asymmetric filtering thresholds when two threshold values are provided. The first value is used as the down-regulated threshold and the second value is used as the up-regulated threshold.

Usage

```
.calculateDoubleThreshold(threshold)
```

Arguments

`threshold`	A numeric vector of length 2 containing the threshold values. The first element is the down-regulated threshold (typically negative), and the second element is the up-regulated threshold (typically positive).

Details

This function enables asymmetric filtering where different absolute thresholds can be applied to up-regulated and down-regulated genes. This is useful when you want to apply stricter criteria to one direction of regulation than the other.

Value

A named list with two elements:

- `downThreshold`: The down-regulated threshold (`threshold[1]`)
- `upThreshold`: The up-regulated threshold (`threshold[2]`)

Examples

```
## Not run:
# Create asymmetric thresholds
thresholds <- .calculateDoubleThreshold(c(-2.0, 1.5))
# Returns: list(downThreshold = -2.0, upThreshold = 1.5)

# Stricter threshold for down-regulation
thresholds <- .calculateDoubleThreshold(c(-1.0, 0.5))
# Returns: list(downThreshold = -1.0, upThreshold = 0.5)

# Equal but explicit thresholds
thresholds <- .calculateDoubleThreshold(c(-1.5, 1.5))
# Returns: list(downThreshold = -1.5, upThreshold = 1.5)

## End(Not run)
```

`.calculateProportionalThreshold` *Calculate thresholds using proportional values*

Description

This internal function calculates filtering thresholds based on quantiles of the log fold-change distribution in the signature data. This enables proportion-based filtering that adapts to the data distribution.

Usage

```
.calculateProportionalThreshold(signature, prop)
```

Arguments

`signature`	A data.frame-like object containing the signature data. Must have a column named "Value_LogDiffExp" containing log fold-change values.
`prop`	A numeric value between 0 and 1 specifying the proportion of genes to select from each tail of the distribution.

Details

This function calculates thresholds using the `quantile` function:

- `downThreshold`: The `prop` quantile of the expression values
- `upThreshold`: The `1-prop` quantile of the expression values

For example, with `prop = 0.1`:

- `downThreshold`: 10th percentile (bottom 10% of values)
- `upThreshold`: 90th percentile (top 10% of values)

This approach is particularly useful when you want to select a fixed proportion of the most differentially expressed genes regardless of their absolute expression values.

Value

A named list with two elements: *`downThreshold`: The quantile threshold for down-regulated genes (quantile at prop) *`upThreshold`: The quantile threshold for up-regulated genes (quantile at 1-prop)

Examples

```
## Not run:
# Create sample signature data
signature <- data.frame(
    Value_LogDiffExp = c(-3, -2, -1, 0, 1, 2, 3, 4, 5, 6)
)

# Calculate thresholds for top/bottom 20%
thresholds <- .calculateProportionalThreshold(signature, 0.2)
# Returns thresholds based on 20th and 80th percentiles
```

```
# Calculate thresholds for top/bottom 10%
thresholds <- .calculateProportionalThreshold(signature, 0.1)
# Returns thresholds based on 10th and 90th percentiles

# Calculate thresholds for top/bottom 5% (most extreme)
thresholds <- .calculateProportionalThreshold(signature, 0.05)
# Returns thresholds based on 5th and 95th percentiles

## End(Not run)
```

`.calculateSingleThreshold` *Calculate thresholds from single threshold value*

Description

This internal function creates symmetric filtering thresholds from a single threshold value. The input value is used as the positive threshold, and its negative is used as the negative threshold.

Usage

```
.calculateSingleThreshold(threshold)
```

Arguments

`threshold`	A single positive numeric value representing the absolute threshold for filtering.

Details

This function is used when a single threshold value is provided to `filterSignature`. It creates symmetric thresholds where genes with log fold-change values greater than or equal to the positive threshold (up-regulated) or less than or equal to the negative threshold (down-regulated) are retained.

Value

A named list with two elements:

- `downThreshold`: The negative threshold (-threshold)
- `upThreshold`: The positive threshold (threshold)

Examples

```
## Not run:
# Create symmetric thresholds from threshold = 1.5
thresholds <- .calculateSingleThreshold(1.5)
# Returns: list(downThreshold = -1.5, upThreshold = 1.5)

# Create symmetric thresholds from threshold = 0.8
thresholds <- .calculateSingleThreshold(0.8)
# Returns: list(downThreshold = -0.8, upThreshold = 0.8)

## End(Not run)
```

.cleanupGetConcordants

Clean up temporary signature file

Description

This internal function removes the temporary signature file created during the getConcordants operation to prevent accumulation of temporary files.

Usage

```
.cleanupGetConcordants(signatureFile)
```

Arguments

signatureFile	Character string path to the temporary signature file to be removed.

Details

The function checks if the specified file exists and removes it using `unlink()`. This cleanup is performed automatically at the end of the getConcordants operation.

Value

Invisible NULL. The function is called for its side effect of removing the temporary file.

Examples

```
NULL
```

.combineConcordantsData

Combine concordants dataframes for consensus analysis

Description

This internal function combines one or more concordants dataframes into a single dataframe for further processing.

Usage

```
.combineConcordantsData(dots)
```

Arguments

dots	A list of dataframes to combine.

Details

This function:

1. Combines multiple dataframes using row binding
2. Preserves all columns from input dataframes
3. Handles cases where dataframes have different column sets

Value

A combined dataframe with all input data.

Examples

```
## Not run:
df1 <- data.frame(similarity = 0.5, compound = "A")
df2 <- data.frame(similarity = -0.3, compound = "B")
combined <- .combineConcordantsData(list(df1, df2))

## End(Not run)
```

.computeConsensusFromSignature

Compute consensus concordant signatures from a single input signature

---

Description

This internal helper wraps the common paired / unpaired workflow used by `investigateSignature()` and `investigateTarget()` for a single already prepared or retrieved signature. It applies directional filtering, queries iLINCS for concordant signatures, and collapses results via `consensusConcordants()`.

Usage

```
.computeConsensusFromSignature(
  signature,
  outputLib,
  filterThreshold = NULL,
  filterProp = NULL,
  similarityThreshold = 0.321,
  paired = TRUE,
  outputCellLines = NULL
)
```

Arguments

`signature`	A data.frame / tibble / DataFrame produced by `prepareSignature()` or `getSignature()` with standard signature columns.
`outputLib`	Character. One of "OE", "KD", or "CP" indicating the iLINCS library to search for concordant signatures.

`filterThreshold`
: Numeric (optional). Absolute threshold(s) passed to `filterSignature()`. Use either `filterThreshold` or `filterProp`.

`filterProp`
: Numeric in (0, 0.5] (optional). Proportion for quantile based filtering in `filterSignature()`. Ignored if `filterThreshold` is supplied.

`similarityThreshold`
: Numeric in 0..1. Minimum absolute similarity retained by `consensusConcordants()`.

`paired`
: Logical. If TRUE perform separate up / down filtering and concordance queries; otherwise aggregate direction = "any".

`outputCellLines`
: Optional character vector restricting target cell lines during consensus filtering. Passed to `consensusConcordants()`.

Details

Error handling is delegated to component functions:

- Library validation via `stopIfInvalidLibraries()`
- Signature structure via `stopIfInvalidSignature()` (indirectly used by `getConcordants()`)
- Filtering parameter validation via `.validateFilterSignatureInput()`
- Concordance / network errors via internal iLINCS response processors.

If both `filterThreshold` and `filterProp` are supplied an error is raised upstream in `filterSignature()`. Provide only one.

Value

A tibble of consensus concordant signatures with standardized target columns (already renamed via internal consensus pipeline). Columns include `TargetSignature`, `Target`, `TargetCellLine`, `Similarity`, `pValue`, `InputSigDirection`, `SignatureType`, and optional time / concentration.

Examples

```
NULL
```

.createSignatureRequest

Create HTTP request for iLINCS signature retrieval

Description

This internal function constructs and configures the HTTP request object for retrieving signature data from the iLINCS API.

Usage

```
.createSignatureRequest(sigId)
```

Arguments

`sigId`
: A character string containing the iLINCS signature ID to retrieve.

Details

This function builds a complete HTTP request by:

1. Setting the base URL using `.ilincsBaseUrl()`
2. Appending the API path: "ilincsR/downloadSignature"
3. Adding query parameters: sigID and noOfTopGenes (set to Inf for all genes)
4. Setting the HTTP method to POST
5. Adding a user agent string using `.returnUserAgent()`

The request is configured but not executed - it must be performed using the request execution function.

Value

An httr2 request object configured for the iLINCS downloadSignature endpoint.

Examples

```
NULL
```

.detectSignatureDirection *Detect signature direction from expression values*

Description

This internal function analyzes the log fold-change values in a signature to determine the overall direction of regulation.

Usage

```
.detectSignatureDirection(signature)
```

Arguments

`signature`	A data.frame-like object containing the signature data. Must have a column named "Value_LogDiffExp" with log fold-change values.

Details

The function examines the "Value_LogDiffExp" column to determine direction:

- "Up": All expression values are greater than or equal to zero
- "Down": All expression values are less than or equal to zero
- "Any": Mixed positive and negative values

Note that zero values are considered "Up" direction. This direction information is used by iLINCS for signature analysis and is included in the output results.

Value

Character string indicating signature direction: "Up", "Down", or "Any".

Examples

```
NULL
```

`.executeIlincsRequest` *Execute iLINCS API request with error handling*

Description

This internal function safely executes an httr2 request and captures errors instead of raising them, allowing downstream functions to handle them appropriately.

Usage

```
.executeIlincsRequest(request, verbose = FALSE)
```

Arguments

`request`	An httr2_request object to be executed.
`verbose`	Logical indicating whether to display request details. Default is FALSE.

Details

This function configures the request to not raise errors automatically on HTTP error status codes (4xx, 5xx) by using `httr2::req_error()`. Instead, error responses are returned as response objects that can be processed by `.processIlincsResponse()` to generate appropriate error messages with context.

The function handles:

- Network connection errors
- HTTP error status codes (400, 401, 403, 404, 500, etc.)
- Timeout errors
- Other httr2 request failures

Value

An httr2_response object, including error responses that would normally cause httr2 to raise an error.

Examples

```
NULL
```

`.executeSignatureRequest` *Execute iLINCS API request with error handling*

Description

This internal function safely executes an httr2 request and captures errors instead of raising them, allowing downstream functions to handle them appropriately.

Usage

```
.executeSignatureRequest(request, verbose = FALSE)
```

Arguments

`request`	An httr2_request object to be executed.
`verbose`	Logical indicating whether to display request details. Default is FALSE.

Details

This function configures the request to not raise errors automatically on HTTP error status codes (4xx, 5xx) by using `httr2::req_error()`. Instead, error responses are returned as response objects that can be processed by `.processSignatureResponse()` to generate appropriate error messages with context.

The function handles:

- Network connection errors
- HTTP error status codes (400, 401, 403, 404, 500, etc.)
- Timeout errors
- Other httr2 request failures

Value

An httr2_response object, including error responses that would normally cause httr2 to raise an error.

Examples

```
NULL
```

.filterByCellLine *Filter concordants data by cell line*

Description

This internal function filters the concordants data to include only the specified cell lines.

Usage

```
.filterByCellLine(concordants, cellLine)
```

Arguments

concordants	A dataframe containing concordants data.
cellLine	A character vector of cell lines to include, or NULL for no filtering.

Details

This function:

1. Filters data based on the cellline column
2. Returns original data if cellLine is NULL
3. Handles cases where no data matches the specified cell lines

Value

A filtered dataframe containing only the specified cell lines.

Examples

```
## Not run:
testData <- data.frame(
    similarity = c(0.5, -0.3, 0.7),
    cellline = c("A375", "PC3", "MCF7")
)
filtered <- .filterByCellLine(testData, c("A375", "PC3"))

## End(Not run)
```

.generateIlincsRequest *Create iLINCS API request*

Description

This internal function constructs and executes the HTTP request to the iLINCS API for concordant signature analysis.

Usage

```
.generateIlincsRequest(signatureFile, ilincsLibrary)
```

Arguments

signatureFile	Character string path to the signature file to upload.
ilincsLibrary	Character string specifying the iLINCS library to search. Must be one of "OE", "KD", or "CP".

Details

The function:

1. Maps the library name to the internal iLINCS library ID
2. Constructs a multipart POST request with the signature file
3. Includes appropriate user agent and API endpoint
4. Executes the request and returns the response

The library mapping is:

- CP (Chemical Perturbagen): LIB_5
- KD (Knockdown): LIB_6
- OE (Overexpression): LIB_11

Value

An httr2 response object from the iLINCS API.

Examples

```
NULL
```

.groupByTargetAndSelectMax

Group concordants by target and select maximum similarity entries

---

Description

This internal function groups concordants data by target (compound or treatment) and retains only the entries with maximum absolute similarity for each target.

Usage

```
.groupByTargetAndSelectMax(concordants)
```

Arguments

concordants	A dataframe containing filtered concordants data.

Details

This function:

1. Groups by treatment or compound columns (whichever is available)
2. For each group, retains only entries with maximum absolute similarity
3. Handles ties by keeping all tied entries
4. Preserves the structure for downstream processing

Value

A dataframe with deduplicated targets, keeping maximum similarity entries.

Examples

```
## Not run:
testData <- data.frame(
    compound = c("A", "A", "B", "B"),
    similarity = c(0.5, 0.8, -0.3, -0.7),
    cellline = c("A375", "PC3", "A375", "PC3")
)
grouped <- .groupByTargetAndSelectMax(testData)
# Returns entries with max |similarity| for each compound

## End(Not run)
```

.ilincsBaseUrl *Parameterize the base URL for the iLINCS API*

Description

Parameterize the base URL for the iLINCS API

Usage

```
.ilincsBaseUrl()
```

Value

a fixed string URL

.isValidSignatureId *Check if a signature ID exists in the metadata tables*

Description

This internal function validates whether a signature ID exists in any of the metadata tables (CP, KD, or OE).

Usage

```
.isValidSignatureId(sigId)
```

Arguments

sigId	A character string or vector containing the signature ID(s) to validate.

Details

This function searches all three metadata tables:

- Chemical Perturbagen (CP) metadata
- Knockdown (KD) metadata
- Overexpression (OE) metadata

The function checks the "SourceSignature" column in each metadata table for the provided signature ID(s).

Value

A logical value or vector: TRUE if the signature exists, FALSE otherwise. Returns a vector of the same length as the input when given a vector.

Examples

```
NULL
```

.loadMetadata *Load the correct metadata table for a given library*

Description

This internal function retrieves the appropriate metadata table based on the specified iLINCS library type.

Usage

```
.loadMetadata(lib)
```

Arguments

lib	A character string specifying the library type. Must be one of "OE", "KD", or "CP".

Details

The function loads pre-compiled metadata tables for each library:

- "OE": Overexpression metadata (oeMetadata)
- "KD": Knockdown metadata (kdMetadata)
- "CP": Chemical Perturbagen metadata (cpMetadata)

These metadata tables are included with the package and contain information about available signatures in each iLINCS library.

Value

A tibble containing the metadata for the specified library. The structure varies by library type but typically includes columns for signature identifiers, treatments, cell lines, and other metadata.

Examples

```
NULL
```

`.mapToL1000WithoutPvalues`

Map filtered data to L1000 format without p-values

Description

This internal function maps the filtered expression data to the standardized L1000 signature format, without p-value information.

Usage

```
.mapToL1000WithoutPvalues(filteredData, geneColumn, logfcColumn)
```

Arguments

`filteredData`	A dataframe containing filtered differential expression data.
`geneColumn`	Character string specifying the column name containing gene symbols.
`logfcColumn`	Character string specifying the column name containing log fold-change values.

Value

A tibble with standardized L1000 signature format without p-values.

`.mapToL1000WithPvalues`

Map filtered data to L1000 format with p-values

Description

This internal function maps the filtered differential expression data to the standardized L1000 signature format, including p-value information.

Usage

```
.mapToL1000WithPvalues(filteredData, geneColumn, logfcColumn, pvalColumn)
```

Arguments

`filteredData`	A dataframe containing filtered differential expression data.
`geneColumn`	Character string specifying the column name containing gene symbols.
`logfcColumn`	Character string specifying the column name containing log fold-change values.
`pvalColumn`	Character string specifying the column name containing p-values.

Value

A tibble with standardized L1000 signature format including p-values.

.prepareSignatureFile *Prepare signature file for iLINCS upload*

Description

This internal function creates a temporary file containing the signature data formatted for upload to the iLINCS API.

Usage

```
.prepareSignatureFile(signature)
```

Arguments

signature	A data.frame-like object containing the signature data.

Details

The function creates a temporary file with a ".xls" extension and writes the signature data as a tab-separated file. The file is automatically cleaned up by the system when the R session ends.

Value

Character string path to the temporary signature file.

Examples

```
NULL
```

.processConsensusPipeline *Process concordants data through the complete consensus pipeline*

Description

This internal function orchestrates the complete processing pipeline for consensus concordants analysis.

Usage

```
.processConsensusPipeline(concordants, cutoff, cellLine)
```

Arguments

concordants	A combined dataframe containing all concordants data.
cutoff	Numeric similarity cutoff value.
cellLine	Character vector of cell lines to include, or NULL.

Details

This function coordinates the following processing steps:

1. Cell line filtering (if specified)
2. Similarity cutoff application
3. Target grouping and maximum similarity selection
4. Column selection and ordering
5. Target column renaming

Value

A processed dataframe with consensus concordants results.

Examples

```
## Not run:
testData <- data.frame(
    similarity = c(0.5, -0.8, 0.2),
    compound = c("A", "B", "C"),
    cellline = c("A375", "PC3", "A375")
)
processed <- .processConsensusPipeline(testData, 0.3, "A375")

## End(Not run)
```

.processIlincsResponse

Process iLINCS API response into concordant signatures

Description

This internal function dispatches to appropriate handlers based on the response status and content from the iLINCS API.

Usage

```
.processIlincsResponse(response, sigDirection, ilincsLibrary)
```

Arguments

`response`	An httr2 response object from the iLINCS API.
`sigDirection`	Character string indicating the signature direction ("Up", "Down", or "Any").
`ilincsLibrary`	Character string specifying the iLINCS library used ("CP", "KD", or "OE").

Details

The function dispatches to specialized handlers:

1. `.processIlincsResponseError` for HTTP error responses
2. `.processIlincsResponseEmpty` for empty concordance tables
3. `.processIlincsResponseSuccess` for successful responses with data

The resulting tibble always contains these columns in order:

- `signatureid`: Unique signature identifier
- `treatment`: Drug/treatment name (compound renamed for CP library)
- `concentration`: Drug concentration (NA for KD/OE libraries)
- `time`: Treatment duration
- `cellline`: Cell line used
- `similarity`: Similarity score (rounded to 8 decimal places)
- `pValue`: Statistical significance (rounded to 20 decimal places)
- `sig_direction`: Signature direction ("Up", "Down", or "Any")
- `sig_type`: Library type description

Value

A tibble containing concordant signature data with standardized column names and rounded numerical values.

`.processIlincsResponseEmpty` *Handle empty concordance table responses*

Description

This internal function creates an empty tibble with the correct structure when the iLINCS API returns no concordant signatures.

Usage

```
.processIlincsResponseEmpty(sigDirection, ilincsLibrary)
```

Arguments

`sigDirection`	Character string indicating the signature direction.
`ilincsLibrary`	Character string specifying the iLINCS library used.

Value

A tibble with zero rows and the correct column structure.

.processIlincsResponseError

Handle iLINCS API response errors

Description

This internal function processes error responses from the iLINCS API and generates appropriate error messages.

Usage

```
.processIlincsResponseError(response)
```

Arguments

response	An httr2 response object from the iLINCS API.

Value

This function always stops execution with an error message.

.processIlincsResponseSuccess

Handle successful concordance table responses

Description

This internal function processes successful responses from the iLINCS API and formats the concordant signature data with standardized columns.

Usage

```
.processIlincsResponseSuccess(concordanceTables, sigDirection, ilincsLibrary)
```

Arguments

concordanceTables	List containing concordance table data from API response.
sigDirection	Character string indicating the signature direction.
ilincsLibrary	Character string specifying the iLINCS library used.

Value

A tibble containing processed concordant signature data.

.processSignatureResponse *Process iLINCS API response for signature retrieval*

Description

This internal function dispatches to appropriate handlers based on the response status from the iLINCS API.

Usage

```
.processSignatureResponse(response)
```

Arguments

response	An httr2 response object from the iLINCS API.

Details

The function dispatches to specialized handlers:

1. `.processSignatureResponseError()` for HTTP error responses
2. `.processSuccessfulResponse()` for successful responses with data

The resulting tibble contains these columns:

- `signatureID`: The signature identifier
- `ID_geneid`: Character gene identifiers
- `Name_GeneSymbol`: Gene symbols
- `Value_LogDiffExp`: Log fold-change values (rounded to 12 decimal places)
- `Significance_pvalue`: P-values (rounded to 12 decimal places)

Value

A tibble containing signature data with standardized columns.

.processSignatureResponseError *Handle API error responses for signature retrieval*

Description

This internal function processes error responses from the iLINCS API and generates appropriate error messages for signature retrieval failures.

Usage

```
.processSignatureResponseError(response)
```

Arguments

response	An httr2 response object from the iLINCS API.

Value

This function always stops execution with an error message.

.processSuccessfulResponse *Process successful API response into signature data frame*

Description

This internal function processes a successful HTTP response from the iLINCS API and converts it into a standardized signature data frame.

Usage

```
.processSuccessfulResponse(response)
```

Arguments

response	An httr2 response object from a successful iLINCS API call.

Details

This function:

1. Extracts JSON data from the response body
2. Maps the "signature" elements from the response
3. Flattens the nested structure into a data frame
4. Removes the "PROBE" column (not needed for analysis)
5. Converts gene IDs to character format
6. Rounds numeric values to 12 decimal places for consistency
7. Adds signature metadata including L1000 status

The rounding ensures consistent precision across different platforms and prevents floating-point precision issues in downstream analyses.

Value

A tibble containing the signature data with standardized columns: * signatureID: The signature identifier * ID_geneid: Character gene identifiers * Name_GeneSymbol: Gene symbols * Value_LogDiffExp: Log fold-change values (rounded to 12 decimal places) * Significance_pvalue: P-values (rounded to 12 decimal places)

Examples

```
NULL
```

.processToL1000Signature *Process differential expression data into L1000 signature format*

Description

This internal function orchestrates the conversion of filtered differential expression data into the standardized L1000 signature format.

Usage

```
.processToL1000Signature(
  filteredData,
  geneColumn,
  logfcColumn,
  pvalColumn = NA
)
```

Arguments

filteredData	A dataframe containing filtered differential expression data.
geneColumn	Character string specifying the column name containing gene symbols.
logfcColumn	Character string specifying the column name containing log fold-change values.
pvalColumn	Character string specifying the column name containing p-values, or NA.

Details

This function dispatches to appropriate mapping functions based on whether p-value information is available:

1. `.mapToL1000WithPvalues` when p-value column is specified
2. `.mapToL1000WithoutPvalues` when p-value column is NA

Value

A tibble with the standardized L1000 signature format.

.returnLibrary *Return the internal iLINCS Library ID for a given library*

Description

This internal function maps user-friendly library names to the internal library identifiers used by the iLINCS API.

Usage

```
.returnLibrary(lib)
```

Arguments

lib	A character string specifying the library name. Must be one of "OE", "KD", or "CP".

Details

The mapping between user library names and iLINCS internal IDs is:

- "OE" (Overexpression) -> "LIB_11"
- "KD" (Knockdown) -> "LIB_6"
- "CP" (Chemical Perturbagen) -> "LIB_5"

The function validates the input library name before mapping and will stop execution if an invalid library is provided.

Value

A character string containing the corresponding iLINCS library ID.

See Also

[stopIfInvalidLibraries()] for library validation details

Examples

```
NULL
```

.returnResults *Return results in appropriate format based on input type*

Description

This internal function formats the output results to match the input signature type, ensuring consistent data type handling.

Usage

```
.returnResults(result, inputClass)
```

Arguments

result	A tibble containing the processed results from iLINCS API.
inputClass	A character vector containing the class of the original input signature (from .validateGetConcordantsInput).

Details

This function ensures that the output format matches the input format for consistency. If the original signature was provided as an S4Vectors::DataFrame, the results are converted back to DataFrame. Otherwise, results are returned as a tibble.

Value

The results in the appropriate format: * S4Vectors::DataFrame if input was a DataFrame * tibble otherwise (for data.frame, tibble inputs)

Examples

```
NULL
```

`.returnUserAgent` *Return a string suitable as a User-Agent for the iLINCS API*

Description

This internal function constructs a standardized User-Agent string for HTTP requests to the iLINCS API, including package name, version, and repository URL for identification and debugging purposes.

Usage

```
.returnUserAgent()
```

Details

The User-Agent string follows the format: "drugfindR/<current version>; https://github.com/CogDisResLab/d

This helps iLINCS administrators identify requests from this package and assists with debugging if issues arise. The version is automatically retrieved from the package metadata.

Value

A character string formatted as a User-Agent header value.

Examples

```
NULL
```

`.selectAndOrderResults` *Select and order consensus results columns*

Description

This internal function selects the relevant columns for consensus results and orders them appropriately for output.

Usage

```
.selectAndOrderResults(concordants)
```

Arguments

concordants	A dataframe containing processed concordants data.

Details

This function:

1. Selects standard consensus output columns
2. Orders results by descending absolute similarity
3. Handles both CP/KD libraries (with concentration) and OE libraries (without)

Value

A dataframe with selected and ordered columns for consensus output.

Examples

```
## Not run:
testData <- data.frame(
    signatureid = "SIG1",
    compound = "A",
    cellline = "A375",
    similarity = 0.8,
    sig_direction = "Up",
    pValue = 0.01
)
selected <- .selectAndOrderResults(testData)

## End(Not run)
```

.stopIfContainsMissingValues

Validate signature for missing values

Description

This internal function checks if the signature data frame contains any missing (NA) values, which are not allowed in iLINCS signature data.

Usage

```
.stopIfContainsMissingValues(signature)
```

Arguments

signature	A data.frame-like object containing signature data.

Details

The function scans the entire signature data frame for missing values. iLINCS requires complete data for all signature analysis, so any NA values will cause the function to stop with an informative error message indicating which columns contain missing values.

Value

Invisible NULL. The function throws an error if validation fails.

Examples

```
NULL
```

`.stopIfInvalidColNames` *Validate signature column names*

Description

This internal function checks if the signature data frame has the expected column names in the correct order for iLINCS compatibility.

Usage

```
.stopIfInvalidColNames(signature)
```

Arguments

`signature`	A data.frame-like object containing signature data.

Details

The function validates that the signature has exactly the following columns in the specified order:

1. `signatureID`: Signature identifier
2. `ID_geneid`: Gene ID
3. `Name_GeneSymbol`: Gene symbol
4. `Value_LogDiffExp`: Log fold-change expression value
5. `Significance_pvalue`: Statistical significance p-value

Value

Invisible NULL. The function throws an error if validation fails.

Examples

```
NULL
```

`.validateConsensusConcordantsInput` *Validate consensusConcordants input parameters*

Description

This internal function validates all input parameters for the consensusConcordants function to ensure they meet the required constraints.

Usage

```
.validateConsensusConcordantsInput(dots, paired, cutoff, cellLine)
```

Arguments

`dots`	A list of dataframes passed via ... parameter.
`paired`	Logical indicating whether paired analysis is requested.
`cutoff`	Numeric similarity cutoff value.
`cellLine`	Character vector of cell lines, or NULL.

Details

This function performs the following validations:

1. Ensures paired analysis has exactly two dataframes
2. Ensures unpaired analysis has exactly one dataframe
3. Validates cutoff is numeric and within reasonable range
4. Validates cellLine parameter format

Value

Invisible NULL. The function throws an error if validation fails.

Examples

```
## Not run:
# Valid calls (no errors)
testData <- data.frame(similarity = c(0.5, -0.3), compound = c("A", "B"))
.validateConsensusConcordantsInput(list(testData), FALSE, 0.3, NULL)
.validateConsensusConcordantsInput(list(testData, testData), TRUE, 0.3, "A375")

# Invalid calls (will throw errors)
.validateConsensusConcordantsInput(list(), FALSE, 0.3, NULL) # No data
.validateConsensusConcordantsInput(list(testData), TRUE, 0.3, NULL) # Paired needs 2 dataframes

## End(Not run)
```

.validateFilterSignatureInput

Validate filterSignature input parameters

Description

This internal function validates all input parameters for the filterSignature function to ensure they meet the required constraints and are mutually compatible.

Usage

```
.validateFilterSignatureInput(signature, direction, threshold, prop)
```

Arguments

signature	A data.frame-like object (data.frame, tibble, or DataFrame) containing the L1000 signature data.
direction	Character string specifying the filtering direction. Must be one of "up", "down", or "any".
threshold	Numeric value or vector specifying absolute threshold(s). Can be NULL, a single value, or a vector of two values. Cannot be specified together with prop.
prop	Numeric value specifying the proportion for quantile-based filtering. Must be between 0 and 1. Cannot be specified together with threshold.

Details

This function performs the following validations in order:

1. Ensures signature is a data.frame-like object
2. Validates direction is one of the allowed values
3. Verifies that only one of threshold or prop is specified
4. For threshold: checks length (1-2 values) and order (lower, higher)
5. For prop: checks it's a single value between 0 and 1

Value

Invisible NULL. The function throws an error if validation fails.

Examples

```
## Not run:
# Valid calls (no errors)
sig <- data.frame(Value_LogDiffExp = c(-2, -1, 0, 1, 2))
.validateFilterSignatureInput(sig, "any", 1.0, NULL)
.validateFilterSignatureInput(sig, "up", NULL, 0.1)
.validateFilterSignatureInput(sig, "down", c(-1.5, 1.0), NULL)

# Invalid calls (will throw errors)
.validateFilterSignatureInput(sig, "invalid", 1.0, NULL) # Invalid direction
.validateFilterSignatureInput(sig, "any", 1.0, 0.1) # Both threshold and prop
.validateFilterSignatureInput(sig, "any", NULL, NULL) # Neither threshold nor prop
```

```
.validateFilterSignatureInput(sig, "any", c(1, 2, 3), NULL) # Too many thresholds
.validateFilterSignatureInput(sig, "any", c(2, 1), NULL) # Wrong threshold order
.validateFilterSignatureInput(sig, "any", NULL, 1.5) # Proportion > 1
.validateFilterSignatureInput(sig, "any", NULL, -0.1) # Proportion < 0

## End(Not run)
```

`.validateGetConcordantsInput` *Validate getConcordants input parameters*

Description

This internal function validates the input parameters for the getConcordants function to ensure they meet the required constraints.

Usage

```
.validateGetConcordantsInput(signature, ilincsLibrary)
```

Arguments

`signature`	A data.frame-like object containing the signature data. Must have the required iLINCS signature structure with columns: signatureID, ID_geneid, Name_GeneSymbol, Value_LogDiffExp, Significance_pvalue.
`ilincsLibrary`	Character string specifying the iLINCS library to search. Must be one of "OE", "KD", or "CP".

Details

This function performs comprehensive validation:

1. Ensures `signature` is a data.frame-like object (data.frame, tibble, or S4Vectors::DataFrame)
2. Validates complete signature structure via `stopIfInvalidSignature()`
3. Validates `ilincsLibrary` is one of the supported libraries

The signature must conform to the iLINCS expected structure. Use `prepareSignature()` to ensure proper formatting.

Value

A character vector containing the class of the input signature. This is used internally to determine the return type format.

See Also

`[ stopIfInvalidSignature() ]` for signature structure validation, `[ prepareSignature() ]` for signature preparation

Examples

```
NULL
```

.validateGetSignatureInput *Validate getSignature input parameters*

Description

This internal function validates all input parameters for the getSignature function to ensure they meet the required constraints.

Usage

```
.validateGetSignatureInput(sigId)
```

Arguments

sigId	A character string containing the iLINCS signature ID to retrieve.

Details

This function performs the following validations:

- Ensures `sigId` is a character vector of length 1
- Ensures `sigId` is not empty or whitespace-only
- Validates that the signature exists in the metadata tables

Value

Invisible NULL. The function throws an error if validation fails.

Examples

```
NULL
```

.validateLibrary *Check if a single library is valid*

Description

This internal function validates whether a single library name is one of the supported iLINCS library types.

Usage

```
.validateLibrary(lib)
```

Arguments

lib	A character string containing a single library name to validate.

Details

Valid library names are:

- "CP": Chemical Perturbagen library
- "KD": Knockdown library
- "OE": Overexpression library

Value

A logical value: TRUE if the library is valid, FALSE otherwise.

Examples

```
NULL
```

.validatePrepareSignatureInput

Validate prepareSignature input parameters

Description

This internal function validates all input parameters for the prepareSignature function to ensure they meet the required constraints.

Usage

```
.validatePrepareSignatureInput(dge, geneColumn, logfcColumn, pvalColumn)
```

Arguments

dge	A dataframe-like object containing differential gene expression data.
geneColumn	Character string specifying the column name containing gene symbols.
logfcColumn	Character string specifying the column name containing log fold-change values.
pvalColumn	Character string specifying the column name containing p-values, or NA.

Details

This function performs the following validations:

1. Ensures all column names are character strings
2. Validates that specified columns exist in the input dataframe
3. Checks that the dataframe is not empty

Value

Invisible NULL. The function throws an error if validation fails.

Examples

```
NULL
```

`consensusConcordants` *Generate a Consensus list of Targets* **[Stable]**

Description

This function takes a list of (optionally split) concordance dataframes and returns a ranked list of gene or drug targets that have been chose for their maximal similarity to the signature

Usage

```
consensusConcordants(..., paired = FALSE, cutoff = 0.321, cellLine = NULL)
```

Arguments

`...`	One or Two (see paired) Data Frames with the concordants
`paired`	Logical indicating whether you split the dataframes by up and down regulated in prior analysis
`cutoff`	A similarity cutoff value. Defaults to 0.321
`cellLine`	A character vector of Cell Lines you are interested in.

Value

A tibble with the filtered and deduplicated results

Examples

```
# Create mock concordants data for demonstration
mockConcordants <- data.frame(
    signatureid = paste0("SIG", 1:10),
    treatment = c(
        "TP53", "TP53", "MYC", "MYC", "EGFR",
        "EGFR", "KRAS", "BRCA1", "BRCA1", "PIK3CA"
    ),
    cellline = c(
        "A375", "PC3", "A375", "MCF7", "A375",
        "PC3", "A375", "A375", "MCF7", "A375"
    ),
    time = rep("24H", 10),
    concentration = rep(NA, 10),
    sig_direction = rep("DOWN", 10),
    sig_type = rep("single", 10),
    similarity = c(
        0.85, 0.72, -0.68, -0.45, 0.55,
        0.38, 0.42, 0.51, 0.33, 0.29
    ),
    pValue = rep(0.001, 10)
)

# Example 1: Basic consensus with default cutoff
consensus <- consensusConcordants(mockConcordants)
nrow(consensus) # Targets with |similarity| >= 0.321

# Example 2: Consensus with higher cutoff
```

```
consensus_strict <- consensusConcordants(mockConcordants, cutoff = 0.5)
nrow(consensus_strict) # Fewer targets with higher threshold

# Example 3: Filter by cell line
consensus_A375 <- consensusConcordants(mockConcordants, cellLine = "A375")
unique(consensus_A375$CellLine) # Only A375


# Network-dependent examples using real iLINCS data
# Get the L1000 signature for LINCSKD_28
kdSignature <- getSignature("LINCSKD_28")

# Get concordant gene knockdown signatures
concordantSignatures <- getConcordants(kdSignature, ilincsLibrary = "KD")

# Get the consensus list with different parameters
consensus <- consensusConcordants(concordantSignatures, cutoff = 0.5)

# Paired analysis example
filteredUp <- filterSignature(kdSignature, direction = "up", threshold = 0.5)
filteredDown <- filterSignature(kdSignature, direction = "down", threshold = -0.5)
concordants_up <- getConcordants(filteredUp, ilincsLibrary = "KD")
concordants_down <- getConcordants(filteredDown, ilincsLibrary = "KD")
consensus <- consensusConcordants(concordants_up, concordants_down, paired = TRUE)
```

`filterSignature` *Filter the L1000 Signature* **[Stable]**

Description

This function filters the L1000 signature to a given threshold, identifying up-regulated, down-regulated, or both up- and down-regulated genes. The function supports both absolute threshold filtering and proportional filtering based on quantiles of the expression data.

Usage

```
filterSignature(signature, direction = "any", threshold = NULL, prop = NULL)
```

Arguments

`signature`	A data.frame, tibble, or DataFrame containing the L1000 signature. Must contain a column named "Value_LogDiffExp" with log fold-change values.
`direction`	Character string specifying the direction to filter. Must be one of "up" (up-regulated genes only), "down" (down-regulated genes only), or "any" (both up- and down-regulated genes). Defaults to "any".
`threshold`	Numeric value or vector specifying the log fold-change threshold(s). Can be: * A single positive value: Creates symmetric thresholds ($\pm threshold$) * A vector of two values: First value is the down-regulated threshold, second value is the up-regulated threshold Cannot be specified together with `prop`. One of `threshold` or `prop` must be provided.

`prop` Numeric value between 0 and 1 specifying the proportion of genes to select from the top and bottom of the expression distribution. For example, `prop = 0.1` selects the top 10% most up-regulated and bottom 10% most down-regulated genes. Cannot be specified together with `threshold`.

Details

The filtering process follows these steps:

1. Input validation: Checks data frame structure and parameter consistency
2. Threshold calculation: Computes filtering thresholds based on either absolute values (`threshold`) or quantiles (`prop`)
3. Direction-based filtering: Applies the computed thresholds according to the specified direction

When using `threshold`:

- Single value: Genes with |logFC| >= threshold are retained
- Two values: Genes with logFC <= `threshold[1]` OR logFC >= `threshold[2]`

When using `prop`:

- Thresholds are calculated as quantiles of the expression distribution
- Down threshold = quantile(logFC, prop)
- Up threshold = quantile(logFC, 1 - prop)

Value

A tibble containing the filtered L1000 signature with the same structure as the input but containing only genes that meet the filtering criteria.

See Also

`\link{getSignature}` for retrieving L1000 signatures from iLINCS, `\link{prepareSignature}` for preparing custom signatures, `\link{getConcordants}` for finding concordant signatures

Examples

```
# Create a mock signature for demonstration
mockSignature <- data.frame(
    signatureID = rep("MOCK001", 20),
    Name_GeneSymbol = paste0("GENE", 1:20),
    ID_geneid = 1:20,
    Value_LogDiffExp = c(
        -3.5, -2.8, -2.1, -1.5, -1.2, -0.8, -0.5, -0.3,
        -0.1, 0.1, 0.3, 0.6, 0.9, 1.2, 1.6, 2.0, 2.4, 2.9, 3.3, 3.8
    )
)

# Example 1: Filter by symmetric absolute threshold
# Keeps genes with |logFC| >= 1.5
filteredSymmetric <- filterSignature(mockSignature, threshold = 1.5)
nrow(filteredSymmetric) # Should return 8 genes

# Example 2: Filter by asymmetric absolute thresholds
# Keeps genes with logFC <= -2.0 OR logFC >= 2.5
```

```
filteredAsymmetric <- filterSignature(mockSignature, threshold = c(-2.0, 2.5))
nrow(filteredAsymmetric) # Should return 5 genes

# Example 3: Filter by proportion (top and bottom 20%)
filteredProportion <- filterSignature(mockSignature, prop = 0.2)
nrow(filteredProportion) # Should return 8 genes (4 up + 4 down)

# Example 4: Filter only up-regulated genes by threshold
upRegulated <- filterSignature(mockSignature, direction = "up", threshold = 1.0)
all(upRegulated$Value_LogDiffExp >= 1.0) # Should be TRUE

# Example 5: Filter only down-regulated genes by threshold
downRegulated <- filterSignature(mockSignature, direction = "down", threshold = 1.0)
all(downRegulated$Value_LogDiffExp <= -1.0) # Should be TRUE

# Network-dependent examples using real iLINCS data
# Get the L1000 signature for LINCSKD_28
kdSignature <- getSignature("LINCSKD_28")

# Filter for top 5% most extreme genes
topExtreme <- filterSignature(kdSignature, prop = 0.05)

# Get top 20% most up-regulated genes
topUpregulated <- filterSignature(kdSignature, direction = "up", prop = 0.2)
```

`getConcordants` *Get concordant signatures from iLINCS database* **[Stable]**

Description

This function queries the iLINCS (Integrative Library of Integrated Network-based Cellular Signatures) database to find signatures that are concordant (similar) to a given input signature.

Usage

```
getConcordants(signature, ilincsLibrary = "CP")
```

Arguments

`signature`	A data.frame, tibble, or S4Vectors::DataFrame containing the signature data. Must conform to iLINCS signature structure with columns: * `signatureID`: Signature identifier * `ID_geneid`: Gene IDs * `Name_GeneSymbol`: Gene symbols * `Value_LogDiffExp`: Log fold-change values * `Significance_pvalue`: Statistical significance p-values Use `prepareSignature()` to ensure proper formatting.
`ilincsLibrary`	Character string specifying the iLINCS library to search. Must be one of: * `"CP"`: Chemical Perturbagen library (default) * `"KD"`: Knockdown library * `"OE"`: Overexpression library

Details

The function performs the following steps:

1. Validates input parameters
2. Creates a temporary file with signature data
3. Detects signature direction from expression values
4. Sends a multipart POST request to the iLINCS API
5. Processes the JSON response into a standardized tibble
6. Cleans up temporary files

The signature direction is determined as follows:

- `"Up"`: All expression values are greater than or equal to zero
- `"Down"`: All expression values are less than or equal to zero
- `"Any"`: Mixed positive and negative values

Value

A data structure containing concordant signatures. The return type matches the input signature type: * tibble for data.frame or tibble inputs * S4Vectors::DataFrame for DataFrame inputs

Contains the following columns: * `signatureid`: Unique signature identifier * `compound` or `treatment`: Drug/treatment name * `concentration`: Drug concentration (CP library only) * `time`: Treatment duration * `cellline`: Cell line used * `similarity`: Similarity score (rounded to 8 decimal places) * `pValue`: Statistical significance (rounded to 20 decimal places) * `sig_direction`: Signature direction ("Up", "Down", or "Any")

API Details

This function interfaces with the iLINCS web service API. The signature is uploaded as a tab-separated file and analyzed against the specified library. Results are returned as JSON and parsed into a tibble.

Error Handling

The function will stop execution with informative error messages for:

- Invalid signature data types (must be data.frame, tibble, or DataFrame)
- Invalid signature structure (missing required columns, wrong order, etc.)
- Missing values in signature data
- Unsupported iLINCS library names
- HTTP errors from the iLINCS API
- Invalid or empty API responses

References

iLINCS Portal: http://www.ilincs.org/

Pilarczyk et al. (2020). Connecting omics signatures and revealing biological mechanisms with iLINCS. Nature Communications, 11(1), 4058.

See Also

[prepareSignature()] for signature preparation, [filterSignature()] for signature filtering, [investigateSignature()] for signature investigation

Examples

```
# Input validation examples (no API calls)
# These demonstrate proper signature structure
mockSig <- data.frame(
    signatureID = rep("TEST", 3),
    ID_geneid = c("123", "456", "789"),
    Name_GeneSymbol = c("TP53", "MYC", "EGFR"),
    Value_LogDiffExp = c(1.5, -2.0, 0.8)
)

# Validate library parameter (should produce error)
tryCatch(
    getConcordants(mockSig, ilincsLibrary = "INVALID"),
    error = function(e) message("Expected error: invalid library")
)

# This example requires network access to the iLINCS API

# Load example differential expression data
dge_file <- system.file("extdata", "dCovid_diffexp.tsv",
    package = "drugfindR"
)
dge_data <- read.delim(dge_file)

# Prepare signature to ensure proper structure
signature <- prepareSignature(
    dge_data[1:50, ],
    geneColumn = "hgnc_symbol",
    logfcColumn = "logFC",
    pvalColumn = "PValue"
)

# Find concordant chemical perturbagens
cpConcordants <- getConcordants(signature, ilincsLibrary = "CP")
head(cpConcordants)

# Find concordant knockdown signatures
kdConcordants <- getConcordants(signature, ilincsLibrary = "KD")
head(kdConcordants)

# Find concordant overexpression signatures
oeConcordants <- getConcordants(signature, ilincsLibrary = "OE")
head(oeConcordants)

# Works with different data frame types
signatureDf <- as.data.frame(signature)
cpConcordantsDf <- getConcordants(signatureDf, "CP")

# Works with S4Vectors::DataFrame
signatureDataFrame <- S4Vectors::DataFrame(signature)
cpConcordantsDataFrame <- getConcordants(signatureDataFrame, "CP")
# Returns S4Vectors::DataFrame to match input type
```

getSignature *Get the L1000 Signature from iLINCS* **[Stable]**

Description

This function acts as the entrypoint to the iLINCS database. This takes in an ID and returns the signature after making a call to the iLINCS database. The function automatically detects whether the signature is an L1000 signature based on the signature ID and metadata tables, and retrieves all available genes for comprehensive signature analysis.

Usage

```
getSignature(sigId)
```

Arguments

sigId	character. The ilincs signature_id

Value

a tibble with the signature data containing the following columns: * `signatureID`: The signature identifier * `ID_geneid`: Gene IDs (Entrez) * `Name_GeneSymbol`: Gene symbols * `Value_LogDiffExp`: Log fold-change values * `Significance_pvalue`: Statistical significance p-values

Examples

```
# Input validation example (no API call)
# Demonstrates proper signature ID format validation
tryCatch(
    getSignature(""), # Empty string should error
    error = function(e) message("Expected error: empty signature ID")
)


# These examples require network access to the iLINCS API

# Get the L1000 signature for LINCSKD_28
kdSignature <- getSignature("LINCSKD_28")
head(kdSignature)

# Get an overexpression signature (L1000 status is automatically detected)
oeSignature <- getSignature("LINCSOE_1000")
head(oeSignature)

# Check the structure of retrieved signature
str(kdSignature)
```

`investigateSignature` *Investigate a given DGE dataset* **[Stable]**

Description

This function takes a DGE Data frame and then finds concordant signatures to that. This generates an L1000 signature from the DGE dataset and then uploads that signature to iLINCS to find the relevant concordant (or discordant) signatures

Usage

```
investigateSignature(
  expr,
  outputLib,
  filterThreshold = NULL,
  filterProp = NULL,
  similarityThreshold = 0.2,
  paired = TRUE,
  outputCellLines = NULL,
  geneColumn = "Symbol",
  logfcColumn = "logFC",
  pvalColumn = "PValue",
  sourceName = "Input",
  sourceCellLine = NA,
  sourceTime = NA,
  sourceConcentration = NA
)
```

Arguments

`expr`	A dataframe that has differential gene expression analysis
`outputLib`	The library to search
`filterThreshold`	The Filtering threshold.
`filterProp`	The Filtering proportion.
`similarityThreshold`	The Similarity Threshold
`paired`	Logical. Whether to query iLINCS separately for up and down regulated genes
`outputCellLines`	A character vector of cell lines to restrict the output search to.
`geneColumn`	The name of the column that has gene symbols
`logfcColumn`	The name of the column that has log_2 fold-change values
`pvalColumn`	The name of the column that has p-values
`sourceName`	(Optional) An annotation column to identify the signature by name
`sourceCellLine`	(Optional) An annotation column to specify the cell line for the input data
`sourceTime`	(Optional) An annotation column to specify the time for the input data
`sourceConcentration`	(Optional) An annotation column to specify the concentration for the input data

Value

A tibble with the the similarity scores and signature metadata

Examples

```
# Input validation example (no API calls)
mockExpr <- data.frame(
    Symbol = c("TP53", "MYC"),
    logFC = c(2.5, -1.8),
    PValue = c(0.001, 0.01)
)

# Validate library parameter (should produce error)
tryCatch(
    investigateSignature(mockExpr, outputLib = "INVALID"),
    error = function(e) message("Expected error: invalid library")
)


# This function makes multiple API calls to iLINCS and may take several minutes

# Load differential expression data
inputSignature <- read.table(
    system.file("extdata", "dCovid_diffexp.tsv", package = "drugfindR"),
    header = TRUE
)

# Investigate the signature against chemical perturbagen library
investigatedSignature <- investigateSignature(
    inputSignature,
    outputLib = "CP",
    filterThreshold = 0.5,
    geneColumn = "hgnc_symbol",
    logfcColumn = "logFC",
    pvalColumn = "PValue"
)
head(investigatedSignature)
```

`investigateTarget` *Investigate concordant signatures for a gene or drug* **[Stable]**

Description

Given the name of a target (gene knockdown/overexpression or compound) this high–level convenience wrapper:

Usage

```
investigateTarget(
  target,
  inputLib,
  outputLib,
```

```
  filterThreshold = 0.85,
  similarityThreshold = 0.321,
  paired = TRUE,
  inputCellLines = NULL,
  outputCellLines = NULL
)
```

Arguments

target	Character scalar. Gene symbol (for KD/OE libraries), or drug / compound name (for CP library) used to locate source signatures.
inputLib	Character ("OE", "KD", or "CP"). Library from which source signatures for target are drawn.
outputLib	Character ("OE", "KD", or "CP"). Library queried for concordant signatures.
filterThreshold	Numeric in \(0,1]. Minimum absolute (or directional) change used to retain genes in each source signature prior to concordance. Default 0.85 is conservative; consider lowering (e.g. 0.5) for broader coverage.
similarityThreshold	Numeric in [0,1]. Minimum similarity score retained in the final consensus result set. Default 0.321 (~ upper third).
paired	Logical. If TRUE (default) computes concordance separately for up and down regulated gene sets; if FALSE uses all selected genes together.
inputCellLines	Optional character vector restricting the search for source signatures to specified cell line(s). If NULL all available are considered.
outputCellLines	Optional character vector restricting target signatures (during consensus formation) to specified cell line(s). If NULL all are considered.

Details

1. Locates iLINCS source signatures for the target in the specified input library.
2. Optionally filters by source cell line(s).
3. Retrieves each source signature and filters genes by direction and magnitude.
4. Queries iLINCS for concordant signatures in the chosen output library.
5. Computes paired or unpaired consensus concordance across up/down regulated sets.
6. Returns an augmented tibble of similarity scores and rich source/target metadata.

The paired workflow evaluates concordance separately for up- and down-regulated genes and then combines (via consensusConcordants()) the two result sets. When paired = FALSE a single aggregate signature (direction = "any") is used.

Network access: This function performs remote API calls (unless tests are run under a mocking context such as httptest2::with_mock_api()). Examples are wrapped in \donttest{} to avoid false negatives on CRAN / Bioconductor builders without network access.

Errors are raised if:

- No source signatures match target in the requested inputLib (empty set).
- Invalid library codes are supplied.

Internally this function orchestrates: getSignature(), filterSignature(), getConcordants() and consensusConcordants(). It returns a vertically concatenated result across all matching source signatures.

Value

A tibble (data frame) with one row per consensus concordant target signature. Typical columns include:

- `Source` / `Target` – gene or compound names.
- `Similarity` – numeric concordance score in [-1,1].
- `SourceSignature`, `TargetSignature` – iLINCS signature identifiers.
- `SourceCellLine`, `TargetCellLine` – originating cell lines (if applicable).
- `SourceConcentration`, `TargetConcentration` – dosing information for CP.
- `SourceTime`, `TargetTime` – time point metadata.

Thresholds

- `filterThreshold` controls gene selection within each source signature. It is passed to `filterSignature()` as the absolute (or directional) threshold.
- `similarityThreshold` is applied when forming the consensus concordants to discard low similarity entries.

See Also

`getSignature()`, `filterSignature()`, `getConcordants()`, `consensusConcordants()`, `prepareSignature()` for lower-level operations.

Examples

```
# Input validation examples (no API calls)
# Demonstrate library parameter validation
tryCatch(
    investigateTarget(target = "TP53", inputLib = "INVALID", outputLib = "CP"),
    error = function(e) message("Expected error: invalid inputLib")
)

tryCatch(
    investigateTarget(target = "TP53", inputLib = "KD", outputLib = "INVALID"),
    error = function(e) message("Expected error: invalid outputLib")
)


# This function makes multiple API calls to iLINCS and may take several minutes
# Basic paired investigation of a knockdown signature against compound library
set.seed(1)
res <- investigateTarget(
    target = "AATK",
    inputLib = "KD",
    outputLib = "CP",
    filterThreshold = 0.5,
    similarityThreshold = 0.3,
    paired = TRUE
)
head(res)

# Unpaired (aggregate) workflow — often faster, returns a single consensus set
res_unpaired <- investigateTarget(
    target = "AATK", inputLib = "KD", outputLib = "CP",
```

```
    filterThreshold = 0.5, similarityThreshold = 0.3, paired = FALSE
)
head(res_unpaired)

# Restrict source signatures to specific cell lines (if available)
# and target signatures to a subset of cell lines during consensus
res_filtered <- investigateTarget(
    target = "AATK", inputLib = "KD", outputLib = "CP",
    outputCellLines = c("MCF7"),
    filterThreshold = 0.5, similarityThreshold = 0.3
)
head(res_filtered)

# Using httptest2 (if installed) to mock network calls:
# httptest2::with_mock_api({
#   mock_res <- investigateTarget("AATK", "KD", "CP", filterThreshold = 0.5)
#   print(head(mock_res))
# })
```

`prepareSignature` *Prepare an L1000 Signature from a given differential gene expression output* **[Stable]**

Description

This function takes a differential gene expression output from any pipeline like edgeR or DeSeq2 or any that give you the gene symbol, log_2 fold-change and p-value and transforms that into an L1000 signature for later processing.

Usage

```
prepareSignature(
  dge,
  geneColumn = "Symbol",
  logfcColumn = "logFC",
  pvalColumn = "PValue"
)
```

Arguments

`dge`	A dataframe-like object that has the differential gene expression information
`geneColumn`	The name of the column that has gene symbols
`logfcColumn`	The name of the column that has log_2 fold-change values
`pvalColumn`	The name of the column that has p-values

Value

A tibble with the L1000 signature.

Examples

```
# Load example differential expression data from package
dge_file <- system.file("extdata", "dCovid_diffexp.tsv",
    package = "drugfindR"
)
dge_data <- read.delim(dge_file)

# Prepare signature with p-values (standard workflow)
signature <- prepareSignature(
    dge_data,
    geneColumn = "hgnc_symbol",
    logfcColumn = "logFC",
    pvalColumn = "PValue"
)
head(signature)

# Prepare signature without p-values
signature_no_pval <- prepareSignature(
    dge_data,
    geneColumn = "hgnc_symbol",
    logfcColumn = "logFC",
    pvalColumn = NA
)
head(signature_no_pval)

# Custom column names example
custom_dge <- data.frame(
    Gene = c("TP53", "MYC", "BRCA1", "EGFR"),
    FC = c(2.5, -1.8, 3.2, -2.1),
    Pval = c(0.001, 0.01, 0.0001, 0.005)
)

custom_signature <- prepareSignature(
    custom_dge,
    geneColumn = "Gene",
    logfcColumn = "FC",
    pvalColumn = "Pval"
)
print(custom_signature)
```

---

stopIfInvalidLibraries

Stop if the libraries are invalid

Description

This internal function validates library specifications and stops execution with an informative error message if any invalid libraries are found.

Usage

```
stopIfInvalidLibraries(libs)
```

Arguments

libs	A character vector of library names to validate. Each library must be one of "OE", "KD", or "CP".

Details

This function validates that all provided library names are supported:

- "OE": Overexpression library (LIB_11)
- "KD": Knockdown library (LIB_6)
- "CP": Chemical Perturbagen library (LIB_5)

If any invalid libraries are found, the function provides a detailed error message listing the invalid libraries and the expected options.

Value

Invisible NULL. The function throws an error if validation fails.

See Also

[validateLibraries()] for the underlying validation logic, [.validateLibrary()] for single library validation

Examples

```
NULL
```

---

stopIfInvalidSignature

Validate signature data structure and content

Description

This function performs comprehensive validation of signature data to ensure it meets the requirements for iLINCS analysis.

Usage

```
stopIfInvalidSignature(signature)
```

Arguments

signature	A data.frame-like object containing signature data that should be validated for iLINCS compatibility.

Details

This function performs two main validation checks:

1. Column structure validation via .stopIfInvalidColNames()
2. Missing value validation via [.stopIfContainsMissingValues()]

The signature must have exactly the required columns in the correct order and cannot contain any missing (NA) values.

Value

Invisible NULL. The function throws an error if validation fails.

See Also

`[ prepareSignature() ]` for preparing signatures that meet these requirements, `[ .stopIfInvalidColNames() ]` for column validation details, `[ .stopIfContainsMissingValues() ]` for missing value validation details

Examples

```
NULL
```

`targetRename` *Rename target-related columns to user-facing output names*

Description

Standardizes internal concordants/consensus column names to the user-facing output schema used by this package.

Usage

```
targetRename(inputNames)
```

Arguments

`inputNames`	Character vector of column names to rename. See Details for the expected input ordering and mapping.

Details

Expected input columns (by position) are the internal concordants fields:

1. `signatureid`, 2) `treatment` (or compound pre-renaming),
2. `cellline`, 4) `time`, 5) `concentration`, 6) `sig_direction`,
3. `sig_type`, 8) `similarity`, 9) `pValue`.

These are mapped to the user-facing names returned by functions like `consensusConcordants()` and downstream investigation helpers:

- `TargetSignature`, `Target`, `TargetCellLine`, `TargetTime`, `TargetConcentration`, `InputSigDirection`, `SignatureType`, `Similarity`, `pValue`.

Only the names are returned; the renaming is applied via `dplyr::rename_with(x, targetRename)`.

Value

Character vector of output (renamed) column names.

Examples

```
NULL
```

validateLibraries *Check if multiple libraries are valid*

Description

This function validates whether all provided library names are supported iLINCS library types.

Usage

```
validateLibraries(libs)
```

Arguments

libs	A character vector of library names to validate.

Details

This function uses `.validateLibrary()` to check each library individually and returns TRUE only if all libraries are valid. It's used internally to validate library parameters before API calls.

Value

A logical value: TRUE if all libraries are valid, FALSE if any are invalid.

See Also

`[ .validateLibrary() ]` for single library validation, `[ stopIfInvalidLibraries() ]` for validation with error handling

Examples

```
NULL
```

Index

.applyDirectionFilter, 5
.applySimilarityCutoff, 6
.applyTargetRenaming, 7
.calculateAbsoluteThresholds, 8
.calculateDoubleThreshold, 9
.calculateProportionalThreshold, 10
.calculateSingleThreshold, 11
.cleanupGetConcordants, 12
.combineConcordantsData, 12
.computeConsensusFromSignature, 13
.createSignatureRequest, 14
.detectSignatureDirection, 15
.executeIlincsRequest, 16
.executeSignatureRequest, 17
.filterByCellLine, 18
.generateIlincsRequest, 18
.groupByTargetAndSelectMax, 19
.ilincsBaseUrl, 20
.isValidSignatureId, 20
.loadMetadata, 21
.mapToL1000WithPvalues, 22
.mapToL1000WithoutPvalues, 22
.prepareSignatureFile, 23
.processConsensusPipeline, 23
.processIlincsResponse, 24
.processIlincsResponseEmpty, 25
.processIlincsResponseError, 26
.processIlincsResponseSuccess, 26
.processSignatureResponse, 27
.processSignatureResponseError, 27
.processSuccessfulResponse, 28
.processToL1000Signature, 29
.returnLibrary, 29
.returnResults, 30