

Package: MSstatsBioNet (via r-universe)

October 23, 2026

Type Package

Title Network Analysis for MS-based Proteomics Experiments

Version 1.5.5

Description A set of tools for network analysis using mass spectrometry-based proteomics data and network databases. The package takes as input the output of MSstats differential abundance analysis and provides functions to perform enrichment analysis and visualization in the context of prior knowledge from past literature. Notably, this package integrates with INDRA, which is a database of biological networks extracted from the literature using text mining techniques.

License file LICENSE

Depends R (>= 4.4.0), MSstats

Imports httr, jsonlite, r2r, tidyr, htmlwidgets, grDevices, stats, text2vec, stopwords, xml2, rentrez, methods

Suggests data.table, BiocStyle, knitr, rmarkdown, testthat (>= 3.1.7), mockery, MSstatsConvert, shiny

VignetteBuilder knitr

biocViews ImmunoOncology, MassSpectrometry, Proteomics, Software, QualityControl, NetworkEnrichment, Network

Encoding UTF-8

URL <http://msstats.org>, <https://vitek-lab.github.io/MSstatsBioNet/>

BugReports <https://groups.google.com/forum/#!forum/msstats>

Config/testthat/edition 3

Config/roxygen2/version 8.1.0

Config/pak/sysreqs cmake make libicu-dev libuv1-dev libxml2-dev libssl-dev

Repository <https://bioc.r-universe.dev>

Date/Publication 2026-10-23 14:27:02 UTC

RemoteUrl <https://github.com/bioc/MSstatsBioNet>

RemoteRef HEAD

RemoteSha a9e97749a7cd72fd6533b0194c4fcf6d4f4b0058

Contents

.validateAnnotateProteinInfoFromIndraInput	3
add_entity_values	3
annotateProteinInfoFromIndra	5
backend_capabilities	6
bootstrapTopicModels	7
compareTopicModels	9
convert_ids	11
cytoscapeNetwork	12
cytoscapeNetworkOutput	13
decomposeSubnetworkByTopic	14
decomposeSubnetworkIntoHierarchicalTopics	17
deleteEdgeFromNetwork	20
exportNetworkToHTML	20
filter_by_curation	21
filter_by_ptm_site	23
filterSubnetworkByContext	24
get_curations	26
get_entity_properties	27
get_evidence	28
get_network	29
getSubnetworkFromIndra	32
indra_backend	34
indra_evidence_sources	35
load_network	36
merge_networks	37
network_queries	39
NetworkBackend-class	40
NetworkQuery-class	40
prepare_entities	41
previewNetworkInBrowser	42
print.topicHierarchy	43
renderCytoscapeNetwork	44
save_network	45
select_entities	46
subnetwork_query	47
validate_network	48

Index

50

```
.validateAnnotateProteinInfoFromIndraInput
```

Validate Annotate Protein Info Input

Description

Validate Annotate Protein Info Input

Usage

```
.validateAnnotateProteinInfoFromIndraInput(df, proteinIdType)
```

Arguments

<code>df</code>	A data frame containing protein information.
<code>proteinIdType</code>	The <code>proteinIdType</code> supplied by the caller.

Value

None. Throws an error if validation fails.

<code>add_entity_values</code>	<i>Copy the values of an entity table onto a network's nodes</i>
--------------------------------	--

Description

Sets `logFC` and `adj.pvalue` of the nodes that match a row of entities, without querying a back-end. Use it to show the results of another experiment on a network: for example, a network built from `MSstatsPTM`'s `ADJUSTED.Model`, with its protein nodes colored by the global profiling results (`PROTEIN.Model`).

Usage

```
add_entity_values(network, entities, overwrite = FALSE)
```

Arguments

<code>network</code>	list of nodes and edges from <code>get_network()</code>
<code>entities</code>	entity table from <code>prepare_entities()</code> , grounded by <code>convert_ids()</code> . Its <code>included_in_query</code> column is ignored.
<code>overwrite</code>	whether to replace the <code>logFC</code> and <code>adj.pvalue</code> of node rows that already have them. <code>FALSE</code> (default) stops and names those nodes.

Details

A node row matches an entity row with the same node id and site (for a PTM site row, its parent protein and site; for a protein, site = NA). Node rows that match no entity row by id are then matched by grounding (namespace and entity_id) and site. This finds nodes whose id is the backend's name, such as a kinase added with `get_network(include_entities=)` that INDRA calls "SRC", while the entity table has "P12931". An entity row that matched a node by id is not used again by grounding, so an isoform node doesn't take the values of its canonical protein. A node row that matches several entity rows gets no values, with a message.

A protein that has only PTM site rows in the network, and matches a protein-level entity row, gets a new protein-level row (site = NA) with that row's values, so `cytoscapeNetwork()` colors the protein node by it.

Matched node rows get `measured = TRUE`. Rows of entities that match no node are not added: `get_network()` decides which nodes are in the network. Edges and the other elements of network are unchanged. The node table does not record which experiment the values come from.

Value

network, with the values of entities on the matching nodes

See Also

`filter_by_ptm_site()`, `merge_networks()`

Examples

```
# A PTM network: phosphorylation of two EGFR sites by SRC, which is not
# in the PTM results, so it has INDRA's name as its id
network <- list(
  nodes = data.frame(
    id = c("P00533", "P00533", "SRC"),
    entity_type = c("ptm_site", "ptm_site", "protein"),
    entity_name = c("EGFR", "EGFR", "SRC"),
    namespace = "HGNC", entity_id = c("3236", "3236", "11283"),
    measured = c(TRUE, TRUE, FALSE), included_in_query = TRUE,
    node_role = c("passed_cutoffs", "passed_cutoffs", "user_added"),
    site = c("S1039", "Y1092", NA),
    has_measured_sites = c(TRUE, TRUE, FALSE),
    logFC = c(1.5, -0.4, NA), adj.pvalue = c(0.01, 0.3, NA)),
  edges = data.frame(
    source = "SRC", target = "P00533", interaction = "Phosphorylation",
    directed = TRUE, site = "Y1092", confidence = 0.9,
    evidence_count = 3L, evidence_url = "https://example.org/1",
    statement_id = "1", backend_database = "INDRA",
    query_type = "subnetwork")
)
# Global profiling results, grounded (convert_ids() would fill these)
global <- prepare_entities(
  data.frame(Protein = c("P00533", "P12931"), log2FC = c(0.2, 1.1),
    adj.pvalue = c(0.6, 0.02)),
  entity_type = "protein", id_type = "uniprot")
```

```
global$namespace <- "HGNC"
global$entity_id <- c("3236", "11283")
global$entity_name <- c("EGFR", "SRC")
network <- add_entity_values(network, global)
network$nodes[, c("id", "site", "measured", "logFC")]
```

annotateProteinInfoFromIndra

Annotate Protein Information from Indra

Description

This function standardizes entity identifiers from protein, compound, or gene inputs to a unified namespace using ID conversion from INDRA cogex or Gilda grounding.

Usage

```
annotateProteinInfoFromIndra(df, proteinIdType)
```

Arguments

df	output of groupComparison function's comparisonResult table. Must contain a Protein column whose values are interpreted according to proteinIdType. A value may name a protein group – several identifiers for the same quantified analyte joined by ";", e.g. "P13747;P23132" – in which case every member is grounded independently and the results are pooled onto the row.
proteinIdType	A character string specifying the type of analyte identifier in the Protein column. One of "Uniprot", "Uniprot_Mnemonic", "Hgnc_Name", or "Metabolite". The "Metabolite" value treats inputs as metabolite names and grounds them through Gilda, keeping whatever namespace Gilda returns (CHEBI / PUBCHEM / ChEMBL / ...).

Value

A data frame with the following columns:

Protein Character. The original identifier from the input.

GlobalProtein Character. The input identifier without the PTM site suffix (typically _<amino acid><site number>, e.g. _S148) stripped from each protein group member, used as the grounding key. NA when the input holds no usable identifier.

UniprotId Character. The Uniprot ID of the protein, semicolon-joined in the case of multiple proteins, or NA for "Hgnc_Name" and "Metabolite" inputs.

EntityNamespace Character. The grounding namespace (e.g. "HGNC", "CHEBI"). When a row grounds to multiple candidates – whether from a protein group or from an ambiguous single input – namespaces are semicolon-joined and positionally aligned with EntityId and EntityName.

EntityId Character. The bare grounding identifier within its namespace (e.g. "1097" for HGNC, "28748" for CHEBI). Semicolon-joined when multi-grounded.

EntityName Character. The canonical display name from the grounding source. Semicolon-joined when multi-grounded, with "NA" in the positions whose name lookup failed.

IsTranscriptionFactor Logical. NA for proteinIdType == "Metabolite" and for multi-grounded rows.

IsKinase Logical. NA for proteinIdType == "Metabolite" and for multi-grounded rows.

IsPhosphatase Logical. NA for proteinIdType == "Metabolite" and for multi-grounded rows.

Examples

```
df <- data.frame(Protein = c("CLH1_HUMAN"))
annotated_df <- annotateProteinInfoFromIndra(df, "Uniprot_Mnemonic")
head(annotated_df)
```

backend_capabilities *What a backend supports*

Description

Lists the queries, identifier conversions, entity properties, and limits of a backend, so a choice can be checked before a query is sent.

Usage

```
backend_capabilities(backend)

## S4 method for signature 'NetworkBackend'
backend_capabilities(backend)

## S4 method for signature 'IndraBackend'
backend_capabilities(backend)
```

Arguments

backend a NetworkBackend, e.g. from [indra_backend\(\)](#)

Value

named list:

query_types the query_type values of the queries the backend answers, e.g. "subnetwork" for [subnetwork_query\(\)](#)

id_conversions for each entity_type, the id_type values [convert_ids\(\)](#) can convert

entity_properties the properties [get_entity_properties\(\)](#) can add

interaction_types the edges\$interaction values the backend returns

evidence_sources the evidence_sources values [get_network\(\)](#) filters on, as a list with database and text_mined elements, e.g. [INDRA_DATABASE_SOURCES](#)

max_nodes for each query type, the largest number of groundings one query can take

See Also

[network_queries](#)

Examples

```
backend_capabilities(indra_backend())
```

bootstrapTopicModels *Bootstrap the topic decomposition to find each topic's robust top words*

Description

Refits the NMF topic model on many bootstrap resamples of the papers and reports, for every topic, how reliably each word stays among the topic's top terms. This separates words that genuinely characterise a topic from words that only surface in a single lucky fit, and lets you see how the top-word lists change with and without the PPI view (run once per include_ppi).

Usage

```
bootstrapTopicModels(
  subnetwork,
  n_boot = 50,
  n_topics = 5,
  include_ppi = TRUE,
  n_top_terms = 10,
  min_term_count = 2,
  max_iter = 200,
  tol = 1e-04,
  seed = 1,
  backend = NULL
)
```

Arguments

subnetwork	list with nodes and edges data.frames, e.g. the output of getSubnetworkFromIndra .
n_boot	number of bootstrap resamples. Default 50.
n_topics	number of topics (rank of the factorization). Default 5.
include_ppi	logical; factorize the PPI/edge view jointly with the text (TRUE, default) or use paper words only (FALSE). See decomposeSubnetworkByTopic .
n_top_terms	number of top words that define a topic's "top list" in each resample (the cutoff for the selection-frequency tally). Default 10.

<code>min_term_count</code>	minimum corpus frequency for a word to be kept when building the text matrix. Default 2.
<code>max_iter</code>	maximum number of NMF multiplicative-update iterations. Default 200.
<code>tol</code>	relative-change tolerance for NMF early stopping. Default 1e-4.
<code>seed</code>	random seed for the reference fit, the resampling, and each bootstrap NMF. Default 1.
<code>backend</code>	the backend to get the evidence from, e.g. <code>indra_backend()</code> . NULL (default) uses the default backend named in each edge's <code>backend_database</code> . See <code>get_evidence()</code> .

Details

Because topic indices are arbitrary across fits (label switching), each resample's topics are first aligned to a reference fit on the full data by cosine similarity of their topic-word vectors. The papers are resampled with replacement; the word vocabulary is held fixed (from the full data) so topics remain comparable across resamples, and the NMF seed is held fixed so the variability reported reflects *data* resampling rather than random initialization.

Value

A list with

include_ppi, n_boot, n_topics the settings used.

topTerms named list `topic_1 ... topic_k`. Each is a data.frame sorted by `selection_freq`, with columns `term`, `selection_freq` (fraction of resamples the word was in this topic's top `n_top_terms`), and `mean_weight` (mean within-topic word weight across resamples). A word with `selection_freq` near 1 is a stable signature of the topic.

reference named list of the top `n_top_terms` words per topic from the single full-data fit, for comparison.

Note

Beta feature: This function is experimental and the API may change without notice in future versions.

See Also

[decomposeSubnetworkByTopic](#), [compareTopicModels](#)

Examples

```
## Not run:
input <- data.table::fread(system.file(
  "extdata/groupComparisonModel.csv",
  package = "MSstatsBioNet"
))
subnetwork <- getSubnetworkFromIndra(input)

# Top words with PPIs included vs. words only:
boot_ppi <- bootstrapTopicModels(subnetwork, include_ppi = TRUE)
```

```

boot_text <- bootstrapTopicModels(subnetwork, include_ppi = FALSE)

head(boot_ppi$topTerms$topic_1)      # robust signature words for topic 1
boot_text$topTerms$topic_1

## End(Not run)

```

compareTopicModels	<i>Test whether including PPIs changes topic structure beyond random chance</i>
--------------------	---

Description

Quantifies how much the topic decomposition produced by [decomposeSubnetworkByTopic](#) changes when the PPI/edge view is included (`include_ppi = TRUE`) versus excluded (`include_ppi = FALSE`), and separates that change from the run-to-run variability that NMF produces just from its random initialization.

Usage

```

compareTopicModels(
  subnetwork,
  seeds = seq_len(20),
  n_topics = 5,
  unit = c("edges", "papers"),
  min_term_count = 2,
  max_iter = 200,
  tol = 1e-04,
  backend = NULL
)

```

Arguments

subnetwork	list with nodes and edges data.frames, e.g. the output of getSubnetworkFromIndra .
seeds	integer vector of NMF seeds to fit (at least 2). Default 1:20.
n_topics	number of topics (rank of the factorization). Default 5.
unit	either "edges" (compare edge-to-topic assignments, the default, matching the subnetworks the decomposition returns) or "papers" (compare paper-to-topic assignments).
min_term_count	minimum corpus frequency for a word to be kept when building the text matrix. Default 2.
max_iter	maximum number of NMF multiplicative-update iterations. Default 200.
tol	relative-change tolerance for NMF early stopping. Default 1e-4.
backend	the backend to get the evidence from, e.g. indra_backend() . NULL (default) uses the default backend named in each edge's backend_database. See get_evidence() .

Details

NMF converges to a local optimum that depends on the random seed, so a single joint-vs-text comparison conflates the real effect of the PPI view with optimization noise. This function instead refits both modes across many seeds and compares three distributions of partition agreement (Adjusted Rand Index, ARI):

within_joint ARI between pairs of joint runs (different seeds) — how much the joint solution wobbles on its own.

within_text ARI between pairs of text-only runs — the same for the text-only solution.

between ARI between the joint and text-only run at the *same* seed. Because both modes draw `W` and `H_text` from the same seeded stream, a matched seed gives both modes an identical initialization, so this isolates the effect of adding the PPI view from the starting point.

If the between-mode ARI is systematically lower than the within-mode ARIs, the PPI view changes the topic structure more than chance would — a one-sided Wilcoxon rank-sum test (between < within) puts a p-value on it. If the between distribution sits inside the within distributions, the apparent difference is just optimization noise.

The expensive, network-bound steps (evidence extraction, abstract fetching, matrix construction) run once; only the NMF is repeated per seed.

Value

A list with

unit the comparison unit used.

seeds the seeds fitted.

n_topics the effective number of topics.

ari list of numeric vectors `within_joint`, `within_text`, and `between` (matched seeds).

summary data.frame of median/mean ARI and count per comparison.

test the `wilcox.test` object comparing the between distribution against the pooled within distributions (`alternative = "less"`).

consensus list of consensus (co-membership) matrices, `joint` and `text`, across seeds.

dispersion named numeric vector of consensus dispersion coefficients (1 = identical clustering across all seeds).

partitions list of the raw per-seed partitions, `joint` and `text`, for further inspection.

Note

Beta feature: This function is experimental and the API may change without notice in future versions.

See Also

[decomposeSubnetworkByTopic](#)

Examples

```
## Not run:
input <- data.table::fread(system.file(
  "extdata/groupComparisonModel.csv",
  package = "MSstatsBioNet"
))
subnetwork <- getSubnetworkFromIndra(input)
cmp <- compareTopicModels(subnetwork, seeds = 1:20, n_topics = 5)
cmp$summary
cmp$test          # p < 0.05 => PPI changes topics beyond chance
cmp$dispersion    # how stable each mode is across seeds

## End(Not run)
```

convert_ids	<i>Ground entities in a backend's namespaces</i>
-------------	--

Description

Fills in namespace, entity_id, and entity_name from each row's id (parent_id for ptm_site rows), read as the identifier system in id_type. Rows that don't ground are left NA. backend_capabilities(backend)\$id_c lists the (entity type, identifier system) pairs a backend can convert.

Usage

```
convert_ids(backend, entities, ...)

## S4 method for signature 'NetworkBackend'
convert_ids(backend, entities, ...)

## S4 method for signature 'IndraBackend'
convert_ids(backend, entities, ...)
```

Arguments

backend	a NetworkBackend, e.g. from indra_backend()
entities	entity table from prepare_entities()
...	passed to methods

Details

Ground the full results table, not only the significant rows, so that [get_network\(\)](#) can recognize every node that is in the input.

For INDRA, UniProt IDs and mnemonics are mapped through CoGEx, and gene symbols and chemical names are grounded with Gilda. When an identifier is a protein group ("P1;P2") or a name with several candidates, the groundings are ";"-joined and positionally aligned.

Value

entities with the grounding columns filled in

See Also

[get_entity_properties\(\)](#)

Examples

```
df <- data.frame(Protein = c("P04637", "Q00610"))
entities <- prepare_entities(df, entity_type = "protein",
                             id_type = "uniprot")
convert_ids(indra_backend(), entities)
```

cytoscapeNetwork

Render a Cytoscape network visualisation

Description

Creates an interactive network diagram powered by Cytoscape.js and the dagre layout algorithm. Nodes can carry log fold-change (logFC) values which are mapped to a blue-grey-red colour gradient. PTM (post-translational modification) site information is shown as small satellite nodes and edge overlaps are surfaced as hover tooltips.

Usage

```
cytoscapeNetwork(
  nodes,
  edges = data.frame(),
  displayLabelType = "id",
  nodeFontSize = 12,
  layoutOptions = NULL,
  width = NULL,
  height = NULL,
  elementId = NULL
)
```

Arguments

nodes	Data frame with at minimum an id column. Optional columns: logFC (numeric), entity_name (character; may be semicolon-joined for multi-grounded rows), entity_id (character), site (character, underscore-separated PTM site list), and the node status columns of validate_network() : entity_type sets the node shape (e.g. a hexagon for "metabolite"); measured = FALSE draws a node with no fill and a dashed grey border ("not in input data"); included_in_query = FALSE fades it. A node with an NA logFC has no fill. A protein with site rows takes its colour from its row with no site; without one, only its sites are coloured. Missing status columns count as TRUE.
-------	--

edges	Data frame with columns source, target, interaction. Optional: directed (an edge with FALSE has no arrow and is drawn once for both directions; without the column, Complex and Association edges are undirected), site, evidence_url, opened when an edge is clicked.
displayLabelType	"id" (default) or "entity_name": which column is used as the visible node label. "entityName" is deprecated and is treated as "entity_name" with a warning.
nodeFontSize	Font size (px) for node labels. Default 12.
layoutOptions	Named list of dagre layout options to override the defaults (e.g. list(rankDir = "LR")).
width, height	Widget dimensions passed to createWidget .
elementId	Optional explicit HTML element id.

Value

An htmlwidget object that renders in R Markdown, Shiny, or the RStudio Viewer pane.

Examples

```
## Not run:
nodes <- data.frame(
  id    = c("TP53", "MDM2", "CDKN1A"),
  logFC = c(1.5, -0.8, 2.1),
  stringsAsFactors = FALSE
)
edges <- data.frame(
  source    = c("TP53", "MDM2"),
  target    = c("MDM2", "TP53"),
  interaction = c("Activation", "Inhibition"),
  stringsAsFactors = FALSE
)
cytoscapeNetwork(nodes, edges)

## End(Not run)
```

cytoscapeNetworkOutput

Shiny output binding for cytoscapeNetwork

Description

Creates a Shiny output binding for a Cytoscape network visualization, allowing the network to be rendered within Shiny applications.

Usage

```
cytoscapeNetworkOutput(outputId, width = "100%", height = "500px")
```

Arguments

outputId	output variable to read from
width, height	Must be a valid CSS unit (like "100%", "400px", "auto") or a number, which will be coerced to a string and have "px" appended.

Value

A Shiny output binding for a Cytoscape network visualization.

Examples

```
## Not run:
library(shiny)

ui <- fluidPage(
  cytoscapeNetworkOutput("cytoNetwork")
)

server <- function(input, output, session) {
  output$cytoNetwork <- renderCytoscapeNetwork({
    nodes <- data.frame(
      id = c("TP53", "MDM2", "CDKN1A"),
      logFC = c(1.5, -0.8, 2.1),
      stringsAsFactors = FALSE
    )
    edges <- data.frame(
      source = c("TP53", "MDM2"),
      target = c("MDM2", "TP53"),
      interaction = c("Activation", "Inhibition"),
      stringsAsFactors = FALSE
    )
    cytoscapeNetwork(nodes, edges)
  })
}

shinyApp(ui, server)

## End(Not run)
```

decomposeSubnetworkByTopic

Decompose a subnetwork into topic-specific subnetworks via joint NMF

Description

Takes a subnetwork (the output of [getSubnetworkFromIndra](#)) and splits it into a list of smaller, topic-specific subnetworks discovered with unsupervised non-negative matrix factorization (NMF).

Usage

```
decomposeSubnetworkByTopic(
  subnetwork,
  n_topics = 5,
  edge_topic_cutoff = 0.2,
  n_top_terms = 10,
  min_term_count = 2,
  max_iter = 200,
  tol = 1e-04,
  seed = 1,
  include_ppi = TRUE,
  evidence = NULL,
  abstracts = NULL,
  backend = NULL
)
```

Arguments

subnetwork	list with nodes and edges data.frames, e.g. the output of getSubnetworkFromIndra .
n_topics	number of topics (rank of the factorization). Default 5.
edge_topic_cutoff	numeric in $[0, 1]$; an edge is added to a topic's subnetwork when the topic carries at least this share of the edge's total loading. Each edge is always included in at least its highest-loading topic. Default 0.2.
n_top_terms	number of top words to report per topic. Default 10.
min_term_count	minimum corpus frequency for a word to be kept when building X_{text} . Default 2.
max_iter	maximum number of NMF multiplicative-update iterations. Default 200.
tol	relative-change tolerance for NMF early stopping. Default $1e-4$.
seed	random seed for NMF initialization. Default 1.
include_ppi	logical; if TRUE (default) the PPI/edge matrix is factorized jointly with the text matrix via a shared basis. If FALSE, NMF is run on the paper-word matrix only and edge-topic loadings are derived afterwards by folding edge counts onto the text-learned topics, so the PPIs do not influence the topics themselves.
evidence	optional pre-fetched evidence data.frame, e.g. <code>attr(topics, "corpus")\$evidence</code> from a previous call. It is subset to the edges of subnetwork, so the evidence gathered for a parent network can be reused for any of its topic subnetworks. When NULL (default) the evidence is queried from the backend.
abstracts	optional named character vector (or list) mapping PMID to abstract text, e.g. <code>attr(topics, "corpus")\$abstracts</code> . Only PMIDs missing from it are fetched from PubMed. Default NULL fetches all.

backend the backend to get the evidence from, e.g. `indra_backend()`. NULL (default) uses the default backend named in each edge's `backend_database`. See `get_evidence()`.

Details

The procedure is:

1. For every edge, the supporting evidence is retrieved from the backend (see `get_evidence`) and the PubMed abstract of each referenced PMID is fetched. Papers (PMIDs) are the shared unit of analysis.
2. Two matrices are built that share the same rows (papers): X_{text} (papers x words, term counts from the abstracts) and X_{edges} (papers x unique source_target_interaction combinations, evidence-sentence counts).
3. NMF learns a basis matrix W (papers x topics). When `include_ppi = TRUE` (the default) a *joint* NMF learns a single shared W such that $X_{text} \approx WH_{text}$ and $X_{edges} \approx WH_{edges}$, tying each learned topic to both a set of words and a set of edges. When `include_ppi = FALSE` the factorization uses only X_{text} ($X_{text} \approx WH_{text}$); the PPI evidence is excluded from the modeling and edge-topic loadings are instead derived afterwards by folding the edge counts onto the text-learned topics ($H_{edges} = W^T X_{edges}$). This lets you compare topic structure with and without the PPI view.
4. Each topic becomes its own subnetwork: an edge is included in a topic when that topic carries at least `edge_topic_cutoff` of the edge's loading (soft, overlapping assignment), and nodes are restricted to those touched by the kept edges.

Value

A list of length `n_topics`, named `topic_1 ... topic_k`. Each element is a topic-specific subnetwork: a list with

nodes nodes data.frame restricted to the topic's edges.

edges edges data.frame for the topic, with an added `topicWeight` column (the edge's topic share).

topic the topic index.

topTerms character vector of the topic's top words.

pmids PMIDs whose strongest topic loading is this topic.

The full factorization (W , H_{text} , H_{edges} , etc.) is attached as the `"nmf"` attribute of the returned list. The evidence and abstracts used are attached as the `"corpus"` attribute (a list with evidence and abstracts) so they can be passed back in via the `evidence` and `abstracts` arguments.

Note

Beta feature: This function is experimental and the API may change without notice in future versions.

See Also

`getSubnetworkFromIndra`, `filterSubnetworkByContext`, `decomposeSubnetworkIntoHierarchicalTopics`

Examples

```
## Not run:
input <- data.table::fread(system.file(
  "extdata/groupComparisonModel.csv",
  package = "MSstatsBioNet"
))
subnetwork <- getSubnetworkFromIndra(input)
topics <- decomposeSubnetworkByTopic(subnetwork, n_topics = 5)
topics$topic_1$topTerms
exportNetworkToHTML(topics$topic_1$nodes, topics$topic_1$edges)

# Re-decompose a topic without re-querying the backend / PubMed.
corpus <- attr(topics, "corpus")
topics_deeper <- decomposeSubnetworkByTopic(
  topics$topic_1, n_topics = 5,
  evidence = corpus$evidence, abstracts = corpus$abstracts
)

## End(Not run)
```

decomposeSubnetworkIntoHierarchicalTopics

Recursively decompose a subnetwork into a hierarchy of topic subnetworks

Description

Repeatedly applies [decomposeSubnetworkByTopic](#) to its own topic subnetworks until every branch is small enough to inspect by hand (at most `max_edges` edges), producing a topic tree: broad themes near the root and increasingly specific sub-themes towards the leaves.

Usage

```
decomposeSubnetworkIntoHierarchicalTopics(
  subnetwork,
  max_edges = 10,
  n_topics = 5,
  edge_topic_cutoff = 0.9,
  max_depth = 5,
  evidence = NULL,
  abstracts = NULL,
  backend = NULL,
  ...
)
```

Arguments

<code>subnetwork</code>	list with nodes and edges data.frames, e.g. the output of getSubnetworkFromIndra .
<code>max_edges</code>	a branch with at most this many edges is not split further. Default 10.
<code>n_topics</code>	number of topics per split. Default 5.
<code>edge_topic_cutoff</code>	topic-share threshold for assigning an edge to a topic at each split; see decomposeSubnetworkByTopic . The default of 0.9 gives a near-partition, so sibling topics rarely share edges. Lower values allow an edge to appear in several sibling topics.
<code>max_depth</code>	maximum depth of the tree (the root is depth 0). Default 5.
<code>evidence</code>	optional pre-fetched evidence data.frame, e.g. <code>attr(topics, "corpus")\$evidence</code> from decomposeSubnetworkByTopic or <code>result\$corpus\$evidence</code> from a previous call of this function. Default NULL queries the backend once.
<code>abstracts</code>	optional named character vector mapping PMID to abstract text. Only missing PMIDs are fetched from PubMed. Default NULL.
<code>backend</code>	the backend to get the evidence from, e.g. indra_backend() . NULL (default) uses the default backend named in each edge's <code>backend_database</code> . See get_evidence() .
<code>...</code>	further arguments passed to decomposeSubnetworkByTopic , e.g. <code>n_top_terms</code> , <code>min_term_count</code> , <code>include_ppi</code> , <code>seed</code> .

Details

Edge evidence and PubMed abstracts are gathered once for the input subnetwork and reused for every sub-decomposition, so no further network requests are made during the recursion. Each sub-decomposition rebuilds its vocabulary and refits the NMF on only the papers supporting that branch's edges, which lets finer topics emerge.

A branch stops splitting (becomes a leaf) when any of the following holds, recorded in the `stop_reason` column of tree:

small_enough it has at most `max_edges` edges.

max_depth it sits at depth `max_depth`.

too_few_papers fewer than two papers support its edges.

no_split every child topic contained all of its edges (or none), so splitting would make no progress.

failed: <message> the decomposition raised an error, e.g. no usable words in the branch's abstracts.

Edges without any PMID-backed evidence cannot be assigned to a topic and only appear at the root.

Value

An object of class `topicHierarchy`: a list with

tree data.frame with one row per topic, in depth-first order. Columns: `id` ("root", "1", "1.2", ...), `parent_id`, `depth`, `topic` (index within the parent's split), `n_edges`, `n_nodes`, `n_papers` (papers supporting the topic's edges), `n_children`, `is_leaf`, `stop_reason`, `mean_topic_weight` (mean share of the topic's edges' loading, a cohesion score), `top_terms` (collapsed with ", ", label, and `pathString` ("root/1/1.2", for `data.tree`).

subnetworks named list keyed by id; each element is a subnetwork (nodes, edges, topTerms, pmids) that can be passed to [cytoscapeNetwork](#) or [exportNetworkToHTML](#).

edge_membership long data.frame with one row per (topic, edge): id, depth, is_leaf, source, target, interaction, topicWeight. Filter on is_leaf to see which fine-grained topic(s) each edge ends up in.

corpus list with the evidence and abstracts used, for reuse via the evidence and abstracts arguments.

params the settings used.

Note

Beta feature: This function is experimental and the API may change without notice in future versions.

See Also

[decomposeSubnetworkByTopic](#)

Examples

```
## Not run:
input <- data.table::fread(system.file(
  "extdata/groupComparisonModel.csv",
  package = "MSstatsBioNet"
))
subnetwork <- getSubnetworkFromIndra(input)
hierarchy <- decomposeSubnetworkIntoHierarchicalTopics(
  subnetwork, max_edges = 10, n_topics = 5, edge_topic_cutoff = 0.9
)
hierarchy                                     # indented topic tree
leaves <- hierarchy$tree[hierarchy$tree$is_leaf, ]
leaves[order(-leaves$mean_topic_weight), c("id", "n_edges", "top_terms")]

# Inspect one fine-grained topic as a network.
leaf <- hierarchy$subnetworks[["1.2"]]
exportNetworkToHTML(leaf$nodes, leaf$edges)

# Tree visualization with other packages, e.g.
# data.tree::as.Node(hierarchy$tree)
# igraph::graph_from_data_frame(
#   hierarchy$tree[-1, c("parent_id", "id")])

## End(Not run)
```

`deleteEdgeFromNetwork` *Delete an edge from a network edges data frame*

Description

Removes the row(s) from an edges data frame that match the given source, target, and interaction values. This is the programmatic counterpart of the interactive Ctrl+click / right-click edge deletion available in [cytoscapeNetwork](#).

Usage

```
deleteEdgeFromNetwork(edges, source, target, interaction)
```

Arguments

<code>edges</code>	Data frame with at minimum columns source, target, and interaction.
<code>source</code>	Character. The source node identifier of the edge to remove.
<code>target</code>	Character. The target node identifier of the edge to remove.
<code>interaction</code>	Character. The interaction type of the edge to remove.

Value

The edges data frame with the matching row(s) removed.

Examples

```
edges <- data.frame(  
  source = c("TP53", "MDM2", "CDKN1A"),  
  target = c("MDM2", "TP53", "TP53"),  
  interaction = c("Activation", "Inhibition", "Activation"),  
  stringsAsFactors = FALSE  
)  
deleteEdgeFromNetwork(edges, "MDM2", "TP53", "Inhibition")
```

`exportNetworkToHTML` *Export network data with Cytoscape visualization*

Description

Convenience function that takes nodes and edges data directly and creates both the configuration and HTML export in one step.

Usage

```
exportNetworkToHTML(
  nodes,
  edges,
  filename = "network_visualization.html",
  displayLabelType = "id",
  nodeFontSize = 12,
  ...
)
```

Arguments

nodes	Data frame with at minimum an id column. Optional columns: logFC (numeric), entity_name (character; may be semicolon-joined for multi-grounded rows), entity_id (character), site (character, underscore-separated PTM site list), and the node status columns of validate_network() : entity_type sets the node shape (e.g. a hexagon for "metabolite"); measured = FALSE draws a node with no fill and a dashed grey border ("not in input data"); included_in_query = FALSE fades it. A node with an NA logFC has no fill. A protein with site rows takes its colour from its row with no site; without one, only its sites are coloured. Missing status columns count as TRUE.
edges	Data frame with columns source, target, interaction. Optional: directed (an edge with FALSE has no arrow and is drawn once for both directions; without the column, Complex and Association edges are undirected), site, evidence_url, opened when an edge is clicked.
filename	Output HTML filename
displayLabelType	"id" (default) or "entity_name": which column is used as the visible node label. "entityName" is deprecated and is treated as "entity_name" with a warning.
nodeFontSize	Font size (px) for node labels. Default 12.
...	Additional arguments passed to exportCytoscapeToHTML()

Value

Invisibly returns the file path of the created HTML file

filter_by_curation	<i>Remove evidence curated as incorrect</i>
--------------------	---

Description

Subtracts the evidence curated as incorrect, from [get_curations\(\)](#), from each edge's evidence_count, then drops the edges left below min_evidence, and the nodes those edges leave without any edge. Edges left with no evidence are dropped whatever min_evidence is. A message says how many edges were dropped.

Usage

```
filter_by_curation(network, min_evidence = 1, backend = NULL)
```

Arguments

network	list of nodes and edges from get_network() . edges needs the columns source, target, statement_id, and evidence_count, and backend_database unless backend is given.
min_evidence	minimum evidence count per edge, after subtracting the incorrect evidence
backend	the backend to get the curations from, e.g. indra_backend() . NULL (default) uses the default backend named in each edge's backend_database. Pass a backend to use one built with non-default settings, such as indra_backend(curation_url =) . A backend without curations, such as one with no get_curations() method, gives an error naming it.

Details

Run it after any filter that pools edges, such as the PTM-site filter of [getSubnetworkFromIndra\(\)](#): dropping edges first can change which edges such a filter keeps.

Value

network, with edges\$evidence_count reduced and the dropped edges and nodes removed. Other elements of network are kept.

See Also

[get_curations\(\)](#)

Examples

```
input <- data.table::fread(system.file(
  "extdata/groupComparisonModel.csv",
  package = "MSstatsBioNet"
))
indra <- indra_backend()
entities <- prepare_entities(input, entity_type = "protein",
  id_type = "uniprot")
entities <- convert_ids(indra, entities)
entities <- select_entities(entities, pvalue_cutoff = 0.05)
network <- get_network(indra, entities, interaction_types = "Complex")
network <- filter_by_curation(network, min_evidence = 2)
head(network$edges)
```

filter_by_ptm_site	<i>Keep the edges that modify a PTM site in the network</i>
--------------------	---

Description

Keeps an edge when its site is one of the PTM sites of its target, that is, of a node row with the target's id. For a network built from MSstatsPTM results, these are the edges that explain a site in the input, e.g. a kinase phosphorylating a site whose phosphorylation changed. Each edge is checked on its own, so the order of this filter and [filter_by_curation\(\)](#) doesn't matter.

Usage

```
filter_by_ptm_site(network)
```

Arguments

network	list of nodes and edges from get_network() , with PTM site rows in nodes (entity tables with entity_type = "ptm_site")
---------	--

Details

Edges without a site are dropped. These include every edge that isn't a modification (Complex, Activation, ...), and every edge from a backend that reports no sites, such as STRING: filter an INDRA network before merging it with one from such a backend, if those edges should stay. Nodes are dropped when they had edges and lost them all. Nodes that had no edge, and other elements of network, such as provenance, are kept.

Node sites are split on "_" (several sites of one peptide, e.g. "S1039_S1042") and ";" (members of a protein group); edge sites on ";".

Value

network, with the edges whose site is not a site of their target removed, and the nodes they leave without edges. Stops when no edges remain.

See Also

[add_entity_values\(\)](#)

Examples

```
# A phosphorylation network of CPTAC CCRCC sites and ten kinases, saved
# by inst/script/cptac-ccrcc-network.R
saved <- readRDS(system.file("extdata/cptac-ccrcc-network.rds",
                             package = "MSstatsBioNet"))
network <- filter_by_ptm_site(saved$network)
head(network$edges[, c("source", "target", "site")])
```

filterSubnetworkByContext

Filter a subnetwork by contextual relevance

Description

Fetches PubMed abstracts for evidence PMIDs, scores each abstract against a user-supplied query, and returns only the nodes, edges, and evidence rows whose abstracts meet the scoring cutoff.

Usage

```
filterSubnetworkByContext(
  nodes,
  edges,
  query = NULL,
  cutoff = NULL,
  method = c("tag_count", "cosine"),
  exclude_keywords = NULL,
  backend = NULL
)
```

Arguments

nodes	A dataframe of network nodes.
edges	A dataframe of network edges with columns: source, target, interaction, site, evidence_url, statement_id.
query	For method = "tag_count": a character vector of tags, e.g. c("CHEK1", "DNA damage", "DNA damage repair"). For method = "cosine": a single character string. May be NULL (default) when exclude_keywords is supplied; abstracts are then not scored (score is NA) and only the keyword exclusion is applied.
cutoff	Numeric threshold applied to the chosen scoring method. "tag_count": integer ≥ 0; abstracts must contain at least this many tags. Max possible value is length(query). Default 1. "cosine": numeric in $[-1, 1]$; abstracts must score $\geq$ this value. Default 0.10.
method	One of "tag_count" (default) or "cosine".
exclude_keywords	Optional character vector of keywords. Abstracts containing any of them as a whole word or phrase (case-insensitive) are removed, regardless of their score. For example, "colon" matches "colon" and "colon-specific" but not "colonize" or "colons"; list variants such as plurals explicitly. To exclude by keyword only, omit query. Default NULL excludes nothing.
backend	the backend to get the evidence from, e.g. indra_backend() . NULL (default) uses the default backend named in each edge's backend_database. Pass a backend built with non-default settings, such as a different URL, here. See get_evidence() .

Details

Two scoring methods are available, controlled by the method argument:

"tag_count" (**default**) Counts how many tags from query appear as whole words or phrases in the abstract (case-insensitive), so "colon" does not match "colony" or "colonize". The score for each abstract is an integer in $[0, \text{length}(\text{query})]$. Set cutoff to the minimum number of tags that must appear - e.g. cutoff = 2 keeps abstracts that mention at least 2 of your tags. query must be a character *vector* of tags when using this method.

"cosine" Scores abstracts using TF-IDF cosine similarity against query. Scores are in $[-1, 1]$ (in practice $[0, 1]$ for text). Set cutoff to a decimal threshold - e.g. cutoff = 0.10. query should be a single character string; expand it with synonyms and related terms for better recall under exact token matching.

Value

A named list with four elements:

nodes	Filtered nodes dataframe (only nodes present in kept edges)
edges	Filtered edges dataframe
evidence	Dataframe with columns: source, target, interaction, site, evidence_url, statement_id, text, pmid, score. The score column contains tag counts (integer) or cosine similarities (numeric) depending on the method used.
abstracts	Named character vector mapping each PMID in evidence to its abstract text.

The evidence and abstracts elements can be passed to the same-named arguments of [decomposeSubnetworkByTopic](#) or [decomposeSubnetworkIntoHierarchicalTopics](#), together with the returned list as subnetwork, so the backend and PubMed are not queried again.

Note

Beta feature: This function is experimental and the API may change without notice in future versions.

Examples

```
## Not run:
filtered <- filterSubnetworkByContext(
  subnetwork$nodes, subnetwork$edges,
  query = c("DNA damage", "DNA repair"),
  exclude_keywords = c("review")
)
hierarchy <- decomposeSubnetworkIntoHierarchicalTopics(
  filtered,
  evidence = filtered$evidence,
  abstracts = filtered$abstracts
)

## End(Not run)
```

get_curations	<i>Get the curations of edges from a backend</i>
---------------	--

Description

Curators mark single pieces of evidence for a statement as correct or incorrect. `get_curations()` counts, for each statement, the evidence curated as incorrect. INDRA curations come from the INDRA database, with one request per statement. Most users call `filter_by_curation()`, which subtracts these counts from `evidence_count`.

Usage

```
get_curations(backend, edges, ...)

## S4 method for signature 'NetworkBackend'
get_curations(backend, edges, ...)

## S4 method for signature 'IndraBackend'
get_curations(backend, edges, ...)
```

Arguments

<code>backend</code>	a <code>NetworkBackend</code> , e.g. from <code>indra_backend()</code>
<code>edges</code>	the edges of a network from <code>get_network()</code> . Needs the column <code>statement_id</code> .
<code>...</code>	passed to methods

Value

`data.frame` with one row per unique `statement_id` and the columns `statement_id` (character) and `incorrect_count` (integer). A failed request warns and counts as 0. A method for another backend may leave out statements with no curations, which `filter_by_curation()` counts as 0, but must return `statement_id` as character: `filter_by_curation()` errors otherwise, since a numeric hash can lose precision and match no edge.

See Also

[filter_by_curation\(\)](#)

Examples

```
input <- data.table::fread(system.file(
  "extdata/groupComparisonModel.csv",
  package = "MSstatsBioNet"
))
indra <- indra_backend()
entities <- prepare_entities(input, entity_type = "protein",
  id_type = "uniprot")
entities <- convert_ids(indra, entities)
```

```

entities <- select_entities(entities, pvalue_cutoff = 0.05)
network <- get_network(indra, entities, interaction_types = "Complex")
get_curations(indra, head(network$edges, 2))

```

get_entity_properties *Add a backend's properties of each entity*

Description

Adds one column per property, e.g. `is_kinase`. A property is NA for rows whose `entity_type` it doesn't apply to, and for rows the backend has no answer for. `backend_capabilities(backend)$entity_properties` lists the properties a backend supports.

Usage

```

get_entity_properties(backend, entities, properties = NULL, ...)

## S4 method for signature 'NetworkBackend'
get_entity_properties(backend, entities, properties = NULL, ...)

## S4 method for signature 'IndraBackend'
get_entity_properties(backend, entities, properties = NULL, ...)

```

Arguments

<code>backend</code>	a <code>NetworkBackend</code> , e.g. from <code>indra_backend()</code>
<code>entities</code>	entity table, grounded by <code>convert_ids()</code>
<code>properties</code>	the properties to add. <code>NULL</code> adds every property the backend supports.
<code>...</code>	passed to methods

Details

For INDRA, the properties are `is_transcription_factor`, `is_kinase`, and `is_phosphatase`, looked up by gene symbol. Only rows with a single HGNC grounding get values. A PTM site gets the properties of its parent protein.

Value

entities with one column per property

Examples

```

df <- data.frame(Protein = c("P04637", "Q00610"))
indra <- indra_backend()
entities <- prepare_entities(df, entity_type = "protein",
                             id_type = "uniprot")
entities <- convert_ids(indra, entities)
get_entity_properties(indra, entities, properties = "is_kinase")

```

get_evidence	<i>Get the evidence behind network edges</i>
--------------	--

Description

Looks up the evidence sentences and their PubMed IDs for each edge, using the edge's `statement_id`. Edges that share a `statement_id` get the same evidence.

Usage

```
get_evidence(backend, edges, ...)

## S4 method for signature 'NetworkBackend'
get_evidence(backend, edges, ...)

## S4 method for signature 'IndraBackend'
get_evidence(backend, edges, ...)
```

Arguments

<code>backend</code>	a <code>NetworkBackend</code> , e.g. from <code>indra_backend()</code>
<code>edges</code>	the edges of a network from <code>get_network()</code> . Needs the columns <code>source</code> , <code>target</code> , <code>interaction</code> , <code>site</code> , <code>evidence_url</code> , and <code>statement_id</code> .
<code>...</code>	passed to methods

Details

For INDRA, the evidence comes from CoGEx. Evidence without text is left out, and `pmid` is "" for evidence from a source with no PubMed ID.

`filterSubnetworkByContext()` and the topic functions call `get_evidence()` with the backend named in each edge's `backend_database`, unless their `backend` argument is given.

Value

`data.frame` with one row per (edge, evidence sentence) pair and the columns `source`, `target`, `interaction`, `site`, `evidence_url`, `statement_id`, `text`, and `pmid`. It has no rows, with a warning, when no edge has evidence text.

See Also

`filterSubnetworkByContext()`

Examples

```
input <- data.table::fread(system.file(
  "extdata/groupComparisonModel.csv",
  package = "MSstatsBioNet"
))
indra <- indra_backend()
entities <- prepare_entities(input, entity_type = "protein",
  id_type = "uniprot")
entities <- convert_ids(indra, entities)
entities <- select_entities(entities, pvalue_cutoff = 0.05)
network <- get_network(indra, entities, interaction_types = "Complex")
evidence <- get_evidence(indra, head(network$edges, 2))
head(evidence[, c("source", "target", "pmid", "text")])
```

get_network

Get a network from a backend

Description

Sends the selected entities to a backend and returns the network that answers the query, as nodes and edges tables.

Usage

```
get_network(
  backend,
  entities,
  query = subnetwork_query(),
  interaction_types = NULL,
  min_evidence = 1,
  min_confidence = NULL,
  evidence_sources = NULL,
  include_entities = NULL,
  ...
)

## S4 method for signature 'NetworkBackend,missing'
get_network(
  backend,
  entities,
  query = subnetwork_query(),
  interaction_types = NULL,
  min_evidence = 1,
  min_confidence = NULL,
  evidence_sources = NULL,
  include_entities = NULL,
```

```

    ...
)

## S4 method for signature 'NetworkBackend,NetworkQuery'
get_network(
  backend,
  entities,
  query = subnetwork_query(),
  interaction_types = NULL,
  min_evidence = 1,
  min_confidence = NULL,
  evidence_sources = NULL,
  include_entities = NULL,
  ...
)

## S4 method for signature 'IndraBackend,SubnetworkQuery'
get_network(
  backend,
  entities,
  query = subnetwork_query(),
  interaction_types = NULL,
  min_evidence = 1,
  min_confidence = NULL,
  evidence_sources = NULL,
  include_entities = NULL,
  ...
)

```

Arguments

backend	a NetworkBackend, e.g. from indra_backend()
entities	entity table from prepare_entities() , grounded by convert_ids() and flagged by select_entities()
query	a NetworkQuery saying which question to ask, e.g. subnetwork_query() (the default). See network_queries .
interaction_types	values of edges\$interaction to keep (INDRA statement types, e.g. "Activation"). NULL keeps all.
min_evidence	minimum evidence count per edge
min_confidence	minimum confidence per edge, in [0, 1]. Edges with no confidence score (NA) are dropped too, and a message says how many. NULL applies no cutoff.
evidence_sources	keeps edges with evidence from at least one of these sources, e.g. c("reach"). NULL keeps all. <code>backend_capabilities(backend)\$evidence_sources</code> lists a backend's sources; for INDRA, INDRA_DATABASE_SOURCES keeps edges with curated-database evidence and INDRA_TEXT_MINED_SOURCES edges with text-mined evidence.

```

include_entities
    "namespace:identifier" groundings to add to the query, e.g. "HGNC:1234".
    Use this for entities outside the input; to keep entities of the input that fail the
    cutoffs, use select_entities(force_include = ).
...
    passed to methods

```

Details

Only the rows of entities with `included_in_query = TRUE` are sent to the backend. Every node the backend returns is matched against all rows, so a node in the input gets `measured = TRUE` and its statistics, and a node not in the input gets `measured = FALSE` and NA statistics.

Each node is one row of entities (a PTM site row is drawn on its parent protein's node). When several rows share a grounding, e.g. two isoforms that both ground to the same gene, or a protein and a protein group that contains it, each of their nodes gets the backend's edges, with its own statistics. So one backend statement can give several edges, and edges can share a `statement_id`. A statement from a grounding to itself, such as a homodimer, gives each matching node a self-loop, and no edges between those nodes. To count statements rather than edges, count unique `backend_database` and `statement_id` pairs.

`get_network()` prints the question it asks as a message, with the number of entities, so the query in a saved script or log is readable without the documentation.

Confidence values are comparable within one backend, not across backends. For INDRA, confidence is the INDRA belief score.

Value

list of nodes and edges data.frames that meets the contract checked by `validate_network()`, and provenance: a data.frame with one row recording the query, with the columns `backend_database`, `query_type`, `retrieved_at` (when the response arrived, in UTC), `backend_version` (NA for INDRA, which has no data versions), `backend_url`, `organism` (NCBI taxon IDs of the queried rows), `parameters` (the query arguments, as JSON), and `package_version` (of MSstatsBioNet). It joins to the edges on `backend_database` and `query_type`. `merge_networks()` combines the provenance of its networks, and `save_network()` saves it with the network. It is lost when the network is rebuilt with `list(nodes = , edges = )`.

See Also

[network_queries](#), [backend_capabilities\(\)](#)

Examples

```

input <- data.table::fread(system.file(
  "extdata/groupComparisonModel.csv",
  package = "MSstatsBioNet"
))
entities <- prepare_entities(input, entity_type = "protein",
                             id_type = "uniprot")

indra <- indra_backend()
entities <- convert_ids(indra, entities)

```

```

entities <- select_entities(entities, pvalue_cutoff = 0.05)
network <- get_network(indra, entities, subnetwork_query(),
                      interaction_types = "Complex")
head(network$nodes)
head(network$edges)

```

getSubnetworkFromIndra

Get subnetwork from INDRA database

Description

Using differential abundance results from MSstats, this function retrieves a subnetwork of protein interactions from INDRA database.

Usage

```

getSubnetworkFromIndra(
  input,
  protein_level_data = NULL,
  pvalueCutoff = NULL,
  statement_types = NULL,
  paper_count_cutoff = 1,
  evidence_count_cutoff = 1,
  correlation_cutoff = 0.3,
  sources_filter = NULL,
  logfc_cutoff = NULL,
  force_include_other = NULL,
  filter_by_curation = FALSE,
  filter_by_ptm_site = FALSE,
  include_infinite_fc = FALSE,
  direction = c("both", "up", "down")
)

```

Arguments

input	output of groupComparison function's comparisonResult table, annotated by annotateProteinInfoFromIndra . Must contain Protein, EntityNamespace, and EntityId columns (and typically also EntityName, log2FC, adj.pvalue). When an analyte grounds to multiple candidates the three Entity* columns are semicolon-joined and positionally aligned.
protein_level_data	Deprecated, and will be removed in a future release. Output of the dataProcess function's ProteinLevelData table, used to annotate edges with correlations and apply correlation_cutoff. Supplying it gives a deprecation warning.
pvalueCutoff	p-value cutoff for filtering. Default is NULL, i.e. no filtering

statement_types	list of interaction types to filter on. Equivalent to statement type in INDRA. Default is NULL.
paper_count_cutoff	Deprecated, and will be removed in a future release. It is ignored: paper counts are not available from INDRA, so this filter never had an effect for 1 and removed every edge for larger values. Supplying it gives a deprecation warning.
evidence_count_cutoff	number of evidence to filter on for each paper. E.g. A paper may have 5 sentences describing the same interaction vs 1 sentence. Default is 1.
correlation_cutoff	Deprecated, and will be removed in a future release. If protein_level_data is not NULL, remove edges whose absolute correlation is below this cutoff. Default is 0.3. Supplying it gives a deprecation warning.
sources_filter	filtering only on specific sources. Default is no filter, i.e. NULL. See INDRA_DATABASE_SOURCES for INDRA's source names. Otherwise, should be a list, e.g. c('reach', 'med-scan').
logfc_cutoff	absolute log fold change cutoff for filtering proteins. Only proteins with llogFCI greater than this value will be retained. Default is NULL, i.e. no logFC filtering.
force_include_other	character vector of identifiers to include in the network, regardless if those ids are in the input data. Should be formatted as "namespace:identifier", e.g. "HGNC:1234" or "CHEBI:4911".
filter_by_curation	logical, whether to filter out statements that have been curated as incorrect in INDRA. Default is FALSE. Runs filter_by_curation() with min_evidence = evidence_count_cutoff.
filter_by_ptm_site	logical, whether to filter edges based on whether the site information from INDRA matches with the PTM site in the input. Default is FALSE. Only applicable for differential PTM abundance results.
include_infinite_fc	logical, whether to include proteins with infinite log fold change (i.e. proteins that are only detected in one condition). Default is FALSE.
direction	Character string specifying the direction of regulation to include. One of "both" (default), "up" (upregulated only), or "down" (downregulated only).

Value

list of 2 data.frames, nodes and edges, that meets the contract checked by [validate_network](#).

edges has one row per INDRA statement and pair of nodes it connects (a statement whose grounding matches several Protein rows, such as two isoforms of one gene, gives each of them an edge): source, target, interaction (INDRA statement type), directed (FALSE for symmetric types such as Complex), site (PTM site on the target, or NA), confidence (INDRA belief score), evidence_count, evidence_url (INDRA page for this statement), statement_id (INDRA statement hash, as character), backend_database ("INDRA"), query_type ("subnetwork"),

evidence_sources (evidence count per source, as JSON), and the deprecated paperCount (and correlation when protein_level_data is given).

nodes has one row per analyte (one per site for PTM input, all with the protein's id): id, entity_type ("protein", "ptm_site", "metabolite", ...), entity_name, namespace, entity_id, measured (TRUE for analytes in input), included_in_query, node_role ("passed_cutoffs", or "user_added" for nodes that are there only through force_include_other), site, has_measured_sites (TRUE for proteins with PTM site rows), logFC, and adj.pvalue. Nodes from force_include_other that are not in input have measured = FALSE and NA statistics.

Examples

```
input <- data.table::fread(system.file(
  "extdata/groupComparisonModel.csv",
  package = "MSstatsBioNet"
))
subnetwork <- getSubnetworkFromIndra(input)
head(subnetwork$nodes)
head(subnetwork$edges)
```

indra_backend

Create an INDRA backend

Description

INDRA is a knowledge graph of mechanisms (activations, phosphorylations, complexes, ...) assembled from the literature and curated databases. The backend queries INDRA CoGEx for networks and grounds gene symbols and chemical names with Gilda, INDRA's grounding service. Curations, which mark evidence as correct or incorrect, come from the INDRA database. `backend_capabilities(indra_backend())` lists what it supports.

Usage

```
indra_backend(
  cogex_url = INDRA_API_URL,
  grounding_url = GILDA_API_URL,
  curation_url = INDRA_DB_URL
)
```

Arguments

cogex_url	base URL of INDRA CoGEx
grounding_url	base URL of Gilda
curation_url	base URL of the INDRA database, which holds the curations

Details

This function includes third-party software components that are licensed under the BSD 2-Clause License. Include the third-party licensing agreements if redistributing this package or results based on it. See the LICENSE file for details.

Value

an IndraBackend object, to pass to [convert_ids\(\)](#), [get_entity_properties\(\)](#), [get_network\(\)](#), [get_evidence\(\)](#), and [get_curations\(\)](#)

See Also

[NetworkBackend-class](#)

Examples

```
indra <- indra_backend()
backend_capabilities(indra)$query_types
```

```
indra_evidence_sources
```

Evidence sources of INDRA

Description

The names INDRA gives its evidence sources, in two groups: INDRA_DATABASE_SOURCES for curated databases (e.g. SIGNOR, BioGRID, PhosphoSitePlus) and INDRA_TEXT_MINED_SOURCES for the text-mining systems that read the literature (e.g. REACH, Sparser). Pass one to [get_network\(evidence_sources = \)](#) to keep only the edges with evidence from a curated database, or only those with text-mined evidence.

Usage

```
INDRA_DATABASE_SOURCES
```

```
INDRA_TEXT_MINED_SOURCES
```

Format

```
character vectors
```

Details

These are the names INDRA uses internally, which are the ones in `edges$evidence_sources`, and some differ from the full source names: "psp" (PhosphoSitePlus), "pc" (Pathway Commons), "pe" (Phospho.ELM), "vhn" (VirHostNet), and "bel_1c" (BEL large corpus).

Source

db_sources and reader_sources in INDRA's `indra.util.statement_presentation` module (INDRA 1.23.0), which INDRA CoGEx uses for its own source lists. "bel" is added to the databases, as CoGEx does.

See Also

[get_network\(\)](#), [backend_capabilities\(\)](#)

Examples

```
INDRA_DATABASE_SOURCES
INDRA_TEXT_MINED_SOURCES

input <- data.table::fread(system.file(
  "extdata/groupComparisonModel.csv",
  package = "MSstatsBioNet"
))
indra <- indra_backend()
entities <- prepare_entities(input, entity_type = "protein",
                             id_type = "uniprot")
entities <- convert_ids(indra, entities)
entities <- select_entities(entities, pvalue_cutoff = 0.05)
curated <- get_network(indra, entities,
                       evidence_sources = INDRA_DATABASE_SOURCES)
head(curated$edges$evidence_sources)
```

load_network	<i>Load a network saved with <code>save_network()</code></i>
--------------	--

Description

Reads the network and checks it with [validate_network\(\)](#), so a file from an older version of the contract gives a clear error here instead of a failure in a later function.

Usage

```
load_network(file)
```

Arguments

file path of an .rds file written by [save_network\(\)](#)

Value

the network: list of nodes and edges, plus provenance and regulators when it had them

See Also[save_network\(\)](#)**Examples**

```

network <- list(
  nodes = data.frame(
    id = c("A", "B"), entity_type = "protein",
    entity_name = c("A", "B"), namespace = "HGNC",
    entity_id = c("1", "2"), measured = TRUE,
    included_in_query = TRUE, node_role = "passed_cutoffs"
  ),
  edges = data.frame(
    source = "A", target = "B", interaction = "Activation",
    directed = TRUE, site = NA_character_, confidence = 0.9,
    evidence_count = 3L, evidence_url = "https://example.org/1",
    statement_id = "1", backend_database = "INDRA",
    query_type = "subnetwork"
  )
)
file <- tempfile(fileext = ".rds")
suppressWarnings(save_network(network, file))
load_network(file)$edges

```

merge_networks

*Combine networks from several queries or backends***Description**

Combines the networks from several [get_network\(\)](#) calls, for example two queries against one backend, or one query against two backends, into one network that meets the same contract ([validate_network\(\)](#)).

Usage

```
merge_networks(..., entities = NULL)
```

Arguments

<code>...</code>	networks: lists of nodes and edges from get_network()
<code>entities</code>	NULL (default), or the entity table from prepare_entities() to recompute node status from

Details

Edges are the same edge when they have the same source, target, backend_database, and statement_id. One row is kept per edge: the copy with the largest evidence_count (the first one on ties), with the query_type values of all copies joined by ";", e.g. "subnetwork;mediated". A message names the statements whose copies differ in evidence_count or confidence. The same

relation from two backends stays two rows, one per backend_database, since each has its own confidence and evidence_url; confidences are never combined across backends. Columns that only some networks have are filled with NA.

Nodes are the same node when they have the same id and site (a protein has one row per PTM site). Their node_role values are joined by ";", e.g. "passed_cutoffs;mediator", included_in_query is TRUE if it is in any network, and other columns take the first non-NA value. has_measured_sites describes the protein, so it is TRUE on every row of a protein that has it in any network: merging a protein network with a PTM network marks the protein's protein-level row too. included_in_query describes the query, so it can differ between networks built from one entity table: a node can be a latent regulator in one query and part of another.

Without entities, measured, logFC, and adj.pvalue must agree for each node, because networks built from one entity table always agree; otherwise merge_networks() stops and names the nodes. To merge networks built from different entity tables, pass the entity table as entities: measured, logFC, adj.pvalue, and has_measured_sites are then recomputed from it (nodes not in it are measured = FALSE with NA statistics).

The regulators and provenance tables of the networks, when present, are combined by row, with columns filled with NA. Other list elements are dropped, with a message.

Value

list of nodes and edges, plus regulators and provenance when any network has them

See Also

[get_network\(\)](#), [validate_network\(\)](#)

Examples

```
input <- data.table::fread(system.file(
  "extdata/groupComparisonModel.csv",
  package = "MSstatsBioNet"
))
indra <- indra_backend()
entities <- prepare_entities(input, entity_type = "protein",
  id_type = "uniprot")
entities <- convert_ids(indra, entities)
# Two selections from one entity table: the edges among the proteins
# that pass both cutoffs are in both networks
loose <- get_network(indra, select_entities(entities, pvalue_cutoff = 0.05))
strict <- get_network(indra, select_entities(entities, pvalue_cutoff = 0.01))
network <- merge_networks(loose, strict)
# The shared edges come back once
c(loose = nrow(loose$edges), strict = nrow(strict$edges),
  merged = nrow(network$edges))
```

network_queries	<i>Questions you can ask of a network backend</i>
-----------------	---

Description

Each query constructor asks one question of a backend. Pass the query to `get_network()`. Every query returns the same nodes and edges tables (see `validate_network()`), so visualization and filtering work the same whichever question produced the network. The `query_type` column of edges names the query, without `_query`.

Details

Constructor	Question	Nodes added
<code>subnetwork_query()</code>	How are my selected entities connected to each other, with no other nodes added?	none, other than

More queries (shared regulators, regulator enrichment, paths, and others) are planned. `backend_capabilities(backend)$q` lists the ones a backend supports.

Glossary

entity One row of the entity table from `prepare_entities()`: one analyte of the input, such as a protein, a PTM site, or a metabolite.

selected An entity with `included_in_query = TRUE`, set by `select_entities()`: it passed the cutoffs, or was forced in. Only selected entities are sent to the backend.

in the input (measured) A node that matches a row of the entity table, whether or not it was selected. It carries that row's statistics.

latent A node that matches no row of the entity table, such as an entity added through `include_entities`. It has `measured = FALSE` and NA statistics. It may still have been measured in another experiment, or filtered out before the entity table was built.

grounding A "namespace:identifier" pair that names an entity in a backend, such as "HGNC:11998" (TP53).

See Also

`get_network()`, `backend_capabilities()`

NetworkBackend-class *Network backend classes*

Description

A backend is a source of prior-knowledge networks, such as INDRA. [get_network\(\)](#) dispatches on the backend and the query, so each (backend, query) pair has its own method. NetworkBackend is virtual: create a backend with a constructor such as [indra_backend\(\)](#). Other packages can add a backend by extending NetworkBackend and writing methods for the generics.

Details

IndraBackend queries INDRA CoGEx for networks and grounds names with Gilda, INDRA's grounding service. Curations come from the INDRA database.

Slots

`cogex_url` base URL of INDRA CoGEx

`grounding_url` base URL of Gilda

`curation_url` base URL of the INDRA database, which holds the curations

See Also

[indra_backend\(\)](#), [backend_capabilities\(\)](#)

NetworkQuery-class *Network query classes*

Description

A query says which question [get_network\(\)](#) asks of the backend. NetworkQuery is virtual: create a query with a constructor such as [subnetwork_query\(\)](#). See [network_queries](#) for the questions each query answers.

See Also

[network_queries](#)

prepare_entities	<i>Prepare an entity table from MSstats results</i>
------------------	---

Description

Builds the table that `convert_ids()` grounds, `select_entities()` flags, and `get_network()` queries. It has one row per analyte, and keeps every analyte, not only the significant ones, so that nodes returned by a backend can be recognized as being in the input.

Usage

```
prepare_entities(
  df,
  id_column = "Protein",
  entity_type,
  id_type,
  organism = "9606",
  label = NULL,
  logfc_column = NULL
)
```

Arguments

<code>df</code>	output of <code>groupComparison()</code> 's <code>ComparisonResult</code> table, or any table with one row per analyte.
<code>id_column</code>	name of the column holding the analyte identifiers.
<code>entity_type</code>	one of the entity types ("protein", "gene", "transcript", "ptm_site", "metabolite", "lipid", "drug", "complex", "family", "other"), or the name of a column of <code>df</code> holding one per row.
<code>id_type</code>	the identifier system of <code>id_column</code> ("uniprot", "uniprot_mnemonic", "hgnc_symbol", "ensembl_protein", "ensembl_gene", "entrez", "chemical_name", "inchikey", "hmdb", "chebi", "chembl"), or the name of a column of <code>df</code> holding one per row. Which of them a backend can convert is listed by <code>backend_capabilities(backend)\$id_conversion</code> . For PTM sites, the identifier system of the parent protein. "uniprot" and "uniprot_mnemonic" identifiers may also be UniProt FASTA headers, as FragPipe writes them ("sp P32004 L1CAM_HUMAN", or "sp P32004 L1CAM_HUMAN_S1194" for a site): <code>id</code> keeps the header, and <code>convert_ids()</code> uses its accession or mnemonic. "chemical_name" is a metabolite, lipid, or drug name, common or IUPAC (e.g. "glucose"), grounded by text matching.
<code>organism</code>	NCBI taxon ID, as a string.
<code>label</code>	the comparison to keep, when <code>df</code> has a <code>Label</code> column with more than one value.
<code>logfc_column</code>	the fold-change column to copy to <code>logFC</code> . <code>NULL</code> uses whichever of <code>log2FC</code> , <code>log10FC</code> , or <code>logFC</code> <code>df</code> has. Values are copied unchanged, in the log base of the input.

Details

For PTM sites (`entity_type = "ptm_site"`), the site is parsed from the end of each identifier, e.g. "P00533_S1039_S1042" gives parent "P00533" and site "S1039_S1042". A `GlobalProtein` column, as `MSstatsPTM` writes, overrides the parsed parent, except where it is "missing" (`MSstatsPTM`'s value for sites whose protein has no global profiling result).

Value

data.frame with columns `id`, `entity_type`, `id_type`, `namespace`, `entity_id`, `entity_name` (all NA until `convert_ids()`), `included_in_query` (TRUE until `select_entities()`), `site` and `parent_id` (for `ptm_site` rows), `organism`, and `logFC` and `adj.pvalue` when `df` has them. Other columns of `df` are not copied; join them back on `id`.

Examples

```
input <- data.table::fread(system.file(
  "extdata/groupComparisonModel.csv",
  package = "MSstatsBioNet"
))
entities <- prepare_entities(input, entity_type = "protein",
                             id_type = "uniprot")

head(entities)

# MSstatsPTM results: the parent protein and site are parsed from the id
ptm <- data.frame(Protein = c("P00533_S1039_S1042", "P04637_S15"),
                  log2FC = c(1.2, -0.8), adj.pvalue = c(0.01, 0.2))
prepare_entities(ptm, entity_type = "ptm_site", id_type = "uniprot")
```

```
previewNetworkInBrowser
```

Preview network in browser

Description

Generates a temporary HTML file for the network visualization and opens it in the default web browser for quick preview.

Usage

```
previewNetworkInBrowser(
  nodes,
  edges,
  displayLabelType = "id",
  nodeFontSize = 12
)
```

Arguments

nodes	Data frame with at minimum an id column. Optional columns: logFC (numeric), entity_name (character; may be semicolon-joined for multi-grounded rows), entity_id (character), site (character, underscore-separated PTM site list), and the node status columns of <code>validate_network()</code> : entity_type sets the node shape (e.g. a hexagon for "metabolite"); measured = FALSE draws a node with no fill and a dashed grey border ("not in input data"); included_in_query = FALSE fades it. A node with an NA logFC has no fill. A protein with site rows takes its colour from its row with no site; without one, only its sites are coloured. Missing status columns count as TRUE.
edges	Data frame with columns source, target, interaction. Optional: directed (an edge with FALSE has no arrow and is drawn once for both directions; without the column, Complex and Association edges are undirected), site, evidence_url, opened when an edge is clicked.
displayLabelType	"id" (default) or "entity_name": which column is used as the visible node label. "entityName" is deprecated and is treated as "entity_name" with a warning.
nodeFontSize	Font size (px) for node labels. Default 12.

Value

Invisibly returns the file path of the temporary HTML file.

Examples

```
## Not run:
nodes <- data.frame(id = c("A", "B", "C"))
edges <- data.frame(source = c("A", "B"), target = c("B", "C"))
previewNetworkInBrowser(nodes, edges)

## End(Not run)
```

`print.topicHierarchy` *Print a topic hierarchy as an indented tree*

Description

Print a topic hierarchy as an indented tree

Usage

```
## S3 method for class 'topicHierarchy'
print(x, n_terms = 5, ...)
```

Arguments

`x` a topicHierarchy object from [decomposeSubnetworkIntoHierarchicalTopics](#).
`n_terms` number of top terms to show per topic. Default 5.
`...` ignored.

Value

`x`, invisibly.

renderCytoscapeNetwork

Render a Cytoscape network in a Shiny application. This function is used to render a Cytoscape network visualization within a Shiny application.

Description

Render a Cytoscape network in a Shiny application. This function is used to render a Cytoscape network visualization within a Shiny application.

Usage

```
renderCytoscapeNetwork(expr, env = parent.frame())
```

Arguments

`expr` An expression that generates an HTML widget (or a **promise** of an HTML widget).
`env` The environment in which to evaluate `expr`.

Value

A rendered Cytoscape network widget for use in Shiny applications.

Examples

```
## Not run:
library(shiny)
library(MSstatsBioNet)

ui <- fluidPage(
  cytoscapeNetworkOutput("cytoNetwork")
)

server <- function(input, output, session) {
  output$cytoNetwork <- renderCytoscapeNetwork({
    nodes <- data.frame(
```

```

      id      = c("TP53", "MDM2", "CDKN1A"),
      logFC   = c(1.5, -0.8, 2.1),
      stringsAsFactors = FALSE
    )
    edges <- data.frame(
      source    = c("TP53", "MDM2"),
      target    = c("MDM2", "TP53"),
      interaction = c("Activation", "Inhibition"),
      stringsAsFactors = FALSE
    )
    cytoscapeNetwork(nodes, edges)
  })
}

shinyApp(ui, server)

## End(Not run)

```

save_network

*Save a network to a file***Description**

Saves the whole network, including provenance (when and from which backend it was retrieved) and regulators, to an .rds file that [load_network\(\)](#) reads back. Backends such as INDRA have no data versions, so the same query can give a different network later: save the network to keep the one an analysis used.

Usage

```
save_network(network, file)
```

Arguments

network	list of nodes and edges, e.g. from get_network() or merge_networks()
file	path of the file to write, ending in .rds

Details

The network is checked with [validate_network\(\)](#) first. A network without provenance is saved with a warning: it is lost when a network is rebuilt with `list(nodes = , edges = )`.

Value

file, invisibly

See Also

[load_network\(\)](#)

Examples

```
network <- list(
  nodes = data.frame(
    id = c("CHK1_HUMAN", "CDC25A_HUMAN"),
    entity_type = "protein",
    entity_name = c("CHEK1", "CDC25A"),
    namespace = "HGNC",
    entity_id = c("1925", "1725"),
    measured = TRUE,
    included_in_query = TRUE,
    node_role = "passed_cutoffs"
  ),
  edges = data.frame(
    source = "CHK1_HUMAN",
    target = "CDC25A_HUMAN",
    interaction = "Phosphorylation",
    directed = TRUE,
    site = "S76",
    confidence = 0.99,
    evidence_count = 12L,
    evidence_url = paste0("https://db.indra.bio/statements/",
                          "from_hash/-1234?format=html"),
    statement_id = "-1234",
    backend_database = "INDRA",
    query_type = "subnetwork"
  ),
  provenance = data.frame(
    backend_database = "INDRA",
    query_type = "subnetwork",
    retrieved_at = as.POSIXct("2026-10-09 12:00:00", tz = "UTC")
  )
)
file <- tempfile(fileext = ".rds")
save_network(network, file)
identical(load_network(file), network)
```

select_entities	<i>Flag the entities to query</i>
-----------------	-----------------------------------

Description

Sets `included_in_query` from the statistical cutoffs. Only these rows are sent to the backend by `get_network()`. Rows that fail are kept, so that `get_network()` can still recognize them as being in the input when a backend returns them. Rows with a missing `adj.pvalue` are not selected.

Usage

```
select_entities(
  entities,
```

```

    pvalue_cutoff = NULL,
    logfc_cutoff = NULL,
    direction = c("both", "up", "down"),
    include_infinite_fc = FALSE,
    force_include = NULL
  )

```

Arguments

entities	entity table from <code>prepare_entities()</code>
pvalue_cutoff	keep rows with adj.pvalue below this. NULL applies no cutoff.
logfc_cutoff	keep rows with abs(logFC) above this, on the log scale of the input. NULL applies no cutoff.
direction	"both", "up" (logFC > 0), or "down" (logFC < 0).
include_infinite_fc	whether rows with infinite logFC (detected in one condition only) are selected regardless of pvalue_cutoff and logfc_cutoff. direction still applies.
force_include	values of id, or "namespace:identifier" groundings (e.g. "HGNC:1234"), selected regardless of the cutoffs. Entities outside the table are added with <code>get_network(include_entities =)</code> instead.

Value

entities with included_in_query set, and user_added (TRUE for rows selected only through force_include).

Examples

```

input <- data.table::fread(system.file(
  "extdata/groupComparisonModel.csv",
  package = "MSstatsBioNet"
))
entities <- prepare_entities(input, entity_type = "protein",
  id_type = "uniprot")
entities <- select_entities(entities, pvalue_cutoff = 0.01,
  direction = "up")
table(entities$included_in_query)

```

subnetwork_query

How are my selected entities connected to each other?

Description

Asks which edges connect the selected entities directly, with no other nodes added. Only entities passed to `get_network()` as include_entities are added.

Usage

```
subnetwork_query()
```

Details

Uses the rows of entities with `included_in_query = TRUE`. Each node the backend returns is matched against all rows, so that it carries its statistics. Edges get `query_type = "subnetwork"`. For INDRA, the query goes to CoGEx `indra_subnetwork_relations`, and takes fewer than 400 groundings (`backend_capabilities(indra_backend())$max_nodes`).

Value

a SubnetworkQuery object, to pass to [get_network\(\)](#)

See Also

[network_queries](#) for the other questions

Examples

```
subnetwork_query()
```

validate_network	<i>Validate a network against the edge and node contract</i>
------------------	--

Description

Checks that a network returned by [getSubnetworkFromIndra](#), or built by hand from another source, has the columns, types, and vocabularies that the visualization and filtering functions rely on.

Usage

```
validate_network(network)
```

Arguments

`network` list with nodes and edges data.frames.

Details

Required edge columns: `source`, `target` (both matching `nodes$id`), `interaction` (an INDRA statement type such as "Activation" or "Complex"), `directed` (logical), `site` (PTM site on the target such as "S148", ;-joined when there are several, or NA), `confidence` (in [0, 1], or NA when the source provides no score), `evidence_count` (whole number, at least 1, not NA), `evidence_url`, `statement_id` (character), `backend_database`, and `query_type`. Edges of the symmetric statement types "Complex" and "Association" must have `directed = FALSE`. Several edges can share

a statement_id: an undirected statement can be listed in both directions, and a statement reaches every node with its grounding (see [get_network\(\)](#)).

Required node columns: id, entity_type (e.g. "protein", "ptm_site", "metabolite", "family"), entity_name, namespace and entity_id (the grounding, NA when unknown), measured (logical: the node is in the input data), included_in_query (logical: the node was part of the query), and node_role (why the node is in the network, e.g. "passed_cutoffs" or "user_added", or several joined by ";" after [merge_networks\(\)](#)). An id can repeat, once per PTM site row of the same protein. When present, site, has_measured_sites, logFC, and adj.pvalue are type-checked. Nodes with measured == FALSE must have NA statistics.

Confidence values are comparable within one backend_database, not across sources.

Value

network, invisibly. Stops with an error listing every problem found.

Examples

```
network <- list(
  nodes = data.frame(
    id = c("CHK1_HUMAN", "CDC25A_HUMAN"),
    entity_type = "protein",
    entity_name = c("CHEK1", "CDC25A"),
    namespace = "HGNC",
    entity_id = c("1925", "1725"),
    measured = TRUE,
    included_in_query = TRUE,
    node_role = "passed_cutoffs"
  ),
  edges = data.frame(
    source = "CHK1_HUMAN",
    target = "CDC25A_HUMAN",
    interaction = "Phosphorylation",
    directed = TRUE,
    site = "S76",
    confidence = 0.99,
    evidence_count = 12L,
    evidence_url = paste0("https://db.indra.bio/statements/",
                          "from_hash/-1234?format=html"),
    statement_id = "-1234",
    backend_database = "INDRA",
    query_type = "subnetwork"
  )
)
validate_network(network)
```

Index

.validateAnnotateProteinInfoFromIndraInput, 3

add_entity_values, 3, 23

annotateProteinInfoFromIndra, 5, 32

backend_capabilities, 6, 31, 36, 39, 40

backend_capabilities, IndraBackend-method (backend_capabilities), 6

backend_capabilities, NetworkBackend-method (backend_capabilities), 6

bootstrapTopicModels, 7

compareTopicModels, 8, 9

convert_ids, 3, 6, 11, 27, 30, 35, 41

convert_ids, IndraBackend-method (convert_ids), 11

convert_ids, NetworkBackend-method (convert_ids), 11

createWidget, 13

cytoscapeNetwork, 4, 12, 19, 20

cytoscapeNetworkOutput, 13

dataProcess, 32

decomposeSubnetworkByTopic, 7–10, 14, 17–19, 25

decomposeSubnetworkIntoHierarchicalTopics, 16, 17, 25, 44

deleteEdgeFromNetwork, 20

exportNetworkToHTML, 19, 20

filter_by_curation, 21, 23, 26, 33

filter_by_ptm_site, 4, 23

filterSubnetworkByContext, 16, 24, 28

get_curations, 21, 22, 26, 35

get_curations, IndraBackend-method (get_curations), 26

get_curations, NetworkBackend-method (get_curations), 26

get_entity_properties, 6, 12, 27, 35

get_entity_properties, IndraBackend-method (get_entity_properties), 27

get_entity_properties, NetworkBackend-method (get_entity_properties), 27

get_evidence, 8, 9, 16, 18, 24, 28, 35

get_evidence, IndraBackend-method (get_evidence), 28

get_evidence, NetworkBackend-method (get_evidence), 28

get_network, 3, 4, 7, 11, 22, 23, 26, 28, 29, 35–41, 45, 46, 48, 49

get_network, IndraBackend, SubnetworkQuery-method (get_network), 29

get_network, NetworkBackend, missing-method (get_network), 29

get_network, NetworkBackend, NetworkQuery-method (get_network), 29

getSubnetworkFromIndra, 7, 9, 15, 16, 18, 22, 32, 48

groupComparison, 5, 32

indra_backend, 6, 8, 9, 11, 16, 18, 22, 24, 26–28, 30, 34, 40

INDRA_DATABASE_SOURCES, 7, 30, 33

INDRA_DATABASE_SOURCES (indra_evidence_sources), 35

indra_evidence_sources, 35

INDRA_TEXT_MINED_SOURCES, 30

INDRA_TEXT_MINED_SOURCES (indra_evidence_sources), 35

IndraBackend-class (NetworkBackend-class), 40

load_network, 36, 45

merge_networks, 4, 31, 37, 45, 49

network_queries, 7, 30, 31, 39, 40, 48

NetworkBackend-class, 40

NetworkQuery-class, [40](#)

prepare_entities, [3](#), [11](#), [30](#), [37](#), [39](#), [41](#), [47](#)

previewNetworkInBrowser, [42](#)

print.topicHierarchy, [43](#)

renderCytoscapeNetwork, [44](#)

save_network, [31](#), [36](#), [37](#), [45](#)

select_entities, [30](#), [39](#), [41](#), [46](#)

subnetwork_query, [6](#), [30](#), [39](#), [40](#), [47](#)

SubnetworkQuery-class
 (NetworkQuery-class), [40](#)

validate_network, [12](#), [21](#), [31](#), [33](#), [36–39](#), [43](#),
 [45](#), [48](#)

wilcox.test, [10](#)