

Package: LBDDiscover (via r-universe)

June 2, 2026

Title Literature-Based Discovery Tools for Biomedical Research

Version 0.1.0

Date 2025-05-14

Description A suite of tools for literature-based discovery in biomedical research. Provides functions for retrieving scientific articles from PubMed and other NCBI databases, extracting biomedical entities (diseases, drugs, genes, etc.), building co-occurrence networks, and applying various discovery models including ABC, AnC, LSI, and BITOLA. The package also includes visualization tools for exploring discovered connections.

License GPL-3

URL <https://github.com/chaoliu-cl/LBDDiscover>,
<http://liu-chao.site/LBDDiscover/>,
<https://liu-chao.site/LBDDiscover/>

BugReports <https://github.com/chaoliu-cl/LBDDiscover/issues>

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

Depends R (>= 4.0.0)

Imports httr (>= 1.4.0), xml2 (>= 1.3.0), igraph (>= 1.2.0), Matrix (>= 1.3.0), utils, stats, grDevices, graphics, tools, rentrez (>= 1.2.0), jsonlite (>= 1.7.0)

Suggests openxlsx (>= 4.2.0), SnowballC (>= 0.7.0), visNetwork (>= 2.1.0), spacyr (>= 1.2.0), parallel, digest (>= 0.6.0), irlba (>= 2.3.0), knitr, rmarkdown, base64enc, reticulate, testthat (>= 3.0.0), mockery, covr, withr, htmltools

VignetteBuilder knitr

Config/testthat/edition 3

Config/pak/sysreqs libgplk-dev libxml2-dev libssl-dev

Repository <https://chaoliu-cl.r-universe.dev>

Date/Publication 2025-10-05 00:54:04 UTC

RemoteUrl <https://github.com/chaoliu-cl/lbdiscover>

RemoteRef HEAD

RemoteSha 60e965ad1fbd39738ecfab72cec9eff617055719

Contents

abc_model	3
abc_model_opt	5
abc_model_sig	6
abc_timeslice	7
anc_model	8
bitola_model	9
calc_bibliometrics	10
calc_doc_sim	10
clear_pubmed_cache	11
cluster_docs	12
compare_terms	13
create_citation_net	14
create_comat	14
create_report	15
create_sparse_comat	16
create_tdm	17
create_term_document_matrix	18
detect_lang	18
diversify_abc	19
enhance_abc_kb	20
eval_evidence	20
export_chord	21
export_chord_diagram	22
export_network	23
extract_entities	24
extract_entities_workflow	25
extract_ner	27
extract_ngrams	27
extract_terms	28
extract_topics	29
filter_by_type	30
find_abc_all	30
find_similar_docs	31
find_term	32
gen_report	32
get_dict_cache	33
get_pmc_fulltext	34

get_term_vars	35
get_type_dist	35
is_valid_biomedical_entity	36
load_dictionary	37
load_results	38
lsi_model	39
map_ontology	40
merge_entities	41
merge_results	42
min_results	43
ncbi_search	44
parallel_analysis	45
perm_test_abc	46
plot_heatmap	46
plot_network	48
prep_articles	49
preprocess_text	50
pubmed_search	51
query_external_api	52
query_mesh	52
query_umls	53
run_lbd	53
safe_diversify	54
sanitize_dictionary	56
save_results	57
segment_sentences	57
valid_entities	58
validate_abc	59
validate_biomedical_entity	60
validate_entity_comprehensive	60
validate_entity_with_nlp	61
validate_umls_key	62
vec_preprocess	62
vis_abc_heatmap	64
vis_heatmap	64
vis_network	65
visualize_abc_network	67

Index **68**

abc_model	<i>Apply the ABC model for literature-based discovery with improved filtering</i>
-----------	---

Description

This function implements the ABC model for literature-based discovery with enhanced term filtering and validation.

Usage

```

abc_model(
  co_matrix,
  a_term,
  c_term = NULL,
  min_score = 0.1,
  n_results = 100,
  scoring_method = c("multiplication", "average", "combined", "jaccard"),
  b_term_types = NULL,
  c_term_types = NULL,
  exclude_general_terms = TRUE,
  filter_similar_terms = TRUE,
  similarity_threshold = 0.8,
  enforce_strict_typing = TRUE,
  validation_method = "pattern"
)

```

Arguments

co_matrix	A co-occurrence matrix produced by create_comat().
a_term	Character string, the source term (A).
c_term	Character string, the target term (C). If NULL, all potential C terms will be evaluated.
min_score	Minimum score threshold for results.
n_results	Maximum number of results to return.
scoring_method	Method to use for scoring.
b_term_types	Character vector of entity types allowed for B terms.
c_term_types	Character vector of entity types allowed for C terms.
exclude_general_terms	Logical. If TRUE, excludes common general terms.
filter_similar_terms	Logical. If TRUE, filters out B-terms that are too similar to A-term.
similarity_threshold	Numeric. Maximum allowed string similarity between A and B terms.
enforce_strict_typing	Logical. If TRUE, enforces stricter entity type validation.
validation_method	Character. Method to use for entity validation: "pattern", "nlp", "api", or "comprehensive".

Value

A data frame with ranked discovery results.

Examples

```
# Create a simple example co-occurrence matrix
set.seed(123)
terms <- c("migraine", "headache", "pain", "serotonin", "sumatriptan", "CGRP")
n_terms <- length(terms)
co_matrix <- matrix(runif(n_terms^2, 0, 1), nrow = n_terms, ncol = n_terms)
rownames(co_matrix) <- colnames(co_matrix) <- terms
diag(co_matrix) <- 1 # Perfect self-similarity

# Add entity types
entity_types <- c("disease", "symptom", "symptom", "chemical", "drug", "protein")
names(entity_types) <- terms
attr(co_matrix, "entity_types") <- entity_types

# Apply ABC model
results <- abc_model(co_matrix, a_term = "migraine", min_score = 0.1, n_results = 5)
print(results)
```

abc_model_opt

Optimize ABC model calculations for large matrices

Description

This function implements an optimized version of the ABC model calculation that's more efficient for large co-occurrence matrices.

Usage

```
abc_model_opt(
  co_matrix,
  a_term,
  c_term = NULL,
  min_score = 0.1,
  n_results = 100,
  chunk_size = 500
)
```

Arguments

co_matrix	A co-occurrence matrix produced by create_cooccurrence_matrix().
a_term	Character string, the source term (A).
c_term	Character string, the target term (C). If NULL, all potential C terms will be evaluated.
min_score	Minimum score threshold for results.
n_results	Maximum number of results to return.
chunk_size	Number of B terms to process in each chunk.

Value

A data frame with ranked discovery results.

Examples

```
## Not run:
abc_results <- abc_model_opt(co_matrix, a_term = "migraine")

## End(Not run)
```

abc_model_sig	<i>Apply the ABC model with statistical significance testing</i>
---------------	--

Description

This function extends the ABC model with statistical significance testing to evaluate the strength of discovered connections.

Usage

```
abc_model_sig(
  co_matrix,
  a_term,
  c_term = NULL,
  a_type = NULL,
  c_type = NULL,
  min_score = 0.1,
  n_results = 100,
  n_permutations = 1000,
  scoring_method = c("multiplication", "average", "combined", "jaccard")
)
```

Arguments

co_matrix	A co-occurrence matrix produced by create_cooccurrence_matrix().
a_term	Character string, the source term (A).
c_term	Character string, the target term (C). If NULL, all potential C terms will be evaluated.
a_type	Character string, the entity type for A terms. If NULL, all types are considered.
c_type	Character string, the entity type for C terms. If NULL, all types are considered.
min_score	Minimum score threshold for results.
n_results	Maximum number of results to return.
n_permutations	Number of permutations for significance testing.
scoring_method	Method to use for scoring ABC connections.

Value

A data frame with ranked discovery results and p-values.

Examples

```
## Not run:
abc_results <- abc_model_sig(co_matrix, a_term = "migraine",
                             scoring_method = "combined")

## End(Not run)
```

abc_timeslice	<i>Apply time-sliced ABC model for validation</i>
---------------	---

Description

This function implements a time-sliced ABC model for validation. It uses historical data to predict connections that will appear in the future.

Usage

```
abc_timeslice(
  entity_data,
  time_column = "publication_year",
  split_time,
  a_term,
  a_type = NULL,
  c_type = NULL,
  min_score = 0.1,
  n_results = 100
)
```

Arguments

entity_data	A data frame of entity data with time information.
time_column	Name of the column containing time information.
split_time	Time point to split historical and future data.
a_term	Character string, the source term (A).
a_type	Character string, the entity type for A terms.
c_type	Character string, the entity type for C terms.
min_score	Minimum score threshold for results.
n_results	Maximum number of results to return.

Value

A list with prediction results and validation metrics.

Examples

```
## Not run:
validation <- abc_timeslice(entity_data,
                             time_column = "publication_year",
                             split_time = 2010,
                             a_term = "migraine")

## End(Not run)
```

anc_model	<i>ANC model for literature-based discovery with biomedical term filtering</i>
-----------	--

Description

This function implements an improved ANC model that ensures only biomedical terms are used as intermediaries.

Usage

```
anc_model(
  co_matrix,
  a_term,
  n_b_terms = 3,
  c_type = NULL,
  min_score = 0.1,
  n_results = 100,
  enforce_biomedical_terms = TRUE,
  b_term_types = c("protein", "gene", "chemical", "pathway", "drug", "disease",
                  "biological_process"),
  validation_function = is_valid_biomedical_entity
)
```

Arguments

co_matrix	A co-occurrence matrix produced by create_cooccurrence_matrix().
a_term	Character string, the source term (A).
n_b_terms	Number of intermediate B terms to consider.
c_type	Character string, the entity type for C terms. If NULL, all types are considered.
min_score	Minimum score threshold for results.
n_results	Maximum number of results to return.
enforce_biomedical_terms	Logical. If TRUE, enforces strict biomedical term filtering.
b_term_types	Character vector of entity types allowed for B terms.
validation_function	Function to validate biomedical terms.

Value

A data frame with ranked discovery results.

bitola_model	<i>Apply BITOLA-style discovery model</i>
--------------	---

Description

This function implements a BITOLA-style discovery model based on MeSH term co-occurrence and semantic type filtering.

Usage

```
bitola_model(  
  co_matrix,  
  a_term,  
  a_semantic_type = NULL,  
  c_semantic_type = NULL,  
  min_score = 0.1,  
  n_results = 100  
)
```

Arguments

co_matrix	A co-occurrence matrix produced by create_cooccurrence_matrix().
a_term	Character string, the source term (A).
a_semantic_type	Character string, the semantic type for A term.
c_semantic_type	Character string, the semantic type for C terms.
min_score	Minimum score threshold for results.
n_results	Maximum number of results to return.

Value

A data frame with ranked discovery results.

Examples

```
## Not run:  
bitola_results <- bitola_model(co_matrix, a_term = "migraine",  
                              a_semantic_type = "Disease",  
                              c_semantic_type = "Gene")  
  
## End(Not run)
```

calc_bibliometrics *Calculate basic bibliometric statistics*

Description

This function calculates basic bibliometric statistics from article data.

Usage

```
calc_bibliometrics(article_data, by_year = TRUE)
```

Arguments

article_data A data frame containing article data.
by_year Logical. If TRUE, calculates statistics by year.

Value

A list containing bibliometric statistics.

Examples

```
## Not run:  
stats <- calc_bibliometrics(article_data)  
  
## End(Not run)
```

calc_doc_sim *Calculate document similarity using TF-IDF and cosine similarity*

Description

This function calculates the similarity between documents using TF-IDF weighting and cosine similarity.

Usage

```
calc_doc_sim(  
  text_data,  
  text_column = "abstract",  
  min_term_freq = 2,  
  max_doc_freq = 0.9  
)
```

Arguments

text_data	A data frame containing text data.
text_column	Name of the column containing text to analyze.
min_term_freq	Minimum frequency for a term to be included.
max_doc_freq	Maximum document frequency (as a proportion) for a term to be included.

Value

A similarity matrix for the documents.

Examples

```
## Not run:  
sim_matrix <- calc_doc_sim(article_data, text_column = "abstract")  
  
## End(Not run)
```

clear_pubmed_cache *Clear PubMed cache*

Description

Removes all cached PubMed search results

Usage

```
clear_pubmed_cache()
```

Value

NULL invisibly

Examples

```
# Clear the PubMed cache (always safe to run)  
clear_pubmed_cache()
```

cluster_docs	<i>Cluster documents using K-means</i>
--------------	--

Description

This function clusters documents using K-means based on their TF-IDF vectors.

Usage

```
cluster_docs(  
  text_data,  
  text_column = "abstract",  
  n_clusters = 5,  
  min_term_freq = 2,  
  max_doc_freq = 0.9,  
  random_seed = 42  
)
```

Arguments

text_data	A data frame containing text data.
text_column	Name of the column containing text to analyze.
n_clusters	Number of clusters to create.
min_term_freq	Minimum frequency for a term to be included.
max_doc_freq	Maximum document frequency (as a proportion) for a term to be included.
random_seed	Seed for random number generation (for reproducibility).

Value

A data frame with the original data and cluster assignments.

Examples

```
## Not run:  
clustered_data <- cluster_docs(article_data, text_column = "abstract", n_clusters = 5)  
  
## End(Not run)
```

compare_terms	<i>Compare term frequencies between two corpora</i>
---------------	---

Description

This function compares term frequencies between two sets of articles.

Usage

```
compare_terms(  
  corpus1,  
  corpus2,  
  text_column = "abstract",  
  corpus1_name = "Corpus1",  
  corpus2_name = "Corpus2",  
  n = 100,  
  remove_stopwords = TRUE  
)
```

Arguments

corpus1	First corpus (data frame).
corpus2	Second corpus (data frame).
text_column	Name of the column containing the text to analyze.
corpus1_name	Name for the first corpus in the output.
corpus2_name	Name for the second corpus in the output.
n	Number of top terms to return.
remove_stopwords	Logical. If TRUE, removes stopwords.

Value

A data frame containing term frequency comparisons.

Examples

```
## Not run:  
comparison <- compare_terms(corpus1, corpus2,  
                             corpus1_name = "Migraine",  
                             corpus2_name = "Magnesium")  
  
## End(Not run)
```

create_citation_net *Create a citation network from article data*

Description

This function creates a citation network from article data. Note: Currently a placeholder as it requires citation data not available through basic PubMed queries.

Usage

```
create_citation_net(article_data, citation_data = NULL)
```

Arguments

article_data A data frame containing article data.
citation_data A data frame containing citation data (optional).

Value

An igraph object representing the citation network.

Examples

```
## Not run:  
network <- create_citation_net(article_data)  
  
## End(Not run)
```

create_comat *Create co-occurrence matrix without explicit entity type constraints*

Description

This function creates a co-occurrence matrix from entity data while preserving entity type information as an attribute without enforcing type constraints.

Usage

```
create_comat(  
  entity_data,  
  doc_id_col = "doc_id",  
  entity_col = "entity",  
  count_col = NULL,  
  type_col = "entity_type",  
  normalize = TRUE,  
  normalization_method = c("cosine", "jaccard", "dice")  
)
```

Arguments

entity_data	A data frame with document IDs and entities.
doc_id_col	Name of the column containing document IDs.
entity_col	Name of the column containing entity names.
count_col	Name of the column containing entity counts (optional).
type_col	Name of the column containing entity types (optional).
normalize	Logical. If TRUE, normalizes the co-occurrence matrix.
normalization_method	Method for normalization ("cosine", "jaccard", or "dice").

Value

A co-occurrence matrix with entity types stored as an attribute.

Examples

```
## Not run:
co_matrix <- create_comat(entities,
                          doc_id_col = "doc_id",
                          entity_col = "entity",
                          type_col = "entity_type")

## End(Not run)
```

create_report	<i>Generate a comprehensive discovery report</i>
---------------	--

Description

This function generates an HTML report summarizing discovery results without enforcing entity type constraints. It includes data validation to avoid errors with publication years and other data issues.

Usage

```
create_report(
  results,
  visualizations = NULL,
  articles = NULL,
  output_file = "discovery_report.html"
)
```

Arguments

results	A list containing discovery results from different approaches.
visualizations	A list containing file paths to visualizations.
articles	A data frame containing the original articles.
output_file	File path for the output HTML report.

Value

The file path of the created HTML report (invisibly).

create_sparse_comat *Create a sparse co-occurrence matrix*

Description

This function creates a sparse co-occurrence matrix from entity data, which is more memory-efficient for large datasets.

Usage

```
create_sparse_comat(  
  entity_data,  
  doc_id_col = "doc_id",  
  entity_col = "entity",  
  count_col = NULL,  
  type_col = NULL,  
  normalize = TRUE  
)
```

Arguments

entity_data	A data frame with document IDs and entities.
doc_id_col	Name of the column containing document IDs.
entity_col	Name of the column containing entity names.
count_col	Name of the column containing entity counts (optional).
type_col	Name of the column containing entity types (optional).
normalize	Logical. If TRUE, normalizes the co-occurrence matrix.

Value

A sparse matrix of entity co-occurrences.

Examples

```
## Not run:
co_matrix <- create_sparse_comat(entities,
                                doc_id_col = "doc_id",
                                entity_col = "entity")

## End(Not run)
```

create_tdm	<i>Create a term-document matrix from preprocessed text</i>
------------	---

Description

This function creates a term-document matrix from preprocessed text data.

Usage

```
create_tdm(preprocessed_data, min_df = 2, max_df = 0.9)
```

Arguments

preprocessed_data	A data frame with preprocessed text data.
min_df	Minimum document frequency for a term to be included.
max_df	Maximum document frequency (as a proportion) for a term to be included.

Value

A term-document matrix.

Examples

```
## Not run:
preprocessed <- preprocess_text(article_data, text_column = "abstract")
tdm <- create_tdm(preprocessed)

## End(Not run)
```

`create_term_document_matrix`*Create a term-document matrix from preprocessed text*

Description

This function creates a term-document matrix from preprocessed text data. It's a simplified version of `create_tdm()` for direct use in models.

Usage

```
create_term_document_matrix(preprocessed_data, min_df = 2, max_df = 0.9)
```

Arguments

<code>preprocessed_data</code>	A data frame with preprocessed text data.
<code>min_df</code>	Minimum document frequency for a term to be included.
<code>max_df</code>	Maximum document frequency (as a proportion) for a term to be included.

Value

A term-document matrix.

Examples

```
## Not run:  
preprocessed <- preprocess_text(article_data, text_column = "abstract")  
tdm <- create_term_document_matrix(preprocessed)  
  
## End(Not run)
```

`detect_lang`*Detect language of text*

Description

This function attempts to detect the language of a text string. It implements a simple n-gram based approach that doesn't require additional packages.

Usage

```
detect_lang(text, sample_size = 1000)
```

Arguments

text Text string to analyze
sample_size Maximum number of characters to sample for language detection

Value

Character string containing the ISO 639-1 language code

Examples

```
## Not run:  
lang <- detect_lang("This is English text")  
# Returns "en"  
  
## End(Not run)
```

diversify_abc *Enforce diversity in ABC model results*

Description

This function applies diversity enforcement to ABC model results by:

1. Removing duplicate paths to the same C term
2. Ensuring B term diversity by selecting top results from each B term group
3. Preventing A and C terms from appearing as B terms

Usage

```
diversify_abc(  
  abc_results,  
  diversity_method = c("both", "b_term_groups", "unique_c_paths"),  
  max_per_group = 3,  
  min_score = 0.1  
)
```

Arguments

abc_results A data frame containing ABC results.
diversity_method Method for enforcing diversity: "b_term_groups", "unique_c_paths", or "both".
max_per_group Maximum number of results to keep per B term or C term.
min_score Minimum score threshold for including connections.

Value

A data frame with diverse ABC results.

Examples

```
## Not run:  
diverse_results <- diversify_abc(abc_results)  
  
## End(Not run)
```

enhance_abc_kb	<i>Enhance ABC results with external knowledge</i>
----------------	--

Description

This function enhances ABC results with information from external knowledge bases.

Usage

```
enhance_abc_kb(abc_results, knowledge_base = c("umls", "mesh"), api_key = NULL)
```

Arguments

`abc_results` A data frame containing ABC results.
`knowledge_base` Character string, the knowledge base to use ("umls" or "mesh").
`api_key` Character string. API key for the knowledge base (if needed).

Value

A data frame with enhanced ABC results.

Examples

```
## Not run:  
enhanced_results <- enhance_abc_kb(abc_results, knowledge_base = "mesh")  
  
## End(Not run)
```

eval_evidence	<i>Evaluate literature support for discovery results</i>
---------------	--

Description

This function evaluates the top results by searching for supporting evidence in the literature for the connections.

Usage

```
eval_evidence(  
  results,  
  max_results = 5,  
  base_term = NULL,  
  max_articles = 5,  
  verbose = TRUE  
)
```

Arguments

results	The results to evaluate
max_results	Maximum number of results to evaluate (default: 5)
base_term	The base term for direct connection queries (e.g., "migraine")
max_articles	Maximum number of articles to retrieve per search (default: 5)
verbose	Logical; if TRUE, print evaluation results (default: TRUE)

Value

A list containing evaluation results

Examples

```
# Create example results for evaluation  
results <- data.frame(  
  a_term = rep("migraine", 2),  
  b_term = c("serotonin", "CGRP"),  
  c_term = c("sumatriptan", "fremanezumab"),  
  abc_score = c(0.8, 0.7),  
  c_type = rep("drug", 2)  
)  
  
## Not run:  
# This requires internet connection for PubMed search  
evaluation <- eval_evidence(results, max_results = 2, base_term = "migraine")  
print(names(evaluation))  
  
## End(Not run)
```

export_chord

Export interactive HTML chord diagram for ABC connections

Description

This function creates an HTML chord diagram visualization for ABC connections.

Usage

```
export_chord(  
  abc_results,  
  output_file = "abc_chord.html",  
  top_n = 50,  
  min_score = 0.1,  
  open = TRUE  
)
```

Arguments

abc_results	A data frame containing ABC results.
output_file	File path for the output HTML file.
top_n	Number of top results to visualize.
min_score	Minimum score threshold for including connections.
open	Logical. If TRUE, opens the HTML file after creation.

Value

The file path of the created HTML file (invisibly).

Examples

```
## Not run:  
export_chord(abc_results, output_file = "abc_chord.html")  
  
## End(Not run)
```

export_chord_diagram *Export interactive HTML chord diagram for ABC connections*

Description

This function creates an HTML chord diagram visualization for ABC connections, properly coloring the arcs based on whether each term is an A, B, or C term.

Usage

```
export_chord_diagram(  
  abc_results,  
  output_file = "abc_chord.html",  
  top_n = 50,  
  min_score = 0.1,  
  open = TRUE  
)
```

Arguments

abc_results	A data frame containing ABC results.
output_file	File path for the output HTML file.
top_n	Number of top results to visualize.
min_score	Minimum score threshold for including connections.
open	Logical. If TRUE, opens the HTML file after creation.

Value

The file path of the created HTML file (invisibly).

Examples

```
## Not run:
export_chord_diagram(abc_results, output_file = "abc_chord.html")

## End(Not run)
```

export_network	<i>Export ABC results to simple HTML network</i>
----------------	--

Description

This function exports ABC results to a simple HTML file with a visualization. If the visNetwork package is available, it will use it for a more interactive visualization.

Usage

```
export_network(
  abc_results,
  output_file = "abc_network.html",
  top_n = 50,
  min_score = 0.1,
  open = TRUE
)
```

Arguments

abc_results	A data frame containing ABC results from apply_abc_model().
output_file	File path for the output HTML file.
top_n	Number of top results to visualize.
min_score	Minimum score threshold for including connections.
open	Logical. If TRUE, opens the HTML file after creation.

Value

The file path of the created HTML file (invisibly).

Examples

```
## Not run:  
export_network(abc_results, output_file = "abc_network.html")  
  
## End(Not run)
```

extract_entities	<i>Extract and classify entities from text with multi-domain types</i>
------------------	--

Description

This function extracts entities from text and optionally assigns them to specific semantic categories based on dictionaries.

Usage

```
extract_entities(  
  text_data,  
  text_column = "abstract",  
  dictionary = NULL,  
  case_sensitive = FALSE,  
  overlap_strategy = c("priority", "all", "longest"),  
  sanitize_dict = TRUE  
)
```

Arguments

text_data	A data frame containing article text data.
text_column	Name of the column containing text to process.
dictionary	Combined dictionary or list of dictionaries for entity extraction.
case_sensitive	Logical. If TRUE, matching is case-sensitive.
overlap_strategy	How to handle terms that match multiple dictionaries: "priority", "all", or "longest".
sanitize_dict	Logical. If TRUE, sanitizes the dictionary before extraction.

Value

A data frame with extracted entities, their types, and positions.

Examples

```
# Create example text data
text_data <- data.frame(
  doc_id = 1:3,
  abstract = c(
    "Migraine is a neurological disorder causing severe headache and photophobia.",
    "Serotonin receptors play a role in migraine pathophysiology.",
    "Sumatriptan is an effective treatment for migraine attacks."
  )
)

# Create example dictionary
dictionary <- data.frame(
  term = c("migraine", "headache", "photophobia", "serotonin", "sumatriptan"),
  type = c("disease", "symptom", "symptom", "chemical", "drug")
)

# Extract entities
entities <- extract_entities(text_data, dictionary = dictionary)
print(entities)
```

extract_entities_workflow

Extract entities from text with improved efficiency using only base R

Description

This function provides a complete workflow for extracting entities from text using dictionaries from multiple sources, with improved performance and robust error handling.

Usage

```
extract_entities_workflow(
  text_data,
  text_column = "abstract",
  entity_types = c("disease", "drug", "gene"),
  dictionary_sources = c("local", "mesh", "umls"),
  additional_mesh_queries = NULL,
  sanitize = TRUE,
  api_key = NULL,
  custom_dictionary = NULL,
  max_terms_per_type = 200,
  verbose = TRUE,
  batch_size = 500,
  parallel = FALSE,
  num_cores = 2,
  cache_dictionaries = TRUE
)
```

Arguments

text_data	A data frame containing article text data.
text_column	Name of the column containing text to process.
entity_types	Character vector of entity types to include.
dictionary_sources	Character vector of sources for entity dictionaries.
additional_mesh_queries	Named list of additional MeSH queries.
sanitize	Logical. If TRUE, sanitizes dictionaries before extraction.
api_key	API key for UMLS access (if "umls" is in dictionary_sources).
custom_dictionary	A data frame containing custom dictionary entries to incorporate into the entity extraction process.
max_terms_per_type	Maximum number of terms to fetch per entity type. Default is 200.
verbose	Logical. If TRUE, prints detailed progress information.
batch_size	Number of documents to process in a single batch. Default is 500.
parallel	Logical. If TRUE, uses parallel processing when available. Default is FALSE.
num_cores	Number of cores to use for parallel processing. Default is 2.
cache_dictionaries	Logical. If TRUE, caches dictionaries for faster reuse. Default is TRUE.

Value

A data frame with extracted entities, their types, and positions.

Examples

```
# Create example text data
text_data <- data.frame(
  doc_id = 1:2,
  abstract = c(
    "Migraine is a neurological disorder.",
    "Serotonin plays a role in headache."
  )
)

# Extract entities using workflow
entities <- extract_entities_workflow(
  text_data,
  entity_types = c("disease", "chemical"),
  dictionary_sources = "local",
  max_terms_per_type = 10
)
print(head(entities))
```

extract_ner	<i>Perform named entity recognition on text</i>
-------------	---

Description

This function performs a simple dictionary-based named entity recognition. For more advanced NER, consider using external tools via reticulate.

Usage

```
extract_ner(  
  text,  
  entity_types = c("disease", "drug", "gene"),  
  custom_dictionaries = NULL  
)
```

Arguments

text	Character vector of texts to process
entity_types	Character vector of entity types to recognize
custom_dictionaries	List of custom dictionaries (named by entity type)

Value

A data frame containing found entities, their types, and positions

Examples

```
## Not run:  
entities <- extract_ner(abstracts,  
                        entity_types = c("disease", "drug", "gene"))  
  
## End(Not run)
```

extract_ngrams	<i>Extract n-grams from text</i>
----------------	----------------------------------

Description

This function extracts n-grams (sequences of n words) from text.

Usage

```
extract_ngrams(text, n = 1, min_freq = 2)
```

Arguments

text Character vector of texts to process
n Integer specifying the n-gram size (1 for unigrams, 2 for bigrams, etc.)
min_freq Minimum frequency to include an n-gram

Value

A data frame containing n-grams and their frequencies

Examples

```
## Not run:  
bigrams <- extract_ngrams(abstracts, n = 2)  
  
## End(Not run)
```

extract_terms *Extract common terms from a corpus*

Description

This function extracts and counts the most common terms in a corpus.

Usage

```
extract_terms(  
  article_data,  
  text_column = "abstract",  
  n = 100,  
  remove_stopwords = TRUE,  
  min_word_length = 3  
)
```

Arguments

article_data A data frame containing article data.
text_column Name of the column containing the text to analyze.
n Number of top terms to return.
remove_stopwords Logical. If TRUE, removes stopwords.
min_word_length Minimum word length to include.

Value

A data frame containing term counts.

Examples

```
## Not run:  
common_terms <- extract_terms(article_data, text_column = "abstract")  
  
## End(Not run)
```

extract_topics	<i>Apply topic modeling to a corpus</i>
----------------	---

Description

This function implements a simple non-negative matrix factorization (NMF) approach to topic modeling, without requiring additional packages.

Usage

```
extract_topics(  
  text_data,  
  text_column = "abstract",  
  n_topics = 5,  
  max_terms = 10,  
  n_iterations = 50  
)
```

Arguments

text_data	A data frame containing the text data
text_column	Name of the column containing the text
n_topics	Number of topics to extract
max_terms	Maximum number of terms per topic to return
n_iterations	Number of iterations for the NMF algorithm

Value

A list containing topic-term and document-topic matrices

Examples

```
## Not run:  
topics <- extract_topics(article_data, text_column = "abstract", n_topics = 5)  
  
## End(Not run)
```

filter_by_type	<i>Filter a co-occurrence matrix by entity type</i>
----------------	---

Description

Filter a co-occurrence matrix by entity type

Usage

```
filter_by_type(co_matrix, types)
```

Arguments

co_matrix	A co-occurrence matrix produced by create_typed_comat().
types	Character vector of entity types to include.

Value

A filtered co-occurrence matrix.

Examples

```
## Not run:  
# Keep only disease and drug entities  
filtered_matrix <- filter_by_type(co_matrix, types = c("disease", "drug"))  
  
## End(Not run)
```

find_abc_all	<i>Find all potential ABC connections</i>
--------------	---

Description

This function finds all potential ABC connections in a co-occurrence matrix.

Usage

```
find_abc_all(  
  co_matrix,  
  a_type = NULL,  
  c_type = NULL,  
  min_score = 0.1,  
  n_results = 1000  
)
```

Arguments

co_matrix	A co-occurrence matrix produced by create_comat().
a_type	Character string, the entity type for A terms.
c_type	Character string, the entity type for C terms.
min_score	Minimum score threshold for results.
n_results	Maximum number of results to return.

Value

A data frame with ranked discovery results.

Examples

```
## Not run:  
all_abc <- find_abc_all(co_matrix, a_type = "source", c_type = "target")  
  
## End(Not run)
```

find_similar_docs *Find similar documents for a given document*

Description

This function finds documents similar to a given document based on TF-IDF and cosine similarity.

Usage

```
find_similar_docs(text_data, doc_id, text_column = "abstract", n_similar = 5)
```

Arguments

text_data	A data frame containing text data.
doc_id	ID of the document to find similar documents for.
text_column	Name of the column containing text to analyze.
n_similar	Number of similar documents to return.

Value

A data frame with similar documents and their similarity scores.

Examples

```
## Not run:  
similar_docs <- find_similar_docs(article_data, doc_id = 1,  
                                text_column = "abstract", n_similar = 5)  
  
## End(Not run)
```

find_term	<i>Find primary term in co-occurrence matrix</i>
-----------	--

Description

This function verifies that the primary term exists in the co-occurrence matrix, and if not, attempts to find a suitable variation.

Usage

```
find_term(co_matrix, primary_term, verbose = TRUE)
```

Arguments

co_matrix	The co-occurrence matrix
primary_term	The primary term to find
verbose	Logical; if TRUE, print status messages (default: TRUE)

Value

The found term (either exact match or variation)

Examples

```
# Create example co-occurrence matrix
terms <- c("migraine", "headache", "pain", "serotonin")
co_matrix <- matrix(runif(16, 0, 1), nrow = 4, ncol = 4)
rownames(co_matrix) <- colnames(co_matrix) <- terms

# Find term in matrix
found_term <- find_term(co_matrix, "migraine")
print(found_term)
```

gen_report	<i>Generate comprehensive discovery report</i>
------------	--

Description

This function creates a comprehensive HTML report from discovery results and visualizations.

Usage

```
gen_report(  
  results_list,  
  visualizations = NULL,  
  articles = NULL,  
  output_file = "discoveries.html",  
  verbose = TRUE  
)
```

Arguments

results_list	A list of result data frames from different approaches
visualizations	A list with paths to visualization files
articles	Prepared article data
output_file	Filename for the output HTML report
verbose	Logical; if TRUE, print status messages (default: TRUE)

Value

Invisible output_file path

Examples

```
# Create example data for report generation  
results_list <- list(  
  abc_results = data.frame(  
    a_term = "migraine",  
    c_term = "sumatriptan",  
    abc_score = 0.8  
  )  
)  
  
# Generate report to temporary file  
temp_file <- tempfile(fileext = ".html")  
gen_report(results_list, output_file = temp_file)
```

get_dict_cache	<i>Get dictionary cache environment</i>
----------------	---

Description

Get dictionary cache environment

Usage

```
get_dict_cache()
```

Value

The environment containing cached dictionary data

Examples

```
# Get the dictionary cache environment
cache_env <- get_dict_cache()
print(ls(cache_env))
```

get_pmc_fulltext	<i>Retrieve full text from PubMed Central</i>
------------------	---

Description

This function retrieves full text articles from PubMed Central.

Usage

```
get_pmc_fulltext(pmids, api_key = NULL)
```

Arguments

pmids	Character vector of PubMed IDs.
api_key	Character string. NCBI API key for higher rate limits (optional).

Value

A data frame containing article metadata and full text.

Examples

```
## Not run:
full_texts <- get_pmc_fulltext(c("12345678", "23456789"))

## End(Not run)
```

get_term_vars	<i>Extract term variations from text corpus</i>
---------------	---

Description

This function identifies variations of a primary term within a corpus of articles.

Usage

```
get_term_vars(articles, primary_term, text_col = "abstract")
```

Arguments

articles	A data frame containing article data with text columns
primary_term	The primary term to find variations of
text_col	Name of the column containing the text to search

Value

A character vector of unique term variations, sorted by length

Examples

```
# Create example articles
articles <- data.frame(
  abstract = c(
    "Migraine headaches are debilitating",
    "Migraines affect quality of life",
    "Migraine disorders require treatment"
  )
)

# Get term variations
variations <- get_term_vars(articles, "migrain")
print(variations)
```

get_type_dist	<i>Get entity type distribution from co-occurrence matrix</i>
---------------	---

Description

Get entity type distribution from co-occurrence matrix

Usage

```
get_type_dist(co_matrix)
```

Arguments

co_matrix A co-occurrence matrix produced by create_typed_comat().

Value

A data frame with entity type counts and percentages.

Examples

```
## Not run:  
type_dist <- get_type_dist(co_matrix)  
  
## End(Not run)
```

is_valid_biomedical_entity

Determine if a term is likely a specific biomedical entity with improved accuracy

Description

Determine if a term is likely a specific biomedical entity with improved accuracy

Usage

```
is_valid_biomedical_entity(term, claimed_type = NULL)
```

Arguments

term Character string, the term to check
claimed_type Character string, the claimed entity type of the term

Value

Logical, TRUE if the term is likely a valid biomedical entity, FALSE otherwise

Examples

```
# Test biomedical entity validation  
is_valid_biomedical_entity("migraine", "disease") # Should return TRUE  
is_valid_biomedical_entity("receptor", "protein") # Should return TRUE  
is_valid_biomedical_entity("optimization", "chemical") # Should return FALSE  
is_valid_biomedical_entity("europe", "disease") # Should return FALSE
```

load_dictionary	<i>Load biomedical dictionaries with improved error handling</i>
-----------------	--

Description

This function loads pre-defined biomedical dictionaries or fetches terms from MeSH/UMLS.

Usage

```
load_dictionary(
  dictionary_type = NULL,
  custom_path = NULL,
  source = c("local", "mesh", "umls"),
  api_key = NULL,
  n_terms = 200,
  mesh_query = NULL,
  semantic_type_filter = NULL,
  sanitize = TRUE,
  extended_mesh = FALSE,
  mesh_queries = NULL
)
```

Arguments

dictionary_type	Type of dictionary to load. For local dictionaries, limited to "disease", "drug", "gene". For MeSH and UMLS, expanded to include more semantic categories.
custom_path	Optional path to a custom dictionary file.
source	The source to fetch terms from: "local", "mesh", or "umls".
api_key	UMLS API key for authentication (required if source = "umls").
n_terms	Number of terms to fetch.
mesh_query	Additional query to filter MeSH terms (only if source = "mesh").
semantic_type_filter	Filter by semantic type (used mainly with UMLS).
sanitize	Logical. If TRUE, sanitizes the dictionary terms.
extended_mesh	Logical. If TRUE and source is "mesh", uses PubMed search for additional terms.
mesh_queries	Named list of MeSH queries for different categories (only if extended_mesh = TRUE).

Value

A data frame containing the dictionary.

Examples

```
# Load a disease dictionary from local source
disease_dict <- load_dictionary("disease", source = "local")
head(disease_dict)

# Load with custom terms
custom_dict <- data.frame(
  term = c("migraine", "headache", "photophobia"),
  type = c("disease", "symptom", "symptom"),
  id = c("D001", "D002", "D003"),
  source = rep("custom", 3)
)
print(custom_dict)
```

load_results

Load saved results from a file

Description

This function loads previously saved results from a file.

Usage

```
load_results(file_path)
```

Arguments

file_path File path to load the results from.

Value

A data frame containing the loaded results.

Examples

```
## Not run:
results <- load_results("search_results.csv")

## End(Not run)
```

l _{si} _model	<i>LSI model with enhanced biomedical term filtering and NLP verification</i>
------------------------	---

Description

This function implements an improved LSI model that more rigorously filters out non-biomedical terms from the results to ensure clinical relevance.

Usage

```
lsi_model(
  term_doc_matrix,
  a_term,
  n_factors = 100,
  n_results = 100,
  enforce_biomedical_terms = TRUE,
  c_term_types = NULL,
  entity_types = NULL,
  validation_function = is_valid_biomedical_entity,
  min_word_length = 3,
  use_nlp = TRUE,
  nlp_threshold = 0.7
)
```

Arguments

term_doc_matrix	A term-document matrix.
a_term	Character string, the source term (A).
n_factors	Number of factors to use in LSI.
n_results	Maximum number of results to return.
enforce_biomedical_terms	Logical. If TRUE, enforces strict biomedical term filtering.
c_term_types	Character vector of entity types allowed for C terms.
entity_types	Named vector of entity types (if NULL, will try to detect).
validation_function	Function to validate biomedical terms.
min_word_length	Minimum word length to include.
use_nlp	Logical. If TRUE, uses NLP-based validation for biomedical terms.
nlp_threshold	Numeric between 0 and 1. Minimum confidence for NLP validation.

Value

A data frame with ranked discovery results.

`map_ontology`*Map terms to biomedical ontologies*

Description

This function maps terms to standard biomedical ontologies like MeSH or UMLS.

Usage

```
map_ontology(  
  terms,  
  ontology = c("mesh", "umls"),  
  api_key = NULL,  
  fuzzy_match = FALSE,  
  similarity_threshold = 0.8,  
  mesh_query = NULL,  
  semantic_types = NULL,  
  dictionary_type = "disease"  
)
```

Arguments

<code>terms</code>	Character vector of terms to map
<code>ontology</code>	Character string. The ontology to use: "mesh" or "umls"
<code>api_key</code>	UMLS API key (required if ontology = "umls")
<code>fuzzy_match</code>	Logical. If TRUE, allows fuzzy matching of terms
<code>similarity_threshold</code>	Numeric between 0 and 1. Minimum similarity for fuzzy matching
<code>mesh_query</code>	Additional query to filter MeSH terms (only if ontology = "mesh")
<code>semantic_types</code>	Vector of semantic types to filter UMLS results
<code>dictionary_type</code>	Type of dictionary to use ("disease", "drug", "gene", etc.)

Value

A data frame with mapped terms and ontology identifiers

Examples

```
## Not run:  
# Map terms to MeSH  
mesh_mappings <- map_ontology(c("headache", "migraine"), ontology = "mesh")  
  
# Map terms to UMLS with API key  
umls_mappings <- map_ontology(c("headache", "migraine"), ontology = "umls",  
                             api_key = "your_api_key")
```

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

merge_entities	<i>Combine and deduplicate entity datasets</i>
----------------	--

Description

This function combines custom and standard entity datasets, handling the case where one or both might be empty, and removes duplicates.

Usage

```
merge_entities(  
  custom_entities,  
  standard_entities,  
  primary_term,  
  primary_type = "disease",  
  verbose = TRUE  
)
```

Arguments

custom_entities	Data frame of custom entities (can be NULL)
standard_entities	Data frame of standard entities (can be NULL)
primary_term	The primary term of interest
primary_type	The entity type of the primary term (default: "disease")
verbose	Logical; if TRUE, print status messages (default: TRUE)

Value

A data frame of combined entities

Examples

```
# Create example entity datasets  
custom_entities <- data.frame(  
  doc_id = c(1, 1, 2),  
  entity = c("migraine", "headache", "pain"),  
  entity_type = c("disease", "symptom", "symptom"),  
  start_pos = c(1, 10, 5),  
  end_pos = c(8, 18, 9),  
  sentence = c("sent1", "sent1", "sent2"),  
  frequency = c(2, 1, 1)  
)
```

```
standard_entities <- data.frame(  
  doc_id = c(1, 2, 2),  
  entity = c("serotonin", "migraine", "therapy"),  
  entity_type = c("chemical", "disease", "treatment"),  
  start_pos = c(20, 1, 15),  
  end_pos = c(29, 8, 22),  
  sentence = c("sent1", "sent2", "sent2"),  
  frequency = c(1, 1, 1)  
)  
  
# Merge entities  
merged <- merge_entities(custom_entities, standard_entities, "migraine")  
print(merged)
```

merge_results

Merge multiple search results

Description

This function merges multiple search results into a single data frame.

Usage

```
merge_results(..., remove_duplicates = TRUE)
```

Arguments

... Data frames containing search results.
remove_duplicates Logical. If TRUE, removes duplicate articles.

Value

A merged data frame.

Examples

```
## Not run:  
merged_results <- merge_results(results1, results2, results3)  
  
## End(Not run)
```

min_results	<i>Ensure minimum results for visualization</i>
-------------	---

Description

This function ensures there are sufficient results for visualization, creating placeholder data if necessary.

Usage

```
min_results(  
  diverse_results,  
  top_results,  
  a_term,  
  min_results = 3,  
  fallback_count = 15,  
  verbose = TRUE  
)
```

Arguments

diverse_results	Current diversified results
top_results	Original top results
a_term	The primary term for the analysis
min_results	Minimum number of desired results (default: 3)
fallback_count	Number of top results to use as fallback (default: 15)
verbose	Logical; if TRUE, print status messages (default: TRUE)

Value

A data frame with sufficient results for visualization

Examples

```
# Create example diverse results (empty case)  
diverse_results <- data.frame()  
top_results <- data.frame(  
  a_term = rep("migraine", 3),  
  b_term = c("serotonin", "CGRP", "cortisol"),  
  c_term = c("sumatriptan", "fremanezumab", "propranolol"),  
  abc_score = c(0.8, 0.7, 0.6)  
)  
  
# Ensure minimum results  
final_results <- min_results(diverse_results, top_results, "migraine")  
print(final_results)
```

`ncbi_search`*Search NCBI databases for articles or data*

Description

This function searches various NCBI databases using the E-utilities API via the `rentrez` package.

Usage

```
ncbi_search(  
  query,  
  database = "pubmed",  
  max_results = 1000,  
  use_mesh = FALSE,  
  date_range = NULL,  
  api_key = NULL,  
  retry_count = 3,  
  retry_delay = 2  
)
```

Arguments

<code>query</code>	Character string containing the search query.
<code>database</code>	Character string. The NCBI database to search (e.g., "pubmed", "pmc", "gene", "protein").
<code>max_results</code>	Maximum number of results to return.
<code>use_mesh</code>	Logical. If TRUE, will attempt to map query terms to MeSH terms (for PubMed only).
<code>date_range</code>	Character vector of length 2 with start and end dates in format "YYYY/MM/DD".
<code>api_key</code>	Character string. NCBI API key for higher rate limits (optional).
<code>retry_count</code>	Integer. Number of times to retry failed requests.
<code>retry_delay</code>	Integer. Delay between retries in seconds.

Value

A data frame containing the search results with IDs, titles, and other metadata.

Examples

```
## Not run:  
results <- ncbi_search("migraine headache", database = "pubmed", max_results = 100)  
gene_results <- ncbi_search("BRCA1", database = "gene", max_results = 10)  
  
## End(Not run)
```

parallel_analysis	<i>Apply parallel processing for document analysis</i>
-------------------	--

Description

This function uses parallel processing to analyze documents faster.

Usage

```
parallel_analysis(  
  text_data,  
  analysis_function,  
  text_column = "abstract",  
  ...,  
  n_cores = NULL  
)
```

Arguments

text_data	A data frame containing text data.
analysis_function	Function to apply to each document.
text_column	Name of the column containing text to analyze.
...	Additional arguments passed to the analysis function.
n_cores	Number of cores to use for parallel processing. If NULL, uses all available cores minus 1.

Value

A data frame with analysis results.

Examples

```
## Not run:  
# Define a simple analysis function  
count_words <- function(text) {  
  words <- unlist(strsplit(tolower(text), "\\s+"))  
  return(length(words))  
}  
  
# Apply parallel processing  
results <- parallel_analysis(article_data, count_words, text_column = "abstract")  
  
## End(Not run)
```

perm_test_abc	<i>Perform randomization test for ABC model</i>
---------------	---

Description

This function assesses the significance of ABC model results through randomization. It generates a null distribution by permuting the co-occurrence matrix.

Usage

```
perm_test_abc(abc_results, co_matrix, n_permutations = 1000, alpha = 0.05)
```

Arguments

abc_results	A data frame containing ABC results.
co_matrix	The co-occurrence matrix used to generate the ABC results.
n_permutations	Number of permutations to perform.
alpha	Significance level.

Value

A data frame with ABC results and permutation-based significance measures.

Examples

```
## Not run:  
randomized_results <- perm_test_abc(abc_results, co_matrix, n_permutations = 1000)  
  
## End(Not run)
```

plot_heatmap	<i>Create heatmap visualization from results</i>
--------------	--

Description

This function creates a heatmap visualization from ABC results.

Usage

```
plot_heatmap(
  results,
  output_file = "heatmap.png",
  width = 1200,
  height = 900,
  resolution = 120,
  top_n = 15,
  min_score = 1e-04,
  color_palette = "blues",
  show_entity_types = TRUE,
  verbose = TRUE
)
```

Arguments

results	The results to visualize
output_file	Filename for the output PNG (default: "heatmap.png")
width	Width of the output image (default: 1200)
height	Height of the output image (default: 900)
resolution	Resolution of the output image (default: 120)
top_n	Maximum number of results to include (default: 15)
min_score	Minimum score threshold (default: 0.0001)
color_palette	Color palette for the heatmap (default: "blues")
show_entity_types	Logical; if TRUE, show entity types (default: TRUE)
verbose	Logical; if TRUE, print status messages (default: TRUE)

Value

Invisible NULL (creates a file as a side effect)

Examples

```
# Create example results for visualization
results <- data.frame(
  a_term = rep("migraine", 4),
  b_term = c("serotonin", "CGRP", "cortisol", "dopamine"),
  c_term = c("sumatriptan", "fremanezumab", "propranolol", "amitriptyline"),
  abc_score = c(0.8, 0.7, 0.6, 0.5),
  b_type = c("chemical", "protein", "hormone", "chemical"),
  c_type = rep("drug", 4)
)

# These require graphics capabilities
plot_heatmap(results, output_file = tempfile(fileext = ".png"))
```

```
plot_network(results, output_file = tempfile(fileext = ".png"))
```

plot_network

Create network visualization from results

Description

This function creates a network visualization from ABC results.

Usage

```
plot_network(
  results,
  output_file = "network.png",
  width = 1200,
  height = 900,
  resolution = 120,
  top_n = 15,
  min_score = 1e-04,
  node_size_factor = 5,
  color_by = "type",
  title = "Network Visualization",
  show_entity_types = TRUE,
  label_size = 1,
  verbose = TRUE
)
```

Arguments

results	The results to visualize
output_file	Filename for the output PNG (default: "network.png")
width	Width of the output image (default: 1200)
height	Height of the output image (default: 900)
resolution	Resolution of the output image (default: 120)
top_n	Maximum number of results to include (default: 15)
min_score	Minimum score threshold (default: 0.0001)
node_size_factor	Factor for scaling node sizes (default: 5)
color_by	Column to use for node colors (default: "type")
title	Plot title (default: "Network Visualization")
show_entity_types	Logical; if TRUE, show entity types (default: TRUE)
label_size	Relative size for labels (default: 1.0)
verbose	Logical; if TRUE, print status messages (default: TRUE)

Value

Invisible NULL (creates a file as a side effect)

Examples

```
# Create example results for visualization
results <- data.frame(
  a_term = rep("migraine", 4),
  b_term = c("serotonin", "CGRP", "cortisol", "dopamine"),
  c_term = c("sumatriptan", "fremanezumab", "propranolol", "amitriptyline"),
  abc_score = c(0.8, 0.7, 0.6, 0.5),
  b_type = c("chemical", "protein", "hormone", "chemical"),
  c_type = rep("drug", 4)
)

# These require graphics capabilities
plot_heatmap(results, output_file = tempfile(fileext = ".png"))
plot_network(results, output_file = tempfile(fileext = ".png"))
```

prep_articles

Prepare articles for report generation

Description

This function ensures article data is valid for report generation, particularly handling publication years.

Usage

```
prep_articles(articles, verbose = TRUE)
```

Arguments

articles The article data frame (can be NULL)
verbose Logical; if TRUE, print status messages (default: TRUE)

Value

A data frame of articles with validated publication years

Examples

```
# Create example article data
articles <- data.frame(
  title = c("Migraine Study 1", "Headache Research"),
  publication_year = c("2020", "not_a_year")
)

# Prepare articles
prepared <- prep_articles(articles)
print(prepared)
```

preprocess_text	<i>Preprocess article text</i>
-----------------	--------------------------------

Description

This function preprocesses article text for further analysis.

Usage

```
preprocess_text(
  text_data,
  text_column = "abstract",
  remove_stopwords = TRUE,
  custom_stopwords = NULL,
  stem_words = FALSE,
  min_word_length = 3,
  max_word_length = 50
)
```

Arguments

text_data	A data frame containing article text data (title, abstract, etc.).
text_column	Name of the column containing text to process.
remove_stopwords	Logical. If TRUE, removes stopwords.
custom_stopwords	Character vector of additional stopwords to remove.
stem_words	Logical. If TRUE, applies stemming to words.
min_word_length	Minimum word length to keep.
max_word_length	Maximum word length to keep.

Value

A data frame with processed text and extracted terms.

Examples

```
## Not run:
processed_data <- preprocess_text(article_data, text_column = "abstract")

## End(Not run)
```

pubmed_search	<i>Search PubMed for articles with optimized performance</i>
---------------	--

Description

This function searches PubMed using the NCBI E-utilities API via the `rentrez` package. The implementation includes optimizations for speed, memory efficiency, and reliability.

Usage

```
pubmed_search(
  query,
  max_results = 1000,
  use_mesh = FALSE,
  date_range = NULL,
  api_key = NULL,
  batch_size = 200,
  verbose = TRUE,
  use_cache = TRUE,
  retry_count = 3,
  retry_delay = 1
)
```

Arguments

<code>query</code>	Character string containing the search query.
<code>max_results</code>	Maximum number of results to return.
<code>use_mesh</code>	Logical. If TRUE, will attempt to map query terms to MeSH terms.
<code>date_range</code>	Character vector of length 2 with start and end dates in format "YYYY/MM/DD".
<code>api_key</code>	Character string. NCBI API key for higher rate limits (optional).
<code>batch_size</code>	Integer. Number of records to fetch in each batch (default: 200).
<code>verbose</code>	Logical. If TRUE, prints progress information.
<code>use_cache</code>	Logical. If TRUE, cache results to avoid redundant API calls.
<code>retry_count</code>	Integer. Number of times to retry failed API calls.
<code>retry_delay</code>	Integer. Initial delay between retries in seconds.

Value

A data frame containing the search results with PubMed IDs, titles, and other metadata.

Examples

```
## Not run:
results <- pubmed_search("migraine headache", max_results = 100)

## End(Not run)
```

query_external_api	<i>Query external biomedical APIs to validate entity types</i>
--------------------	--

Description

Query external biomedical APIs to validate entity types

Usage

```
query_external_api(term, claimed_type)
```

Arguments

term	Character string, the term to validate
claimed_type	Character string, the claimed entity type

Value

Logical indicating if the term was found in the appropriate database

query_mesh	<i>Query for MeSH terms using E-utilities</i>
------------	---

Description

Query for MeSH terms using E-utilities

Usage

```
query_mesh(term, api_key = NULL)
```

Arguments

term	Character string, the term to query.
api_key	Character string. NCBI API key (optional).

Value

A data frame with MeSH information for the term.

query_umls	<i>Query UMLS for term information</i>
------------	--

Description

Query UMLS for term information

Usage

```
query_umls(term, api_key, version = "current")
```

Arguments

term	Character string, the term to query.
api_key	Character string. UMLS API key.
version	Character string. UMLS version to use.

Value

A data frame with UMLS information for the term.

run_lbd	<i>Perform comprehensive literature-based discovery without type constraints</i>
---------	--

Description

This function performs a comprehensive literature-based discovery analysis using multiple approaches without enforcing entity type constraints.

Usage

```
run_lbd(
  search_query,
  a_term,
  max_results = 100,
  discovery_approaches = c("abc", "anc", "lsi", "bitola"),
  include_visualizations = TRUE,
  output_file = "discovery_report.html",
  api_key = NULL,
  dictionary_sources = c("local", "mesh", "umls"),
  entity_categories = c("disease", "drug", "gene")
)
```

Arguments

search_query	Character string, the search query for retrieving initial articles.
a_term	Character string, the source term (A) for discovery.
max_results	Maximum number of results to return for each approach.
discovery_approaches	Character vector, the discovery approaches to use.
include_visualizations	Logical. If TRUE, generates visualizations.
output_file	File path for the output report.
api_key	Character string. API key for PubMed and other services.
dictionary_sources	Character vector. Sources for entity dictionaries: "local", "mesh", "umls".
entity_categories	Character vector. Entity categories to include.

Value

A list containing discovery results from all approaches.

Examples

```
## Not run:
discovery_results <- run_lbd(
  search_query = "migraine headache",
  a_term = "migraine",
  discovery_approaches = c("abc", "anc", "lsi"),
  dictionary_sources = c("mesh"),
  entity_categories = c("disease", "drug", "gene", "protein")
)

## End(Not run)
```

safe_diversify

Diversify ABC results with error handling

Description

This function diversifies ABC results to avoid redundancy, with error handling to ensure results are always returned.

Usage

```
safe_diversify(  
  top_results,  
  diversity_method = "both",  
  max_per_group = 5,  
  min_score = 1e-04,  
  min_results = 5,  
  fallback_count = 15,  
  verbose = TRUE  
)
```

Arguments

top_results	The top ABC results to diversify
diversity_method	Method for diversification (default: "both")
max_per_group	Maximum results per group (default: 5)
min_score	Minimum score threshold (default: 0.0001)
min_results	Minimum number of desired results (default: 5)
fallback_count	Number of top results to use if diversification fails (default: 15)
verbose	Logical; if TRUE, print status messages (default: TRUE)

Value

A data frame of diversified results

Examples

```
# Create example results  
top_results <- data.frame(  
  a_term = rep("migraine", 6),  
  b_term = c("serotonin", "serotonin", "CGRP", "CGRP", "cortisol", "dopamine"),  
  c_term = c("sumatriptan", "rizatriptan", "fremanezumab", "galcanezumab", "propranolol", "amitriptyline"),  
  abc_score = c(0.8, 0.75, 0.7, 0.65, 0.6, 0.55)  
)  
  
# Apply diversification  
diverse_results <- safe_diversify(top_results, max_per_group = 2)  
print(diverse_results)
```

sanitize_dictionary *Enhanced sanitize dictionary function*

Description

This function sanitizes dictionary terms to ensure they're valid for entity extraction.

Usage

```
sanitize_dictionary(  
  dictionary,  
  term_column = "term",  
  type_column = "type",  
  validate_types = TRUE,  
  verbose = TRUE  
)
```

Arguments

dictionary	A data frame containing dictionary terms.
term_column	The name of the column containing the terms to sanitize.
type_column	The name of the column containing entity types.
validate_types	Logical. If TRUE, validates terms against their claimed type.
verbose	Logical. If TRUE, prints information about the filtering process.

Value

A data frame with sanitized terms.

Examples

```
# Create a dictionary with problematic terms  
dirty_dict <- data.frame(  
  term = c("migraine", "europe", "optimization", "receptor", "123", ""),  
  type = c("disease", "location", "process", "protein", "number", "empty")  
)  
  
# Sanitize the dictionary  
clean_dict <- sanitize_dictionary(dirty_dict)  
print(clean_dict)
```

save_results	<i>Save search results to a file</i>
--------------	--------------------------------------

Description

This function saves search results to a file.

Usage

```
save_results(results, file_path, format = c("csv", "rds", "xlsx"))
```

Arguments

results	A data frame containing search results.
file_path	File path to save the results.
format	File format to use. One of "csv", "rds", or "xlsx".

Value

The file path (invisibly).

Examples

```
## Not run:  
save_results(search_results, file_path = "search_results.csv")  
  
## End(Not run)
```

segment_sentences	<i>Perform sentence segmentation on text</i>
-------------------	--

Description

This function splits text into sentences.

Usage

```
segment_sentences(text)
```

Arguments

text	Character vector of texts to process
------	--------------------------------------

Value

A list where each element contains a character vector of sentences

Examples

```
## Not run:
sentences <- segment_sentences(abstracts)

## End(Not run)
```

valid_entities	<i>Filter entities to include only valid biomedical terms</i>
----------------	---

Description

This function applies validation to ensure only legitimate biomedical entities are included, while preserving trusted terms.

Usage

```
valid_entities(
  entities,
  primary_term,
  primary_term_variations = NULL,
  validation_function = NULL,
  verbose = TRUE,
  entity_col = "entity",
  type_col = "entity_type"
)
```

Arguments

entities	Data frame of entities to filter
primary_term	The primary term to trust
primary_term_variations	Vector of variations of the primary term to trust
validation_function	Function to validate entities (default: is_valid_biomedical_entity)
verbose	Logical; if TRUE, print status messages (default: TRUE)
entity_col	Name of the column containing entity names (default: "entity")
type_col	Name of the column containing entity types (default: "entity_type")

Value

A data frame of filtered entities

Examples

```
# Create example entities
entities <- data.frame(
  entity = c("migraine", "optimization", "receptor", "europe"),
  entity_type = c("disease", "process", "protein", "location")
)

# Validate entities
validated <- valid_entities(entities, "migraine", c("migrain", "headache"))
print(validated)
```

validate_abc	<i>Apply statistical validation to ABC model results with support for large matrices</i>
--------------	--

Description

This function performs statistical tests to validate ABC model results. It calculates p-values using hypergeometric tests and applies correction for multiple testing. The function is optimized to work with very large co-occurrence matrices.

Usage

```
validate_abc(
  abc_results,
  co_matrix,
  alpha = 0.05,
  correction = c("BH", "bonferroni", "none"),
  filter_by_significance = FALSE
)
```

Arguments

abc_results	A data frame containing ABC results.
co_matrix	The co-occurrence matrix used to generate the ABC results.
alpha	Significance level (p-value threshold).
correction	Method for multiple testing correction.
filter_by_significance	Logical. If TRUE, only returns significant results.

Value

A data frame with ABC results and statistical significance measures.

Examples

```
## Not run:  
validated_results <- validate_abc(abc_results, co_matrix)  
  
## End(Not run)
```

validate_biomedical_entity

Validate biomedical entities using BioBERT or other ML models

Description

Validate biomedical entities using BioBERT or other ML models

Usage

```
validate_biomedical_entity(term, claimed_type)
```

Arguments

term	Character string, the term to validate
claimed_type	Character string, the claimed entity type

Value

Logical indicating if the term is validated

validate_entity_comprehensive

Comprehensive entity validation using multiple techniques

Description

Comprehensive entity validation using multiple techniques

Usage

```
validate_entity_comprehensive(  
  term,  
  claimed_type,  
  use_nlp = TRUE,  
  use_pattern = TRUE,  
  use_external_api = FALSE  
)
```

Arguments

term	Character string, the term to validate
claimed_type	Character string, the claimed entity type
use_nlp	Logical, whether to use NLP-based validation
use_pattern	Logical, whether to use pattern-based validation
use_external_api	Logical, whether to query external APIs

Value

Logical indicating if the term is validated

validate_entity_with_nlp

Validate entity types using NLP-based entity recognition with improved accuracy

Description

Validate entity types using NLP-based entity recognition with improved accuracy

Usage

```
validate_entity_with_nlp(term, claimed_type, nlp_model = NULL)
```

Arguments

term	Character string, the term to validate
claimed_type	Character string, the claimed entity type
nlp_model	The loaded NLP model to use for validation

Value

Logical indicating if the term is likely of the claimed type

validate_umls_key	<i>Validate a UMLS API key</i>
-------------------	--------------------------------

Description

This function validates a UMLS API key using the validation endpoint.

Usage

```
validate_umls_key(api_key, validator_api_key = NULL)
```

Arguments

api_key	UMLS API key to validate
validator_api_key	Your application's UMLS API key (for third-party validation)

Value

Logical indicating if the API key is valid

Examples

```
## Not run:  
is_valid <- validate_umls_key("user_api_key")  
  
## End(Not run)
```

vec_preprocess	<i>Vectorized preprocessing of text</i>
----------------	---

Description

This function preprocesses text data using vectorized operations for better performance.

This function preprocesses text data using vectorized operations for better performance.

Usage

```
vec_preprocess(  
  text_data,  
  text_column = "abstract",  
  remove_stopwords = TRUE,  
  custom_stopwords = NULL,  
  min_word_length = 3,  
  max_word_length = 50,  
  chunk_size = 100
```

```
)  
  
vec_preprocess(  
  text_data,  
  text_column = "abstract",  
  remove_stopwords = TRUE,  
  custom_stopwords = NULL,  
  min_word_length = 3,  
  max_word_length = 50,  
  chunk_size = 100  
)
```

Arguments

`text_data` A data frame containing text data.

`text_column` Name of the column containing text to process.

`remove_stopwords`
 Logical. If TRUE, removes stopwords.

`custom_stopwords`
 Character vector of additional stopwords to remove.

`min_word_length`
 Minimum word length to keep.

`max_word_length`
 Maximum word length to keep.

`chunk_size` Number of documents to process in each chunk.

Value

A data frame with processed text.

A data frame with processed text.

Examples

```
## Not run:  
processed_data <- vec_preprocess(article_data, text_column = "abstract")  
  
## End(Not run)  
## Not run:  
processed_data <- vec_preprocess(article_data, text_column = "abstract")  
  
## End(Not run)
```

vis_abc_heatmap	<i>Create a heatmap of ABC connections</i>
-----------------	--

Description

This function creates a heatmap visualization of ABC connections using base R graphics.

Usage

```
vis_abc_heatmap(
  abc_results,
  top_n = 25,
  min_score = 0.1,
  show_labels = TRUE,
  title = "ABC Connections Heatmap"
)
```

Arguments

abc_results	A data frame containing ABC results from apply_abc_model().
top_n	Number of top results to visualize.
min_score	Minimum score threshold for including connections.
show_labels	Logical. If TRUE, shows labels on the tiles.
title	Plot title.

Value

NULL invisibly. The function creates a plot as a side effect.

Examples

```
## Not run:
vis_abc_heatmap(abc_results, top_n = 20)

## End(Not run)
```

vis_heatmap	<i>Create an enhanced heatmap of ABC connections</i>
-------------	--

Description

This function creates an improved heatmap visualization of ABC connections that can display entity type information when available, without enforcing type constraints.

Usage

```
vis_heatmap(  
  abc_results,  
  top_n = 25,  
  min_score = 0.1,  
  show_significance = TRUE,  
  color_palette = "blues",  
  title = "ABC Connections Heatmap",  
  show_entity_types = TRUE  
)
```

Arguments

abc_results	A data frame containing ABC results.
top_n	Number of top results to visualize.
min_score	Minimum score threshold for including connections.
show_significance	Logical. If TRUE, marks significant connections.
color_palette	Character. Color palette to use for the heatmap.
title	Plot title.
show_entity_types	Logical. If TRUE, includes entity types in axis labels.

Value

NULL invisibly. The function creates a plot as a side effect.

Examples

```
## Not run:  
vis_heatmap(abc_results, top_n = 20, show_significance = TRUE)  
  
## End(Not run)
```

vis_network

Create an enhanced network visualization of ABC connections

Description

This function creates an improved network visualization of ABC connections that displays entity types when available, without enforcing type constraints.

Usage

```
vis_network(  
  abc_results,  
  top_n = 25,  
  min_score = 0.1,  
  show_significance = TRUE,  
  node_size_factor = 5,  
  color_by = "type",  
  title = "ABC Model Network",  
  show_entity_types = TRUE,  
  label_size = 1  
)
```

Arguments

abc_results	A data frame containing ABC results.
top_n	Number of top results to visualize.
min_score	Minimum score threshold for including connections.
show_significance	Logical. If TRUE, highlights significant connections.
node_size_factor	Factor for scaling node sizes.
color_by	Column to use for node colors. Default is 'type'.
title	Plot title.
show_entity_types	Logical. If TRUE, includes entity types in node labels.
label_size	Relative size for labels. Default is 1.

Value

NULL invisibly. The function creates a plot as a side effect.

Examples

```
## Not run:  
vis_network(abc_results, top_n = 20, show_significance = TRUE,  
            show_entity_types = TRUE)  
  
## End(Not run)
```

visualize_abc_network *Visualize ABC model results as a network*

Description

Create a network visualization of ABC connections using base R graphics.

Usage

```
vis_abc_network(  
  abc_results,  
  top_n = 25,  
  min_score = 0.1,  
  node_size_factor = 3,  
  edge_width_factor = 1,  
  color_by = "type",  
  title = "ABC Model Network"  
)
```

Arguments

<code>abc_results</code>	A data frame containing ABC results from <code>apply_abc_model()</code> .
<code>top_n</code>	Number of top results to visualize.
<code>min_score</code>	Minimum score threshold for including connections.
<code>node_size_factor</code>	Factor for scaling node sizes.
<code>edge_width_factor</code>	Factor for scaling edge widths.
<code>color_by</code>	Column to use for node colors. Default is 'type'.
<code>title</code>	Plot title.

Value

NULL invisibly. The function creates a plot as a side effect.

Examples

```
## Not run:  
# Create a network visualization of ABC model results  
vis_abc_network(abc_results, top_n = 20)  
  
## End(Not run)
```

Index

abc_model, 3
abc_model_opt, 5
abc_model_sig, 6
abc_timeslice, 7
anc_model, 8

bitola_model, 9

calc_bibliometrics, 10
calc_doc_sim, 10
clear_pubmed_cache, 11
cluster_docs, 12
compare_terms, 13
create_citation_net, 14
create_comat, 14
create_report, 15
create_sparse_comat, 16
create_tdm, 17
create_term_document_matrix, 18

detect_lang, 18
diversify_abc, 19

enhance_abc_kb, 20
eval_evidence, 20
export_chord, 21
export_chord_diagram, 22
export_network, 23
extract_entities, 24
extract_entities_workflow, 25
extract_ner, 27
extract_ngrams, 27
extract_terms, 28
extract_topics, 29

filter_by_type, 30
find_abc_all, 30
find_similar_docs, 31
find_term, 32
gen_report, 32

get_dict_cache, 33
get_pmc_fulltext, 34
get_term_vars, 35
get_type_dist, 35

is_valid_biomedical_entity, 36

load_dictionary, 37
load_results, 38
lsi_model, 39

map_ontology, 40
merge_entities, 41
merge_results, 42
min_results, 43

ncbi_search, 44

parallel_analysis, 45
perm_test_abc, 46
plot_heatmap, 46
plot_network, 48
prep_articles, 49
preprocess_text, 50
pubmed_search, 51

query_external_api, 52
query_mesh, 52
query_umls, 53

run_lbd, 53

safe_diversify, 54
sanitize_dictionary, 56
save_results, 57
segment_sentences, 57

valid_entities, 58
validate_abc, 59
validate_biomedical_entity, 60
validate_entity_comprehensive, 60

validate_entity_with_nlp, [61](#)
validate_umls_key, [62](#)
vec_preprocess, [62](#)
vis_abc_heatmap, [64](#)
vis_abc_network
 (visualize_abc_network), [67](#)
vis_heatmap, [64](#)
vis_network, [65](#)
visualize_abc_network, [67](#)