Package ‘genular’

October 19, 2024

Version 1.0.1

Date 2024-10-19

Title 'Genular' Database API

Author Ivan Tomic [aut, cre, cph] (<https://orcid.org/0000-0003-3596-681X>),
Adriana Tomic [aut, ctb, cph, fnd]
(<https://orcid.org/0000-0001-9885-3535>),
Stephanie Hao [aut] (<https://orcid.org/0000-0002-3760-8234>)

Description
Provides an interface to the 'Genular' database API (<https://genular.atomic-lab.org>), allowing efficient retrieval and integration of genomic, proteomic, and single-cell data. It supports operations like fetching gene annotations, cell expression profiles, and other information as defined in the 'Genular' database, enabling seamless incorporation of biological data into R workflows. With functions tailored for bioinformatics and machine learning, the package facilitates exploration of cellular heterogeneity, gene-disease relationships, and pathway analysis, streamlining multi-omics data analysis.

Maintainer Ivan Tomic <`info@ivantomic.com`>

Imports httr, jsonlite, dplyr, tidyr, stats

Suggests plyr, purrr, ggplot2

Depends R (>= 3.4.0)

URL https://github.com/atomiclaboratory/genular-database,
<https://genular.atomic-lab.org>

BugReports https://github.com/atomiclaboratory/genular-database/issues

License GPL-3

Encoding UTF-8

LazyLoad yes

RoxygenNote 7.3.1

NeedsCompilation no

Repository CRAN

Date/Publication 2024-10-19 02:20:02 UTC

Contents

cells_search 2
cells_search_format 3
cells_suggest 4
cells_to_gene_signature 5
convert_gene_expression_to_pathway_features 6
extract_data 7
fetch_all_gene_search_results 8
gene_search 10
pathways_suggest 12
pathway_to_cell_signature 13
summerize_by_category 14

Index **17**

`cells_search` *Search Cell Information Based on Query Conditions*

Description

This function interacts with the 'Genular' API to search for cell information based on specific query conditions related to cell IDs and expression marker scores. It sends a POST request with these conditions and retrieves matching cell information.

Usage

```
cells_search(
  queryValues,
  fieldsFilter = c("geneID", "symbol", "crossReference.enseGeneID"),
  excludeFields = NULL,
  page = 1,
  limit = 10,
  searchType = "and",
  orderBy = "geneID",
  sortDirection = "asc",
  responseType = "json",
  matchType = "exact",
  organismType = list(c("9606")),
  debug = 0,
  options = list()
)
```

Arguments

`queryValues`	A named list where keys are cell IDs and values are condition strings indicating the expression score criteria.
`fieldsFilter`	A character vector specifying which fields to include in the response.

`excludeFields`	If fieldsFilter is not provided (empty), all fields are returned, here you can specify the fields you want to exclude.
`page`	An integer specifying the page number for pagination of results.
`limit`	An integer specifying the maximum number of results to return per page.
`searchType`	A character string indicating whether to use 'and' or 'or' logic for multiple search conditions.
`orderBy`	The field name by which to order the returned results.
`sortDirection`	The direction of sorting, which can be either "asc" for ascending or "desc" for descending.
`responseType`	A character string indicating the type of response to expect ('json' or 'csv').
`matchType`	A character string indicating the type of match to perform ('exact' or 'regex').
`organismType`	A list of organism type IDs to filter the search results.
`debug`	An integer value indicating whether to enable debug mode (1) or not (0).
`options`	A list containing additional options for the API request, including the endpoint, api_key, timeout, and user-agent.

Value

A list containing detailed information about the cells that meet the search criteria, including the requested fields.

Examples

```
queryValues <- list("CL0001082" = ">= 250")
fieldsFilter <- c("geneID", "symbol")
cell_search_results <- cells_search(queryValues, fieldsFilter)
print(cell_search_results)
```

`cells_search_format` *Format Cell Search Results for Query Based on Lineage*

Description

This function formats the results from cell search queries into a structured list based on specified lineage criteria. It allows subsetting of the cell data to include parent cells, child cells, or both in the output, and converts them into a named list where each key is a `cell_id` and each value is a string representing a condition on the `marker_score`.

Usage

```
cells_search_format(cell_list, cells_lineages = "both")
```

Arguments

`cell_list`	A list of cells, each including details such as `cell_id` and `marker_score`, and optionally containing nested lists of child cells.
`cells_lineages`	A character string specifying the lineage subset to include in the output. Options are "parent" for only parent cells, "childs" for only child cells, and "both" for including both parent and child cells. Defaults to "both".

Value

A named list where keys are `cell_ids` and values are strings formatted as conditions on the `marker_score`. This list can be used for constructing query conditions in further API requests.

Examples

```
cells <- list(
  list(cell_id = "CL0000235", cell_name = "macrophage",
  marker_score = 1112.325, childs = list()),
  list(cell_id = "CL0000784", cell_name = "plasmacytoid dendritic cell",
  marker_score = 537.7737, childs = list(
    list(cell_id = "CL0001058", cell_name = "plasmacytoid dendritic cell, human",
  marker_score = 262.985)
  ))
)
formatted_query_values <- cells_search_format(cells, cells_lineages = "both")
print(formatted_query_values)
```

`cells_suggest` *Suggest Cell Matches Based on Query Values*

Description

This function communicates with the 'Genular' API to suggest cell matches based on an array of query values. It sends a POST request with the query values and retrieves suggested cell matches, including details and scores.

Usage

```
cells_suggest(queryValues, responseType = "json", debug = 0, options = list())
```

Arguments

`queryValues`	A character vector of cell names or identifiers to find matches for.
`responseType`	A character string indicating the type of response to expect ('json' or 'csv').
`debug`	An integer value indicating whether to enable debug mode (1) or not (0).
`options`	A list that specifies the API endpoint, api_key, timeout duration, and user agent string, with default values preset for the 'Genular' API cell suggestion endpoint.

Value

A list containing suggested cell matches, each with associated details like keys, values, search scores, and expression marker scores.

Examples

```
queryValues <- c("endothelial cell", "T cell")
cell_suggest_results <- cells_suggest(queryValues)
print(cell_suggest_results)
```

`cells_to_gene_signature`

Cells to Gene Signature

Description

Queries for cells based on given values, fetches genes associated with those cells, and filters for unique gene identifiers based on specified criteria such as fold change. This function streamlines the process of identifying significant gene signatures from cell query results. The function further filters results based on a threshold determined by the mean and standard deviation of foldChange values.

Usage

```
cells_to_gene_signature(
  queryValues,
  data,
  dataMeta,
  uniqueRowID = "",
  options = list(timeout = 10000)
)
```

Arguments

`queryValues`	A vector of query values to search for cells.
`data`	A data frame containing gene expression data to be mapped.
`dataMeta`	A data frame containing metadata for each entry in `data`.
`uniqueRowID`	The name of the column in `dataMeta` that uniquely identifies each row.
`options`	A list of options for the API call, including endpoint and timeout settings.

Value

A list containing `data`, the final data frame with gene signatures, and `mapping`, a data frame of gene to cell mappings.

Examples

```
## Not run:
cells_to_gene_signature(
  queryValues = c("CL0001054", "CL0002397"),
  data = yourData,
  dataMeta = yourDataMeta,
  uniqueRowID = "yourUniqueIdentifierColumnName"
)

## End(Not run)
```

---

convert_gene_expression_to_pathway_features

Convert Gene Expression Data to Pathway-Level Features

Description

Transforms a gene expression matrix into pathway-level features per sample suitable for machine learning applications. This function maps genes to their corresponding biological pathways, removes redundant pathways, calculates a `PathwayGeneScore` based on median gene expression and pathway variance, and optionally includes pathways not shared across multiple genes. Unmapped genes are retained as individual features in the final dataset.

Usage

```
convert_gene_expression_to_pathway_features(
  input_data,
  data_transposed,
  keep_non_shared = TRUE
)
```

Arguments

`input_data`	A dataframe containing gene expression data, where rows represent samples and columns represent genes. Each cell contains the expression level of a gene in a specific sample.
`data_transposed`	A dataframe containing gene-to-pathway mappings, with at least two columns: `mappedSymbol` (gene symbols) and `mappedId` (unique pathway identifiers).
`keep_non_shared`	A logical flag indicating whether to include pathways mapped to a single gene. Defaults to `TRUE`. If set to `FALSE`, pathways mapped to fewer than two genes will be excluded from the final dataset.

Value

A dataframe where each row corresponds to a sample, and each column represents either a pathway-level feature (`PathwayGeneScore`) or an unmapped gene's expression. Pathway features encapsulate the median expression of genes within the pathway, adjusted by gene count and pathway variance. Unmapped genes are included as individual features to retain comprehensive gene expression information.

Examples

```
# Sample gene expression data
input_data <- data.frame(
  A1CF = c(2, 3, 3, 3),
  A2M = c(3, 4, 3, 3),
  A4GALT = c(3, 4, 3, 4),
  A4GNT = c(3, 4, 3, 3),
  ABC1 = c(2, 2, 2, 2),
  ABC2 = c(4, 4, 4, 4)
)

# Sample gene-pathway mapping data
data_transposed <- data.frame(
  mappedSymbol = c("A4GNT", "A4GALT", "A2M", "A4GALT", "A2M", "A2M", "ABC1", "ABC2"),
  mappedId = c("GO:0000139", "GO:0000139", "GO:0001553", "GO:0001576",
               "GO:0001869", "GO:0002020", "GO:0000139", "GO:0000139")
)

# Convert gene expression data to pathway-level features, including non-shared pathways
final_data <- convert_gene_expression_to_pathway_features(input_data, data_transposed,
                                                          keep_non_shared = TRUE)
print(final_data)
```

`extract_data` *Extract Data Based on Mappings*

Description

This function iterates over a list of gene results, extracting and transforming data according to a provided mapping schema. It handles both direct mappings and nested array mappings, creating a comprehensive data frame with extracted data.

Usage

```
extract_data(
  all_gene_results,
  mappings = list(geneID = "mappedGeneID", symbol = "mappedSymbol",
   `crossReference$enseGeneID` = "mappedEnseGeneID", `mRNAExpressions$proteinAtlas` =
   list(c(c = "mappedC")), ontology = list(c(id = "mappedId", term = "mappedTerm", cat =
     "mappedCat")))
)
```

Arguments

`all_gene_results`	A list of gene results, where each element is a list containing gene information that might include nested structures.
`mappings`	A list defining the mapping from input data structure to output data frame columns. It supports direct mappings as well as mappings for nested structures. The default mappings are provided. Each mapping should be a character vector for direct mappings or a list of vectors for nested mappings.

Value

A data frame where each row corresponds to an entry in the input list, and each column corresponds to one of the specified mappings. For nested array mappings, multiple rows will be generated based on array entries, duplicating other information as needed.

Examples

```
# Assuming all_gene_results is your input data

all_gene_results <- fetch_all_gene_search_results(
  queryFields = list(c("symbol")),
  queryValues = c("A1CF", "A2M", "A4GALT", "A4GNT"),
  fieldsFilter = c("geneID", "symbol", "crossReference.enseGeneID",
                   "mRNAExpressions.proteinAtlas.c", "ontology.id",
                   "ontology.term", "ontology.cat"),
  searchType = "or",
  orderBy = "geneID",
  sortDirection = "asc",
  responseType = "json",
  matchType = "exact",
  organismType = list(c(9606)),
  ontologyCategories = list(),
  limit = 100,
  options = list(api_key = "3147dda5fa023c9763d38bddb472dd28", timeout = 10000)
)

data_transposed <- extract_data(all_gene_results, list(
    "geneID" = "mappedGeneID",
    "symbol" = "mappedSymbol",
    "crossReference$enseGeneID" = "mappedEnseGeneID",
    "mRNAExpressions$proteinAtlas" = list(c("c" = "mappedC")),
    "ontology" = list(c("id" = "mappedId", "term" = "mappedTerm", "cat" = "mappedCat"))
))
```

`fetch_all_gene_search_results`

Fetch All Pages of Gene Search Results

Description

This function iteratively calls the `gene_search` function to retrieve all available search results across pages for a given query.

Usage

```
fetch_all_gene_search_results(
  queryFields,
  queryValues,
  fieldsFilter = c("geneID", "symbol", "crossReference.enseGeneID", "ontology.id",
    "ontology.term", "ontology.cat"),
  searchType = "or",
  orderBy = "geneID",
  sortDirection = "asc",
  responseType = "json",
  matchType = "exact",
  organismType = list(c(9606)),
  ontologyCategories = list(),
  limit = 5,
  debug = 0,
  options = list()
)
```

Arguments

`queryFields`	A character vector specifying the fields to search within the gene data.
`queryValues`	A numeric/character vector representing the values to search for within the specified fields.
`fieldsFilter`	A vector specifying which fields to include in the response.
`searchType`	Indicates whether to use 'and' or 'or' logic for multiple search conditions.
`orderBy`	Specifies which field to sort the results by.
`sortDirection`	Indicates the sort direction ('asc' or 'desc').
`responseType`	Indicates the type of response to expect ('json' or 'csv').
`matchType`	Indicates the type of match to perform ('exact' or 'regex').
`organismType`	A list of organism type IDs to filter the search results.
`ontologyCategories`	A list of ontology category IDs to filter the search results.
`limit`	The maximum number of results to return per page.
`debug`	An integer value indicating whether to enable debug mode (1) or not (0).
`options`	A list of additional options for the API request, including endpoint, api_key, timeout, and user-agent.

Value

A list of gene search results aggregated from all retrieved pages.

Examples

```
all_gene_results <- fetch_all_gene_search_results(
  queryFields = list(c("symbol")),
  queryValues = c("A1CF", "A2M", "A4GALT", "A4GNT"),
  fieldsFilter = c("geneID", "symbol", "crossReference.enseGeneID",
                   "ontology.id", "ontology.term", "ontology.cat"),
  searchType = "or",
  orderBy = "geneID",
  sortDirection = "asc",
  responseType = "json",
  matchType = "exact",
  organismType = list(c(9606)),
  ontologyCategories = list(),
  limit = 5
)
```

`gene_search` *Search for Gene Information Based on a Query*

Description

This function allows users to search for gene information by sending a POST request to the 'Genular' API. It accepts various search parameters and returns information about genes that match the search criteria.

Usage

```
gene_search(
  queryFields,
  queryValues,
  fieldsFilter = c("geneID", "symbol", "crossReference.enseGeneID"),
  excludeFields = NULL,
  page = 1,
  limit = 10,
  searchType = "and",
  orderBy = "geneID",
  sortDirection = "asc",
  responseType = "json",
  matchType = "exact",
  organismType = list(c("9606")),
  ontologyCategories = list(),
  debug = 0,
  options = list()
)
```

Arguments

`queryFields`	A character vector specifying the fields to search within the gene data.
`queryValues`	A numeric vector representing the values to search for within the specified fields.
`fieldsFilter`	An optional character vector specifying which fields to include in the response.
`excludeFields`	If fieldsFilter is not provided (empty), all fields are returned, here you can specify the fields you want to exclude.
`page`	An integer specifying the page number of the search results to retrieve.
`limit`	An integer specifying the maximum number of results to return per page.
`searchType`	A character string indicating whether to use ’and’ or ’or’ logic for multiple search conditions.
`orderBy`	A character string specifying which field to sort the results by.
`sortDirection`	A character string indicating the sort direction (’asc’ or ’desc’).
`responseType`	A character string indicating the type of response to expect (’json’ or ’csv’).
`matchType`	A character string indicating the type of match to perform (’exact’ or ’regex’).
`organismType`	A list of organism type IDs to filter the search results.
`ontologyCategories`	A list of ontology category IDs to filter the search results.
`debug`	An integer value indicating whether to enable debug mode (1) or not (0).
`options`	A list of additional options for the API request, including endpoint, api_key, timeout, and user-agent.

Value

Depending on the responseType parameter, this function returns a list with different elements: If responseType is ’json’, the function returns a list containing the HTTP status code (’status_code’), the parsed JSON content (’content’) representing gene information matching the search criteria, and the original request body sent to the API (’request_body’). If responseType is ’csv’, the function returns a list containing the HTTP status code (’status_code’), a data frame (’content’) constructed from the CSV response representing gene information, and the original request body sent to the API (’request_body’). In case of an HTTP status code different from 200, the content part of the return value provides the received error message or data.

Examples

```
# Define search parameters
queryFields <- list(c("geneID")) # or c("geneID", "symbol" ...)
queryValues <- c(1, 56, 70)
searchType <- "or"
fieldsFilter <- c("geneID", "symbol", "crossReference.enseGeneID")

# Execute the search
gene_search_results <- gene_search(queryFields, queryValues,
                                     fieldsFilter, searchType = searchType,
                                     page = 1, limit = 10)
```

```
# Print the results
print(gene_search_results)
```

`pathways_suggest` *Suggest Pathway Matches Based on Query Values*

Description

This function queries the Genular API to suggest pathway matches based on an array of query values. It is useful for identifying pathways related to specific terms or concepts provided in the query.

Usage

```
pathways_suggest(queryValues, options = list())
```

Arguments

`queryValues`	A character vector representing the search terms or values to find corresponding pathways.
`options`	A list of options to customize the API request, including the API endpoint URL, api_key, timeout duration, and user-agent string, with sensible defaults set for querying the Genular pathways suggestion endpoint.

Value

A list containing suggested pathway matches including their identifiers and other relevant details based on the provided query values.

Examples

```
queryValues <- c("apoptosis", "signal transduction")
pathway_suggest_results <- pathways_suggest(queryValues)
print(pathway_suggest_results)
```

`pathway_to_cell_signature` *Pathway to Cell Signature*

Description

This function queries for pathways based on given values, fetches genes associated with those pathways, and filters for unique cell identifiers based on specified criteria. If pathway IDs are already known, they can be directly provided to skip the query step.

Usage

```
pathway_to_cell_signature(
  queryValues = NULL,
  pathway_ids = NULL,
  options = list(timeout = 10000)
)
```

Arguments

`queryValues`	A vector of query values to search for pathways. Optional if `pathway_ids` is provided.
`pathway_ids`	A vector of pathway IDs to be used directly. Optional if `queryValues` is provided.
`options`	A list of options for the API call, including endpoint and timeout settings.

Value

A data frame of unique effect sizes, filtered by a foldChange threshold and deduplicated by cell_id.

Examples

```
## Not run:
pathway_to_cell_signature(
  queryValues = c("Adaptive Immune System"),
  options = list(timeout = 10000)
)

## End(Not run)
```

`summerize_by_category` *Summarize Data by Category*

Description

This function summarizes input data by categories defined in the mapping data. It supports summary methods such as median and mean, and allows additional options like retaining missing categories or appending category IDs to names.

Usage

```
summerize_by_category(
  input_data,
  mapping_data,
  identifier = "symbol",
  keep_missing = FALSE,
  keep_ids = FALSE,
  summary_method = "median"
)
```

Arguments

`input_data`	A data frame where each column represents a gene or an identifier, and each row represents an observation or a sample.
`mapping_data`	A data frame that maps identifiers to categories, which must include the columns specified by `identifier` and 'category'. Optionally, it can contain 'category_id' for additional categorization details.
`identifier`	The name of the column in `mapping_data` that corresponds to the identifiers in the columns of `input_data`.
`keep_missing`	A logical value indicating whether to retain identifiers in `input_data` that are not found in `mapping_data`. If TRUE, they are kept as separate categories.
`keep_ids`	A logical value indicating whether to append category IDs to the category names in the summary output.
`summary_method`	The method used for summarizing within categories. Currently supports "median" and "mean".

Value

A data frame where each column represents a category and each row represents the summarized value of that category for the corresponding observation/sample.

Examples

```
# Create a sample input data frame with gene expression levels
input_data <- data.frame(
    A1CF = c(2, 3, 3, 3),
```

```
    A2M = c(3, 4, 3, 3),
    A4GALT = c(3, 4, 3, 4),
    A4GNT = c(3, 4, 3, 3)
)

# Fetch gene-related data based on specified fields and conditions
# The function `fetch_all_gene_search_results` is presumably defined elsewhere
# and retrieves information from a biological database
all_gene_results <- fetch_all_gene_search_results(
    queryFields = list(c("symbol")),                # Query by gene symbols
    queryValues = colnames(input_data),              # Gene symbols to query
    fieldsFilter = c(                               # Fields to extract from the results
        "geneID",
        "symbol",
        "crossReference.enseGeneID",
        "mRNAExpressions.proteinAtlas.c",
        "ontology.id",
        "ontology.term",
        "ontology.cat"
    ),
    searchType = "or",                           # Search type (OR condition for queries)
    orderBy = "geneID",                              # Ordering criteria
    sortDirection = "asc",                           # Sort direction (ascending)
    responseType = "json",                           # Format of the returned data
    matchType = "exact",                             # Type of match for the query
    organismType = list(c(9606)),                   # Organism type (e.g., Homo sapiens)
    ontologyCategories = list(),                     # Ontology categories to include
    limit = 100,                                     # Limit on the number of results
    options = list(api_key = "your_api_key", timeout = 10000)  # Additional options
)

# Transform the fetched gene data based on specified mappings
data_transposed <- extract_data(
    all_gene_results,
    list(
        "geneID" = "mappedGeneID",
        "symbol" = "mappedSymbol",
        "crossReference$enseGeneID" = "mappedEnseGeneID",
        "mRNAExpressions$proteinAtlas" = list(c("c" = "mappedC")),
        "ontology" = list(c(
            "id" = "mappedId",
            "term" = "mappedTerm",
            "cat" = "mappedCat"
        ))
    )
)

# Manually create a similar structure to the expected output of `extract_data`
# This mimics the processed and transposed gene data
data_transposed <- data.frame(
    mappedGeneID = c(2, 2, 2, 2, 2, 2),
    mappedSymbol = rep("A2M", 6),
    mappedEnseGeneID = rep("ENSG00000175899", 6),
```

```
    mappedC = c("gdT-cell", NA, NA, NA, NA, NA),
    mappedId = c(
        NA,
        "R-HSA-109582",
        "R-HSA-1474244",
        "R-HSA-382551",
        "R-HSA-140877",
        "R-HSA-1474228"
    ),
    mappedTerm = c(
        NA,
        "Hemostasis",
        "Extracellular matrix organization",
        "Transport of small molecules",
        "Formation of Fibrin Clot (Clotting Cascade)",
        "Degradation of the extracellular matrix"
    ),
    mappedCat = c(NA, 10, 10, 10, 11, 11),
    stringsAsFactors = FALSE
)

library(dplyr)
# Process and group the data by symbol, then summarize and arrange by terms
data_transposed_pathways <- data_transposed %>%
    dplyr::group_by(mappedSymbol) %>%
    dplyr::arrange(mappedTerm, .by_group = TRUE) %>%
    dplyr::summarize(
        category = first(mappedTerm),
        category_id = first(mappedId)
    )

# Display the first few rows of the grouped data
# print(head(data_transposed_pathways))

# Summarize the original input data by the categories defined in the processed gene data
# This function call summarizes expression levels by the gene's associated pathway or term
result_data_pathways <- summerize_by_category(
    input_data,
    data_transposed_pathways,
    identifier = "mappedSymbol",
    keep_missing = FALSE,
    keep_ids = FALSE,
    summary_method = "median"
)
```

Index

cells_search, 2
cells_search_format, 3
cells_suggest, 4
cells_to_gene_signature, 5
convert_gene_expression_to_pathway_features, 6

extract_data, 7

fetch_all_gene_search_results, 8

gene_search, 10

pathway_to_cell_signature, 13
pathways_suggest, 12

summerize_by_category, 14