Package 'snapcount'

April 10, 2025

Type Package

Title R/Bioconductor Package for interfacing with Snaptron for rapid querying of expression counts

Version 1.19.0

Description snapcount is a client interface to the Snaptron webservices which support querying by gene name or genomic region. Results include raw expression counts derived from alignment of RNA-seq samples and/or various summarized measures of expression across one or more regions/genes per-sample (e.g. percent spliced in).

Depends R (>= 4.0.0)

Imports R6, httr, rlang, purrr, jsonlite, assertthat, data.table, Matrix, magrittr, methods, stringr, stats, IRanges, GenomicRanges, SummarizedExperiment

Suggests BiocManager, bit64, covr, knitcitations, knitr (>= 1.6), devtools, BiocStyle (>= 2.5.19), rmarkdown (>= 0.9.5), testthat (>= 2.1.0)

License MIT + file LICENSE

Encoding UTF-8

LazyData False

RoxygenNote 7.1.1

Roxygen list(markdown = TRUE, roclets = c(``rd", ``namespace", ``collate"))

URL https://github.com/langmead-lab/snapcount

BugReports https://github.com/langmead-lab/snapcount/issues

biocViews Coverage, GeneExpression, RNASeq, Sequencing, Software, DataImport

VignetteBuilder knitr

ByteCompile true

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

git_branch devel

git_last_commit 9751037

git_last_commit_date 2024-10-29

Repository Bioconductor 3.21

Date/Publication 2025-04-09

Author Rone Charles [aut, cre]

Maintainer Rone Charles <rcharle8@jh.edu>

Contents

`snapcount-package` *snapcount: an R package for interfacing with Snaptron*

Description

snapcount is a client interface to the Snaptron webservice which supports querying by gene name or genomic region.

Details

Results include raw expression counts derived from alignment of RNA-seq samples and/or various summarized measures of expression across one or more regions/genes per-sample (e.g. percent spliced in).

To learn more about snapcount, check out the vignette: `browseVignettes(package = "snapcount")`

Package options

`snapcount.host` Change the host that snapcount uses when connecting to Snaptron. Default: `snaptron.cs.jhu.edu`

`snapcount.port` Change the port that snapcount uses when connecting to Snaptron. Default: `80`

Author(s)

Maintainer: Rone Charles <rcharle8@jh.edu>

See Also

Useful links:

- https://github.com/langmead-lab/snapcount
- Report bugs at https://github.com/langmead-lab/snapcount/issues

---

`Compilation` *Enum for Snaptron compilations*

Description

The variants for this enum will be populated dynamically after the package has been loaded. If the package cannot connect to the internet the variants will default to:

Usage

```
Compilation
```

Format

An object of class `environment` of length 21.

Details

- gtex
- tcga
- srav2
- sra

See Also

http://snaptron.cs.jhu.edu/data.html for more information about Snaptron compilations.

Examples

```
qb <- QueryBuilder(compilation = Compilation$gtex, regions = "KCNIP4")
query_jx(qb)
```

Coordinates *Enum for Snaptron Coordinate modifiers*

Description

Enum for Snaptron Coordinate modifiers

Usage

```
Coordinates
```

Format

An object of class `environment` of length 4.

Fields

`Exact` Return junctions whose start and end coordinates match the boundaries of the region requested.

`Within` Return junctions whose start and end coordinates are within the boundaries of the region requested.

`StartIsExactOrWithin` Return junctions whose start coordinate matches, or is within, the boundaries of the region requested.

`EndIsExactOrWithin` Return junctions whose end coordinate matches, or is within, the boundaries of the region requested.

Examples

```
qb <- QueryBuilder(compilation = "gtex", regions = "CD99")
qb <- set_coordinate_modifier(qb, Coordinates$Exact)
qb
```

from_url *Constructs a* `QueryBuilder` *object from the given url*

Description

Constructs a `QueryBuilder` object from the given url

Usage

```
from_url(url)
```

Arguments

url	a well-formed url preferably obtained from a call to the uri_of_last_successful_request function

Value

Returns a QueryBuilder object with attributes set from the parsed url.

Examples

```
sb <- from_url("http://snaptron.cs.jhu.edu/gtex/snaptron?regions=CD99")
get_regions(sb)
get_compilation(sb)
```

---

get_column_filters *Get or set sample-related contraints for query*

Description

Get or set sample-related contraints for query

Usage

```
get_column_filters(qb)

set_column_filters(qb, ...)
```

Arguments

qb	a QueryBuilder object constructed using the QueryBuilder function
...	one or more boolean predicates as either strings or unevaluated expressions

Value

get_column_filters returns the current filters as a list of strings. set_column_filters returns a new QueryBuilder object with the column filters set to the value of column_filters.

Examples

```
qb <- QueryBuilder(compilation = "gtex", regions = "CD99")
# column filters set using a string
qb <- set_column_filters(qb, "SMTS == Brain")
get_column_filters(qb)
# column filters set using unevaluated expression
qb <- set_column_filters(qb, SMTS == "Spleen")
get_column_filters(qb)
```

get_compilation *Get or set query compilation*

Description

Get or set query compilation

Usage

```
get_compilation(qb)

set_compilation(qb, compilation)
```

Arguments

qb	A QueryBuilder object constructed using the QueryBuilder function.
compilation	A single string containing the name of the Snaptron data source. Any variant of the Compilation enum can also be passed as an argument.

Value

get_compilation returns the current compilation as string. set_compilation returns a new QueryBuilder object with the compilation set to the value of compilation.

Examples

```
qb <- QueryBuilder(compilation = "gtex", regions = "CD99")
get_compilation(qb)
qb <- set_compilation(qb, Compilation$tcga)
get_compilation(qb)
```

get_coordinate_modifier *Get or set coordinate modifiers for the query.*

Description

Get or set coordinate modifiers for the query.

Usage

```
get_coordinate_modifier(qb)

set_coordinate_modifier(qb, coordinate_modifier)
```

Arguments

qb	a QueryBuilder object constructed using the QueryBuilder function.
coordinate_modifier	any of the variants of the Coordinates enum.

Value

get_coordinate_modifier returns the current coodinate modifier as a string. set_coordinate_modifier returns a new QueryBuilder object with the coordinate modifier set to the value of coordinate_modifier.

Examples

```
qb <- QueryBuilder(compilation = "gtex", regions = "CD99")
qb <- set_coordinate_modifier(qb, Coordinates$Within)
get_coordinate_modifier(qb)
```

---

get_regions *Get or set query regions*

Description

Get or set query regions

Usage

```
get_regions(qb)

set_regions(qb, regions)
```

Arguments

qb	A QueryBuilder object constructed using the QueryBuilder function.
regions	Either a list of 1 more HUGO gene names as strings e.g. "BRCA1" or a Granges class object containing one or more geneomic intervals (e.g. "chr1:1-1000").

Value

get_regions returns the current regions as a list of strings. set_regions returns a new QueryBuilder object with the regions set to the value of regions.

Examples

```
qb <- QueryBuilder(compilation = "gtex", regions = "CD99")
get_regions(qb)
qb <- set_regions(qb, "chr1:1-1000")
get_regions(qb)
qb <- set_regions(qb, GenomicRanges::GRanges("chr1", "1-1000"))
get_regions(qb)
```

get_row_filters *Get or set range-related contraints for query*

Description

Get or set range-related contraints for query

Usage

```
get_row_filters(qb)

set_row_filters(qb, ...)
```

Arguments

qb	a QueryBuilder object constructed using the QueryBuilder function.
...	one or more boolean predicates as either strings or unevaluated expressions.

Value

`get_row_filters` returns the current row filters as list of strings. `set_row_filters` returns a new `QueryBuilder` object with the row filters set to the value of `row_filters`.

Examples

```
qb <- QueryBuilder(compilation = "gtex", regions = "CD99")
# row filters set as a string
qb <- set_row_filters(qb, "strand == +")
get_row_filters(qb)
# row filters set using unevaluated expression
qb <- set_row_filters(qb, strand == "+")
get_row_filters(qb)
```

get_sids *Get or set query sample ids*

Description

Get or set query sample ids

Usage

```
get_sids(qb)

set_sids(qb, sids)
```

Arguments

qb	a QueryBuilder object constructed using the QueryBuilder function.
sids	a vector or 1 or more whole numbers to filter results on.

Value

get_sids returns the current sample ids as a vector of integers. set_sids returns a new QueryBuilder object with the sample ids set to the value of sids.

Examples

```
qb <- QueryBuilder(compilation = "gtex", regions = "CD99")
qb <- set_sids(qb, c(1, 2, 3))
get_sids(qb)
```

---

junction_inclusion_ratio

Relative measure of splice variant usage similar to PSI that allows for 2 arbitrarily defined groups of junctions (not limited to cassette exons).

Description

Calculates a coverage summary statistic per sample of the normalized coverage difference between two sets of separate junctions defined by at least two basic queries and organized into two groups.

Usage

```
junction_inclusion_ratio(group1, group2, group_names = NULL)
```

Arguments

group1, group2	Each group is a list of 1 or more QueryBuilder objects
group_names	Optional vector of strings representing the group names

Details

The summary statistic is as follows: If the coverage of the first group is "A" and the second is "B":

JIR(A,B)=(A - B) / (A+B+1)

This is calculated for every sample that occurs in one or the other (or both) groups results.

Value

A DataFrame of samples, with their JIR score and metadata, which had > 0 coverage in at least one resulting row in at least one of the groups

Examples

```
sb1 <- QueryBuilder(compilation = "srav2", regions = "chr2:29446395-30142858")
sb1 <- set_coordinate_modifier(sb1, Coordinates$Within)
sb1 <- set_row_filters(sb1, strand == "-")

sb2 <- QueryBuilder(compilation = "srav2", regions = "chr2:29416789-29446394")
sb2 <- set_coordinate_modifier(sb2, Coordinates$Within)
sb2 <- set_row_filters(sb2, strand == "-")

junction_inclusion_ratio(list(sb1), list(sb2))
```

---

junction_intersection *Get the intersection of junctions from 2 or more compilations which are on the same reference*

Description

This function operates similar to the `junction_union()` function, i.e it is cross compilation and merging of the same junction from multiple compilations will be handled exactly the same way. But instead of every junction which appears in at least one compilation, only the junctions which appear in *every* compilation will be returned.

Usage

```
junction_intersection(...)
```

Arguments

`...`	One or more QueryBuilder objects

Value

A RangedSummarizedExperiment of junctions common across compilations

See Also

`junction_union()`

Examples

```
# Using query builder wrappers
sb1 <- QueryBuilder(compilation = "gtex", regions = "chr1:1879786-1879786")
sb1 <- set_coordinate_modifier(sb1, Coordinates$EndIsExactOrWithin)
sb1 <- set_row_filters(sb1, strand == "-")

sb2 <- QueryBuilder(compilation = "tcga", regions = "chr1:1879786-1879786")
sb2 <- set_coordinate_modifier(sb2, Coordinates$EndIsExactOrWithin)
sb2 <- set_row_filters(sb2, strand == "-")

junction_intersection(sb1, sb2)
```

`junction_union` *Get the union of junctions from 2 or more compilations which are on the same reference*

Description

This function queries 2 or more compilations which are on the same reference version (e.g. hg38) and merges the resulting junctions into a single output table, unioning the sample coverage columns and the snaptron_id (jx ID) columns (the latter delimiter will be ":"). All sample IDs will be disjoint between compilations.

Usage

```
junction_union(...)
```

Arguments

...	One or more QueryBuilder objects

Details

Union is based on the following fields (combined into a comparison key):

- group
- chromosome
- start
- end
- strand

The goal is to have a single list of junctions where every junction occurs in at least one compilation and if a junction occurs in > 1 compilations it still only has a single row representing all the samples across compilations that it appears in. Sample aggregate statistics will be recalculated for junctions which are merged across *all* samples from all compilations:

- sample_count
- coverage_sum
- coverage_avg
- coverage_median

Value

A RangedSummarizedExperiment of junctions appearing in at least one compilation

See Also

`junction_intersection()`

Examples

```
# Using query builder wrappers
sb1 <- QueryBuilder(compilation = "gtex", regions = "chr1:1879786-1879786")
sb1 <- set_coordinate_modifier(sb1, Coordinates$EndIsExactOrWithin)
sb1 <- set_row_filters(sb1, strand == "-")

sb2 <- QueryBuilder(compilation = "tcga", regions = "chr1:1879786-1879786")
sb2 <- set_coordinate_modifier(sb2, Coordinates$EndIsExactOrWithin)
sb2 <- set_row_filters(sb2, strand == "-")

junction_union(sb1, sb2)
```

---

percent_spliced_in *Relative measure of splice variant usage, limited currently to cassette exon splice variants*

Description

Similar to the JIR, this calculates Percent Spliced In (PSI) statistics for the definition of 2 different groups: inclusion and exclusion. Currently this function only supports the cassette exon use case.

Usage

```
percent_spliced_in(
  inclusion_group1,
  inclusion_group2,
  exclusion_group,
  min_count = 20,
  group_names = NULL
)
```

Arguments

`inclusion_group1`, `inclusion_group2`, `exclusion_group`	Where each is a list of 1 or more QueryBuilder objects
`min_count`	minimum total count (denominator) required to not be assigned -1
`group_names`	Optional vector of strings representing the group names

Details

Inclusion typically defines 2 basic queries, one for the junction preceding the cassette exon, and the second for the junction following the cassette exon. The exclusion group contains one basic query which defines the junction which skips the cassette exon.

The PSI itself is implemented as:

`PSI(inclusion1, inclusion2, exclusion) = mean(inclusion1, inclusion2) / (mean(inclusion1, inclusion2) + exclusion)`

where each term denotes the coverage of junctions that resulted from the basic queries in that group in the current sample.

Value

A DataFrame of samples, with their PSI score and metadata, which had > 0 coverage in at least one resulting row in at least one of the groups

Examples

```
in1 <- QueryBuilder(compilation = "srav2", regions = "chr1:94468008-94472172")
in1 <- set_coordinate_modifier(in1, Coordinates$Exact)
in1 <- set_row_filters(in1, strand == "+")

in2 <- QueryBuilder(compilation = "srav2", regions = "chr1:94468008-94472172")
in2 <- set_coordinate_modifier(in2, Coordinates$Exact)
in2 <- set_row_filters(in2, strand == "+")

ex <- QueryBuilder(compilation = "srav2", regions = "chr1:94468008-94475142")
ex <- set_coordinate_modifier(ex, Coordinates$Exact)
ex <- set_row_filters(ex, strand == "+")

percent_spliced_in(list(in1), list(in2), list(ex))
```

QueryBuilder *Construct a QueryBuilder object given a compilation and one or regions.*

Description

Construct a QueryBuilder object given a compilation and one or regions.

Usage

```
QueryBuilder(compilation, regions)
```

Arguments

compilation	A single string containing the name of the Snaptron data source. Any variant of the Compilation enum can also be passed an argument.
regions	Either a list of 1 more HUGO gene names as strings e.g. "BRCA1" or a Granges class object containing one or more geneomic intervals (e.g. "chr1:1-1000").

Value

A QueryBuilder object.

Examples

```
# contruct a query builder for GTEX data source and BRAC1 gene
qb <- QueryBuilder(compilation = Compilation$gtex, regions = "BRCA1")

# contruct a query builder for TCGA data source and chromosome region
qb <- QueryBuilder(compilation = "tcga", regions = "chr1:1-1000")

# construct a query builder for TCGA data source using GRanges object
library(GenomicRanges)
qb <- QueryBuilder(compilation = "tcga", regions = GRanges("chr1", "1-1000"))
```

---

query_jx *Query Junctions/Genes/Exons*

Description

Given one or more gene names or genomic range intervals it will return a list of 0 or more genes, junctions, or exons (depending on which query form is used) which overlap the ranges.

Usage

```
query_jx(sb, return_rse = TRUE, split_by_region = FALSE)

query_gene(sb, return_rse = TRUE, split_by_region = FALSE)

query_exon(sb, return_rse = TRUE, split_by_region = FALSE)
```

Arguments

sb	A SnaptronQueryBuilder object
return_rse	Should the query data be returned as a simple data frame or converted to a RangedSummarizedExperiment.
split_by_region	By default the results from multiple queries will be returned in a RangedSummarizedExperiment object with a rowData entry for each, labeling each result row according to the query it resulted from. However, if this is set to TRUE, the result will be a list of RangedSummarizedExperiment objects, one per original interval/gene. This latter option may be useful, but it requires a separate copy of the sample metadata for each original interval/gene.

Value

Functions will return either a RangedSummarizedExperiment or data.table depending on whether the return_rse parameter is set to TRUE or FALSE.

Examples

```
# Contruct a QueryBuilder object
qb <- QueryBuilder(compilation = "gtex", regions = "chr1:1-100000")
qb <- set_row_filters(qb, samples_count >= 20)
query_jx(qb)

qb <- set_row_filters(qb, NULL)
qb <- set_column_filters(qb, SMTS == "Brain")
query_gene(qb)
```

`shared_sample_counts` *Shared Sample Count (SSC): counts total number of samples in which 2 different junctions both occur in.*

Description

This produces a list of user-specified groups and the read coverage of the junctions in all the samples which were shared across all the basic queries occurring in each group.

Usage

```
shared_sample_counts(..., group_names = NULL)
```

Arguments

`...`	One or more lists of QueryBuilder objects
`group_names`	Optional vector of character strings representing group names

Details

Example: User defines a single group of junctions "GroupA" made up of 2 separate regions (two basic queries).

An SSC query will return a single line for GroupA which will have the total number of samples which had at least one junction which was returned from both basic queries. It will also report a summary statistic of the total number of groups which had one or more samples that were shared across the basic queries, in this case it would be 1. Also, it will report the number of groups which had at least one shared sample and which had matching junctions (from the query) which were fully annotated.

This function can be used to determine how much cross-sample support there is for a particular junction configuration (typically a cassette exon).

Value

A DataFrame of results based on the list of groups passed in via "group_names". Each group is reported with the # of unique samples which occurred in all of its defined set of related basic queries (e.g. two inclusion basic queries in a cassette exon scenario).

Examples

```
g1 <- QueryBuilder(compilation = "gtex", regions = "chr1:1879786-1879786")
g1 <- set_coordinate_modifier(g1, Coordinates$EndIsExactOrWithin)
g1 <- set_row_filters(g1, strand == "-")

g2 <- QueryBuilder(compilation = "gtex", regions = "chr1:1879903-1879903")
g2 <- set_coordinate_modifier(g2, Coordinates$StartIsExactOrWithin)
g2 <- set_row_filters(g2, strand == "-")

ssc<-shared_sample_counts(list(g1, g2))
```

`tissue_specificity` *Tissue Specificity (TS): produces a list of samples with their tissues marked which either contain queried junctions (1) or not (0); can be used as input to significance testing methods such as Kruskal-Wallis to look for tissue enrichment (currently only works for the GTEx compilation).*

Description

Lists the number of samples labeled with a specific tissue type. Samples are filtered for ones which have junctions across all the user-specified groups. That is, if a sample only appears in the results of some of the groups (from their basic queries) it will be assigned a 0, otherwise if it is in all of the groups' results it will be assigned a 1. This is similar to the SSC high level query type, but doesn't sum the coverage.

Usage

```
tissue_specificity(..., group_names = NULL)
```

Arguments

`...`	One or more QueryBuilder objects
`group_names`	Optional vector of strings representing the group names

Details

The samples are then grouped by their tissue type (e.g. Brain). This is useful for determining if there's an enrichment for a specific tissue in the set of junctions queried. Results from this can be fed to a statistical test, such as the Kruskal-wallis non-parametric rank test. This query is limited to GTEx only, due to the fact that GTEx is one of the few compilations that has consistent and complete tissue metadata.

Value

A DataFrame of all samples in the compilation with either a 0 or 1 indicating their occurrence and shared status (if > 1 group passed in). Occurrence here is if the sample has at least one result with > 0 coverage, and further, if > 1 group is passed in, then if it occurs in the results of all groups. Also includes the sample tissue type and sample_id.

Examples

```
in1 <- QueryBuilder(compilation = "gtex", regions = "chr4:20763023-20763023")
in1 <- set_coordinate_modifier(in1, Coordinates$EndIsExactOrWithin)
in1 <- set_row_filters(in1, strand == "-")

in2 <- QueryBuilder(compilation = "gtex", regions = "chr4:20763098-20763098")
in2 <- set_coordinate_modifier(in2, Coordinates$StartIsExactOrWithin)
in2 <- set_row_filters(in2, strand == "-")

tissue_specificity(list(in1, in2))
```

---

uri_of_last_successful_request

Return the URI of the last successful request to Snaptron

Description

This function can be paired with the `from_url` method from the QueryBuilder class, allowing users to share sources of data from Snaptron.

Usage

```
uri_of_last_successful_request()
```

Value

URI of last successful request to Snaptron or `NULL` if there have not been any successful requests.

Examples

```
qb <- QueryBuilder(compilation = "gtex", regions = "CD99")
query_jx(qb)
uri_of_last_successful_request()
```

Index