Package 'TxRegInfra'

November 5, 2020

Title Metadata management for multiomic specification of transcriptional regulatory networks

Description This package provides interfaces to genomic metadata employed in regulatory network creation, with a focus on noSQL solutions. Currently quantitative representations of eQTLs, DnaseI hypersensitivity sites and digital genomic footprints are assembled using an out-of-memory extension of the RaggedExperiment API.

Version 1.10.0

Author Vince Carey

Depends R (>= 3.5), RaggedExperiment (>= 1.3.11), mongolite

Imports methods, rjson, GenomicRanges, IRanges, BiocParallel, GenomeInfoDb, S4Vectors, SummarizedExperiment, utils

Suggests knitr, GenomicFiles, EnsDb.Hsapiens.v75, testthat, shiny, biovizBase (>= 1.27.2), Gviz, AnnotationFilter, ensembldb, ontoProc, rjson, graph, TFutils (>= 1.5.4)

Maintainer VJ Carey <stvjc@channing.harvard.edu>

License Artistic-2.0

LazyLoad yes

LazyData yes

biocViews Network

VignetteBuilder knitr

RoxygenNote 6.1.1

Encoding UTF-8

PackageStatus Deprecated

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

git_branch RELEASE_3_12

git_last_commit 386b161

git_last_commit_date 2020-10-27

Date/Publication 2020-11-04

R topics documented:

`addsyms` *add gene symbols to a GRanges that uses ensembl identifiers*

Description

add gene symbols to a GRanges that uses ensembl identifiers

Usage

```
addsyms(x, EnsDb = EnsDb.Hsapiens.v75::EnsDb.Hsapiens.v75)
```

Arguments

`x`	a GRanges assumed to have mcols element 'gene_id' with ensembl gene identifiers
`EnsDb`	an instance of EnsDb

Value

a GRanges with mcols column 'symbol'

Note

At this time no checking of reference genome consistency is performed.

`basicColData` *basicColData: metadata about a small collection of files for demonstrating TxRegInfra*

Description

basicColData: metadata about a small collection of files for demonstrating TxRegInfra

Usage

```
basicColData
```

Format

DataFrame from S4Vectors

Examples

```
data(basicColData)
head(basicColData)
```

`basicFormatter` *operate on a character vector to derive a DataFrame, splitting on a tokenand retrieving first and last split fragments as 'base' and 'type' fields*

Description

operate on a character vector to derive a DataFrame, splitting on a tokenand retrieving first and last split fragments as 'base' and 'type' fields

Usage

```
basicFormatter(x, spltok = "_")
```

Arguments

`x`	character vector
`spltok`	token to use in strsplit

Value

a DataFrame instance

Examples

```
some = c('Adipose_Subcutaneous_allpairs_v7_eQTL',
  'CD14_DS17215_hg19_FP',
  'CD19_DS17186_hg19_FP',
  'ENCFF001WGV_hg19_HS',
  'ENCFF994OCD_hg19_HS')
basicFormatter(some)
```

---

demo_eQTL_granges *a GRanges instance with eQTL metadata returned by 'sbov'*

Description

a GRanges instance with eQTL metadata returned by 'sbov'

Usage

```
demo_eQTL_granges
```

Format

a GRanges instance

Examples

```
names(mcols(demo_eQTL_granges))
```

---

demo_fimo_granges *a list of GRanges instances with TF FIMO scores returned by 'fimo_granges'*

Description

a list of GRanges instances with TF FIMO scores returned by 'fimo_granges'

Usage

```
demo_fimo_granges
```

Format

a list of GRanges instances

Examples

```
names(demo_fimo_granges)
head(mcols(demo_fimo_granges$VDR[[1]]))
```

dgf_meta *dgf_meta: metadata about a small collection of bed files for demonstrating TxRegInfra*

Description

dgf_meta: metadata about a small collection of bed files for demonstrating TxRegInfra

Usage

```
dgf_meta
```

Format

data.frame

Examples

```
data(dgf_meta)
head(dgf_meta)
```

dnmeta *metadata about DNaseI hotspots from ENCODE*

Description

metadata about DNaseI hotspots from ENCODE

Usage

```
dnmeta
```

Format

data.frame

Examples

```
data(dnmeta)
head(dnmeta[,1:10])
```

dnmetaApp *support search of hotspot/peak data from ENCODE*

Description

support search of hotspot/peak data from ENCODE

Usage

```
dnmetaApp()
```

Value

data.frame of metadat about selections

Examples

```
if (interactive()) {
    oask = options()$example.ask
    if (askYesNo("start app?"))
           require("shiny")
           dnmetaApp()
    options(example.ask=oask)
    }
```

getDocumentFields *determine the fields present in a txregnet document*

Description

determine the fields present in a txregnet document

Usage

```
getDocumentFields(rme, docTypeName = "type")
```

Arguments

rme	instance of RaggedMongoExperiment
docTypeName	character(1) telling the name of the column of colData(rme) that supplies information on document type

Value

a character vector

Examples

```
getDocumentFields
```

getFieldNames *get names of fields in a collection in remote txregnet*

Description

get names of fields in a collection in remote txregnet

Usage

```
getFieldNames(collection, check = TRUE, url = URL_txregInAWS(),
  db = "txregnet", limitn = 1)
```

Arguments

collection	character(1) name of collection
check	logical(1) if TRUE will verify that coll is present
url	character(1) mongodb url
db	character(1) mongodb db name
limitn	numeric(1) number of records to probe to get field names

Value

a vector of strings

Examples

```
getFieldNames('CD34_DS12274_hg19_FP', check=FALSE) # we know this collection is there
```

grConverter *convert a GRanges to a JSON query for mongodb*

Description

convert a GRanges to a JSON query for mongodb

Usage

```
grConverter(queryGRange, cfields = c(chrom = "chrom", start =
  "chromStart", end = "chromEnd"))
```

Arguments

queryGRange	a GRanges-class instance of length 1
cfields	a named character(3) vector with names 'chrom', 'start', 'end'; the element values will be used to name document fields in the query

Value

a JSON document generated by rjson::toJSON

Examples

```
gr = GenomicRanges::GRanges('chr1', IRanges(1,25000))
grConverter(gr, cfields=c(chrom='chr', start='start', end='end'))
```

`importBedToMongo` *arrange import to mongo using mongoimport, setting up type and fields appropriately*

Description

arrange import to mongo using mongoimport, setting up type and fields appropriately

Usage

```
importBedToMongo(path, collectionName, bedType = "narrowPeak",
  dbname = "db", importCmd = "mongoimport", host = "127.0.0.1")
```

Arguments

`path`	path to bed file (not compressed)
`collectionName`	name to use in mongodb
`bedType`	one of 'narrowPeak', 'broadPeak', 'chromHMM': contact developers for other types if desired
`dbname`	mongodb database name, used directly with system2('mongoimort ...')
`importCmd`	how to invoke 'mongoimport', default is to assume it can be found in PATH
`host`	host identifier for mongoimport, defaults to 127.0.0.1

Value

if error encountered, return the try-error content, otherwise TRUE

Examples

```
f1 = dir(system.file('bedfiles', package='TxRegInfra'), full=TRUE, patt='ENCFF971VCD')
f2 = dir(system.file('bedfiles', package='TxRegInfra'), full=TRUE, patt='E096_imp12')
if (verifyHasMongoCmd('mongoimport')) {
chk1 = importBedToMongo(f1, 'vjc1', db='txregnet')
stopifnot(chk1)
chk2 = importBedToMongo(f2, 'vjc2', db='txregnet', bedType='chromHMM')
stopifnot(chk2)
system2("mongo", args=c("txregnet", "--eval", "'db.vjc1.remove({})'"))
system2("mongo", args=c("txregnet", "--eval", "'db.vjc2.remove({})'"))
}
```

listAllCollections *list all collections in a database, using command-line interface*

Description

list all collections in a database, using command-line interface

Usage

```
listAllCollections(url = "mongodb://127.0.0.1:27017", db = "test",
  lisproc = function(x) {     ind = grep("MongoDB server", x)[1]
  x[-seq_len(ind)] })
```

Arguments

url	character(1) mongodb URL
db	character(1) mongodb database name
lisproc	a function that processes the reply to 'mongo ... –eval 'db.getCollectionNames()' to extract JSON, defaults to a function that removes all (header) records up to the one containing 'MongoDB server'

Value

vector of strings

Examples

```
if (verifyRunningMongodb()) listAllCollections()
```

makeAggregator *generate JSON to aggregate (counting records, and, by default, averaging a given variable) within a collection*

Description

generate JSON to aggregate (counting records, and, by default, averaging a given variable) within a collection

Usage

```
makeAggregator(by = "chrom", vbl = "chromStart", opname = "average",
  op = "$avg")
```

Arguments

by	character(1) telling the field for stratifying records for aggregation
vbl	character(1) telling field with numerical value for which a statistic will be computed within strata defined by 'by'
opname	character(1) define the name of the aggregation
op	character(1) evaluating to a mongo aggregation operator like '$avg' or '$min'

Value

a JSON document as produced by rjson::toJSON

Note

This produces json that can be used as an argument to m$aggregate() for m a mongolite::mongo instance

Examples

```
makeAggregator()
if (interactive() & verifyHasMongoCmd()) {
   remURL = URL_txregInAWS()
   colls = listAllCollections( url=remURL, db = 'txregnet')
   m1 = mongo(url = remURL, db = 'txregnet',
     collection='CD14_DS17215_hg19_FP')
# find minimum value of statistic 'stat' per chromosome
   newagg = makeAggregator( by='chr',
     vbl='stat', op='$min', opname='min')
   tab = m1$aggregate( newagg )
   head(tab)
   }
```

---

makeColData *generate a colData component corresponding to a mongodb*

Description

generate a colData component corresponding to a mongodb

Usage

```
makeColData(url = URL_txregInAWS(), db = "txregnet",
  formatter = basicFormatter)
```

Arguments

url	character(1) url for mongodb
db	character(1) database name
formatter	a function that takes in a character vector and returns a DataFrame with number of rows equal to the length of input

Value

a DataFrame instance

Examples

```
if (verifyHasMongoCmd()) makeColData()
```

`makeGRConverterList` *generate a list of GRanges to JSON for queries to mongo*

Description

generate a list of GRanges to JSON for queries to mongo

Usage

```
makeGRConverterList(rme, map = basicCfieldsMap(), docTypeName = "type")
```

Arguments

`rme`	RaggedMongoExperiment instance
`map`	list of lists of named character vectors
`docTypeName`	character(1) that identifies sample 'type'

Value

a list of JSON documents

`oldbasicColData` *oldbasicColData: metadata about a small collection of files for demonstrating TxRegInfra*

Description

oldbasicColData: metadata about a small collection of files for demonstrating TxRegInfra

Usage

```
oldbasicColData
```

Format

DataFrame from S4Vectors

Examples

```
data(oldbasicColData)
head(oldbasicColData)
```

ragged41FP *ragged41FP: A RaggedExperiment instance with digital genomic footprints over the coding region of ORMDL3*

Description

ragged41FP: A RaggedExperiment instance with digital genomic footprints over the coding region of ORMDL3

Usage

```
ragged41FP
```

Format

DataFrame

Note

The text on plot refers to FOS = 'footprint occupancy score' as in Neph et al, Nature 489, 6 Sept 2012 p 84.

Examples

```
data(ragged41FP)
ragged41FP
dim(ca <- compactAssay(ragged41FP,3)) # stat
dim(sparseAssay(ragged41FP,3)) # stat
opar = par(no.readonly=TRUE)
par(mar=c(4,11,4,3), bg='lightgray')
image(ca,
    main='over ORMDL3', axes=FALSE)
labs = gsub('_DS.*._hg19_FP', '', colnames(ragged41FP))
axis(2, at=seq(0,1,length=41), ylab='41 tissues',
    labels=labs, cex.axis=.6, las=2)
mtext('positions on chr17 not to scale\n(red = lower FOS = stronger binding capacity)', 1, line=1)
## Not run:  # if (interactive()) {
  m1 = mongolite::mongo(url=URL_txregInAWS(), db='txregnet')
  cd = makeColData(url=URL_txregInAWS(), db='txregnet')
  rme1 = RaggedMongoExpt(m1, cd[which(cd$type=='FP'),])
  BiocParallel::register(BiocParallel::SerialParam()) # necessary for mac?
  raggHHIP = sbov(rme1, GRanges('chr4', IRanges(145565173, 145605173)))
  ca = compactAssay(raggHHIP,3)[seq_len(200),]
  image(ca, main='over HHIP', axes=FALSE)
  labs = gsub('_DS.*._hg19_FP', '', colnames(ca))
  axis(2, at=seq(0,1,length=ncol(ca)), ylab=paste(ncol(ca), 'tissues'),
    labels=labs, cex.axis=.6, las=2)
 mtext('positions on chr4 not to scale\n(red = lower FOS = stronger binding capacity', 1, line=1)
#  }

## End(Not run)
par(opar)
```

RaggedMongoExpt *bind colData to a mongo-based ragged-experiment incubator*

Description

bind colData to a mongo-based ragged-experiment incubator

Usage

```
RaggedMongoExpt(con, colData)
```

Arguments

con	a mongolite::mongo instance
colData	a DataFrame instance

Value

instance of RaggedMongoExpt

sbov *prototype of subsetter for mongo resource*

Description

prototype of subsetter for mongo resource

Usage

```
sbov(rme, gr, map = basicCfieldsMap(), docTypeName = "type",
  simplify = TRUE)
```

Arguments

rme	RaggedMongoExpt instance
gr	GRanges instance to subset by
map	list with one element per document type telling what fields are chr, start, stop
docTypeName	character(1) naming column of colData(rme) that has document type
simplify	logical(1) if TRUE, a GRanges is assembled; if FALSE, a RaggedExperiment is returned

Value

if simplify is TRUE (the default), a GRanges is returned with assay components in the mcols; otherwise a RaggedExperiment instance, with metadata component 'docType' set to propagate the input type

Note

This is a placeholder for a proper subsetByOverlaps method. While it would appear sensible to allow subsetting for diverse assay types (e.g., eQTL, DGF), in version 1.5.x this is not supported. Each call can operate on only one sample type.

Examples

```
requireNamespace('mongolite')
if (verifyHasMongoCmd()) {  # for makeColData, which must be able to enumerate collections,
                  # and thus must be able to run system (as opposed to mongolite function) 'mongo'
 m1 = mongolite::mongo(url=URL_txregInAWS(), db='txregnet')
 #cd = makeColData(url=URL_txregInAWS(), db='txregnet')
 cd = TxRegInfra::basicColData
 rme1 = RaggedMongoExpt(m1, cd[which(cd$type=='FP'),][seq_len(8),])
 BiocParallel::register(BiocParallel::SerialParam())
 qrng = GRanges('chr1', IRanges(1e6, 1.5e6))
 GenomeInfoDb::genome(qrng) = "hg19"
 ss = sbov(rme1, GRanges('chr1', IRanges(1e6, 1.5e6)), simplify=FALSE)
}
```

---

sbov_output_eQTL *a GRanges instance with eQTL metadata returned by 'sbov'*

Description

a GRanges instance with eQTL metadata returned by 'sbov'

Usage

```
sbov_output_eQTL
```

Format

a GRanges instance

Examples

```
head(sbov_output_eQTL)
```

---

sbov_output_FP *a GRanges instance with digital genomic footprint metadata returned by 'sbov'*

Description

a GRanges instance with digital genomic footprint metadata returned by 'sbov'

Usage

```
sbov_output_FP
```

Format

a GRanges instance

Examples

```
head(sbov_output_FP)
```

---

sbov_output_HS *a GRanges instance with DnaseI hotspot metadata returned by 'sbov'*

Description

a GRanges instance with DnaseI hotspot metadata returned by 'sbov'

Usage

```
sbov_output_HS
```

Format

a GRanges instance

Examples

```
head(sbov_output_HS)
```

---

sbov_to_graphNEL *convert sbov() output to annotated graphNEL*

Description

convert sbov() output to annotated graphNEL

Usage

```
sbov_to_graphNEL(sbovout, ...)
```

Arguments

sbovout	output of sbov() when applied to a TxRegInfra eQTL resource from mongodb
...	not used

Value

instance of graphNEL from Bioc graph package

Note

can convert to igraph via igraph::igraph.from.graphNEL

Examples

```
sbov_to_graphNEL(demo_eQTL_granges)
```

txmodels *use Gviz to render transcript models via GeneRegionTrack, but keep lightweight through requireNamespace and suggestion for installation*

Description

use Gviz to render transcript models via GeneRegionTrack, but keep lightweight through requireNamespace and suggestion for installation

Usage

```
txmodels(sym, gr, edb = "EnsDb.Hsapiens.v75", plot.it = FALSE,
  radius = 0, ...)
```

Arguments

sym	a gene symbol to be looked up in biovizBase::genesymbol table
gr	a GRanges instance, anticipated to be length 1
edb	a character(1) name of an EnsDb annotation package
plot.it	a logical(1) specifying whether Gviz::plotTracks should be run
radius	a numeric(1) specifying number to add to IRanges instance used to subset gene models from ensembldb::exonsBy output
...	passed to Gviz::GeneRegionTrack

Value

an instance of Gviz::GeneRegionTrack, invisibly returned

Examples

```
t0 = txmodels('ORMDL3', plot.it=TRUE, name='ORMDL3')
t1 = txmodels('ORMDL3', plot.it=FALSE, name='meta', collapseTranscripts='meta')
requireNamespace('Gviz')
Gviz::plotTracks(list(Gviz::GenomeAxisTrack(), t0, t1), showId=TRUE)
```

txregCollections *list collections in AWS mongo server for txregnet*

Description

list collections in AWS mongo server for txregnet

Usage

```
txregCollections(ignore = NULL, url = URL_txregInAWS(),
  db = "txregnet", cliparms = "--quiet --eval")
```

Arguments

ignore	NULL by default; otherwise an integer vector telling which lines of mongo db.getCollectionNames() result should be ignored
url	a valid mongodb URL
db	character(1) db name
cliparms	character(1) arguments to 'mongo', defaults to '–quiet –eval'

Value

a character vector of collection names

Note

Different mongodb servers can have different response prologues. The ignore parameter is there to bypass some of the initial text. However, with the –quiet option this may not be needed. We now search for "[" to start parsing the collection list output.

Examples

```
if (verifyHasMongoCmd()) txregCollections()[seq_len(5)]
```

URL_txregInAWS *return mongodb URL for working mongo server*

Description

return mongodb URL for working mongo server

Usage

```
URL_txregInAWS()
```

Value

character(1) URL for a hosted resource

Examples

```
URL_txregInAWS
```

URL_txregLocal *local mongodb txregnet*

Description

local mongodb txregnet

Usage

```
URL_txregLocal()
```

Value

a string with 127.0.0.1 instead of localhost, useful on macosx

verifyHasMongoCmd *check for existence of 'mongo' command, for db.getCollectionNames etc.*

Description

check for existence of 'mongo' command, for db.getCollectionNames etc.

Usage

```
verifyHasMongoCmd(cmd = "mongo")
```

Arguments

cmd	character(1) either 'mongo' or 'mongoimport'

Value

logical(1)

Note

we use mongoimport command to import tsv files; mongolite import 'method' not immediately useful for this

Examples

```
if (interactive()) verifyHasMongoCmd()
```

verifyRunningMongodb *check for accessible local mongodb*

Description

check for accessible local mongodb

Usage

```
verifyRunningMongodb(url = "mongodb://127.0.0.1")
```

Arguments

url	character(1) defining mongodb server

Value

logical(1)

Examples

```
if (interactive()) verifyRunningMongodb()
```

Index