cmapR Documentation

Release 1.0.0

Ted Natoli

Nov 20, 2018

Contents

Provided by the Connectivity Map, Broad Institute of MIT and Harvard. More information on our website

CHAPTER 1

Where to Start

- *Installation*
- Summary of Available Modules

CHAPTER 2

Other resources

- GitHub project

CHAPTER 3

Introductory info

3.1 Installation

3.1.1 Dependencies

Dependencies are listed in the `DESCRIPTION` file.

3.1.2 Installing from source

The easiest way to install the `cmapR` repository is to point your R's `install.packages` function at a tarball of the `cmapR` archive. You can generate this archive by cloning this repository and doing the following:

```
# make a gzip tar ball of the repo
R CMD build cmapR # makes cmapR_1.0.tar.gz

# check that the package is ok
R CMD check cmapR_1.0.tar.gz
```

Once you have created the tarball, open an R terminal and execute the following:

```
install.packages("cmapR_1.0.tar.gz", type="source", repos=NULL)
library("cmapR")
```

You can also source individual files as needed instead of installing the entire package.

```
# For example, just load the IO methods
source("cmapR/R/io.R")``
```

CHAPTER 4

High-level API reference

4.1 Data (data.R)

Loads data sets available for use. These include:

- `"gene_set"`: These are a collection of gene sets as used in the Lamb 2006 CMap paper.
- `"cdesc_char"`: An example table of metadata, as would be parsed from a text file or using parse.gctx.

> **Warning:** Initially, all columns are of type character.

- `"ds"`: An example of a GCT object with row and column data, and gene expression values in the matrix.

4.2 IO (io.R)

Generally, this module contains the GCT class and relevant method definitions. These are:

4.2.1 GCT Object (S4 class)

```
#An S4 class to represent a GCT object

@slot mat a numeric matrix
@slot rid a character vector of row ids
@slot cid a character vector of column ids
@slot rdesc a \code{data.frame} of row descriptors
@slot rdesc a \code{data.frame} of column descriptors
@slot src a character indicating the source (usually file path) of the data

@description The GCT class serves to represent annotated
```

(continues on next page)

(continued from previous page)

```
  matrices. The \code{mat} slot contains said data and the
  \code{rdesc} and \code{cdesc} slots contain data frames with
  annotations about the rows and columns, respectively

@seealso \code{\link{parse.gctx}}, \code{\link{write.gctx}}, \code{\link{read.gctx.
↪meta}}, \code{\link{read.gctx.ids}}
@seealso \link{http://clue.io/help} for more information on the GCT format

setClass("GCT",
         representation(
             mat = "matrix",
             rid = "character",
             cid = "character",
             rdesc = "data.frame",
             cdesc = "data.frame",
             version = "character",
             src = "character"
         )
)
```

4.2.2 GCTX parsing functions

Parse a .gct or .gctx file to GCT object

```
parse.gctx <- function(fname, rid=NULL, cid=NULL, set_annot_rownames=F, matrix_only=F)

@param fname path to the GCTX file on disk
@param rid either a vector of character or integer
  row indices or a path to a grp file containing character
  row indices. Only these indicies will be parsed from the
  file.
@param cid either a vector of character or integer
  column indices or a path to a grp file containing character
  column indices. Only these indicies will be parsed from the
  file.
@param set_annot_rownames boolean indicating whether to set the
  rownames on the row/column metadata data.frames. Set this to
  false if the GCTX file has duplicate row/column ids.
@param matrix_only boolean indicating whether to parse only
  the matrix (ignoring row and column annotations)

@details \code{parse.gctx} also supports parsing of plain text
  GCT files, so this function can be used as a general GCT parser.

@examples
gct_file <- system.file("extdata", "modzs_n272x978.gctx", package="roller")
(ds <- parse.gctx(gct_file))

# matrix only
(ds <- parse.gctx(gct_file, matrix_only=T))

# only the first 10 rows and columns
(ds <- parse.gctx(gct_file, rid=1:10, cid=1:10))

@family GCTX parsing functions
```

Parse row/column metadata only

```
read.gctx.meta <- function(gctx_path, dimension="row", ids=NULL, set_annot_rownames=T)

@param gctx_path the path to the GCTX file
@param dimension which metadata to read (row or column)
@param ids a character vector of a subset of row/column ids
  for which to read the metadata
@param set_annot_rownames a boolean indicating whether to set the
  \code{rownames} addtribute of the returned \code{data.frame} to
  the corresponding row/column ids.

@return a \code{data.frame} of metadata

@examples
gct_file <- system.file("extdata", "modzs_n272x978.gctx", package="roller")
# row meta
row_meta <- read.gctx.meta(gct_file)
str(row_meta)
# column meta
col_meta <- read.gctx.meta(gct_file, dimension="column")
str(col_meta)
# now for only the first 10 ids
col_meta_first10 <- read.gctx.meta(gct_file, dimension="column", ids=col_meta
↪$id[1:10])
str(col_meta_first10)

@family GCTX parsing functions
```

Parse row/column ids only

```
read.gctx.ids <- function(gctx_path, dimension="row")

#Read GCTX row or column ids

@param gctx_path path to the GCTX file
@param dimension which ids to read (row or column)

@return a character vector of row or column ids from the provided file

@examples
gct_file <- system.file("extdata", "modzs_n272x978.gctx", package="roller")
# row ids
rid <- read.gctx.ids(gct_file)
head(rid)
# column ids
cid <- read.gctx.ids(gct_file, dimension="column")
head(cid)

@family GCTX parsing functions
```

4.2.3 GCTX writing functions

Write a GCT object to disk in .gct format

```
write.gct <- function(ds, ofile, precision=4, appenddim=T, ver=3)

@param ds the GCT object
@param ofile the desired output filename
@param precision the numeric precision at which to
  save the matrix. See \code{details}.
@param appenddim boolean indicating whether to append
  matrix dimensions to filename
@param ver the GCT version to write. See \code{details}.

@details Since GCT is text format, the higher \code{precision}
  you choose, the larger the file size.
  \code{ver} is assumed to be 3, aka GCT version 1.3, which supports
  embedded row and column metadata in the GCT file. Any other value
  passed to \code{ver} will result in a GCT version 1.2 file which
  contains only the matrix data and no annotations.

@return NULL

@examples
\dontrun{
write.gct(ds, "dataset", precision=2)
}
@family GCTX parsing functions
```

Write a GCT object to disk in .gctx format

```
write.gctx <- function(ds, ofile, appenddim=T, compression_level=0, matrix_only=F)

@param ds a GCT object
@param ofile the desired file path for writing
@param appenddim boolean indicating whether the
  resulting filename will have dimensions appended
  (e.g. my_file_n384x978.gctx)
@param compression_level integer between 1-9 indicating
  how much to compress data before writing. Higher values
  result in smaller files but slower read times.
@param matrix_only boolean indicating whether to write
  only the matrix data (and skip row, column annotations)

@examples
\dontrun{
# assume ds is a GCT object
write.gctx(ds, "my/desired/outpath/and/filename")
}

@family GCTX parsing functions
```

Write a ‘‘data.frame‘‘ of metadata only to a GCTX file

```
write.gctx.meta <- function(ofile, df, dimension="row")

@param ofile the desired file path for writing
@param df the \code{data.frame} of annotations
@param dimension the dimension to annotate
  (row or column)
```

(continues on next page)

(continued from previous page)

```
@examples
\dontrun{
# assume ds is a GCT object
write.gctx.meta("/my/file/path", cdesc_char, dimension="col")
}
@family GCTX parsing functions
@keywords internal
```

4.2.4 Parsing GRP files

Parse a .grp file to vector

```
parse.grp <- function(fname)

@param fname the file path to be parsed
@return a vector of the contents of \code{fname}

@examples
grp_path <- system.file("extdata", "lm_epsilon_n978.grp", package="roller")
values <- parse.grp(grp_path)
str(values)

@family CMap parsing functions
@seealso \link{http://clue.io/help} for details on the GRP file format
```

4.2.5 Writing to .grp files

Write a vector to a .grp file

```
write.grp <- function(vals, fname)

@param vals the vector of values to be written
@param fname the desired file name

@examples
\dontrun{
write.grp(letters, "letter.grp")
}

@family CMap parsing functions
@seealso \link{http://clue.io/help} for details on the GRP file format
```

4.2.6 Parsing GMX files

Parse a .gmx file to a list

```
parse.gmx <- function(fname)

@param fname the file path to be parsed

@return a list of the contents of \code{fname}. See details.
```

(continues on next page)

(continued from previous page)

```
@details \code{parse.gmx} returns a nested list object. The top
  level contains one list per column in \code{fname}. Each of
  these is itself a list with the following fields:
  - \code{head}: the name of the data (column in \code{fname})
  - \code{desc}: description of the corresponding data
  - \code{len}: the number of data items
  - \code{entry}: a vector of the data items

@examples
gmx_path <- system.file("extdata", "lm_probes.gmx", package="roller")
gmx <- parse.gmx(gmx_path)
str(gmx)

@family CMap parsing functions
@seealso \link{http://clue.io/help} for details on the GMX file format
```

4.2.7 Parsing GMT files

Parse a .gmt file to a list

```
parse.gmt <- function(fname)

@param fname the file path to be parsed

@return a list of the contents of \code{fname}. See details.

@details \code{parse.gmt} returns a nested list object. The top
  level contains one list per row in \code{fname}. Each of
  these is itself a list with the following fields:
  - \code{head}: the name of the data (row in \code{fname})
  - \code{desc}: description of the corresponding data
  - \code{len}: the number of data items
  - \code{entry}: a vector of the data items

@examples
gmt_path <- system.file("extdata", "query_up.gmt", package="roller")
gmt <- parse.gmt(gmt_path)
str(gmt)

@family CMap parsing functions
@seealso \link{http://clue.io/help} for details on the GMT file format
```

4.2.8 Writing to GMT files

```
write.gmt <- function(lst, fname)

@param lst the nested list to write. See \code{details}.
@param fname the desired file name

@details \code{lst} needs to be a nested list where each
  sub-list is itself a list with the following fields:
  - \code{head}: the name of the data
```

(continues on next page)

(continued from previous page)

```
  - \code{desc}: description of the corresponding data
  - \code{len}: the number of data items
  - \code{entry}: a vector of the data items

@examples
\dontrun{
write.gmt(gene_set, "gene_set.gmt")
}

@family CMap parsing functions
@seealso \link{http://clue.io/help} for details on the GMT file format
```

4.2.9 Writing a `data.frame` to a tsv file

```
write.tbl <- function(tbl, ofile, ...)

@param tbl the \code{data.frame} to be written
@param ofile the desired file name
@param ... additional arguments passed on to \code{write.table}

@details This method simply calls \code{write.table} with some
  preset arguments that generate a unquoted, tab-delimited file
  without row names.

@examples
\dontrun{
write.tbl(cdesc_char, "col_meta.txt")
}

@seealso \code{\link{write.table}}
```

4.3 utils (utils.R)

4.3.1 Melting GCT objects

Transform a GCT object to long form (aka 'melt').

```
melt.gct(g, suffixes=NULL, remove_symmetries=F, keep_rdesc=T, keep_cdesc=T)

@description Utilizes the \code{\link{data.table::melt}} function to transform the
  matrix into long form. Optionally can include the row and column
  annotations in the transformed \code{\link{data.table}}.

@param g the GCT object
@param keep_rdesc boolean indicating whether to keep the row
  descriptors in the final result
@param keep_cdesc boolean indicating whether to keep the column
  descriptors in the final result
@param remove_symmetries boolean indicating whether to remove
  the lower triangle of the matrix (only applies if \code{g@mat} is symmetric)
@param suffixes the character suffixes to be applied if there are
```

(continues on next page)

(continued from previous page)

```
  collisions between the names of the row and column descriptors

@return a \code{\link{data.table}} object with the row and column ids and the matrix
  values and (optinally) the row and column descriptors

@examples
# simple melt, keeping both row and column meta
head(melt.gct(ds))

# update row/colum suffixes to indicate rows are genes, columns experiments
head(melt.gct(ds, suffixes = c("_gene", "_experiment")))

# ignore row/column meta
head(melt.gct(ds, keep_rdesc = F, keep_cdesc = F))

@family GCT utilities
```

4.3.2 Concatenating

Merge two GCT objects

```
merge.gct(g1, g2, dimension="row", matrix_only=F)

@param g1 the first GCT object
@param g2 the second GCT object
@param dimension the dimension on which to merge (row or column)
@param matrix_only boolean idicating whether to keep only the
  data matrices from \code{g1} and \code{g2} and ignore their
  row and column meta data

@examples
# take the first 10 and last 10 rows of an object
# and merge them back together
(a <- subset.gct(ds, rid=1:10))
(b <- subset.gct(ds, rid=969:978))
(merged <- merge.gct(a, b, dimension="row"))

@family GCT utilities
@export
```

4.3.3 Slicing

Slice a GCT object using the provided row and/or column ids

```
subset.gct(g, rid=NULL, cid=NULL)

@param g a gct object
@param rid a vector of character ids or integer indices for ROWS
@param cid a vector of character ids or integer indices for COLUMNS

@examples
# first 10 rows and columns by index
(a <- subset.gct(ds, rid=1:10, cid=1:10))
```

(continues on next page)

(continued from previous page)

```
# first 10 rows and columns using character ids
(b <- subset.gct(ds, rid=ds@rid[1:10], cid=ds@cid[1:10]))

identical(a, b) # TRUE

@family GCT utilities
```

4.3.4 Annotating

Given a GCT object and either a `data.frame` or a path to an annotation table, apply the annotations to the GCT using the given `keyfield`.

```
annotate.gct(g, annot, dimension="row", keyfield="id")

@description Given a GCT object and either a \code{\link{data.frame}} or
a path to an annotation table, apply the annotations to the
gct using the given \code{keyfield}.

@param g a GCT object
@param annot a \code{\link{data.frame}} or path to text table of annotations
@param dimension either 'row' or 'column' indicating which dimension
  of \code{g} to annotate
@param keyfield the character name of the column in \code{annot} that
  matches the row or column identifiers in \code{g}

@return a GCT object with annotations applied to the specified
  dimension

@examples
\dontrun{
 g <- parse.gctx('/path/to/gct/file')
 g <- annotate.gct(g, '/path/to/annot')
}

@family GCT utilities
```

4.3.5 Transpose

```
transpose.gct(g)

@param g the \code{GCT} object

@return a modified verion of the input \code{GCT} object
  where the matrix has been transposed and the row and column
  ids and annotations have been swapped.

@examples
transpose.gct(ds)

@family GCT utilties
@export
```

4.3.6 Math

Convert values in a matrix to ranks

```
rank.gct(g, dim="row")
@param g the \code{GCT} object to rank
@param dim the dimension along which to rank
  (row or column)

@return a modified version of \code{g}, with the
  values in the matrix converted to ranks

@examples
(ranked <- rank.gct(ds, dim="column"))
# scatter rank vs. score for a few columns
plot(ds@mat[, 1:3], ranked@mat[, 1:3],
  xlab="score", ylab="rank")

@family GCT utilities
```

CHAPTER 5

Meta-info about cmapR

5.1 Contribution guidelines

We welcome contributors! For your pull requests, please include the following:

- Sample code/file that reproducibly causes the bug/issue
- Documented code (include a docstring for new functions!) providing fix
- Unit tests evaluating added/modified methods.

5.2 FAQ

We will be adding FAQs as they come up.

5.3 BSD 3-Clause License

5.4 Citation Information

If you use GCTx and/or cmapR in your work, please cite Enache et al.