cmapPy Documentation

Release 3.0.0

Oana Enache, David Lahr, Lev Litichevskiy

Jun 22, 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

High-level API reference

2.1 API (clue_api_client)

To use the CLUE API client, put a copy of the file `example_cmapPy_config_file.cfg` in your home directory and name the copy `.cmapPy.cfg`. Replace the clue_api_user_key entries in that file with your CLUE API user key that you obtained from the CLUE website.

class cmapPy.clue_api_client.clue_api_client.**ClueApiClient**(*base_url=None*, *user_key=None*)

Basic class for running queries against CLUE api

run_count_query(*resource_name*, *where_clause*)

run a query (get) against CLUE api

Args: resource_name: str - name of the resource / collection to query - e.g. genes, perts, cells etc. where_clause: dictionary - contains where clause to pass to API to; uses loopback specification

Returns: dictionary containing the results of the query

run_filter_query(*resource_name*, *filter_clause*)

run a query (get) against the CLUE api, using the API and user key fields of self and the fitler_clause provided

Args: resource_name: str - name of the resource / collection to query - e.g. genes, perts, cells etc. filter_clause: dictionary - contains filter to pass to API to; uses loopback specification

Returns: list of dictionaries containing the results of the query

cmapPy.clue_api_client.gene_queries.**are_genes_in_api**(*my_clue_api_client*, *gene_symbols*)

determine if genes are present in the API

Args: my_clue_api_client: gene_symbols: collection of gene symbols to query the API with

Returns: set of the found gene symbols

2.2 GCT, GCTx (pandasGEXpress)

pandasGEXpress package (integrated with Python's pandas package) allowing users to easily read, modify, and write .gct and .gctx files. Note that .gctx files are more performant than .gct, and we recommend their use.

2.2.1 GCToo Class

class cmapPy.pandasGEXpress.GCToo.**GCToo**(*data_df*, *row_metadata_df=None*, *col_metadata_df=None*, *src=None*, *version=None*, *make_multiindex=False*, *logger_name='cmap_logger'*)

Class representing parsed gct(x) objects as pandas dataframes. Contains 3 component dataframes (row_metadata_df, column_metadata_df, and data_df) as well as an assembly of these 3 into a multi index df that provides an alternate way of selecting data.

2.2.2 Parsing

cmapPy.pandasGEXpress.parse.**parse**(*file_path*, *convert_neg_666=True*, *rid=None*, *cid=None*, *ridx=None*, *cidx=None*, *row_meta_only=False*, *col_meta_only=False*, *make_multiindex=False*)

Identifies whether file_path corresponds to a .gct or .gctx file and calls the correct corresponding parse method.

Input: Mandatory: - gct(x)_file_path (str): full path to gct(x) file you want to parse.

Optional: - convert_neg_666 (bool): whether to convert -666 values to numpy.nan or not

(see Note below for more details on this). Default = False.

- rid (list of strings): list of row ids to specifically keep from gctx. Default=None.
- cid (list of strings): list of col ids to specifically keep from gctx. Default=None.
- **ridx (list of integers): only read the rows corresponding to this** list of integer ids. Default=None.
- **cidx (list of integers): only read the columns corresponding to this** list of integer ids. Default=None.
- **row_meta_only (bool): Whether to load data + metadata (if False), or just row metadata (if True)** as pandas DataFrame
- **col_meta_only (bool): Whether to load data + metadata (if False), or just col metadata (if True)** as pandas DataFrame
- **make_multiindex (bool): whether to create a multi-index df combining** the 3 component dfs

Output:

- **out (GCToo object or pandas df): if row_meta_only or col_meta_only, then** out is a metadata df; otherwise, it's a GCToo instance containing content of parsed gct(x) file

Note: why does convert_neg_666 exist?

- In CMap–for somewhat obscure historical reasons–we use "-666" as our null value

for metadata. However (so that users can take full advantage of pandas' methods, including those for filtering nan's etc) we provide the option of converting these into numpy.NaN values, the pandas default.

2.2.3 Writing

cmapPy.pandasGEXpress.write_gctx.**write**(*gctoo_object*, *out_file_name*, *convert_back_to_neg_666=True*, *gzip_compression_level=6*, *max_chunk_kb=1024*, *matrix_dtype=<type 'numpy.float32'>*)

Writes a GCToo instance to specified file.

Input:

- gctoo_object (GCToo): A GCToo instance.
- out_file_name (str): file name to write gctoo_object to.
- convert_back_to_neg_666 (bool): whether to convert np.NAN in metadata back to "-666"
- gzip_compression_level (int, default=6): Compression level to use for metadata.
- max_chunk_kb (int, default=1024): The maximum number of KB a given chunk will occupy
- matrix_dtype (numpy dtype, default=numpy.float32): Storage data type for data matrix.

cmapPy.pandasGEXpress.write_gct.**write**(*gctoo*, *out_fname*, *data_null='NaN'*, *metadata_null='-666'*, *filler_null='-666'*, *data_float_format='%.4f'*)

Write a gctoo object to a gct file.

Args: gctoo (gctoo object) out_fname (string): filename for output gct file data_null (string): how to represent missing values in the data (default = "NaN") metadata_null (string): how to represent missing values in the metadata (default = "-666") filler_null (string): what value to fill the top-left filler block with (default = "-666") data_float_format (string): how many decimal points to keep in representing data

(default = 4 digits; None will keep all digits)

Returns: None

2.2.4 Concatenating

concat.py

This function is for concatenating gct(x) files together. You can tell it to find files using the file_wildcard argument, or you can tell it exactly which files you want to concatenate using the input_filepaths argument. The meat of this function are the hstack (i.e. horizontal concatenation of GCToo objects) and vstack (i.e. vertical concatenation).

Terminology: 'Common' metadata refers to the metadata that is shared between the loaded GCToo's. For example, if horizontally concatenating, the 'common' metadata is the row metadata. 'Concatenated' metadata is the other one; it's the metadata for the entries being concatenated together. For example, if horizontally concatenating, the 'concatenated' metadata is the column metadata because columns are being concatenated together.

There are 2 arguments that allow you to work around certain obstacles of concatenation.

1) If the 'common' metadata contains fields that are not the same in all files, then you will need to remove these fields using the fields_to_remove argument.

2) If the 'concatenated' metadata ids are not unique between different files, and you try to concatenate the files, an invalid GCToo would be formed (duplicate ids). To overcome this, use the reset_sample_ids argument. This will move the 'new' metadata ids to a new metadata field and replace the original ids with unique integers.

N.B. This script sorts everything!

exception cmapPy.pandasGEXpress.concat.**MismatchCommonMetadataConcatException**

cmapPy.pandasGEXpress.concat.**assemble_common_meta**(*common_meta_dfs*, *fields_to_remove*, *sources*, *remove_all_metadata_fields*, *error_report_file*)

Assemble the common metadata dfs together. Both indices are sorted. Fields that are not in all the dfs are dropped.

Args: common_meta_dfs (list of pandas dfs) fields_to_remove (list of strings): fields to be removed from the

common metadata because they don't agree across files

Returns: all_meta_df_sorted (pandas df)

cmapPy.pandasGEXpress.concat.**assemble_concatenated_meta**(*concated_meta_dfs*, *remove_all_metadata_fields*)

Assemble the concatenated metadata dfs together. For example, if horizontally concatenating, the concatenated metadata dfs are the column metadata dfs. Both indices are sorted.

Args: concated_meta_dfs (list of pandas dfs)

Returns: all_concated_meta_df_sorted (pandas df)

cmapPy.pandasGEXpress.concat.**assemble_data**(*data_dfs*, *concat_direction*)

Assemble the data dfs together. Both indices are sorted.

Args: data_dfs (list of pandas dfs) concat_direction (string): 'horiz' or 'vert'

Returns: all_data_df_sorted (pandas df)

cmapPy.pandasGEXpress.concat.**build_common_all_meta_df**(*common_meta_dfs*, *fields_to_remove*, *remove_all_metadata_fields*)

concatenate the entries in common_meta_dfs, removing columns selectively (fields_to_remove) or entirely (
remove_all_metadata_fields=True; in this case, effectively just merges all the indexes in common_meta_dfs).

Returns 2 dataframes (in a tuple): the first has duplicates removed, the second does not.

Args:

common_meta_dfs: collection of pandas DataFrames containing the metadata in the "common" direction of the
concatenation operation

fields_to_remove: columns to be removed (if present) from the common_meta_dfs remove_all_metadata_fields: boolean indicating that all metadata fields should be removed from the

common_meta_dfs; overrides fields_to_remove if present

Returns:

tuple containing all_meta_df: pandas dataframe that is the concatenation of the dataframes in common_meta_dfs, all_meta_df_with_dups:

cmapPy.pandasGEXpress.concat.**build_mismatched_common_meta_report**(*common_meta_df_shapes*, *sources*, *all_meta_df*, *all_meta_df_with_dups*)

Generate a report (dataframe) that indicates for the common metadata that does not match across the common metadata
which source file had which of the different mismatch values

Args: common_meta_df_shapes: list of tuples that are the shapes of the common meta dataframes sources: list of the source files that the dataframes were loaded from all_meta_df: produced from build_common_all_meta_df all_meta_df_with_dups: produced from build_common_all_meta_df

Returns: all_report_df: dataframe indicating the mismatched row metadata values and the corresponding source file

cmapPy.pandasGEXpress.concat.**concat_main**(*args*)
Separate method from main() in order to make testing easier and to enable command-line access.

cmapPy.pandasGEXpress.concat.**do_reset_ids**(*concatenated_meta_df*, *data_df*, *concat_direction*)
Reset ids in concatenated metadata and data dfs to unique integers and save the old ids in a metadata column.

Note that the dataframes are modified in-place.

Args: concatenated_meta_df (pandas df) data_df (pandas df) concat_direction (string): 'horiz' or 'vert'

Returns: None (dfs modified in-place)

cmapPy.pandasGEXpress.concat.**get_file_list**(*wildcard*)
Search for files to be concatenated. Currently very basic, but could expand to be more sophisticated.

Args: wildcard (regular expression string)

Returns: files (list of full file paths)

cmapPy.pandasGEXpress.concat.**hstack**(*gctoos*, *remove_all_metadata_fields=False*, *error_report_file=None*, *fields_to_remove=[]*, *reset_ids=False*)
Horizontally concatenate gctoos.

Args: gctoos (list of gctoo objects) remove_all_metadata_fields (bool): ignore/strip all common metadata when combining gctoos error_report_file (string): path to write file containing error report indicating

problems that occurred during hstack, mainly for inconsistencies in common metadata

fields_to_remove (list of strings): fields to be removed from the common metadata because they don't agree across files

reset_ids (bool): set to True if sample ids are not unique

Return: concated (gctoo object)

cmapPy.pandasGEXpress.concat.**reset_ids_in_meta_df**(*meta_df*)
Meta_df is modified inplace.

cmapPy.pandasGEXpress.concat.**vstack**(*gctoos*, *remove_all_metadata_fields=False*, *error_report_file=None*, *fields_to_remove=[]*, *reset_ids=False*)
Vertically concatenate gctoos.

Args: gctoos (list of gctoo objects) remove_all_metadata_fields (bool): ignore/strip all common metadata when combining gctoos error_report_file (string): path to write file containing error report indicating

problems that occurred during vstack, mainly for inconsistencies in common metadata

fields_to_remove (list of strings): fields to be removed from the common metadata because they don't agree across files

reset_ids (bool): set to True if row ids are not unique

Return: concated (gctoo object)

2.2.5 Converting .gct <-> .gctx

Command-line script to convert a .gct file to .gctx.

Main method takes in a .gct file path (and, optionally, an out path and/or name to which to save the equivalent .gctx) and saves the enclosed content to a .gctx file.

Note: Only supports v1.3 .gct files.

cmapPy.pandasGEXpress.gct2gctx.**gct2gctx_main**(*args*)
Separate from main() in order to make command-line tool.

Command-line script to convert a .gctx file to .gct.

Main method takes in a .gctx file path (and, optionally, an out path and/or name to which to save the equivalent .gct) and saves the enclosed content to a .gct file.

Note: Only supports v1.0 .gctx files.

cmapPy.pandasGEXpress.gctx2gct.**gctx2gct_main**(*args*)
Separate from main() in order to make command-line tool.

2.2.6 Extracting from .grp files

2.2.7 Subsetting

Slices a random subset of a GCToo instance of a user-specified size.

cmapPy.pandasGEXpress.random_slice.**make_specified_size_gctoo**(*og_gctoo*, *num_entries*, *dim*)
Subsets a GCToo instance along either rows or columns to obtain a specified size.

Input:

- og_gctoo (GCToo): a GCToo instance
- num_entries (int): the number of entries to keep
- dim (str): the dimension along which to subset. Must be "row" or "col"

Output:

- new_gctoo (GCToo): the GCToo instance subsetted as specified.

subset.py

Extract a subset of data from a GCT(x) file using the command line. ids can be provided as a list or as a path to a grp file. See subset_gctoo for the equivalent method to be used on GCToo objects.

cmapPy.pandasGEXpress.subset.**build_parser**()
Build argument parser.

cmapPy.pandasGEXpress.subset.**subset_main**(*args*)
Separate method from main() in order to make testing easier and to enable command-line access.

2.3 GRP, GMT (set_io)

set_io contains simple scripts for parsing two other common file types used by the Connectivity Map: GRP and GMT files. The GRP file is used for storing a single set of things (e.g. a single gene set), while the GMT file is used for

storing multiple sets of things (e.g. several gene sets).

Further details on GRP and GMT files can be found here.

2.3.1 Reading GRP files

cmapPy.set_io.grp.**read**(*in_path*)
Read a grp file at the path specified by in_path.

Args: in_path (string): path to GRP file

Returns: grp (list)

2.3.2 Writing GRP files

cmapPy.set_io.grp.**write**(*grp*, *out_path*)
Write a GRP to a text file.

Args: grp (list): GRP object to write to new-line delimited text file out_path (string): output path

Returns: None

2.3.3 Reading GMT files

cmapPy.set_io.gmt.**read**(*file_path*)
Read a gmt file at the path specified by file_path.

Args: file_path (string): path to gmt file

Returns:

gmt (GMT object): list of dicts, where each dict corresponds to one line of the GMT file

2.3.4 Verifying GMT integrity

cmapPy.set_io.gmt.**verify_gmt_integrity**(*gmt*)
Make sure that set ids are unique.

Args: gmt (GMT object): list of dicts

Returns: None

2.3.5 Writing GMT files

cmapPy.set_io.gmt.**write**(*gmt*, *out_path*)
Write a GMT to a text file.

Args: gmt (GMT object): list of dicts out_path (string): output path

Returns: None

CHAPTER 3

Other resources

- GitHub project
- Tutorials and additional reference

CHAPTER 4

Meta-info about cmapPy

4.1 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.

4.2 FAQ

We will be adding FAQs as they come up.

4.3 BSD 3-Clause License

4.4 Citation Information

If you use GCTx and/or cmapPy, please cite Enache et al.

Python Module Index

C

Index

W