codemetrics Documentation

Release 0.9.6

Elmotec

Sep 29, 2019

Contents:

CHAPTER 1

codemetrics

Mine your SCM for insight on your software. A work of love inspired by Adam Tornhill's books.

Code metrics is a simple Python module that leverage pandas and your source control management (SCM) tool togenerate insight on your code base.

- pandas: for data munching.
- lizard: for code complexity calculation.
- cloc.pl (script): for line counts from cloc
- For now, only Subversion and git are supported.

1.1 Installation

To install codemetrics, simply use pip:

```
pip install codemetrics
```

1.2 Usage

This is a simple tool that makes it easy to retrieve information from your Source Control Management (SCM) repository and hopefully gain insight from it.

```
import codemetrics as cm
import cm.git

log_df = cm.get_git_log()
ages_df = cm.get_ages(log_df)
```

To retrieve the number of lines changed by revision with Subversion:

```
import codemetrics as cm
import cm.git

log_df = cm.get_svn_log().set_index(['revision', 'path'])
log_df.loc[:, ['added', 'removed']] = log_df.reset_index().\
                                        groupby('revision').\
                                        apply(cm.svn.get_diff_stats, chunks=False)
```

See module documentation for more advanced functions or the example notebook

1.3 License

Licensed under the term of MIT License. See attached file LICENSE.txt.

1.4 Credits

- This package was inspired by Adam Tornhill's books.
- This package was created with Cookiecutter.

CHAPTER 2

Installation

2.1 Stable release

To install codemetrics, run this command in your terminal:

```
$ pip install codemetrics
```

This is the preferred method to install codemetrics, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process_log_entries.

2.2 From sources

The sources for codemetrics can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/elmotec/codemetrics
```

Or download the tarball:

```
$ curl  -OL https://github.com/elmotec/codemetrics/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER 3

codemetrics interface

Getting useful data from your source control management tool is really a 2 steps process: first you need to get the log entries (e.g. `svn log` or `git log`) as a pandas.DataFrame, then process this output with the functions described below.

The pandas.DataFrame returned by each SCM specific function contains colums corresponding to the fields of *codemetrics.scm.LogEntry*:

class codemetrics.scm.**LogEntry**(*revision*, *author*, *date*, *path*, *message*, *kind*, *action=None*, *textmods=True*, *propmods=False*, *copyfromrev=None*, *copyfrompath=None*, *added=None*, *removed=None*)

Data structure to hold git or svn data entries.

3.1 codemetrics.scm

Common logic for source control management tools.

Factor things common to git and svn.

class codemetrics.scm.**ChunkStats**(*path*, *chunk*, *first*, *last*, *added*, *removed*)

added
Alias for field number 4

chunk
Alias for field number 1

first
Alias for field number 2

last
Alias for field number 3

path
Alias for field number 0

removed
Alias for field number 5

class codemetrics.scm.**DownloadResult**(*revision*, *path*, *content*)

content
Alias for field number 2

path
Alias for field number 1

revision
Alias for field number 0

class codemetrics.scm.**LogEntry**(*revision*, *author*, *date*, *path*, *message*, *kind*, *action=None*, *textmods=True*, *propmods=False*, *copyfromrev=None*, *copyfrompath=None*, *added=None*, *removed=None*)
Data structure to hold git or svn data entries.

astuple()
Return the data as tuple.

changed
Sum of lines added and lines removed.

class codemetrics.scm.**ScmDownloader**(*command*, *client*)
Abstract class that defines a common interface for SCM downloaders.

download(*revision*, *path*)
Download content specific to a revision and path.

Runs checks and forward the call to _download (template method).

Parameters

- **revision** (str) – identify the commit ID
- **path** (Union[None, str]) – file path. Can be left as None if all files in the commit are to be retrieved.

Return type *DownloadResult*

codemetrics.scm.**parse_diff_as_tuples**(*download*)
Parse download result looking for diff chunks.

Parameters **download** (*DownloadResult*) – Download result.

Yields statistics, one tuple for each chunk (begin, end, added, removed).

Return type Generator[*ChunkStats*, None, None]

codemetrics.scm.**parse_diff_chunks**(*download*)
Concatenate chunks data returned by parse_diff_as_tuples into a frame

Return type DataFrame

3.2 codemetrics.svn

Getting your data from Subversion.

_SvnLogCollector related functions.

class codemetrics.svn.**SvnDownloader**(*command*, *svn_client='svn'*)
Download files from Subversion.

codemetrics.svn.**download**(*data*, *svn_client='svn'*)
Download results from Subversion.

Parameters

- **data** (Union[DataFrame, Series]) – pd.Series containing at least revision and path.
- **svn_client** (str) – Subversion client executable. Defaults to svn.

Return type *DownloadResult*

Returns list of file contents.

codemetrics.svn.**get_diff_stats**(*data*, *svn_client='svn'*, *chunks=None*)
Download diff chunks statistics from Subversion.

Parameters

- **data** (DataFrame) – revision ID of the change set.
- **svn_client** (str) – Subversion client executable. Defaults to svn.
- **chunks** – if True, return statistics by chunk. Otherwise, return just added, and removed column for each path. If chunk is None, defaults to true for data frame and false for series.

Return type Union[None, DataFrame]

Returns Dataframe containing the statistics for each chunk.

Example:

```
import pandas as pd
import codemetrics as cm
log = cm.get_svn_log().set_index(['revision', 'path'])
log.loc[:, ['added', 'removed']] = log.reset_index().\
                                   groupby('revision').\
                                   apply(cm.svn.get_diff_stats,
                                   chunks=False)
```

codemetrics.svn.**get_svn_log**(*path='.'*, *after=None*, *before=None*, *progress_bar=None*, *svn_client='svn'*, *relative_url=None*)
Entry point to retrieve svn log.

Parameters

- **path** (str) – location of checked out subversion repository root.
- **after** (Optional[datetime]) – only get the log after time stamp. Defaults to one year ago.
- **before** (Optional[datetime]) – only get the log before time stamp. Defaults to now.
- **progress_bar** (Optional[tqdm]) – tqdm.tqdm progress bar.
- **svn_client** (str) – Subversion client executable. Defaults to svn.
- **relative_url** (Optional[str]) – Subversion relative url (e.g. /project/trunk/).

Return type DataFrame

Returns pandas.DataFrame with columns matching the fields of *codemetrics.scm.LogEntry*.

Example:

```
last_year = datetime.datetime.now() - datetime.timedelta(365)
log_df = cm.svn.get_svn_log(path='src', after=last_year)
```

codemetrics.svn.**to_bool**(*bool_str*)
Convert str to bool.

codemetrics.svn.**to_date**(*datestr*)
Convert str to datetime.datetime.

The date returned by _SvnLogCollector is UTC according to git-svn man page. Date tzinfo is set to UTC.

added and removed columns are set to np.nan for now.

3.3 codemetrics.git

Getting your data from git.

Git related functions.

codemetrics.git.**download**(*data*, *git_client='git'*)
Downloads files from Subversion.

Parameters

- **data** (DataFrame) – dataframe containing at least a (path, revision) columns to identify the files to download.
- **git_client** (str) – Subversion client executable. Defaults to git.

Return type *DownloadResult*

Returns list of scm.DownloadResult.

codemetrics.git.**get_git_log**(*path='.'*, *after=None*, *before=None*, *progress_bar=None*, *git_client='git'*, *_pdb=False*)
Entry point to retrieve git log.

Parameters

- **path** (str) – location of checked out subversion repository root. Defaults to .
- **after** (Optional[datetime]) – only get the log after time stamp. Defaults to one year ago.
- **before** (Optional[datetime]) – only get the log before time stamp. Defaults to now.
- **git_client** (str) – git client executable (defaults to git).
- **progress_bar** (Optional[tqdm]) – tqdm.tqdm progress bar.
- **_pdb** (bool) – drop in debugger on parsing errors.

Return type DataFrame

Returns pandas.DataFrame with columns matching the fields of codemetrics.scm.LogEntry.

Example:

```
last_year = datetime.datetime.now() - datetime.timedelta(365)
log_df = cm.git.get_git_log(path='src', after=last_year)
```

3.4 codemetrics.core

The main functions are located in core but can be accessed directly from the main module.

For instance:

```
>>>import codemetrics as cm
>>>import cm.svn
>>>log_df = cm.svn.get_svn_log()
>>>ages_df = cm.get_ages(log_df)
```

codemetrics.core.**get_mass_changes**(*log*, *min_path=None*, *max_changes_per_path=None*)

Extract mass changesets from the SCM log data frame.

Calculate the number of files changed by each revision and extract that list according to the threshold.

Parameters

- **log** (DataFrame) – SCM log data is expected to contain at least revision, added, removed, and path columns.
- **min_path** (Optional[int]) – threshold for the number of files changed to consider the revision a mass change.
- **max_changes_per_path** (Optional[float]) – threshold for the number of changed lines (added + removed) per file that changed.

Return type DataFrame

Returns revisions that had more files changed than the threshold.

codemetrics.core.**get_ages**(*data*, *by=None*)

Generate age of each file based on last change.

Takes the output of a SCM log or just the date column and return get_ages.

Parameters

- **data** (DataFrame) – log or date column of log.
- **by** (Optional[Sequence[str]]) – keys used to group data before calculating the age. See pandas.DataFrame.groupby. Defaults to ['path'].

Return type DataFrame

Returns age of most recent modification as pandas.DataFrame.

Example:

```
get_ages = codemetrics.get_ages(log_df)
```

codemetrics.core.**get_hot_spots**(*log*, *loc*, *by=None*, *count_one_change_per=None*)

Generate hot spots from SCM and loc data.

Cross SCM log and lines of code as an approximation of complexity to determine paths that are complex and change often.

Parameters

- **log** – output log from SCM.
- **loc** – output from cloc.
- **by** – aggregation level can be path (default), another column.

- **count_one_change_per** – allows one to count one change by day or one change per JIRA instead of one change by revision.

Returns pandas.DataFrame

codemetrics.core.**get_co_changes**(*log=None*, *by=None*, *on=None*)

Generate co-changes report.

Returns a DataFrame with the following columns: - primary: first path changed. - secondary: second path changed. - coupling: how often do the path change together.

Parameters

- **log** – output log from SCM.
- **by** – aggregation level. Defaults to path.
- **on** – Field name to join/merge on. Defaults to revision.

Returns pandas.DataFrame

codemetrics.core.**guess_components**(*paths*, *stop_words=None*, *n_clusters=8*)

Guess components from an iterable of paths.

Parameters

- **paths** – list of string containing file paths in the project.
- **stop_words** – stop words. Passed to TfidfVectorizer.
- **n_clusters** – number of clusters. Passed to MiniBatchKMeans.

Returns pandas.DataFrame

See also:

sklearn.feature_extraction.text.TfidfVectorizer sklearn.cluster.MiniBatchKMeans

codemetrics.core.**get_complexity**(*group*, *download_func=None*)

Generate complexity information for files and revisions in dataframe.

For each pair of (path, revision) in the input dataframe, analyze the code with lizard and return the output.

Parameters

- **group** (Union[DataFrame, Series]) – contains at least path and revision values.
- **download_func** (Optional[Callable]) – callable that downloads a path on a given revision in a temporary directory and return that file in an object of type *codemetrics.scm.DownloadResult*.

Return type DataFrame

Returns Dataframe containing output of function-level lizard.analyze

Example:

```
import codemetrics as cm
log = cm.get_git_log()
log.groupby(['revision', 'path']).            apply(get_complexity, download_
↪func=cm.git.download)
```

3.5 codemetrics.vega

Brdges visualization in Jupyter notebooks with Vega and Altair.

codemetrics.vega.**build_hierarchy**(*data*, *get_parent=<function dirname>*, *root=''*, *max_iter=100*, *col_name=None*)

Build a hierarchy from a data set and a get_parent relationship.

The output frame adds 2 columns in front: id and parent. Both are numerical where the parent id identifies the id of the parent as returned by the get_parent function.

The id of the root element is set to 0 and the parent is set to np.nan.

Parameters

- **data** (DataFrame) – data containing the leaves of the tree.
- **get_parent** – function returning the parent of an element.
- **root** (str) – expected root of the hierarchy.
- **max_iter** (int) – maximum number of iterations.
- **col_name** (Optional[str]) – name of the column to use as input (default to column 0).

Return type DataFrame

Returns pandas.DataFrame with the columns id, parent and col_name. The parent value identifies the id of the parent in the hierarchy where the id 0 is the root. The columns other than col_name are discarded.

codemetrics.vega.**vis_ages**(*df*, *height=300*, *width=400*, *colorscheme='greenblue'*)

Convert get_ages output to a json vega dict.

Parameters

- **df** (DataFrame) – input data returned by codemetrics.get_ages()
- **height** (int) – vertical size of the figure.
- **width** (int) – horizontal size of the figure.
- **colorscheme** (str) – color scheme. See https://vega.github.io/vega/docs/schemes/

Return type dict

Returns Vega description suitable to be use with Altair.

Example:

```
import codemetrics as cm
from altair.vega.v4 import Vega
ages = cm.get_ages(loc_df, log_df)
desc = cm.vega.vis_ages(ages)
Vega(desc)  # display the visualization inline in you notebook.
```

See also:

Vega circle pack example

codemetrics.vega.**vis_hot_spots**(*df*, *height=300*, *width=400*, *size_column='lines'*, *color_column='changes'*, *colorscheme='yelloworangered'*)

Convert get_hot_spots output to a json vega dict.

Parameters

- **df** (DataFrame) – input data returned by `codemetrics.get_hot_spots()`
- **height** (int) – vertical size of the figure.
- **width** (int) – horizontal size of the figure.
- **size_column** (str) – column that drives the size of the circles.
- **color_column** (str) – column that drives the color intensity of the circles.
- **colorscheme** (str) – color scheme. See https://vega.github.io/vega/docs/schemes/

Return type dict

Returns Vega description suitable to be use with Altair.

Example:

```
import codemetrics as cm
from altair.vega.v4 import Vega
hspots = cm.get_hot_spots(loc_df, log_df)
desc = cm.vega.vis_hot_spots(hspots)
Vega(desc)  # display the visualization inline in you notebook.
```

See also:

Vega circle pack example

3.6 Command line scripts

3.6.1 cm_func_stats

The *codemetrics interface* offers a command line tool *cm_func_stats* to compute statistics on functions.

For now the statistics are limited to the number of line of code (LOC), the complexity of the function (CCN), and the most frequent tokens together with the their span (see https://www.fluentcpp.com/2018/10/23/word-counting-span/ for more information):

```
>cm_func_stats --help
Usage: cm_func_stats [OPTIONS] FILE_PATH LINE_NO

  Generate statistics on the function specified by FILE_PATH LINE_NO.

Options:
  --version  Show the version and exit.
  --help     Show this message and exit.
```

And for an example:

```
>cm_func_stats codemetrics\cmdline.py 42
codemetrics\cmdline.py(39): cm_func_stats@39-55@codemetrics\cmdline.py, NLOC: 16,␣
↪CCN: 4
codemetrics\cmdline.py(43): f occurs 7 time(s), spans 13 lines (76.47%)
codemetrics\cmdline.py(41): func_info occurs 4 time(s), spans 9 lines (52.94%)
codemetrics\cmdline.py(49): token occurs 3 time(s), spans 4 lines (23.53%)
codemetrics\cmdline.py(45): write occurs 2 time(s), spans 9 lines (52.94%)
codemetrics\cmdline.py(45): sys occurs 2 time(s), spans 9 lines (52.94%)
codemetrics\cmdline.py(45): stdout occurs 2 time(s), spans 9 lines (52.94%)
codemetrics\cmdline.py(48): span occurs 2 time(s), spans 5 lines (29.41%)
```

(continues on next page)

(continued from previous page)

```
codemetrics\cmdline.py(48): func_span occurs 2 time(s), spans 5 lines (29.41%)
codemetrics\cmdline.py(43): msg occurs 2 time(s), spans 2 lines (11.76%)
codemetrics\cmdline.py(39): line_no occurs 2 time(s), spans 3 lines (17.65%)
codemetrics\cmdline.py(39): file_path occurs 2 time(s), spans 3 lines (17.65%)
```

CHAPTER 4

Recipes

4.1 Getting added and removed lines from Subversion log

Subversion log (unlike git) does not provide the number of lines added and removed for each commit so *codemetrics* resort to a 2 passes process retrieving the log first:

```
log = cm.get_svn_log()
```

At this point, added and removed column will be NaN. To populate then run the second pass. It is slow because it relies on repeatedly calling `svn diff -c` for each revision:

```
log.loc[:, ['added', 'removed']] = log.groupby('revision').apply(cm.get_diff_stats)
```

Note `chunks=True` returns diff stats with a row for each diff chunks.

See also:

Function *codemetrics.svn.get_diff_stats*

4.2 Leverage dask to speed up retrieval of added and removed line with Subversion

Retrieving added and removed line with Subversion can be slow because codemetrics makes repeated calls to `svn diff --git -c XXX` to count the number of pluses and minuses. To speed up the process somewhat, one can try to leverage dask like so:

```
import dask.dataframe as dd
import dask.diagnostics as ddiags
import multiprocessing as mp

n_cpus = mp.cpu_count()
```

(continues on next page)

(continued from previous page)

```
log = cm.get_svn_log().reset_index()
meta = get_diff_stats(log[-1:]).iloc[0:0]
partitioned_log = dd.from_pandas(log, npartitions=n_cpus)
wf = partioned_log.groupby('revision').apply(get_diff_stats, chunks=False, meta=meta)
with ddiags.ProgressBar():  # optional
    addrem_df = wf.conpute()    # returns a pandas.DataFrame
```

Note that there is a significant overhead to start the parallel process.

CHAPTER 5

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

5.1 Types of Contributions

5.1.1 Report Bugs

Report bugs at https://github.com/elmotec/codemetrics/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

5.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

5.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

5.1.4 Write Documentation

codemetrics could always use more documentation, whether as part of the official codemetrics docs, in docstrings, or even on the web in blog posts, articles, and such.

5.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/elmotec/codemetrics/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

5.2 Get Started!

Ready to contribute? Here's how to set up *codemetrics interface* for local development.

1. Fork the *codemetrics interface* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:elmotec/codemetrics.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv codemetrics
$ cd codemetrics/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 codemetrics tests
$ python setup.py test or py.test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

5.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for all supported versions of Python. Check https://travis-ci.org/elmotec/codemetrics/pull_requests and make sure that the tests pass for all versions.

5.4 Tips

To run a subset of tests:

```
$ python -m unittest tests.test_codemetrics
```

5.5 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ python setup.py test  # there should be no errors.
$ bump2version patch  # possible: major / minor / patch.
$ git push
$ git push --tags
$ python setup.py sdist  # will generate new .tar.gz in dist folder.
$ twine check dist/<dist>.tar.gz  # status should be Passed
$ twine upload dist/<dist>.tar.gz
```

Travis will then deploy to PyPI if tests pass.

CHAPTER 6

Credits

6.1 Development Lead

- Elmotec

6.2 Contributors

None yet. Why not be the first?

6.3 Inspiration

Adam Tornhill

CHAPTER 7

History

7.1 0.9.6 (2019-09-29)

- Fixed incorrect usage of subprocess.run(). See https://github.com/elmotec/codemetrics/issues/1.

7.2 0.9.5 (2019-09-05)

- Factored common logic between git and svn. Bug fixes.

7.3 0.9.4 (2019-09-02)

- Fixed test_core following https://github.com/pandas-dev/pandas/pull/24748 (Pandas 0.25.X)
- Added script *cm_func_stats* that generates statistics on the function passed as argument.
- Added appveyor support for Windows.
- Documentation.

7.4 0.9.3 (2019-04-01)

- Fixed retrieval of added and removed lines when there are spaces in a file name.
- Fixed indexed input in `get_mass_changes`.
- Fixed handling of removed files in `svn.get_diff_stats`.
- Fixed handling of branches in `svn.get_diff_stats`.

7.5 0.9 (2019-03-19)

- Started changing interfaces to leverage apply and groupby.
- Added lines added/removed for Subversion.

7.6 0.8.2 (2019-02-26)

- Added `svn.get_diff_stats` to retrieve line changes stats per diff.

7.7 0.8 (2019-02-13)

- Integrated lizard to calculate average and function level cyclomatic complexity.

7.8 0.7 (2019-01-09)

- Function oriented interface.
- Visualization via Vega, Altair.
- Documentation.

7.9 0.6

- Alpha work.

7.10 0.5 (2018-05-12)

- First release on PyPI.

CHAPTER 8

Search and Index

- genindex
- search

Python Module Index

C

Index