unicity Documentation

Release 0.2

David Dempsey

Sep 01, 2019

CONTENTS

I wrote this library to test the code in the hundreds of Python files submitted by my programming class. It also can be used to spot code submissions that are suspiciously similar.

To get started, first install unicity using `pip`.

```
pip install unicity
```

Then download the examples from https://github.com/ddempsey/unicity. These will show you how to load a project, run a simple test, perform comparisons, and a few other tricks.

Some disclaimers:

- I have discovered that students learning to code are very innovative in the ways in which they will break your work flow. Therefore, I have built in catches for **directory changes** and **infinite loops**. But there will be other things I have not anticipated. You may catch and handle these in your test function, but I'd also like to hear if you think something should be added to make the testing more robust.
- The similarity checking should be treated as a screening tool to highlight *potential* instances of copying. It does not (it cannot) assert that copying has actually occurred, and the best way to do that is by interview.

CHAPTER

ONE

PYTHON CLASSES

Each submitter is called a **client**. Each client submits a **portfolio** of Python code. The set of all portfolios is called a **project**. The set of all clients is called a **cohort**.

A project is a collection of Python files submitted by several different clients. They should be contained in a zip archive or folder. Each file should follow a particular naming convention

‘‘ {clientname}_*{expected_file}*.{ext} ‘‘

where `{expected_file.ext}` is passed as an argument to the `Project` constructor.

Passing a path to a `cohort` file will give the `Project` additional data to tag each portfolio and report missing files.

Test parts of the submission using the `test` method. This will require you to define a test function, that in turn calls functions or classes from a client's portfolio.

Check similarity metrics across the entire project with the `compare` method. You can compare particular methods, exclude similarity deriving from a common template, and compare with past projects.

class unicity.**Project**(*project, expecting, ignore=['*'], cohort=None, root=None*)

Class for all client portfolios.

project [str] Path to folder or zip archive containing client portfolios.

expecting [list (optional)] List of expected file names in each portfolio.

ignore [list (optional)] List of file names to ignore (unix identifiers fine).

cohort [str (optional)] Path to file containing information about clients.

root [str (optional)] String for naming of output files.

client [dict] Dictionary of Client objects, keyed by client name.

clientlist [list] List of Client objects, ordered alphabetically by client name.

portfolio_status [dict] Contains completeness information about individual client portfolios.

test_status [dict] Contains testing information about individual client portfolios.

findstr Look for string instances in client portfolios.

dump_docstrings Write docstring information to file.

compare Compute similarity metrics between client portfolios.

similarity_report Produce a plot of similarity across the project.

summarise Output a summary of the project.

test Batch testing of the code.

Cohort file should contain client information in CSV format. The first row defines column names and must at least contain a 'name' column. Client information is given on separate rows.

For example, the contents of 'cohort.txt'

name, firstname, surname, email johnsonjunko, junko, johnson, j.johnson@unicity.co.nz trosttrisha, trisha, trost, t.trost@unicity.co.nz romoricki, ricki, romo, r.romo@unicity.co.nz mccardellmirta, mirta,mccardell, m.mccardell@unicity.co.nz

Each column is provisioned as a separate attribute for each client as discovered. If the 'email' attribute is present, this will be reported in output.

compare(*routine*, *metric='command_freq'*, *ncpus=1*, *template=None*, *prior_project=None*, *prior_routine=None*)

Compares pairs of portfolios for similarity between implemented Python routine.

routine [str] Callable sequence in unicity format for comparison at three levels: {file}, {file}/{function} or {file}/{class}.{method}.

metric [str, callable (optional)] Distance metric for determining similarity. See below for more information on the different metrics available. Users can pass their own metric functions but these must adhere to the specification below. (default 'command_freq')

ncpus [int (optional)] Number of CPUs to use when running comparisons. (default 1, single thread)

template [str (optional)] File path to Python template file for referencing the comparison.

prior_project [unicity.Project (optional)] Project object for different cohort. Check current cohort for similarities with previous. Comparisons will not be made between clients of the previous project.

prior_routine [str (optional)] Callable sequence in unicity format for comparison from prior_project. If not given, will default to routine.

c [unicity.Comparison] Object containing comparison information.

A template file is specified when Python files are likely to exhibit similarity due to portfolios developed from a common template. Each portfolio is compared to the template for similarity, and this is 'subtracted' from any similarity between a pair of portfolios.

Exercise caution when drawing conclusions of similarity between short code snippets as this increases the likelihood of false positives.

command_freq (default) [Compares the frequency of distinct commands in two scripts.] Commands counted include all callables (function/methods), control statements (if/elif/else), looping statements (for/while/continue/break), boolean operators (and/or/not) and try/except.

jaro : Compares the order of the statements above for matches and transpositions.

User can pass their own callable that computes a distance metric between two files. The specification for this callable is

file1 [File] Python File object for client 1.

file2 [File] Python File object for client 2.

template [File] Python File object for template file.

name1 [str] Routine name in first file for comparison. If not given, the comparison is assumed to operate on entire file.

name2 [str] Routine name in second file for comparison. If not given, the comparison is assumed to operate on entire file.

dist [float] Float between 0 and 1 indicating degree of similarity (0 = highly similar, 1 = highly dissimilar).

User metrics that raise exceptions are caught and passed, with the error message written to unicity_compare_errors.txt for debugging.

See examples/similarity_check.py for more details and examples.

dump_docstrings(*routine*, *save*)
Writes docstring information out to file.

routine [str] Callable sequence in unicity format {file}/{function}, or {file}/{class}.{method}.

save [str] File path to write docstrings.

findstr(*string*, *location=None*, *verbose=False*, *save=None*)
Check portfolio for instances of string.

string [str] String to locate in portfolios.

location [str (optional)] Search limited to particular file, class or routine, specified in unicity format *file*, *file/function*, or *file/class*.*method*.

verbose [bool (optional)] Flag to print discovered strings to screen (default False).

save [str (optional)] File to save results of string search.

found_clients [list] Client objects whose portfolios contain the search string.

similarity_report(*comparison*, *client=None*, *save=None*)
Creates a summary of similarity metrics.

comparison [unicity.Comparison] Comparison object produced by Project.compare.

client [str (optional)] Name of client to highlight or 'anon' for anonymised report.

save [str (optional)] Name of output file (default to {clientname}_{function}.png).

summarise(*save*, *verbose=False*)
Create log file with portfolio information.

save [str] File path to save output summary.

verbose [bool (optional)] Print output to screen as well.

test(*routine*, *ncpus=1*, *language='python'*, *client=None*, *timeout=None*, ***kwargs*)
Runs test suite for function name.

routine [str] Name of unit test function (not a function handle).

ncpus [int (optional)] Number of CPUs to use when running comparisons. (default 1, single thread)

client [str (optional)] Run test for individual client, writing their code out to file. (default is to run test for all clients.)

timeout [float (optional)] Time after which test should be exited. Only use timeout if you think there is the possiblity of infinite loops. Using a timeout will impact performance. Timeout only implemented for serial. (default is no timeout)

language [python] Type of test to run. Base class supports 'python' only. Additional tests can be added by subclassing and defining new method _test_{language}.

****kwargs** Passed to language specific method _test_{language}

Testing proceeds by running a unit test function defined within the script from which the test method is called.

The unit test should contain at least one import statement to the generically named client file (the file name passed to 'expecting' when constructing the Project.)

The unit test should raise an error if the client's code is in error. For instance an assert statement can be used to check a result is returned correctly.

See examples/batch_testing.py for more details and examples.

class unicity.**Client**(*name*, *cohort=None*)
Class for individual client.

name [str] Name associated with client.

files [dict] Dictionary of File objects, keyed by file name, associated with the client's portfolio.

missing [list] Expected files missing from the client's portfolio.

When created by a Project object with an associated cohort file, Client objects are provisioned attributes corresponding to column information in the cohort file.

class unicity.**Comparison**(*loadfile*, *prior_project=None*)
A class for batch comparisons of code.

loadfile [str] File path to previously saved comparison object.

prior_project [unicity.Project (optional)] If comparison was constructed against a prior project, pass this object to read prior client information.

matrix [array-like] Distance matrix of similarities.

routine [str] Code unit for comparison in unicity string format.

prior_routine [str] Code unit for prior comparison in unicity string format.

save Write comparison data to file.

save(*savefile*)
Saves a precomputed comparison file.

savefile [str] Name of file to save similarity data.

class unicity.**PythonFile**(*filename*, *zipfile*)
Class containing information about Python file.

all_calls [list] All callables in file, in order of appearance.

all_keywords [list] All callables and reserved keywords in file, in order of appearance.

classes [dict] ClassInfo objects corresponding to classes defined in file.

filename [str] Name of file.

functions [dict] FunctionInfo objects corresponding to callables defined in file.

import_froms [list] List of [module, thing] import from statements in from {module} import {thing} format.

import_lines [list] Line numbers of import statements.

lns [list] Text of file.

reserved [dict] Frequency of reserved keywords.

class unicity.**FunctionInfo**(***kwargs*)

Class containing information about Python callable.

all_keywords [list] All callables and reserved keywords in file, in order of appearance.

docstring [str] Docstring for the function.

funcs [list] Names of functions called within this function.

import_froms [list] List of [module, thing] import from statements in from {module} import {thing} format.

lineno [list] First and last line number of function in file.

lns [list] Text of function.

methods [list] Names of methods called within the function.

name [str] Name of the funtion.

names [list] A python ast category I don't fully understand but useful for catching when the client invokes an Exception.

user_classes [list] Names of classes defined by the client and used within this function.

user_funcs [list] Names of functions defined by the client and used within this funcion.

class unicity.**ClassInfo**(***kwargs*)

Class containing information about Python class.

all_keywords [list] All callables and reserved keywords in file, in order of appearance.

base [str] Base class.

defs [list] Names of methods defined within the class.

docstring [string] Docstring for the class.

lineno: list First and last line number of class definition in file.

lns [list] Text of class.

methods [list] Names of methods called within the class.

name [str] Name of the class.

INDEX