UndefinedOffsetNine

Developer: Cortney Mood
Developer: Courtney Profera
Developer: Katie Vaughan
Project Manager: Ronald Zielaznicki

Developing a testing framework for:

FUZZYWUZZY

developed by seatgeek

Part 1

Introduction

The objective of this project is to create a testing framework for an open-source application of our choice. The project is being developed by Ronald Zielaznicki, Courtney Profera, Cortney Mood, and Katherine Vaughan; Computer Science students at the College of Charleston.

Chosen Project

FuzzyWuzzy is a string matching routine that depends on the difflib python library. Difflib gives classes and function for comparing sequences. In the case of FuzzyWuzzy, SequenceMatcher is used and the multiple string processings are condensed into a single operation, in order to compare two strings and produce a percentage of how alike the two strings are.

The code is originally located in Github
Github: https://github.com/seatgeek/fuzzywuzzy

While Difflib is the python library that it utilizes
Difflib: https://docs.python.org/2/library/difflib.html

And seakgeek gives a brief explanation as to how it all works
Explanation: http://chairnerd.seatgeek.com/fuzzywuzzy-fuzzy-string-matching-in-python/

Information on the Testing Framework

For our framework we will be testing the fuzz.py methods: ratio, partial_ratio, token_sort_ratio, and token_set_ratio, each of which uses a different method for comparing the strings.

Initial Installation and Working with FuzzyWuzzy

Environments:
VirtualBox with Ubuntu 14.04, python 2.7.6, installed using Git
Archlinux, python 2.7.4, installed using Git
VMware Player with Ubuntu 14.04, python 2.7.6, installed using Git

The initially installation:
Git clone
Cd into the file structure
Python setup.py install

Then, in python:
>>> from fuzzywuzzy import fuzz
>>> from fuzzywuzzy import process

Upon looking at the files:

fuzz.py contains the actual functions used and called to calculate the string compatibility.

Process.py finds the actual matches within the file system used, since the creators, seatgeek, are looking to match stings for tickets to different events.

String_processing.py processes the string input. This processing involves eliminating non-letters and non-numbers, stripping whitespace, and altering to upper or lower case letters.

Utils.py involves being able to validate the string to not be 0, ascii letters, and making sure the type is consistent to be processed and compared.

StringMatcher.py seems to be what pulls everything together.

python test_fuzzywuzzy.py was run and returned:

Ran 41 tests in 0.050s

OK

This seems more to check that the system is installed correctly and, since no errors were returned, that everything is working, although no printed output is available.

Then the tests presented in the readme were run

>>>Fuzz.ratio("this is a test", "this is a test!")
97
Readme says this should be 96

>>>Fuzz.partial_ratio("this is a test", "this is a test!")
100
As predicted in the readme

>>>Fuzz.ratio("fuzzy wuzzy was a bear", "wuzzy fuzzy was a bear")
91
Readme says 90

>>>Fuzz.token_set_ratio("fuzzy wuzzy was a bear", "wuzzy fuzzy was a bear")
100
Same as the readme

>>>Fuzz.token_sort_ratio("fuzzy was a bear", "fuzzy fuzzy was a bear")
84
Same as the readme

>>>Fuzz.token_set_ratio("fuzzy was a bear", "fuzzy fuzzy was a bear")
100
Same as readme

Part 4

Results of Testing

fuzzyWuzzy passed all test cases we passed to it using our testing framework. After developing 25 test cases for fuzzyWuzzy, we found that the project does compare strings for special cases, and for normal words without issue.

Tested Methods

With this project it was chosen that we remain within the realm of the fuzz.py file and the actual string comparisons. process.py allows for the user to compare strings but it pulls the best match from an array instead of giving the percentage of similarity between two strings. We felt that it would be easier to test between two strings because we are able to, as humans, see the comparison between the two strings instead of dealing with an array of strings. fuzz.py includes the methods ratio, partial_ratio, token_sort_ratio, and token_set_ratio. Each method goes about testing in a slightly different fashion, pulling from the difflib python library.

Ratio uses SequenceMatcher() from the difflib library, SequenceMatcher(isjunk, string1, string2) where isjunk is asking if the element should be ignored, usually set to None. Then difflib's ratio() is used to return the calculated similarity as a float.

Partial_ratio also uses SequenceMatcher() as well as get_matching_blocks(). This method has each block representing a sequence of matching characters in the string. So instead of comparing multiple bits of a long string to a short string to try to find if it is a match, the blocks will automatically compare the substring, using SequenceMatcher() and ratio() yet again to find the percentage.

token_sort_ratio calls _token_sort. This method allows for checking similarities between strings even if they are out of order. If a string is not in a proper order the similarity results will be low, even if the substrings are all included in each string. _token_sort splits the string and pulls the individual tokens so that they can be sorted and joined back together. It is then that partial_ratio and ratio are called on the strings.

partial_token_set_ratio is similar to token_sort_ratio although the tokens are split up and compared so that there can be a comparison of substrings specifically. By pulling the tokens and checking the difference, the method is able to find when a larger string may include a shorter string as part of a substring. Again, partial_ratio and ratio_are used to produce the actual percentages.

Tested Inputs

Not all inputs can be tested to see what kind of ratio they will produce as the possibilities are infinite. The initial set of tests compare a string with an empty string (Test1-Test5). Test6-Test10 are used to compare the special character ':' to a string to be sure that the testing framework does not break when this is done because within parseTestCase() in our framework, lines are split on ':'. The next set of test cases, Test11-Test20 compare strings that are exactly alike and strings that are approximately 50% alike. Test21-Test24 are used to compare strings that are completely different (0%). Test25 is used to compare strings that have 25% similarity. This is used to check that the comparisons are occurring correctly for "ideal" situations and what would be a typical use scenario.

Sample Output

Test Number	Requirement	Component	Method	Inputs	Driver	Expected Output	Actual Output	Pass/Fail
5	Compare null to String values	fuzz	token_sort_ratio	"","testing empty string"	fuzzTokenSortRatioDriver	0	0	Pass
13	Produce a 100	fuzz	token_set_ratio	"completely alike","completely alike"	fuzzTokenSetRatioDriver	100	100	Pass
7	Comparing special character : to a string	fuzz	ratio	":","String"	fuzzRatioDriver	0	0	Pass
4	Compare String to null values	fuzz	token_set_ratio	"Fun Stuff",""	fuzzTokenSetRatioDriver	0	0	Pass
10	Comparing special character : to a string	fuzz	token_sort_ratio	":","String"	fuzzTokenSortRatioDriver	0	0	Pass
3	Compare String to null values	fuzz	token_set_ratio	"","testing empty string"	fuzzTokenSetRatioDriver	0	0	Pass
2	Compare null to String values	fuzz	ratio	"","testing empty string"	fuzzRatioDriver	0	0	Pass
12	Produce a 100	fuzz	partial_ratio	"completely alike","completely alike"	fuzzPartialRatioDriver	100	100	Pass
19	Compare two strings	fuzz	token_set_ratio	"Hello","HelloWorld"	fuzzTokenSetRatioDriver	50	67	Fail
14	Produce a 100	fuzz	token_sort_ratio	"completely alike","completely alike"	fuzzTokenSortRatioDriver	100	100	Pass
16	Comparing two identical strings	fuzz	partial_ratio	"Hello world","Hello world"	fuzzPartialRatioDriver	100	100	Pass
15	Produce a 50	fuzz	ratio	"abc123","abcdef"	fuzzRatioDriver	50	50	Pass
9	Comparing special character : to a string	fuzz	token_set_ratio	":","String"	fuzzTokenSetRatioDriver	0	0	Pass
8	Comparing special character : to a string	fuzz	token_set_ratio	":","String"	fuzzTokenSetRatioDriver	0	0	Pass
11	Produce a 100	fuzz	ratio	"completely alike","completely alike"	fuzzRatioDriver	100	100	Pass
17	Compare two strings	fuzz	ratio	"Hello","HelloWorld"	fuzzRatioDriver	50	67	Fail
20	Compare null to String values	fuzz	token_sort_ratio	"Hello","HelloWorld"	fuzzTokenSortRatioDriver	50	67	Fail
1	Compare null to String values	fuzz	partial_ratio	"","testing empty string"	fuzzPartialRatioDriver	0	0	Pass
18	Compare two strings	fuzz	token_set_ratio	"Hello","HelloWorld"	fuzzTokenSetRatioDriver	50	67	Fail
6	Comparing special character : to a string	fuzz	partial_ratio	":","String"	fuzzPartialRatioDriver	0	0	Pass