FUZZYWUZZY TEST REPORT

Deepthi S Kumar
CS 562 – Winter 2016
Oregon State University

Software under test
FuzzyWuzzy (https://github.com/seatgeek/fuzzywuzzy)

It is a pure python library offering the following four string matching techniques that are tested.

- Ratio - Encapsulates existing method 'ratio()' from python's difflib or using Levenshtein distance, if the package is imported.
- Partial Ratio - Computes the best partial match between 2 strings of noticeably different lengths. If the shorter string is length m, and the longer string is length n, it returns the score of the best matching length-m substring.
- Token Sort Ratio - Similarity between 2 strings which are out of order in their constructions. As the name suggests, this technique involves tokenizing the string, sorting the tokens in alphabetical order, form a string of the sorted token and then compute simple ratio of the sorted string.
- Token Set Ratio - Similar to token sort ratio, but provides better result when the difference in string lengths is quite big.

Apart from the above-mentioned functions, the following functions that use the above mentioned functions are be tested.

- extract() - Extracts best matches of string from a list of strings based on the limit specified. The default technique used here is weighted Ratio but can be changed by passing relevant arguments.
- extractOne() - Same as extract() but returns exactly one string that has highest match score from the list of strings.
- extractBests() - Returns n best matches from a list with weighted Ratio() above a specified threshold (by default 0).

Scenarios
Following are the scenarios tested using TSTL for all the four functions viz., ratio(), partial_ratio(), token_set_ratio(), token_sort_ratio().

1. Similarity between two empty strings is 0
2. Similarity between two equal strings is 100
3. Swap the arguments and check if the similarity is still the same (1 bug found)
4. fuzz.ratio() should behave in the same way as ratio() in difflib/levenshtein library
5. Partial ratio of two strings containing a common word is 100
6. Token sort ratio of 2 strings with same set of words arranged differently is 100
7. Token set ratio of 2 strings is 100 if second string contains at least one of the words from the first string
8. Sanity check on all the functions
9. extractOne(), extractBests(), extract() using list of words containing at least one 100% matching string

Bug Report

Following are the 3 bugs that the tester was able to find.

1. Similarity ratio is different when the strings being compared are passed in reverse order. This occurred for partial_ratio() and token_sort_ratio().

```
FINAL VERSION OF TEST, WITH LOGGED REPLAY:

ascii0 = 85                                                    # STEP 0

Coverage.py warning: No data was collected.

str20 = chr(ascii0)                                            # STEP 1

Coverage.py warning: No data was collected.

ascii0 = 66                                                    # STEP 2

Coverage.py warning: No data was collected.

str20 += chr(ascii0)                                           # STEP 3

Coverage.py warning: No data was collected.

str10 = chr(ascii0)                                            # STEP 4

Coverage.py warning: No data was collected.

str20 += ' '                                                   # STEP 5

Coverage.py warning: No data was collected.

str10 += ' '                                                   # STEP 6

Coverage.py warning: No data was collected.

str10 += chr(ascii0)                                           # STEP 7

Coverage.py warning: No data was collected.

str10 += chr(ascii0)                                           # STEP 8

Coverage.py warning: No data was collected.

ascii0 = 73                                                    # STEP 9

Coverage.py warning: No data was collected.

str20 += chr(ascii0)                                           # STEP 10
```

```
Coverage.py warning: No data was collected.

assert (fuzz.token_set_ratio(str10,str20) ==
fuzz.token_set_ratio(str20,str10))  # STEP 11

ERROR: (<type 'exceptions.AssertionError'>, AssertionError(),
<traceback object at 0x7f82d0733cb0>)

TRACEBACK:

  File "/home/eecs/Documents/CS562/Project/sut.py", line 1763, in
safely

    act[2]()

  File "/home/eecs/Documents/CS562/Project/sut.py", line 1069, in
act45

    assert (fuzz.token_set_ratio(self.p_str1[0],self.p_str2[0]) ==
fuzz.token_set_ratio(self.p_str2[0],self.p_str1[0]))
```

Test for token_sort_ratio()

```
FINAL VERSION OF TEST, WITH LOGGED REPLAY:

ascii0 = 72                                                    # STEP 0

str20 = chr(ascii0)                                            # STEP 1

str10 = chr(ascii0)                                            # STEP 2

ascii0 = 87                                                    # STEP 3

str20 += ' '                                                   # STEP 4

str10 += chr(ascii0)                                           # STEP 5

ascii0 = 90                                                    # STEP 6

str10 += ' '                                                   # STEP 7

str10 += chr(ascii0)                                           # STEP 8

ascii0 = 87                                                    # STEP 9

str10 += chr(ascii0)                                          # STEP 10

str20 += chr(ascii0)                                          # STEP 11

assert (fuzz.token_sort_ratio(str10,str20) ==
```

```
fuzz.token_sort_ratio(str20,str10))                          # STEP 12

ERROR: (<type 'exceptions.AssertionError'>, AssertionError(),
<traceback object at 0x7f6c709ee320>)

TRACEBACK:

  File "/home/eecs/Documents/CS562/Project/sut.py", line 1634, in
safely

    act[2]()

  File "/home/eecs/Documents/CS562/Project/sut.py", line 973, in
act41

    assert (fuzz.token_sort_ratio(self.p_str1[0],self.p_str2[0]) ==
fuzz.token_sort_ratio(self.p_str2[0],self.p_str1[0]))
```

The above 2 bugs were found while using diffLib. Same functionality was broken when using Levenshtein library. Following is the test.

```
FINAL VERSION OF TEST, WITH LOGGED REPLAY:

letter0 = 'C'                                                # STEP 0

str20 = letter0                                              # STEP 1

letter0 = 'H'                                                # STEP 2

str10 = letter0                                              # STEP 3

str20 += ' '                                                 # STEP 4

str20 += letter0                                             # STEP 5

letter0 = 'C'                                                # STEP 6

str10 += letter0                                             # STEP 7

str10 += ' '                                                 # STEP 8

assertPartialRatioSwap(str10,str20)                          # STEP 9

Partial Ratio  HC  , C H

ERROR: (<type 'exceptions.AssertionError'>, AssertionError(),
<traceback object at 0x7f07b3d08200>)

TRACEBACK:
```

```
  File "/home/eecs/Documents/CS562/Project/sut.py", line 6399, in
safely act[2]()

  File "/home/eecs/Documents/CS562/Project/sut.py", line 1619, in
act66 assertPartialRatioSwap(self.p_str1[0],self.p_str2[0])

  File "/home/eecs/Documents/CS562/Project/sut.py", line 29, in
assertPartialRatioSwap assert (fuzz.partial_ratio(str1,str2) ==
fuzz.partial_ratio(str2,str1))
```

The test basically is to check if all the different ratio functions return the same value even when arguments are swapped.

For example, in the above test partial_ratio("QJ","JJ") is different than partial_ratio("JJ","QJ"). However, the ratio("QJ","JJ") = ratio("JJ","QJ"). The similarity of two strings should be the same irrespective of the order in which they are passed to the function. Also, edit distance is symmetric i.e., d(a,b) = d(b,a).

TSTL

The language is simple and easy to use. It allows using the language of the software being tested which is helpful in many cases. However, in depth knowledge about the SUT language for generating test harnesses is not required.
Using tstl to generate random tests has been effective for testing this library since different permutations and combinations of characters are needed.

Some more documentation on tstl apart from the demo and research papers will be helpful.

Coverage report

Name	Stmts	Miss	Branch	BrPart	Cover
fuzzywuzzy/StringMatcher.py	52	37	18	0	29%
fuzzywuzzy/fuzz.py	118	35	30	5	72%
fuzzywuzzy/string_processing.py	14	13	2	0	6%
fuzzywuzzy/process.py	54	17	30	7	71%
fuzzywuzzy/utils.py	55	28	24	5	53%

Most of the statements and branches have been covered. The statements missed are mostly comments and other annotations.