jellyfish Documentation

Release 0.8.3

James Turk

Mar 11, 2021

Contents

CHAPTER 1

Overview

jellyfish is a library of functions for approximate and phonetic matching of strings.

Source code is available on GitHub

The library provides implementations of the following algorithms:

1.1 Phonetic Encoding

These algorithms convert a string to a normalized phonetic encoding, converting a word to a representation of its pronunciation. Each takes a single string and returns a coded representation.

1.1.1 American Soundex

soundex(*s*)
Calculate the American Soundex of the string s.

Soundex is an algorithm to convert a word (typically a name) to a four digit code in the form 'A123' where 'A' is the first letter of the name and the digits represent similar sounds.

For example `soundex('Ann') == soundex('Anne') == 'A500'` and `soundex('Rupert') == soundex('Robert') == 'R163'`.

See the Soundex article at Wikipedia for more details.

1.1.2 Metaphone

metaphone(*s*)
Calculate the metaphone code for the string s.

The metaphone algorithm was designed as an improvement on Soundex. It transforms a word into a string consisting of '0BFHJKLMNPRSTWXY' where '0' is pronounced 'th' and 'X' is a '[sc]h' sound.

For example `metaphone('Klumpz') == metaphone('Clumps') == 'KLMPS'`.

See the Metaphone article at Wikipedia for more details.

1.1.3 NYSIIS

nysiis(*s*)
Calculate the NYSIIS code for the string s.

The NYSIIS algorithm is an algorithm developed by the New York State Identification and Intelligence System. It transforms a word into a phonetic code. Like soundex and metaphone it is primarily intended for use on names (as they would be pronounced in English).

For example `nysiis('John') == nysiis('Jan') == JAN`.

See the NYSIIS article at Wikipedia for more details.

1.1.4 Match Rating Approach (codex)

match_rating_codex(*s*)
Calculate the match rating approach value (also called PNI) for the string s.

The Match rating approach algorithm is an algorithm for determining whether or not two names are pronounced similarly. The algorithm consists of an encoding function (similar to soundex or nysiis) which is implemented here as well as *match_rating_comparison()* which does the actual comparison.

See the Match Rating Approach article at Wikipedia for more details.

1.2 Stemming

1.2.1 Porter Stemmer

porter_stem(*s*)
Reduce the string s to its stem using the common Porter stemmer.

Stemming is the process of reducing a word to its root form, for example 'stemmed' to 'stem'.

Martin Porter's algorithm is a common algorithm used for stemming that works for many purposes.

See the official homepage for the Porter Stemming Algorithm for more details.

1.3 String Comparison

These methods are all measures of the difference (aka *edit distance*) between two strings.

1.3.1 Levenshtein Distance

levenshtein_distance(*s1*, *s2*)
Compute the Levenshtein distance between s1 and s2.

Levenshtein distance represents the number of insertions, deletions, and substitutions required to change one word to another.

For example: `levenshtein_distance('berne', 'born') == 2` representing the transformation of the first e to o and the deletion of the second e.

See the Levenshtein distance article at Wikipedia for more details.

1.3.2 Damerau-Levenshtein Distance

damerau_levenshtein_distance(*s1*, *s2*)
Compute the Damerau-Levenshtein distance between s1 and s2.

A modification of Levenshtein distance, Damerau-Levenshtein distance counts transpositions (such as ifsh for fish) as a single edit.

Where `levenshtein_distance('fish', 'ifsh') == 2` as it would require a deletion and an insertion, though `damerau_levenshtein_distance('fish', 'ifsh') == 1` as this counts as a transposition.

See the Damerau-Levenshtein distance article at Wikipedia for more details.

1.3.3 Hamming Distance

hamming_distance(*s1*, *s2*)
Compute the Hamming distance between s1 and s2.

Hamming distance is the measure of the number of characters that differ between two strings.

Typically Hamming distance is undefined when strings are of different length, but this implementation considers extra characters as differing. For example `hamming_distance('abc', 'abcd') == 1`.

See the Hamming distance article at Wikipedia for more details.

1.3.4 Jaro Similarity

jaro_similarity(*s1*, *s2*)
Compute the Jaro similarity between s1 and s2.

Jaro distance is a string-edit distance that gives a floating point response in [0,1] where 0 represents two completely dissimilar strings and 1 represents identical strings.

> **Warning:** Prior to 0.8.1 this function was named jaro_distance. That name is still available, but is no longer recommended. It will be replaced in 1.0 with a correct version.

1.3.5 Jaro-Winkler Similarity

jaro_winkler_similarity(*s1*, *s2*)
Compute the Jaro-Winkler distance between s1 and s2.

Jaro-Winkler is a modification/improvement to Jaro distance, like Jaro it gives a floating point response in [0,1] where 0 represents two completely dissimilar strings and 1 represents identical strings.

Warning: Prior to 0.8.1 this function was named jaro_winkler. That name is still available, but is no longer recommended. It will be replaced in 1.0 with a correct version.

See the Jaro-Winkler distance article at Wikipedia for more details.

1.3.6 Match Rating Approach (comparison)

match_rating_comparison(*s1*, *s2*)

Compare s1 and s2 using the match rating approach algorithm, returns `True` if strings are considered equivalent or `False` if not. Can also return `None` if s1 and s2 are not comparable (length differs by more than 3).

The Match rating approach algorithm is an algorithm for determining whether or not two names are pronounced similarly. Strings are first encoded using *match_rating_codex()* then compared according to the MRA algorithm.

See the Match Rating Approach article at Wikipedia for more details.

1.4 Changelog

1.4.1 0.8.3 - 11 March 2021

- build changes
- include OSX and Windows wheels

1.4.2 0.8.2 - 21 May 2020

- fix jaro_winkler/jaro_winkler_similarity mix-up
- deprecate jaro_distance in favor of jaro_similarity backwards compatible shim left in place, will be removed in 1.0
- (note: 0.8.1 was a broken release without proper C libraries)

1.4.3 0.8.0 - 21 May 2020

- rename jaro_winkler to jaro_winkler_similarity to match other functions backwards compatible shim added, but will be removed in 1.0
- fix soundex bug with W/H cases, #83
- fix metaphone bug with WH prefix, #108
- fix C match rating codex bug with duplicate letters, #121
- fix metaphone bug with leading vowels and 'kn' pair, #123
- fix Python jaro_winkler bug #124
- fix Python 3.9 deprecation warning
- add manylinux wheels

1.4.4 0.7.2 - 5 June 2019

- fix CJellyfish damerau_levenshtein w/ unicode, thanks to immerrr
- fix final H in NYSIIS
- fix issue w/ trailing W in metaphone

1.4.5 0.7.1 - 10 January 2019

- restrict install to Python >= 3.4

1.4.6 0.7.0 - 10 January 2019

- drop Python 2 compatibility & legacy code
- add bugfix for NYSIIS for words starting with PF

1.4.7 0.6.1 - April 16 2018

- fixed wheel release issue

1.4.8 0.6.0 - April 7 2018

- fix quite a few bugs & differences between C/Py implementations
- add wagner-fischer testdata
- uppercase soundex result
- better error handling in nysiis, soundex, and jaro

1.4.9 0.5.6 - June 23 2016

- bugfix for metaphone & soundex raising unexpected TypeErrors on Windows (#54)

1.4.10 0.5.5 - June 21 2016

- bugfix for metaphone WH case

1.4.11 0.5.4 - May 13 2016

- bugfix for C version of damerau_levenshtein thanks to Tyler Sellon

1.4.12 0.5.3 - March 15 2016

- style/packaging changes

1.4.13 0.5.2 - February 3 2016

- testing fixes for Python 3.5
- bugfix for Metaphone w/ silent H thanks to Jeremy Carbaugh

1.4.14 0.5.1 - July 12 2015

- bugfixes for NYSIIS
- bugfixes for metaphone
- bugfix for C version of jaro_winkler

1.4.15 0.5.0 - April 23 2015

- consistent unicode behavior, all functions take unicode and reject bytes on Py2 and 3, C and Python
- parametrize tests
- Windows compiler support

1.4.16 0.4.0 - March 27 2015

- tons of new tests
- documentation
- split out cjellyfish
- test all w/ unicode and plenty of fixes to accommodate
- 100% test coverage

1.4.17 0.3.4 - February 4 2015

- fix segfaults and memory leaks via Danrich Parrol

1.4.18 0.3.3 - November 20 2014

- fix bugs in damerau and NYSIIS

1.4.19 0.3.2 - August 11 2014

- fix for jaro-winkler from David McKean
- more packaging fixes

1.4.20 0.3.1 - July 16 2014

- packaging fix for C/Python alternative

1.4.21 0.3.0 - July 15 2014

- python alternatives where C isn't available

1.4.22 0.2.2 - March 14 2014

- testing fixes
- assorted bugfixes in NYSIIS

1.4.23 0.2.0 - January 26 2012

- incorporate some speed changes from Peter Scott
- segfault bugfixes.

1.4.24 0.1.2 - September 16 2010

- initial working release

CHAPTER 2

Implementation

Each algorithm has C and Python implementations.

On a typical CPython install the C implementation will be used. The Python versions are available for PyPy and systems where compiling the CPython extension is not possible.

To explicitly use a specific implementation, refer to the appropriate module:

```
import jellyfish._jellyfish as pyjellyfish
import jellyfish.cjellyfish as cjellyfish
```

If you've already imported jellyfish and are not sure what implementation you are using, you can check by querying `jellyfish.library`:

```
if jellyfish.library == 'Python':
    # Python implementation
elif jellyfish.library == 'C':
    # C implementation
```

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Index