Abydos Documentation

Release 0.5.0

Christopher C. Little

Jan 10, 2020

CONTENTS:

CHAPTER

ONE

INTRODUCTION

1.1 Abydos

Abydos NLP/IR library
Copyright 2014-2020 by Christopher C. Little

Abydos is a library of phonetic algorithms, string distance measures & metrics, stemmers, and string fingerprinters.

1.2 Installation

Required libraries:

- NumPy
- deprecation

Optional libraries (all available on PyPI, some available on conda or conda-forge):

- SyllabiPy
- NLTK
- PyLZSS

- paq

To install Abydos (master) from Github source:

```
git clone https://github.com/chrislit/abydos.git --recursive
cd abydos
python setup install
```

If your default python command calls Python 2.7 but you want to install for Python 3, you may instead need to call:

```
python3 setup install
```

To install Abydos (latest release) from PyPI using pip:

```
pip install abydos
```

To install from conda-forge:

```
conda install abydos
```

It should run on Python 3.5-3.8.

1.3 Testing & Contributing

To run the whole test-suite just call tox:

```
tox
```

The tox setup has the following environments: black, py37, doctest, regression, fuzz, pylint, pydocstyle, flake8, doc8, docs, sloccount, badges, & build. So if you only want to generate documentation (in HTML, EPUB, & PDF formats), just call:

```
tox -e docs
```

In order to only run & generate Flake8 reports, call:

```
tox -e flake8
```

Contributions such as bug reports, PRs, suggestions, desired new features, etc. are welcome through Github Issues & Pull requests.

1.4 Badges

The project's main page has quite a few badges, some seemingly redundant, and a bit of explanation is perhaps warranted.

- CI & Test Status
  - Travis-CI is the primary CI used for Linux CI of all supported Python platforms (2.7-3.8-dev). Only the tests in the tests directory are run.
  - CircleCI runs only the Python 3.6 tests on Linux and is used for quick tests of each commit.
  - Azure DevOps is used to perform tests on Linux, MaxOS, and Windows on Python 2.7, 3.5, 3.6, & 3.7 using pytest.

  - Semaphore is used to run the tests in the tests directory, doctests, regression tests, and fuzz tests.
  - Coveralls is used to track test coverage.
- Code Quality (some may be removed at a later date)
  - Code Climate is used to check maintainability, but mostly just complains about McCabe complexity.
  - Scrutinizer is used to check complexity and compliance with best practices.
  - Codacy is used to check code style, security issues, etc.
  - CodeFactor is used to track hotspot files in need of attention.
- Dependencies
  - Requires.io tracks whether Abydos can be used with the most recent releases of its dependencies.
  - Snyk tracks whether there are security vulnerabilities in any dependencies.
  - Pyup.io tracks updates and security vulnerabilities in dependencies.
  - CII Best Practices identifies compliance with Core Infrastructure Initiative best practices.
- Local Analysis
  - Pylint score, run locally
  - flake8 score, run locally, should be 0.
  - pydocstyle score, run locally, should be 0.
  - SLOCCount shows the total source lines of code.
  - Black code style signals that Black is used for code styling.
- Usage
  - Read the Docs hosts Abydos documentation online.
  - Binder provides an online notebook environment for the demo notebooks.
  - GPL v3+ is the license used by Abydos.
  - Libraries.io assigns a SourceRank to indicate project quality and popularity.
  - zenodo publishes the DOI and citation information for Abydos.
- Contribution
  - OpenHub tracks project activity and KLOC and estimates project value.
  - The commit activity shows commit rate.
  - The issues badge indicates the number of issues closed.
  - The GitHub stars badge indicates the number of stars received.
- PyPI
  - PyPI hosts the pip installable packages. The pypi badge indicates the most recent pip installable version.
  - The downloads badge indicates the number of downloads from PyPI per month.
  - The python badge indicates the versions of Python that are supported.
- conda-forge
  - conda-forge hosts the conda installable packages. The conda-forge badge indicates the most recent conda installable version.

- The downloads badge indicates the number of downloads from conda-forge.
- The platform badge indicates that Abydos is a pure Python project, without platform-specific builds.

1.5 License

Abydos is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <https://www.gnu.org/licenses/gpl.txt>.

CHAPTER

TWO

FAQ

2.1 Why is the library licensed under GPL3+? Can you change the license?

GPL3 is the only license compatible with all of the various parts of Abydos that have been ported to Python from other languages. For example, the Beider-Morse Phonetic Matching algorithm implementation included in Abydos was ported from their reference implementation in PHP, which is itself licensed under GPL3.

Accordingly, it's not possible to change to a different license without removing parts of the library. However, if you have a need for a specific part of the library and can't use GPL3+ code, contact us and we may be able to provide it separately or can give guidance on its underlying licensing status.

2.2 What is the purpose of this library?

A. Abydos is intended to facilitate any manner of string transformation and comparison might be useful for string matching or record linkage. The two most significant parts of the library are string distance/similarity measures and phonetic algorithms/string fingerprint algorithms, but a large collection of tokenizers, corpus classes, compression algorithms, & phonetics functions support these and afford greater customization.

2.3 Can you add this new feature?

Maybe. Open an issue at https://github.com/chrislit/abydos/issues and propose your new feature.

Additional string distance/similarity measures, phonetic algorithms, string fingerprint algorithms, and string tokenizers will certainly be added if possible -- but it's helpful to point them out since we may not be aware of them.

2.4 Can I contribute to the project?

Absolutely. You can take on an unclaimed issue, report bugs, add new classes, or whatever piques your interest. You are welcome to open an issue at https://github.com/chrislit/abydos/issues proposing what you'd like to work on, or you can submit a pull request if you have something ready to contribute to the repository.

2.5 Will you add Metaphone 3?

No. Although Lawrence Philips (author of Metaphone, Double Metaphone, and Metaphone 3) released Metaphone 3 version 2.1.3 under the BSD 3-clause license as part of Google Refine, which became OpenRefine (https://github.com/OpenRefine/OpenRefine/blob/master/main/src/com/google/refine/clustering/binning/Metaphone3.java), he doesn't want that code used for ports to other languages or used in any way outside of OpenRefine. In accordance with his wishes, no one has released Metaphone 3 ports to other languages or included it other libraries.

2.6 Why have you included algorithm X when it is already a part of NLTK/SciPy/...?

Abydos is a collection of algorithms with common class & function interfaces and options. So, while NLTK has Levenshtein & Jaccard string similarity measures, they don't allow for tunable edit costs or using the tokenizer of your choice.

2.7 Are there similar projects for languages other than Python?

Yes, there are libraries such as:

- Talisman for JavaScript
- Phonics for R (phonetic algorithms)
- stringmetric for Scala

2.8 What is the process for adding a new class to the library?

The process of adding a new class follows roughly the following steps:

- Discover that a new (unimplemented) measure/algorithm/method exists
- Locate the original source of the algorithm (a journal article, a reference implementation, etc.). And save the reference to it in docs/abydos.bib.
  - If the original source cannot be located for reference, use an adequate secondary source and add its reference info to docs/abydos.bib.
- Implement the class based on its description/reference implementation.
- Create a test class and add all examples and test cases from the original source. Add other reliable test cases from other sources, if they are available.
- Ensure that the class passes all test cases.
- Add test cases, as necessary, until test coverage reaches 100%, or as close to 100% as possible.

2.9 Are these really Frequently Asked Questions?

No. Most of these questions have never been explicitly asked.

CHAPTER

THREE

ABYDOS

3.1 abydos package

abydos.

Abydos NLP/IR library by Christopher C. Little

There are nine major packages that make up Abydos:

- *compression* for string compression classes
- *corpus* for document corpus classes
- *distance* for string distance measure & metric classes
- *fingerprint* for string fingerprint classes
- *phones* for functions relating to phones and phonemes
- *phonetic* for phonetic algorithm classes
- *stats* for statistical functions and a confusion table class
- *stemmer* for stemming classes
- *tokenizer* for tokenizer classes

Classes with each package have consistent method names, as discussed below. A tenth package, *util*, contains functions not intended for end-user use.

3.1.1 Subpackages

3.1.1.1 abydos.compression package

abydos.compression.

The compression package defines compression and compression-related functions for use within Abydos, including implementations of the following:

- *Arithmetic* for arithmetic coding
- *BWT* for Burrows-Wheeler Transform
- *RLE* for Run-Length Encoding

Each class exposes `encode` and `decode` methods for performing and reversing its encoding. For example, the Burrows-Wheeler Transform can be performed by creating a *BWT* object and then calling *BWT.encode()* on a string:

```
>>> bwt = BWT()
>>> bwt.encode('^BANANA')
'ANNB^AA\x00'
```

---

class abydos.compression.**Arithmetic**(*text=None*)

Bases: object

Arithmetic Coder.

This is based on Andrew Dalke's public domain implementation [Dal05]. It has been ported to use the fractions.Fraction class.

New in version 0.3.6.

Initialize arithmetic coder object.

Parameters **text** (*str*) -- The training text

New in version 0.3.6.

decode(*longval*, *nbits*)

Decode the number to a string using the given statistics.

Parameters

- **longval** (*int*) -- The first part of an encoded tuple from encode
- **nbits** (*int*) -- The second part of an encoded tuple from encode

Returns The arithmetically decoded text

Return type str

Example

```
>>> ac = Arithmetic('the quick brown fox jumped over the lazy dog')
>>> ac.decode(16720586181, 34)
'align'
```

New in version 0.1.0.

Changed in version 0.3.6: Encapsulated in class

encode(*text*)

Encode a text using arithmetic coding.

Text and the 0-order probability statistics -> longval, nbits

The encoded number is Fraction(longval, 2**nbits)

Parameters **text** (*str*) -- A string to encode

Returns The arithmetically coded text

Return type tuple

Example

```
>>> ac = Arithmetic('the quick brown fox jumped over the lazy dog')
>>> ac.encode('align')
(16720586181, 34)
```

New in version 0.1.0.

Changed in version 0.3.6: Encapsulated in class

get_probs()
Return the probs dictionary.

Returns The dictionary of probabilities

Return type dict

New in version 0.3.6.

set_probs(*probs*)
Set the probs dictionary.

Parameters **probs** (*dict*) -- The dictionary of probabilities

New in version 0.3.6.

train(*text*)
Generate a probability dict from the provided text.

Text to 0-order probability statistics as a dict

Parameters **text** (*str*) -- The text data over which to calculate probability statistics. This must not contain the NUL (0x00) character because that is used to indicate the end of data.

Example

```
>>> ac = Arithmetic()
>>> ac.train('the quick brown fox jumped over the lazy dog')
>>> ac.get_probs()
{' ': (Fraction(0, 1), Fraction(8, 45)),
 'o': (Fraction(8, 45), Fraction(4, 15)),
 'e': (Fraction(4, 15), Fraction(16, 45)),
 'u': (Fraction(16, 45), Fraction(2, 5)),
 't': (Fraction(2, 5), Fraction(4, 9)),
 'r': (Fraction(4, 9), Fraction(22, 45)),
 'h': (Fraction(22, 45), Fraction(8, 15)),
 'd': (Fraction(8, 15), Fraction(26, 45)),
 'z': (Fraction(26, 45), Fraction(3, 5)),
 'y': (Fraction(3, 5), Fraction(28, 45)),
 'x': (Fraction(28, 45), Fraction(29, 45)),
 'w': (Fraction(29, 45), Fraction(2, 3)),
 'v': (Fraction(2, 3), Fraction(31, 45)),
 'q': (Fraction(31, 45), Fraction(32, 45)),
 'p': (Fraction(32, 45), Fraction(11, 15)),
 'n': (Fraction(11, 15), Fraction(34, 45)),
 'm': (Fraction(34, 45), Fraction(7, 9)),
 'l': (Fraction(7, 9), Fraction(4, 5)),
 'k': (Fraction(4, 5), Fraction(37, 45)),
 'j': (Fraction(37, 45), Fraction(38, 45)),
```

(continues on next page)

(continued from previous page)

```
 'i': (Fraction(38, 45), Fraction(13, 15)),
 'g': (Fraction(13, 15), Fraction(8, 9)),
 'f': (Fraction(8, 9), Fraction(41, 45)),
 'c': (Fraction(41, 45), Fraction(14, 15)),
 'b': (Fraction(14, 15), Fraction(43, 45)),
 'a': (Fraction(43, 45), Fraction(44, 45)),
 '\x00': (Fraction(44, 45), Fraction(1, 1))}
```

New in version 0.1.0.

Changed in version 0.3.6: Encapsulated in class

abydos.compression.**ac_decode**(*longval*, *nbits*, *probs*)

Decode the number to a string using the given statistics.

This is a wrapper for *Arithmetic.decode()*.

Parameters

- **longval** (*int*) -- The first part of an encoded tuple from ac_encode
- **nbits** (*int*) -- The second part of an encoded tuple from ac_encode
- **probs** (*dict*) -- A probability statistics dictionary generated by *Arithmetic.train()*

Returns The arithmetically decoded text

Return type str

Example

```
>>> pr = ac_train('the quick brown fox jumped over the lazy dog')
>>> ac_decode(16720586181, 34, pr)
'align'
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the Arithmetic.decode method instead.

abydos.compression.**ac_encode**(*text*, *probs*)

Encode a text using arithmetic coding with the provided probabilities.

This is a wrapper for *Arithmetic.encode()*.

Parameters

- **text** (*str*) -- A string to encode
- **probs** (*dict*) -- A probability statistics dictionary generated by *Arithmetic.train()*

Returns The arithmetically coded text

Return type tuple

Example

```
>>> pr = ac_train('the quick brown fox jumped over the lazy dog')
>>> ac_encode('align', pr)
(16720586181, 34)
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the Arithmetic.encode method instead.

abydos.compression.**ac_train**(*text*)

Generate a probability dict from the provided text.

This is a wrapper for *Arithmetic.train()*.

Parameters **text** (*str*) -- The text data over which to calculate probability statistics. This must not contain the NUL (0x00) character because that's used to indicate the end of data.

Returns A probability dict

Return type dict

Example

```
>>> ac_train('the quick brown fox jumped over the lazy dog')
{' ': (Fraction(0, 1), Fraction(8, 45)),
 'o': (Fraction(8, 45), Fraction(4, 15)),
 'e': (Fraction(4, 15), Fraction(16, 45)),
 'u': (Fraction(16, 45), Fraction(2, 5)),
 't': (Fraction(2, 5), Fraction(4, 9)),
 'r': (Fraction(4, 9), Fraction(22, 45)),
 'h': (Fraction(22, 45), Fraction(8, 15)),
 'd': (Fraction(8, 15), Fraction(26, 45)),
 'z': (Fraction(26, 45), Fraction(3, 5)),
 'y': (Fraction(3, 5), Fraction(28, 45)),
 'x': (Fraction(28, 45), Fraction(29, 45)),
 'w': (Fraction(29, 45), Fraction(2, 3)),
 'v': (Fraction(2, 3), Fraction(31, 45)),
 'q': (Fraction(31, 45), Fraction(32, 45)),
 'p': (Fraction(32, 45), Fraction(11, 15)),
 'n': (Fraction(11, 15), Fraction(34, 45)),
 'm': (Fraction(34, 45), Fraction(7, 9)),
 'l': (Fraction(7, 9), Fraction(4, 5)),
 'k': (Fraction(4, 5), Fraction(37, 45)),
 'j': (Fraction(37, 45), Fraction(38, 45)),
 'i': (Fraction(38, 45), Fraction(13, 15)),
 'g': (Fraction(13, 15), Fraction(8, 9)),
 'f': (Fraction(8, 9), Fraction(41, 45)),
 'c': (Fraction(41, 45), Fraction(14, 15)),
 'b': (Fraction(14, 15), Fraction(43, 45)),
 'a': (Fraction(43, 45), Fraction(44, 45)),
 '\x00': (Fraction(44, 45), Fraction(1, 1))}
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the Arithmetic.train method instead.

class abydos.compression.**BWT**(*terminator='x00'*)

Bases: object

Burrows-Wheeler Transform.

The Burrows-Wheeler transform is an attempt at placing similar characters together to improve compression. Cf. [BW94].

New in version 0.3.6.

Initialize BWT instance.

Parameters **terminator** (*str*) -- A character added to signal the end of the string

New in version 0.4.0.

decode(*code*)

Return a word decoded from BWT form.

Parameters

- **code** (*str*) -- The word to transform from BWT form
- **terminator** (*str*) -- A character added to signal the end of the string

Returns Word decoded by BWT

Return type str

Raises **ValueError** -- Specified terminator absent from code.

Examples

```
>>> bwt = BWT()
>>> bwt.decode('n\x00ilag')
'align'
>>> bwt.decode('annb\x00aa')
'banana'
```

```
>>> bwt = BWT('@')
>>> bwt.decode('annb@aa')
'banana'
```

New in version 0.1.0.

Changed in version 0.3.6: Encapsulated in class

encode(*word*)

Return the Burrows-Wheeler transformed form of a word.

Parameters **word** (*str*) -- The word to transform using BWT

Returns Word encoded by BWT

Return type str

Raises **ValueError** -- Specified terminator absent from code.

Examples

```
>>> bwt = BWT()
>>> bwt.encode('align')
'n\x00ilag'
>>> bwt.encode('banana')
'annb\x00aa'
```

```
>>> bwt = BWT('@')
>>> bwt.encode('banana')
'annb@aa'
```

New in version 0.1.0.

Changed in version 0.3.6: Encapsulated in class

abydos.compression.**bwt_decode**(*code*, *terminator='\x00'*)

Return a word decoded from BWT form.

This is a wrapper for *BWT.decode()*.

Parameters

- **code** (*str*) -- The word to transform from BWT form
- **terminator** (*str*) -- A character added to signal the end of the string

Returns Word decoded by BWT

Return type str

Examples

```
>>> bwt_decode('n\x00ilag')
'align'
>>> bwt_decode('annb\x00aa')
'banana'
>>> bwt_decode('annb@aa', '@')
'banana'
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the BWT.decode method instead.

abydos.compression.**bwt_encode**(*word*, *terminator='\x00'*)

Return the Burrows-Wheeler transformed form of a word.

This is a wrapper for *BWT.encode()*.

Parameters

- **word** (*str*) -- The word to transform using BWT
- **terminator** (*str*) -- A character added to signal the end of the string

Returns Word encoded by BWT

Return type str

Examples

```
>>> bwt_encode('align')
'n\x00ilag'
>>> bwt_encode('banana')
'annb\x00aa'
>>> bwt_encode('banana', '@')
'annb@aa'
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the BWT.encode method instead.

class abydos.compression.**RLE**

Bases: object

Run-Length Encoding.

Cf. [RC67].

Based on http://rosettacode.org/wiki/Run-length_encoding#Python [Cod18b]. This is licensed GFDL 1.2.

Digits 0-9 cannot be in text.

New in version 0.3.6.

decode(*text*)

Perform decoding of run-length-encoding (RLE).

Parameters **text** (*str*) -- A text string to decode

Returns Word decoded by RLE

Return type str

Examples

```
>>> rle = RLE()
>>> bwt = BWT()
>>> bwt.decode(rle.decode('n\x00ilag'))
'align'
>>> rle.decode('align')
'align'
```

```
>>> bwt.decode(rle.decode('annb\x00aa'))
'banana'
>>> rle.decode('banana')
'banana'
```

```
>>> bwt.decode(rle.decode('ab\x00abbab5a'))
'aaabaababababa'
>>> rle.decode('3abaababababa')
'aaabaababababa'
```

New in version 0.1.0.

Changed in version 0.3.6: Encapsulated in class

encode(*text*)

Perform encoding of run-length-encoding (RLE).

Parameters **text** (*str*) -- A text string to encode

Returns Word decoded by RLE

Return type str

Examples

```
>>> rle = RLE()
>>> bwt = BWT()
>>> rle.encode(bwt.encode('align'))
'n\x00ilag'
>>> rle.encode('align')
'align'
```

```
>>> rle.encode(bwt.encode('banana'))
'annb\x00aa'
>>> rle.encode('banana')
'banana'
```

```
>>> rle.encode(bwt.encode('aaabaababababa'))
'ab\x00abbab5a'
>>> rle.encode('aaabaababababa')
'3abaababababa'
```

New in version 0.1.0.

Changed in version 0.3.6: Encapsulated in class

abydos.compression.**rle_decode**(*text*, *use_bwt=True*)

Perform decoding of run-length-encoding (RLE).

This is a wrapper for *RLE.decode()*.

Parameters

- **text** (*str*) -- A text string to decode
- **use_bwt** (*bool*) -- Indicates whether to perform BWT decoding after RLE decoding

Returns Word decoded by RLE

Return type str

Examples

```
>>> rle_decode('n\x00ilag')
'align'
>>> rle_decode('align', use_bwt=False)
'align'
```

```
>>> rle_decode('annb\x00aa')
'banana'
>>> rle_decode('banana', use_bwt=False)
'banana'
```

```
>>> rle_decode('ab\x00abbab5a')
'aaabaababababa'
>>> rle_decode('3abaababababa', False)
'aaabaababababa'
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the RLE.decode method instead.

abydos.compression.**rle_encode**(*text*, *use_bwt=True*)

Perform encoding of run-length-encoding (RLE).

This is a wrapper for *RLE.encode()*.

Parameters

- **text** (*str*) -- A text string to encode
- **use_bwt** (*bool*) -- Indicates whether to perform BWT encoding before RLE encoding

Returns Word decoded by RLE

Return type str

Examples

```
>>> rle_encode('align')
'n\x00ilag'
>>> rle_encode('align', use_bwt=False)
'align'
```

```
>>> rle_encode('banana')
'annb\x00aa'
>>> rle_encode('banana', use_bwt=False)
'banana'
```

```
>>> rle_encode('aaabaababababa')
'ab\x00abbab5a'
>>> rle_encode('aaabaababababa', False)
'3abaababababa'
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the RLE.encode method instead.

3.1.1.2 abydos.corpus package

abydos.corpus.

The corpus package includes basic and n-gram corpus classes:

- *Corpus*
- *NGramCorpus*
- *UnigramCorpus*

As a quick example of *Corpus*:

```
>>> tqbf = 'The quick brown fox jumped over the lazy dog.\n\n'
>>> tqbf += 'And then it slept.\n\n And the dog ran off.'
>>> corp = Corpus(tqbf)
>>> corp.docs()
[[['The', 'quick', 'brown', 'fox', 'jumped', 'over', 'the', 'lazy', 'dog.']],
[['And', 'then', 'it', 'slept.']], [['And', 'the', 'dog', 'ran', 'off.']]]
>>> round(corp.idf('dog'), 10)
1.0986122887
>>> round(corp.idf('the'), 10)
0.4054651081
```

Here, each sentence is a separate "document". We can retrieve IDF values from the *Corpus*. The same *Corpus* can be used to initialize an *NGramCorpus* and calculate TF values:

```
>>> ngcorp = NGramCorpus(corp)
>>> ngcorp.get_count('the')
2
>>> ngcorp.get_count('fox')
1
```

---

class abydos.corpus.**Corpus**(*corpus_text=''*, *doc_split='nn'*, *sent_split='n'*, *filter_chars=''*, *stop_words=None*, *word_tokenizer=None*)

Bases: object

Corpus class.

Internally, this is a list of lists or lists. The corpus itself is a list of documents. Each document is an ordered list of sentences in those documents. And each sentence is an ordered list of words that make up that sentence.

New in version 0.1.0.

Initialize Corpus.

By default, when importing a corpus:

- two consecutive newlines divide documents
- single newlines divide sentences
- other whitespace divides words

Parameters

- **corpus_text** (*str*) -- The corpus text as a single string
- **doc_split** (*str*) -- A character or string used to split corpus_text into documents
- **sent_split** (*str*) -- A character or string used to split documents into sentences
- **filter_chars** (*list*) -- A list of characters (as a string, tuple, set, or list) to filter out of the corpus text
- **stop_words** (*list*) -- A list of words (as a tuple, set, or list) to filter out of the corpus text
- **word_tokenizer** (*_Tokenizer*) -- A tokenizer to apply to each sentence in order to retrieve the individual "word" tokens. If set to none, str.split() will be used.

Example

```
>>> tqbf = 'The quick brown fox jumped over the lazy dog.\n'
>>> tqbf += 'And then it slept.\n And the dog ran off.'
>>> corp = Corpus(tqbf)
```

New in version 0.1.0.

docs()
Return the docs in the corpus.

Each list within a doc represents the sentences in that doc, each of which is in turn a list of words within that sentence.

Returns The docs in the corpus as a list of lists of lists of strs

Return type [[[str]]]

Example

```
>>> tqbf = 'The quick brown fox jumped over the lazy dog.\n'
>>> tqbf += 'And then it slept.\n And the dog ran off.'
>>> corp = Corpus(tqbf)
>>> corp.docs()
[['The', 'quick', 'brown', 'fox', 'jumped', 'over', 'the', 'lazy',
'dog.'], ['And', 'then', 'it', 'slept.'], ['And', 'the', 'dog',
'ran', 'off.']]]
>>> len(corp.docs())
1
```

New in version 0.1.0.

docs_of_words()
Return the docs in the corpus, with sentences flattened.

Each list within the corpus represents all the words of that document. Thus the sentence level of lists has been flattened.

Returns The docs in the corpus as a list of list of strs

Return type [[str]]

Example

```
>>> tqbf = 'The quick brown fox jumped over the lazy dog.\n'
>>> tqbf += 'And then it slept.\n And the dog ran off.'
>>> corp = Corpus(tqbf)
>>> corp.docs_of_words()
[['The', 'quick', 'brown', 'fox', 'jumped', 'over', 'the', 'lazy',
'dog.', 'And', 'then', 'it', 'slept.', 'And', 'the', 'dog', 'ran',
'off.']]
>>> len(corp.docs_of_words())
1
```

New in version 0.1.0.

idf(*term*, *transform=None*)
Calculate the Inverse Document Frequency of a term in the corpus.

Parameters

- **term** (*str*) -- The term to calculate the IDF of
- **transform** (*function*) -- A function to apply to each document term before checking for the presence of term

Returns The IDF

Return type float

Examples

```
>>> tqbf = 'The quick brown fox jumped over the lazy dog.\n\n'
>>> tqbf += 'And then it slept.\n\n And the dog ran off.'
>>> corp = Corpus(tqbf)
>>> print(corp.docs())
[[['The', 'quick', 'brown', 'fox', 'jumped', 'over', 'the', 'lazy',
'dog.']],
[['And', 'then', 'it', 'slept.']],
[['And', 'the', 'dog', 'ran', 'off.']]]
>>> round(corp.idf('dog'), 10)
1.0986122887
>>> round(corp.idf('the'), 10)
0.4054651081
```

New in version 0.1.0.

paras()

Return the paragraphs in the corpus.

Each list within a paragraph represents the sentences in that doc, each of which is in turn a list of words within that sentence. This is identical to the docs() member function and exists only to mirror part of NLTK's API for corpora.

Returns The paragraphs in the corpus as a list of lists of lists of strs

Return type [[[str]]]

Example

```
>>> tqbf = 'The quick brown fox jumped over the lazy dog.\n'
>>> tqbf += 'And then it slept.\n And the dog ran off.'
>>> corp = Corpus(tqbf)
>>> corp.paras()
[[['The', 'quick', 'brown', 'fox', 'jumped', 'over', 'the', 'lazy',
'dog.'], ['And', 'then', 'it', 'slept.'], ['And', 'the', 'dog',
'ran', 'off.']]]
>>> len(corp.paras())
1
```

New in version 0.1.0.

raw()

Return the raw corpus.

This is reconstructed by joining sub-components with the corpus' split characters

Returns The raw corpus

Return type str

Example

```
>>> tqbf = 'The quick brown fox jumped over the lazy dog.\n'
>>> tqbf += 'And then it slept.\n And the dog ran off.'
>>> corp = Corpus(tqbf)
>>> print(corp.raw())
The quick brown fox jumped over the lazy dog.
And then it slept.
And the dog ran off.
>>> len(corp.raw())
85
```

New in version 0.1.0.

sents()

Return the sentences in the corpus.

Each list within a sentence represents the words within that sentence.

Returns The sentences in the corpus as a list of lists of strs

Return type [[str]]

Example

```
>>> tqbf = 'The quick brown fox jumped over the lazy dog.\n'
>>> tqbf += 'And then it slept.\n And the dog ran off.'
>>> corp = Corpus(tqbf)
>>> corp.sents()
[['The', 'quick', 'brown', 'fox', 'jumped', 'over', 'the', 'lazy',
'dog.'], ['And', 'then', 'it', 'slept.'], ['And', 'the', 'dog',
'ran', 'off.']]
>>> len(corp.sents())
3
```

words()

Return the words in the corpus as a single list.

Returns The words in the corpus as a list of strs

Return type [str]

Example

```
>>> tqbf = 'The quick brown fox jumped over the lazy dog.\n'
>>> tqbf += 'And then it slept.\n And the dog ran off.'
>>> corp = Corpus(tqbf)
>>> corp.words()
['The', 'quick', 'brown', 'fox', 'jumped', 'over', 'the', 'lazy',
'dog.', 'And', 'then', 'it', 'slept.', 'And', 'the', 'dog', 'ran',
'off.']
>>> len(corp.words())
18
```

New in version 0.1.0.

class abydos.corpus.**NGramCorpus**(*corpus=None*)

Bases: object

The NGramCorpus class.

Internally, this is a set of recursively embedded dicts, with n layers for a corpus of n-grams. E.g. for a trigram corpus, this will be a dict of dicts of dicts. More precisely, collections.Counter is used in place of dict, making multiset operations valid and allowing unattested n-grams to be queried.

The key at each level is a word. The value at the most deeply embedded level is a numeric value representing the frequency of the trigram. E.g. the trigram frequency of 'colorless green ideas' would be the value stored in self.ngcorpus['colorless']['green']['ideas'][None].

New in version 0.3.0.

Initialize Corpus.

Parameters **corpus** (Corpus) -- The *Corpus* from which to initialize the n-gram corpus. By default, this is None, which initializes an empty NGramCorpus. This can then be populated using NGramCorpus methods.

Raises **TypeError** -- Corpus argument must be None or of type abydos.Corpus

Example

```
>>> tqbf = 'The quick brown fox jumped over the lazy dog.\n'
>>> tqbf += 'And then it slept.\n And the dog ran off.'
>>> ngcorp = NGramCorpus(Corpus(tqbf))
```

New in version 0.3.0.

corpus_importer(*corpus*, *n_val=1*, *bos='_START_'*, *eos='_END_'*)

Fill in self.ngcorpus from a Corpus argument.

Parameters

- **corpus** (Corpus) -- The Corpus from which to initialize the n-gram corpus
- **n_val** (*int*) -- Maximum n value for n-grams
- **bos** (*str*) -- String to insert as an indicator of beginning of sentence
- **eos** (*str*) -- String to insert as an indicator of end of sentence

Raises **TypeError** -- Corpus argument of the Corpus class required.

Example

```
>>> tqbf = 'The quick brown fox jumped over the lazy dog.\n'
>>> tqbf += 'And then it slept.\n And the dog ran off.'
>>> ngcorp = NGramCorpus()
>>> ngcorp.corpus_importer(Corpus(tqbf))
```

New in version 0.3.0.

get_count(*ngram*, *corpus=None*)

Get the count of an n-gram in the corpus.

Parameters

- **ngram** (*str*) -- The n-gram to retrieve the count of from the n-gram corpus
- **corpus** (Corpus) -- The corpus

Returns The n-gram count

Return type int

Examples

```
>>> tqbf = 'The quick brown fox jumped over the lazy dog.\n'
>>> tqbf += 'And then it slept.\n And the dog ran off.'
>>> ngcorp = NGramCorpus(Corpus(tqbf))
>>> ngcorp.get_count('the')
2
>>> ngcorp.get_count('fox')
1
```

New in version 0.3.0.

gng_importer(*corpus_file*)

Fill in self.ngcorpus from a Google NGram corpus file.

Parameters **corpus_file** (*file*) -- The Google NGram file from which to initialize the n-gram corpus

New in version 0.3.0.

class abydos.corpus.**UnigramCorpus**(*corpus_text=''*, *documents=0*, *word_transform=None*, *word_tokenizer=None*)

Bases: object

Unigram corpus class.

Largely intended for calculating inverse document frequence (IDF) from a large corpus of unigram (or smaller) tokens, this class encapsulates a dict object. Each key is a unigram token whose value is a tuple consisting of the number of times a term appeared and the number of distinct documents in which it appeared.

New in version 0.4.0.

Initialize UnigramCorpus.

Parameters

- **corpus_text** (*str*) -- The corpus text as a single string
- **documents** (*int*) -- The number of documents in the corpus. If equal to 0 (the default) then the maximum from the internal dictionary's distinct documents count.
- **word_transform** (*function*) -- A function to apply to each term before term tokenization and addition to the corpus. One might use this, for example, to apply Soundex encoding to each term.
- **word_tokenizer** (*_Tokenizer*) -- A tokenizer to apply to each sentence in order to retrieve the individual "word" tokens. If set to none, str.split() will be used.

Example

```
>>> tqbf = 'The quick brown fox jumped over the lazy dog.\n'
>>> tqbf += 'And then it slept.\n And the dog ran off.'
>>> corp = UnigramCorpus(tqbf)
```

New in version 0.4.0.

add_document (*doc*)
Add a new document to the corpus.

Parameters **doc** (*str*) -- A string, representing the document to be added.

New in version 0.4.0.

gng_importer (*corpus_file*)
Fill in self.corpus from a Google NGram corpus file.

Parameters **corpus_file** (*file*) -- The Google NGram file from which to initialize the n-gram corpus

New in version 0.4.0.

idf (*term*)
Calculate the Inverse Document Frequency of a term in the corpus.

Parameters **term** (*str*) -- The term to calculate the IDF of

Returns The IDF

Return type float

Examples

```
>>> tqbf = 'the quick brown fox jumped over the lazy dog\n\n'
>>> tqbf += 'and then it slept\n\n and the dog ran off'
>>> corp = UnigramCorpus(tqbf)
>>> round(corp.idf('dog'), 10)
0.6931471806
>>> round(corp.idf('the'), 10)
0.6931471806
```

New in version 0.4.0.

load_corpus (*filename*)
Load the corpus from a file.

This employs pickle to load the corpus (a defaultdict). Other parameters of the corpus, such as its word_tokenizer, will not be affected and should be set during initialization.

Parameters **filename** (*str*) -- The filename to load the corpus from.

New in version 0.4.0.

save_corpus (*filename*)
Save the corpus to a file.

This employs pickle to save the corpus (a defaultdict). Other parameters of the corpus, such as its word_tokenizer, will not be affected and should be set during initialization.

Parameters **filename** (*str*) -- The filename to save the corpus to.

New in version 0.4.0.

3.1.1.3 abydos.distance package

abydos.distance.

The distance package implements string distance measure and metric classes:

These include traditional Levenshtein edit distance and related algorithms:

- Levenshtein distance (*Levenshtein*)
- Optimal String Alignment distance (*Levenshtein* with `mode='osa'`)
- Damerau-Levenshtein distance (*DamerauLevenshtein*)
- Yujian-Bo normalized edit distance (*YujianBo*)
- Higuera-Micó contextual normalized edit distance (*HigueraMico*)
- Indel distance (*Indel*)
- Syllable Alignment Pattern Searching similarity (*distance.SAPS*)
- Meta-Levenshtein distance (*MetaLevenshtein*)
- Covington distance (*Covington*)
- ALINE distance (*ALINE*)
- FlexMetric distance (*FlexMetric*)
- BI-SIM similarity (*BISIM*)
- Discounted Levenshtein distance (*DiscountedLevenshtein*)
- Phonetic edit distance (*PhoneticEditDistance*)

Hamming distance (*Hamming*), Relaxed Hamming distance (*RelaxedHamming*), and the closely related Modified Language-Independent Product Name Search distance (*MLIPNS*) are provided.

Block edit distances:

- Tichy edit distance (*Tichy*)
- Levenshtein distance with block operations (*BlockLevenshtein*)
- Rees-Levenshtein distance (*ReesLevenshtein*)
- Cormode's LZ distance (*CormodeLZ*)
- Shapira-Storer I edit distance with block moves, greedy algorithm (*ShapiraStorerI*)

Distance metrics developed for the US Census or derived from them are included:

- Jaro distance (*JaroWinkler* with `mode='Jaro'`)
- Jaro-Winkler distance (*JaroWinkler*)
- Strcmp95 distance (*Strcmp95*)
- Iterative-SubString (I-Sub) correlation (*IterativeSubString*)

A large set of multi-set token-based distance metrics are provided, including:

- AMPLE similarity (*AMPLE*)
- AZZOO similarity (*AZZOO*)
- Anderberg's D similarity (*Anderberg*)
- Andres & Marzo's Delta correlation (*AndresMarzoDelta*)

- Baroni-Urbani & Buser I similarity (*BaroniUrbaniBuserI*)
- Baroni-Urbani & Buser II correlation (*BaroniUrbaniBuserII*)
- Batagelj & Bren similarity (*BatageljBren*)
- Baulieu I distance (*BaulieuI*)
- Baulieu II distance (*BaulieuII*)
- Baulieu III distance (*BaulieuIII*)
- Baulieu IV distance (*BaulieuIV*)
- Baulieu V distance (*BaulieuV*)
- Baulieu VI distance (*BaulieuVI*)
- Baulieu VII distance (*BaulieuVII*)
- Baulieu VIII distance (*BaulieuVIII*)
- Baulieu IX distance (*BaulieuIX*)
- Baulieu X distance (*BaulieuX*)
- Baulieu XI distance (*BaulieuXI*)
- Baulieu XII distance (*BaulieuXII*)
- Baulieu XIII distance (*BaulieuXIII*)
- Baulieu XIV distance (*BaulieuXIV*)
- Baulieu XV distance (*BaulieuXV*)
- Benini I correlation (*BeniniI*)
- Benini II correlation (*BeniniII*)
- Bennet's S correlation (*Bennet*)
- Braun-Blanquet similarity (*BraunBlanquet*)
- Canberra distance (*Canberra*)
- Cao similarity (*Cao*)
- Chao's Dice similarity (*ChaoDice*)
- Chao's Jaccard similarity (*ChaoJaccard*)
- Chebyshev distance (*Chebyshev*)
- Chord distance (*Chord*)
- Clark distance (*Clark*)
- Clement similarity (*Clement*)
- Cohen's Kappa similarity (*CohenKappa*)
- Cole correlation (*Cole*)
- Consonni & Todeschini I similarity (*ConsonniTodeschiniI*)
- Consonni & Todeschini II similarity (*ConsonniTodeschiniII*)
- Consonni & Todeschini III similarity (*ConsonniTodeschiniIII*)
- Consonni & Todeschini IV similarity (*ConsonniTodeschiniIV*)

- Consonni & Todeschini V correlation (*ConsonniTodeschiniV*)
- Cosine similarity (*Cosine*)
- Dennis similarity (*Dennis*)
- Dice's Asymmetric I similarity (*DiceAsymmetricI*)
- Dice's Asymmetric II similarity (*DiceAsymmetricII*)
- Digby correlation (*Digby*)
- Dispersion correlation (*Dispersion*)
- Doolittle similarity (*Doolittle*)
- Dunning similarity (*Dunning*)
- Euclidean distance (*Euclidean*)
- Eyraud similarity (*Eyraud*)
- Fager & McGowan similarity (*FagerMcGowan*)
- Faith similarity (*Faith*)
- Fidelity similarity (*Fidelity*)
- Fleiss correlation (*Fleiss*)
- Fleiss-Levin-Paik similarity (*FleissLevinPaik*)
- Forbes I similarity (*ForbesI*)
- Forbes II correlation (*ForbesII*)
- Fossum similarity (*Fossum*)
- Generalized Fleiss correlation (*GeneralizedFleiss*)
- Gilbert correlation (*Gilbert*)
- Gilbert & Wells similarity (*GilbertWells*)
- Gini I correlation (*GiniI*)
- Gini II correlation (*GiniII*)
- Goodall similarity (*Goodall*)
- Goodman & Kruskal's Lambda similarity (*GoodmanKruskalLambda*)
- Goodman & Kruskal's Lambda-r correlation (*GoodmanKruskalLambdaR*)
- Goodman & Kruskal's Tau A similarity (*GoodmanKruskalTauA*)
- Goodman & Kruskal's Tau B similarity (*GoodmanKruskalTauB*)
- Gower & Legendre similarity (*GowerLegendre*)
- Guttman Lambda A similarity (*GuttmanLambdaA*)
- Guttman Lambda B similarity (*GuttmanLambdaB*)
- Gwet's AC correlation (*GwetAC*)
- Hamann correlation (*Hamann*)
- Harris & Lahey similarity (*HarrisLahey*)
- Hassanat distance (*Hassanat*)

- Hawkins & Dotson similarity (*HawkinsDotson*)
- Hellinger distance (*Hellinger*)
- Henderson-Heron similarity (*HendersonHeron*)
- Horn-Morisita similarity (*HornMorisita*)
- Hurlbert correlation (*Hurlbert*)
- Jaccard similarity (*Jaccard*) & Tanimoto coefficient (*Jaccard.tanimoto_coeff()*)
- Jaccard-NM similarity (*JaccardNM*)
- Johnson similarity (*Johnson*)
- Kendall's Tau correlation (*KendallTau*)
- Kent & Foster I similarity (*KentFosterI*)
- Kent & Foster II similarity (*KentFosterII*)
- Köppen I correlation (*KoppenI*)
- Köppen II similarity (*KoppenII*)
- Kuder & Richardson correlation (*KuderRichardson*)
- Kuhns I correlation (*KuhnsI*)
- Kuhns II correlation (*KuhnsII*)
- Kuhns III correlation (*KuhnsIII*)
- Kuhns IV correlation (*KuhnsIV*)
- Kuhns V correlation (*KuhnsV*)
- Kuhns VI correlation (*KuhnsVI*)
- Kuhns VII correlation (*KuhnsVII*)
- Kuhns VIII correlation (*KuhnsVIII*)
- Kuhns IX correlation (*KuhnsIX*)
- Kuhns X correlation (*KuhnsX*)
- Kuhns XI correlation (*KuhnsXI*)
- Kuhns XII similarity (*KuhnsXII*)
- Kulczynski I similarity (*KulczynskiI*)
- Kulczynski II similarity (*KulczynskiII*)
- Lorentzian distance (*Lorentzian*)
- Maarel correlation (*Maarel*)
- Manhattan distance (*Manhattan*)
- Morisita similarity (*Morisita*)
- marking distance (*Marking*)
- marking metric (*MarkingMetric*)
- MASI similarity (*MASI*)
- Matusita distance (*Matusita*)

- Maxwell & Pilliner correlation (*MaxwellPilliner*)
- McConnaughey correlation (*McConnaughey*)
- McEwen & Michael correlation (*McEwenMichael*)
- mean squared contingency correlation (*MSContingency*)
- Michael similarity (Michael)
- Michelet similarity (*Michelet*)
- Millar distance (*Millar*)
- Minkowski distance (*Minkowski*)
- Mountford similarity (*Mountford*)
- Mutual Information similarity (*MutualInformation*)
- Overlap distance (*Overlap*)
- Pattern difference (*Pattern*)
- Pearson & Heron II correlation (*PearsonHeronII*)
- Pearson II similarity (*PearsonII*)
- Pearson III correlation (*PearsonIII*)
- Pearson's Chi-Squared similarity (*PearsonChiSquared*)
- Pearson's Phi correlation (*PearsonPhi*)
- Peirce correlation (*Peirce*)
- q-gram distance (*QGram*)
- Raup-Crick similarity (*RaupCrick*)
- Rogers & Tanimoto similarity (*RogersTanimoto*)
- Rogot & Goldberg similarity (*RogotGoldberg*)
- Russell & Rao similarity (*RussellRao*)
- Scott's Pi correlation (*ScottPi*)
- Shape difference (*Shape*)
- Size difference (*Size*)
- Sokal & Michener similarity (*SokalMichener*)
- Sokal & Sneath I similarity (*SokalSneathI*)
- Sokal & Sneath II similarity (*SokalSneathII*)
- Sokal & Sneath III similarity (*SokalSneathIII*)
- Sokal & Sneath IV similarity (*SokalSneathIV*)
- Sokal & Sneath V similarity (*SokalSneathV*)
- Sørensen–Dice coefficient (*Dice*)
- Sorgenfrei similarity (*Sorgenfrei*)
- Steffensen similarity (*Steffensen*)
- Stiles similarity (*Stiles*)

- Stuart's Tau correlation (*StuartTau*)
- Tarantula similarity (*Tarantula*)
- Tarwid correlation (*Tarwid*)
- Tetrachoric correlation coefficient (Tetrachronic)
- Tulloss' R similarity (*TullossR*)
- Tulloss' S similarity (*TullossS*)
- Tulloss' T similarity (*TullossT*)
- Tulloss' U similarity (*TullossU*)
- Tversky distance (*Tversky*)
- Weighted Jaccard similarity (*WeightedJaccard*)
- Unigram subtuple similarity (*UnigramSubtuple*)
- Unknown A correlation (*UnknownA*)
- Unknown B similarity (*UnknownB*)
- Unknown C similarity (*UnknownC*)
- Unknown D similarity (*UnknownD*)
- Unknown E correlation (*UnknownE*)
- Unknown F similarity (*UnknownF*)
- Unknown G similarity (*UnknownG*)
- Unknown H similarity (*UnknownH*)
- Unknown I similarity (*UnknownI*)
- Unknown J similarity (*UnknownJ*)
- Unknown K distance (*UnknownK*)
- Unknown L similarity (*UnknownL*)
- Unknown M similarity (*UnknownM*)
- Upholt similarity (*Upholt*)
- Warrens I correlation (*WarrensI*)
- Warrens II similarity (*WarrensII*)
- Warrens III correlation (*WarrensIII*)
- Warrens IV similarity (*WarrensIV*)
- Warrens V similarity (*WarrensV*)
- Whittaker distance (*Whittaker*)
- Yates' Chi-Squared similarity (*YatesChiSquared*)
- Yule's Q correlation (*YuleQ*)
- Yule's Q II distance (*YuleQII*)
- Yule's Y correlation (*YuleY*)
- YJHHR distance (*YJHHR*)

- Bhattacharyya distance (*Bhattacharyya*)
- Brainerd-Robinson similarity (*BrainerdRobinson*)
- Quantitative Cosine similarity (*QuantitativeCosine*)
- Quantitative Dice similarity (*QuantitativeDice*)
- Quantitative Jaccard similarity (*QuantitativeJaccard*)
- Roberts similarity (*Roberts*)
- Average linkage distance (*AverageLinkage*)
- Single linkage distance (*SingleLinkage*)
- Complete linkage distance (*CompleteLinkage*)
- Bag distance (*Bag*)
- Soft cosine similarity (*SoftCosine*)
- Monge-Elkan distance (*MongeElkan*)
- TF-IDF similarity (*TFIDF*)
- SoftTF-IDF similarity (*SoftTFIDF*)
- Jensen-Shannon divergence (*JensenShannon*)
- Simplified Fellegi-Sunter distance (*FellegiSunter*)
- MinHash similarity (*MinHash*)
- BLEU similarity (*BLEU*)
- Rouge-L similarity (*RougeL*)
- Rouge-W similarity (*RougeW*)
- Rouge-S similarity (*RougeS*)
- Rouge-SU similarity (*RougeSU*)
- Positional Q-Gram Dice distance (*PositionalQGramDice*)
- Positional Q-Gram Jaccard distance (*PositionalQGramJaccard*)
- Positional Q-Gram Overlap distance (*PositionalQGramOverlap*)

Three popular sequence alignment algorithms are provided:

- Needleman-Wunsch score (*NeedlemanWunsch*)
- Smith-Waterman score (*SmithWaterman*)
- Gotoh score (*Gotoh*)

Classes relating to substring and subsequence distances include:

- Longest common subsequence (*LCSseq*)
- Longest common substring (*LCSstr*)
- Ratcliff-Obserhelp distance (*RatcliffObershelp*)

A number of simple distance classes provided in the package include:

- Identity distance (*Ident*)
- Length distance (*Length*)

- Prefix distance (*Prefix*)
- Suffix distance (*Suffix*)

Normalized compression distance classes for a variety of compression algorithms are provided:

- zlib (*NCDzlib*)
- bzip2 (*NCDbz2*)
- lzma (*NCDlzma*)
- LZSS (*NCDlzss*)
- arithmetic coding (*NCDarith*)
- PAQ9A (*NCDpaq9a*)
- BWT plus RLE (*NCDbwtrle*)
- RLE (*NCDrle*)

Three similarity measures from SeatGeek's FuzzyWuzzy:

- FuzzyWuzzy Partial String similarity (*FuzzyWuzzyPartialString*)
- FuzzyWuzzy Token Sort similarity (*FuzzyWuzzyTokenSort*)
- FuzzyWuzzy Token Set similarity (*FuzzyWuzzyTokenSet*)

A convenience class, allowing one to pass a list of string transforms (phonetic algorithms, string transforms, and/or stemmers) and, optionally, a string distance measure to compute the similarity/distance of two strings that have undergone each transform, is provided in:

- Phonetic distance (*PhoneticDistance*)

The remaining distance measures & metrics include:

- Western Airlines' Match Rating Algorithm comparison (*distance.MRA*)
- Editex (*Editex*)
- Bavarian Landesamt für Statistik distance (*Baystat*)
- Eudex distance (*distance.Eudex*)
- Sift4 distance (*Sift4*, *Sift4Simplest*, *Sift4Extended*)
- Typo distance (*Typo*)
- Synoname (*Synoname*)
- Ozbay metric (*Ozbay*)
- Indice de Similitude-Guth (*ISG*)
- INClusion Programme (*Inclusion*)
- Guth (*Guth*)
- Victorian Panel Study (*VPS*)
- LIG3 (*LIG3*)
- String subsequence kernel (SSK) (*SSK*)

Most of the distance and similarity measures have `sim` and `dist` methods, which return a measure that is normalized to the range $[0, 1]$. The normalized distance and similarity are always complements, so the normalized distance will always equal 1 - the similarity for a particular measure supplied with the same input. Some measures have an absolute distance method `dist_abs` that is not limited to any range.

All three methods can be demonstrated using the *DamerauLevenshtein* class:

```
>>> dl = DamerauLevenshtein()
>>> dl.dist_abs('orange', 'strange')
2
>>> dl.dist('orange', 'strange')
0.2857142857142857
>>> dl.sim('orange', 'strange')
0.7142857142857143
```

---

abydos.distance.**sim**(*src*, *tar*, *method=<function sim_levenshtein>*)

Return a similarity of two strings.

This is a generalized function for calling other similarity functions.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison
- **method** (*function*) -- Specifies the similarity metric (*sim_levenshtein()* by default)

Returns Similarity according to the specified function

Return type float

Raises **AttributeError** -- Unknown distance function

Examples

```
>>> round(sim('cat', 'hat'), 12)
0.666666666667
>>> round(sim('Niall', 'Neil'), 12)
0.4
>>> sim('aluminum', 'Catalan')
0.125
>>> sim('ATCG', 'TAGC')
0.25
```

New in version 0.1.0.

abydos.distance.**dist**(*src*, *tar*, *method=<function sim_levenshtein>*)

Return a distance between two strings.

This is a generalized function for calling other distance functions.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison
- **method** (*function*) -- Specifies the similarity metric (*sim_levenshtein()* by default) -- Note that this takes a similarity metric function, not a distance metric function.

Returns Distance according to the specified function

Return type float

Raises **AttributeError** -- Unknown distance function

Examples

```
>>> round(dist('cat', 'hat'), 12)
0.333333333333
>>> round(dist('Niall', 'Neil'), 12)
0.6
>>> dist('aluminum', 'Catalan')
0.875
>>> dist('ATCG', 'TAGC')
0.75
```

New in version 0.1.0.

class abydos.distance.**Levenshtein**(*mode='lev'*, *cost=(1*, *1*, *1*, *1)*, *normalizer=<built-in function max>*, *taper=False*, ***kwargs*)

Bases: abydos.distance._distance._Distance

Levenshtein distance.

This is the standard edit distance measure. Cf. [Lev65][Lev66].

Optimal string alignment (aka restricted Damerau-Levenshtein distance) [Boy11] is also supported.

The ordinary Levenshtein & Optimal String Alignment distance both employ the Wagner-Fischer dynamic programming algorithm [WF74].

Levenshtein edit distance ordinarily has unit insertion, deletion, and substitution costs.

New in version 0.3.6.

Changed in version 0.4.0: Added taper option

Initialize Levenshtein instance.

Parameters

- **mode** (*str*) -- Specifies a mode for computing the Levenshtein distance:
  - lev (default) computes the ordinary Levenshtein distance, in which edits may include inserts, deletes, and substitutions
  - osa computes the Optimal String Alignment distance, in which edits may include inserts, deletes, substitutions, and transpositions but substrings may only be edited once
- **cost** (*tuple*) -- A 4-tuple representing the cost of the four possible edits: inserts, deletes, substitutions, and transpositions, respectively (by default: (1, 1, 1, 1))
- **normalizer** (*function*) -- A function that takes an list and computes a normalization term by which the edit distance is divided (max by default). Another good option is the sum function.
- **taper** (*bool*) -- Enables cost tapering. Following [ZD96], it causes edits at the start of the string to "just [exceed] twice the minimum penalty for replacement or deletion at the end of the string".
- ****kwargs** -- Arbitrary keyword arguments

New in version 0.4.0.

alignment(*src*, *tar*)

Return the Levenshtein alignment of two strings.

Parameters

- **src** (*str*) -- Source string for comparison

- **tar** (*str*) -- Target string for comparison

Returns A tuple containing the Levenshtein distance and the two strings, aligned.

Return type tuple

Examples

```
>>> cmp = Levenshtein()
>>> cmp.alignment('cat', 'hat')
(1.0, 'cat', 'hat')
>>> cmp.alignment('Niall', 'Neil')
(3.0, 'N-iall', 'Nei-l-')
>>> cmp.alignment('aluminum', 'Catalan')
(7.0, '-aluminum', 'Catalan--')
>>> cmp.alignment('ATCG', 'TAGC')
(3.0, 'ATCG-', '-TAGC')
```

```
>>> cmp = Levenshtein(mode='osa')
>>> cmp.alignment('ATCG', 'TAGC')
(2.0, 'ATCG', 'TAGC')
>>> cmp.alignment('ACTG', 'TAGC')
(4.0, 'ACT-G-', '--TAGC')
```

New in version 0.4.1.

dist (*src*, *tar*)

Return the normalized Levenshtein distance between two strings.

The Levenshtein distance is normalized by dividing the Levenshtein distance (calculated by either of the two supported methods) by the greater of the number of characters in src times the cost of a delete and the number of characters in tar times the cost of an insert. For the case in which all operations have $cost = 1$, this is equivalent to the greater of the length of the two strings src & tar.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns The normalized Levenshtein distance between src & tar

Return type float

Examples

```
>>> cmp = Levenshtein()
>>> round(cmp.dist('cat', 'hat'), 12)
0.333333333333
>>> round(cmp.dist('Niall', 'Neil'), 12)
0.6
>>> cmp.dist('aluminum', 'Catalan')
0.875
>>> cmp.dist('ATCG', 'TAGC')
0.75
```

New in version 0.1.0.

Changed in version 0.3.6: Encapsulated in class

dist_abs(*src*, *tar*)

Return the Levenshtein distance between two strings.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns The Levenshtein distance between src & tar

Return type int (may return a float if cost has float values)

Examples

```
>>> cmp = Levenshtein()
>>> cmp.dist_abs('cat', 'hat')
1
>>> cmp.dist_abs('Niall', 'Neil')
3
>>> cmp.dist_abs('aluminum', 'Catalan')
7
>>> cmp.dist_abs('ATCG', 'TAGC')
3
```

```
>>> cmp = Levenshtein(mode='osa')
>>> cmp.dist_abs('ATCG', 'TAGC')
2
>>> cmp.dist_abs('ACTG', 'TAGC')
4
```

New in version 0.1.0.

Changed in version 0.3.6: Encapsulated in class

abydos.distance.**levenshtein**(*src*, *tar*, *mode='lev'*, *cost=(1, 1, 1, 1)*)

Return the Levenshtein distance between two strings.

This is a wrapper of *Levenshtein.dist_abs()*.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison
- **mode** (*str*) -- Specifies a mode for computing the Levenshtein distance:
  - `lev` (default) computes the ordinary Levenshtein distance, in which edits may include inserts, deletes, and substitutions
  - `osa` computes the Optimal String Alignment distance, in which edits may include inserts, deletes, substitutions, and transpositions but substrings may only be edited once
- **cost** (*tuple*) -- A 4-tuple representing the cost of the four possible edits: inserts, deletes, substitutions, and transpositions, respectively (by default: (1, 1, 1, 1))

Returns The Levenshtein distance between src & tar

Return type int (may return a float if cost has float values)

Examples

```
>>> levenshtein('cat', 'hat')
1
>>> levenshtein('Niall', 'Neil')
3
>>> levenshtein('aluminum', 'Catalan')
7
>>> levenshtein('ATCG', 'TAGC')
3
```

```
>>> levenshtein('ATCG', 'TAGC', mode='osa')
2
>>> levenshtein('ACTG', 'TAGC', mode='osa')
4
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the Levenshtein.dist_abs method instead.

abydos.distance.**dist_levenshtein**(*src*, *tar*, *mode='lev'*, *cost=(1, 1, 1, 1)*)

Return the normalized Levenshtein distance between two strings.

This is a wrapper of *Levenshtein.dist()*.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison
- **mode** (*str*) -- Specifies a mode for computing the Levenshtein distance:
  - `lev` (default) computes the ordinary Levenshtein distance, in which edits may include inserts, deletes, and substitutions
  - `osa` computes the Optimal String Alignment distance, in which edits may include inserts, deletes, substitutions, and transpositions but substrings may only be edited once
- **cost** (*tuple*) -- A 4-tuple representing the cost of the four possible edits: inserts, deletes, substitutions, and transpositions, respectively (by default: (1, 1, 1, 1))

Returns The Levenshtein distance between src & tar

Return type float

Examples

```
>>> round(dist_levenshtein('cat', 'hat'), 12)
0.333333333333
>>> round(dist_levenshtein('Niall', 'Neil'), 12)
0.6
>>> dist_levenshtein('aluminum', 'Catalan')
0.875
>>> dist_levenshtein('ATCG', 'TAGC')
0.75
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the Levenshtein.dist method instead.

abydos.distance.**sim_levenshtein**(*src*, *tar*, *mode='lev'*, *cost=(1, 1, 1, 1)*)

Return the Levenshtein similarity of two strings.

This is a wrapper of `Levenshtein.sim()`.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison
- **mode** (*str*) -- Specifies a mode for computing the Levenshtein distance:
  - `lev` (default) computes the ordinary Levenshtein distance, in which edits may include inserts, deletes, and substitutions
  - `osa` computes the Optimal String Alignment distance, in which edits may include inserts, deletes, substitutions, and transpositions but substrings may only be edited once
- **cost** (*tuple*) -- A 4-tuple representing the cost of the four possible edits: inserts, deletes, substitutions, and transpositions, respectively (by default: (1, 1, 1, 1))

Returns The Levenshtein similarity between src & tar

Return type float

Examples

```
>>> round(sim_levenshtein('cat', 'hat'), 12)
0.666666666667
>>> round(sim_levenshtein('Niall', 'Neil'), 12)
0.4
>>> sim_levenshtein('aluminum', 'Catalan')
0.125
>>> sim_levenshtein('ATCG', 'TAGC')
0.25
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the Levenshtein.sim method instead.

class abydos.distance.**DamerauLevenshtein**(*cost=(1, 1, 1, 1)*, *normalizer=<built-in function max>*, ***kwargs*)

Bases: `abydos.distance._distance._Distance`

Damerau-Levenshtein distance.

This computes the Damerau-Levenshtein distance [Dam64]. Damerau-Levenshtein code is based on Java code by Kevin L. Stern [Ste14], under the MIT license: https://github.com/KevinStern/software-and-algorithms/blob/master/src/main/java/blogspot/software_and_algorithms/stern_library/string/DamerauLevenshteinAlgorithm.java

Initialize Levenshtein instance.

Parameters

- **cost** (*tuple*) -- A 4-tuple representing the cost of the four possible edits: inserts, deletes, substitutions, and transpositions, respectively (by default: (1, 1, 1, 1))
- **normalizer** (*function*) -- A function that takes an list and computes a normalization term by which the edit distance is divided (max by default). Another good option is the sum function.

- ****kwargs** -- Arbitrary keyword arguments

New in version 0.4.0.

dist(*src*, *tar*)

Return the Damerau-Levenshtein similarity of two strings.

Damerau-Levenshtein distance normalized to the interval [0, 1].

The Damerau-Levenshtein distance is normalized by dividing the Damerau-Levenshtein distance by the greater of the number of characters in src times the cost of a delete and the number of characters in tar times the cost of an insert. For the case in which all operations have $cost = 1$, this is equivalent to the greater of the length of the two strings src & tar.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns The normalized Damerau-Levenshtein distance

Return type float

Examples

```
>>> cmp = DamerauLevenshtein()
>>> round(cmp.dist('cat', 'hat'), 12)
0.333333333333
>>> round(cmp.dist('Niall', 'Neil'), 12)
0.6
>>> cmp.dist('aluminum', 'Catalan')
0.875
>>> cmp.dist('ATCG', 'TAGC')
0.5
```

New in version 0.1.0.

Changed in version 0.3.6: Encapsulated in class

dist_abs(*src*, *tar*)

Return the Damerau-Levenshtein distance between two strings.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns The Damerau-Levenshtein distance between src & tar

Return type int (may return a float if cost has float values)

Raises **ValueError** -- Unsupported cost assignment; the cost of two transpositions must not be less than the cost of an insert plus a delete.

Examples

```
>>> cmp = DamerauLevenshtein()
>>> cmp.dist_abs('cat', 'hat')
1
>>> cmp.dist_abs('Niall', 'Neil')
3
>>> cmp.dist_abs('aluminum', 'Catalan')
7
>>> cmp.dist_abs('ATCG', 'TAGC')
2
```

New in version 0.1.0.

Changed in version 0.3.6: Encapsulated in class

abydos.distance.**damerau_levenshtein**(*src*, *tar*, *cost=(1, 1, 1, 1)*)

Return the Damerau-Levenshtein distance between two strings.

This is a wrapper of *DamerauLevenshtein.dist_abs()*.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison
- **cost** (*tuple*) -- A 4-tuple representing the cost of the four possible edits: inserts, deletes, substitutions, and transpositions, respectively (by default: (1, 1, 1, 1))

Returns The Damerau-Levenshtein distance between src & tar

Return type int (may return a float if cost has float values)

Examples

```
>>> damerau_levenshtein('cat', 'hat')
1
>>> damerau_levenshtein('Niall', 'Neil')
3
>>> damerau_levenshtein('aluminum', 'Catalan')
7
>>> damerau_levenshtein('ATCG', 'TAGC')
2
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the DamerauLevenshtein.dist_abs method instead.

abydos.distance.**dist_damerau**(*src*, *tar*, *cost=(1, 1, 1, 1)*)

Return the Damerau-Levenshtein similarity of two strings.

This is a wrapper of *DamerauLevenshtein.dist()*.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

- **cost** (*tuple*) -- A 4-tuple representing the cost of the four possible edits: inserts, deletes, substitutions, and transpositions, respectively (by default: (1, 1, 1, 1))

Returns The normalized Damerau-Levenshtein distance

Return type float

Examples

```
>>> round(dist_damerau('cat', 'hat'), 12)
0.333333333333
>>> round(dist_damerau('Niall', 'Neil'), 12)
0.6
>>> dist_damerau('aluminum', 'Catalan')
0.875
>>> dist_damerau('ATCG', 'TAGC')
0.5
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the DamerauLevenshtein.dist method instead.

abydos.distance.**sim_damerau**(*src*, *tar*, *cost=(1, 1, 1, 1)*)

Return the Damerau-Levenshtein similarity of two strings.

This is a wrapper of `DamerauLevenshtein.sim()`.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison
- **cost** (*tuple*) -- A 4-tuple representing the cost of the four possible edits: inserts, deletes, substitutions, and transpositions, respectively (by default: (1, 1, 1, 1))

Returns The normalized Damerau-Levenshtein similarity

Return type float

Examples

```
>>> round(sim_damerau('cat', 'hat'), 12)
0.666666666667
>>> round(sim_damerau('Niall', 'Neil'), 12)
0.4
>>> sim_damerau('aluminum', 'Catalan')
0.125
>>> sim_damerau('ATCG', 'TAGC')
0.5
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the DamerauLevenshtein.sim method instead.

class abydos.distance.**ShapiraStorerI**(*cost=(1, 1)*, *prime=False*, ***kwargs*)

Bases: abydos.distance._distance._Distance

Shapira & Storer I edit distance with block moves, greedy algorithm.

Shapira & Storer's greedy edit distance [SS07] is similar to Levenshtein edit distance, but with two important distinctions:

- It considers blocks of characters, if they occur in both the source and target strings, so the edit distance between 'abcab' and 'abc' is only 1, since the substring 'ab' occurs in both and can be inserted as a block into 'abc'.
- It allows three edit operations: insert, delete, and move (but not substitute). Thus the distance between 'abcde' and 'deabc' is only 1 because the block 'abc' can be moved in 1 move operation, rather than being deleted and inserted in 2 separate operations.

If prime is set to True at initialization, this employs the greedy' algorithm, which limits replacements of blocks in the two strings to matching occurrences of the LCS.

New in version 0.4.0.

Initialize ShapiraStorerI instance.

Parameters

- **prime** (*bool*) -- If True, employs the greedy' algorithm rather than greedy
- ****kwargs** -- Arbitrary keyword arguments

New in version 0.4.0.

dist(*src*, *tar*)

Return the normalized Shapira & Storer I distance.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns The normalized Shapira & Storer I distance between src & tar

Return type float

Examples

```
>>> cmp = ShapiraStorerI()
>>> round(cmp.dist('cat', 'hat'), 12)
0.333333333333
>>> round(cmp.dist('Niall', 'Neil'), 12)
0.333333333333
>>> cmp.dist('aluminum', 'Catalan')
0.6
>>> cmp.dist('ATCG', 'TAGC')
0.25
```

New in version 0.4.0.

dist_abs(*src*, *tar*)

Return the Shapira & Storer I edit distance between two strings.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns The Shapira & Storer I edit distance between src & tar

Return type int

Examples

```
>>> cmp = ShapiraStorerI()
>>> cmp.dist_abs('cat', 'hat')
2
>>> cmp.dist_abs('Niall', 'Neil')
3
>>> cmp.dist_abs('aluminum', 'Catalan')
9
>>> cmp.dist_abs('ATCG', 'TAGC')
2
```

New in version 0.4.0.

class abydos.distance.**Marking**(***kwargs*)

Bases: abydos.distance._distance._Distance

Ehrenfeucht & Haussler's marking distance.

This edit distance [EH88] is the number of *marked* characters in one word that must be masked in order for that word to consist entirely of substrings of another word.

It is normalized by the length of the first word.

New in version 0.4.0.

Initialize Marking instance.

Parameters ****kwargs** -- Arbitrary keyword arguments

New in version 0.4.0.

dist(*src*, *tar*)

Return the normalized marking distance of two strings.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns marking distance

Return type float

Examples

```
>>> cmp = Marking()
>>> cmp.dist('cat', 'hat')
0.3333333333333333
>>> cmp.dist('Niall', 'Neil')
0.6
>>> cmp.dist('aluminum', 'Catalan')
0.625
>>> cmp.dist('ATCG', 'TAGC')
0.5
```

(continues on next page)

(continued from previous page)

```
>>> cmp.dist('cbaabdcb', 'abcba')
0.25
```

New in version 0.4.0.

dist_abs(*src*, *tar*)
Return the marking distance of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns marking distance

Return type int

Examples

```
>>> cmp = Marking()
>>> cmp.dist_abs('cat', 'hat')
1
>>> cmp.dist_abs('Niall', 'Neil')
3
>>> cmp.dist_abs('aluminum', 'Catalan')
5
>>> cmp.dist_abs('ATCG', 'TAGC')
2
>>> cmp.dist_abs('cbaabdcb', 'abcba')
2
```

New in version 0.4.0.

class abydos.distance.**MarkingMetric**(***kwargs*)
Bases: abydos.distance._marking.Marking

Ehrenfeucht & Haussler's marking metric.

This metric [EH88] is the base 2 logarithm of the product of the marking distances between each term plus 1 computed in both orders. For strings x and y, this is:

$$dist_{MarkingMetric}(x, y) = log_2((diff(x, y) + 1)(diff(y, x) + 1))$$

The function diff is Ehrenfeucht & Haussler's marking distance *Marking*.

New in version 0.4.0.

Initialize MarkingMetric instance.

Parameters ****kwargs** -- Arbitrary keyword arguments

New in version 0.4.0.

dist(*src*, *tar*)
Return the normalized marking distance of two strings.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns marking distance

Return type float

Examples

```
>>> cmp = Marking()
>>> cmp.dist('cat', 'hat')
0.3333333333333333
>>> cmp.dist('Niall', 'Neil')
0.6
>>> cmp.dist('aluminum', 'Catalan')
0.625
>>> cmp.dist('ATCG', 'TAGC')
0.5
>>> cmp.dist('cbaabdcb', 'abcba')
0.25
```

New in version 0.4.0.

dist_abs (*src*, *tar*)

Return the marking distance of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns marking distance

Return type int

Examples

```
>>> cmp = MarkingMetric()
>>> cmp.dist_abs('cat', 'hat')
2.0
>>> cmp.dist_abs('Niall', 'Neil')
3.584962500721156
>>> cmp.dist_abs('aluminum', 'Catalan')
4.584962500721156
>>> cmp.dist_abs('ATCG', 'TAGC')
3.169925001442312
>>> cmp.dist_abs('cbaabdcb', 'abcba')
2.584962500721156
```

New in version 0.4.0.

class abydos.distance.**YujianBo**(*cost=(1, 1, 1, 1)*, ***kwargs*)

Bases: abydos.distance._levenshtein.Levenshtein

Yujian-Bo normalized Levenshtein distance.

Yujian-Bo's normalization of Levenshtein distance [YB07], given Levenshtein distance $GLD(X, Y)$ between two strings X and Y, is

$$dist_{N-GLD}(X, Y) = \frac{2 \cdot GLD(X, Y)}{|X| + |Y| + GLD(X, Y)}$$

New in version 0.4.0.

Initialize YujianBo instance.

Parameters ****kwargs** -- Arbitrary keyword arguments

New in version 0.4.0.

dist(*src*, *tar*)

Return the Yujian-Bo normalized edit distance between strings.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns The Yujian-Bo normalized edit distance between src & tar

Return type float

Examples

```
>>> cmp = YujianBo()
>>> round(cmp.dist('cat', 'hat'), 12)
0.285714285714
>>> round(cmp.dist('Niall', 'Neil'), 12)
0.5
>>> cmp.dist('aluminum', 'Catalan')
0.6363636363636364
>>> cmp.dist('ATCG', 'TAGC')
0.5454545454545454
```

New in version 0.4.0.

dist_abs(*src*, *tar*)

Return the Yujian-Bo normalized edit distance between two strings.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns The Yujian-Bo normalized edit distance between src & tar

Return type int

Examples

```
>>> cmp = YujianBo()
>>> cmp.dist_abs('cat', 'hat')
0.2857142857142857
>>> cmp.dist_abs('Niall', 'Neil')
0.5
>>> cmp.dist_abs('aluminum', 'Catalan')
0.6363636363636364
>>> cmp.dist_abs('ATCG', 'TAGC')
0.5454545454545454
```

New in version 0.4.0.

class abydos.distance.**HigueraMico**(***kwargs*)

Bases: abydos.distance._distance._Distance

The Higuera-Micó contextual normalized edit distance.

This is presented in [delHigueraMico08].

This measure is not normalized to a particular range. Indeed, for an string of infinite length as and a string of 0 length, the contextual normalized edit distance would be infinity. But so long as the relative difference in string lengths is not too great, the distance will generally remain below 1.0

Notes

The "normalized" version of this distance, implemented in the dist method is merely the minimum of the distance and 1.0.

New in version 0.4.0.

Initialize Levenshtein instance.

Parameters ****kwargs** -- Arbitrary keyword arguments

New in version 0.4.0.

dist(*src*, *tar*)

Return the bounded Higuera-Micó distance between two strings.

This is the distance bounded to the range [0, 1].

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns The bounded Higuera-Micó distance between src & tar

Return type float

Examples

```
>>> cmp = HigueraMico()
>>> cmp.dist('cat', 'hat')
0.3333333333333333
>>> cmp.dist('Niall', 'Neil')
0.5333333333333333
>>> cmp.dist('aluminum', 'Catalan')
0.7916666666666667
>>> cmp.dist('ATCG', 'TAGC')
0.6000000000000001
```

New in version 0.4.0.

dist_abs(*src*, *tar*)
Return the Higuera-Micó distance between two strings.

This is a straightforward implementation of Higuera & Micó pseudocode from [delHigueraMico08], ported to Numpy.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns The Higuera-Micó distance between src & tar

Return type float

Examples

```
>>> cmp = HigueraMico()
>>> cmp.dist_abs('cat', 'hat')
0.3333333333333333
>>> cmp.dist_abs('Niall', 'Neil')
0.5333333333333333
>>> cmp.dist_abs('aluminum', 'Catalan')
0.7916666666666667
>>> cmp.dist_abs('ATCG', 'TAGC')
0.6000000000000001
```

New in version 0.4.0.

class abydos.distance.**Indel**(***kwargs*)
Bases: abydos.distance._levenshtein.Levenshtein

Indel distance.

This is equivalent to Levenshtein distance, when only inserts and deletes are possible.

New in version 0.3.6.

Initialize Levenshtein instance.

Parameters ****kwargs** -- Arbitrary keyword arguments

New in version 0.4.0.

dist(*src*, *tar*)
Return the normalized indel distance between two strings.

This is equivalent to normalized Levenshtein distance, when only inserts and deletes are possible.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns Normalized indel distance

Return type float

Examples

```
>>> cmp = Indel()
>>> round(cmp.dist('cat', 'hat'), 12)
0.333333333333
>>> round(cmp.dist('Niall', 'Neil'), 12)
0.333333333333
>>> round(cmp.dist('Colin', 'Cuilen'), 12)
0.454545454545
>>> cmp.dist('ATCG', 'TAGC')
0.5
```

New in version 0.3.6.

abydos.distance.**indel**(*src*, *tar*)

Return the indel distance between two strings.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns Indel distance

Return type int

Examples

```
>>> indel('cat', 'hat')
2
>>> indel('Niall', 'Neil')
3
>>> indel('Colin', 'Cuilen')
5
>>> indel('ATCG', 'TAGC')
4
```

New in version 0.3.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the Indel.dist_abs method instead.

abydos.distance.**dist_indel**(*src*, *tar*)

Return the normalized indel distance between two strings.

This is equivalent to normalized Levenshtein distance, when only inserts and deletes are possible.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns Normalized indel distance

Return type float

Examples

```
>>> round(dist_indel('cat', 'hat'), 12)
0.333333333333
>>> round(dist_indel('Niall', 'Neil'), 12)
0.333333333333
>>> round(dist_indel('Colin', 'Cuilen'), 12)
0.454545454545
>>> dist_indel('ATCG', 'TAGC')
0.5
```

New in version 0.3.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the Indel.dist method instead.

abydos.distance.**sim_indel**(*src*, *tar*)

Return the normalized indel similarity of two strings.

This is equivalent to normalized Levenshtein similarity, when only inserts and deletes are possible.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns Normalized indel similarity

Return type float

Examples

```
>>> round(sim_indel('cat', 'hat'), 12)
0.666666666667
>>> round(sim_indel('Niall', 'Neil'), 12)
0.666666666667
>>> round(sim_indel('Colin', 'Cuilen'), 12)
0.545454545455
>>> sim_indel('ATCG', 'TAGC')
0.5
```

New in version 0.3.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the Indel.sim method instead.

class abydos.distance.**SAPS**(*cost=(1, -1, -4, 6, -2, -1, -3)*, *normalizer=<built-in function max>*, *tokenizer=None*, ***kwargs*)

Bases: abydos.distance._distance._Distance

Syllable Alignment Pattern Searching tokenizer.

This is the alignment and similarity calculation described on p. 917-918 of [RY05].

New in version 0.4.0.

Initialize SAPS instance.

Parameters

- **cost** (*tuple*) -- A 7-tuple representing the cost of the four possible matches:
  - syllable-internal match
  - syllable-internal mis-match
  - syllable-initial match or mismatch with syllable-internal
  - syllable-initial match
  - syllable-initial mis-match
  - syllable-internal gap
  - syllable-initial gap

  (by default: (1, -1, -4, 6, -2, -1, -3))
- **normalizer** (*function*) -- A function that takes an list and computes a normalization term by which the edit distance is divided (max by default). Another good option is the sum function.
- ****kwargs** -- Arbitrary keyword arguments

New in version 0.4.0.

sim(*src*, *tar*)

Return the normalized SAPS similarity between two strings.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns The normalized SAPS similarity between src & tar

Return type float

Examples

```
>>> cmp = SAPS()
>>> round(cmp.sim('cat', 'hat'), 12)
0.0
>>> round(cmp.sim('Niall', 'Neil'), 12)
0.2
>>> cmp.sim('aluminum', 'Catalan')
0.0
>>> cmp.sim('ATCG', 'TAGC')
0.0
```

New in version 0.4.0.

sim_score(*src*, *tar*)

Return the SAPS similarity between two strings.

Parameters

- **src** (*str*) -- Source string for comparison

- **tar** (*str*) -- Target string for comparison

Returns The SAPS similarity between src & tar

Return type int

Examples

```
>>> cmp = SAPS()
>>> cmp.sim_score('cat', 'hat')
0
>>> cmp.sim_score('Niall', 'Neil')
3
>>> cmp.sim_score('aluminum', 'Catalan')
-11
>>> cmp.sim_score('ATCG', 'TAGC')
-1
>>> cmp.sim_score('Stevenson', 'Stinson')
16
```

New in version 0.4.0.

class abydos.distance.**MetaLevenshtein**(*tokenizer=None*, *corpus=None*, *metric=None*, *normalizer=<built-in function max>*, ***kwargs*)

Bases: abydos.distance._distance._Distance

Meta-Levenshtein distance.

Meta-Levenshtein distance [MYCappe08] combines Soft-TFIDF with Levenshtein alignment.

New in version 0.4.0.

Initialize MetaLevenshtein instance.

Parameters

- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **corpus** (UnigramCorpus) -- A unigram corpus UnigramCorpus. If None, a corpus will be created from the two words when a similarity function is called.
- **metric** (*_Distance*) -- A string distance measure class for making soft matches, by default Jaro-Winkler.
- **normalizer** (*function*) -- A function that takes an list and computes a normalization term by which the edit distance is divided (max by default). Another good option is the sum function.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.

New in version 0.4.0.

dist(*src*, *tar*)

Return the normalized Levenshtein distance between two strings.

The Levenshtein distance is normalized by dividing the Levenshtein distance (calculated by any of the three supported methods) by the greater of the number of characters in src times the cost of a delete and the number of characters in tar times the cost of an insert. For the case in which all operations have $cost = 1$, this is equivalent to the greater of the length of the two strings src & tar.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns The normalized Levenshtein distance between src & tar

Return type float

Examples

```
>>> cmp = MetaLevenshtein()
>>> round(cmp.dist('cat', 'hat'), 12)
0.205186754296
>>> round(cmp.dist('Niall', 'Neil'), 12)
0.507780131444
>>> cmp.dist('aluminum', 'Catalan')
0.8675933954313434
>>> cmp.dist('ATCG', 'TAGC')
0.8077801314441113
```

New in version 0.1.0.

Changed in version 0.3.6: Encapsulated in class

dist_abs(*src*, *tar*)

Return the Meta-Levenshtein distance of two strings.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns Meta-Levenshtein distance

Return type float

Examples

```
>>> cmp = MetaLevenshtein()
>>> cmp.dist_abs('cat', 'hat')
0.6155602628882225
>>> cmp.dist_abs('Niall', 'Neil')
2.538900657220556
>>> cmp.dist_abs('aluminum', 'Catalan')
6.940747163450747
>>> cmp.dist_abs('ATCG', 'TAGC')
3.2311205257764453
```

New in version 0.4.0.

class abydos.distance.**Covington**(*weights=(0, 5, 10, 30, 60, 100, 40, 50)*, ***kwargs*)

Bases: abydos.distance._distance._Distance

Covington distance.

Covington distance [Cov96]

New in version 0.4.0.

Initialize Covington instance.

Parameters

- **weights** (*tuple*) -- An 8-tuple of costs for each kind of match or mismatch described in Covington's paper:
  - exact consonant or glide match
  - exact vowel match
  - vowel-vowel length mismatch or i and y or u and w
  - vowel-vowel mismatch
  - consonant-consonant mismatch
  - consonant-vowel mismatch
  - skip preceded by a skip
  - skip not preceded by a skip

  The weights used in Covington's first approximation can be used by supplying the tuple (0.0, 0.0, 0.5, 0.5, 0.5, 1.0, 0.5, 0.5)
- ****kwargs** -- Arbitrary keyword arguments

New in version 0.4.0.

alignment(*src*, *tar*)

Return the top Covington alignment of two strings.

This returns only the top alignment in a standard (score, source alignment, target alignment) tuple format.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns Covington score & alignment

Return type tuple(float, str, str)

Examples

```
>>> cmp = Covington()
>>> cmp.alignment('hart', 'kordis')
(240, 'hart--', 'kordis')
>>> cmp.alignment('niy', 'genu')
(170, '--niy', 'genu-')
```

New in version 0.4.1.

alignments(*src*, *tar*, *top_n=None*)

Return the Covington alignments of two strings.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison
- **top_n** (*int*) -- The number of alignments to return. If None, all alignments will be returned. If 0, all alignments with the top score will be returned.

Returns Covington alignments

Return type list

Examples

```
>>> cmp = Covington()
>>> cmp.alignments('hart', 'kordis', top_n=1)[0]
Alignment(src='hart--', tar='kordis', score=240)
>>> cmp.alignments('niy', 'genu', top_n=1)[0]
Alignment(src='--niy', tar='genu-', score=170)
```

New in version 0.4.0.

dist (*src*, *tar*)

Return the normalized Covington distance of two strings.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns Normalized Covington distance

Return type float

Examples

```
>>> cmp = Covington()
>>> cmp.dist('cat', 'hat')
0.19117647058823528
>>> cmp.dist('Niall', 'Neil')
0.25555555555555554
>>> cmp.dist('aluminum', 'Catalan')
0.43333333333333335
>>> cmp.dist('ATCG', 'TAGC')
0.45454545454545453
```

New in version 0.4.0.

dist_abs (*src*, *tar*)

Return the Covington distance of two strings.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns Covington distance

Return type float

Examples

```
>>> cmp = Covington()
>>> cmp.dist_abs('cat', 'hat')
65
>>> cmp.dist_abs('Niall', 'Neil')
115
>>> cmp.dist_abs('aluminum', 'Catalan')
325
>>> cmp.dist_abs('ATCG', 'TAGC')
200
```

New in version 0.4.0.

class abydos.distance.**ALINE**(*epsilon=0*, *c_skip=-10*, *c_sub=35*, *c_exp=45*, *c_vwl=10*, *mode='local'*, *phones='aline'*, *normalizer=<built-in function max>*, ***kwargs*)

Bases: abydos.distance._distance._Distance

ALINE alignment, similarity, and distance.

ALINE alignment was developed by [Kon00][Kon02][DHC+08], and establishes an alignment algorithm based on multivalued phonetic features and feature salience weights. Along with the alignment itself, the algorithm produces a term similarity score.

[DHC+08] develops ALINE's similarity score into a similarity measure & distance measure:

$$sim_{ALINE} = \frac{2\dot{s}core_{ALINE}(src, tar)}{score_{ALINE}(src, src) + score_{ALINE}(tar, tar)}$$

However, because the average of the two self-similarity scores is not guaranteed to be greater than or equal to the similarity score between the two strings, by default, this formula is not used here in order to guarantee that the similarity measure is bounded to [0, 1]. Instead, Kondrak's similarity measure is employed:

$$sim_{ALINE} = \frac{score_{ALINE}(src, tar)}{max(score_{ALINE}(src, src), score_{ALINE}(tar, tar))}$$

New in version 0.4.0.

Initialize ALINE instance.

Parameters

- **epsilon** (*float*) -- The portion (out of 1.0) of the maximum ALINE score, above which alignments are returned. If set to 0, only the alignments matching the maximum alignment score are returned. If set to 1, all alignments scoring 0 or higher are returned.
- **c_skip** (*int*) -- The cost of an insertion or deletion
- **c_sub** (*int*) -- The cost of a substitution
- **c_exp** (*int*) -- The cost of an expansion or contraction
- **c_vwl** (*int*) -- The additional cost of a vowel substitution, expansion, or contraction
- **mode** (*str*) -- Alignment mode, which can be `local` (default), `global`, `half-local`, or `semi-global`
- **phones** (*str*) --

Phonetic symbol set, which can be:

- `aline` selects Kondrak's original symbols set
- `ipa` selects IPA symbols

- **normalizer** (*function*) -- A function that takes an list and computes a normalization term by which the edit distance is divided (max by default). For the normalization proposed by Downey, et al. (2008), set this to: `lambda x: sum(x)/len(x)`
- ****kwargs** -- Arbitrary keyword arguments

New in version 0.4.0.

alignment(*src*, *tar*)

Return the top ALINE alignment of two strings.

The *top* ALINE alignment is the first alignment with the best score. The purpose of this function is to have a single tuple as a return value.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns ALINE alignment and its score

Return type tuple(float, str, str)

Examples

```
>>> cmp = ALINE()
>>> cmp.alignment('cat', 'hat')
(50.0, 'c || a t ||', 'h || a t ||')
>>> cmp.alignment('niall', 'neil')
(90.0, '|| n i a ll ||', '|| n e i l  ||')
>>> cmp.alignment('aluminum', 'catalan')
(81.5, '|| a l u m || inum', 'cat || a l a n ||')
>>> cmp.alignment('atcg', 'tagc')
(65.0, '|| a t c || g', 't || a g c ||')
```

New in version 0.4.1.

alignments(*src*, *tar*, *score_only=False*)

Return the ALINE alignments of two strings.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison
- **score_only** (*bool*) -- Return the score only, not the alignments

Returns ALINE alignments and their scores or the top score

Return type list(tuple(float, str, str) or float

Examples

```
>>> cmp = ALINE()
>>> cmp.alignments('cat', 'hat')
[(50.0, 'c || a t ||', 'h || a t ||')]
>>> cmp.alignments('niall', 'neil')
[(90.0, '|| n i a ll ||', '|| n e i l  ||')]
>>> cmp.alignments('aluminum', 'catalan')
[(81.5, '|| a l u m || inum', 'cat || a l a n ||')]
>>> cmp.alignments('atcg', 'tagc')
[(65.0, '|| a t c || g', 't || a g c ||'), (65.0, 'a || tc - g ||',
'|| t  a g || c')]
```

New in version 0.4.0.

Changed in version 0.4.1: Renamed from .alignment to .alignments

c_features = {'aspirated', 'lateral', 'manner', 'nasal', 'place', 'retroflex', 'syllab

feature_weights = {'affricate': 0.9, 'alveolar': 0.85, 'approximant': 0.6, 'back':

minus', 'place': phones_ipa = {'a': {'aspirated': 'minus', 'back': 'minus', 'high': 'minus', 'lateral': 'minus', 'long': 'minus', 'nasal': 'minus', 'retroflex': 'minus', 'round': 'minus', 'syllabic': 'plus', 'voice': 'plus'}, 'b': {'aspirated': 'minus', 'back': 'minus', 'high': 'minus', 'lateral': 'minus', 'long': 'minus', 'nasal': 'minus', 'retroflex': 'minus', 'round': 'minus', 'syllabic': 'minus', 'voice': 'plus'}, 'c': {'aspirated': 'minus', 'back': 'minus', 'high': 'minus', 'lateral': 'minus',

phones_kondrak = {'A': {'aspirated': 'plus', 'supplemental': True}, 'B': {'back': '

salience = {'aspirated': 5, 'back': 5, 'high': 5, 'lateral': 10, 'long': 1, 'mann

sim(*src*, *tar*)

Return the normalized ALINE similarity of two strings.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns Normalized ALINE similarity

Return type float

Examples

```
>>> cmp = ALINE()
>>> cmp.dist('cat', 'hat')
0.4117647058823529
>>> cmp.dist('niall', 'neil')
0.3333333333333333
>>> cmp.dist('aluminum', 'catalan')
0.5925
>>> cmp.dist('atcg', 'tagc')
0.45833333333333337
```

New in version 0.4.0.

sim_score(*src*, *tar*)

Return the ALINE alignment score of two strings.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns ALINE alignment score

Return type float

Examples

```
>>> cmp = ALINE()
>>> cmp.sim_score('cat', 'hat')
50.0
>>> cmp.sim_score('niall', 'neil')
90.0
>>> cmp.sim_score('aluminum', 'catalan')
81.5
>>> cmp.sim_score('atcg', 'tagc')
65.0
```

New in version 0.4.0.

v_features = {'back', 'high', 'long', 'nasal', 'retroflex', 'round', 'syllabic'}

class abydos.distance.**FlexMetric**(*normalizer=<built-in function max>*, *indel_costs=None*, *subst_costs=None*, ***kwargs*)

Bases: abydos.distance._distance._Distance

FlexMetric distance.

FlexMetric distance [Kem05]

New in version 0.4.0.

Initialize FlexMetric instance.

Parameters

- **normalizer** (*function*) -- A function that takes an list and computes a normalization term by which the edit distance is divided (max by default). Another good option is the sum function.
- **indel_costs** (*list of tuples*) -- A list of insertion and deletion costs. Each list element should be a tuple consisting of an iterable (sets are best) and a float value. The iterable consists of those letters whose insertion or deletion has a cost equal to the float value.
- **subst_costs** (*list of tuples*) -- A list of substitution costs. Each list element should be a tuple consisting of an iterable (sets are best) and a float value. The iterable consists of the letters in each letter class, which may be substituted for each other at cost equal to the float value.
- ****kwargs** -- Arbitrary keyword arguments

New in version 0.4.0.

dist(*src*, *tar*)

Return the normalized FlexMetric distance of two strings.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns Normalized FlexMetric distance

Return type float

Examples

```
>>> cmp = FlexMetric()
>>> cmp.dist('cat', 'hat')
0.2666666666666666
>>> cmp.dist('Niall', 'Neil')
0.3
>>> cmp.dist('aluminum', 'Catalan')
0.8375
>>> cmp.dist('ATCG', 'TAGC')
0.5499999999999999
```

New in version 0.4.0.

dist_abs(*src*, *tar*)

Return the FlexMetric distance of two strings.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns FlexMetric distance

Return type float

Examples

```
>>> cmp = FlexMetric()
>>> cmp.dist_abs('cat', 'hat')
0.8
>>> cmp.dist_abs('Niall', 'Neil')
1.5
>>> cmp.dist_abs('aluminum', 'Catalan')
6.7
>>> cmp.dist_abs('ATCG', 'TAGC')
2.1999999999999997
```

New in version 0.4.0.

class abydos.distance.**BISIM**(*qval=2*, ***kwargs*)

Bases: abydos.distance._distance._Distance

BI-SIM similarity.

BI-SIM similarity [KD03] is an n-gram based, edit-distance derived similarity measure.

New in version 0.4.0.

Initialize BISIM instance.

Parameters

- **qval** (*int*) -- The number of characters to consider in each n-gram (q-gram). By default this is 2, hence BI-SIM. But TRI-SIM can be calculated by setting this to 3.
- ****kwargs** -- Arbitrary keyword arguments

New in version 0.4.0.

sim(*src*, *tar*)

Return the BI-SIM similarity of two strings.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns BI-SIM similarity

Return type float

Examples

```
>>> cmp = BISIM()
>>> cmp.sim('cat', 'hat')
0.5
>>> cmp.sim('Niall', 'Neil')
0.4
>>> cmp.sim('aluminum', 'Catalan')
0.3125
>>> cmp.sim('ATCG', 'TAGC')
0.375
```

New in version 0.4.0.

class abydos.distance.**DiscountedLevenshtein**(*mode='lev'*, *normalizer=<built-in function max>*, *discount_from=1*, *discount_func='log'*, *vowels='aeiou'*, ***kwargs*)

Bases: abydos.distance._levenshtein.Levenshtein

Discounted Levenshtein distance.

This is a variant of Levenshtein distance for which edits later in a string have discounted cost, on the theory that earlier edits are less likely than later ones.

New in version 0.4.1.

Initialize DiscountedLevenshtein instance.

Parameters

- **mode** (*str*) -- Specifies a mode for computing the discounted Levenshtein distance:
  - lev (default) computes the ordinary Levenshtein distance, in which edits may include inserts, deletes, and substitutions
  - osa computes the Optimal String Alignment distance, in which edits may include inserts, deletes, substitutions, and transpositions but substrings may only be edited once
- **normalizer** (*function*) -- A function that takes an list and computes a normalization term by which the edit distance is divided (max by default). Another good option is the sum function.
- **discount_from** (*int or str*) -- If an int is supplied, this is the first character whose edit cost will be discounted. If the str coda is supplied, discounting will start with the first non-vowel after the first vowel (the first syllable coda).

- **discount_func** (*str or function*) -- The two supported str arguments are `log`, for a logarithmic discount function, and `exp` for a exponential discount function. See notes below for information on how to supply your own discount function.
- **vowels** (*str*) -- These are the letters to consider as vowels when discount_from is set to `coda`. It defaults to the English vowels 'aeiou', but it would be reasonable to localize this to other languages or to add orthographic semi-vowels like 'y', 'w', and even 'h'.
- ****kwargs** -- Arbitrary keyword arguments

Notes

This class is highly experimental and will need additional tuning.

The discount function can be passed as a callable function. It should expect an integer as its only argument and return a float, ideally less than or equal to 1.0. The argument represents the degree of discounting to apply.

New in version 0.4.1.

dist(*src*, *tar*)

Return the normalized Levenshtein distance between two strings.

The Levenshtein distance is normalized by dividing the Levenshtein distance (calculated by any of the three supported methods) by the greater of the number of characters in src times the cost of a delete and the number of characters in tar times the cost of an insert. For the case in which all operations have $cost = 1$, this is equivalent to the greater of the length of the two strings src & tar.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns The normalized Levenshtein distance between src & tar

Return type float

Examples

```
>>> cmp = DiscountedLevenshtein()
>>> cmp.dist('cat', 'hat')
0.3513958291799864
>>> cmp.dist('Niall', 'Neil')
0.5909885886270658
>>> cmp.dist('aluminum', 'Catalan')
0.8348163322045603
>>> cmp.dist('ATCG', 'TAGC')
0.7217609721523955
```

New in version 0.4.1.

dist_abs(*src*, *tar*)

Return the Levenshtein distance between two strings.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns The Levenshtein distance between src & tar

Return type float (may return a float if cost has float values)

Examples

```
>>> cmp = DiscountedLevenshtein()
>>> cmp.dist_abs('cat', 'hat')
1
>>> cmp.dist_abs('Niall', 'Neil')
2.526064024369237
>>> cmp.dist_abs('aluminum', 'Catalan')
5.053867269967515
>>> cmp.dist_abs('ATCG', 'TAGC')
2.594032108779918
```

```
>>> cmp = DiscountedLevenshtein(mode='osa')
>>> cmp.dist_abs('ATCG', 'TAGC')
1.7482385137517997
>>> cmp.dist_abs('ACTG', 'TAGC')
3.342270622531718
```

New in version 0.4.1.

class abydos.distance.**PhoneticEditDistance**(*mode='lev'*, *cost=(1, 1, 1, 0.33333)*, *normalizer=<built-in function max>*, *weights=None*, ***kwargs*)

Bases: abydos.distance._levenshtein.Levenshtein

Phonetic edit distance.

This is a variation on Levenshtein edit distance, intended for strings in IPA, that compares individual phones based on their featural similarity.

New in version 0.4.1.

Initialize PhoneticEditDistance instance.

Parameters

- **mode** (*str*) -- Specifies a mode for computing the edit distance:
  - `lev` (default) computes the ordinary Levenshtein distance, in which edits may include inserts, deletes, and substitutions
  - `osa` computes the Optimal String Alignment distance, in which edits may include inserts, deletes, substitutions, and transpositions but substrings may only be edited once
- **cost** (*tuple*) -- A 4-tuple representing the cost of the four possible edits: inserts, deletes, substitutions, and transpositions, respectively (by default: (1, 1, 1, 0.33333)). Note that transpositions cost a relatively low 0.33333. If this were 1.0, no phones would ever be transposed under the normal weighting, since even quite dissimilar phones such as [a] and [p] still agree in nearly 63% of their features.
- **normalizer** (*function*) -- A function that takes an list and computes a normalization term by which the edit distance is divided (max by default). Another good option is the sum function.
- **weights** (*None or list or tuple or dict*) -- If None, all features are of equal significance and a simple normalized hamming distance of the features is calculated. If a list or tuple of numeric values is supplied, the values are inferred as the weights for each feature, in order of the features listed in abydos.phones._phones._FEATURE_MASK. If a dict is

supplied, its key values should match keys in abydos.phones._phones._FEATURE_MASK to which each weight (value) should be assigned. Missing values in all cases are assigned a weight of 0 and will be omitted from the comparison.

- ****kwargs** -- Arbitrary keyword arguments

New in version 0.4.1.

dist(*src*, *tar*)

Return the normalized phonetic edit distance between two strings.

The edit distance is normalized by dividing the edit distance (calculated by either of the two supported methods) by the greater of the number of characters in src times the cost of a delete and the number of characters in tar times the cost of an insert. For the case in which all operations have $cost = 1$, this is equivalent to the greater of the length of the two strings src & tar.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns The normalized Levenshtein distance between src & tar

Return type float

Examples

```
>>> cmp = PhoneticEditDistance()
>>> round(cmp.dist('cat', 'hat'), 12)
0.059139784946
>>> round(cmp.dist('Niall', 'Neil'), 12)
0.232258064516
>>> cmp.dist('aluminum', 'Catalan')
0.3084677419354839
>>> cmp.dist('ATCG', 'TAGC')
0.2983870967741935
```

New in version 0.4.1.

dist_abs(*src*, *tar*)

Return the phonetic edit distance between two strings.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns The phonetic edit distance between src & tar

Return type int (may return a float if cost has float values)

Examples

```
>>> cmp = PhoneticEditDistance()
>>> cmp.dist_abs('cat', 'hat')
0.17741935483870974
>>> cmp.dist_abs('Niall', 'Neil')
1.161290322580645
>>> cmp.dist_abs('aluminum', 'Catalan')
2.467741935483871
>>> cmp.dist_abs('ATCG', 'TAGC')
1.193548387096774
```

```
>>> cmp = PhoneticEditDistance(mode='osa')
>>> cmp.dist_abs('ATCG', 'TAGC')
0.46236225806451603
>>> cmp.dist_abs('ACTG', 'TAGC')
1.2580645161290323
```

New in version 0.4.1.

class abydos.distance.**Hamming**(*diff_lens=True*, ***kwargs*)

Bases: abydos.distance._distance._Distance

Hamming distance.

Hamming distance [Ham50] equals the number of character positions at which two strings differ. For strings of unequal lengths, it is not normally defined. By default, this implementation calculates the Hamming distance of the first n characters where n is the lesser of the two strings' lengths and adds to this the difference in string lengths.

New in version 0.3.6.

Initialize Hamming instance.

Parameters

- **diff_lens** (*bool*) -- If True (default), this returns the Hamming distance for those characters that have a matching character in both strings plus the difference in the strings' lengths. This is equivalent to extending the shorter string with obligatorily non-matching characters. If False, an exception is raised in the case of strings of unequal lengths.
- ****kwargs** -- Arbitrary keyword arguments

New in version 0.4.0.

dist(*src*, *tar*)

Return the normalized Hamming distance between two strings.

Hamming distance normalized to the interval [0, 1].

The Hamming distance is normalized by dividing it by the greater of the number of characters in src & tar (unless diff_lens is set to False, in which case an exception is raised).

The arguments are identical to those of the hamming() function.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns Normalized Hamming distance

Return type float

Examples

```
>>> cmp = Hamming()
>>> round(cmp.dist('cat', 'hat'), 12)
0.333333333333
>>> cmp.dist('Niall', 'Neil')
0.6
>>> cmp.dist('aluminum', 'Catalan')
1.0
>>> cmp.dist('ATCG', 'TAGC')
1.0
```

New in version 0.1.0.

Changed in version 0.3.6: Encapsulated in class

dist_abs(*src*, *tar*)

Return the Hamming distance between two strings.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns The Hamming distance between src & tar

Return type int

Raises **ValueError** -- Undefined for sequences of unequal length; set diff_lens to True for Hamming distance between strings of unequal lengths.

Examples

```
>>> cmp = Hamming()
>>> cmp.dist_abs('cat', 'hat')
1
>>> cmp.dist_abs('Niall', 'Neil')
3
>>> cmp.dist_abs('aluminum', 'Catalan')
8
>>> cmp.dist_abs('ATCG', 'TAGC')
4
```

New in version 0.1.0.

Changed in version 0.3.6: Encapsulated in class

abydos.distance.**hamming**(*src*, *tar*, *diff_lens=True*)

Return the Hamming distance between two strings.

This is a wrapper for *Hamming.dist_abs()*.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

- **diff_lens** (*bool*) -- If True (default), this returns the Hamming distance for those characters that have a matching character in both strings plus the difference in the strings' lengths. This is equivalent to extending the shorter string with obligatorily non-matching characters. If False, an exception is raised in the case of strings of unequal lengths.

Returns The Hamming distance between src & tar

Return type int

Examples

```
>>> hamming('cat', 'hat')
1
>>> hamming('Niall', 'Neil')
3
>>> hamming('aluminum', 'Catalan')
8
>>> hamming('ATCG', 'TAGC')
4
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the Hamming.dist_abs method instead.

abydos.distance.**dist_hamming**(*src*, *tar*, *diff_lens=True*)

Return the normalized Hamming distance between two strings.

This is a wrapper for *Hamming.dist()*.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison
- **diff_lens** (*bool*) -- If True (default), this returns the Hamming distance for those characters that have a matching character in both strings plus the difference in the strings' lengths. This is equivalent to extending the shorter string with obligatorily non-matching characters. If False, an exception is raised in the case of strings of unequal lengths.

Returns The normalized Hamming distance

Return type float

Examples

```
>>> round(dist_hamming('cat', 'hat'), 12)
0.333333333333
>>> dist_hamming('Niall', 'Neil')
0.6
>>> dist_hamming('aluminum', 'Catalan')
1.0
>>> dist_hamming('ATCG', 'TAGC')
1.0
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the Hamming.dist method instead.

abydos.distance.**sim_hamming**(*src*, *tar*, *diff_lens=True*)

Return the normalized Hamming similarity of two strings.

This is a wrapper for `Hamming.sim()`.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison
- **diff_lens** (*bool*) -- If True (default), this returns the Hamming distance for those characters that have a matching character in both strings plus the difference in the strings' lengths. This is equivalent to extending the shorter string with obligatorily non-matching characters. If False, an exception is raised in the case of strings of unequal lengths.

Returns The normalized Hamming similarity

Return type float

Examples

```
>>> round(sim_hamming('cat', 'hat'), 12)
0.666666666667
>>> sim_hamming('Niall', 'Neil')
0.4
>>> sim_hamming('aluminum', 'Catalan')
0.0
>>> sim_hamming('ATCG', 'TAGC')
0.0
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the Hamming.sim method instead.

class abydos.distance.**MLIPNS**(*threshold=0.25*, *max_mismatches=2*, ***kwargs*)

Bases: abydos.distance._distance._Distance

MLIPNS similarity.

Modified Language-Independent Product Name Search (MLIPNS) is described in [SA10]. This function returns only 1.0 (similar) or 0.0 (not similar). LIPNS similarity is identical to normalized Hamming similarity.

New in version 0.3.6.

Initialize MLIPNS instance.

Parameters

- **threshold** (*float*) -- A number [0, 1] indicating the maximum similarity score, below which the strings are considered 'similar' (0.25 by default)
- **max_mismatches** (*int*) -- A number indicating the allowable number of mismatches to remove before declaring two strings not similar (2 by default)
- ****kwargs** -- Arbitrary keyword arguments

New in version 0.4.0.

sim(*src*, *tar*)

Return the MLIPNS similarity of two strings.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns MLIPNS similarity

Return type float

Examples

```
>>> sim_mlipns('cat', 'hat')
1.0
>>> sim_mlipns('Niall', 'Neil')
0.0
>>> sim_mlipns('aluminum', 'Catalan')
0.0
>>> sim_mlipns('ATCG', 'TAGC')
0.0
```

New in version 0.1.0.

Changed in version 0.3.6: Encapsulated in class

abydos.distance.**dist_mlipns**(*src*, *tar*, *threshold=0.25*, *max_mismatches=2*)

Return the MLIPNS distance between two strings.

This is a wrapper for `MLIPNS.dist()`.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison
- **threshold** (*float*) -- A number [0, 1] indicating the maximum similarity score, below which the strings are considered 'similar' (0.25 by default)
- **max_mismatches** (*int*) -- A number indicating the allowable number of mismatches to remove before declaring two strings not similar (2 by default)

Returns MLIPNS distance

Return type float

Examples

```
>>> dist_mlipns('cat', 'hat')
0.0
>>> dist_mlipns('Niall', 'Neil')
1.0
>>> dist_mlipns('aluminum', 'Catalan')
1.0
>>> dist_mlipns('ATCG', 'TAGC')
1.0
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the MLIPNS.dist method instead.

abydos.distance.**sim_mlipns**(*src*, *tar*, *threshold=0.25*, *max_mismatches=2*)

Return the MLIPNS similarity of two strings.

This is a wrapper for *MLIPNS.sim()*.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison
- **threshold** (*float*) -- A number [0, 1] indicating the maximum similarity score, below which the strings are considered 'similar' (0.25 by default)
- **max_mismatches** (*int*) -- A number indicating the allowable number of mismatches to remove before declaring two strings not similar (2 by default)

Returns MLIPNS similarity

Return type float

Examples

```
>>> sim_mlipns('cat', 'hat')
1.0
>>> sim_mlipns('Niall', 'Neil')
0.0
>>> sim_mlipns('aluminum', 'Catalan')
0.0
>>> sim_mlipns('ATCG', 'TAGC')
0.0
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the MLIPNS.sim method instead.

class abydos.distance.**RelaxedHamming**(*tokenizer=None*, *maxdist=2*, *discount=0.2*, ***kwargs*)

Bases: abydos.distance._distance._Distance

Relaxed Hamming distance.

This is a variant of Hamming distance in which positionally close matches are considered partially matching.

New in version 0.4.1.

Initialize DiscountedHamming instance.

Parameters

- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **maxdist** (*int*) -- The maximum distance to consider for discounting.
- **discount** (*float*) -- The discount factor multiplied by the distance from the source string position.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.

New in version 0.4.1.

dist(*src*, *tar*)

Return the normalized relaxed Hamming distance between strings.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns Normalized relaxed Hamming distance

Return type float

Examples

```
>>> cmp = RelaxedHamming()
>>> round(cmp.dist('cat', 'hat'), 12)
0.333333333333
>>> cmp.dist('Niall', 'Neil')
0.27999999999999997
>>> cmp.dist('aluminum', 'Catalan')
0.8
>>> cmp.dist('ATCG', 'TAGC')
0.2
```

New in version 0.4.1.

dist_abs(*src*, *tar*)

Return the discounted Hamming distance between two strings.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns Relaxed Hamming distance

Return type float

Examples

```
>>> cmp = RelaxedHamming()
>>> cmp.dist_abs('cat', 'hat')
1.0
>>> cmp.dist_abs('Niall', 'Neil')
1.4
>>> cmp.dist_abs('aluminum', 'Catalan')
6.4
>>> cmp.dist_abs('ATCG', 'TAGC')
0.8
```

New in version 0.4.1.

class abydos.distance.**Tichy**(*cost=(1, 1)*, ***kwargs*)

Bases: abydos.distance._distance._Distance

Tichy edit distance.

Tichy described an algorithm, implemented below, in [Tic84]. Following this, [Cor03] identifies an interpretation of this algorithm's output as a distance measure, which is largely followed by the methods below.

Tichy's algorithm locates substrings of a string S to be copied in order to create a string T. The only other operation used by his algorithms for string reconstruction are add operations.

Notes

While [Cor03] counts only move operations to calculate distance, I give the option (enabled by default) of counting add operations as part of the distance measure. To ignore the cost of add operations, set the cost value to (1, 0), for example, when initializing the object. Further, in the case that S and T are identical, a distance of 0 will be returned, even though this would still be counted as a single move operation spanning the whole of string S.

New in version 0.4.0.

Initialize Tichy instance.

Parameters

- **cost** (*tuple*) -- A 2-tuple representing the cost of the two possible edits: block moves and adds (by default: (1, 1))
- ****kwargs** -- Arbitrary keyword arguments

New in version 0.4.0.

dist(*src*, *tar*)

Return the normalized Tichy edit distance between two strings.

The Tichy distance is normalized by dividing the distance by the length of the tar string.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns The normalized Tichy distance between src & tar

Return type float

Examples

```
>>> cmp = Tichy()
>>> round(cmp.dist('cat', 'hat'), 12)
0.666666666667
>>> round(cmp.dist('Niall', 'Neil'), 12)
1.0
>>> cmp.dist('aluminum', 'Catalan')
0.8571428571428571
>>> cmp.dist('ATCG', 'TAGC')
1.0
```

New in version 0.4.0.

dist_abs(*src*, *tar*)

Return the Tichy distance between two strings.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns The Tichy distance between src & tar

Return type int (may return a float if cost has float values)

Examples

```
>>> cmp = Tichy()
>>> cmp.dist_abs('cat', 'hat')
2
>>> cmp.dist_abs('Niall', 'Neil')
4
>>> cmp.dist_abs('aluminum', 'Catalan')
6
>>> cmp.dist_abs('ATCG', 'TAGC')
4
```

New in version 0.4.0.

class abydos.distance.**BlockLevenshtein**(*cost=(1, 1, 1, 1)*, *normalizer=<built-in function max>*, ***kwargs*)

Bases: abydos.distance._levenshtein.Levenshtein

Levenshtein distance with block operations.

In addition to character-level insert, delete, and replace operations, this version of the Levenshtein distance supports block-level insert, delete, and replace, provided that the block occurs in both input strings.

New in version 0.4.0.

Initialize BlockLevenshtein instance.

Parameters ****kwargs** -- Arbitrary keyword arguments

New in version 0.4.0.

dist(*src*, *tar*)

Return the normalized block Levenshtein distance between strings.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns The normalized Levenshtein distance with blocks between src & tar

Return type float

Examples

```
>>> cmp = BlockLevenshtein()
>>> round(cmp.dist('cat', 'hat'), 12)
0.333333333333
>>> round(cmp.dist('Niall', 'Neil'), 12)
0.6
>>> cmp.dist('aluminum', 'Catalan')
0.875
>>> cmp.dist('ATCG', 'TAGC')
0.75
```

New in version 0.4.0.

dist_abs(*src*, *tar*)

Return the block Levenshtein edit distance between two strings.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns The block Levenshtein edit distance between src & tar

Return type int

Examples

```
>>> cmp = BlockLevenshtein()
>>> cmp.dist_abs('cat', 'hat')
1
>>> cmp.dist_abs('Niall', 'Neil')
3
>>> cmp.dist_abs('aluminum', 'Catalan')
7
>>> cmp.dist_abs('ATCG', 'TAGC')
3
```

New in version 0.4.0.

class abydos.distance.**CormodeLZ**(***kwargs*)

Bases: abydos.distance._distance._Distance

Cormode's LZ distance.

Cormode's LZ distance [CPSV00][Cor03]

New in version 0.4.0.

Initialize CormodeLZ instance.

Parameters ****kwargs** -- Arbitrary keyword arguments

New in version 0.4.0.

dist(*src*, *tar*)

Return the normalized Cormode's LZ distance of two strings.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns Cormode's LZ distance

Return type float

Examples

```
>>> cmp = CormodeLZ()
>>> cmp.dist('cat', 'hat')
0.3333333333333333
>>> cmp.dist('Niall', 'Neil')
0.8
>>> cmp.dist('aluminum', 'Catalan')
0.625
>>> cmp.dist('ATCG', 'TAGC')
0.75
```

New in version 0.4.0.

dist_abs(*src*, *tar*)

Return the Cormode's LZ distance of two strings.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns Cormode's LZ distance

Return type float

Examples

```
>>> cmp = CormodeLZ()
>>> cmp.dist_abs('cat', 'hat')
2
>>> cmp.dist_abs('Niall', 'Neil')
5
>>> cmp.dist_abs('aluminum', 'Catalan')
6
>>> cmp.dist_abs('ATCG', 'TAGC')
4
```

New in version 0.4.0.

class abydos.distance.**JaroWinkler**(*qval=1*, *mode='winkler'*, *long_strings=False*, *boost_threshold=0.7*, *scaling_factor=0.1*, ***kwargs*)

Bases: abydos.distance._distance._Distance

Jaro-Winkler distance.

Jaro(-Winkler) distance is a string edit distance initially proposed by Jaro and extended by Winkler [Jar89][Win90].

This is Python based on the C code for strcmp95: http://web.archive.org/web/20110629121242/http://www.census.gov/geo/msb/stand/strcmp.c [WMJL94]. The above file is a US Government publication and, accordingly, in the public domain.

New in version 0.3.6.

Initialize JaroWinkler instance.

Parameters

- **qval** (*int*) -- The length of each q-gram (defaults to 1: character-wise matching)

- **mode** (*str*) -- Indicates which variant of this distance metric to compute:
  - `winkler` -- computes the Jaro-Winkler distance (default) which increases the score for matches near the start of the word
  - `jaro` -- computes the Jaro distance
- **long_strings** (*bool*) -- Set to True to "Increase the probability of a match when the number of matched characters is large. This option allows for a little more tolerance when the strings are large. It is not an appropriate test when comparing fixed length fields such as phone and social security numbers." (Used in 'winkler' mode only.)
- **boost_threshold** (*float*) -- A value between 0 and 1, below which the Winkler boost is not applied (defaults to 0.7). (Used in 'winkler' mode only.)
- **scaling_factor** (*float*) -- A value between 0 and 0.25, indicating by how much to boost scores for matching prefixes (defaults to 0.1). (Used in 'winkler' mode only.)

New in version 0.4.0.

sim(*src*, *tar*)

Return the Jaro or Jaro-Winkler similarity of two strings.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns Jaro or Jaro-Winkler similarity

Return type float

Raises

- **ValueError** -- Unsupported boost_threshold assignment; boost_threshold must be between 0 and 1.
- **ValueError** -- Unsupported scaling_factor assignment; scaling_factor must be between 0 and 0.25.'

Examples

```
>>> round(sim_jaro_winkler('cat', 'hat'), 12)
0.777777777778
>>> round(sim_jaro_winkler('Niall', 'Neil'), 12)
0.805
>>> round(sim_jaro_winkler('aluminum', 'Catalan'), 12)
0.60119047619
>>> round(sim_jaro_winkler('ATCG', 'TAGC'), 12)
0.833333333333
```

```
>>> round(sim_jaro_winkler('cat', 'hat', mode='jaro'), 12)
0.777777777778
>>> round(sim_jaro_winkler('Niall', 'Neil', mode='jaro'), 12)
0.783333333333
>>> round(sim_jaro_winkler('aluminum', 'Catalan', mode='jaro'), 12)
0.60119047619
>>> round(sim_jaro_winkler('ATCG', 'TAGC', mode='jaro'), 12)
0.833333333333
```

New in version 0.1.0.

Changed in version 0.3.6: Encapsulated in class

abydos.distance.**dist_jaro_winkler**(*src*, *tar*, *qval=1*, *mode='winkler'*, *long_strings=False*, *boost_threshold=0.7*, *scaling_factor=0.1*)

Return the Jaro or Jaro-Winkler distance between two strings.

This is a wrapper for `JaroWinkler.dist()`.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison
- **qval** (*int*) -- The length of each q-gram (defaults to 1: character-wise matching)
- **mode** (*str*) -- Indicates which variant of this distance metric to compute:
  - `winkler` -- computes the Jaro-Winkler distance (default) which increases the score for matches near the start of the word
  - `jaro` -- computes the Jaro distance
- **long_strings** (*bool*) -- Set to True to "Increase the probability of a match when the number of matched characters is large. This option allows for a little more tolerance when the strings are large. It is not an appropriate test when comparing fixedlength fields such as phone and social security numbers." (Used in 'winkler' mode only.)
- **boost_threshold** (*float*) -- A value between 0 and 1, below which the Winkler boost is not applied (defaults to 0.7). (Used in 'winkler' mode only.)
- **scaling_factor** (*float*) -- A value between 0 and 0.25, indicating by how much to boost scores for matching prefixes (defaults to 0.1). (Used in 'winkler' mode only.)

Returns Jaro or Jaro-Winkler distance

Return type float

Examples

```
>>> round(dist_jaro_winkler('cat', 'hat'), 12)
0.222222222222
>>> round(dist_jaro_winkler('Niall', 'Neil'), 12)
0.195
>>> round(dist_jaro_winkler('aluminum', 'Catalan'), 12)
0.39880952381
>>> round(dist_jaro_winkler('ATCG', 'TAGC'), 12)
0.166666666667
```

```
>>> round(dist_jaro_winkler('cat', 'hat', mode='jaro'), 12)
0.222222222222
>>> round(dist_jaro_winkler('Niall', 'Neil', mode='jaro'), 12)
0.216666666667
>>> round(dist_jaro_winkler('aluminum', 'Catalan', mode='jaro'), 12)
0.39880952381
>>> round(dist_jaro_winkler('ATCG', 'TAGC', mode='jaro'), 12)
0.166666666667
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the JaroWinkler.dist method instead.

abydos.distance.**sim_jaro_winkler**(*src*, *tar*, *qval=1*, *mode='winkler'*, *long_strings=False*, *boost_threshold=0.7*, *scaling_factor=0.1*)

Return the Jaro or Jaro-Winkler similarity of two strings.

This is a wrapper for *JaroWinkler.sim()*.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison
- **qval** (*int*) -- The length of each q-gram (defaults to 1: character-wise matching)
- **mode** (*str*) -- Indicates which variant of this distance metric to compute:
  - `winkler` -- computes the Jaro-Winkler distance (default) which increases the score for matches near the start of the word
  - `jaro` -- computes the Jaro distance
- **long_strings** (*bool*) -- Set to True to "Increase the probability of a match when the number of matched characters is large. This option allows for a little more tolerance when the strings are large. It is not an appropriate test when comparing fixedlength fields such as phone and social security numbers." (Used in 'winkler' mode only.)
- **boost_threshold** (*float*) -- A value between 0 and 1, below which the Winkler boost is not applied (defaults to 0.7). (Used in 'winkler' mode only.)
- **scaling_factor** (*float*) -- A value between 0 and 0.25, indicating by how much to boost scores for matching prefixes (defaults to 0.1). (Used in 'winkler' mode only.)

Returns Jaro or Jaro-Winkler similarity

Return type float

Examples

```
>>> round(sim_jaro_winkler('cat', 'hat'), 12)
0.777777777778
>>> round(sim_jaro_winkler('Niall', 'Neil'), 12)
0.805
>>> round(sim_jaro_winkler('aluminum', 'Catalan'), 12)
0.60119047619
>>> round(sim_jaro_winkler('ATCG', 'TAGC'), 12)
0.833333333333
```

```
>>> round(sim_jaro_winkler('cat', 'hat', mode='jaro'), 12)
0.777777777778
>>> round(sim_jaro_winkler('Niall', 'Neil', mode='jaro'), 12)
0.783333333333
>>> round(sim_jaro_winkler('aluminum', 'Catalan', mode='jaro'), 12)
0.60119047619
>>> round(sim_jaro_winkler('ATCG', 'TAGC', mode='jaro'), 12)
0.833333333333
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the JaroWinkler.sim method instead.

class abydos.distance.**Strcmp95**(*long_strings=False*, ***kwargs*)
Bases: abydos.distance._distance._Distance

Strcmp95.

This is a Python translation of the C code for strcmp95: http://web.archive.org/web/20110629121242/http://www.census.gov/geo/msb/stand/strcmp.c [WMJL94]. The above file is a US Government publication and, accordingly, in the public domain.

This is based on the Jaro-Winkler distance, but also attempts to correct for some common typos and frequently confused characters. It is also limited to uppercase ASCII characters, so it is appropriate to American names, but not much else.

New in version 0.3.6.

Initialize Strcmp95 instance.

Parameters

- **long_strings** (*bool*) -- Set to True to increase the probability of a match when the number of matched characters is large. This option allows for a little more tolerance when the strings are large. It is not an appropriate test when comparing fixed length fields such as phone and social security numbers.
- ****kwargs** -- Arbitrary keyword arguments

New in version 0.4.0.

sim(*src*, *tar*)
Return the strcmp95 similarity of two strings.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns Strcmp95 similarity

Return type float

Examples

```
>>> cmp = Strcmp95()
>>> cmp.sim('cat', 'hat')
0.7777777777777777
>>> cmp.sim('Niall', 'Neil')
0.8454999999999999
>>> cmp.sim('aluminum', 'Catalan')
0.6547619047619048
>>> cmp.sim('ATCG', 'TAGC')
0.8333333333333334
```

New in version 0.1.0.

Changed in version 0.3.6: Encapsulated in class

abydos.distance.**dist_strcmp95**(*src*, *tar*, *long_strings=False*)
Return the strcmp95 distance between two strings.

This is a wrapper for `Strcmp95.dist()`.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison
- **long_strings** (*bool*) -- Set to True to increase the probability of a match when the number of matched characters is large. This option allows for a little more tolerance when the strings are large. It is not an appropriate test when comparing fixed length fields such as phone and social security numbers.

Returns Strcmp95 distance

Return type float

Examples

```
>>> round(dist_strcmp95('cat', 'hat'), 12)
0.222222222222
>>> round(dist_strcmp95('Niall', 'Neil'), 12)
0.1545
>>> round(dist_strcmp95('aluminum', 'Catalan'), 12)
0.345238095238
>>> round(dist_strcmp95('ATCG', 'TAGC'), 12)
0.166666666667
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the Strcmp95.dist method instead.

abydos.distance.**sim_strcmp95**(*src*, *tar*, *long_strings=False*)
Return the strcmp95 similarity of two strings.

This is a wrapper for *Strcmp95.sim()*.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison
- **long_strings** (*bool*) -- Set to True to increase the probability of a match when the number of matched characters is large. This option allows for a little more tolerance when the strings are large. It is not an appropriate test when comparing fixed length fields such as phone and social security numbers.

Returns Strcmp95 similarity

Return type float

Examples

```
>>> sim_strcmp95('cat', 'hat')
0.7777777777777777
>>> sim_strcmp95('Niall', 'Neil')
0.8454999999999999
>>> sim_strcmp95('aluminum', 'Catalan')
0.6547619047619048
>>> sim_strcmp95('ATCG', 'TAGC')
0.8333333333333334
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the Strcmp95.sim method instead.

class abydos.distance.**IterativeSubString**(*hamacher=0.6*, *normalize_strings=False*, ***kwargs*)

Bases: abydos.distance._distance._Distance

Iterative-SubString correlation.

Iterative-SubString (I-Sub) correlation [SSK05]

This is a straightforward port of the primary author's Java implementation: http://www.image.ece.ntua.gr/~gstoil/software/I_Sub.java

New in version 0.4.0.

Initialize IterativeSubString instance.

Parameters

- **hamacher** (*float*) -- The constant factor for the Hamacher product
- **normalize_strings** (*bool*) -- Normalize the strings by removing the characters in '._ ' and lower casing
- ****kwargs** -- Arbitrary keyword arguments

New in version 0.4.0.

corr(*src*, *tar*)

Return the Iterative-SubString correlation of two strings.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns Iterative-SubString correlation

Return type float

Examples

```
>>> cmp = IterativeSubString()
>>> cmp.corr('cat', 'hat')
-1.0
>>> cmp.corr('Niall', 'Neil')
-0.9
>>> cmp.corr('aluminum', 'Catalan')
-1.0
>>> cmp.corr('ATCG', 'TAGC')
-1.0
```

New in version 0.4.0.

sim(*src*, *tar*)

Return the Iterative-SubString similarity of two strings.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns Iterative-SubString similarity

Return type float

Examples

```
>>> cmp = IterativeSubString()
>>> cmp.sim('cat', 'hat')
0.0
>>> cmp.sim('Niall', 'Neil')
0.04999999999999999
>>> cmp.sim('aluminum', 'Catalan')
0.0
>>> cmp.sim('ATCG', 'TAGC')
0.0
```

New in version 0.4.0.

class abydos.distance.**AMPLE**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

AMPLE similarity.

The AMPLE similarity [DLZ05][AZvanGemund07] is defined in getAverageSequenceWeight() in the AverageSequenceWeightEvaluator.java file of AMPLE's source code. For two sets X and Y and a population N, it is

$$sim_{AMPLE}(X, Y) = \left|\frac{|X \cap Y|}{|X|} - \frac{|Y \setminus X|}{|N \setminus X|}\right|$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$sim_{AMPLE} = \left|\frac{a}{a+b} - \frac{c}{c+d}\right|$$

Notes

This measure is asymmetric. The first ratio considers how similar the two strings are, while the second considers how dissimilar the second string is. As a result, both very similar and very dissimilar strings will score high on this measure, provided the unique aspects are present chiefly in the latter string.

New in version 0.4.0.

Initialize AMPLE instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in `_TokenDistance` for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in `_TokenDistance` for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

sim(*src*, *tar*)

Return the AMPLE similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns AMPLE similarity

Return type float

Examples

```
>>> cmp = AMPLE()
>>> cmp.sim('cat', 'hat')
0.49743589743589745
>>> cmp.sim('Niall', 'Neil')
0.32947729220222793
>>> cmp.sim('aluminum', 'Catalan')
0.10209049255441008
>>> cmp.sim('ATCG', 'TAGC')
0.006418485237483954
```

New in version 0.4.0.

class abydos.distance.**AZZOO**(*sigma=0.5*, *alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

AZZOO similarity.

For two sets X and Y, and alphabet N, and a parameter σ, AZZOO similarity [CTY06] is

$$sim_{AZZOO_\sigma}(X, Y) = \sum s_i$$

where $s_i = 1$ if $X_i = Y_i = 1$, $s_i = \sigma$ if $X_i = Y_i = 0$, and $s_i = 0$ otherwise.

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$sim_{AZZOO} = a + \sigma \cdot d$$

New in version 0.4.0.

Initialize AZZOO instance.

Parameters

- **sigma** (*float*) -- Sigma designates the contribution to similarity given by the 0-0 samples in the set.
- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in _TokenDistance for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the soft and fuzzy variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the fuzzy variant.

New in version 0.4.0.

sim(*src*, *tar*)

Return the AZZOO similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns AZZOO similarity

Return type float

Examples

```
>>> cmp = AZZOO()
>>> cmp.sim('cat', 'hat')
0.9923857868020305
>>> cmp.sim('Niall', 'Neil')
0.9860759493670886
>>> cmp.sim('aluminum', 'Catalan')
0.9710327455919395
>>> cmp.sim('ATCG', 'TAGC')
0.9809885931558935
```

New in version 0.4.0.

sim_score(*src*, *tar*)

Return the AZZOO similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns AZZOO similarity

Return type float

Examples

```
>>> cmp = AZZOO()
>>> cmp.sim_score('cat', 'hat')
391.0
>>> cmp.sim_score('Niall', 'Neil')
389.5
>>> cmp.sim_score('aluminum', 'Catalan')
385.5
>>> cmp.sim_score('ATCG', 'TAGC')
387.0
```

New in version 0.4.0.

class abydos.distance.**Anderberg**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Anderberg's D.

For two sets X and Y and a population N, Anderberg's D [And73] is

$$t_1 = max(|X \cap Y|, |X \setminus Y|) + max(|Y \setminus X|, |(N \setminus X) \setminus Y|) + max(|X \cap Y|, |Y \setminus X|) + max(|X \setminus Y|, |(N \setminus X) \setminus Y|)$$

$$t_2 = max(|Y|, |N \setminus Y|) + max(|X|, |N \setminus X|)$$

$$sim_{Anderberg}(X, Y) = \frac{t_1 - t_2}{2|N|}$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$sim_{Anderberg} = \frac{(max(a,b) + max(c,d) + max(a,c) + max(b,d)) - (max(a+b, b+d) + max(a+b, c+d))}{2n}$$

Notes

There are various references to another "Anderberg similarity", $sim_{Anderberg} = \frac{8a}{8a+b+c}$, but I cannot substantiate the claim that this appears in [And73]. In any case, if you want to use this measure, you may instatiate *WeightedJaccard* with *weight=8*.

Anderberg states that "[t]his quantity is the actual reduction in the error probability (also the actual increase in the correct prediction) as a consequence of using predictor information" [And73]. It ranges [0, 0.5] so a `sim` method ranging [0, 1] is provided in addition to `sim_score`, which gives the value D itself.

It is difficult to term this measure a similarity score. Identical strings often fail to gain high scores. Also, strings that would otherwise be considered quite similar often earn lower scores than those that are less similar.

New in version 0.4.0.

Initialize Anderberg instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in `_TokenDistance` for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in `_TokenDistance` for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

sim(*src*, *tar*)

Return the normalized Anderberg's D similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Normalized Anderberg similarity

Return type float

Examples

```
>>> cmp = Anderberg()
>>> cmp.sim('cat', 'hat')
0.0
>>> cmp.sim('Niall', 'Neil')
0.0
>>> cmp.sim('aluminum', 'Catalan')
0.0
>>> cmp.sim('ATCG', 'TAGC')
0.0
```

New in version 0.4.0.

sim_score(*src*, *tar*)

Return the Anderberg's D similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Anderberg similarity

Return type float

Examples

```
>>> cmp = Anderberg()
>>> cmp.sim_score('cat', 'hat')
0.0
>>> cmp.sim_score('Niall', 'Neil')
0.0
>>> cmp.sim_score('aluminum', 'Catalan')
0.0
>>> cmp.sim_score('ATCG', 'TAGC')
0.0
```

New in version 0.4.0.

class abydos.distance.**AndresMarzoDelta**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Andres & Marzo's Delta correlation.

For two sets X and Y and a population N, Andres & Marzo's Δ correlation [AndresM04] is

$$corr_{AndresMarzo_{\Delta}}(X, Y) = \Delta = \frac{|X \cap Y| + |(N \setminus X) \setminus Y| - 2\sqrt{|X \setminus Y| \cdot |Y \setminus X|}}{|N|}$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$corr_{AndresMarzo_{\Delta}} = \Delta = \frac{a + d - 2\sqrt{b \cdot c}}{n}$$

New in version 0.4.0.

Initialize AndresMarzoDelta instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in `_TokenDistance` for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in `_TokenDistance` for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

corr(*src*, *tar*)

Return the Andres & Marzo's Delta correlation of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Andres & Marzo's Delta correlation

Return type float

Examples

```
>>> cmp = AndresMarzoDelta()
>>> cmp.corr('cat', 'hat')
0.9897959183673469
>>> cmp.corr('Niall', 'Neil')
0.9822344346552608
>>> cmp.corr('aluminum', 'Catalan')
0.9618259496215341
>>> cmp.corr('ATCG', 'TAGC')
0.9744897959183674
```

New in version 0.4.0.

sim(*src*, *tar*)

Return the Andres & Marzo's Delta similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison

- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Andres & Marzo's Delta similarity

Return type float

Examples

```
>>> cmp = AndresMarzoDelta()
>>> cmp.sim('cat', 'hat')
0.9948979591836735
>>> cmp.sim('Niall', 'Neil')
0.9911172173276304
>>> cmp.sim('aluminum', 'Catalan')
0.980912974810767
>>> cmp.sim('ATCG', 'TAGC')
0.9872448979591837
```

New in version 0.4.0.

class abydos.distance.**BaroniUrbaniBuserI**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Baroni-Urbani & Buser I similarity.

For two sets X and Y and a population N, the Baroni-Urbani & Buser I similarity [BUB76] is

$$sim_{BaroniUrbaniBuserI}(X, Y) = \frac{\sqrt{|X \cap Y| \cdot |(N \setminus X) \setminus Y|} + |X \cap Y|}{\sqrt{|X \cap Y| \cdot |(N \setminus X) \setminus Y|} + |X \cap Y| + |X \setminus Y| + |Y \setminus X|}$$

This is the second, but more commonly used and referenced of the two similarities proposed by Baroni-Urbani & Buser.

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$sim_{BaroniUrbaniBuserI} = \frac{\sqrt{ad} + a}{\sqrt{ad} + a + b + c}$$

New in version 0.4.0.

Initialize BaroniUrbaniBuserI instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in _TokenDistance for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

sim(*src*, *tar*)

Return the Baroni-Urbani & Buser I similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Baroni-Urbani & Buser I similarity

Return type float

Examples

```
>>> cmp = BaroniUrbaniBuserI()
>>> cmp.sim('cat', 'hat')
0.9119837740878104
>>> cmp.sim('Niall', 'Neil')
0.8552823175014205
>>> cmp.sim('aluminum', 'Catalan')
0.656992712054851
>>> cmp.sim('ATCG', 'TAGC')
0.0
```

New in version 0.4.0.

class abydos.distance.**BaroniUrbaniBuserII**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: `abydos.distance._token_distance._TokenDistance`

Baroni-Urbani & Buser II correlation.

For two sets X and Y and a population N, the Baroni-Urbani & Buser II correlation [BUB76] is

$$corr_{BaroniUrbaniBuserII}(X,Y) = \frac{\sqrt{|X \cap Y| \cdot |(N \setminus X) \setminus Y|} + |X \cap Y| - |X \setminus Y| - |Y \setminus X|}{\sqrt{|X \cap Y| \cdot |(N \setminus X) \setminus Y|} + |X \cap Y| + |X \setminus Y| + |Y \setminus X|}$$

This is the first, but less commonly used and referenced of the two similarities proposed by Baroni-Urbani & Buser.

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$corr_{BaroniUrbaniBuserII} = \frac{\sqrt{ad} + a - b - c}{\sqrt{ad} + a + b + c}$$

New in version 0.4.0.

Initialize BaroniUrbaniBuserII instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in `_TokenDistance` for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in `_TokenDistance` for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

corr(*src*, *tar*)

Return the Baroni-Urbani & Buser II correlation of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Baroni-Urbani & Buser II correlation

Return type float

Examples

```
>>> cmp = BaroniUrbaniBuserII()
>>> cmp.corr('cat', 'hat')
0.8239675481756209
>>> cmp.corr('Niall', 'Neil')
0.7105646350028408
>>> cmp.corr('aluminum', 'Catalan')
0.31398542410970204
>>> cmp.corr('ATCG', 'TAGC')
-1.0
```

New in version 0.4.0.

sim(*src*, *tar*)

Return the Baroni-Urbani & Buser II similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison

- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Baroni-Urbani & Buser II similarity

Return type float

Examples

```
>>> cmp = BaroniUrbaniBuserII()
>>> cmp.sim('cat', 'hat')
0.9119837740878105
>>> cmp.sim('Niall', 'Neil')
0.8552823175014204
>>> cmp.sim('aluminum', 'Catalan')
0.656992712054851
>>> cmp.sim('ATCG', 'TAGC')
0.0
```

New in version 0.4.0.

class abydos.distance.**BatageljBren**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Batagelj & Bren distance.

For two sets X and Y and a population N, the Batagelj & Bren distance [BB95], Batagelj & Bren's Q_0, is

$$dist_{BatageljBren}(X, Y) = \frac{|X \setminus Y| \cdot |Y \setminus X|}{|X \cap Y| \cdot |(N \setminus X) \setminus Y|}$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$dist_{BatageljBren} = \frac{bc}{ad}$$

New in version 0.4.0.

Initialize BatageljBren instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in _TokenDistance for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.

- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

dist(*src*, *tar*)

Return the normalized Batagelj & Bren distance of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Normalized Batagelj & Bren distance

Return type float

Examples

```
>>> cmp = BatageljBren()
>>> cmp.dist('cat', 'hat')
3.2789465400556106e-06
>>> cmp.dist('Niall', 'Neil')
9.874917709019092e-06
>>> cmp.dist('aluminum', 'Catalan')
9.276668350823718e-05
>>> cmp.dist('ATCG', 'TAGC')
1.0
```

New in version 0.4.0.

dist_abs(*src*, *tar*)

Return the Batagelj & Bren distance of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Batagelj & Bren distance

Return type float

Examples

```
>>> cmp = BatageljBren()
>>> cmp.dist_abs('cat', 'hat')
0.002570694087403599
>>> cmp.dist_abs('Niall', 'Neil')
0.007741935483870968
>>> cmp.dist_abs('aluminum', 'Catalan')
0.07282184655396619
>>> cmp.dist_abs('ATCG', 'TAGC')
inf
```

New in version 0.4.0.

class abydos.distance.**BaulieuI**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Baulieu I distance.

For two sets X and Y and a population N, Baulieu I distance [Bau89] is

$$sim_{BaulieuI}(X, Y) = \frac{|X| \cdot |Y| - |X \cap Y|^2}{|X| \cdot |Y|}$$

This is Baulieu's 12th dissimilarity coefficient.

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$sim_{BaulieuI} = \frac{(a+b)(a+c) - a^2}{(a+b)(a+c)}$$

New in version 0.4.0.

Initialize BaulieuI instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in _TokenDistance for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the soft and fuzzy variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the fuzzy variant.

New in version 0.4.0.

dist(*src*, *tar*)

Return the Baulieu I distance of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Baulieu I distance

Return type float

Examples

```
>>> cmp = BaulieuI()
>>> cmp.dist('cat', 'hat')
0.75
>>> cmp.dist('Niall', 'Neil')
0.8666666666666667
>>> cmp.dist('aluminum', 'Catalan')
0.9861111111111112
>>> cmp.dist('ATCG', 'TAGC')
1.0
```

New in version 0.4.0.

class abydos.distance.**BaulieuII**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Baulieu II similarity.

For two sets X and Y and a population N, Baulieu II similarity [Bau89] is

$$sim_{BaulieuII}(X, Y) = \frac{|X \cap Y|^2 \cdot |(N \setminus X) \setminus Y|^2}{|X| \cdot |Y| \cdot |N \setminus X| \cdot |N \setminus Y|}$$

This is based on Baulieu's 13th dissimilarity coefficient.

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$sim_{BaulieuII} = \frac{a^2 d^2}{(a+b)(a+c)(b+d)(c+d)}$$

New in version 0.4.0.

Initialize BaulieuII instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in _TokenDistance for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the soft and fuzzy variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the fuzzy variant.

New in version 0.4.0.

sim(*src*, *tar*)

Return the Baulieu II similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Baulieu II similarity

Return type float

Examples

```
>>> cmp = BaulieuII()
>>> cmp.sim('cat', 'hat')
0.24871959237343852
>>> cmp.sim('Niall', 'Neil')
0.13213719608444902
>>> cmp.sim('aluminum', 'Catalan')
0.013621892326789235
>>> cmp.sim('ATCG', 'TAGC')
0.0
```

New in version 0.4.0.

class abydos.distance.**BaulieuIII**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Baulieu III distance.

For two sets X and Y and a population N, Baulieu III distance [Bau89] is

$$sim_{BaulieuIII}(X, Y) = \frac{|N|^2 - 4(|X \cap Y| \cdot |(N \setminus X) \setminus Y| - |X \setminus Y| \cdot |Y \setminus X|)}{2 \cdot |N|^2}$$

This is based on Baulieu's 20th dissimilarity coefficient.

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$sim_{BaulieuIII} = \frac{n^2 - 4(ad - bc)}{2n^2}$$

Notes

It should be noted that this is *based on* Baulieu's 20th dissimilarity coefficient. This distance is exactly half Baulieu's 20th dissimilarity. According to [Bau89], the 20th dissimilarity should be a value in the range [0.0, 1.0], meeting the article's (P1) property, but the formula given ranges [0.0, 2.0], so dividing by 2 corrects the formula to meet the article's expectations.

New in version 0.4.0.

Initialize BaulieuIII instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in `_TokenDistance` for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in `_TokenDistance` for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

dist(*src*, *tar*)

Return the Baulieu III distance of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Baulieu III distance

Return type float

Examples

```
>>> cmp = BaulieuIII()
>>> cmp.dist('cat', 'hat')
0.4949500208246564
>>> cmp.dist('Niall', 'Neil')
0.4949955747605165
>>> cmp.dist('aluminum', 'Catalan')
0.49768591017891195
>>> cmp.dist('ATCG', 'TAGC')
0.5000813463140358
```

New in version 0.4.0.

class abydos.distance.**BaulieuIV**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, *positive_irrational=2.718281828459045*, ***kwargs*)

Bases: `abydos.distance._token_distance._TokenDistance`

Baulieu IV distance.

For two sets X and Y, a population N, and a positive irractional number k, Baulieu IV distance [Bau97] is

$$dist_{BaulieuIV}(X, Y) = \frac{|X \setminus Y| + |Y \setminus X| - (|X \cap Y| + \frac{1}{2}) \cdot (|(N \setminus X) \setminus Y| + \frac{1}{2}) \cdot |(N \setminus X) \setminus Y| \cdot k}{|N|}$$

This is Baulieu's 22nd dissimilarity coefficient.

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$dist_{BaulieuIV} = \frac{b + c - (a + \frac{1}{2})(d + \frac{1}{2})dk}{n}$$

Notes

The default value of k is Euler's number e, but other irrationals such as π or $\sqrt{2}$ could be substituted at initialization.

New in version 0.4.0.

Initialize BaulieuIV instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in `_TokenDistance` for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in `_TokenDistance` for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

dist(*src*, *tar*)

Return the normalized Baulieu IV distance of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Normalized Baulieu IV distance

Return type float

Examples

```
>>> cmp = BaulieuIV()
>>> cmp.dist('cat', 'hat')
0.49999799606535283
>>> cmp.dist('Niall', 'Neil')
0.49999801148659684
>>> cmp.dist('aluminum', 'Catalan')
0.49999883126809364
>>> cmp.dist('ATCG', 'TAGC')
0.4999996033268451
```

New in version 0.4.0.

dist_abs(*src*, *tar*)

Return the Baulieu IV distance of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Baulieu IV distance

Return type float

Examples

```
>>> cmp = BaulieuIV()
>>> cmp.dist_abs('cat', 'hat')
-5249.96272285802
>>> cmp.dist_abs('Niall', 'Neil')
-5209.561726488335
>>> cmp.dist_abs('aluminum', 'Catalan')
-3073.6070822721244
>>> cmp.dist_abs('ATCG', 'TAGC')
-1039.2151656463932
```

New in version 0.4.0.

class abydos.distance.**BaulieuV**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Baulieu V distance.

For two sets X and Y and a population N, Baulieu V distance [Bau97] is

$$dist_{BaulieuV}(X, Y) = \frac{|X \setminus Y| + |Y \setminus X| + 1}{|X \cap Y| + |X \setminus Y| + |Y \setminus X| + 1}$$

This is Baulieu's 23rd dissimilarity coefficient. This coefficient fails Baulieu's (P2) property, that $D(a, 0, 0, 0) = 0$. Rather, $D(a, 0, 0, 0) > 0$, but $\lim_{a \to \infty} D(a, 0, 0, 0) = 0$.

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$dist_{BaulieuV} = \frac{b + c + 1}{a + b + c + 1}$$

New in version 0.4.0.

Initialize BaulieuV instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in `_TokenDistance` for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in `_TokenDistance` for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

dist(*src*, *tar*)

Return the Baulieu V distance of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Baulieu V distance

Return type float

Examples

```
>>> cmp = BaulieuV()
>>> cmp.dist('cat', 'hat')
0.7142857142857143
>>> cmp.dist('Niall', 'Neil')
0.8
>>> cmp.dist('aluminum', 'Catalan')
0.9411764705882353
>>> cmp.dist('ATCG', 'TAGC')
1.0
```

New in version 0.4.0.

class abydos.distance.**BaulieuVI**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Baulieu VI distance.

For two sets X and Y and a population N, Baulieu VI distance [Bau97] is

$$dist_{BaulieuVI}(X, Y) = \frac{|X \setminus Y| + |Y \setminus X|}{|X \cap Y| + |X \setminus Y| + |Y \setminus X| + 1}$$

This is Baulieu's 24th dissimilarity coefficient. This coefficient fails Baulieu's (P3) property, that $D(a, b, c, d) = 1$ for some (a,b,c,d). Rather, $D(a, b, c, d) < 1$, but $\lim_{b\to\infty,c\to\infty} D(a, b, c, d) = 0$ for $a = 0$.

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$dist_{BaulieuVI} = \frac{b + c}{a + b + c + 1}$$

New in version 0.4.0.

Initialize BaulieuVI instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in _TokenDistance for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the soft and fuzzy variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the fuzzy variant.

New in version 0.4.0.

dist(*src*, *tar*)

Return the Baulieu VI distance of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Baulieu VI distance

Return type float

Examples

```
>>> cmp = BaulieuVI()
>>> cmp.dist('cat', 'hat')
0.5714285714285714
>>> cmp.dist('Niall', 'Neil')
0.7
>>> cmp.dist('aluminum', 'Catalan')
0.8823529411764706
>>> cmp.dist('ATCG', 'TAGC')
0.9090909090909091
```

New in version 0.4.0.

class abydos.distance.**BaulieuVII**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Baulieu VII distance.

For two sets X and Y and a population N, Baulieu VII distance [Bau97] is

$$dist_{BaulieuVII}(X, Y) = \frac{|X \setminus Y| + |Y \setminus X|}{|N| + |X \cap Y| \cdot (|X \cap Y| - 4)^2}$$

This is Baulieu's 25th dissimilarity coefficient. This coefficient fails Baulieu's (P4) property, that $D(a + 1, b, c, d) \leq D(a, b, c, d) = 0$ with equality holding iff $D(a, b, c, d) = 0$.

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$dist_{BaulieuVII} = \frac{b + c}{n + a \cdot (a - 4)^2}$$

New in version 0.4.0.

Initialize BaulieuVII instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in _TokenDistance for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the soft and fuzzy variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the fuzzy variant.

New in version 0.4.0.

dist (*src*, *tar*)

Return the Baulieu VII distance of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Baulieu VII distance

Return type float

Examples

```
>>> cmp = BaulieuVII()
>>> cmp.dist('cat', 'hat')
0.005050505050505051
>>> cmp.dist('Niall', 'Neil')
0.008838383838383838
>>> cmp.dist('aluminum', 'Catalan')
0.018891687657430732
>>> cmp.dist('ATCG', 'TAGC')
0.012755102040816327
```

New in version 0.4.0.

class abydos.distance.**BaulieuVIII** (*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Baulieu VIII distance.

For two sets X and Y and a population N, Baulieu VIII distance [Bau97] is

$$dist_{BaulieuVIII}(X, Y) = \frac{(|X \setminus Y| - |Y \setminus X|)^2}{|N|^2}$$

This is Baulieu's 26th dissimilarity coefficient. This coefficient fails Baulieu's (P5) property, that $D(a, b + 1, c, d) \geq D(a, b, c, d)$, with equality holding if $D(a, b, c, d) = 1$.

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$dist_{BaulieuVIII} = \frac{(b - c)^2}{n^2}$$

New in version 0.4.0.

Initialize BaulieuVIII instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in _TokenDistance for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package

- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in `_TokenDistance` for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

dist(*src*, *tar*)

Return the Baulieu VIII distance of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Baulieu VIII distance

Return type float

Examples

```
>>> cmp = BaulieuVIII()
>>> cmp.dist('cat', 'hat')
0.0
>>> cmp.dist('Niall', 'Neil')
1.6269262807163682e-06
>>> cmp.dist('aluminum', 'Catalan')
1.6227838857560144e-06
>>> cmp.dist('ATCG', 'TAGC')
0.0
```

New in version 0.4.0.

class abydos.distance.**BaulieuIX**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: `abydos.distance._token_distance._TokenDistance`

Baulieu IX distance.

For two sets X and Y and a population N, Baulieu IX distance [Bau97] is

$$dist_{BaulieuIX}(X, Y) = \frac{|X \setminus Y| + 2 \cdot |Y \setminus X|}{|N| + |Y \setminus X|}$$

This is Baulieu's 27th dissimilarity coefficient. This coefficient fails Baulieu's (P7) property, that $D(a, b, c, d) = D(a, c, b, d)$.

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$dist_{BaulieuIX} = \frac{b + 2c}{a + b + 2c + d}$$

New in version 0.4.0.

Initialize BaulieuIX instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in `_TokenDistance` for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in `_TokenDistance` for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

dist(*src*, *tar*)

Return the Baulieu IX distance of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Baulieu IX distance

Return type float

Examples

```
>>> cmp = BaulieuIX()
>>> cmp.dist('cat', 'hat')
0.007633587786259542
>>> cmp.dist('Niall', 'Neil')
0.012706480304955527
>>> cmp.dist('aluminum', 'Catalan')
0.027777777777777776
>>> cmp.dist('ATCG', 'TAGC')
0.019011406844106463
```

New in version 0.4.0.

class abydos.distance.**BaulieuX**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Baulieu X distance.

For two sets X and Y and a population N, Baulieu X distance [Bau97] is

$$dist_{BaulieuX}(X, Y) = \frac{|X \setminus Y| + |Y \setminus X| + max(|X \setminus Y|, |Y \setminus X|)}{|N| + max(|X \setminus Y|, |Y \setminus X|)}$$

This is Baulieu's 28th dissimilarity coefficient. This coefficient fails Baulieu's (P8) property, that D is a rational function whose numerator and denominator are both (total) linear.

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$dist_{BaulieuX} = \frac{b + c + max(b, c)}{n + max(b, c)}$$

New in version 0.4.0.

Initialize BaulieuX instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in _TokenDistance for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the soft and fuzzy variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the fuzzy variant.

New in version 0.4.0.

dist(*src*, *tar*)

Return the Baulieu X distance of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Baulieu X distance

Return type float

Examples

```
>>> cmp = BaulieuX()
>>> cmp.dist('cat', 'hat')
0.007633587786259542
>>> cmp.dist('Niall', 'Neil')
0.013959390862944163
>>> cmp.dist('aluminum', 'Catalan')
0.029003783102143757
>>> cmp.dist('ATCG', 'TAGC')
0.019011406844106463
```

New in version 0.4.0.

class abydos.distance.**BaulieuXI**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Baulieu XI distance.

For two sets X and Y and a population N, Baulieu XI distance [Bau97] is

$$dist_{BaulieuXI}(X, Y) = \frac{|X \setminus Y| + |Y \setminus X|}{|X \setminus Y| + |Y \setminus X| + |(N \setminus X) \setminus Y|}$$

This is Baulieu's 29th dissimilarity coefficient. This coefficient fails Baulieu's (P4) property, that $D(a + 1, b, c, d) \leq D(a, b, c, d) = 0$ with equality holding iff $D(a, b, c, d) = 0$.

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$dist_{BaulieuXI} = \frac{b + c}{b + c + d}$$

New in version 0.4.0.

Initialize BaulieuXI instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in _TokenDistance for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the soft and fuzzy variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the fuzzy variant.

New in version 0.4.0.

dist(*src*, *tar*)

Return the Baulieu XI distance of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Baulieu XI distance

Return type float

Examples

```
>>> cmp = BaulieuXI()
>>> cmp.dist('cat', 'hat')
0.005115089514066497
>>> cmp.dist('Niall', 'Neil')
0.008951406649616368
>>> cmp.dist('aluminum', 'Catalan')
0.01913265306122449
>>> cmp.dist('ATCG', 'TAGC')
0.012755102040816327
```

New in version 0.4.0.

class abydos.distance.**BaulieuXII**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Baulieu XII distance.

For two sets X and Y and a population N, Baulieu XII distance [Bau97] is

$$dist_{BaulieuXII}(X, Y) = \frac{|X \setminus Y| + |Y \setminus X|}{|X \cap Y| + |X \setminus Y| + |Y \setminus X| - 1}$$

This is Baulieu's 30th dissimilarity coefficient. This coefficient fails Baulieu's (P5) property, that $D(a, b + 1, c, d) \geq D(a, b, c, d)$, with equality holding if $D(a, b, c, d) = 1$.

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$dist_{BaulieuXII} = \frac{b + c}{a + b + c - 1}$$

Notes

In the special case of comparisons where the intersection (a) contains 0 members, the size of the intersection is set to 1, resulting in a distance of 1.0. This prevents the distance from exceeding 1.0 and similarity from becoming negative.

New in version 0.4.0.

Initialize BaulieuXII instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in `_TokenDistance` for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in `_TokenDistance` for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

dist (*src*, *tar*)

Return the Baulieu XII distance of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Baulieu XII distance

Return type float

Examples

```
>>> cmp = BaulieuXII()
>>> cmp.dist('cat', 'hat')
0.8
>>> cmp.dist('Niall', 'Neil')
0.875
>>> cmp.dist('aluminum', 'Catalan')
1.0
>>> cmp.dist('ATCG', 'TAGC')
1.0
```

New in version 0.4.0.

class abydos.distance.**BaulieuXIII**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Baulieu XIII distance.

For two sets X and Y and a population N, Baulieu XIII distance [Bau97] is

$$dist_{BaulieuXIII}(X, Y) = \frac{|X \setminus Y| + |Y \setminus X|}{|X \cap Y| + |X \setminus Y| + |Y \setminus X| + |X \cap Y| \cdot (|X \cap Y| - 4)^2}$$

This is Baulieu's 31st dissimilarity coefficient. This coefficient fails Baulieu's (P4) property, that $D(a + 1, b, c, d) \leq D(a, b, c, d) = 0$ with equality holding iff $D(a, b, c, d) = 0$.

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$dist_{BaulieuXIII} = \frac{b + c}{a + b + c + a \cdot (a - 4)^2}$$

New in version 0.4.0.

Initialize BaulieuXIII instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in _TokenDistance for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the soft and fuzzy variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the fuzzy variant.

New in version 0.4.0.

dist(*src*, *tar*)

Return the Baulieu XIII distance of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Baulieu XIII distance

Return type float

Examples

```
>>> cmp = BaulieuXIII()
>>> cmp.dist('cat', 'hat')
0.2857142857142857
>>> cmp.dist('Niall', 'Neil')
0.4117647058823529
>>> cmp.dist('aluminum', 'Catalan')
0.6
>>> cmp.dist('ATCG', 'TAGC')
1.0
```

New in version 0.4.0.

class abydos.distance.**BaulieuXIV**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Baulieu XIV distance.

For two sets X and Y and a population N, Baulieu XIV distance [Bau97] is

$$dist_{BaulieuXIV}(X, Y) = \frac{|X \setminus Y| + 2 \cdot |Y \setminus X|}{|X \cap Y| + |X \setminus Y| + 2 \cdot |Y \setminus X|}$$

This is Baulieu's 32nd dissimilarity coefficient. This coefficient fails Baulieu's (P7) property, that $D(a, b, c, d) = D(a, c, b, d)$.

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$dist_{BaulieuXIV} = \frac{b + 2c}{a + b + 2c}$$

New in version 0.4.0.

Initialize BaulieuXIV instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in _TokenDistance for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the soft and fuzzy variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the fuzzy variant.

New in version 0.4.0.

dist(*src*, *tar*)

Return the Baulieu XIV distance of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Baulieu XIV distance

Return type float

Examples

```
>>> cmp = BaulieuXIV()
>>> cmp.dist('cat', 'hat')
0.75
>>> cmp.dist('Niall', 'Neil')
0.8333333333333334
>>> cmp.dist('aluminum', 'Catalan')
0.9565217391304348
>>> cmp.dist('ATCG', 'TAGC')
1.0
```

New in version 0.4.0.

class abydos.distance.**BaulieuXV**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Baulieu XV distance.

For two sets X and Y and a population N, Baulieu XV distance [Bau97] is

$$dist_{BaulieuXV}(X, Y) = \frac{|X \setminus Y| + |Y \setminus X| + max(|X \setminus Y|, |Y \setminus X|)}{|X \cap Y| + |X \setminus Y| + |Y \setminus X| + max(|X \setminus Y|, |Y \setminus X|)}$$

This is Baulieu's 33rd dissimilarity coefficient. This coefficient fails Baulieu's (P8) property, that D is a rational function whose numerator and denominator are both (total) linear.

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$dist_{BaulieuXV} = \frac{b + c + max(b, c)}{a + b + c + max(b, c)}$$

New in version 0.4.0.

Initialize BaulieuXV instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in _TokenDistance for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package

- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the soft and fuzzy variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the fuzzy variant.

New in version 0.4.0.

dist(*src*, *tar*)

Return the Baulieu XV distance of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Baulieu XV distance

Return type float

Examples

```
>>> cmp = BaulieuXV()
>>> cmp.dist('cat', 'hat')
0.75
>>> cmp.dist('Niall', 'Neil')
0.8461538461538461
>>> cmp.dist('aluminum', 'Catalan')
0.9583333333333334
>>> cmp.dist('ATCG', 'TAGC')
1.0
```

New in version 0.4.0.

class abydos.distance.**BeniniI**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

BeniniI correlation.

For two sets X and Y and a population N, Benini I correlation, Benini's Index of Attraction, [Ben01] is

$$corr_{BeniniI}(X, Y) = \frac{|X \cap Y| \cdot |(N \setminus X) \setminus Y| - |X \setminus Y| \cdot |Y \setminus X|}{|Y| \cdot |N \setminus X|}$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$corr_{BeniniI} = \frac{ad - bc}{(a + c)(c + d)}$$

New in version 0.4.0.

Initialize BeniniI instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in `_TokenDistance` for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in `_TokenDistance` for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

corr(*src*, *tar*)

Return the Benini I correlation of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Benini I correlation

Return type float

Examples

```
>>> cmp = BeniniI()
>>> cmp.corr('cat', 'hat')
0.49743589743589745
>>> cmp.corr('Niall', 'Neil')
0.3953727506426735
>>> cmp.corr('aluminum', 'Catalan')
0.11485180412371133
>>> cmp.corr('ATCG', 'TAGC')
-0.006418485237483954
```

New in version 0.4.0.

sim(*src*, *tar*)

Return the Benini I similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison

- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Benini I similarity

Return type float

Examples

```
>>> cmp = BeniniI()
>>> cmp.sim('cat', 'hat')
0.7487179487179487
>>> cmp.sim('Niall', 'Neil')
0.6976863753213367
>>> cmp.sim('aluminum', 'Catalan')
0.5574259020618557
>>> cmp.sim('ATCG', 'TAGC')
0.496790757381258
```

New in version 0.4.0.

class abydos.distance.**BeniniII**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

BeniniII correlation.

For two sets X and Y and a population N, Benini II correlation, Benini's Index of Repulsion, [Ben01] is

$$corr_{BeniniII}(X, Y) = \frac{|X \cap Y| \cdot |(N \setminus X) \setminus Y| - |X \setminus Y| \cdot |Y \setminus X|}{min(|Y| \cdot |N \setminus X|, |X| \cdot |N \setminus Y|}$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$corr_{BeniniII} = \frac{ad - bc}{min((a + c)(c + d), (a + b)(b + d))}$$

New in version 0.4.0.

Initialize BeniniII instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in _TokenDistance for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.

- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

corr(*src*, *tar*)

Return the Benini II correlation of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Benini II correlation

Return type float

Examples

```
>>> cmp = BeniniII()
>>> cmp.corr('cat', 'hat')
0.49743589743589745
>>> cmp.corr('Niall', 'Neil')
0.3953727506426735
>>> cmp.corr('aluminum', 'Catalan')
0.11485180412371133
>>> cmp.corr('ATCG', 'TAGC')
-0.006418485237483954
```

New in version 0.4.0.

sim(*src*, *tar*)

Return the Benini II similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Benini II similarity

Return type float

Examples

```
>>> cmp = BeniniII()
>>> cmp.sim('cat', 'hat')
0.7487179487179487
>>> cmp.sim('Niall', 'Neil')
0.6976863753213367
>>> cmp.sim('aluminum', 'Catalan')
0.5574259020618557
>>> cmp.sim('ATCG', 'TAGC')
0.496790757381258
```

New in version 0.4.0.

class abydos.distance.**Bennet**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Bennet's S correlation.

For two sets X and Y and a population N, Bennet's S correlation [BAG54] is

$$corr_{Bennet}(X, Y) = S = \frac{p_o - p_e^S}{1 - p_e^S}$$

where

$$p_o = \frac{|X \cap Y| + |(N \setminus X) \setminus Y|}{|N|}$$

$$p_e^S = \frac{1}{2}$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$p_o = \frac{a + d}{n}$$

$$p_e^S = \frac{1}{2}$$

New in version 0.4.0.

Initialize Bennet instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in _TokenDistance for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the soft and fuzzy variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the fuzzy variant.

New in version 0.4.0.

corr(*src*, *tar*)

Return the Bennet's S correlation of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Bennet's S correlation

Return type float

Examples

```
>>> cmp = Bennet()
>>> cmp.corr('cat', 'hat')
0.989795918367347
>>> cmp.corr('Niall', 'Neil')
0.9821428571428572
>>> cmp.corr('aluminum', 'Catalan')
0.9617834394904459
>>> cmp.corr('ATCG', 'TAGC')
0.9744897959183674
```

New in version 0.4.0.

sim(*src*, *tar*)

Return the Bennet's S similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Bennet's S similarity

Return type float

Examples

```
>>> cmp = Bennet()
>>> cmp.sim('cat', 'hat')
0.9948979591836735
>>> cmp.sim('Niall', 'Neil')
0.9910714285714286
>>> cmp.sim('aluminum', 'Catalan')
0.9808917197452229
>>> cmp.sim('ATCG', 'TAGC')
0.9872448979591837
```

New in version 0.4.0.

class abydos.distance.**BraunBlanquet**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Braun-Blanquet similarity.

For two sets X and Y and a population N, the Braun-Blanquet similarity [BB32] is

$$sim_{BraunBlanquet}(X, Y) = \frac{|X \cap Y|}{max(|X|, |Y|)}$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$sim_{BraunBlanquet} = \frac{a}{max(a + b, a + c)}$$

New in version 0.4.0.

Initialize BraunBlanquet instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in `_TokenDistance` for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in `_TokenDistance` for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

sim(*src*, *tar*)

Return the Braun-Blanquet similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Braun-Blanquet similarity

Return type float

Examples

```
>>> cmp = BraunBlanquet()
>>> cmp.sim('cat', 'hat')
0.5
>>> cmp.sim('Niall', 'Neil')
0.3333333333333333
>>> cmp.sim('aluminum', 'Catalan')
0.1111111111111111
>>> cmp.sim('ATCG', 'TAGC')
0.0
```

New in version 0.4.0.

class abydos.distance.**Canberra**(*tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Canberra distance.

For two sets X and Y, the Canberra distance [LW66][LW67b] is

$$sim_{Canberra}(X, Y) = \frac{|X \triangle Y|}{|X| + |Y|}$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$sim_{Canberra} = \frac{b + c}{(a + b) + (a + c)}$$

New in version 0.4.0.

Initialize Canberra instance.

Parameters

- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the soft and fuzzy variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the fuzzy variant.

New in version 0.4.0.

dist(*src*, *tar*)

Return the Canberra distance of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Canberra distance

Return type float

Examples

```
>>> cmp = Canberra()
>>> cmp.dist('cat', 'hat')
0.5
>>> cmp.dist('Niall', 'Neil')
0.6363636363636364
>>> cmp.dist('aluminum', 'Catalan')
0.8823529411764706
>>> cmp.dist('ATCG', 'TAGC')
1.0
```

New in version 0.4.0.

class abydos.distance.**Cao**(***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Cao's CY dissimilarity.

Given X_{ij} (the number of individuals of speecies j in sample i), X_{kj} (the number of individuals of speecies j in sample k), and N (the total number of speecies present in both samples), Cao dissimilarity (CYd) [CBW97] is:

$$dist_{Cao}(X,Y) = CYd = \frac{1}{N}\sum\left(\frac{(X_{ij}+X_{kj})log_{10}\left(\frac{X_{ij}+X_{kj}}{2}\right) - X_{ij}log_{10}X_{kj} - X_{kj}log_{10}X_{ij}}{X_{ij}+X_{kj}}\right)$$

In the above formula, whenever $X_{ij} = 0$ or $X_{kj} = 0$, the value 0.1 is substituted.

Since this measure ranges from 0 to ∞, a similarity measure, CYs, ranging from 0 to 1 was also developed.

$$sim_{Cao}(X,Y) = CYs = 1 - \frac{Observed\,CYd}{Maximum\,CYd}$$

where

$$Observed\,CYd = \sum\left(\frac{(X_{ij}+X_{kj})log_{10}\left(\frac{X_{ij}+X_{kj}}{2}\right) - X_{ij}log_{10}X_{kj} - X_{kj}log_{10}X_{ij}}{X_{ij}+X_{kj}}\right)$$

and with a (the number of species present in both samples), b (the number of species present in sample i only), and c (the number of species present in sample j only),

$$Maximum\,CYd = D_1 + D_2 + D_3$$

with

$$D_1 = \sum_{j=1}^{b}\left(\frac{(X_{ij}+0.1)log_{10}\left(\frac{X_{ij}+0.1}{2}\right) - X_{ij}log_{10}0.1 - 0.1log_{10}X_{ij}}{X_{ij}+0.1}\right)$$

$$D_2 = \sum_{j=1}^{c}\left(\frac{(X_{kj}+0.1)log_{10}\left(\frac{X_{kj}+0.1}{2}\right) - X_{kj}log_{10}0.1 - 0.1log_{10}X_{kj}}{X_{kj}+0.1}\right)$$

$$D_1 = \sum_{j=1}^{a}\frac{a}{2}\left(\frac{(D_i+1)log_{10}\left(\frac{D_i+1}{2}\right) - log_{10}D_i}{D_i+1} + \frac{(D_k+1)log_{10}\left(\frac{D_k+1}{2}\right) - log_{10}D_k}{D_k+1}\right)$$

with

$$D_i = \frac{\sum X_{ij} - \frac{a}{2}}{\frac{a}{2}}$$
$$D_k = \frac{\sum X_{kj} - \frac{a}{2}}{\frac{a}{2}}$$

for

$$X_{ij} \geq 1$$
$$X_{kj} \geq 1$$

New in version 0.4.1.

Initialize Cao instance.

Parameters ****kwargs** -- Arbitrary keyword arguments

New in version 0.4.1.

dist_abs(*src*, *tar*)

Return Cao's CY dissimilarity (CYd) of two strings.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns Cao's CY dissimilarity

Return type float

Examples

```
>>> cmp = Cao()
>>> cmp.dist_abs('cat', 'hat')
0.3247267992925765
>>> cmp.dist_abs('Niall', 'Neil')
0.4132886536450973
>>> cmp.dist_abs('aluminum', 'Catalan')
0.5530666041976232
>>> cmp.dist_abs('ATCG', 'TAGC')
0.6494535985851531
```

New in version 0.4.1.

sim(*src*, *tar*)

Return Cao's CY similarity (CYs) of two strings.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns Cao's CY similarity

Return type float

Examples

```
>>> cmp = Cao()
>>> cmp.sim('cat', 'hat')
0.0
>>> cmp.sim('Niall', 'Neil')
0.0
>>> cmp.sim('aluminum', 'Catalan')
0.0
>>> cmp.sim('ATCG', 'TAGC')
0.0
```

New in version 0.4.1.

class abydos.distance.**ChaoDice**(***kwargs*)

Bases: abydos.distance._chao_jaccard.ChaoJaccard

Chao's Dice similarity.

Chao's Dice similarity [CCCS04]

New in version 0.4.1.

Initialize ChaoDice instance.

Parameters ****kwargs** -- Arbitrary keyword arguments

New in version 0.4.1.

sim(*src*, *tar*)

Return the normalized Chao's Dice similarity of two strings.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns Normalized Chao's Dice similarity

Return type float

Examples

```
>>> import random
>>> random.seed(0)
>>> cmp = ChaoDice()
>>> cmp.sim('cat', 'hat')
0.3666666666666664
>>> cmp.sim('Niall', 'Neil')
0.27868852459016397
>>> cmp.sim('aluminum', 'Catalan')
0.0
>>> cmp.sim('ATCG', 'TAGC')
0.0
```

New in version 0.4.1.

sim_score(*src*, *tar*)

Return the Chao's Dice similarity of two strings.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns Chao's Dice similarity

Return type float

Examples

```
>>> import random
>>> random.seed(0)
>>> cmp = ChaoDice()
>>> cmp.sim_score('cat', 'hat')
0.3666666666666664
>>> cmp.sim_score('Niall', 'Neil')
0.27868852459016397
>>> cmp.sim_score('aluminum', 'Catalan')
0.0
>>> cmp.sim_score('ATCG', 'TAGC')
0.0
```

New in version 0.4.1.

class abydos.distance.**ChaoJaccard**(***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Chao's Jaccard similarity.

Chao's Jaccard similarity [CCCS04]

New in version 0.4.1.

Initialize ChaoJaccard instance.

Parameters ****kwargs** -- Arbitrary keyword arguments

New in version 0.4.1.

sim(*src*, *tar*)

Return normalized Chao's Jaccard similarity of two strings.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns Normalized Chao's Jaccard similarity

Return type float

Examples

```
>>> import random
>>> random.seed(0)
>>> cmp = ChaoJaccard()
>>> cmp.sim('cat', 'hat')
0.22448979591836735
>>> cmp.sim('Niall', 'Neil')
0.1619047619047619
>>> cmp.sim('aluminum', 'Catalan')
0.0
>>> cmp.sim('ATCG', 'TAGC')
0.0
```

New in version 0.4.1.

sim_score(*src*, *tar*)

Return Chao's Jaccard similarity of two strings.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns Chao's Jaccard similarity

Return type float

Examples

```
>>> import random
>>> random.seed(0)
>>> cmp = ChaoJaccard()
>>> cmp.sim_score('cat', 'hat')
0.22448979591836735
>>> cmp.sim_score('Niall', 'Neil')
0.1619047619047619
>>> cmp.sim_score('aluminum', 'Catalan')
0.0
>>> cmp.sim_score('ATCG', 'TAGC')
0.0
```

New in version 0.4.1.

class abydos.distance.**Chebyshev**(*alphabet=0*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._minkowski.Minkowski

Chebyshev distance.

Euclidean distance is the chessboard distance, equivalent to Minkowski distance in L^{∞}-space.

New in version 0.3.6.

Initialize Euclidean instance.

Parameters

- **alphabet** (*collection or int*) -- The values or size of the alphabet

- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in `_TokenDistance` for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

dist (**args*, ***kwargs*)
Raise exception when called.

Parameters

- ***args** -- Variable length argument list
- ****kwargs** -- Arbitrary keyword arguments

Raises **NotImplementedError** -- Method disabled for Chebyshev distance

New in version 0.3.6.

dist_abs (*src*, *tar*)
Return the Chebyshev distance between two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns The Chebyshev distance

Return type float

Examples

```
>>> cmp = Chebyshev()
>>> cmp.dist_abs('cat', 'hat')
1.0
>>> cmp.dist_abs('Niall', 'Neil')
1.0
>>> cmp.dist_abs('Colin', 'Cuilen')
1.0
>>> cmp.dist_abs('ATCG', 'TAGC')
1.0
```

```
>>> cmp = Chebyshev(qval=1)
>>> cmp.dist_abs('ATCG', 'TAGC')
0.0
```

(continues on next page)

(continued from previous page)

```
>>> cmp.dist_abs('ATCGATTCGGAATTTC', 'TAGCATAATCGCCG')
3.0
```

New in version 0.3.0.

Changed in version 0.3.6: Encapsulated in class

sim(**args*, ***kwargs*)
Raise exception when called.

Parameters

- ***args** -- Variable length argument list
- ****kwargs** -- Arbitrary keyword arguments

Raises **NotImplementedError** -- Method disabled for Chebyshev distance

New in version 0.3.6.

abydos.distance.**chebyshev**(*src*, *tar*, *qval=2*, *alphabet=0*)
Return the Chebyshev distance between two strings.

This is a wrapper for the *Chebyshev.dist_abs()*.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison
- **qval** (*int*) -- The length of each q-gram
- **alphabet** (*collection or int*) -- The values or size of the alphabet

Returns The Chebyshev distance

Return type float

Examples

```
>>> chebyshev('cat', 'hat')
1.0
>>> chebyshev('Niall', 'Neil')
1.0
>>> chebyshev('Colin', 'Cuilen')
1.0
>>> chebyshev('ATCG', 'TAGC')
1.0
>>> chebyshev('ATCG', 'TAGC', qval=1)
0.0
>>> chebyshev('ATCGATTCGGAATTTC', 'TAGCATAATCGCCG', qval=1)
3.0
```

New in version 0.3.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the Chebyshev.dist_abs method instead.

class abydos.distance.**Chord**(*tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)
Bases: abydos.distance._token_distance._TokenDistance

Chord distance.

For two sets X and Y drawn from an alphabet S, the chord distance [Orloci67] is

$$sim_{chord}(X, Y) = \sqrt{\sum_{i \in S} \left(\frac{X_i}{\sqrt{\sum_{j \in X} X_j^2}} - \frac{Y_i}{\sqrt{\sum_{j \in Y} Y_j^2}} \right)^2}$$

New in version 0.4.0.

Initialize Chord instance.

Parameters

- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in `_TokenDistance` for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

dist(*src*, *tar*)

Return the normalized Chord distance of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Normalized chord distance

Return type float

Examples

```
>>> cmp = Chord()
>>> cmp.dist('cat', 'hat')
0.707106781186547
>>> cmp.dist('Niall', 'Neil')
0.796775770420944
>>> cmp.dist('aluminum', 'Catalan')
0.94519820240106
>>> cmp.dist('ATCG', 'TAGC')
1.0
```

New in version 0.4.0.

dist_abs(*src*, *tar*)
Return the Chord distance of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Chord distance

Return type float

Examples

```
>>> cmp = Chord()
>>> cmp.dist_abs('cat', 'hat')
1.0
>>> cmp.dist_abs('Niall', 'Neil')
1.126811100699571
>>> cmp.dist_abs('aluminum', 'Catalan')
1.336712116966249
>>> cmp.dist_abs('ATCG', 'TAGC')
1.414213562373095
```

New in version 0.4.0.

class abydos.distance.**Clark**(***kwargs*)
Bases: abydos.distance._token_distance._TokenDistance

Clark's coefficient of divergence.

For two sets X and Y and a population N, Clark's coefficient of divergence [Cla52] is:

$$dist_{Clark}(X, Y) = \sqrt{\frac{\sum_{i=0}^{|N|} \left(\frac{x_i - y_i}{x_i + y_i}\right)^2}{|N|}}$$

New in version 0.4.1.

Initialize Clark instance.

Parameters ****kwargs** -- Arbitrary keyword arguments

New in version 0.4.1.

dist(*src*, *tar*)
Return Clark's coefficient of divergence of two strings.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns Clark's coefficient of divergence

Return type float

Examples

```
>>> cmp = Clark()
>>> cmp.dist('cat', 'hat')
0.816496580927726
>>> cmp.dist('Niall', 'Neil')
0.8819171036881969
>>> cmp.dist('aluminum', 'Catalan')
0.9660917830792959
>>> cmp.dist('ATCG', 'TAGC')
1.0
```

New in version 0.4.1.

class abydos.distance.**Clement**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Clement similarity.

For two sets X and Y and a population N, Clement similarity [Cle76] is defined as

$$sim_{Clement}(X, Y) = \frac{|X \cap Y|}{|X|}\left(1 - \frac{|X|}{|N|}\right) + \frac{|(N \setminus X) \setminus Y|}{|N \setminus X|}\left(1 - \frac{|N \setminus X|}{|N|}\right)$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$sim_{Clement} = \frac{a}{a+b}\left(1 - \frac{a+b}{n}\right) + \frac{d}{c+d}\left(1 - \frac{c+d}{n}\right)$$

New in version 0.4.0.

Initialize Clement instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in _TokenDistance for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the soft and fuzzy variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the fuzzy variant.

New in version 0.4.0.

sim(*src*, *tar*)
Return the Clement similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Clement similarity

Return type float

Examples

```
>>> cmp = Clement()
>>> cmp.sim('cat', 'hat')
0.5025379382522239
>>> cmp.sim('Niall', 'Neil')
0.33840586363079933
>>> cmp.sim('aluminum', 'Catalan')
0.12119877280918714
>>> cmp.sim('ATCG', 'TAGC')
0.006336616803332366
```

New in version 0.4.0.

class abydos.distance.**CohenKappa**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)
Bases: abydos.distance._token_distance._TokenDistance

Cohen's Kappa similarity.

For two sets X and Y and a population N, Cohen's kappa similarity [Coh60] is

$$sim_{Cohen_\kappa}(X, Y) = \kappa = \frac{p_o - p_e^\kappa}{1 - p_e^\kappa}$$

where

$$p_o = \frac{|X \cap Y| + |(N \setminus X) \setminus Y|}{|N|}$$

$$p_e^\kappa = \frac{|X|}{|N|} \cdot \frac{|Y|}{|N|} + \frac{|N \setminus X|}{|N|} \cdot \frac{|N \setminus Y|}{|N|}$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$p_o = \frac{a+d}{n}$$

$$p_e^\kappa = \frac{a+b}{n} \cdot \frac{a+c}{n} + \frac{c+d}{n} \cdot \frac{b+d}{n}$$

New in version 0.4.0.

Initialize CohenKappa instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in `_TokenDistance` for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in `_TokenDistance` for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

sim(*src*, *tar*)

Return Cohen's Kappa similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Cohen's Kappa similarity

Return type float

Examples

```
>>> cmp = CohenKappa()
>>> cmp.sim('cat', 'hat')
0.9974358974358974
>>> cmp.sim('Niall', 'Neil')
0.9955041746949261
>>> cmp.sim('aluminum', 'Catalan')
0.9903412749517064
>>> cmp.sim('ATCG', 'TAGC')
0.993581514762516
```

New in version 0.4.0.

class abydos.distance.**Cole**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: `abydos.distance._token_distance._TokenDistance`

Cole correlation.

For two sets X and Y and a population N, the Cole correlation [Col49] has three formulae:

- If $|X \cap Y| \cdot |(N \setminus X) \setminus Y| \geq |X \setminus Y| \cdot |Y \setminus Y|$ then

$$corr_{Cole}(X, Y) = \frac{|X \cap Y| \cdot |(N \setminus X) \setminus Y| - |X \setminus Y| \cdot |Y \setminus X|}{(|X \cap Y| + |X \setminus Y|) \cdot (|X \setminus Y| + |(N \setminus X) \setminus Y|)}$$

- If $|(N \setminus X) \setminus Y| \geq |X \cap Y|$ then

$$corr_{Cole}(X, Y) = \frac{|X \cap Y| \cdot |(N \setminus X) \setminus Y| - |X \setminus Y| \cdot |Y \setminus X|}{(|X \cap Y| + |X \setminus Y|) \cdot (|X \cap Y| + |Y \setminus X|)}$$

- Otherwise

$$corr_{Cole}(X, Y) = \frac{|X \cap Y| \cdot |(N \setminus X) \setminus Y| - |X \setminus Y| \cdot |Y \setminus X|}{(|X \setminus Y| + |(N \setminus X) \setminus Y|) \cdot (|Y \setminus X| + |(N \setminus X) \setminus Y|)}$$

Cole terms this measurement the Coefficient of Interspecific Association.

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$corr_{Cole} = \begin{cases} \frac{ad-bc}{(a+b)(b+d)} & \text{if } ad \geq bc \\ \frac{ad-bc}{(a+b)(a+c)} & \text{if } d \geq a \\ \frac{ad-bc}{(b+d)(c+d)} & \text{otherwise} \end{cases}$$

New in version 0.4.0.

Initialize Cole instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in `_TokenDistance` for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in `_TokenDistance` for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

corr(*src*, *tar*)

Return the Cole correlation of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Cole correlation

Return type float

Examples

```
>>> cmp = Cole()
>>> cmp.corr('cat', 'hat')
0.49743589743589745
>>> cmp.corr('Niall', 'Neil')
0.3290543431750107
>>> cmp.corr('aluminum', 'Catalan')
0.10195910195910196
>>> cmp.corr('ATCG', 'TAGC')
-1.0
```

New in version 0.4.0.

sim(*src*, *tar*)

Return the Cole similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for similarity
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for similarity

Returns Cole similarity

Return type float

Examples

```
>>> cmp = Cole()
>>> cmp.sim('cat', 'hat')
0.7487179487179487
>>> cmp.sim('Niall', 'Neil')
0.6645271715875054
>>> cmp.sim('aluminum', 'Catalan')
0.550979550979551
>>> cmp.sim('ATCG', 'TAGC')
0.0
```

New in version 0.4.0.

class abydos.distance.**ConsonniTodeschiniI**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Consonni & Todeschini I similarity.

For two sets X and Y and a population N, Consonni & Todeschini I similarity [CT12] is

$$sim_{ConsonniTodeschiniI}(X, Y) = \frac{log(1 + |X \cap Y| + |(N \setminus X) \setminus Y|)}{log(1 + |N|)}$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$sim_{ConsonniTodeschiniI} = \frac{log(1 + a + d)}{log(1 + n)}$$

New in version 0.4.0.

Initialize ConsonniTodeschiniI instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in `_TokenDistance` for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in `_TokenDistance` for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

sim(*src*, *tar*)

Return the Consonni & Todeschini I similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Consonni & Todeschini I similarity

Return type float

Examples

```
>>> cmp = ConsonniTodeschiniI()
>>> cmp.sim('cat', 'hat')
0.9992336018090547
>>> cmp.sim('Niall', 'Neil')
0.998656222829757
>>> cmp.sim('aluminum', 'Catalan')
0.9971098629456009
>>> cmp.sim('ATCG', 'TAGC')
0.9980766131469967
```

New in version 0.4.0.

class abydos.distance.**ConsonniTodeschiniII**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Consonni & Todeschini II similarity.

For two sets X and Y and a population N, Consonni & Todeschini II similarity [CT12] is

$$sim_{ConsonniTodeschiniII}(X, Y) = \frac{log(1 + |N|) - log(1 + |X \setminus Y| + |Y \setminus X|}{log(1 + |N|)}$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$sim_{ConsonniTodeschiniII} = \frac{log(1 + n) - log(1 + b + c)}{log(1 + n)}$$

New in version 0.4.0.

Initialize ConsonniTodeschiniII instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in _TokenDistance for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the soft and fuzzy variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the fuzzy variant.

New in version 0.4.0.

sim(*src*, *tar*)
Return the Consonni & Todeschini II similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Consonni & Todeschini II similarity

Return type float

Examples

```
>>> cmp = ConsonniTodeschiniII()
>>> cmp.sim('cat', 'hat')
0.7585487129939101
>>> cmp.sim('Niall', 'Neil')
0.6880377723094788
>>> cmp.sim('aluminum', 'Catalan')
0.5841297898633079
>>> cmp.sim('ATCG', 'TAGC')
0.640262668568961
```

New in version 0.4.0.

class abydos.distance.**ConsonniTodeschiniIII**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)
Bases: abydos.distance._token_distance._TokenDistance

Consonni & Todeschini III similarity.

For two sets X and Y and a population N, Consonni & Todeschini III similarity [CT12] is

$$sim_{ConsonniTodeschiniIII}(X, Y) = \frac{log(1 + |X \cap Y|)}{log(1 + |N|)}$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$sim_{ConsonniTodeschiniIII} = \frac{log(1 + a)}{log(1 + n)}$$

New in version 0.4.0.

Initialize ConsonniTodeschiniIII instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in _TokenDistance for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

sim(*src*, *tar*)

Return the Consonni & Todeschini III similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Consonni & Todeschini III similarity

Return type float

Examples

```
>>> cmp = ConsonniTodeschiniIII()
>>> cmp.sim('cat', 'hat')
0.1648161441769704
>>> cmp.sim('Niall', 'Neil')
0.1648161441769704
>>> cmp.sim('aluminum', 'Catalan')
0.10396755253417303
>>> cmp.sim('ATCG', 'TAGC')
0.0
```

New in version 0.4.0.

class abydos.distance.**ConsonniTodeschiniIV**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Consonni & Todeschini IV similarity.

For two sets X and Y and a population N, Consonni & Todeschini IV similarity [CT12] is

$$sim_{ConsonniTodeschiniIV}(X, Y) = \frac{log(1 + |X \cap Y|)}{log(1 + |X \cup Y|)}$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$sim_{ConsonniTodeschiniIV} = \frac{log(1 + a)}{log(1 + a + b + c)}$$

New in version 0.4.0.

Initialize ConsonniTodeschiniIV instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in `_TokenDistance` for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in `_TokenDistance` for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

sim(*src*, *tar*)

Return the Consonni & Todeschini IV similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Consonni & Todeschini IV similarity

Return type float

Examples

```
>>> cmp = ConsonniTodeschiniIV()
>>> cmp.sim('cat', 'hat')
0.5645750340535796
>>> cmp.sim('Niall', 'Neil')
0.4771212547196623
>>> cmp.sim('aluminum', 'Catalan')
0.244650542118226
>>> cmp.sim('ATCG', 'TAGC')
0.0
```

New in version 0.4.0.

class abydos.distance.**ConsonniTodeschiniV**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: `abydos.distance._token_distance._TokenDistance`

Consonni & Todeschini V correlation.

For two sets X and Y and a population N, Consonni & Todeschini V correlation [CT12] is

$$corr_{ConsonniTodeschiniV}(X, Y) = \frac{log(1 + |X \cap Y| \cdot |(N \setminus X) \setminus Y|) - log(1 + |X \setminus Y| \cdot |Y \setminus X|)}{log(1 + \frac{|N|^2}{4})}$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$corr_{ConsonniTodeschiniV} = \frac{log(1 + ad) - log(1 + bc)}{log(1 + \frac{n^2}{4})}$$

New in version 0.4.0.

Initialize ConsonniTodeschiniV instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in _TokenDistance for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the soft and fuzzy variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the fuzzy variant.

New in version 0.4.0.

corr(*src*, *tar*)

Return the Consonni & Todeschini V correlation of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Consonni & Todeschini V correlation

Return type float

Examples

```
>>> cmp = ConsonniTodeschiniV()
>>> cmp.corr('cat', 'hat')
0.48072545510682463
>>> cmp.corr('Niall', 'Neil')
0.4003930264973547
>>> cmp.corr('aluminum', 'Catalan')
0.21794239483504532
>>> cmp.corr('ATCG', 'TAGC')
-0.2728145951429799
```

New in version 0.4.0.

sim(*src*, *tar*)

Return the Consonni & Todeschini V similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Consonni & Todeschini V similarity

Return type float

Examples

```
>>> cmp = ConsonniTodeschiniV()
>>> cmp.sim('cat', 'hat')
0.7403627275534124
>>> cmp.sim('Niall', 'Neil')
0.7001965132486774
>>> cmp.sim('aluminum', 'Catalan')
0.6089711974175227
>>> cmp.sim('ATCG', 'TAGC')
0.36359270242851005
```

New in version 0.4.0.

class abydos.distance.**Cosine**(*tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Cosine similarity.

For two sets X and Y, the cosine similarity, Otsuka-Ochiai coefficient, or Ochiai coefficient [Ots36][Och57] is

$$sim_{cosine}(X, Y) = \frac{|X \cap Y|}{\sqrt{|X| \cdot |Y|}}$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$sim_{cosine} = \frac{a}{\sqrt{(a+b)(a+c)}}$$

Notes

This measure is also known as the Fowlkes-Mallows index [FM83] for two classes and G-measure, the geometric mean of precision & recall.

New in version 0.3.6.

Initialize Cosine instance.

Parameters

- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in `_TokenDistance` for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

sim(*src*, *tar*)

Return the cosine similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Cosine similarity

Return type float

Examples

```
>>> cmp = Cosine()
>>> cmp.sim('cat', 'hat')
0.5
>>> cmp.sim('Niall', 'Neil')
0.3651483716701107
>>> cmp.sim('aluminum', 'Catalan')
0.11785113019775793
>>> cmp.sim('ATCG', 'TAGC')
0.0
```

New in version 0.1.0.

Changed in version 0.3.6: Encapsulated in class

abydos.distance.**dist_cosine**(*src*, *tar*, *qval=2*)
Return the cosine distance between two strings.

This is a wrapper for `Cosine.dist()`.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison
- **qval** (*int*) -- The length of each q-gram

Returns Cosine distance

Return type float

Examples

```
>>> dist_cosine('cat', 'hat')
0.5
>>> dist_cosine('Niall', 'Neil')
0.6348516283298893
>>> dist_cosine('aluminum', 'Catalan')
0.882148869802242
>>> dist_cosine('ATCG', 'TAGC')
1.0
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the Cosine.dist method instead.

abydos.distance.**sim_cosine**(*src*, *tar*, *qval=2*)
Return the cosine similarity of two strings.

This is a wrapper for *Cosine.sim()*.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison
- **qval** (*int*) -- The length of each q-gram

Returns Cosine similarity

Return type float

Examples

```
>>> sim_cosine('cat', 'hat')
0.5
>>> sim_cosine('Niall', 'Neil')
0.3651483716701107
>>> sim_cosine('aluminum', 'Catalan')
0.11785113019775793
>>> sim_cosine('ATCG', 'TAGC')
0.0
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the Cosine.sim method instead.

class abydos.distance.**Dennis**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Dennis similarity.

For two sets X and Y and a population N, Dennis similarity [Den65] is

$$sim_{Dennis}(X, Y) = \frac{|X \cap Y| - \frac{|X| \cdot |Y|}{|N|}}{\sqrt{\frac{|X| \cdot |Y|}{|N|}}}$$

This is the fourth of Dennis' association measures, and that which she claims is the best of the four.

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$sim_{Dennis} = \frac{a - \frac{(a+b)(a+c)}{n}}{\sqrt{\frac{(a+b)(a+c)}{n}}}$$

New in version 0.4.0.

Initialize Dennis instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in _TokenDistance for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the soft and fuzzy variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the fuzzy variant.

New in version 0.4.0.

corr(*src*, *tar*)

Return the Dennis correlation of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Dennis correlation

Return type float

Examples

```
>>> cmp = Dennis()
>>> cmp.corr('cat', 'hat')
0.494897959183673
>>> cmp.corr('Niall', 'Neil')
0.358162114559075
>>> cmp.corr('aluminum', 'Catalan')
0.107041854561785
>>> cmp.corr('ATCG', 'TAGC')
-0.006377551020408
```

New in version 0.4.0.

sim(*src*, *tar*)

Return the normalized Dennis similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Normalized Dennis similarity

Return type float

Examples

```
>>> cmp = Dennis()
>>> cmp.sim('cat', 'hat')
0.6632653061224487
>>> cmp.sim('Niall', 'Neil')
0.5721080763727167
>>> cmp.sim('aluminum', 'Catalan')
0.4046945697078567
>>> cmp.sim('ATCG', 'TAGC')
0.32908163265306134
```

New in version 0.4.0.

sim_score(*src*, *tar*)

Return the Dennis similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Dennis similarity

Return type float

Examples

```
>>> cmp = Dennis()
>>> cmp.sim_score('cat', 'hat')
13.857142857142858
>>> cmp.sim_score('Niall', 'Neil')
10.028539207654113
>>> cmp.sim_score('aluminum', 'Catalan')
2.9990827802847835
>>> cmp.sim_score('ATCG', 'TAGC')
-0.17857142857142858
```

New in version 0.4.0.

class abydos.distance.**Dice**(*tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._tversky.Tversky

Sørensen–Dice coefficient.

For two sets X and Y, the Sørensen–Dice coefficient [Dic45][Sorensen48][Cze09][MDobrzanskiZ50] is

$$sim_{Dice}(X, Y) = \frac{2 \cdot |X \cap Y|}{|X| + |Y|}$$

This is the complement of Bray & Curtis dissimilarity [BC57], also known as the Lance & Williams dissimilarity [LW67a].

This is identical to the Tanimoto similarity coefficient [Tan58] and the Tversky index [Tve77] for $\alpha = \beta = 0.5$.

In the Ruby text library this is identified as White similarity, after [Whid.].

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$sim_{Dice} = \frac{2a}{2a + b + c}$$

Notes

In terms of a confusion matrix, this is equivalent to F_1 score `ConfusionTable.f1_score()`.

The multiset variant is termed Gleason similarity [Gle20].

New in version 0.3.6.

Initialize Dice instance.

Parameters

- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in `_TokenDistance` for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

sim(*src*, *tar*)

Return the Sørensen–Dice coefficient of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Sørensen–Dice similarity

Return type float

Examples

```
>>> cmp = Dice()
>>> cmp.sim('cat', 'hat')
0.5
>>> cmp.sim('Niall', 'Neil')
0.36363636363636365
>>> cmp.sim('aluminum', 'Catalan')
0.11764705882352941
>>> cmp.sim('ATCG', 'TAGC')
0.0
```

New in version 0.1.0.

Changed in version 0.3.6: Encapsulated in class

abydos.distance.**dist_dice**(*src*, *tar*, *qval=2*)

Return the Sørensen–Dice distance between two strings.

This is a wrapper for `Dice.dist()`.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison
- **qval** (*int*) -- The length of each q-gram

Returns Sørensen–Dice distance

Return type float

Examples

```
>>> dist_dice('cat', 'hat')
0.5
>>> dist_dice('Niall', 'Neil')
0.6363636363636364
>>> dist_dice('aluminum', 'Catalan')
0.8823529411764706
>>> dist_dice('ATCG', 'TAGC')
1.0
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the Dice.dist method instead.

abydos.distance.**sim_dice**(*src*, *tar*, *qval=2*)

Return the Sørensen–Dice coefficient of two strings.

This is a wrapper for *Dice.sim()*.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison
- **qval** (*int*) -- The length of each q-gram

Returns Sørensen–Dice similarity

Return type float

Examples

```
>>> sim_dice('cat', 'hat')
0.5
>>> sim_dice('Niall', 'Neil')
0.36363636363636365
>>> sim_dice('aluminum', 'Catalan')
0.11764705882352941
>>> sim_dice('ATCG', 'TAGC')
0.0
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the Dice.sim method instead.

class abydos.distance.**DiceAsymmetricI**(*tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Dice's Asymmetric I similarity.

For two sets X and Y and a population N, Dice's Asymmetric I similarity [Dic45] is

$$sim_{DiceAsymmetricI}(X, Y) = \frac{|X \cap Y|}{|X|}$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$sim_{DiceAsymmetricI} = \frac{a}{a+b}$$

Notes

In terms of a confusion matrix, this is equivalent to precision or positive predictive value `ConfusionTable.precision()`.

New in version 0.4.0.

Initialize DiceAsymmetricI instance.

Parameters

- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in `_TokenDistance` for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

sim(*src*, *tar*)

Return the Dice's Asymmetric I similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Dice's Asymmetric I similarity

Return type float

Examples

```
>>> cmp = DiceAsymmetricI()
>>> cmp.sim('cat', 'hat')
0.5
>>> cmp.sim('Niall', 'Neil')
0.3333333333333333
>>> cmp.sim('aluminum', 'Catalan')
0.1111111111111111
>>> cmp.sim('ATCG', 'TAGC')
0.0
```

New in version 0.4.0.

class abydos.distance.**DiceAsymmetricII**(*tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Dice's Asymmetric II similarity.

For two sets X and Y, Dice's Asymmetric II similarity [Dic45] is

$$sim_{DiceAsymmetricII}(X, Y) = \frac{|X \cap Y|}{|Y|}$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$sim_{DiceAsymmetricII} = \frac{a}{a+c}$$

Notes

In terms of a confusion matrix, this is equivalent to recall, sensitivity, or true positive rate `ConfusionTable.recall()`.

New in version 0.4.0.

Initialize DiceAsymmetricII instance.

Parameters

- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in `_TokenDistance` for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

sim(*src*, *tar*)

Return the Dice's Asymmetric II similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Dice's Asymmetric II similarity

Return type float

Examples

```
>>> cmp = DiceAsymmetricII()
>>> cmp.sim('cat', 'hat')
0.5
>>> cmp.sim('Niall', 'Neil')
0.4
>>> cmp.sim('aluminum', 'Catalan')
0.125
>>> cmp.sim('ATCG', 'TAGC')
0.0
```

New in version 0.4.0.

class abydos.distance.**Digby**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Digby correlation.

For two sets X and Y and a population N, Digby's approximation of the tetrachoric correlation coefficient [Dig83] is

$$corr_{Digby}(X, Y) = \frac{(|X \cap Y| \cdot |(N \setminus X) \setminus Y|)^{\frac{3}{4}} - (|X \setminus Y| \cdot |Y \setminus X|)^{\frac{3}{4}}}{(|X \cap Y| \cdot |(N \setminus X) \setminus Y|)^{\frac{3}{4}} + (|X \setminus Y| \cdot |Y \setminus X|)^{\frac{3}{4}}}$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$corr_{Digby} = \frac{ad^{\frac{3}{4}} - bc^{\frac{3}{4}}}{ad^{\frac{3}{4}} + bc^{\frac{3}{4}}}$$

New in version 0.4.0.

Initialize Digby instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in _TokenDistance for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the soft and fuzzy variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the fuzzy variant.

New in version 0.4.0.

corr(*src*, *tar*)

Return the Digby correlation of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Digby correlation

Return type float

Examples

```
>>> cmp = Digby()
>>> cmp.corr('cat', 'hat')
0.9774244829419212
>>> cmp.corr('Niall', 'Neil')
0.9491281473458171
>>> cmp.corr('aluminum', 'Catalan')
0.7541039303781305
>>> cmp.corr('ATCG', 'TAGC')
-1.0
```

New in version 0.4.0.

sim(*src*, *tar*)

Return the Digby similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Digby similarity

Return type float

Examples

```
>>> cmp = Digby()
>>> cmp.sim('cat', 'hat')
0.9887122414709606
>>> cmp.sim('Niall', 'Neil')
0.9745640736729085
>>> cmp.sim('aluminum', 'Catalan')
0.8770519651890653
>>> cmp.sim('ATCG', 'TAGC')
0.0
```

New in version 0.4.0.

class abydos.distance.**Dispersion**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Dispersion correlation.

For two sets X and Y and a population N, the dispersion correlation [Cor17] is

$$corr_{dispersion}(X, Y) = \frac{|X \cap Y| \cdot |(N \setminus X) \setminus Y| - |X \setminus Y| \cdot |Y \setminus X|}{|N|^2}$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$corr_{dispersion} = \frac{ad - bc}{n^2}$$

New in version 0.4.0.

Initialize Dispersion instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in `_TokenDistance` for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in `_TokenDistance` for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

corr(*src*, *tar*)

Return the Dispersion correlation of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Dispersion correlation

Return type float

Examples

```
>>> cmp = Dispersion()
>>> cmp.corr('cat', 'hat')
0.002524989587671803
>>> cmp.corr('Niall', 'Neil')
0.002502212619741774
>>> cmp.corr('aluminum', 'Catalan')
0.0011570449105440383
>>> cmp.corr('ATCG', 'TAGC')
-4.06731570179092e-05
```

New in version 0.4.0.

sim(*src*, *tar*)

Return the Dispersion similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Dispersion similarity

Return type float

Examples

```
>>> cmp = Dispersion()
>>> cmp.sim('cat', 'hat')
0.5012624947938359
>>> cmp.sim('Niall', 'Neil')
0.5012511063098709
>>> cmp.sim('aluminum', 'Catalan')
0.500578522455272
>>> cmp.sim('ATCG', 'TAGC')
0.499979663421491
```

New in version 0.4.0.

class abydos.distance.**Doolittle**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Doolittle similarity.

For two sets X and Y and a population N, the Doolittle similarity [Doo84] is

$$sim_{Doolittle}(X, Y) = \frac{(|X \cap Y| \cdot |N| - |X| \cdot |Y|)^2}{|X| \cdot |Y| \cdot |N \setminus Y| \cdot |N \setminus X|}$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$sim_{Doolittle} = \frac{(an - (a+b)(a+c))^2}{(a+b)(a+c)(b+d)(c+d)}$$

New in version 0.4.0.

Initialize Doolittle instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in `_TokenDistance` for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in `_TokenDistance` for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

sim(*src*, *tar*)

Return the Doolittle similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Doolittle similarity

Return type float

Examples

```
>>> cmp = Doolittle()
>>> cmp.sim('cat', 'hat')
0.24744247205785666
>>> cmp.sim('Niall', 'Neil')
0.13009912077202224
>>> cmp.sim('aluminum', 'Catalan')
0.011710186806836291
>>> cmp.sim('ATCG', 'TAGC')
4.1196952743799446e-05
```

New in version 0.4.0.

class abydos.distance.**Dunning**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: `abydos.distance._token_distance._TokenDistance`

Dunning similarity.

For two sets X and Y and a population N, Dunning log-likelihood [Dun93], following [CGHH91], is

$$sim_{Dunning}(X, Y) = \lambda = |X \cap Y| \cdot log_2(|X \cap Y|) +$$
$$|X \setminus Y| \cdot log_2(|X \setminus Y|) + |Y \setminus X| \cdot log_2(|Y \setminus X|) +$$
$$|(N \setminus X) \setminus Y| \cdot log_2(|(N \setminus X) \setminus Y|) -$$
$$(|X| \cdot log_2(|X|) + |Y| \cdot log_2(|Y|) +$$
$$|N \setminus Y| \cdot log_2(|N \setminus Y|) + |N \setminus X| \cdot log_2(|N \setminus X|))$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$sim_{Dunning} = \lambda = a \cdot log_2(a) +$$
$$b \cdot log_2(b) + c \cdot log_2(c) + d \cdot log_2(d) -$$
$$((a+b) \cdot log_2(a+b) + (a+c) \cdot log_2(a+c) +$$
$$(b+d) \cdot log_2(b+d) + (c+d) log_2(c+d))$$

Notes

To avoid NaNs, every logarithm is calculated as the logarithm of 1 greater than the value in question. (Python's math.log1p function is used.)

New in version 0.4.0.

Initialize Dunning instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in `_TokenDistance` for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in `_TokenDistance` for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

sim(*src*, *tar*)

Return the normalized Dunning similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Normalized Dunning similarity

Return type float

Examples

```
>>> cmp = Dunning()
>>> cmp.sim('cat', 'hat')
0.33462839191969423
>>> cmp.sim('Niall', 'Neil')
0.19229445539929793
>>> cmp.sim('aluminum', 'Catalan')
0.03220862737070572
>>> cmp.sim('ATCG', 'TAGC')
0.0010606026735052122
```

New in version 0.4.0.

sim_score(*src*, *tar*)

Return the Dunning similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Dunning similarity

Return type float

Examples

```
>>> cmp = Dunning()
>>> cmp.sim('cat', 'hat')
0.33462839191969423
>>> cmp.sim('Niall', 'Neil')
0.19229445539929793
>>> cmp.sim('aluminum', 'Catalan')
0.03220862737070572
>>> cmp.sim('ATCG', 'TAGC')
0.0010606026735052122
```

New in version 0.4.0.

class abydos.distance.**Euclidean**(*alphabet=0*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._minkowski.Minkowski

Euclidean distance.

Euclidean distance is the straigh-line or as-the-crow-flies distance, equivalent to Minkowski distance in L^2-space.

New in version 0.3.6.

Initialize Euclidean instance.

Parameters

- **alphabet** (*collection or int*) -- The values or size of the alphabet

- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in `_TokenDistance` for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

dist (*src*, *tar*)

Return the normalized Euclidean distance between two strings.

The normalized Euclidean distance is a distance metric in L^2-space, normalized to [0, 1].

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns The normalized Euclidean distance

Return type float

Examples

```
>>> cmp = Euclidean()
>>> round(cmp.dist('cat', 'hat'), 12)
0.57735026919
>>> round(cmp.dist('Niall', 'Neil'), 12)
0.683130051064
>>> round(cmp.dist('Colin', 'Cuilen'), 12)
0.727606875109
>>> cmp.dist('ATCG', 'TAGC')
1.0
```

New in version 0.3.0.

Changed in version 0.3.6: Encapsulated in class

dist_abs (*src*, *tar*, *normalized=False*)

Return the Euclidean distance between two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison
- **normalized** (*bool*) -- Normalizes to [0, 1] if True

Returns The Euclidean distance

Return type float

Examples

```
>>> cmp = Euclidean()
>>> cmp.dist_abs('cat', 'hat')
2.0
>>> round(cmp.dist_abs('Niall', 'Neil'), 12)
2.645751311065
>>> cmp.dist_abs('Colin', 'Cuilen')
3.0
>>> round(cmp.dist_abs('ATCG', 'TAGC'), 12)
3.162277660168
```

New in version 0.3.0.

Changed in version 0.3.6: Encapsulated in class

abydos.distance.**euclidean**(*src*, *tar*, *qval=2*, *normalized=False*, *alphabet=0*)

Return the Euclidean distance between two strings.

This is a wrapper for *Euclidean.dist_abs()*.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison
- **qval** (*int*) -- The length of each q-gram
- **normalized** (*bool*) -- Normalizes to [0, 1] if True
- **alphabet** (*collection or int*) -- The values or size of the alphabet

Returns float

Return type The Euclidean distance

Examples

```
>>> euclidean('cat', 'hat')
2.0
>>> round(euclidean('Niall', 'Neil'), 12)
2.645751311065
>>> euclidean('Colin', 'Cuilen')
3.0
>>> round(euclidean('ATCG', 'TAGC'), 12)
3.162277660168
```

New in version 0.3.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the Euclidean.dist_abs method instead.

abydos.distance.**dist_euclidean**(*src*, *tar*, *qval=2*, *alphabet=0*)

Return the normalized Euclidean distance between two strings.

This is a wrapper for *Euclidean.dist()*.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison
- **qval** (*int*) -- The length of each q-gram
- **alphabet** (*collection or int*) -- The values or size of the alphabet

Returns The normalized Euclidean distance

Return type float

Examples

```
>>> round(dist_euclidean('cat', 'hat'), 12)
0.57735026919
>>> round(dist_euclidean('Niall', 'Neil'), 12)
0.683130051064
>>> round(dist_euclidean('Colin', 'Cuilen'), 12)
0.727606875109
>>> dist_euclidean('ATCG', 'TAGC')
1.0
```

New in version 0.3.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the Euclidean.dist method instead.

abydos.distance.**sim_euclidean**(*src*, *tar*, *qval=2*, *alphabet=0*)

Return the normalized Euclidean similarity of two strings.

This is a wrapper for `Euclidean.sim()`.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison
- **qval** (*int*) -- The length of each q-gram
- **alphabet** (*collection or int*) -- The values or size of the alphabet

Returns The normalized Euclidean similarity

Return type float

Examples

```
>>> round(sim_euclidean('cat', 'hat'), 12)
0.42264973081
>>> round(sim_euclidean('Niall', 'Neil'), 12)
0.316869948936
>>> round(sim_euclidean('Colin', 'Cuilen'), 12)
0.272393124891
>>> sim_euclidean('ATCG', 'TAGC')
0.0
```

New in version 0.3.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the Euclidean.sim method instead.

class abydos.distance.**Eyraud**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Eyraud similarity.

For two sets X and Y and a population N, the Eyraud similarity [Eyr38] is

$$sim_{Eyraud}(X, Y) = \frac{|X \cap Y| - |X| \cdot |Y|}{|X| \cdot |Y| \cdot |N \setminus Y| \cdot |N \setminus X|}$$

For lack of access to the original, this formula is based on the concurring formulae presented in [Shi93] and [Hubalek08].

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$sim_{Eyraud} = \frac{a - (a+b)(a+c)}{(a+b)(a+c)(b+d)(c+d)}$$

New in version 0.4.0.

Initialize Eyraud instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in _TokenDistance for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the soft and fuzzy variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the fuzzy variant.

New in version 0.4.0.

sim(*src*, *tar*)

Return the normalized Eyraud similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Normalized Eyraud similarity

Return type float

Examples

```
>>> cmp = Eyraud()
>>> cmp.sim('cat', 'hat')
1.438198553583169e-06
>>> cmp.sim('Niall', 'Neil')
1.5399964580081465e-06
>>> cmp.sim('aluminum', 'Catalan')
1.6354719962967386e-06
>>> cmp.sim('ATCG', 'TAGC')
1.6478781097519779e-06
```

New in version 0.4.0.

sim_score(*src*, *tar*)
Return the Eyraud similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Eyraud similarity

Return type float

Examples

```
>>> cmp = Eyraud()
>>> cmp.sim_score('cat', 'hat')
-1.438198553583169e-06
>>> cmp.sim_score('Niall', 'Neil')
-1.5399964580081465e-06
>>> cmp.sim_score('aluminum', 'Catalan')
-1.6354719962967386e-06
>>> cmp.sim_score('ATCG', 'TAGC')
-1.6478781097519779e-06
```

New in version 0.4.0.

class abydos.distance.**FagerMcGowan**(*tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)
Bases: abydos.distance._token_distance._TokenDistance

Fager & McGowan similarity.

For two sets X and Y, the Fager & McGowan similarity [Fag57][FM63] is

$$sim_{FagerMcGowan}(X, Y) = \frac{|X \cap Y|}{\sqrt{|X| \cdot |Y|}} - \frac{1}{2\sqrt{max(|X|, |Y|)}}$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$sim_{FagerMcGowan} = \frac{a}{\sqrt{(a+b)(a+c)}} - \frac{1}{2\sqrt{max(a+b, a+c)}}$$

New in version 0.4.0.

Initialize FagerMcGowan instance.

Parameters

- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

sim(*src*, *tar*)

Return the normalized Fager & McGowan similarity of two strings.

As this similarity ranges from $(-\inf, 1.0)$, this normalization simply clamps the value to the range (0.0, 1.0).

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Normalized Fager & McGowan similarity

Return type float

Examples

```
>>> cmp = FagerMcGowan()
>>> cmp.sim('cat', 'hat')
0.25
>>> cmp.sim('Niall', 'Neil')
0.16102422643817918
>>> cmp.sim('aluminum', 'Catalan')
0.0
>>> cmp.sim('ATCG', 'TAGC')
0.0
```

New in version 0.4.0.

sim_score(*src*, *tar*)

Return the Fager & McGowan similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison

- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Fager & McGowan similarity

Return type float

Examples

```
>>> cmp = FagerMcGowan()
>>> cmp.sim_score('cat', 'hat')
0.25
>>> cmp.sim_score('Niall', 'Neil')
0.16102422643817918
>>> cmp.sim_score('aluminum', 'Catalan')
-0.048815536468908724
>>> cmp.sim_score('ATCG', 'TAGC')
-0.22360679774997896
```

New in version 0.4.0.

class abydos.distance.**Faith**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Faith similarity.

For two sets X and Y and a population N, the Faith similarity [Fai83] is

$$sim_{Faith}(X, Y) = \frac{|X \cap Y| + \frac{|(N \setminus X) \setminus Y|}{2}}{|N|}$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$sim_{Faith} = \frac{a + \frac{d}{2}}{n}$$

New in version 0.4.0.

Initialize Faith instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in _TokenDistance for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.

- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

sim(*src*, *tar*)
Return the Faith similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Faith similarity

Return type float

Examples

```
>>> cmp = Faith()
>>> cmp.sim('cat', 'hat')
0.4987244897959184
>>> cmp.sim('Niall', 'Neil')
0.4968112244897959
>>> cmp.sim('aluminum', 'Catalan')
0.4910828025477707
>>> cmp.sim('ATCG', 'TAGC')
0.49362244897959184
```

New in version 0.4.0.

class abydos.distance.**Fidelity**(*tokenizer=None*, ***kwargs*)
Bases: abydos.distance._token_distance._TokenDistance

Fidelity.

For two multisets X and Y drawn from an alphabet S, fidelity is

$$sim_{Fidelity}(X, Y) = \left(\sum_{i \in S} \sqrt{|\frac{A_i}{|A|} \cdot \frac{B_i}{|B|}|} \right)^2$$

New in version 0.4.0.

Initialize Fidelity instance.

Parameters

- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.

New in version 0.4.0.

sim(*src*, *tar*)
Return the fidelity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns fidelity

Return type float

Examples

```
>>> cmp = Fidelity()
>>> cmp.sim('cat', 'hat')
0.25
>>> cmp.sim('Niall', 'Neil')
0.1333333333333333
>>> cmp.sim('aluminum', 'Catalan')
0.013888888888888888
>>> cmp.sim('ATCG', 'TAGC')
0.0
```

New in version 0.4.0.

class abydos.distance.**Fleiss**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)
Bases: abydos.distance._token_distance._TokenDistance

Fleiss correlation.

For two sets X and Y and a population N, Fleiss correlation [Fle75] is

$$corr_{Fleiss}(X,Y) = \frac{(|X \cap Y| \cdot |(N \setminus X) \setminus Y| - |X \setminus Y| \cdot |Y \setminus X|) \cdot (|X| \cdot |N \setminus X| + |Y| \cdot |N \setminus Y|)}{2 \cdot |X| \cdot |N \setminus X| \cdot |Y| \cdot |N \setminus Y|}$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$corr_{Fleiss} = \frac{(ad - bc)((a+b)(c+d) + (a+c)(b+d))}{2(a+b)(c+d)(a+c)(b+d)}$$

This is Fleiss' $M(A_1)$, $ad - bc$ divided by the harmonic mean of the marginals $p_1q_1 = (a+b)(c+d)$ and $p_2q_2 = (a+c)(b+d)$.

New in version 0.4.0.

Initialize Fleiss instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in _TokenDistance for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package

- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in `_TokenDistance` for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

corr(*src*, *tar*)

Return the Fleiss correlation of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Fleiss correlation

Return type float

Examples

```
>>> cmp = Fleiss()
>>> cmp.corr('cat', 'hat')
0.49743589743589745
>>> cmp.corr('Niall', 'Neil')
0.3621712520061204
>>> cmp.corr('aluminum', 'Catalan')
0.10839724112919989
>>> cmp.corr('ATCG', 'TAGC')
-0.006418485237483954
```

New in version 0.4.0.

sim(*src*, *tar*)

Return the Fleiss similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Fleiss similarity

Return type float

Examples

```
>>> cmp = Fleiss()
>>> cmp.sim('cat', 'hat')
0.7487179487179487
>>> cmp.sim('Niall', 'Neil')
0.6810856260030602
>>> cmp.sim('aluminum', 'Catalan')
0.5541986205645999
>>> cmp.sim('ATCG', 'TAGC')
0.496790757381258
```

New in version 0.4.0.

class abydos.distance.**FleissLevinPaik**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Fleiss-Levin-Paik similarity.

For two sets X and Y and a population N, Fleiss-Levin-Paik similarity [FLP03] is

$$sim_{FleissLevinPaik}(X, Y) = \frac{2|(N \setminus X) \setminus Y|}{2|(N \setminus X) \setminus Y| + |X \setminus Y| + |Y \setminus X|}$$

This is [Mor12]'s 'd Specific Agreement'.

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$sim_{FleissLevinPaik} = \frac{2d}{2d + b + c}$$

New in version 0.4.0.

Initialize FleissLevinPaik instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in _TokenDistance for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the soft and fuzzy variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the fuzzy variant.

New in version 0.4.0.

sim(*src*, *tar*)

Return the Fleiss-Levin-Paik similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Fleiss-Levin-Paik similarity

Return type float

Examples

```
>>> cmp = FleissLevinPaik()
>>> cmp.sim('cat', 'hat')
0.9974358974358974
>>> cmp.sim('Niall', 'Neil')
0.9955041746949261
>>> cmp.sim('aluminum', 'Catalan')
0.9903412749517064
>>> cmp.sim('ATCG', 'TAGC')
0.993581514762516
```

New in version 0.4.0.

class abydos.distance.**ForbesI**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Forbes I similarity.

For two sets X and Y and a population N, the Forbes I similarity [For07][Moz36] is

$$sim_{ForbesI}(X, Y) = \frac{|N| \cdot |X \cap Y|}{|X| \cdot |Y|}$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$sim_{ForbesI} = \frac{na}{(a+b)(a+c)}$$

New in version 0.4.0.

Initialize ForbesI instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in _TokenDistance for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.

- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

sim(*src*, *tar*)

Return the normalized Forbes I similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Normalized Forbes I similarity

Return type float

Examples

```
>>> cmp = ForbesI()
>>> cmp.sim('cat', 'hat')
0.5
>>> cmp.sim('Niall', 'Neil')
0.3333333333333333
>>> cmp.sim('aluminum', 'Catalan')
0.11125283446712018
>>> cmp.sim('ATCG', 'TAGC')
0.0
```

New in version 0.4.0.

sim_score(*src*, *tar*)

Return the Forbes I similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Forbes I similarity

Return type float

Examples

```
>>> cmp = ForbesI()
>>> cmp.sim_score('cat', 'hat')
98.0
>>> cmp.sim_score('Niall', 'Neil')
52.266666666666666
>>> cmp.sim_score('aluminum', 'Catalan')
10.902777777777779
>>> cmp.sim_score('ATCG', 'TAGC')
0.0
```

New in version 0.4.0.

class abydos.distance.**ForbesII**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Forbes II correlation.

For two sets X and Y and a population N, the Forbes II correlation, as described in [For25], is

$$corr_{ForbesII}(X, Y) = \frac{|X \setminus Y| \cdot |Y \setminus X| - |X \cap Y| \cdot |(N \setminus X) \setminus Y|}{|X| \cdot |Y| - |N| \cdot min(|X|, |Y|)}$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$corr_{ForbesII} = \frac{bc - ad}{(a+b)(a+c) - n \cdot min(a+b, a+c)}$$

New in version 0.4.0.

Initialize ForbesII instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in _TokenDistance for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the soft and fuzzy variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the fuzzy variant.

New in version 0.4.0.

corr(*src*, *tar*)

Return the Forbes II correlation of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Forbes II correlation

Return type float

Examples

```
>>> cmp = ForbesII()
>>> cmp.corr('cat', 'hat')
0.49743589743589745
>>> cmp.corr('Niall', 'Neil')
0.3953727506426735
>>> cmp.corr('aluminum', 'Catalan')
0.11485180412371133
>>> cmp.corr('ATCG', 'TAGC')
-0.006418485237483954
```

New in version 0.4.0.

sim(*src*, *tar*)

Return the Forbes II similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Forbes II similarity

Return type float

Examples

```
>>> cmp = ForbesII()
>>> cmp.sim('cat', 'hat')
0.7487179487179487
>>> cmp.sim('Niall', 'Neil')
0.6976863753213367
>>> cmp.sim('aluminum', 'Catalan')
0.5574259020618557
>>> cmp.sim('ATCG', 'TAGC')
0.496790757381258
```

New in version 0.4.0.

class abydos.distance.**Fossum**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Fossum similarity.

For two sets X and Y and a population N, the Fossum similarity [FK66] is

$$sim_{Fossum}(X, Y) = \frac{|N| \cdot \left(|X \cap Y| - \frac{1}{2}\right)^2}{|X| \cdot |Y|}$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$sim_{Fossum} = \frac{n(a - \frac{1}{2})^2}{(a + b)(a + c)}$$

New in version 0.4.0.

Initialize Fossum instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in `_TokenDistance` for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in `_TokenDistance` for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

sim(*src*, *tar*)

Return the normalized Fossum similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Normalized Fossum similarity

Return type float

Examples

```
>>> cmp = Fossum()
>>> cmp.sim('cat', 'hat')
0.1836734693877551
>>> cmp.sim('Niall', 'Neil')
0.08925619834710742
>>> cmp.sim('aluminum', 'Catalan')
0.0038927335640138415
>>> cmp.sim('ATCG', 'TAGC')
0.01234567901234568
```

New in version 0.4.0.

sim_score(*src*, *tar*)

Return the Fossum similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Fossum similarity

Return type float

Examples

```
>>> cmp = Fossum()
>>> cmp.sim_score('cat', 'hat')
110.25
>>> cmp.sim_score('Niall', 'Neil')
58.8
>>> cmp.sim_score('aluminum', 'Catalan')
2.7256944444444446
>>> cmp.sim_score('ATCG', 'TAGC')
7.84
```

New in version 0.4.0.

class abydos.distance.**GeneralizedFleiss**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, *mean_func='arithmetic'*, *marginals='a'*, *proportional=False*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Generalized Fleiss correlation.

For two sets X and Y and a population N, Generalized Fleiss correlation is based on observations from [Fle75].

$$corr_{GeneralizedFleiss}(X, Y) = \frac{|X \cap Y| \cdot |(N \setminus X) \setminus Y| - |X \setminus Y| \cdot |Y \setminus X|}{\mu_{products\ of\ marginals}}$$

The mean function μ may be any of the mean functions in *abydos.stats*. The products of marginals may be one of the following:

- a : $|X| \cdot |N \setminus X|$ & $|Y| \cdot |N \setminus Y|$
- b : $|X| \cdot |Y|$ & $|N \setminus X| \cdot |N \setminus Y|$

- c : $|X| \cdot |N| \setminus Y|$ & $|Y| \cdot |N \setminus X|$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$corr_{GeneralizedFleiss} = \frac{ad - bc}{\mu_{products\ of\ marginals}}$$

And the products of marginals are:

- a : $p_1q_1 = (a+b)(c+d)$ & $p_2q_2 = (a+c)(b+d)$
- b : $p_1p_2 = (a+b)(a+c)$ & $q_1q_2 = (c+d)(b+d)$
- c : $p_1q_2 = (a+b)(b+d)$ & $p_2q_1 = (a+c)(c+d)$

New in version 0.4.0.

Initialize GeneralizedFleiss instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in _TokenDistance for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.
- **mean_func** (*str or function*) -- Specifies the mean function to use. A function taking a list of numbers as its only required argument may be supplied, or one of the following strings will select the specified mean function from *abydos.stats*:
  - arithmetic employs amean(), and this measure will be identical to *MaxwellPilliner* with otherwise default parameters
  - geometric employs gmean(), and this measure will be identical to *PearsonPhi* with otherwise default parameters
  - harmonic employs hmean(), and this measure will be identical to *Fleiss* with otherwise default parameters
  - ag employs the arithmetic-geometric mean agmean()
  - gh employs the geometric-harmonic mean ghmean()
  - agh employs the arithmetic-geometric-harmonic mean aghmean()
  - contraharmonic employs the contraharmonic mean cmean()
  - identric employs the identric mean imean()
  - logarithmic employs the logarithmic mean lmean()
  - quadratic employs the quadratic mean qmean()
  - heronian employs the Heronian mean heronian_mean()
  - hoelder employs the Hölder mean hoelder_mean()
  - lehmer employs the Lehmer mean lehmer_mean()
  - seiffert employs Seiffert's mean seiffert_mean()

- **marginals** (*str*) -- Specifies the pairs of marginals to multiply and calculate the resulting mean of. Can be:
  - a : $p_1q_1 = (a+b)(c+d)$ & $p_2q_2 = (a+c)(b+d)$
  - b : $p_1p_2 = (a+b)(a+c)$ & $q_1q_2 = (c+d)(b+d)$
  - c : $p_1q_2 = (a+b)(b+d)$ & $p_2q_1 = (a+c)(c+d)$
- **proportional** (*bool*) -- If true, each of the values, a, b, c, d and the marginals will be divided by the total $a+b+c+d = n$.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

corr(*src*, *tar*)

Return the Generalized Fleiss correlation of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Generalized Fleiss correlation

Return type float

Examples

```
>>> cmp = GeneralizedFleiss()
>>> cmp.corr('cat', 'hat')
0.49743589743589745
>>> cmp.corr('Niall', 'Neil')
0.35921989956790845
>>> cmp.corr('aluminum', 'Catalan')
0.10803030303030303
>>> cmp.corr('ATCG', 'TAGC')
-0.006418485237483954
```

New in version 0.4.0.

sim(*src*, *tar*)

Return the Generalized Fleiss similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Generalized Fleiss similarity

Return type float

Examples

```
>>> cmp = GeneralizedFleiss()
>>> cmp.sim('cat', 'hat')
0.7487179487179487
>>> cmp.sim('Niall', 'Neil')
0.6796099497839543
>>> cmp.sim('aluminum', 'Catalan')
0.5540151515151515
>>> cmp.sim('ATCG', 'TAGC')
0.496790757381258
```

New in version 0.4.0.

class abydos.distance.**Gilbert**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Gilbert correlation.

For two sets X and Y and a population N, the Gilbert correlation [Gil84] is

$$corr_{Gilbert}(X, Y) = \frac{2(|X \cap Y| \cdot |(N \setminus X) \setminus Y| - |X \setminus Y| \cdot |Y \setminus X|)}{|N|^2 - |X \cap Y|^2 + |X \setminus Y|^2 + |Y \setminus X|^2 - |(N \setminus X) \setminus Y|^2}$$

For lack of access to the original, this formula is based on the concurring formulae presented in [Pei84] and [Doo84].

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$corr_{Gilbert} = \frac{2(ad - cd)}{n^2 - a^2 + b^2 + c^2 - d^2}$$

New in version 0.4.0.

Initialize Gilbert instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in _TokenDistance for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the soft and fuzzy variants.

- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

corr(*src*, *tar*)

Return the Gilbert correlation of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Gilbert correlation

Return type float

Examples

```
>>> cmp = Gilbert()
>>> cmp.corr('cat', 'hat')
0.3310580204778157
>>> cmp.corr('Niall', 'Neil')
0.21890122402504983
>>> cmp.corr('aluminum', 'Catalan')
0.057094811018577836
>>> cmp.corr('ATCG', 'TAGC')
-0.003198976327575176
```

New in version 0.4.0.

sim(*src*, *tar*)

Return the Gilbert similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Gilbert similarity

Return type float

Examples

```
>>> cmp = Gilbert()
>>> cmp.sim('cat', 'hat')
0.6655290102389079
>>> cmp.sim('Niall', 'Neil')
0.6094506120125249
>>> cmp.sim('aluminum', 'Catalan')
0.5285474055092889
>>> cmp.sim('ATCG', 'TAGC')
0.4984005118362124
```

New in version 0.4.0.

class abydos.distance.**GilbertWells**(*alphabet=None*, *tokenizer=None*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Gilbert & Wells similarity.

For two sets X and Y and a population N, the Gilbert & Wells similarity [GW66] is

$$sim_{GilbertWells}(X,Y) = ln\frac{|N|^3}{2\pi|X|\cdot|Y|\cdot|N\setminus Y|\cdot|N\setminus X|} + 2ln\frac{|N|!\cdot|X\cap Y|!\cdot|X\setminus Y|!\cdot|Y\setminus X|!\cdot|(N\setminus X)\setminus Y|!}{|X|!\cdot|Y|!\cdot|N\setminus Y|!\cdot|N\setminus X|!}$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$sim_{GilbertWells} = ln\frac{n^3}{2\pi(a+b)(a+c)(b+d)(c+d)} + 2ln\frac{n!a!b!c!d!}{(a+b)!(a+c)!(b+d)!(c+d)!}$$

Notes

Most lists of similarity & distance measures, including [Hubalek08][CCT10][Mor12] have a quite different formula, which would be $ln\ a - ln\ b - ln\frac{a+b}{n} - ln\frac{a+c}{n} = ln\frac{an}{(a+b)(a+c)}$. However, neither this formula nor anything similar or equivalent to it appears anywhere within the cited work, [GW66]. See :class:UnknownF for this, alternative, measure.

New in version 0.4.0.

Initialize GilbertWells instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in _TokenDistance for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.

New in version 0.4.0.

sim(*src*, *tar*)

Return the normalized Gilbert & Wells similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Normalized Gilbert & Wells similarity

Return type float

Examples

```
>>> cmp = GilbertWells()
>>> cmp.sim('cat', 'hat')
0.4116913723876516
>>> cmp.sim('Niall', 'Neil')
0.2457247406857589
>>> cmp.sim('aluminum', 'Catalan')
0.05800001636414742
>>> cmp.sim('ATCG', 'TAGC')
0.028716013247135602
```

New in version 0.4.0.

sim_score(*src*, *tar*)

Return the Gilbert & Wells similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Gilbert & Wells similarity

Return type float

Examples

```
>>> cmp = GilbertWells()
>>> cmp.sim_score('cat', 'hat')
20.17617447734673
>>> cmp.sim_score('Niall', 'Neil')
16.717742356982733
>>> cmp.sim_score('aluminum', 'Catalan')
5.495096667524002
>>> cmp.sim_score('ATCG', 'TAGC')
1.6845961909440712
```

New in version 0.4.0.

class abydos.distance.**GiniI**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, *normalizer='proportional'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Gini I correlation.

For two sets X and Y and a population N, Gini I correlation [Gin12], using the formula from [GK59], is

$$corr_{GiniI}(X, Y) = \frac{\frac{|X \cap Y| + |(N \setminus X) \setminus Y|}{|N|} - \frac{|X| \cdot |Y|}{|N|} + \frac{|N \setminus Y| \cdot |N \setminus X|}{|N|}}{\sqrt{(1 - (\frac{|X|}{|N|}^2 + \frac{|Y|}{|N|}^2)) \cdot (1 - (\frac{|N \setminus Y|}{|N|}^2 + \frac{|N \setminus X|}{|N|}^2))}}$$

In 2x2 confusion table terms, where a+b+c+d=n, after each term has been converted to a proportion by dividing by n, this is

$$corr_{GiniI} = \frac{(a+d) - (a+b)(a+c) + (b+d)(c+d)}{\sqrt{(1-((a+b)^2+(c+d)^2)) \cdot (1-((a+c)^2+(b+d)^2))}}$$

New in version 0.4.0.

Initialize GiniI instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in `_TokenDistance` for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in `_TokenDistance` for details.
- **normalizer** (*str*) -- Specifies the normalization type. See normalizer description in `_TokenDistance` for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

corr(*src*, *tar*)

Return the Gini I correlation of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Gini I correlation

Return type float

Examples

```
>>> cmp = GiniI()
>>> cmp.corr('cat', 'hat')
0.49722814498933254
>>> cmp.corr('Niall', 'Neil')
0.39649090262533215
>>> cmp.corr('aluminum', 'Catalan')
0.14887105223941113
>>> cmp.corr('ATCG', 'TAGC')
-0.006418485237489576
```

New in version 0.4.0.

sim(*src*, *tar*)
Return the normalized Gini I similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Normalized Gini I similarity

Return type float

Examples

```
>>> cmp = GiniI()
>>> cmp.sim('cat', 'hat')
0.7486140724946663
>>> cmp.sim('Niall', 'Neil')
0.6982454513126661
>>> cmp.sim('aluminum', 'Catalan')
0.5744355261197056
>>> cmp.sim('ATCG', 'TAGC')
0.4967907573812552
```

New in version 0.4.0.

class abydos.distance.**GiniII**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, *normalizer='proportional'*, ***kwargs*)
Bases: abydos.distance._token_distance._TokenDistance

Gini II distance.

For two sets X and Y and a population N, Gini II correlation [Gin15], using the formula from [GK59], is

$$corr_{GiniII}(X, Y) = \frac{\frac{|X \cap Y| + |(N \setminus X) \setminus Y|}{|N|} - (\frac{|X| \cdot |Y|}{|N|} + \frac{|N \setminus Y| \cdot |N \setminus X|}{|N|})}{1 - |\frac{|Y \setminus X| - |X \setminus Y|}{|N|}| - (\frac{|X| \cdot |Y|}{|N|} + \frac{|N \setminus Y| \cdot |N \setminus X|}{|N|})}$$

In 2x2 confusion table terms, where a+b+c+d=n, after each term has been converted to a proportion by dividing by n, this is

$$corr_{GiniII} = \frac{(a+d) - ((a+b)(a+c) + (b+d)(c+d))}{1 - |b-c| - ((a+b)(a+c) + (b+d)(c+d))}$$

New in version 0.4.0.

Initialize GiniII instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in _TokenDistance for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package

- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in `_TokenDistance` for details.
- **normalizer** (*str*) -- Specifies the normalization type. See normalizer description in `_TokenDistance` for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

corr(*src*, *tar*)

Return the Gini II correlation of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Gini II correlation

Return type float

Examples

```
>>> cmp = GiniII()
>>> cmp.corr('cat', 'hat')
0.49722814498933254
>>> cmp.corr('Niall', 'Neil')
0.4240703425535771
>>> cmp.corr('aluminum', 'Catalan')
0.15701415701415936
>>> cmp.corr('ATCG', 'TAGC')
-0.006418485237489576
```

New in version 0.4.0.

sim(*src*, *tar*)

Return the normalized Gini II similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Normalized Gini II similarity

Return type float

Examples

```
>>> cmp = GiniII()
>>> cmp.sim('cat', 'hat')
0.7486140724946663
>>> cmp.sim('Niall', 'Neil')
0.7120351712767885
>>> cmp.sim('aluminum', 'Catalan')
0.5785070785070797
>>> cmp.sim('ATCG', 'TAGC')
0.4967907573812552
```

New in version 0.4.0.

class abydos.distance.**Goodall**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Goodall similarity.

For two sets X and Y and a population N, Goodall similarity [Goo67][AC77] is an angular transformation of Sokal & Michener's simple matching coefficient

$$sim_{Goodall}(X, Y) = \frac{2}{\pi} \sin^{-1}\left(\sqrt{\frac{|X \cap Y| + |(N \setminus X) \setminus Y|}{|N|}}\right)$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$sim_{Goodall} = \frac{2}{\pi} \sin^{-1}\left(\sqrt{\frac{a+d}{n}}\right)$$

New in version 0.4.0.

Initialize Goodall instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in _TokenDistance for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the soft and fuzzy variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the fuzzy variant.

New in version 0.4.0.

sim(*src*, *tar*)

Return the Goodall similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Goodall similarity

Return type float

Examples

```
>>> cmp = Goodall()
>>> cmp.sim('cat', 'hat')
0.9544884026871964
>>> cmp.sim('Niall', 'Neil')
0.9397552079794624
>>> cmp.sim('aluminum', 'Catalan')
0.9117156301536503
>>> cmp.sim('ATCG', 'TAGC')
0.9279473952929225
```

New in version 0.4.0.

class abydos.distance.**GoodmanKruskalLambda**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Goodman & Kruskal's Lambda similarity.

For two sets X and Y and a population N, Goodman & Kruskal's lambda [GK54] is

$$sim_{GK_\lambda}(X, Y) = \frac{\frac{1}{2}(max(|X \cap Y|, |X \setminus Y|) + max(|Y \setminus X|, |(N \setminus X) \setminus Y|) + max(|X \cap Y|, |Y \setminus X|) + max(|X \setminus Y$$
$$|N| - \frac{1}{2}(max(|X|, |N \setminus X|) + max(|Y|, |N$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$sim_{GK_\lambda} = \frac{\frac{1}{2}((max(a, b) + max(c, d) + max(a, c) + max(b, d)) - (max(a + b, c + d) + max(a + c, b + d)))}{n - \frac{1}{2}(max(a + b, c + d) + max(a + c, b + d))}$$

New in version 0.4.0.

Initialize GoodmanKruskalLambda instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in _TokenDistance for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package

- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in `_TokenDistance` for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

sim(*src*, *tar*)

Return Goodman & Kruskal's Lambda similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Goodman & Kruskal's Lambda similarity

Return type float

Examples

```
>>> cmp = GoodmanKruskalLambda()
>>> cmp.sim('cat', 'hat')
0.0
>>> cmp.sim('Niall', 'Neil')
0.0
>>> cmp.sim('aluminum', 'Catalan')
0.0
>>> cmp.sim('ATCG', 'TAGC')
0.0
```

New in version 0.4.0.

class abydos.distance.**GoodmanKruskalLambdaR**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: `abydos.distance._token_distance._TokenDistance`

Goodman & Kruskal Lambda-r correlation.

For two sets X and Y and a population N, Goodman & Kruskal λ_r correlation [GK54] is

$$corr_{GK_{\lambda_r}}(X,Y) = \frac{|X \cap Y| + |(N \setminus X) \setminus Y| - \frac{1}{2}(max(|X|, |N \setminus X|) + max(|Y|, |N \setminus Y|))}{|N| - \frac{1}{2}(max(|X|, |N \setminus X|) + max(|Y|, |N \setminus Y|))}$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$corr_{GK_{\lambda_r}} = \frac{a + d - \frac{1}{2}(max(a+b, c+d) + max(a+c, b+d))}{n - \frac{1}{2}(max(a+b, c+d) + max(a+c, b+d))}$$

New in version 0.4.0.

Initialize GoodmanKruskalLambdaR instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in `_TokenDistance` for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in `_TokenDistance` for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

corr(*src*, *tar*)

Return Goodman & Kruskal Lambda-r correlation of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Goodman & Kruskal Lambda-r correlation

Return type float

Examples

```
>>> cmp = GoodmanKruskalLambdaR()
>>> cmp.corr('cat', 'hat')
0.0
>>> cmp.corr('Niall', 'Neil')
-0.2727272727272727
>>> cmp.corr('aluminum', 'Catalan')
-0.7647058823529411
>>> cmp.corr('ATCG', 'TAGC')
-1.0
```

New in version 0.4.0.

sim(*src*, *tar*)

Return Goodman & Kruskal Lambda-r similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison

- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Goodman & Kruskal Lambda-r similarity

Return type float

Examples

```
>>> cmp = GoodmanKruskalLambdaR()
>>> cmp.sim('cat', 'hat')
0.5
>>> cmp.sim('Niall', 'Neil')
0.36363636363636365
>>> cmp.sim('aluminum', 'Catalan')
0.11764705882352944
>>> cmp.sim('ATCG', 'TAGC')
0.0
```

New in version 0.4.0.

class abydos.distance.**GoodmanKruskalTauA**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, *normalizer='proportional'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Goodman & Kruskal's Tau A similarity.

For two sets X and Y and a population N, Goodman & Kruskal's τ_a similarity [GK54], by analogy with τ_b, is

$$sim_{GK_{\tau_a}}(X,Y) = \frac{\frac{|X \cap Y|^2}{|N|} + \frac{|Y \setminus X|^2}{|N|}}{\frac{|Y|}{|N|}} + \frac{\frac{|X \setminus Y|^2}{|N|} + \frac{|(N \setminus X) \setminus Y|^2}{|N|}}{\frac{|N \setminus X|}{|N|}} - (\frac{|X|}{|N|}^2 + \frac{|N \setminus X|}{|N|}^2)}{1 - (\frac{|X|}{|N|}^2 + \frac{|N \setminus X|}{|N|}^2)}$$

In 2x2 confusion table terms, where a+b+c+d=n, after each term has been converted to a proportion by dividing by n, this is

$$sim_{GK_{\tau_a}} = \frac{\frac{a^2+c^2}{a+c} + \frac{b^2+d^2}{b+d} - ((a+b)^2 + (c+d)^2)}{1 - ((a+b)^2 + (c+d)^2)}$$

New in version 0.4.0.

Initialize GoodmanKruskalTauA instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in _TokenDistance for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.
- **normalizer** (*str*) -- Specifies the normalization type. See normalizer description in _TokenDistance for details.

- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

sim(*src*, *tar*)

Return Goodman & Kruskal's Tau A similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Goodman & Kruskal's Tau A similarity

Return type float

Examples

```
>>> cmp = GoodmanKruskalTauA()
>>> cmp.sim('cat', 'hat')
0.3304969657208484
>>> cmp.sim('Niall', 'Neil')
0.22137604585914503
>>> cmp.sim('aluminum', 'Catalan')
0.05991264724130685
>>> cmp.sim('ATCG', 'TAGC')
4.119695274745721e-05
```

New in version 0.4.0.

class abydos.distance.**GoodmanKruskalTauB**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, *normalizer='proportional'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Goodman & Kruskal's Tau B similarity.

For two sets X and Y and a population N, Goodman & Kruskal's τ_b similarity [GK54] is

$$sim_{GK_{\tau_b}}(X, Y) = \frac{\frac{\frac{|X \cap Y|}{|N|}^2 + \frac{|X \setminus Y|}{|N|}^2}{\frac{|X|}{|N|}} + \frac{\frac{|Y \setminus X|}{|N|}^2 + \frac{|(N \setminus X) \setminus Y|}{|N|}^2}{\frac{|N \setminus X|}{|N|}} - (\frac{|Y|}{|N|}^2 + \frac{|N \setminus Y|}{|N|}^2)}{1 - (\frac{|Y|}{|N|}^2 + \frac{|N \setminus Y|}{|N|}^2)}$$

In 2x2 confusion table terms, where a+b+c+d=n, after each term has been converted to a proportion by dividing by n, this is

$$sim_{GK_{\tau_b}} = \frac{\frac{a^2+b^2}{a+b} + \frac{c^2+d^2}{c+d} - ((a+c)^2 + (b+d)^2)}{1 - ((a+c)^2 + (b+d)^2)}$$

New in version 0.4.0.

Initialize GoodmanKruskalTauB instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in `_TokenDistance` for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in `_TokenDistance` for details.
- **normalizer** (*str*) -- Specifies the normalization type. See normalizer description in `_TokenDistance` for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

sim(*src*, *tar*)

Return Goodman & Kruskal's Tau B similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Goodman & Kruskal's Tau B similarity

Return type float

Examples

```
>>> cmp = GoodmanKruskalTauB()
>>> cmp.sim('cat', 'hat')
0.3304969657208484
>>> cmp.sim('Niall', 'Neil')
0.2346006486710202
>>> cmp.sim('aluminum', 'Catalan')
0.06533810992392582
>>> cmp.sim('ATCG', 'TAGC')
4.119695274745721e-05
```

New in version 0.4.0.

class abydos.distance.**GowerLegendre**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, *theta=0.5*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Gower & Legendre similarity.

For two sets X and Y and a population N, the Gower & Legendre similarity [GL86] is

$$sim_{GowerLegendre}(X, Y) = \frac{|X \cap Y| + |(N \setminus X) \setminus Y|}{|X \cap Y| + |(N \setminus X) \setminus Y| + \theta \cdot |X \triangle Y|}$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$sim_{GowerLegendre} = \frac{a + d}{a + \theta(b + c) + d}$$

New in version 0.4.0.

Initialize GowerLegendre instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in _TokenDistance for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.
- **theta** (*float*) -- The weight to place on the symmetric difference.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the soft and fuzzy variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the fuzzy variant.

New in version 0.4.0.

sim(*src*, *tar*)

Return the Gower & Legendre similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Gower & Legendre similarity

Return type float

Examples

```
>>> cmp = GowerLegendre()
>>> cmp.sim('cat', 'hat')
0.9974424552429667
>>> cmp.sim('Niall', 'Neil')
0.9955156950672646
>>> cmp.sim('aluminum', 'Catalan')
0.9903536977491961
>>> cmp.sim('ATCG', 'TAGC')
0.993581514762516
```

New in version 0.4.0.

class abydos.distance.**GuttmanLambdaA**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Guttman's Lambda A similarity.

For two sets X and Y and a population N, Guttman's λ_a similarity [Gut41] is

$$sim_{Guttman_{\lambda_a}}(X, Y) = \frac{max(|X \cap Y|, |Y \setminus X|) + max(|X \setminus Y|, |(N \setminus X) \setminus Y|) - max(|X|, |N \setminus X|)}{|N| - max(|X|, |N \setminus X|)}$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$sim_{Guttman_{\lambda_a}} = \frac{max(a, c) + max(b, d) - max(a + b, c + d)}{n - max(a + b, c + d)}$$

New in version 0.4.0.

Initialize GuttmanLambdaA instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in _TokenDistance for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the soft and fuzzy variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the fuzzy variant.

New in version 0.4.0.

sim(*src*, *tar*)
Return the Guttman Lambda A similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Guttman's Lambda A similarity

Return type float

Examples

```
>>> cmp = GuttmanLambdaA()
>>> cmp.sim('cat', 'hat')
0.0
>>> cmp.sim('Niall', 'Neil')
0.0
>>> cmp.sim('aluminum', 'Catalan')
0.0
>>> cmp.sim('ATCG', 'TAGC')
0.0
```

New in version 0.4.0.

class abydos.distance.**GuttmanLambdaB**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)
Bases: abydos.distance._token_distance._TokenDistance

Guttman's Lambda B similarity.

For two sets X and Y and a population N, Guttman's λ_b similarity [Gut41] is

$$sim_{Guttman_{\lambda_b}}(X, Y) = \frac{max(|X \cap Y|, |X \setminus Y|) + max(|Y \setminus X|, |(N \setminus X) \setminus Y|) - max(|Y|, |N \setminus Y|)}{|N| - max(|Y|, |N \setminus Y|)}$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$sim_{Guttman_{\lambda_b}} = \frac{max(a, b) + max(c, d) - max(a + c, b + d)}{n - max(a + c, b + d)}$$

New in version 0.4.0.

Initialize GuttmanLambdaB instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in _TokenDistance for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

sim(*src*, *tar*)

Return the Guttman Lambda B similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Guttman's Lambda B similarity

Return type float

Examples

```
>>> cmp = GuttmanLambdaB()
>>> cmp.sim('cat', 'hat')
0.0
>>> cmp.sim('Niall', 'Neil')
0.0
>>> cmp.sim('aluminum', 'Catalan')
0.0
>>> cmp.sim('ATCG', 'TAGC')
0.0
```

New in version 0.4.0.

class abydos.distance.**GwetAC**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Gwet's AC correlation.

For two sets X and Y and a population N, Gwet's AC correlation [Gwe08] is

$$corr_{Gwet_{AC}}(X, Y) = AC = \frac{p_o - p_e^{AC}}{1 - p_e^{AC}}$$

where

$$\begin{aligned} p_o &= \frac{|X \cap Y| + |(N \setminus X) \setminus Y|}{|N|} \\ p_e^{AC} &= \frac{1}{2}\left(\frac{|X|+|Y|}{|N|} \cdot \frac{|X \setminus Y| + |Y \setminus X|}{|N|}\right) \end{aligned}$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$\begin{aligned} p_o &= \frac{a+d}{n} \\ p_e^{AC} &= \frac{1}{2}\left(\frac{2a+b+c}{n} \cdot \frac{2d+b+c}{n}\right) \end{aligned}$$

New in version 0.4.0.

Initialize GwetAC instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in _TokenDistance for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the soft and fuzzy variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the fuzzy variant.

New in version 0.4.0.

corr(*src*, *tar*)

Return the Gwet's AC correlation of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Gwet's AC correlation

Return type float

Examples

```
>>> cmp = GwetAC()
>>> cmp.corr('cat', 'hat')
0.9948456319360438
>>> cmp.corr('Niall', 'Neil')
0.990945276504824
>>> cmp.corr('aluminum', 'Catalan')
0.9804734301840141
>>> cmp.corr('ATCG', 'TAGC')
0.9870811678360627
```

New in version 0.4.0.

sim(*src*, *tar*)
Return the Gwet's AC similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Gwet's AC similarity

Return type float

Examples

```
>>> cmp = GwetAC()
>>> cmp.sim('cat', 'hat')
0.9974228159680218
>>> cmp.sim('Niall', 'Neil')
0.995472638252412
>>> cmp.sim('aluminum', 'Catalan')
0.9902367150920071
>>> cmp.sim('ATCG', 'TAGC')
0.9935405839180314
```

New in version 0.4.0.

class abydos.distance.**Hamann**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)
Bases: abydos.distance._token_distance._TokenDistance

Hamann correlation.

For two sets X and Y and a population N, the Hamann correlation [Ham61] is

$$corr_{Hamann}(X, Y) = \frac{|X \cap Y| + |(N \setminus X) \setminus Y| - |X \setminus Y| - |Y \setminus X|}{|N|}$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$corr_{Hamann} = \frac{a + d - b - c}{n}$$

New in version 0.4.0.

Initialize Hamann instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in _TokenDistance for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

corr(*src*, *tar*)

Return the Hamann correlation of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Hamann correlation

Return type float

Examples

```
>>> cmp = Hamann()
>>> cmp.corr('cat', 'hat')
0.9897959183673469
>>> cmp.corr('Niall', 'Neil')
0.9821428571428571
>>> cmp.corr('aluminum', 'Catalan')
0.9617834394904459
>>> cmp.corr('ATCG', 'TAGC')
0.9744897959183674
```

New in version 0.4.0.

sim(*src*, *tar*)

Return the normalized Hamann similarity of two strings.

Hamann similarity, which has a range [-1, 1] is normalized to [0, 1] by adding 1 and dividing by 2.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Normalized Hamann similarity

Return type float

Examples

```
>>> cmp = Hamann()
>>> cmp.sim('cat', 'hat')
0.9948979591836735
>>> cmp.sim('Niall', 'Neil')
0.9910714285714286
>>> cmp.sim('aluminum', 'Catalan')
0.9808917197452229
>>> cmp.sim('ATCG', 'TAGC')
0.9872448979591837
```

New in version 0.4.0.

class abydos.distance.**HarrisLahey**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Harris & Lahey similarity.

For two sets X and Y and a population N, Harris & Lahey similarity [HL78] is

$$sim_{HarrisLahey}(X, Y) = \frac{|X \cap Y|}{|X \cup Y|} \cdot \frac{|N \setminus Y| + |N \setminus X|}{2|N|} + \frac{|(N \setminus X) \setminus Y|}{|N \setminus (X \cap Y)|} \cdot \frac{|X| + |Y|}{2|N|}$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$sim_{HarrisLahey} = \frac{a}{a+b+c} \cdot \frac{2d+b+c}{2n} + \frac{d}{d+b+c} \cdot \frac{2a+b+c}{2n}$$

Notes

Most catalogs of similarity coefficients [War08][Mor12][Xia13] omit the n terms in the denominators, but the worked example in [HL78] makes it clear that this is intended in the original.

New in version 0.4.0.

Initialize HarrisLahey instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in _TokenDistance for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the soft and fuzzy variants.

- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

sim(*src*, *tar*)
Return the Harris & Lahey similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Harris & Lahey similarity

Return type float

Examples

```
>>> cmp = HarrisLahey()
>>> cmp.sim('cat', 'hat')
0.3367085964820711
>>> cmp.sim('Niall', 'Neil')
0.22761577457069784
>>> cmp.sim('aluminum', 'Catalan')
0.07244410503054725
>>> cmp.sim('ATCG', 'TAGC')
0.006296204706372345
```

New in version 0.4.0.

class abydos.distance.**Hassanat**(*tokenizer=None*, ***kwargs*)
Bases: abydos.distance._token_distance._TokenDistance

Hassanat distance.

For two multisets X and Y drawn from an alphabet S, Hassanat distance [Has14] is

$$dist_{Hassanat}(X, Y) = \sum_{i \in S} D(X_i, Y_i)$$

where

$$D(X_i, Y_i) = \begin{cases} 1 - \frac{1+min(X_i,Y_i)}{1+max(X_i,Y_i)} & , min(X_i, Y_i) \geq 0 \\ 1 - \frac{1+min(X_i,Y_i)+|min(X_i,Y_i)|}{1+max(X_i,Y_i)+|min(X_i,Y_i)|} & , min(X_i, Y_i) < 0 \end{cases}$$

New in version 0.4.0.

Initialize Hassanat instance.

Parameters

- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.

New in version 0.4.0.

dist(*src*, *tar*)

Return the normalized Hassanat distance of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Normalized Hassanat distance

Return type float

Examples

```
>>> cmp = Hassanat()
>>> cmp.dist('cat', 'hat')
0.3333333333333333
>>> cmp.dist('Niall', 'Neil')
0.3888888888888889
>>> cmp.dist('aluminum', 'Catalan')
0.4777777777777778
>>> cmp.dist('ATCG', 'TAGC')
0.5
```

New in version 0.4.0.

dist_abs(*src*, *tar*)

Return the Hassanat distance of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Hassanat distance

Return type float

Examples

```
>>> cmp = Hassanat()
>>> cmp.dist_abs('cat', 'hat')
2.0
>>> cmp.dist_abs('Niall', 'Neil')
3.5
>>> cmp.dist_abs('aluminum', 'Catalan')
7.166666666666667
>>> cmp.dist_abs('ATCG', 'TAGC')
5.0
```

New in version 0.4.0.

class abydos.distance.**HawkinsDotson**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Hawkins & Dotson similarity.

For two sets X and Y and a population N, Hawkins & Dotson similarity [HD73] is the mean of the occurrence agreement and non-occurrence agreement

$$sim_{HawkinsDotson}(X, Y) = \frac{1}{2} \cdot \left(\frac{|X \cap Y|}{|X \cup Y|} + \frac{|(N \setminus X) \setminus Y|}{|N \setminus (X \cap Y)|} \right)$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$sim_{HawkinsDotson} = \frac{1}{2} \cdot \left(\frac{a}{a+b+c} + \frac{d}{b+c+d} \right)$$

New in version 0.4.0.

Initialize HawkinsDotson instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in _TokenDistance for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the soft and fuzzy variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the fuzzy variant.

New in version 0.4.0.

sim(*src*, *tar*)

Return the Hawkins & Dotson similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Hawkins & Dotson similarity

Return type float

Examples

```
>>> cmp = HawkinsDotson()
>>> cmp.sim('cat', 'hat')
0.6641091219096334
>>> cmp.sim('Niall', 'Neil')
0.606635407786303
>>> cmp.sim('aluminum', 'Catalan')
0.5216836734693877
>>> cmp.sim('ATCG', 'TAGC')
0.49362244897959184
```

New in version 0.4.0.

class abydos.distance.**Hellinger**(*tokenizer=None*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Hellinger distance.

For two multisets X and Y drawn from an alphabet S, Hellinger distance [Hel09] is

$$dist_{Hellinger}(X, Y) = \sqrt{2 \cdot \sum_{i \in S} (\sqrt{|A_i|} - \sqrt{|B_i|})^2}$$

New in version 0.4.0.

Initialize Hellinger instance.

Parameters

- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.

New in version 0.4.0.

dist(*src*, *tar*)

Return the normalized Hellinger distance of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Normalized Hellinger distance

Return type float

Examples

```
>>> cmp = Hellinger()
>>> cmp.dist('cat', 'hat')
0.8164965809277261
>>> cmp.dist('Niall', 'Neil')
0.881917103688197
>>> cmp.dist('aluminum', 'Catalan')
0.9128709291752769
>>> cmp.dist('ATCG', 'TAGC')
1.0
```

New in version 0.4.0.

dist_abs(*src*, *tar*)

Return the Hellinger distance of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Hellinger distance

Return type float

Examples

```
>>> cmp = Hellinger()
>>> cmp.dist_abs('cat', 'hat')
2.8284271247461903
>>> cmp.dist_abs('Niall', 'Neil')
3.7416573867739413
>>> cmp.dist_abs('aluminum', 'Catalan')
5.477225575051661
>>> cmp.dist_abs('ATCG', 'TAGC')
4.47213595499958
```

New in version 0.4.0.

class abydos.distance.**HendersonHeron**(***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Henderson-Heron dissimilarity.

For two sets X and Y and a population N, Henderson-Heron dissimilarity [HH77] is:

New in version 0.4.1.

Initialize HendersonHeron instance.

Parameters ****kwargs** -- Arbitrary keyword arguments

New in version 0.4.1.

dist(*src*, *tar*)

Return the Henderson-Heron dissimilarity of two strings.

Parameters

- **src** (*str*) -- Source string for comparison

- **tar** (*str*) -- Target string for comparison

Returns Henderson-Heron dissimilarity

Return type float

Examples

```
>>> cmp = HendersonHeron()
>>> cmp.dist('cat', 'hat')
0.00011668873858680838
>>> cmp.dist('Niall', 'Neil')
0.00048123075776606097
>>> cmp.dist('aluminum', 'Catalan')
0.08534181060514882
>>> cmp.dist('ATCG', 'TAGC')
0.9684367974410505
```

New in version 0.4.1.

class abydos.distance.**HornMorisita**(***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Horn-Morisita index of overlap.

Horn-Morisita index of overlap [Hor66], given two populations X and Y drawn from S species, is:

$$sim_{Horn-Morisita}(X, Y) = C_\lambda = \frac{2\sum_{i=1}^{S} x_i y_i}{(\hat{\lambda}_x + \hat{\lambda}_y)XY}$$

where

$$X = \sum_{i=1}^{S} x_i \;\; ; \;\; Y = \sum_{i=1}^{S} y_i$$

$$\hat{\lambda}_x = \frac{\sum_{i=1}^{S} x_i^2}{X^2} \;\; ; \;\; \hat{\lambda}_y = \frac{\sum_{i=1}^{S} y_i^2}{Y^2}$$

Observe that this is identical to Morisita similarity, except for the definition of the λ values in the denominator.

New in version 0.4.1.

Initialize HornMorisita instance.

Parameters ****kwargs** -- Arbitrary keyword arguments

New in version 0.4.1.

sim(*src*, *tar*)

Return the Horn-Morisita similarity of two strings.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns Horn-Morisita similarity

Return type float

Examples

```
>>> cmp = HornMorisita()
>>> cmp.sim('cat', 'hat')
0.5
>>> cmp.sim('Niall', 'Neil')
0.3636363636363636
>>> cmp.sim('aluminum', 'Catalan')
0.10650887573964497
>>> cmp.sim('ATCG', 'TAGC')
0.0
```

New in version 0.4.1.

class abydos.distance.**Hurlbert**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Hurlbert correlation.

In 2x2 confusion table terms, where a+b+c+d=n, Hurlbert's coefficient of interspecific association [Hur69] is

$$corr_{Hurlbert} = \frac{ad - bc}{|ad - bc|} \sqrt{\frac{Obs_{\chi^2} - Min_{\chi^2}}{Max_{\chi^2} - Min_{\chi^2}}}$$

Where:

$$\begin{array}{lll} Obs_{\chi^2} & = \frac{(ad-bc)^2 n}{(a+b)(a+c)(b+d)(c+d)} & \\ Max_{\chi^2} & = \frac{(a+b)(b+d)n}{(a+c)(c+d)} & \text{when } ad \geq bc \\ Max_{\chi^2} & = \frac{(a+b)(a+c)n}{(b+d)(c+d)} & \text{when } ad < bc \text{ and } a \leq d \\ Max_{\chi^2} & = \frac{(b+d)(c+d)n}{(a+b)(a+c)} & \text{when } ad < bc \text{ and } a > d \\ Min_{\chi^2} & = \frac{n^3(\hat{a}-g(\hat{a}))^2}{(a+b)(a+c)(c+d)(b+d)} & \\ \text{where } \hat{a} & = \frac{(a+b)(a+c)}{n} & \\ \text{and } g(\hat{a}) & = \lfloor \hat{a} \rfloor & \text{when } ad < bc, \\ \text{otherwise } g(\hat{a}) & = \lceil \hat{a} \rceil & \end{array}$$

New in version 0.4.0.

Initialize Hurlbert instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in _TokenDistance for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

corr(*src*, *tar*)

Return the Hurlbert correlation of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Hurlbert correlation

Return type float

Examples

```
>>> cmp = Hurlbert()
>>> cmp.corr('cat', 'hat')
0.497416003373807
>>> cmp.corr('Niall', 'Neil')
0.32899851514665707
>>> cmp.corr('aluminum', 'Catalan')
0.10144329225459262
>>> cmp.corr('ATCG', 'TAGC')
-1.0
```

New in version 0.4.0.

sim(*src*, *tar*)

Return the Hurlbert similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Hurlbert similarity

Return type float

Examples

```
>>> cmp = Hurlbert()
>>> cmp.sim('cat', 'hat')
0.7487080016869034
>>> cmp.sim('Niall', 'Neil')
0.6644992575733285
>>> cmp.sim('aluminum', 'Catalan')
0.5507216461272963
>>> cmp.sim('ATCG', 'TAGC')
0.0
```

New in version 0.4.0.

class abydos.distance.**Jaccard**(*tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._tversky.Tversky

Jaccard similarity.

For two sets X and Y, the Jaccard similarity coefficient [Jac01][Rruvzivcka58] is

$$sim_{Jaccard}(X, Y) = \frac{|X \cap Y|}{|X \cup Y|}`.$$

This is identical to the Tanimoto similarity coefficient [Tan58] and the Tversky index [Tve77] for $\alpha = \beta = 1$.

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$sim_{Jaccard} = \frac{a}{a + b + c}$$

Notes

The multiset variant is termed Ellenberg similarity [Ell56].

New in version 0.3.6.

Initialize Jaccard instance.

Parameters

- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the soft and fuzzy variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the fuzzy variant.

New in version 0.4.0.

sim(*src*, *tar*)

Return the Jaccard similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Jaccard similarity

Return type float

Examples

```
>>> cmp = Jaccard()
>>> cmp.sim('cat', 'hat')
0.3333333333333333
>>> cmp.sim('Niall', 'Neil')
0.2222222222222222
>>> cmp.sim('aluminum', 'Catalan')
0.0625
>>> cmp.sim('ATCG', 'TAGC')
0.0
```

New in version 0.1.0.

Changed in version 0.3.6: Encapsulated in class

tanimoto_coeff(*src*, *tar*)

Return the Tanimoto distance between two strings.

Tanimoto distance [Tan58] is $-log_2 sim_{Tanimoto}(X, Y)$.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Tanimoto distance

Return type float

Examples

```
>>> cmp = Jaccard()
>>> cmp.tanimoto_coeff('cat', 'hat')
-1.5849625007211563
>>> cmp.tanimoto_coeff('Niall', 'Neil')
-2.1699250014423126
>>> cmp.tanimoto_coeff('aluminum', 'Catalan')
-4.0
>>> cmp.tanimoto_coeff('ATCG', 'TAGC')
-inf
```

New in version 0.1.0.

Changed in version 0.3.6: Encapsulated in class

abydos.distance.**dist_jaccard**(*src*, *tar*, *qval=2*)

Return the Jaccard distance between two strings.

This is a wrapper for `Jaccard.dist()`.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison
- **qval** (*int*) -- The length of each q-gram

Returns Jaccard distance

Return type float

Examples

```
>>> dist_jaccard('cat', 'hat')
0.6666666666666667
>>> dist_jaccard('Niall', 'Neil')
0.7777777777777778
>>> dist_jaccard('aluminum', 'Catalan')
0.9375
>>> dist_jaccard('ATCG', 'TAGC')
1.0
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the Jaccard.dist method instead.

abydos.distance.**sim_jaccard**(*src*, *tar*, *qval=2*)

Return the Jaccard similarity of two strings.

This is a wrapper for *Jaccard.sim()*.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison
- **qval** (*int*) -- The length of each q-gram

Returns Jaccard similarity

Return type float

Examples

```
>>> sim_jaccard('cat', 'hat')
0.3333333333333333
>>> sim_jaccard('Niall', 'Neil')
0.2222222222222222
>>> sim_jaccard('aluminum', 'Catalan')
0.0625
>>> sim_jaccard('ATCG', 'TAGC')
0.0
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the Jaccard.sim method instead.

abydos.distance.**tanimoto**(*src*, *tar*, *qval=2*)
Return the Tanimoto coefficient of two strings.

This is a wrapper for *Jaccard.tanimoto_coeff()*.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison
- **qval** (*int*) -- The length of each q-gram

Returns Tanimoto distance

Return type float

Examples

```
>>> tanimoto('cat', 'hat')
-1.5849625007211563
>>> tanimoto('Niall', 'Neil')
-2.1699250014423126
>>> tanimoto('aluminum', 'Catalan')
-4.0
>>> tanimoto('ATCG', 'TAGC')
-inf
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the Jaccard.tanimoto_coeff method instead.

class abydos.distance.**JaccardNM**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)
Bases: abydos.distance._token_distance._TokenDistance

Jaccard-NM similarity.

For two sets X and Y and a population N, Jaccard-NM similarity [NMM11] is

$$sim_{JaccardNM}(X, Y) = \frac{|X \cap Y|}{|N| + |X \cap Y| + |X \setminus Y| + |Y \setminus X|}$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$sim_{JaccardNM} = \frac{a}{2(a+b+c)+d}$$

New in version 0.4.0.

Initialize JaccardNM instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in _TokenDistance for details.

- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in `_TokenDistance` for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

sim(*src*, *tar*)

Return the Jaccard-NM similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Jaccard-NM similarity

Return type float

Examples

```
>>> cmp = JaccardNM()
>>> cmp.sim('cat', 'hat')
0.005063291139240506
>>> cmp.sim('Niall', 'Neil')
0.005044136191677175
>>> cmp.sim('aluminum', 'Catalan')
0.0024968789013732834
>>> cmp.sim('ATCG', 'TAGC')
0.0
```

New in version 0.4.0.

sim_score(*src*, *tar*)

Return the Jaccard-NM similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Jaccard-NM similarity

Return type float

Examples

```
>>> cmp = JaccardNM()
>>> cmp.sim_score('cat', 'hat')
0.002531645569620253
>>> cmp.sim_score('Niall', 'Neil')
0.0025220680958385876
>>> cmp.sim_score('aluminum', 'Catalan')
0.0012484394506866417
>>> cmp.sim_score('ATCG', 'TAGC')
0.0
```

New in version 0.4.0.

class abydos.distance.**Johnson**(*tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Johnson similarity.

For two sets X and Y, the Johnson similarity [Joh67] is

$$sim_{Johnson}(X, Y) = \frac{(|X \cap Y|}{|X|} + \frac{|Y \cap X}{|Y|}`.$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$sim_{Johnson} = \frac{a}{a+b} + \frac{a}{a+c}$$

New in version 0.4.0.

Initialize Johnson instance.

Parameters

- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the soft and fuzzy variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the fuzzy variant.

New in version 0.4.0.

sim(*src*, *tar*)

Return the normalized Johnson similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Normalized Johnson similarity

Return type float

Examples

```
>>> cmp = Johnson()
>>> cmp.sim('cat', 'hat')
0.5
>>> cmp.sim('Niall', 'Neil')
0.3666666666666667
>>> cmp.sim('aluminum', 'Catalan')
0.11805555555555555
>>> cmp.sim('ATCG', 'TAGC')
0.0
```

New in version 0.4.0.

sim_score(*src*, *tar*)

Return the Johnson similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Johnson similarity

Return type float

Examples

```
>>> cmp = Johnson()
>>> cmp.sim_score('cat', 'hat')
1.0
>>> cmp.sim_score('Niall', 'Neil')
0.7333333333333334
>>> cmp.sim_score('aluminum', 'Catalan')
0.2361111111111111
>>> cmp.sim_score('ATCG', 'TAGC')
0.0
```

New in version 0.4.0.

class abydos.distance.**KendallTau**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Kendall's Tau correlation.

For two sets X and Y and a population N, Kendall's Tau correlation [Ken38] is

$$corr_{KendallTau}(X, Y) = \frac{2 \cdot (|X \cap Y| + |(N \setminus X) \setminus Y| - |X \triangle Y|)}{|N| \cdot (|N| - 1)}$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$corr_{KendallTau} = \frac{2 \cdot (a + d - b - c)}{n \cdot (n - 1)}$$

New in version 0.4.0.

Initialize KendallTau instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in `_TokenDistance` for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in `_TokenDistance` for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

corr(*src*, *tar*)

Return the Kendall's Tau correlation of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Kendall's Tau correlation

Return type float

Examples

```
>>> cmp = KendallTau()
>>> cmp.corr('cat', 'hat')
0.0025282143508744493
>>> cmp.corr('Niall', 'Neil')
0.00250866630176975
>>> cmp.corr('aluminum', 'Catalan')
0.0024535291823735866
>>> cmp.corr('ATCG', 'TAGC')
0.0024891182526650506
```

Notes

This correlation is not necessarily bounded to [-1.0, 1.0], but will typically be within these bounds for real data.

New in version 0.4.0.

sim(*src*, *tar*)
Return the Kendall's Tau similarity of two strings.

The Tau correlation is first clamped to the range [-1.0, 1.0] before being converted to a similarity value to ensure that the similarity is in the range [0.0, 1.0].

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Kendall's Tau similarity

Return type float

Examples

```
>>> cmp = KendallTau()
>>> cmp.sim('cat', 'hat')
0.5012641071754372
>>> cmp.sim('Niall', 'Neil')
0.5012543331508849
>>> cmp.sim('aluminum', 'Catalan')
0.5012267645911868
>>> cmp.sim('ATCG', 'TAGC')
0.5012445591263325
```

New in version 0.4.0.

class abydos.distance.**KentFosterI**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)
Bases: abydos.distance._token_distance._TokenDistance

Kent & Foster I similarity.

For two sets X and Y and a population N, Kent & Foster I similarity [KF77], K_{occ}, is

$$sim_{KentFosterI}(X, Y) = \frac{|X \cap Y| - \frac{|X| \cdot |Y|}{|X \cup Y|}}{|X \cap Y| - \frac{|X| \cdot |Y|}{|X \cup Y|} + |X \setminus Y| + |Y \setminus X|}$$

Kent & Foster derived this from Cohen's κ by "subtracting appropriate chance agreement correction figures from the numerators and denominators" to arrive at an occurrence reliability measure.

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$sim_{KentFosterI} = \frac{a - \frac{(a+b)(a+c)}{a+b+c}}{a - \frac{(a+b)(a+c)}{a+b+c} + b + c}$$

New in version 0.4.0.

Initialize KentFosterI instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in `_TokenDistance` for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in `_TokenDistance` for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

sim(*src*, *tar*)

Return the normalized Kent & Foster I similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Normalized Kent & Foster I similarity

Return type float

Examples

```
>>> cmp = KentFosterI()
>>> cmp.sim('cat', 'hat')
0.8
>>> cmp.sim('Niall', 'Neil')
0.7647058823529411
>>> cmp.sim('aluminum', 'Catalan')
0.6956521739130435
>>> cmp.sim('ATCG', 'TAGC')
0.6666666666666667
```

New in version 0.4.0.

sim_score(*src*, *tar*)

Return the Kent & Foster I similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison

- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Kent & Foster I similarity

Return type float

Examples

```
>>> cmp = KentFosterI()
>>> cmp.sim_score('cat', 'hat')
-0.19999999999999996
>>> cmp.sim_score('Niall', 'Neil')
-0.23529411764705888
>>> cmp.sim_score('aluminum', 'Catalan')
-0.30434782608695654
>>> cmp.sim_score('ATCG', 'TAGC')
-0.3333333333333333
```

New in version 0.4.0.

class abydos.distance.**KentFosterII**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Kent & Foster II similarity.

For two sets X and Y and a population N, Kent & Foster II similarity [KF77], K_{nonocc}, is

$$sim_{KentFosterII}(X, Y) = \frac{|(N \setminus X) \setminus Y| - \frac{|X \setminus Y| \cdot |Y \setminus X|}{|N \setminus (X \cap Y)|}}{|(N \setminus X) \setminus Y| - \frac{|X \setminus Y| \cdot |Y \setminus X|}{|N \setminus (X \cap Y)|} + |X \setminus Y| + |Y \setminus X|}$$

Kent & Foster derived this from Cohen's κ by "subtracting appropriate chance agreement correction figures from the numerators and denominators" to arrive at an non-occurrence reliability measure.

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$sim_{KentFosterII} = \frac{d - \frac{(b+d)(c+d)}{b+c+d}}{d - \frac{(b+d)(c+d)}{b+c+d} + b + c}$$

New in version 0.4.0.

Initialize KentFosterII instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in _TokenDistance for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

sim(*src*, *tar*)

Return the normalized Kent & Foster II similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Normalized Kent & Foster II similarity

Return type float

Examples

```
>>> cmp = KentFosterII()
>>> cmp.sim('cat', 'hat')
0.998719590268876
>>> cmp.sim('Niall', 'Neil')
0.9978030025631628
>>> cmp.sim('aluminum', 'Catalan')
0.9952153110047858
>>> cmp.sim('ATCG', 'TAGC')
0.9968010236724241
```

New in version 0.4.0.

sim_score(*src*, *tar*)

Return the Kent & Foster II similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Kent & Foster II similarity

Return type float

Examples

```
>>> cmp = KentFosterII()
>>> cmp.sim_score('cat', 'hat')
-0.0012804097311239404
>>> cmp.sim_score('Niall', 'Neil')
-0.002196997436837158
>>> cmp.sim_score('aluminum', 'Catalan')
-0.004784688995214218
>>> cmp.sim_score('ATCG', 'TAGC')
-0.0031989763275758767
```

New in version 0.4.0.

class abydos.distance.**KoppenI**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, *normalizer='proportional'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Köppen I correlation.

For two sets X and Y and an alphabet N, provided that $|X| = |Y|$, Köppen I correlation [Koppen70][GK59] is

$$corr_{KoppenI}(X, Y) = \frac{|X| \cdot |N \setminus X| - |X \setminus Y|}{|X| \cdot |N \setminus X|}$$

To support cases where $|X| \neq |Y|$, this class implements a slight variation, while still providing the expected results when $|X| = |Y|$:

$$corr_{KoppenI}(X, Y) = \frac{\frac{|X|+|Y|}{2} \cdot \frac{|N\setminus X|+|N\setminus Y|}{2} - \frac{|X \triangle Y|}{2}}{\frac{|X|+|Y|}{2} \cdot \frac{|N\setminus X|+|N\setminus Y|}{2}}$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$sim_{KoppenI} = \frac{\frac{2a+b+c}{2} \cdot \frac{2d+b+c}{2} - \frac{b+c}{2}}{\frac{2a+b+c}{2} \cdot \frac{2d+b+c}{2}}$$

Notes

In the usual case all of the above values should be proportional to the total number of samples n. I.e., a, b, c, d, & n should all be divided by n prior to calculating the coefficient. This class's default normalizer is, accordingly, 'proportional'.

New in version 0.4.0.

Initialize KoppenI instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in _TokenDistance for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package

- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in `_TokenDistance` for details.
- **normalizer** (*str*) -- Specifies the normalization type. See normalizer description in `_TokenDistance` for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

corr(*src*, *tar*)

Return the Köppen I correlation of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Köppen I correlation

Return type float

Examples

```
>>> cmp = KoppenI()
>>> cmp.corr('cat', 'hat')
0.49615384615384617
>>> cmp.corr('Niall', 'Neil')
0.3575056927658083
>>> cmp.corr('aluminum', 'Catalan')
0.1068520131813188
>>> cmp.corr('ATCG', 'TAGC')
-0.006418485237483896
```

New in version 0.4.0.

sim(*src*, *tar*)

Return the Köppen I similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Köppen I similarity

Return type float

Examples

```
>>> cmp = KoppenI()
>>> cmp.sim('cat', 'hat')
0.7480769230769231
>>> cmp.sim('Niall', 'Neil')
0.6787528463829041
>>> cmp.sim('aluminum', 'Catalan')
0.5534260065906594
>>> cmp.sim('ATCG', 'TAGC')
0.49679075738125805
```

New in version 0.4.0.

class abydos.distance.**KoppenII**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Köppen II similarity.

For two sets X and Y, Köppen II similarity [Koppen70][GK59] is

$$sim_{KoppenII}(X, Y) = |X \cap Y| + \frac{|X \setminus Y| + |Y \setminus X|}{2}$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$sim_{KoppenII} = a + \frac{b + c}{2}$$

New in version 0.4.0.

Initialize KoppenII instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in _TokenDistance for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the soft and fuzzy variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the fuzzy variant.

New in version 0.4.0.

sim(*src*, *tar*)
Return the normalized Köppen II similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Normalized Köppen II similarity

Return type float

Examples

```
>>> cmp = KoppenII()
>>> cmp.sim('cat', 'hat')
0.6666666666666666
>>> cmp.sim('Niall', 'Neil')
0.6111111111111112
>>> cmp.sim('aluminum', 'Catalan')
0.53125
>>> cmp.sim('ATCG', 'TAGC')
0.5
```

New in version 0.4.0.

sim_score(*src*, *tar*)
Return the Köppen II similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Köppen II similarity

Return type float

Examples

```
>>> cmp = KoppenII()
>>> cmp.sim_score('cat', 'hat')
4.0
>>> cmp.sim_score('Niall', 'Neil')
5.5
>>> cmp.sim_score('aluminum', 'Catalan')
8.5
>>> cmp.sim_score('ATCG', 'TAGC')
5.0
```

New in version 0.4.0.

class abydos.distance.**KuderRichardson**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)
Bases: abydos.distance._token_distance._TokenDistance

Kuder & Richardson correlation.

For two sets X and Y and a population N, Kuder & Richardson similarity [KR37][Cro51] is

$$corr_{KuderRichardson}(X,Y) = \frac{4(|X \cap Y| \cdot |(N \setminus X) \setminus Y| - |X \setminus Y| \cdot |Y \setminus X|)}{|X| \cdot |N \setminus X| + |Y| \cdot |N \setminus Y| + 2(|X \cap Y| \cdot |(N \setminus X) \setminus Y| - |X \setminus Y| \cdot |Y \setminus X|)}$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$corr_{KuderRichardson} = \frac{4(ad - bc)}{(a+b)(c+d) + (a+c)(b+d) + 2(ad - bc)}$$

New in version 0.4.0.

Initialize KuderRichardson instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in _TokenDistance for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the soft and fuzzy variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the fuzzy variant.

New in version 0.4.0.

corr(*src*, *tar*)

Return the Kuder & Richardson correlation of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Kuder & Richardson correlation

Return type float

Examples

```
>>> cmp = KuderRichardson()
>>> cmp.corr('cat', 'hat')
0.6643835616438356
>>> cmp.corr('Niall', 'Neil')
0.5285677463699631
>>> cmp.corr('aluminum', 'Catalan')
0.19499521400246136
>>> cmp.corr('ATCG', 'TAGC')
-0.012919896640826873
```

New in version 0.4.0.

sim(*src*, *tar*)

Return the Kuder & Richardson similarity of two strings.

Since Kuder & Richardson correlation is unbounded in the negative, this measure is first clamped to [-1.0, 1.0].

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Kuder & Richardson similarity

Return type float

Examples

```
>>> cmp = KuderRichardson()
>>> cmp.sim('cat', 'hat')
0.8321917808219178
>>> cmp.sim('Niall', 'Neil')
0.7642838731849815
>>> cmp.sim('aluminum', 'Catalan')
0.5974976070012307
>>> cmp.sim('ATCG', 'TAGC')
0.4935400516795866
```

New in version 0.4.0.

class abydos.distance.**KuhnsI**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Kuhns I correlation.

For two sets X and Y and a population N, Kuhns I correlation [Kuh64], the excess of separation over its independence value (S), is

$$corr_{KuhnsI}(X, Y) = \frac{2\delta(X, Y)}{|N|}$$

where

$$\delta(X, Y) = |X \cap Y| - \frac{|X| \cdot |Y|}{|N|}$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$corr_{KuhnsI} = \frac{2\delta(a+b, a+c)}{n}$$

where

$$\delta(a+b, a+c) = a - \frac{(a+b)(a+c)}{n}$$

New in version 0.4.0.

Initialize KuhnsI instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in `_TokenDistance` for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in `_TokenDistance` for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

corr(*src*, *tar*)

Return the Kuhns I correlation of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Kuhns I correlation

Return type float

Examples

```
>>> cmp = KuhnsI()
>>> cmp.corr('cat', 'hat')
0.005049979175343606
>>> cmp.corr('Niall', 'Neil')
0.005004425239483548
>>> cmp.corr('aluminum', 'Catalan')
0.0023140898210880765
>>> cmp.corr('ATCG', 'TAGC')
-8.134631403581842e-05
```

New in version 0.4.0.

sim(*src*, *tar*)

Return the Kuhns I similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Kuhns I similarity

Return type float

Examples

```
>>> cmp = KuhnsI()
>>> cmp.sim('cat', 'hat')
0.5050499791753436
>>> cmp.sim('Niall', 'Neil')
0.5050044252394835
>>> cmp.sim('aluminum', 'Catalan')
0.502314089821088
>>> cmp.sim('ATCG', 'TAGC')
0.49991865368596416
```

New in version 0.4.0.

class abydos.distance.**KuhnsII**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Kuhns II correlation.

For two sets X and Y and a population N, Kuhns II correlation [Kuh64], the excess of rectangular distance over its independence value (R), is

$$corr_{KuhnsII}(X, Y) = \frac{\delta(X, Y)}{max(|X|, |Y|)}$$

where

$$\delta(X, Y) = |X \cap Y| - \frac{|X| \cdot |Y|}{|N|}$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$corr_{KuhnsII} = \frac{\delta(a+b, a+c)}{max(a+b, a+c)}$$

where

$$\delta(a+b, a+c) = a - \frac{(a+b)(a+c)}{n}$$

New in version 0.4.0.

Initialize KuhnsII instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in _TokenDistance for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the soft and fuzzy variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the fuzzy variant.

New in version 0.4.0.

corr(*src*, *tar*)

Return the Kuhns II correlation of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Kuhns II correlation

Return type float

Examples

```
>>> cmp = KuhnsII()
>>> cmp.corr('cat', 'hat')
0.49489795918367346
>>> cmp.corr('Niall', 'Neil')
0.32695578231292516
>>> cmp.corr('aluminum', 'Catalan')
0.10092002830856334
>>> cmp.corr('ATCG', 'TAGC')
-0.006377551020408163
```

New in version 0.4.0.

sim(*src*, *tar*)

Return the Kuhns II similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Kuhns II similarity

Return type float

Examples

```
>>> cmp = KuhnsII()
>>> cmp.sim('cat', 'hat')
0.663265306122449
>>> cmp.sim('Niall', 'Neil')
0.5513038548752834
>>> cmp.sim('aluminum', 'Catalan')
0.40061335220570893
>>> cmp.sim('ATCG', 'TAGC')
0.32908163265306123
```

New in version 0.4.0.

class abydos.distance.**KuhnsIII**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Kuhns III correlation.

For two sets X and Y and a population N, Kuhns III correlation [Kuh64], the excess of proportion of overlap over its independence value (P), is

$$corr_{KuhnsIII}(X,Y) = \frac{\delta(X,Y)}{\left(1 - \frac{|X \cap Y|}{|X|+|Y|}\right)\left(|X| + |Y| - \frac{|X| \cdot |Y|}{|N|}\right)}$$

where

$$\delta(X,Y) = |X \cap Y| - \frac{|X| \cdot |Y|}{|N|}$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$corr_{KuhnsIII} = \frac{\delta(a+b, a+c)}{\left(1 - \frac{a}{2a+b+c}\right)\left(2a+b+c - \frac{(a+b)(a+c)}{n}\right)}$$

where

$$\delta(a+b, a+c) = a - \frac{(a+b)(a+c)}{n}$$

Notes

The coefficient presented in [Eid14][Mor12] as Kuhns' "Proportion of overlap above independence" is a significantly different coefficient, not evidenced in [Kuh64].

New in version 0.4.0.

Initialize KuhnsIII instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in `_TokenDistance` for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in `_TokenDistance` for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

corr(*src*, *tar*)

Return the Kuhns III correlation of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Kuhns III correlation

Return type float

Examples

```
>>> cmp = KuhnsIII()
>>> cmp.corr('cat', 'hat')
0.3307757885763001
>>> cmp.corr('Niall', 'Neil')
0.21873141468207793
>>> cmp.corr('aluminum', 'Catalan')
0.05707545392902886
>>> cmp.corr('ATCG', 'TAGC')
-0.003198976327575176
```

New in version 0.4.0.

sim(*src*, *tar*)

Return the Kuhns III similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Kuhns III similarity

Return type float

Examples

```
>>> cmp = KuhnsIII()
>>> cmp.sim('cat', 'hat')
0.498081841432225
>>> cmp.sim('Niall', 'Neil')
0.41404856101155846
>>> cmp.sim('aluminum', 'Catalan')
0.29280659044677165
>>> cmp.sim('ATCG', 'TAGC')
0.24760076775431863
```

New in version 0.4.0.

class abydos.distance.**KuhnsIV**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Kuhns IV correlation.

For two sets X and Y and a population N, Kuhns IV correlation [Kuh64], the excess of conditional probabilities over its independence value (W), is

$$corr_{KuhnsIV}(X, Y) = \frac{\delta(X, Y)}{min(|X|, |Y|)}$$

where

$$\delta(X, Y) = |X \cap Y| - \frac{|X| \cdot |Y|}{|N|}$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$corr_{KuhnsIV} = \frac{\delta(a+b, a+c)}{min(a+b, a+c)}$$

where

$$\delta(a+b, a+c) = a - \frac{(a+b)(a+c)}{n}$$

New in version 0.4.0.

Initialize KuhnsIV instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in _TokenDistance for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the soft and fuzzy variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the fuzzy variant.

New in version 0.4.0.

corr(*src*, *tar*)

Return the Kuhns IV correlation of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Kuhns IV correlation

Return type float

Examples

```
>>> cmp = KuhnsIV()
>>> cmp.corr('cat', 'hat')
0.49489795918367346
>>> cmp.corr('Niall', 'Neil')
0.3923469387755102
>>> cmp.corr('aluminum', 'Catalan')
0.11353503184713376
>>> cmp.corr('ATCG', 'TAGC')
-0.006377551020408163
```

New in version 0.4.0.

sim(*src*, *tar*)

Return the Kuhns IV similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Kuhns IV similarity

Return type float

Examples

```
>>> cmp = KuhnsIV()
>>> cmp.sim('cat', 'hat')
0.7474489795918368
>>> cmp.sim('Niall', 'Neil')
0.696173469387755
>>> cmp.sim('aluminum', 'Catalan')
0.5567675159235669
>>> cmp.sim('ATCG', 'TAGC')
0.4968112244897959
```

New in version 0.4.0.

class abydos.distance.**KuhnsV**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Kuhns V correlation.

For two sets X and Y and a population N, Kuhns V correlation [Kuh64], the excess of probability differences U over its independence value (U), is

$$corr_{KuhnsV}(X,Y) = \frac{\delta(X,Y)}{max\big(|X| \cdot (1 - \frac{|X|}{|N|}), |Y| \cdot (1 - \frac{|Y|}{|N|})\big)}$$

where

$$\delta(X,Y) = |X \cap Y| - \frac{|X| \cdot |Y|}{|N|}$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$corr_{KuhnsV} = \frac{\delta(a+b, a+c)}{max\big((a+b)(1-\frac{a+b}{n}), (a+c)(1-\frac{a+c}{n})\big)}$$

where

$$\delta(a+b, a+c) = a - \frac{(a+b)(a+c)}{n}$$

New in version 0.4.0.

Initialize KuhnsV instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in `_TokenDistance` for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in `_TokenDistance` for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

corr(*src*, *tar*)

Return the Kuhns V correlation of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Kuhns V correlation

Return type float

Examples

```
>>> cmp = KuhnsV()
>>> cmp.corr('cat', 'hat')
0.497435897435897
>>> cmp.corr('Niall', 'Neil')
0.329477292202228
>>> cmp.corr('aluminum', 'Catalan')
0.10209049255441
>>> cmp.corr('ATCG', 'TAGC')
-0.006418485237484
```

New in version 0.4.0.

sim(*src*, *tar*)

Return the Kuhns V similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Kuhns V similarity

Return type float

Examples

```
>>> cmp = KuhnsV()
>>> cmp.sim('cat', 'hat')
0.7487179487179485
>>> cmp.sim('Niall', 'Neil')
0.664738646101114
>>> cmp.sim('aluminum', 'Catalan')
0.551045246277205
>>> cmp.sim('ATCG', 'TAGC')
0.496790757381258
```

New in version 0.4.0.

class abydos.distance.**KuhnsVI**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Kuhns VI correlation.

For two sets X and Y and a population N, Kuhns VI correlation [Kuh64], the excess of probability differences V over its independence value (V), is

$$corr_{KuhnsVI}(X,Y) = \frac{\delta(X,Y)}{min\big(|X| \cdot (1 - \frac{|X|}{|N|}), |Y|(1 - \frac{|Y|}{|N|})\big)}$$

where

$$\delta(X,Y) = |X \cap Y| - \frac{|X| \cdot |Y|}{|N|}$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$corr_{KuhnsVI} = \frac{\delta(a+b, a+c)}{min\big((a+b)(1-\frac{a+b}{n}), (a+c)(1-\frac{a+c}{n})\big)}$$

where

$$\delta(a+b, a+c) = a - \frac{(a+b)(a+c)}{n}$$

New in version 0.4.0.

Initialize KuhnsVI instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in _TokenDistance for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the soft and fuzzy variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the fuzzy variant.

New in version 0.4.0.

corr(*src*, *tar*)

Return the Kuhns VI correlation of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Kuhns VI correlation

Return type float

Examples

```
>>> cmp = KuhnsVI()
>>> cmp.corr('cat', 'hat')
0.497435897435897
>>> cmp.corr('Niall', 'Neil')
0.394865211810013
>>> cmp.corr('aluminum', 'Catalan')
0.11470398970399
>>> cmp.corr('ATCG', 'TAGC')
-0.006418485237484
```

New in version 0.4.0.

sim(*src*, *tar*)

Return the Kuhns VI similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Kuhns VI similarity

Return type float

Examples

```
>>> cmp = KuhnsVI()
>>> cmp.sim('cat', 'hat')
0.7487179487179485
>>> cmp.sim('Niall', 'Neil')
0.6974326059050064
>>> cmp.sim('aluminum', 'Catalan')
0.557351994851995
>>> cmp.sim('ATCG', 'TAGC')
0.496790757381258
```

New in version 0.4.0.

class abydos.distance.**KuhnsVII**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Kuhns VII correlation.

For two sets X and Y and a population N, Kuhns VII correlation [Kuh64], the excess of angle between vector over its independence value (G), is

$$corr_{KuhnsVII}(X, Y) = \frac{\delta(X, Y)}{\sqrt{|X| \cdot |Y|}}$$

where

$$\delta(X, Y) = |X \cap Y| - \frac{|X| \cdot |Y|}{|N|}$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$corr_{KuhnsVII} = \frac{\delta(a+b, a+c)}{\sqrt{(a+b)(a+c)}}$$

where

$$\delta(a+b, a+c) = a - \frac{(a+b)(a+c)}{n}$$

New in version 0.4.0.

Initialize KuhnsVII instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in _TokenDistance for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the soft and fuzzy variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the fuzzy variant.

New in version 0.4.0.

corr(*src*, *tar*)

Return the Kuhns VII correlation of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Kuhns VII correlation

Return type float

Examples

```
>>> cmp = KuhnsVII()
>>> cmp.corr('cat', 'hat')
0.49489795918367346
>>> cmp.corr('Niall', 'Neil')
0.3581621145590755
>>> cmp.corr('aluminum', 'Catalan')
0.10704185456178524
>>> cmp.corr('ATCG', 'TAGC')
-0.006377551020408163
```

New in version 0.4.0.

sim(*src*, *tar*)

Return the Kuhns VII similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Kuhns VII similarity

Return type float

Examples

```
>>> cmp = KuhnsVII()
>>> cmp.sim('cat', 'hat')
0.663265306122449
>>> cmp.sim('Niall', 'Neil')
0.572108076372717
>>> cmp.sim('aluminum', 'Catalan')
0.40469456970785683
>>> cmp.sim('ATCG', 'TAGC')
0.32908163265306123
```

New in version 0.4.0.

class abydos.distance.**KuhnsVIII**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Kuhns VIII correlation.

For two sets X and Y and a population N, Kuhns VIII correlation [Kuh64], the excess of coefficient by the arithmetic mean over its independence value (E), is

$$corr_{KuhnsVIII}(X, Y) = \frac{\delta(X, Y)}{|X \cap Y| + \frac{1}{2} \cdot |X \triangle Y|}$$

where

$$\delta(X, Y) = |X \cap Y| - \frac{|X| \cdot |Y|}{|N|}$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$corr_{KuhnsVIII} = \frac{\delta(a+b, a+c)}{a + \frac{1}{2}(b+c)}$$

where

$$\delta(a+b, a+c) = a - \frac{(a+b)(a+c)}{n}$$

Notes

The coefficient presented in [Eid14][Mor12] as Kuhns' "Coefficient of arithmetic means" is a significantly different coefficient, not evidenced in [Kuh64].

New in version 0.4.0.

Initialize KuhnsVIII instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in `_TokenDistance` for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in `_TokenDistance` for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

corr(*src*, *tar*)

Return the Kuhns VIII correlation of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Kuhns VIII correlation

Return type float

Examples

```
>>> cmp = KuhnsVIII()
>>> cmp.corr('cat', 'hat')
0.49489795918367346
>>> cmp.corr('Niall', 'Neil')
0.35667903525046385
>>> cmp.corr('aluminum', 'Catalan')
0.10685650056200824
>>> cmp.corr('ATCG', 'TAGC')
-0.006377551020408163
```

New in version 0.4.0.

sim(*src*, *tar*)

Return the Kuhns VIII similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Kuhns VIII similarity

Return type float

Examples

```
>>> cmp = KuhnsVIII()
>>> cmp.sim('cat', 'hat')
0.663265306122449
>>> cmp.sim('Niall', 'Neil')
0.5711193568336426
>>> cmp.sim('aluminum', 'Catalan')
0.40457100037467214
>>> cmp.sim('ATCG', 'TAGC')
0.32908163265306123
```

New in version 0.4.0.

class abydos.distance.**KuhnsIX**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Kuhns IX correlation.

For two sets X and Y and a population N, Kuhns IX correlation [Kuh64], the excess of coefficient of linear correlation over its independence value (L), is

$$corr_{KuhnsIX}(X, Y) = \frac{\delta(X, Y)}{\sqrt{|X| \cdot |Y| \cdot (1 - \frac{|X|}{|N|}) \cdot (1 - \frac{|Y|}{|N|})}}$$

where

$$\delta(X, Y) = |X \cap Y| - \frac{|X| \cdot |Y|}{|N|}$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$corr_{KuhnsIX} = \frac{\delta(a+b, a+c)}{\sqrt{(a+b)(a+c)(1-\frac{a+b}{n})(1-\frac{a+c}{n})}}$$

where

$$\delta(a+b, a+c) = a - \frac{(a+b)(a+c)}{n}$$

New in version 0.4.0.

Initialize KuhnsIX instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in `_TokenDistance` for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in `_TokenDistance` for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

corr(*src*, *tar*)

Return the Kuhns IX correlation of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Kuhns IX correlation

Return type float

Examples

```
>>> cmp = KuhnsIX()
>>> cmp.corr('cat', 'hat')
0.49743589743589745
>>> cmp.corr('Niall', 'Neil')
0.36069255713421955
>>> cmp.corr('aluminum', 'Catalan')
0.10821361655002706
>>> cmp.corr('ATCG', 'TAGC')
-0.006418485237483954
```

New in version 0.4.0.

sim(*src*, *tar*)

Return the Kuhns IX similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Kuhns IX similarity

Return type float

Examples

```
>>> cmp = KuhnsIX()
>>> cmp.sim('cat', 'hat')
0.7487179487179487
>>> cmp.sim('Niall', 'Neil')
0.6803462785671097
>>> cmp.sim('aluminum', 'Catalan')
0.5541068082750136
>>> cmp.sim('ATCG', 'TAGC')
0.496790757381258
```

New in version 0.4.0.

class abydos.distance.**KuhnsX**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Kuhns X correlation.

For two sets X and Y and a population N, Kuhns X correlation [Kuh64], the excess of Yule's Q over its independence value (Q), is

$$corr_{KuhnsX}(X,Y) = \frac{|N| \cdot \delta(X,Y)}{|X \cap Y| \cdot |(N \setminus X) \setminus Y| + |X \setminus Y| \cdot |Y \setminus X|}$$

where

$$\delta(X,Y) = |X \cap Y| - \frac{|X| \cdot |Y|}{|N|}$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$corr_{KuhnsX} = \frac{n \cdot \delta(a+b, a+c)}{ad+bc}$$

where

$$\delta(a+b, a+c) = a - \frac{(a+b)(a+c)}{n}$$

New in version 0.4.0.

Initialize KuhnsX instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in `_TokenDistance` for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in `_TokenDistance` for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

corr(*src*, *tar*)

Return the Kuhns X correlation of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Kuhns X correlation

Return type float

Examples

```
>>> cmp = KuhnsX()
>>> cmp.corr('cat', 'hat')
0.994871794871795
>>> cmp.corr('Niall', 'Neil')
0.984635083226633
>>> cmp.corr('aluminum', 'Catalan')
0.864242424242424
>>> cmp.corr('ATCG', 'TAGC')
-1.0
```

New in version 0.4.0.

sim(*src*, *tar*)

Return the Kuhns X similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Kuhns X similarity

Return type float

Examples

```
>>> cmp = KuhnsX()
>>> cmp.sim('cat', 'hat')
0.9974358974358974
>>> cmp.sim('Niall', 'Neil')
0.9923175416133165
>>> cmp.sim('aluminum', 'Catalan')
0.932121212121212
>>> cmp.sim('ATCG', 'TAGC')
0.0
```

New in version 0.4.0.

class abydos.distance.**KuhnsXI**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Kuhns XI correlation.

For two sets X and Y and a population N, Kuhns XI correlation [Kuh64], the excess of Yule's Y over its independence value (Y), is

$$corr_{KuhnsXI}(X, Y) = \frac{|N| \cdot \delta(X, Y)}{(\sqrt{|X \cap Y| \cdot |(N \setminus X) \setminus Y|} + \sqrt{|X \setminus Y| \cdot |Y \setminus X|})^2}$$

where

$$\delta(X, Y) = |X \cap Y| - \frac{|X| \cdot |Y|}{|N|}$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$corr_{KuhnsXI} = \frac{n \cdot \delta(a+b, a+c)}{(\sqrt{ad} + \sqrt{bc})^2}$$

where

$$\delta(a+b, a+c) = a - \frac{(a+b)(a+c)}{n}$$

New in version 0.4.0.

Initialize KuhnsXI instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in _TokenDistance for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the soft and fuzzy variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the fuzzy variant.

New in version 0.4.0.

corr(*src*, *tar*)

Return the Kuhns XI correlation of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Kuhns XI correlation

Return type float

Examples

```
>>> cmp = KuhnsXI()
>>> cmp.corr('cat', 'hat')
0.9034892632818761
>>> cmp.corr('Niall', 'Neil')
0.8382551144735259
>>> cmp.corr('aluminum', 'Catalan')
0.5749826820237787
>>> cmp.corr('ATCG', 'TAGC')
-1.0
```

New in version 0.4.0.

sim(*src*, *tar*)
Return the Kuhns XI similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Kuhns XI similarity

Return type float

Examples

```
>>> cmp = KuhnsXI()
>>> cmp.sim('cat', 'hat')
0.951744631640938
>>> cmp.sim('Niall', 'Neil')
0.919127557236763
>>> cmp.sim('aluminum', 'Catalan')
0.7874913410118893
>>> cmp.sim('ATCG', 'TAGC')
0.0
```

New in version 0.4.0.

class abydos.distance.**KuhnsXII**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)
Bases: abydos.distance._token_distance._TokenDistance

Kuhns XII similarity.

For two sets X and Y and a population N, Kuhns XII similarity [Kuh64], the excess of index of independence over its independence value (I), is

$$sim_{KuhnsXII}(X, Y) = \frac{|N| \cdot \delta(X, Y)}{|X| \cdot |Y|}$$

where

$$\delta(X, Y) = |X \cap Y| - \frac{|X| \cdot |Y|}{|N|}$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$sim_{KuhnsXII} = \frac{n \cdot \delta(a+b, a+c)}{(a+b)(a+c)}$$

where

$$\delta(a+b, a+c) = a - \frac{(a+b)(a+c)}{n}$$

New in version 0.4.0.

Initialize KuhnsXII instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in _TokenDistance for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the soft and fuzzy variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the fuzzy variant.

New in version 0.4.0.

sim(*src*, *tar*)

Return the normalized Kuhns XII similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Normalized Kuhns XII similarity

Return type float

Examples

```
>>> cmp = KuhnsXII()
>>> cmp.sim('cat', 'hat')
0.2493573264781491
>>> cmp.sim('Niall', 'Neil')
0.1323010752688172
>>> cmp.sim('aluminum', 'Catalan')
0.012877474353417137
>>> cmp.sim('ATCG', 'TAGC')
0.0
```

New in version 0.4.0.

sim_score(*src*, *tar*)

Return the Kuhns XII similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Kuhns XII similarity

Return type float

Examples

```
>>> cmp = KuhnsXII()
>>> cmp.sim_score('cat', 'hat')
97.0
>>> cmp.sim_score('Niall', 'Neil')
51.266666666666666
>>> cmp.sim_score('aluminum', 'Catalan')
9.902777777777779
>>> cmp.sim_score('ATCG', 'TAGC')
-1.0
```

New in version 0.4.0.

class abydos.distance.**KulczynskiI**(*tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Kulczynski I similarity.

For two sets X and Y, Kulczynski I similarity [Kulczynski27] is

$$sim_{KulczynskiI}(X, Y) = \frac{|X \cap Y|}{|X \setminus Y| + |Y \setminus X|}$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$sim_{KulczynskiI} = \frac{a}{b + c}$$

New in version 0.4.0.

Initialize KulczynskiI instance.

Parameters

- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

dist(**args*, ***kwargs*)
Raise exception when called.

Parameters

- ***args** -- Variable length argument list
- ****kwargs** -- Arbitrary keyword arguments

Raises **NotImplementedError** -- Method disabled for Kulczynski I similarity.

New in version 0.3.6.

sim(**args*, ***kwargs*)
Raise exception when called.

Parameters

- ***args** -- Variable length argument list
- ****kwargs** -- Arbitrary keyword arguments

Raises **NotImplementedError** -- Method disabled for Kulczynski I similarity.

New in version 0.3.6.

sim_score(*src*, *tar*)
Return the Kulczynski I similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Kulczynski I similarity

Return type float

Examples

```
>>> cmp = KulczynskiI()
>>> cmp.sim_score('cat', 'hat')
0.5
>>> cmp.sim_score('Niall', 'Neil')
0.2857142857142857
>>> cmp.sim_score('aluminum', 'Catalan')
0.06666666666666667
>>> cmp.sim_score('ATCG', 'TAGC')
0.0
```

New in version 0.4.0.

class abydos.distance.**KulczynskiII**(*tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Kulczynski II similarity.

For two sets X and Y, Kulczynski II similarity [Kulczynski27] or Driver & Kroeber similarity [DK32] is

$$sim_{KulczynskiII}(X, Y) = \frac{1}{2}\left(\frac{|X \cap Y|}{|X|} + \frac{|X \cap Y|}{|Y|}\right)$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$sim_{KulczynskiII} = \frac{1}{2}\left(\frac{a}{a+b} + \frac{a}{a+c}\right)$$

New in version 0.4.0.

Initialize KulczynskiII instance.

Parameters

- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the soft and fuzzy variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the fuzzy variant.

New in version 0.4.0.

sim(*src*, *tar*)

Return the Kulczynski II similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Kulczynski II similarity

Return type float

Examples

```
>>> cmp = KulczynskiII()
>>> cmp.sim('cat', 'hat')
0.5
>>> cmp.sim('Niall', 'Neil')
0.3666666666666667
>>> cmp.sim('aluminum', 'Catalan')
0.11805555555555555
>>> cmp.sim('ATCG', 'TAGC')
0.0
```

New in version 0.4.0.

class abydos.distance.**Lorentzian**(*tokenizer=None*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Lorentzian distance.

For two multisets X and Y drawn from an alphabet S, Lorentzian distance is

$$dist_{Lorentzian}(X, Y) = \sum_{i \in S} log(1 + |A_i - B_i|)$$

Notes

No primary source for this measure could be located, but it is included in surveys and catalogues, such as [DD16] and [Cha08].

New in version 0.4.0.

Initialize Lorentzian instance.

Parameters

- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.

New in version 0.4.0.

dist(*src*, *tar*)

Return the normalized Lorentzian distance of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Normalized Lorentzian distance

Return type float

Examples

```
>>> cmp = Lorentzian()
>>> cmp.dist('cat', 'hat')
0.6666666666666667
>>> cmp.dist('Niall', 'Neil')
0.7777777777777778
>>> cmp.dist('aluminum', 'Catalan')
0.9358355851062377
>>> cmp.dist('ATCG', 'TAGC')
1.0
```

New in version 0.4.0.

dist_abs(*src*, *tar*)

Return the Lorentzian distance of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Lorentzian distance

Return type float

Examples

```
>>> cmp = Lorentzian()
>>> cmp.dist_abs('cat', 'hat')
2.772588722239781
>>> cmp.dist_abs('Niall', 'Neil')
4.852030263919617
>>> cmp.dist_abs('aluminum', 'Catalan')
10.1095256359474
>>> cmp.dist_abs('ATCG', 'TAGC')
6.931471805599453
```

New in version 0.4.0.

class abydos.distance.**Maarel**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Maarel correlation.

For two sets X and Y and a population N, Maarel correlation [vandMaarel69] is

$$corr_{Maarel}(X, Y) = \frac{2|X \cap Y| - |X \setminus Y| - |Y \setminus X|}{|X| + |Y|}$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$corr_{Maarel} = \frac{2a - b - c}{2a + b + c}$$

New in version 0.4.0.

Initialize Maarel instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in `_TokenDistance` for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in `_TokenDistance` for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

corr(*src*, *tar*)

Return the Maarel correlation of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Maarel correlation

Return type float

Examples

```
>>> cmp = Maarel()
>>> cmp.corr('cat', 'hat')
0.0
>>> cmp.corr('Niall', 'Neil')
-0.2727272727272727
>>> cmp.corr('aluminum', 'Catalan')
-0.7647058823529411
>>> cmp.corr('ATCG', 'TAGC')
-1.0
```

New in version 0.4.0.

sim(*src*, *tar*)
Return the Maarel similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Maarel similarity

Return type float

Examples

```
>>> cmp = Maarel()
>>> cmp.sim('cat', 'hat')
0.5
>>> cmp.sim('Niall', 'Neil')
0.36363636363636365
>>> cmp.sim('aluminum', 'Catalan')
0.11764705882352944
>>> cmp.sim('ATCG', 'TAGC')
0.0
```

New in version 0.4.0.

class abydos.distance.**Morisita**(***kwargs*)
Bases: abydos.distance._token_distance._TokenDistance

Morisita index of overlap.

Morisita index of overlap [Mor59], following the description of [Hor66], given two populations X and Y drawn from S species, is:

$$sim_{Morisita}(X, Y) = C_\lambda = \frac{2\sum_{i=1}^{S} x_i y_i}{(\lambda_x + \lambda_y)XY}$$

where

$$X = \sum_{i=1}^{S} x_i \;\; ; \;\; Y = \sum_{i=1}^{S} y_i$$

$$\lambda_x = \frac{\sum_{i=1}^{S} x_i(x_i - 1)}{X(X-1)} \;\; ; \;\; \lambda_y = \frac{\sum_{i=1}^{S} y_i(y_i - 1)}{Y(Y-1)}$$

New in version 0.4.1.

Initialize Morisita instance.

Parameters ****kwargs** -- Arbitrary keyword arguments

New in version 0.4.1.

dist(**args*, ***kwargs*)
Raise exception when called.

Parameters

- ***args** -- Variable length argument list
- ****kwargs** -- Arbitrary keyword arguments

Raises **NotImplementedError** -- Method disabled for Morisita similarity.

New in version 0.3.6.

sim(**args*, ***kwargs*)
Raise exception when called.

Parameters

- ***args** -- Variable length argument list
- ****kwargs** -- Arbitrary keyword arguments

Raises **NotImplementedError** -- Method disabled for Morisita similarity.

New in version 0.3.6.

sim_score(*src*, *tar*)
Return the Morisita similarity of two strings.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns Morisita similarity

Return type float

Examples

```
>>> cmp = Morisita()
>>> cmp.sim_score('cat', 'hat')
0.25
>>> cmp.sim_score('Niall', 'Neil')
0.13333333333333333
>>> cmp.sim_score('aluminum', 'Catalan')
1.0
>>> cmp.sim_score('ATCG', 'TAGC')
0.0
```

New in version 0.4.1.

class abydos.distance.**Manhattan**(*alphabet=0*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)
Bases: abydos.distance._minkowski.Minkowski

Manhattan distance.

Manhattan distance is the city-block or taxi-cab distance, equivalent to Minkowski distance in L^1-space.

New in version 0.3.6.

Initialize Manhattan instance.

Parameters

- **alphabet** (*collection or int*) -- The values or size of the alphabet
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.

- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

dist(*src*, *tar*)

Return the normalized Manhattan distance between two strings.

The normalized Manhattan distance is a distance metric in L^1-space, normalized to [0, 1].

This is identical to Canberra distance.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns The normalized Manhattan distance

Return type float

Examples

```
>>> cmp = Manhattan()
>>> cmp.dist('cat', 'hat')
0.5
>>> round(cmp.dist('Niall', 'Neil'), 12)
0.636363636364
>>> round(cmp.dist('Colin', 'Cuilen'), 12)
0.692307692308
>>> cmp.dist('ATCG', 'TAGC')
1.0
```

New in version 0.3.0.

Changed in version 0.3.6: Encapsulated in class

dist_abs(*src*, *tar*, *normalized=False*)

Return the Manhattan distance between two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison
- **normalized** (*bool*) -- Normalizes to [0, 1] if True

Returns The Manhattan distance

Return type float

Examples

```
>>> cmp = Manhattan()
>>> cmp.dist_abs('cat', 'hat')
4.0
>>> cmp.dist_abs('Niall', 'Neil')
7.0
>>> cmp.dist_abs('Colin', 'Cuilen')
9.0
>>> cmp.dist_abs('ATCG', 'TAGC')
10.0
```

New in version 0.3.0.

Changed in version 0.3.6: Encapsulated in class

abydos.distance.**manhattan**(*src*, *tar*, *qval=2*, *normalized=False*, *alphabet=None*)

Return the Manhattan distance between two strings.

This is a wrapper for *Manhattan.dist_abs()*.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison
- **qval** (*int*) -- The length of each q-gram
- **normalized** (*bool*) -- Normalizes to [0, 1] if True
- **alphabet** (*collection or int*) -- The values or size of the alphabet

Returns The Manhattan distance

Return type float

Examples

```
>>> manhattan('cat', 'hat')
4.0
>>> manhattan('Niall', 'Neil')
7.0
>>> manhattan('Colin', 'Cuilen')
9.0
>>> manhattan('ATCG', 'TAGC')
10.0
```

New in version 0.3.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the Manhattan.dist_abs method instead.

abydos.distance.**dist_manhattan**(*src*, *tar*, *qval=2*, *alphabet=0*)

Return the normalized Manhattan distance between two strings.

This is a wrapper for *Manhattan.dist()*.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

- **qval** (*int*) -- The length of each q-gram
- **alphabet** (*collection or int*) -- The values or size of the alphabet

Returns The normalized Manhattan distance

Return type float

Examples

```
>>> dist_manhattan('cat', 'hat')
0.5
>>> round(dist_manhattan('Niall', 'Neil'), 12)
0.636363636364
>>> round(dist_manhattan('Colin', 'Cuilen'), 12)
0.692307692308
>>> dist_manhattan('ATCG', 'TAGC')
1.0
```

New in version 0.3.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the Manhattan.dist method instead.

abydos.distance.**sim_manhattan**(*src*, *tar*, *qval=2*, *alphabet=0*)

Return the normalized Manhattan similarity of two strings.

This is a wrapper for `Manhattan.sim()`.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison
- **qval** (*int*) -- The length of each q-gram
- **alphabet** (*collection or int*) -- The values or size of the alphabet

Returns The normalized Manhattan similarity

Return type float

Examples

```
>>> sim_manhattan('cat', 'hat')
0.5
>>> round(sim_manhattan('Niall', 'Neil'), 12)
0.363636363636
>>> round(sim_manhattan('Colin', 'Cuilen'), 12)
0.307692307692
>>> sim_manhattan('ATCG', 'TAGC')
0.0
```

New in version 0.3.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the Manhattan.sim method instead.

class abydos.distance.**Michelet**(*tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Michelet similarity.

For two sets X and Y and a population N, Michelet similarity [TCLM88] is

$$sim_{Michelet}(X, Y) = \frac{|X \cap Y|^2}{|X| \cdot |Y|}$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$sim_{Michelet} = \frac{a^2}{(a+b)(a+c)}$$

Following [Seq18], this is termed "Michelet", though Turner is most often listed as the first author in papers presenting this measure.

New in version 0.4.0.

Initialize Michelet instance.

Parameters

- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in `_TokenDistance` for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

sim(*src*, *tar*)

Return the Michelet similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Michelet similarity

Return type float

Examples

```
>>> cmp = Michelet()
>>> cmp.sim('cat', 'hat')
0.25
>>> cmp.sim('Niall', 'Neil')
0.13333333333333333
>>> cmp.sim('aluminum', 'Catalan')
0.013888888888888888
>>> cmp.sim('ATCG', 'TAGC')
0.0
```

New in version 0.4.0.

class abydos.distance.**Millar**(***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Millar's binomial deviance dissimilarity.

For two sets X and Y drawn from a population S, Millar's binomial deviance dissimilarity [AM04] is:

$$dist_{Millar}(X,Y) = \sum_{i=0}^{|S|} \frac{1}{x_i + y_i} \left\{ x_i log(\frac{x_i}{x_i + y_i}) + y_i log(\frac{y_i}{x_i + y_i}) - (x_i + y_i) log(\frac{1}{2}) \right\}$$

New in version 0.4.1.

Initialize Millar instance.

Parameters ****kwargs** -- Arbitrary keyword arguments

New in version 0.4.1.

dist(**args*, ***kwargs*)

Raise exception when called.

Parameters

- ***args** -- Variable length argument list
- ****kwargs** -- Arbitrary keyword arguments

Raises **NotImplementedError** -- Method disabled for Millar dissimilarity.

New in version 0.3.6.

dist_abs(*src*, *tar*)

Return Millar's binomial deviance dissimilarity of two strings.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns Millar's binomial deviance dissimilarity

Return type float

Examples

```
>>> cmp = Millar()
>>> cmp.dist_abs('cat', 'hat')
2.772588722239781
>>> cmp.dist_abs('Niall', 'Neil')
4.852030263919617
>>> cmp.dist_abs('aluminum', 'Catalan')
9.704060527839234
>>> cmp.dist_abs('ATCG', 'TAGC')
6.931471805599453
```

New in version 0.4.1.

sim(**args*, ***kwargs*)
Raise exception when called.

Parameters

- ***args** -- Variable length argument list
- ****kwargs** -- Arbitrary keyword arguments

Raises **NotImplementedError** -- Method disabled for Millar dissimilarity.

New in version 0.3.6.

class abydos.distance.**Minkowski**(*pval=1*, *alphabet=0*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)
Bases: abydos.distance._token_distance._TokenDistance

Minkowski distance.

The Minkowski distance [Min10] is a distance metric in $L^p - space$.

New in version 0.3.6.

Initialize Euclidean instance.

Parameters

- **pval** (*int*) -- The p-value of the L^p-space
- **alphabet** (*collection or int*) -- The values or size of the alphabet
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the soft and fuzzy variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the fuzzy variant.

New in version 0.4.0.

dist (*src*, *tar*)

Return normalized Minkowski distance of two strings.

The normalized Minkowski distance [Min10] is a distance metric in L^p-space, normalized to [0, 1].

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns The normalized Minkowski distance

Return type float

Examples

```
>>> cmp = Minkowski()
>>> cmp.dist('cat', 'hat')
0.5
>>> round(cmp.dist('Niall', 'Neil'), 12)
0.636363636364
>>> round(cmp.dist('Colin', 'Cuilen'), 12)
0.692307692308
>>> cmp.dist('ATCG', 'TAGC')
1.0
```

New in version 0.3.0.

Changed in version 0.3.6: Encapsulated in class

dist_abs (*src*, *tar*, *normalized=False*)

Return the Minkowski distance (L^p-norm) of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison
- **normalized** (*bool*) -- Normalizes to [0, 1] if True

Returns The Minkowski distance

Return type float

Examples

```
>>> cmp = Minkowski()
>>> cmp.dist_abs('cat', 'hat')
4.0
>>> cmp.dist_abs('Niall', 'Neil')
7.0
>>> cmp.dist_abs('Colin', 'Cuilen')
9.0
>>> cmp.dist_abs('ATCG', 'TAGC')
10.0
```

New in version 0.3.0.

Changed in version 0.3.6: Encapsulated in class

abydos.distance.**minkowski**(*src*, *tar*, *qval=2*, *pval=1*, *normalized=False*, *alphabet=0*)

Return the Minkowski distance (L^p-norm) of two strings.

This is a wrapper for *Minkowski.dist_abs()*.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison
- **qval** (*int*) -- The length of each q-gram
- **pval** (*int or float*) -- The p-value of the L^p-space
- **normalized** (*bool*) -- Normalizes to [0, 1] if True
- **alphabet** (*collection or int*) -- The values or size of the alphabet

Returns The Minkowski distance

Return type float

Examples

```
>>> minkowski('cat', 'hat')
4.0
>>> minkowski('Niall', 'Neil')
7.0
>>> minkowski('Colin', 'Cuilen')
9.0
>>> minkowski('ATCG', 'TAGC')
10.0
```

New in version 0.3.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the Minkowski.dist_abs method instead.

abydos.distance.**dist_minkowski**(*src*, *tar*, *qval=2*, *pval=1*, *alphabet=0*)

Return normalized Minkowski distance of two strings.

This is a wrapper for *Minkowski.dist()*.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison
- **qval** (*int*) -- The length of each q-gram
- **pval** (*int or float*) -- The p-value of the L^p-space
- **alphabet** (*collection or int*) -- The values or size of the alphabet

Returns The normalized Minkowski distance

Return type float

Examples

```
>>> dist_minkowski('cat', 'hat')
0.5
>>> round(dist_minkowski('Niall', 'Neil'), 12)
0.636363636364
>>> round(dist_minkowski('Colin', 'Cuilen'), 12)
0.692307692308
>>> dist_minkowski('ATCG', 'TAGC')
1.0
```

New in version 0.3.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the Minkowski.dist method instead.

abydos.distance.**sim_minkowski**(*src*, *tar*, *qval=2*, *pval=1*, *alphabet=0*)

Return normalized Minkowski similarity of two strings.

This is a wrapper for `Minkowski.sim()`.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison
- **qval** (*int*) -- The length of each q-gram
- **pval** (*int or float*) -- The p-value of the L^p-space
- **alphabet** (*collection or int*) -- The values or size of the alphabet

Returns The normalized Minkowski similarity

Return type float

Examples

```
>>> sim_minkowski('cat', 'hat')
0.5
>>> round(sim_minkowski('Niall', 'Neil'), 12)
0.363636363636
>>> round(sim_minkowski('Colin', 'Cuilen'), 12)
0.307692307692
>>> sim_minkowski('ATCG', 'TAGC')
0.0
```

New in version 0.3.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the Minkowski.sim method instead.

class abydos.distance.**MASI**(*tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

MASI similarity.

Measuring Agreement on Set-valued Items (MASI) similarity [Pas06] for two sets X and Y is based on Jaccard similarity:

$$sim_{Jaccard}(X, Y) = \frac{|X \cap Y|}{|X \cup Y|}$$

This Jaccard similarity is scaled by a value M, which is:

- 1 if $X = Y$
- $\frac{2}{3}$ if $X \subset Y$ or $Y \subset X$
- $\frac{1}{3}$ if $X \cap Y \neq \emptyset$, $X \setminus Y \neq \emptyset$, and $Y \setminus X \neq \emptyset$
- 0 if $X \cap Y = \emptyset$

New in version 0.4.0.

Initialize MASI instance.

Parameters

- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in `_TokenDistance` for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

sim(*src*, *tar*)

Return the MASI similarity of two strings.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns MASI similarity

Return type float

Examples

```
>>> cmp = MASI()
>>> cmp.sim('cat', 'hat')
0.1111111111111111
>>> cmp.sim('Niall', 'Neil')
0.07407407407407407
>>> cmp.sim('aluminum', 'Catalan')
0.020833333333333332
>>> cmp.sim('ATCG', 'TAGC')
0.0
```

New in version 0.4.0.

class abydos.distance.**Matusita**(*tokenizer=None*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Matusita distance.

For two multisets X and Y drawn from an alphabet S, Matusita distance [Mat55] is

$$dist_{Matusita}(X, Y) = \sqrt{\sum_{i \in S} \left(\sqrt{\frac{|A_i|}{|A|}} - \sqrt{\frac{|B_i|}{|B|}} \right)^2}$$

New in version 0.4.0.

Initialize Matusita instance.

Parameters

- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.

New in version 0.4.0.

dist(*src*, *tar*)

Return the normalized Matusita distance of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Normalized Matusita distance

Return type float

Examples

```
>>> cmp = Matusita()
>>> cmp.dist('cat', 'hat')
0.707106781186547
>>> cmp.dist('Niall', 'Neil')
0.796775770420944
>>> cmp.dist('aluminum', 'Catalan')
0.939227805062351
>>> cmp.dist('ATCG', 'TAGC')
1.0
```

New in version 0.4.0.

dist_abs(*src*, *tar*)

Return the Matusita distance of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison

- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Matusita distance

Return type float

Examples

```
>>> cmp = Matusita()
>>> cmp.dist_abs('cat', 'hat')
1.0
>>> cmp.dist_abs('Niall', 'Neil')
1.126811100699571
>>> cmp.dist_abs('aluminum', 'Catalan')
1.3282687000770907
>>> cmp.dist_abs('ATCG', 'TAGC')
1.414213562373095
```

New in version 0.4.0.

class abydos.distance.**MaxwellPilliner**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Maxwell & Pilliner correlation.

For two sets X and Y and a population N, Maxwell & Pilliner correlation [MP68] is

$$corr_{MaxwellPilliner}(X, Y) = \frac{2(|X \cap Y| \cdot |(N \setminus X) \setminus Y| - |X \setminus Y| \cdot |Y \setminus X|)}{|X| \cdot |N \setminus X| + |Y| \cdot |N \setminus Y|}$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$corr_{MaxwellPilliner} = \frac{2(ad - bc)}{(a + b)(c + d) + (a + c)(b + c)}$$

New in version 0.4.0.

Initialize MaxwellPilliner instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in _TokenDistance for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.

- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

corr(*src*, *tar*)

Return the Maxwell & Pilliner correlation of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Maxwell & Pilliner correlation

Return type float

Examples

```
>>> cmp = MaxwellPilliner()
>>> cmp.corr('cat', 'hat')
0.49743589743589745
>>> cmp.corr('Niall', 'Neil')
0.35921989956790845
>>> cmp.corr('aluminum', 'Catalan')
0.10803030303030303
>>> cmp.corr('ATCG', 'TAGC')
-0.006418485237483954
```

New in version 0.4.0.

sim(*src*, *tar*)

Return the Maxwell & Pilliner similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Maxwell & Pilliner similarity

Return type float

Examples

```
>>> cmp = MaxwellPilliner()
>>> cmp.sim('cat', 'hat')
0.7487179487179487
>>> cmp.sim('Niall', 'Neil')
0.6796099497839543
>>> cmp.sim('aluminum', 'Catalan')
0.5540151515151515
>>> cmp.sim('ATCG', 'TAGC')
0.496790757381258
```

New in version 0.4.0.

class abydos.distance.**McConnaughey**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

McConnaughey correlation.

For two sets X and Y, McConnaughey correlation [McC64] is

$$corr_{McConnaughey}(X, Y) = \frac{|X \cap Y|^2 - |X \setminus Y| \cdot |Y \setminus X|}{|X| \cdot |Y|}$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$corr_{McConnaughey} = \frac{a^2 - bc}{(a+b)(a+c)}$$

New in version 0.4.0.

Initialize McConnaughey instance.

Parameters

- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the soft and fuzzy variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the fuzzy variant.

New in version 0.4.0.

corr(*src*, *tar*)

Return the McConnaughey correlation of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns McConnaughey correlation

Return type float

Examples

```
>>> cmp = McConnaughey()
>>> cmp.corr('cat', 'hat')
0.0
>>> cmp.corr('Niall', 'Neil')
-0.2666666666666666
>>> cmp.corr('aluminum', 'Catalan')
-0.7638888888888888
>>> cmp.corr('ATCG', 'TAGC')
-1.0
```

New in version 0.4.0.

sim(*src*, *tar*)

Return the McConnaughey similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns McConnaughey similarity

Return type float

Examples

```
>>> cmp = McConnaughey()
>>> cmp.sim('cat', 'hat')
0.5
>>> cmp.sim('Niall', 'Neil')
0.3666666666666667
>>> cmp.sim('aluminum', 'Catalan')
0.11805555555555558
>>> cmp.sim('ATCG', 'TAGC')
0.0
```

New in version 0.4.0.

class abydos.distance.**McEwenMichael**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

McEwen & Michael correlation.

For two sets X and Y and a population N, the McEwen & Michael correlation [Mic20] is

$$corr_{McEwenMichael}(X, Y) = \frac{4(|X \cap Y| \cdot |(N \setminus X) \setminus Y| - |X \setminus Y| \cdot |Y \setminus X|)}{(|X \cap Y| + |(N \setminus X) \setminus Y|)^2 + (|X \setminus Y| + |Y \setminus X|)^2}$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$corr_{McEwenMichael} = \frac{4(ad - bc)}{(a + d)^2 + (b + c)^2}$$

New in version 0.4.0.

Initialize Michael instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in `_TokenDistance` for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in `_TokenDistance` for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

corr(*src*, *tar*)

Return the McEwen & Michael correlation of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Michael correlation

Return type float

Examples

```
>>> cmp = McEwenMichael()
>>> cmp.corr('cat', 'hat')
0.010203544942933782
>>> cmp.corr('Niall', 'Neil')
0.010189175491654217
>>> cmp.corr('aluminum', 'Catalan')
0.0048084299262381456
>>> cmp.corr('ATCG', 'TAGC')
-0.00016689587032858459
```

New in version 0.4.0.

sim(*src*, *tar*)

Return the McEwen & Michael similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison

- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Michael similarity

Return type float

Examples

```
>>> cmp = McEwenMichael()
>>> cmp.sim('cat', 'hat')
0.5051017724714669
>>> cmp.sim('Niall', 'Neil')
0.5050945877458272
>>> cmp.sim('aluminum', 'Catalan')
0.502404214963119
>>> cmp.sim('ATCG', 'TAGC')
0.4999165520648357
```

New in version 0.4.0.

class abydos.distance.**Mountford**(*tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Mountford similarity.

For two sets X and Y, the Mountford similarity [Mou62] is

$$sim_{Mountford}(X, Y) = \frac{2|X \cap Y|}{2|X| \cdot |Y| - (|X| + |Y|) \cdot |X \cap Y|}$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$sim_{Mountford} = \frac{2a}{2(a+b)(a+c) - (2a+b+c)a}$$

New in version 0.4.0.

Initialize Mountford instance.

Parameters

- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the soft and fuzzy variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the fuzzy variant.

New in version 0.4.0.

sim(*src*, *tar*)

Return the Mountford similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Mountford similarity

Return type float

Examples

```
>>> cmp = Mountford()
>>> cmp.sim('cat', 'hat')
0.25
>>> cmp.sim('Niall', 'Neil')
0.10526315789473684
>>> cmp.sim('aluminum', 'Catalan')
0.015748031496062992
>>> cmp.sim('ATCG', 'TAGC')
0.0
```

New in version 0.4.0.

class abydos.distance.**MutualInformation**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Mutual Information similarity.

For two sets X and Y and a population N, Mutual Information similarity [CGHH91] is

$$sim_{MI}(X, Y) = log_2(\frac{|X \cap Y| \cdot |N|}{|X| \cdot |Y|})$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$sim_{MI} = log_2(\frac{an}{(a+b)(a+c)})$$

New in version 0.4.0.

Initialize MutualInformation instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in _TokenDistance for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.

- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

sim(*src*, *tar*)

Return the normalized Mutual Information similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Normalized Mutual Information similarity

Return type float

Examples

```
>>> cmp = MutualInformation()
>>> cmp.sim('cat', 'hat')
0.933609253088981
>>> cmp.sim('Niall', 'Neil')
0.8911684881725231
>>> cmp.sim('aluminum', 'Catalan')
0.7600321183863901
>>> cmp.sim('ATCG', 'TAGC')
0.17522996523538537
```

New in version 0.4.0.

sim_score(*src*, *tar*)

Return the Mutual Information similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Mutual Information similarity

Return type float

Examples

```
>>> cmp = MutualInformation()
>>> cmp.sim_score('cat', 'hat')
6.528166795717758
>>> cmp.sim_score('Niall', 'Neil')
5.661433326581222
>>> cmp.sim_score('aluminum', 'Catalan')
3.428560943378589
>>> cmp.sim_score('ATCG', 'TAGC')
-4.700439718141092
```

New in version 0.4.0.

class abydos.distance.**MSContingency**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Mean squared contingency correlation.

For two sets X and Y and a population N, the mean squared contingency correlation [Col49] is

$$corr_{MSContingency}(X,Y) = \frac{\sqrt{2}(|X \cap Y| \cdot |(N \setminus X) \setminus Y| - |X \setminus Y| \cdot |Y \setminus X|)}{\sqrt{(|X \cap Y| \cdot |(N \setminus X) \setminus Y| - |X \setminus Y| \cdot |Y \setminus X|)^2 + |X| \cdot |Y| \cdot |N \setminus X| \cdot |N \setminus Y|}}$$

[Hubalek08] and [CCT10] identify this as Cole similarity. Although Cole discusses this correlation, he does not claim to have developed it. Rather, he presents his coefficient of interspecific association as being his own development: *Cole*.

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$corr_{MSContingency} = \frac{\sqrt{2}(ad - bc)}{\sqrt{(ad - bc)^2 + (a + b)(a + c)(b + d)(c + d)}}$$

New in version 0.4.0.

Initialize MSContingency instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in _TokenDistance for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the soft and fuzzy variants.

- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

corr(*src*, *tar*)

Return the normalized mean squared contingency corr. of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Mean squared contingency correlation

Return type float

Examples

```
>>> cmp = MSContingency()
>>> cmp.corr('cat', 'hat')
0.6298568508557214
>>> cmp.corr('Niall', 'Neil')
0.4798371954796814
>>> cmp.corr('aluminum', 'Catalan')
0.15214891090821628
>>> cmp.corr('ATCG', 'TAGC')
-0.009076921903905553
```

New in version 0.4.0.

sim(*src*, *tar*)

Return the normalized ms contingency similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Mean squared contingency similarity

Return type float

Examples

```
>>> cmp = MSContingency()
>>> cmp.sim('cat', 'hat')
0.8149284254278607
>>> cmp.sim('Niall', 'Neil')
0.7399185977398407
>>> cmp.sim('aluminum', 'Catalan')
0.5760744554541082
>>> cmp.sim('ATCG', 'TAGC')
0.49546153904804724
```

New in version 0.4.0.

class abydos.distance.**Overlap**(*tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Overlap coefficient.

For two sets X and Y, the overlap coefficient [Szy34][Sim49], also called the Szymkiewicz-Simpson coefficient and Simpson's ecological coexistence coefficient, is

$$sim_{overlap}(X, Y) = \frac{|X \cap Y|}{min(|X|, |Y|)}$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$sim_{overlap} = \frac{a}{min(a + b, a + c)}$$

New in version 0.3.6.

Initialize Overlap instance.

Parameters

- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the soft and fuzzy variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the fuzzy variant.

New in version 0.4.0.

sim(*src*, *tar*)

Return the overlap coefficient of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Overlap similarity

Return type float

Examples

```
>>> cmp = Overlap()
>>> cmp.sim('cat', 'hat')
0.5
>>> cmp.sim('Niall', 'Neil')
0.4
>>> cmp.sim('aluminum', 'Catalan')
0.125
>>> cmp.sim('ATCG', 'TAGC')
0.0
```

New in version 0.1.0.

Changed in version 0.3.6: Encapsulated in class

abydos.distance.**dist_overlap**(*src*, *tar*, *qval=2*)

Return the overlap distance between two strings.

This is a wrapper for `Overlap.dist()`.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison
- **qval** (*int*) -- The length of each q-gram

Returns Overlap distance

Return type float

Examples

```
>>> dist_overlap('cat', 'hat')
0.5
>>> dist_overlap('Niall', 'Neil')
0.6
>>> dist_overlap('aluminum', 'Catalan')
0.875
>>> dist_overlap('ATCG', 'TAGC')
1.0
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the Overlap.dist method instead.

abydos.distance.**sim_overlap**(*src*, *tar*, *qval=2*)

Return the overlap coefficient of two strings.

This is a wrapper for *Overlap.sim()*.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison
- **qval** (*int*) -- The length of each q-gram

Returns Overlap similarity

Return type float

Examples

```
>>> sim_overlap('cat', 'hat')
0.5
>>> sim_overlap('Niall', 'Neil')
0.4
>>> sim_overlap('aluminum', 'Catalan')
0.125
>>> sim_overlap('ATCG', 'TAGC')
0.0
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the Overlap.sim method instead.

class abydos.distance.**Pattern**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Pattern difference.

For two sets X and Y and a population N, the pattern difference [BB95], Batagelj & Bren's $-bc-$ is

$$dist_{pattern}(X, Y) = \frac{4 \cdot |X \setminus Y| \cdot |Y \setminus X|}{|N|^2}$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$dist_{pattern} = \frac{4bc}{n^2}$$

In [Cor17], the formula omits the 4 in the numerator: $\frac{bc}{n^2}$.

New in version 0.4.0.

Initialize Pattern instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in _TokenDistance for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the soft and fuzzy variants.

- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

dist(*src*, *tar*)

Return the Pattern difference of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Pattern difference

Return type float

Examples

```
>>> cmp = Pattern()
>>> cmp.dist('cat', 'hat')
2.6030820491461892e-05
>>> cmp.dist('Niall', 'Neil')
7.809246147438568e-05
>>> cmp.dist('aluminum', 'Catalan')
0.0003635035904093472
>>> cmp.dist('ATCG', 'TAGC')
0.0001626926280716368
```

New in version 0.4.0.

class abydos.distance.**PearsonHeronII**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Pearson & Heron II correlation.

For two sets X and Y and a population N, Pearson & Heron II correlation [PH13] is

$$corr_{PearsonHeronII}(X, Y) = \cos\left(\frac{\pi\sqrt{|X \setminus Y| \cdot |Y \setminus X|}}{\sqrt{|X \cap Y| \cdot |(N \setminus X) \setminus Y|} + \sqrt{|X \setminus Y| \cdot |Y \setminus X|}}\right)$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$corr_{PearsonHeronII} = \cos\left(\frac{\pi\sqrt{bc}}{\sqrt{ad} + \sqrt{bc}}\right)$$

New in version 0.4.0.

Initialize PearsonHeronII instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in `_TokenDistance` for details.

- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in `_TokenDistance` for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

corr(*src*, *tar*)

Return the Pearson & Heron II correlation of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Pearson & Heron II correlation

Return type float

Examples

```
>>> cmp = PearsonHeronII()
>>> cmp.corr('cat', 'hat')
0.9885309061036239
>>> cmp.corr('Niall', 'Neil')
0.9678978997263907
>>> cmp.corr('aluminum', 'Catalan')
0.7853000893691571
>>> cmp.corr('ATCG', 'TAGC')
-1.0
```

New in version 0.4.0.

sim(*src*, *tar*)

Return the Pearson & Heron II similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Pearson & Heron II similarity

Return type float

Examples

```
>>> cmp = PearsonHeronII()
>>> cmp.sim('cat', 'hat')
0.994265453051812
>>> cmp.sim('Niall', 'Neil')
0.9839489498631954
>>> cmp.sim('aluminum', 'Catalan')
0.8926500446845785
>>> cmp.sim('ATCG', 'TAGC')
0.0
```

New in version 0.4.0.

class abydos.distance.**PearsonII**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._pearson_chi_squared.PearsonChiSquared

Pearson II similarity.

For two sets X and Y and a population N, the Pearson II similarity [PH13], Pearson's coefficient of mean square contingency, is

$$corr_{PearsonII} = \sqrt{\frac{\chi^2}{|N| + \chi^2}}$$

where

$$\chi^2 = sim_{PearsonChiSquared}(X, Y) = \frac{|N| \cdot (|X \cap Y| \cdot |(N \setminus X) \setminus Y| - |X \setminus Y| \cdot |Y \setminus X|)^2}{|X| \cdot |Y| \cdot |N \setminus X| \cdot |N \setminus Y|}$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$\chi^2 = sim_{PearsonChiSquared} = \frac{n \cdot (ad - bc)^2}{(a+b)(a+c)(b+d)(c+d)}$$

New in version 0.4.0.

Initialize PearsonII instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in _TokenDistance for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

sim(*src*, *tar*)

Return the normalized Pearson II similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Normalized Pearson II similarity

Return type float

Examples

```
>>> cmp = PearsonII()
>>> cmp.sim('cat', 'hat')
0.6298568508557214
>>> cmp.sim('Niall', 'Neil')
0.47983719547968123
>>> cmp.sim('aluminum', 'Catalan')
0.15214891090821628
>>> cmp.sim('ATCG', 'TAGC')
0.009076921903905551
```

New in version 0.4.0.

sim_score(*src*, *tar*)

Return the Pearson II similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Pearson II similarity

Return type float

Examples

```
>>> cmp = PearsonII()
>>> cmp.sim_score('cat', 'hat')
0.44537605041688455
>>> cmp.sim_score('Niall', 'Neil')
0.3392961347892176
>>> cmp.sim_score('aluminum', 'Catalan')
0.10758552665334761
>>> cmp.sim_score('ATCG', 'TAGC')
0.006418353030552324
```

New in version 0.4.0.

class abydos.distance.**PearsonIII**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._pearson_phi.PearsonPhi

Pearson III correlation.

For two sets X and Y and a population N, the Pearson III correlation [PH13], Pearson's coefficient of racial likeness, is

$$corr_{PearsonIII} = \sqrt{\frac{\phi}{|N| + \phi}}$$

where

$$\phi = corr_{PearsonPhi}(X, Y) = \frac{|X \cap Y| \cdot |(N \setminus X) \setminus Y| - |X \setminus Y| \cdot |Y \setminus X|}{\sqrt{|X| \cdot |Y| \cdot |N \setminus X| \cdot |N \setminus Y|}}$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$\phi = corr_{PearsonPhi} = \frac{ad - bc}{\sqrt{(a+b)(a+c)(b+c)(b+d)}}$$

New in version 0.4.0.

Initialize PearsonIII instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in _TokenDistance for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

corr(*src*, *tar*)

Return the Pearson III correlation of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Pearson III correlation

Return type float

Examples

```
>>> cmp = PearsonIII()
>>> cmp.corr('cat', 'hat')
0.025180989806958435
>>> cmp.corr('Niall', 'Neil')
0.021444241017487504
>>> cmp.corr('aluminum', 'Catalan')
0.011740218922356615
>>> cmp.corr('ATCG', 'TAGC')
-0.0028612777635371113
```

New in version 0.4.0.

sim(*src*, *tar*)

Return the Pearson III similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Pearson III similarity

Return type float

Examples

```
>>> cmp = PearsonIII()
>>> cmp.sim('cat', 'hat')
0.5125904949034792
>>> cmp.sim('Niall', 'Neil')
0.5107221205087438
>>> cmp.sim('aluminum', 'Catalan')
0.5058701094611783
>>> cmp.sim('ATCG', 'TAGC')
0.49856936111823147
```

New in version 0.4.0.

class abydos.distance.**PearsonChiSquared**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Pearson's Chi-Squared similarity.

For two sets X and Y and a population N, the Pearson's χ^2 similarity [PH13] is

$$sim_{PearsonChiSquared}(X, Y) = \frac{|N| \cdot (|X \cap Y| \cdot |(N \setminus X) \setminus Y| - |X \setminus Y| \cdot |Y \setminus X|)^2}{|X| \cdot |Y| \cdot |N \setminus X| \cdot |N \setminus Y|}$$

This is also Pearson I similarity.

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$sim_{PearsonChiSquared} = \frac{n(ad - bc)^2}{(a + b)(a + c)(b + d)(c + d)}$$

New in version 0.4.0.

Initialize PearsonChiSquared instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in _TokenDistance for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the soft and fuzzy variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the fuzzy variant.

New in version 0.4.0.

corr(*src*, *tar*)

Return Pearson's Chi-Squared correlation of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Pearson's Chi-Squared correlation

Return type float

Examples

```
>>> cmp = PearsonChiSquared()
>>> cmp.corr('cat', 'hat')
0.2474424720578567
>>> cmp.corr('Niall', 'Neil')
0.1300991207720222
>>> cmp.corr('aluminum', 'Catalan')
0.011710186806836291
>>> cmp.corr('ATCG', 'TAGC')
-4.1196952743799446e-05
```

New in version 0.4.0.

sim(*src*, *tar*)

Return Pearson's normalized Chi-Squared similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Normalized Pearson's Chi-Squared similarity

Return type float

Examples

```
>>> cmp = PearsonChiSquared()
>>> cmp.corr('cat', 'hat')
0.2474424720578567
>>> cmp.corr('Niall', 'Neil')
0.1300991207720222
>>> cmp.corr('aluminum', 'Catalan')
0.011710186806836291
>>> cmp.corr('ATCG', 'TAGC')
-4.1196952743799446e-05
```

New in version 0.4.0.

sim_score(*src*, *tar*)

Return Pearson's Chi-Squared similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Pearson's Chi-Squared similarity

Return type float

Examples

```
>>> cmp = PearsonChiSquared()
>>> cmp.sim_score('cat', 'hat')
193.99489809335964
>>> cmp.sim_score('Niall', 'Neil')
101.99771068526542
>>> cmp.sim_score('aluminum', 'Catalan')
9.19249664336649
>>> cmp.sim_score('ATCG', 'TAGC')
0.032298410951138765
```

New in version 0.4.0.

class abydos.distance.**PearsonPhi**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Pearson's Phi correlation.

For two sets X and Y and a population N, the Pearson's ϕ correlation [Pea00][PH13][Guirk] is

$$corr_{PearsonPhi}(X, Y) = \frac{|X \cap Y| \cdot |(N \setminus X) \setminus Y| - |X \setminus Y| \cdot |Y \setminus X|}{\sqrt{|X| \cdot |Y| \cdot |N \setminus X| \cdot |N \setminus Y|}}$$

This is also Pearson & Heron I similarity.

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$corr_{PearsonPhi} = \frac{ad - bc}{\sqrt{(a+b)(a+c)(b+d)(c+d)}}$$

Notes

In terms of a confusion matrix, this is equivalent to the Matthews correlation coefficient ConfusionTable.mcc().

New in version 0.4.0.

Initialize PearsonPhi instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in _TokenDistance for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package

- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in `_TokenDistance` for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

corr(*src*, *tar*)

Return Pearson's Phi correlation of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Pearson's Phi correlation

Return type float

Examples

```
>>> cmp = PearsonPhi()
>>> cmp.corr('cat', 'hat')
0.49743589743589745
>>> cmp.corr('Niall', 'Neil')
0.36069255713421955
>>> cmp.corr('aluminum', 'Catalan')
0.10821361655002706
>>> cmp.corr('ATCG', 'TAGC')
-0.006418485237483954
```

New in version 0.4.0.

sim(*src*, *tar*)

Return the normalized Pearson's Phi similarity of two strings.

This is normalized to [0, 1] by adding 1 and dividing by 2.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Pearson's Phi similarity

Return type float

Examples

```
>>> cmp = PearsonPhi()
>>> cmp.sim('cat', 'hat')
0.7487179487179487
>>> cmp.sim('Niall', 'Neil')
0.6803462785671097
>>> cmp.sim('aluminum', 'Catalan')
0.5541068082750136
>>> cmp.sim('ATCG', 'TAGC')
0.496790757381258
```

New in version 0.4.0.

class abydos.distance.**Peirce**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Peirce correlation.

For two sets X and Y and a population N, the Peirce correlation [Pei84] is

$$corr_{Peirce}(X, Y) = \frac{|X \cap Y| \cdot |(N \setminus X) \setminus Y| - |X \setminus Y| \cdot |Y \setminus Y|}{|X| \cdot |N \setminus X|}$$

Both [CCT10] and [Hubalek08] present a different formula and incorrectly attribute it to Peirce. Likewise, [Doo84] presents a different formula and incorrectly attributes it to Peirce. This is distinct from the formula he presents and attributes to himself.

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$corr_{Peirce} = \frac{ad - bc}{(a + b)(c + d)}$$

New in version 0.4.0.

Initialize Peirce instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in _TokenDistance for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the soft and fuzzy variants.

- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

corr(*src*, *tar*)

Return the Peirce correlation of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Peirce correlation

Return type float

Examples

```
>>> cmp = Peirce()
>>> cmp.corr('cat', 'hat')
0.49743589743589745
>>> cmp.corr('Niall', 'Neil')
0.32947729220222793
>>> cmp.corr('aluminum', 'Catalan')
0.10209049255441008
>>> cmp.corr('ATCG', 'TAGC')
-0.006418485237483954
```

New in version 0.4.0.

sim(*src*, *tar*)

Return the Peirce similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Peirce similarity

Return type float

Examples

```
>>> cmp = Peirce()
>>> cmp.sim('cat', 'hat')
0.7487179487179487
>>> cmp.sim('Niall', 'Neil')
0.664738646101114
>>> cmp.sim('aluminum', 'Catalan')
0.5510452462772051
>>> cmp.sim('ATCG', 'TAGC')
0.496790757381258
```

New in version 0.4.0.

class abydos.distance.**QGram**(*tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

q-gram distance.

For two multisets X and Y, q-gram distance [Ukk92] is

$$sim_{QGram}(X, Y) = |X \triangle Y|$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$sim_{QGram} = b + c$$

Notes

This class uses bigrams without appended start or stop symbols, by default, as in [Ukk92]'s examples. It is described as the L_1 norm of the difference of two strings' q-gram profiles, which are the vectors of q-gram occurrences. But this norm is simply the symmetric difference of the two multisets.

There aren't any limitations on which tokenizer is used with this class, but, as the name would imply, q-grams are expected and the default.

The normalized form uses the union of X and Y, making it equivalent to the Jaccard distance *Jaccard*, but the Jaccard class, by default uses bigrams with start & stop symbols.

New in version 0.4.0.

Initialize QGram instance.

Parameters

- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the soft and fuzzy variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the fuzzy variant.

New in version 0.4.0.

dist(*src*, *tar*)

Return the normalized q-gram distance of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison

- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns q-gram distance

Return type float

Examples

```
>>> cmp = QGram()
>>> cmp.sim('cat', 'hat')
0.3333333333333333
>>> cmp.sim('Niall', 'Neil')
0.0
>>> cmp.sim('aluminum', 'Catalan')
0.08333333333333337
>>> cmp.sim('ATCG', 'TAGC')
0.0
```

New in version 0.4.0.

dist_abs(*src*, *tar*)

Return the q-gram distance of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns q-gram distance

Return type int

Examples

```
>>> cmp = QGram()
>>> cmp.dist_abs('cat', 'hat')
2
>>> cmp.dist_abs('Niall', 'Neil')
7
>>> cmp.dist_abs('aluminum', 'Catalan')
11
>>> cmp.dist_abs('ATCG', 'TAGC')
6
>>> cmp.dist_abs('01000', '001111')
5
```

New in version 0.4.0.

class abydos.distance.**RaupCrick**(***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Raup-Crick similarity.

For two sets X and Y and a population N, Raup-Crick similarity [RC79] is:

Notes

Observe that Raup-Crick similarity is related to Henderson-Heron similarity in that the former is the sum of all Henderson-Heron similarities for an intersection size ranging from 0 to the true intersection size.

New in version 0.4.1.

Initialize RaupCrick instance.

Parameters ****kwargs** -- Arbitrary keyword arguments

New in version 0.4.1.

sim(*src*, *tar*)
Return the Raup-Crick similarity of two strings.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns Raup-Crick similarity

Return type float

Examples

```
>>> cmp = RaupCrick()
>>> cmp.sim('cat', 'hat')
0.9999998002120004
>>> cmp.sim('Niall', 'Neil')
0.9999975146378747
>>> cmp.sim('aluminum', 'Catalan')
0.9968397599851411
>>> cmp.sim('ATCG', 'TAGC')
0.9684367974410505
```

New in version 0.4.1.

class abydos.distance.**ReesLevenshtein**(*block_limit=2*, *normalizer=<built-in function max>*, ***kwargs*)
Bases: abydos.distance._distance._Distance

Rees-Levenshtein distance.

Rees-Levenshtein distance [Ree14][RB13] is the "Modified Damerau-Levenshtein Distance Algorithm, created by Tony Rees as part of Taxamatch.

New in version 0.4.0.

Initialize ReesLevenshtein instance.

Parameters

- **block_limit** (*int*) -- The block length limit
- **normalizer** (*function*) -- A function that takes an list and computes a normalization term by which the edit distance is divided (max by default). Another good option is the sum function.
- ****kwargs** -- Arbitrary keyword arguments

New in version 0.4.0.

dist (*src*, *tar*)

Return the normalized Rees-Levenshtein distance of two strings.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns Normalized Rees-Levenshtein distance

Return type float

Examples

```
>>> cmp = ReesLevenshtein()
>>> cmp.dist('cat', 'hat')
0.3333333333333333
>>> cmp.dist('Niall', 'Neil')
0.6
>>> cmp.dist('aluminum', 'Catalan')
0.875
>>> cmp.dist('ATCG', 'TAGC')
0.5
```

New in version 0.4.0.

dist_abs (*src*, *tar*)

Return the Rees-Levenshtein distance of two strings.

This is a straightforward port of the PL/SQL implementation at https://confluence.csiro.au/public/taxamatch/the-mdld-modified-damerau-levenshtein-distance-algorithm

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns Rees-Levenshtein distance

Return type float

Examples

```
>>> cmp = ReesLevenshtein()
>>> cmp.dist_abs('cat', 'hat')
1
>>> cmp.dist_abs('Niall', 'Neil')
3
>>> cmp.dist_abs('aluminum', 'Catalan')
7
>>> cmp.dist_abs('ATCG', 'TAGC')
2
```

New in version 0.4.0.

class abydos.distance.**RogersTanimoto**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Rogers & Tanimoto similarity.

For two sets X and Y and a population N, the Rogers-Tanimoto similarity [RT60] is

$$sim_{RogersTanimoto}(X, Y) = \frac{|X \cap Y| + |(N \setminus X) \setminus Y|}{|X \setminus Y| + |Y \setminus X| + |N|}$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$sim_{RogersTanimoto} = \frac{a + d}{b + c + n}$$

New in version 0.4.0.

Initialize RogersTanimoto instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in `_TokenDistance` for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in `_TokenDistance` for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

sim(*src*, *tar*)

Return the Rogers & Tanimoto similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Rogers & Tanimoto similarity

Return type float

Examples

```
>>> cmp = RogersTanimoto()
>>> cmp.sim('cat', 'hat')
0.9898477157360406
>>> cmp.sim('Niall', 'Neil')
0.9823008849557522
>>> cmp.sim('aluminum', 'Catalan')
0.9625
>>> cmp.sim('ATCG', 'TAGC')
0.9748110831234257
```

New in version 0.4.0.

class abydos.distance.**RogotGoldberg**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Rogot & Goldberg similarity.

For two sets X and Y and a population N, Rogot & Goldberg's "second index adjusted agreement" A_2 [RG66] is

$$sim_{RogotGoldberg}(X, Y) = \frac{1}{2}\left(\frac{2|X \cap Y|}{|X| + |Y|} + \frac{2|(N \setminus X) \setminus Y|}{|N \setminus X| + |N \setminus Y|}\right)$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$sim_{RogotGoldberg} = \frac{1}{2}\left(\frac{2a}{2a + b + c} + \frac{2d}{2d + b + c}\right)$$

New in version 0.4.0.

Initialize RogotGoldberg instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in _TokenDistance for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the soft and fuzzy variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the fuzzy variant.

New in version 0.4.0.

sim(*src*, *tar*)

Return the Rogot & Goldberg similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Rogot & Goldberg similarity

Return type float

Examples

```
>>> cmp = RogotGoldberg()
>>> cmp.sim('cat', 'hat')
0.7487179487179487
>>> cmp.sim('Niall', 'Neil')
0.6795702691656449
>>> cmp.sim('aluminum', 'Catalan')
0.5539941668876179
>>> cmp.sim('ATCG', 'TAGC')
0.496790757381258
```

New in version 0.4.0.

class abydos.distance.**RussellRao**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Russell & Rao similarity.

For two sets X and Y and a population N, the Russell & Rao similarity [RR40] is

$$sim_{RussellRao}(X, Y) = \frac{|X \cap Y|}{|N|}$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$sim_{RussellRao} = \frac{a}{n}$$

New in version 0.4.0.

Initialize RussellRao instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in _TokenDistance for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.

- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

sim(*src*, *tar*)

Return the Russell & Rao similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Russell & Rao similarity

Return type float

Examples

```
>>> cmp = RussellRao()
>>> cmp.sim('cat', 'hat')
0.002551020408163265
>>> cmp.sim('Niall', 'Neil')
0.002551020408163265
>>> cmp.sim('aluminum', 'Catalan')
0.0012738853503184713
>>> cmp.sim('ATCG', 'TAGC')
0.0
```

New in version 0.4.0.

class abydos.distance.**ScottPi**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Scott's Pi correlation.

For two sets X and Y and a population N, Scott's π correlation [Sco55] is

$$corr_{Scott_\pi}(X, Y) = \pi = \frac{p_o - p_e^\pi}{1 - p_e^\pi}$$

where

$$\begin{aligned} p_o &= \frac{|X \cap Y| + |(N \setminus X) \setminus Y|}{|N|} \\ p_e^\pi &= \left(\frac{|X| + |Y|}{2 \cdot |N|}\right)^2 + \left(\frac{|N \setminus X| + |N \setminus Y|}{2 \cdot |N|}\right)^2 \end{aligned}$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$\begin{aligned} p_o &= \frac{a+d}{n} \\ p_e^{\pi} &= \left(\frac{2a+b+c}{2n}\right)^2 + \left(\frac{2d+b+c}{2n}\right)^2 \end{aligned}$$

New in version 0.4.0.

Initialize ScottPi instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in _TokenDistance for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the soft and fuzzy variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the fuzzy variant.

New in version 0.4.0.

corr(*src*, *tar*)

Return the Scott's Pi correlation of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Scott's Pi correlation

Return type float

Examples

```
>>> cmp = ScottPi()
>>> cmp.corr('cat', 'hat')
0.49743589743589733
>>> cmp.corr('Niall', 'Neil')
0.35914053833129245
>>> cmp.corr('aluminum', 'Catalan')
0.10798833377524023
>>> cmp.corr('ATCG', 'TAGC')
-0.006418485237489689
```

New in version 0.4.0.

sim(*src*, *tar*)
Return the Scott's Pi similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Scott's Pi similarity

Return type float

Examples

```
>>> cmp = ScottPi()
>>> cmp.sim('cat', 'hat')
0.7487179487179487
>>> cmp.sim('Niall', 'Neil')
0.6795702691656462
>>> cmp.sim('aluminum', 'Catalan')
0.5539941668876202
>>> cmp.sim('ATCG', 'TAGC')
0.49679075738125517
```

New in version 0.4.0.

class abydos.distance.**Shape**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)
Bases: abydos.distance._token_distance._TokenDistance

Penrose's shape difference.

For two sets X and Y and a population N, the Penrose's shape difference [Pen52] is

$$dist_{Shape}(X, Y) = \frac{1}{|N|} \cdot \Big(\sum_{x \in (X \triangle Y)} x^2 \Big) - \Big(\frac{|X \triangle Y|}{|N|} \Big)^2$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$sim_{Shape} = \frac{1}{n} \Big(\sum_{x \in b} x^2 + \sum_{x \in c} x^2 \Big) - \Big(\frac{b+c}{n} \Big)^2$$

In [Cor17], the formula is instead $\frac{n(b+c)-(b-c)^2}{n^2}$, but it is clear from [Pen52] that this should not be an assymmetric value with respect to the ordering of the two sets, among other errors in this formula. Meanwhile, [DD16] gives the formula $\sqrt{\sum((x_i - \bar{x}) - (y_i - \bar{y}))^2}$.

New in version 0.4.0.

Initialize Shape instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in _TokenDistance for details.

- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the soft and fuzzy variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the fuzzy variant.

New in version 0.4.0.

dist (*src*, *tar*)

Return the Penrose's shape difference of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Shape ifference

Return type float

Examples

```
>>> cmp = Shape()
>>> cmp.sim('cat', 'hat')
0.994923990004165
>>> cmp.sim('Niall', 'Neil')
0.9911511479591837
>>> cmp.sim('aluminum', 'Catalan')
0.9787090754188811
>>> cmp.sim('ATCG', 'TAGC')
0.9874075905872554
```

New in version 0.4.0.

class abydos.distance.**Size**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Penrose's size difference.

For two sets X and Y and a population N, the Penrose's size difference [Pen52] is

$$sim_{Size}(X, Y) = \frac{(|X \triangle Y|)^2}{|N|^2}$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$sim_{Size} = \frac{(b+c)^2}{n^2}$$

In [Cor17], the formula is instead $\frac{(b-c)^2}{n^2}$, but it is clear from [Pen52] that this should not be an assymmetric value with respect two the ordering of the two sets. Meanwhile, [DD16] gives a formula that is equivalent to $\sqrt{n} \cdot (b+c)$.

New in version 0.4.0.

Initialize Size instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in `_TokenDistance` for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in `_TokenDistance` for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

dist(*src*, *tar*)

Return the Penrose's size difference of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Size difference

Return type float

Examples

```
>>> cmp = Size()
>>> cmp.sim('cat', 'hat')
0.9999739691795085
>>> cmp.sim('Niall', 'Neil')
0.9999202806122449
>>> cmp.sim('aluminum', 'Catalan')
0.9996348736257049
>>> cmp.sim('ATCG', 'TAGC')
0.9998373073719283
```

New in version 0.4.0.

class abydos.distance.**SokalMichener**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Sokal & Michener similarity.

For two sets X and Y and a population N, the Sokal & Michener's simple matching coefficient [SM58], equivalent to the Rand index [Ran71] is

$$sim_{SokalMichener}(X, Y) = \frac{|X \cap Y| + |(N \setminus X) \setminus Y|}{|N|}$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$sim_{SokalMichener} = \frac{a + d}{n}$$

Notes

The associated distance metric is the mean Manhattan distance and 4 times the value of the variance dissimilarity of [Cor17].

In terms of a confusion matrix, this is equivalent to accuracy `ConfusionTable.accuracy()`.

New in version 0.4.0.

Initialize SokalMichener instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in `_TokenDistance` for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in `_TokenDistance` for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

sim(*src*, *tar*)

Return the Sokal & Michener similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Sokal & Michener similarity

Return type float

Examples

```
>>> cmp = SokalMichener()
>>> cmp.sim('cat', 'hat')
0.9948979591836735
>>> cmp.sim('Niall', 'Neil')
0.9910714285714286
>>> cmp.sim('aluminum', 'Catalan')
0.9808917197452229
>>> cmp.sim('ATCG', 'TAGC')
0.9872448979591837
```

New in version 0.4.0.

class abydos.distance.**SokalSneathI**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Sokal & Sneath I similarity.

For two sets X and Y and a population N, Sokal & Sneath I similarity [SS63] is

$$sim_{SokalSneathI}(X, Y) = \frac{2(|X \cap Y| + |(N \setminus X) \setminus Y|)}{|X \cap Y| + |(N \setminus X) \setminus Y| + |N|}$$

This is the first of five "Unnamed coefficients" presented in [SS63]. It corresponds to the "Matched pairs carry twice the weight of unmatched pairs in the Denominator" with "Negative Matches in Numerator Included". "Negative Matches in Numerator Excluded" corresponds to the Sørensen–Dice coefficient, *Dice*.

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$sim_{SokalSneathI} = \frac{2(a+d)}{a+d+n}$$

New in version 0.4.0.

Initialize SokalSneathI instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in _TokenDistance for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

sim(*src*, *tar*)

Return the Sokal & Sneath I similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Sokal & Sneath I similarity

Return type float

Examples

```
>>> cmp = SokalSneathI()
>>> cmp.sim('cat', 'hat')
0.9974424552429667
>>> cmp.sim('Niall', 'Neil')
0.9955156950672646
>>> cmp.sim('aluminum', 'Catalan')
0.9903536977491961
>>> cmp.sim('ATCG', 'TAGC')
0.993581514762516
```

New in version 0.4.0.

class abydos.distance.**SokalSneathII**(*tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Sokal & Sneath II similarity.

For two sets X and Y, Sokal & Sneath II similarity [SS63] is

$$sim_{SokalSneathII}(X, Y) = \frac{|X \cap Y|}{|X \cap Y| + 2|X \triangle Y|}$$

This is the second of five "Unnamed coefficients" presented in [SS63]. It corresponds to the "Unmatched pairs carry twice the weight of matched pairs in the Denominator" with "Negative Matches in Numerator Excluded". "Negative Matches in Numerator Included" corresponds to the Rogers & Tanimoto similarity, *RogersTanimoto*.

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$sim_{SokalSneathII} = \frac{a}{a + 2(b + c)}$$

New in version 0.4.0.

Initialize SokalSneathII instance.

Parameters

- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

sim(*src*, *tar*)

Return the Sokal & Sneath II similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Sokal & Sneath II similarity

Return type float

Examples

```
>>> cmp = SokalSneathII()
>>> cmp.sim('cat', 'hat')
0.2
>>> cmp.sim('Niall', 'Neil')
0.125
>>> cmp.sim('aluminum', 'Catalan')
0.03225806451612903
>>> cmp.sim('ATCG', 'TAGC')
0.0
```

New in version 0.4.0.

class abydos.distance.**SokalSneathIII**(*tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Sokal & Sneath III similarity.

For two sets X and Y and a population N, Sokal & Sneath III similarity [SS63] is

$$sim_{SokalSneathIII}(X, Y) = \frac{|X \cap Y| + |(N \setminus X) \setminus Y|}{|X \triangle Y|}$$

This is the third of five "Unnamed coefficients" presented in [SS63]. It corresponds to the "Unmatched pairs only in the Denominator" with "Negative Matches in Numerator Excluded". "Negative Matches in Numerator Included" corresponds to the Kulczynski I coefficient, *KulczynskiI*.

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$sim_{SokalSneathIII} = \frac{a+d}{b+c}$$

New in version 0.4.0.

Initialize SokalSneathIII instance.

Parameters

- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in `_TokenDistance` for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

dist(**args*, ***kwargs*)
Raise exception when called.

Parameters

- ***args** -- Variable length argument list
- ****kwargs** -- Arbitrary keyword arguments

Raises **NotImplementedError** -- Method disabled for Sokal & Sneath III similarity.

New in version 0.3.6.

sim(**args*, ***kwargs*)
Raise exception when called.

Parameters

- ***args** -- Variable length argument list
- ****kwargs** -- Arbitrary keyword arguments

Raises **NotImplementedError** -- Method disabled for Sokal & Sneath III similarity.

New in version 0.3.6.

sim_score(*src*, *tar*)
Return the Sokal & Sneath III similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Sokal & Sneath III similarity

Return type float

Examples

```
>>> cmp = SokalSneathIII()
>>> cmp.sim_score('cat', 'hat')
195.0
>>> cmp.sim_score('Niall', 'Neil')
111.0
>>> cmp.sim_score('aluminum', 'Catalan')
51.333333333333336
>>> cmp.sim_score('ATCG', 'TAGC')
77.4
```

New in version 0.4.0.

class abydos.distance.**SokalSneathIV**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Sokal & Sneath IV similarity.

For two sets X and Y and a population N, Sokal & Sneath IV similarity [SS63] is

$$sim_{SokalSneathIV}(X, Y) = \frac{1}{4}\left(\frac{|X \cap Y|}{|X|} + \frac{|X \cap Y|}{|Y|} + \frac{|(N \setminus X) \setminus Y|}{|N \setminus Y|} + \frac{|(N \setminus X) \setminus Y|}{|N \setminus X|}\right)$$

This is the fourth of five "Unnamed coefficients" presented in [SS63]. It corresponds to the first "Marginal totals in the Denominator" with "Negative Matches in Numerator Included". "Negative Matches in Numerator Excluded" corresponds to the Kulczynski II similarity, *KulczynskiII*. This is also Rogot & Goldberg's "adjusted agreement" A_1 [RG66].

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$sim_{SokalSneathIV} = \frac{1}{4}\left(\frac{a}{a+b} + \frac{a}{a+c} + \frac{d}{b+d} + \frac{d}{c+d}\right)$$

New in version 0.4.0.

Initialize SokalSneathIV instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in _TokenDistance for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package

- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the soft and fuzzy variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the fuzzy variant.

New in version 0.4.0.

sim(*src*, *tar*)

Return the Sokal & Sneath IV similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Sokal & Sneath IV similarity

Return type float

Examples

```
>>> cmp = SokalSneathIV()
>>> cmp.sim('cat', 'hat')
0.7487179487179487
>>> cmp.sim('Niall', 'Neil')
0.6810856260030602
>>> cmp.sim('aluminum', 'Catalan')
0.5541986205645999
>>> cmp.sim('ATCG', 'TAGC')
0.496790757381258
```

New in version 0.4.0.

class abydos.distance.**SokalSneathV**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Sokal & Sneath V similarity.

For two sets X and Y and a population N, Sokal & Sneath V similarity [SS63] is

$$sim_{SokalSneathV}(X, Y) = \frac{|X \cap Y| \cdot |(N \setminus X) \setminus Y|}{\sqrt{|X| \cdot |Y| \cdot |N \setminus Y| \cdot |N \setminus X|}}$$

This is the fifth of five "Unnamed coefficients" presented in [SS63]. It corresponds to the second "Marginal totals in the Denominator" with "Negative Matches in Numerator Included", also sometimes referred to as Ochiai II similarity. "Negative Matches in Numerator Excluded" corresponds to the Cosine similarity, *Cosine*.

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$sim_{SokalSneathV} = \frac{ad}{\sqrt{(a+b)(a+c)(b+d)(c+d)}}$$

New in version 0.4.0.

Initialize SokalSneathV instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in _TokenDistance for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the soft and fuzzy variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the fuzzy variant.

New in version 0.4.0.

sim(*src*, *tar*)

Return the Sokal & Sneath V similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Sokal & Sneath V similarity

Return type float

Examples

```
>>> cmp = SokalSneathV()
>>> cmp.sim('cat', 'hat')
0.4987179487179487
>>> cmp.sim('Niall', 'Neil')
0.3635068033537323
>>> cmp.sim('aluminum', 'Catalan')
0.11671286273067434
>>> cmp.sim('ATCG', 'TAGC')
0.0
```

New in version 0.4.0.

class abydos.distance.**Sorgenfrei**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Sorgenfrei similarity.

For two sets X and Y, Sorgenfrei similarity [Sor58] is

$$sim_{Sorgenfrei}(X, Y) = \frac{|X \cap Y|^2}{|X| \cdot |Y|}$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$sim_{Sorgenfrei} = \frac{a^2}{(a+b)(a+c)}$$

New in version 0.4.0.

Initialize Sorgenfrei instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in _TokenDistance for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the soft and fuzzy variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the fuzzy variant.

New in version 0.4.0.

sim(*src*, *tar*)

Return the Sorgenfrei similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Sorgenfrei similarity

Return type float

Examples

```
>>> cmp = Sorgenfrei()
>>> cmp.sim('cat', 'hat')
0.25
>>> cmp.sim('Niall', 'Neil')
0.13333333333333333
>>> cmp.sim('aluminum', 'Catalan')
0.013888888888888888
>>> cmp.sim('ATCG', 'TAGC')
0.0
```

New in version 0.4.0.

class abydos.distance.**Steffensen**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, *normalizer='proportional'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Steffensen similarity.

For two sets X and Y and a population N, Steffensen similarity ψ^2 [Ste34] is

$$sim_{Steffensen_\psi}(X, Y) = \psi^2 = \sum_{i \in X} \sum_{j \in Y} p_{ij} \phi_{ij}^2$$

$$\phi_{ij}^2 = \frac{(p_{ij} - p_{i*} p_{*i})^2}{p_{i*}(1 - p_{i*}) p_{*j}(1 - p_{*j})}$$

Where each value p_{ij} is drawn from the 2x2 contingency table:

	$x \in$ tar	$x \notin$ tar	
$x \in$ src	$p_{11} = a$	$p_{10} = b$	$p_{1*} = a + b$
$x \notin$ src	$p_{01} = c$	$p_{00} = d$	$p_{0*} = c + d$
	$p_{*1} = a + c$	$p_{*0} = b + d$	1

New in version 0.4.0.

Initialize Steffensen instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in _TokenDistance for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.
- **normalizer** (*str*) -- Specifies the normalization type. See normalizer description in _TokenDistance for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.

- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

sim(*src*, *tar*)
Return the Steffensen similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Steffensen similarity

Return type float

Examples

```
>>> cmp = Steffensen()
>>> cmp.sim('cat', 'hat')
0.24744247205786737
>>> cmp.sim('Niall', 'Neil')
0.1300991207720166
>>> cmp.sim('aluminum', 'Catalan')
0.011710186806836031
>>> cmp.sim('ATCG', 'TAGC')
4.1196952743871653e-05
```

New in version 0.4.0.

class abydos.distance.**Stiles**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)
Bases: abydos.distance._token_distance._TokenDistance

Stiles similarity.

For two sets X and Y and a population N, Stiles similarity [Sti61] is

$$sim_{Stiles}(X,Y) = log_{10}\frac{|N|\Big(||X \cap Y| \cdot |N| - |X \setminus Y| \cdot |Y \setminus X|| - \frac{|N|}{2}\Big)^2}{|X \setminus Y| \cdot |Y \setminus X| \cdot (|N| - |X \setminus Y|) \cdot (|N| - |Y \setminus X|)}$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$sim_{Stiles} = log_{10}\frac{n(|an - bc| - \frac{1}{2}n)^2}{bc(n-b)(n-c)}$$

New in version 0.4.0.

Initialize Stiles instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in `_TokenDistance` for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in `_TokenDistance` for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

corr(*src*, *tar*)

Return the Stiles correlation of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Stiles correlation

Return type float

Examples

```
>>> cmp = Stiles()
>>> cmp.corr('cat', 'hat')
0.14701542182970487
>>> cmp.corr('Niall', 'Neil')
0.11767566062554877
>>> cmp.corr('aluminum', 'Catalan')
0.022355640924908403
>>> cmp.corr('ATCG', 'TAGC')
-0.046296656196428934
```

New in version 0.4.0.

sim(*src*, *tar*)

Return the normalized Stiles similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Normalized Stiles similarity

Return type float

Examples

```
>>> cmp = Stiles()
>>> cmp.sim('cat', 'hat')
0.5735077109148524
>>> cmp.sim('Niall', 'Neil')
0.5588378303127743
>>> cmp.sim('aluminum', 'Catalan')
0.5111778204624542
>>> cmp.sim('ATCG', 'TAGC')
0.4768516719017855
```

New in version 0.4.0.

sim_score(*src*, *tar*)

Return the Stiles similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Stiles similarity

Return type float

Examples

```
>>> cmp = Stiles()
>>> cmp.sim_score('cat', 'hat')
2.6436977886009236
>>> cmp.sim_score('Niall', 'Neil')
2.1622951406967723
>>> cmp.sim_score('aluminum', 'Catalan')
0.41925115106844024
>>> cmp.sim_score('ATCG', 'TAGC')
-0.8426334527850912
```

New in version 0.4.0.

class abydos.distance.**StuartTau**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Stuart's Tau correlation.

For two sets X and Y and a population N, Stuart's Tau-C correlation [Stu53] is

$$corr_{Stuart_{\tau_c}}(X, Y) = \frac{4 \cdot (|X \cap Y| + |(N \setminus X) \setminus Y| - |X \triangle Y|)}{|N|^2}$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$corr_{Stuart_{\tau_c}} = \frac{4 \cdot ((a+d) - (b+c))}{n^2}$$

New in version 0.4.0.

Initialize StuartTau instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in `_TokenDistance` for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in `_TokenDistance` for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

corr(*src*, *tar*)

Return the Stuart's Tau correlation of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Stuart's Tau correlation

Return type float

Examples

```
>>> cmp = StuartTau()
>>> cmp.corr('cat', 'hat')
0.005049979175343606
>>> cmp.corr('Niall', 'Neil')
0.005010932944606414
>>> cmp.corr('aluminum', 'Catalan')
0.004900807334983164
>>> cmp.corr('ATCG', 'TAGC')
0.0049718867138692216
```

New in version 0.4.0.

sim(*src*, *tar*)

Return the Stuart's Tau similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison

- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Stuart's Tau similarity

Return type float

Examples

```
>>> cmp = StuartTau()
>>> cmp.sim('cat', 'hat')
0.5025249895876718
>>> cmp.sim('Niall', 'Neil')
0.5025054664723032
>>> cmp.sim('aluminum', 'Catalan')
0.5024504036674916
>>> cmp.sim('ATCG', 'TAGC')
0.5024859433569346
```

New in version 0.4.0.

class abydos.distance.**Tarantula**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Tarantula similarity.

For two sets X and Y and a population N, Tarantula similarity [JH05] is

$$sim_{Tarantula}(X, Y) = \frac{\frac{|X \cap Y|}{|X \cap Y| + |X \setminus Y|}}{\frac{|X \cap Y|}{|X \cap Y| + |X \setminus Y|} + \frac{|Y \setminus X|}{|Y \setminus X| + |(N \setminus X) \setminus Y|}}$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$sim_{Tarantula} = \frac{\frac{a}{a+b}}{\frac{a}{a+b} + \frac{c}{c+d}}$$

New in version 0.4.0.

Initialize Tarantula instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in _TokenDistance for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.

- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

sim(*src*, *tar*)

Return the Tarantula similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Tarantula similarity

Return type float

Examples

```
>>> cmp = Tarantula()
>>> cmp.sim('cat', 'hat')
0.9948979591836735
>>> cmp.sim('Niall', 'Neil')
0.98856416772554
>>> cmp.sim('aluminum', 'Catalan')
0.9249106078665077
>>> cmp.sim('ATCG', 'TAGC')
0.0
```

New in version 0.4.0.

class abydos.distance.**Tarwid**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: `abydos.distance._token_distance._TokenDistance`

Tarwid correlation.

For two sets X and Y and a population N, the Tarwid correlation [Tar60] is

$$corr_{Tarwid}(X, Y) = \frac{|N| \cdot |X \cap Y| - |X| \cdot |Y|}{|N| \cdot |X \cap Y| + |X| \cdot |Y|}$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$corr_{Tarwid} = \frac{na - (a+b)(a+c)}{na + (a+b)(a+c)}$$

New in version 0.4.0.

Initialize Tarwid instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in `_TokenDistance` for details.

- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in `_TokenDistance` for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

corr(*src*, *tar*)

Return the Tarwid correlation of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Tarwid correlation

Return type float

Examples

```
>>> cmp = Tarwid()
>>> cmp.corr('cat', 'hat')
0.9797979797979798
>>> cmp.corr('Niall', 'Neil')
0.9624530663329162
>>> cmp.corr('aluminum', 'Catalan')
0.8319719953325554
>>> cmp.corr('ATCG', 'TAGC')
-1.0
```

New in version 0.4.0.

sim(*src*, *tar*)

Return the Tarwid similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Tarwid similarity

Return type float

Examples

```
>>> cmp = Tarwid()
>>> cmp.sim('cat', 'hat')
0.9898989898989898
>>> cmp.sim('Niall', 'Neil')
0.981226533166458
>>> cmp.sim('aluminum', 'Catalan')
0.9159859976662776
>>> cmp.sim('ATCG', 'TAGC')
0.0
```

New in version 0.4.0.

class abydos.distance.**Tetrachoric**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Tetrachoric correlation coefficient.

For two sets X and Y and a population N, the Tetrachoric correlation coefficient [Pea00] is

$$corr_{Tetrachoric}(X, Y) = \cos\left(\frac{\pi\sqrt{|X \setminus Y| \cdot |Y \setminus X|}}{\sqrt{|X \cap Y| \cdot |(N \setminus X) \setminus Y|} + \sqrt{|X \setminus Y| \cdot |Y \setminus X|}}\right)$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$corr_{Tetrachoric} = \cos\frac{\pi\sqrt{bc}}{\sqrt{ad} + \sqrt{bc}}$$

New in version 0.4.0.

Initialize Tetrachoric instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in _TokenDistance for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the soft and fuzzy variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the fuzzy variant.

New in version 0.4.0.

corr(*src*, *tar*)

Return the Tetrachoric correlation coefficient of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Tetrachoric correlation coefficient

Return type float

Examples

```
>>> cmp = Tetrachoric()
>>> cmp.corr('cat', 'hat')
0.9885309061036239
>>> cmp.corr('Niall', 'Neil')
0.9678978997263907
>>> cmp.corr('aluminum', 'Catalan')
0.7853000893691571
>>> cmp.corr('ATCG', 'TAGC')
-1.0
```

New in version 0.4.0.

sim(*src*, *tar*)

Return the Tetrachoric correlation coefficient of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Tetrachoric correlation coefficient

Return type float

Examples

```
>>> cmp = Tetrachoric()
>>> cmp.sim('cat', 'hat')
0.994265453051812
>>> cmp.sim('Niall', 'Neil')
0.9839489498631954
>>> cmp.sim('aluminum', 'Catalan')
0.8926500446845785
>>> cmp.sim('ATCG', 'TAGC')
0.0
```

New in version 0.4.0.

class abydos.distance.**TullossR**(*tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Tulloss' R similarity.

For two sets X and Y and a population N, Tulloss' R similarity [Tul97] is

$$sim_{Tulloss_R}(X, Y) = \frac{log(1 + \frac{|X \cap Y|}{|X|}) \cdot log(1 + \frac{|X \cap Y|}{|Y|})}{log^2(2)}$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$sim_{Tulloss_R} = \frac{log(1 + \frac{a}{a+b}) \cdot log(1 + \frac{a}{a+c})}{log^2(2)}$$

New in version 0.4.0.

Initialize TullossR instance.

Parameters

- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in `_TokenDistance` for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

sim(*src*, *tar*)

Return Tulloss' R similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Tulloss' R similarity

Return type float

Examples

```
>>> cmp = TullossR()
>>> cmp.sim('cat', 'hat')
0.34218112724994865
>>> cmp.sim('Niall', 'Neil')
0.2014703364316006
>>> cmp.sim('aluminum', 'Catalan')
0.025829125872886074
>>> cmp.sim('ATCG', 'TAGC')
0.0
```

New in version 0.4.0.

class abydos.distance.**TullossS**(*tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Tulloss' S similarity.

For two sets X and Y and a population N, Tulloss' S similarity [Tul97] is

$$sim_{Tulloss_S}(X, Y) = \frac{1}{\sqrt{log_2(2 + \frac{min(|X \setminus Y|, |Y \setminus X|)}{|X \cap Y| + 1})}}$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$sim_{Tulloss_S} = \frac{1}{\sqrt{log_2(2 + \frac{min(b,c)}{a+1})}}$$

New in version 0.4.0.

Initialize TullossS instance.

Parameters

- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the soft and fuzzy variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the fuzzy variant.

New in version 0.4.0.

sim(*src*, *tar*)

Return Tulloss' S similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Tulloss' S similarity

Return type float

Examples

```
>>> cmp = TullossS()
>>> cmp.sim('cat', 'hat')
0.8406515643305636
>>> cmp.sim('Niall', 'Neil')
0.7943108670863427
>>> cmp.sim('aluminum', 'Catalan')
0.6376503816669968
>>> cmp.sim('ATCG', 'TAGC')
0.5968309535438173
```

New in version 0.4.0.

class abydos.distance.**TullossT**(*tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Tulloss' T similarity.

For two sets X and Y and a population N, Tulloss' T similarity [Tul97] is

$$sim_{Tulloss_T}(X, Y) = \sqrt{sim_{Tulloss_U}(X, Y) \cdot sim_{Tulloss_S}(X, Y) \cdot sim_{Tulloss_R}(X, Y)}$$
$$= \sqrt{log_2(1 + \frac{min(|X \setminus Y|, |Y \setminus X|) + |X \cap Y|}{max(|X \setminus Y|, |Y \setminus X|) + |X \cap Y|}) \cdot \frac{1}{\sqrt{log_2(2 + \frac{min(|X \setminus Y|, |Y \setminus X|)}{|X \cap Y| + 1})}} \cdot \frac{log(1 + \frac{|X \cap Y|}{|X|}) \cdot log(1 + \frac{|X \cap Y|}{|Y|})}{log^2(2)}}$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$sim_{Tulloss_T} = \sqrt{log_2\Big(1 + \frac{min(b, c) + a}{max(b, c) + a}\Big) \cdot \frac{1}{\sqrt{log_2(2 + \frac{min(b,c)}{a+1})}} \cdot \frac{log(1 + \frac{a}{a+b}) \cdot log(1 + \frac{a}{a+c})}{log^2(2)}}$$

New in version 0.4.0.

Initialize TullossT instance.

Parameters

- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the soft and fuzzy variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the fuzzy variant.

New in version 0.4.0.

sim(*src*, *tar*)
Return Tulloss' T similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Tulloss' T similarity

Return type float

Examples

```
>>> cmp = TullossT()
>>> cmp.sim('cat', 'hat')
0.5363348766461724
>>> cmp.sim('Niall', 'Neil')
0.3740873705689327
>>> cmp.sim('aluminum', 'Catalan')
0.1229300783095269
>>> cmp.sim('ATCG', 'TAGC')
0.0
```

New in version 0.4.0.

class abydos.distance.**TullossU**(*tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)
Bases: abydos.distance._token_distance._TokenDistance

Tulloss' U similarity.

For two sets X and Y, Tulloss' U similarity [Tul97] is

$$sim_{Tullossu}(X, Y) = log_2\Big(1 + \frac{min(|X \setminus Y|, |Y \setminus X|) + |X \cap Y|}{max(|X \setminus Y|, |Y \setminus X|) + |X \cap Y|}\Big)$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$sim_{Tullossu} = log_2\Big(1 + \frac{min(b, c) + a}{max(b, c) + a}\Big)$$

New in version 0.4.0.

Initialize TullossU instance.

Parameters

- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

sim(*src*, *tar*)

Return Tulloss' U similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Tulloss' U similarity

Return type float

Examples

```
>>> cmp = TullossU()
>>> cmp.sim('cat', 'hat')
1.0
>>> cmp.sim('Niall', 'Neil')
0.8744691179161412
>>> cmp.sim('aluminum', 'Catalan')
0.917537839808027
>>> cmp.sim('ATCG', 'TAGC')
1.0
```

New in version 0.4.0.

class abydos.distance.**Tversky**(*alpha=1.0*, *beta=1.0*, *bias=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Tversky index.

The Tversky index [Tve77] is defined as: For two sets X and Y:

$$sim_{Tversky}(X, Y) = \frac{|X \cap Y|}{|X \cap Y| + \alpha|X - Y| + \beta|Y - X|}$$

$\alpha = \beta = 1$ is equivalent to the Jaccard & Tanimoto similarity coefficients.

$\alpha = \beta = 0.5$ is equivalent to the Sørensen-Dice similarity coefficient [Dic45][Sorensen48].

Unequal and will tend to emphasize one or the other set's contributions:

- $\alpha > \beta$ emphasizes the contributions of X over Y
- $\alpha < \beta$ emphasizes the contributions of Y over X)

Parameter values' relation to 1 emphasizes different types of contributions:

- α and $\beta > 1$ emphsize unique contributions over the intersection

- α and $\beta < 1$ emphsize the intersection over unique contributions

The symmetric variant is defined in [JBG13]. This is activated by specifying a bias parameter.

New in version 0.3.6.

Initialize Tversky instance.

Parameters

- **alpha** (*float*) -- Tversky index parameter as described above
- **beta** (*float*) -- Tversky index parameter as described above
- **bias** (*float*) -- The symmetric Tversky index bias parameter
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in `_TokenDistance` for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

sim(*src*, *tar*)

Return the Tversky index of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Tversky similarity

Return type float

Raises **ValueError** -- Unsupported weight assignment; alpha and beta must be greater than or equal to 0.

Examples

```
>>> cmp = Tversky()
>>> cmp.sim('cat', 'hat')
0.3333333333333333
>>> cmp.sim('Niall', 'Neil')
0.2222222222222222
>>> cmp.sim('aluminum', 'Catalan')
0.0625
>>> cmp.sim('ATCG', 'TAGC')
0.0
```

New in version 0.1.0.

Changed in version 0.3.6: Encapsulated in class

abydos.distance.**dist_tversky**(*src*, *tar*, *qval=2*, *alpha=1.0*, *beta=1.0*, *bias=None*)

Return the Tversky distance between two strings.

This is a wrapper for `Tversky.dist()`.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison
- **qval** (*int*) -- The length of each q-gram
- **alpha** (*float*) -- Tversky index parameter as described above
- **beta** (*float*) -- Tversky index parameter as described above
- **bias** (*float*) -- The symmetric Tversky index bias parameter

Returns Tversky distance

Return type float

Examples

```
>>> dist_tversky('cat', 'hat')
0.6666666666666667
>>> dist_tversky('Niall', 'Neil')
0.7777777777777778
>>> dist_tversky('aluminum', 'Catalan')
0.9375
>>> dist_tversky('ATCG', 'TAGC')
1.0
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the Tversky.dist method instead.

abydos.distance.**sim_tversky**(*src*, *tar*, *qval=2*, *alpha=1.0*, *beta=1.0*, *bias=None*)

Return the Tversky index of two strings.

This is a wrapper for *Tversky.sim()*.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison
- **qval** (*int*) -- The length of each q-gram
- **alpha** (*float*) -- Tversky index parameter as described above
- **beta** (*float*) -- Tversky index parameter as described above
- **bias** (*float*) -- The symmetric Tversky index bias parameter

Returns Tversky similarity

Return type float

Examples

```
>>> sim_tversky('cat', 'hat')
0.3333333333333333
>>> sim_tversky('Niall', 'Neil')
0.2222222222222222
>>> sim_tversky('aluminum', 'Catalan')
0.0625
>>> sim_tversky('ATCG', 'TAGC')
0.0
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the Tversky.sim method instead.

class abydos.distance.**UnigramSubtuple**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Unigram subtuple similarity.

For two sets X and Y and a population N, unigram subtuple similarity [Pec10] is

$$sim_{unigram\ subtuple}(X, Y) = log(\frac{|X \cap Y| \cdot |(N \setminus X) \setminus Y|}{|X \setminus Y| \cdot |Y \setminus Y|}) - 3.29 \cdot \sqrt{\frac{1}{|X \cap Y|} + \frac{1}{|X \setminus Y|} + \frac{1}{|Y \setminus X|} + \frac{1}{|(N \setminus X) \setminus Y|}}$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$sim_{unigram\ subtuple} = log(\frac{ad}{bc}) - 3.29 \cdot \sqrt{\frac{1}{a} + \frac{1}{b} + \frac{1}{c} + \frac{1}{d}}$$

New in version 0.4.0.

Initialize UnigramSubtuple instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in _TokenDistance for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the soft and fuzzy variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the fuzzy variant.

New in version 0.4.0.

sim(*src*, *tar*)

Return the unigram subtuple similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Unigram subtuple similarity

Return type float

Examples

```
>>> cmp = UnigramSubtuple()
>>> cmp.sim('cat', 'hat')
0.6215275850074894
>>> cmp.sim('Niall', 'Neil')
0.39805896767519555
>>> cmp.sim('aluminum', 'Catalan')
0.0
>>> cmp.sim('ATCG', 'TAGC')
0.0
```

New in version 0.4.0.

sim_score(*src*, *tar*)

Return the unigram subtuple similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Unigram subtuple similarity

Return type float

Examples

```
>>> cmp = UnigramSubtuple()
>>> cmp.sim_score('cat', 'hat')
1.9324426894059226
>>> cmp.sim_score('Niall', 'Neil')
1.4347242883606355
>>> cmp.sim_score('aluminum', 'Catalan')
-1.0866724701675263
>>> cmp.sim_score('ATCG', 'TAGC')
-0.461880260111438
```

New in version 0.4.0.

class abydos.distance.**UnknownA**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Unknown A correlation.

For two sets X and Y and a population N, Unknown A correlation is sometimes attributed to [Pei84]. It differs from *Peirce* in that the numerator is the product of the opposite pair of marginals:

$$corr_{UnknownA}(X, Y) = \frac{|X \cap Y| \cdot |(N \setminus X) \setminus Y| - |X \setminus Y| \cdot |Y \setminus Y|}{|Y| \cdot |N \setminus Y|}$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$corr_{UnknownA} = \frac{ad - bc}{(a + c)(b + d)}$$

New in version 0.4.0.

Initialize UnknownA instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in `_TokenDistance` for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in `_TokenDistance` for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

corr(*src*, *tar*)

Return the Unknown A correlation of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Unknown A correlation

Return type float

Examples

```
>>> cmp = UnknownA()
>>> cmp.corr('cat', 'hat')
0.49743589743589745
>>> cmp.corr('Niall', 'Neil')
0.39486521181001283
>>> cmp.corr('aluminum', 'Catalan')
0.1147039897039897
>>> cmp.corr('ATCG', 'TAGC')
-0.006418485237483954
```

New in version 0.4.0.

sim(*src*, *tar*)

Return the Unknown A similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Unknown A similarity

Return type float

Examples

```
>>> cmp = UnknownA()
>>> cmp.sim('cat', 'hat')
0.7487179487179487
>>> cmp.sim('Niall', 'Neil')
0.6974326059050064
>>> cmp.sim('aluminum', 'Catalan')
0.5573519948519948
>>> cmp.sim('ATCG', 'TAGC')
0.496790757381258
```

New in version 0.4.0.

class abydos.distance.**UnknownB**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Unknown B similarity.

For two sets X and Y and a population N, Unknown B similarity, which [Mor12] attributes to [Doo84] but could not be located in that source, is

$$sim_{UnknownB}(X, Y) = \frac{(|X \cap Y| \cdot |(N \setminus X) \setminus Y| - |X \setminus Y| \cdot |Y \setminus X|)^2}{|X| \cdot |Y| \cdot |N \setminus X| \cdot |N \setminus Y|}$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$sim_{UnknownB} = \frac{(ad - bc)^2}{(a + b)(a + c)(b + d)(c + d)}$$

New in version 0.4.0.

Initialize UnknownB instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in `_TokenDistance` for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in `_TokenDistance` for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

sim(*src*, *tar*)

Return the Unknown B similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Unknown B similarity

Return type float

Examples

```
>>> cmp = UnknownB()
>>> cmp.sim('cat', 'hat')
0.24744247205785666
>>> cmp.sim('Niall', 'Neil')
0.13009912077202224
>>> cmp.sim('aluminum', 'Catalan')
0.011710186806836291
>>> cmp.sim('ATCG', 'TAGC')
4.1196952743799446e-05
```

New in version 0.4.0.

class abydos.distance.**UnknownC**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: `abydos.distance._token_distance._TokenDistance`

Unknown C similarity.

For two sets X and Y and a population N, Unknown C similarity, which [Mor12] attributes to [Gow71] but could not be located in that source, is

$$sim_{UnknownC}(X, Y) = \frac{|X \cap Y| + |(N \setminus X) \setminus Y|}{\sqrt{|X| \cdot |Y| \cdot |N \setminus X| \cdot |N \setminus Y|}}$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$sim_{UnknownC} = \frac{a + d}{\sqrt{(a + b)(a + c)(b + d)(c + d)}}$$

New in version 0.4.0.

Initialize UnknownC instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in `_TokenDistance` for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in `_TokenDistance` for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

sim(*src*, *tar*)

Return the Unknown C similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Unknown C similarity

Return type float

Examples

```
>>> cmp = UnknownC()
>>> cmp.sim('cat', 'hat')
0.25
>>> cmp.sim('Niall', 'Neil')
0.18222244271345164
>>> cmp.sim('aluminum', 'Catalan')
0.11686463498390019
>>> cmp.sim('ATCG', 'TAGC')
0.1987163029525032
```

New in version 0.4.0.

class abydos.distance.**UnknownD**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Unknown D similarity.

For two sets X and Y and a population N, Unknown D similarity, which [Mor12] attributes to [Pei84] but could not be located in that source, is

$$sim_{UnknownD}(X, Y) = \frac{|X \cap Y| \cdot |X \setminus Y| + |X \setminus Y| \cdot |Y \setminus X|}{|X \cap Y| \cdot |X \setminus Y| + 2 \cdot |X \setminus Y| \cdot |Y \setminus X| + |Y \setminus X| + |(N \setminus X) \setminus Y|}$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$sim_{UnknownD} = \frac{ab + bc}{ab + 2bc + cd}$$

New in version 0.4.0.

Initialize UnknownD instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in _TokenDistance for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the soft and fuzzy variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the fuzzy variant.

New in version 0.4.0.

sim(*src*, *tar*)

Return the Unknown D similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Unknown D similarity

Return type float

Examples

```
>>> cmp = UnknownD()
>>> cmp.sim('cat', 'hat')
0.00510204081632653
>>> cmp.sim('Niall', 'Neil')
0.00848536274925753
>>> cmp.sim('aluminum', 'Catalan')
0.011630019989096857
>>> cmp.sim('ATCG', 'TAGC')
0.006377551020408163
```

New in version 0.4.0.

class abydos.distance.**UnknownE**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Unknown E correlation.

For two sets X and Y and a population N, Unknown E correlation, which [Mor12] attributes to [GK54] but could not be located in that source, is

$$corr_{UnknownE}(X, Y) = \frac{2 \cdot min(|X \cap Y|, |(N \setminus X) \setminus Y|) - |X \setminus Y| - |Y \setminus X|}{2 \cdot min(|X \cap Y|, |(N \setminus X) \setminus Y|) + |X \setminus Y| + |Y \setminus X|}$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$corr_{UnknownE} = \frac{2 \cdot min(a, d) - b - c}{2 \cdot min(a, d) + b + c}$$

New in version 0.4.0.

Initialize UnknownE instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in _TokenDistance for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package

- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in `_TokenDistance` for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

corr(*src*, *tar*)

Return the Unknown E correlation of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Unknown E correlation

Return type float

Examples

```
>>> cmp = UnknownE()
>>> cmp.corr('cat', 'hat')
0.0
>>> cmp.corr('Niall', 'Neil')
-0.2727272727272727
>>> cmp.corr('aluminum', 'Catalan')
-0.7647058823529411
>>> cmp.corr('ATCG', 'TAGC')
-1.0
```

New in version 0.4.0.

sim(*src*, *tar*)

Return the Unknown E similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Unknown E similarity

Return type float

Examples

```
>>> cmp = UnknownE()
>>> cmp.sim('cat', 'hat')
0.5
>>> cmp.sim('Niall', 'Neil')
0.36363636363636365
>>> cmp.sim('aluminum', 'Catalan')
0.11764705882352944
>>> cmp.sim('ATCG', 'TAGC')
0.0
```

New in version 0.4.0.

class abydos.distance.**UnknownF**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Unknown F similarity.

For two sets X and Y and a population N, Unknown F similarity, which [CCT10] attributes to [GW66] but could not be located in that source, is given as

$$sim(X, Y) = log(|X \cap Y|) - log(|N|) - log\left(\frac{|X|}{|N|}\right) - log\left(\frac{|Y|}{|N|}\right)$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$sim = log(a) - log(n) - log\left(\frac{a+b}{n}\right) - log\left(\frac{a+c}{n}\right)$$

This formula is not very normalizable, so the following formula is used instead:

$$sim_{UnknownF}(X, Y) = min\left(1, 1 + log\left(\frac{|X \cap Y|}{|N|}\right) - \frac{1}{2}\left(log\left(\frac{|X|}{|N|}\right) + log\left(\frac{|Y|}{|N|}\right)\right)\right)$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$sim_{UnknownF} = min\left(1, 1 + log\left(\frac{a}{n}\right) - \frac{1}{2}\left(log\left(\frac{a+b}{n}\right) + log\left(\frac{a+c}{n}\right)\right)\right)$$

New in version 0.4.0.

Initialize UnknownF instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in _TokenDistance for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package

- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the soft and fuzzy variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the fuzzy variant.

New in version 0.4.0.

dist(**args*, ***kwargs*)
Raise exception when called.

Parameters

- ***args** -- Variable length argument list
- ****kwargs** -- Arbitrary keyword arguments

Raises **NotImplementedError** -- Method disabled for Unknown F similarity

New in version 0.4.0.

sim(**args*, ***kwargs*)
Raise exception when called.

Parameters

- ***args** -- Variable length argument list
- ****kwargs** -- Arbitrary keyword arguments

Raises **NotImplementedError** -- Method disabled for Unknown F similarity

New in version 0.4.0.

sim_score(*src*, *tar*)
Return the Unknown F similarity between two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Unknown F similarity

Return type float

Examples

```
>>> cmp = UnknownF()
>>> cmp.sim_score('cat', 'hat')
0.3068528194400555
>>> cmp.sim_score('Niall', 'Neil')
-0.007451510271132555
>>> cmp.sim_score('aluminum', 'Catalan')
-1.1383330595080272
>>> cmp.sim_score('ATCG', 'TAGC')
1.0
```

New in version 0.4.0.

class abydos.distance.**UnknownG**(*tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Unknown G similarity.

For two sets X and Y and a population N, Unknown G similarity, which [CCT10] attributes to [Kulczynski27] but could not be located in that source, is

$$sim_{UnknownG}(X, Y) = \frac{\frac{|X \cap Y|}{2} \cdot (|X| + |Y|)}{|X| \cdot |Y|}$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$sim_{UnknownG} = \frac{\frac{a}{2} \cdot (2a + b + c)}{(a + b)(a + c)}$$

New in version 0.4.0.

Initialize UnknownG instance.

Parameters

- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the soft and fuzzy variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the fuzzy variant.

New in version 0.4.0.

sim(*src*, *tar*)

Return the Unknown G similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Unknown G similarity

Return type float

Examples

```
>>> cmp = UnknownG()
>>> cmp.sim('cat', 'hat')
0.5
>>> cmp.sim('Niall', 'Neil')
0.3666666666666664
>>> cmp.sim('aluminum', 'Catalan')
0.11805555555555555
>>> cmp.sim('ATCG', 'TAGC')
0.0
```

New in version 0.4.0.

class abydos.distance.**UnknownH**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Unknown H similarity.

For two sets X and Y and a population N, Unknown H similarity is a variant of Fager-McGowan index of affinity [Fag57][FM63]. It uses minimum rather than maximum in the denominator of the second term, and is sometimes misidentified as the Fager-McGown index of affinity (cf. [Whi82], for example).

$$sim_{UnknownH}(X, Y) = \frac{|X \cap Y|}{\sqrt{|X| \cdot |Y|}} - \frac{1}{2\sqrt{min(|X|, |Y|)}}$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$sim_{UnknownH} = \frac{a}{\sqrt{(a+b)(a+c)}} - \frac{1}{2\sqrt{min(a+b, a+c)}}$$

New in version 0.4.0.

Initialize UnknownH instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in _TokenDistance for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

sim(*src*, *tar*)

Return the normalized Unknown H similarity of two strings.

As this similarity ranges from $(-\inf, 1.0)$, this normalization simply clamps the value to the range (0.0, 1.0).

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Normalized Unknown H similarity

Return type float

Examples

```
>>> cmp = UnknownH()
>>> cmp.sim('cat', 'hat')
0.25
>>> cmp.sim('Niall', 'Neil')
0.14154157392013175
>>> cmp.sim('aluminum', 'Catalan')
0.0
>>> cmp.sim('ATCG', 'TAGC')
0.0
```

New in version 0.4.0.

sim_score(*src*, *tar*)

Return the Unknown H similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Unknown H similarity

Return type float

Examples

```
>>> cmp = UnknownH()
>>> cmp.sim('cat', 'hat')
0.25
>>> cmp.sim('Niall', 'Neil')
0.14154157392013175
>>> cmp.sim('aluminum', 'Catalan')
0.0
>>> cmp.sim('ATCG', 'TAGC')
0.0
```

New in version 0.4.0.

class abydos.distance.**UnknownI**(*tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Unknown I similarity.

For two sets X and Y, the Unknown I similarity is based on Mountford similarity [Mou62] *Mountford*.

$$sim_{UnknownI}(X, Y) = \frac{2(|X \cap Y| + 1)}{2((|X| + 2) \cdot (|Y| + 2)) - (|X| + |Y| + 4) \cdot (|X \cap Y| + 1)}$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$sim_{UnknownI} = \frac{2(a + 1)}{2(a + b + 2)(a + c + 2) - (2a + b + c + 4)(a + 1)}$$

New in version 0.4.0.

Initialize UnknownI instance.

Parameters

- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the soft and fuzzy variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the fuzzy variant.

New in version 0.4.0.

sim(*src*, *tar*)

Return the Unknown I similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Unknown I similarity

Return type float

Examples

```
>>> cmp = UnknownI()
>>> cmp.sim('cat', 'hat')
0.16666666666666666
>>> cmp.sim('Niall', 'Neil')
0.08955223880597014
>>> cmp.sim('aluminum', 'Catalan')
0.02247191011235955
>>> cmp.sim('ATCG', 'TAGC')
0.023809523809523808
```

New in version 0.4.0.

class abydos.distance.**UnknownJ**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Unknown J similarity.

For two sets X and Y and a population N, Unknown J similarity, which [Seq18] attributes to "Kocher & Wang" but could not be located, is

$$sim_{UnknownJ}(X, Y) = |X \cap Y| \cdot \frac{|N|}{|X| \cdot |N \setminus X|}$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$sim_{UnknownJ} = a \cdot \frac{n}{(a+b)(c+d)}$$

New in version 0.4.0.

Initialize UnknownJ instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in _TokenDistance for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

sim(*src*, *tar*)

Return the normalized Unknown J similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Normalized Unknown J similarity

Return type float

Examples

```
>>> cmp = UnknownJ()
>>> cmp.sim('cat', 'hat')
0.5
>>> cmp.sim('Niall', 'Neil')
0.33333333333333337
>>> cmp.sim('aluminum', 'Catalan')
0.11111111111111112
>>> cmp.sim('ATCG', 'TAGC')
0.0
```

New in version 0.4.0.

sim_score(*src*, *tar*)

Return the Unknown J similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Unknown J similarity

Return type float

Examples

```
>>> cmp = UnknownJ()
>>> cmp.sim_score('cat', 'hat')
0.5025641025641026
>>> cmp.sim_score('Niall', 'Neil')
0.33590402742073694
>>> cmp.sim_score('aluminum', 'Catalan')
0.11239977090492555
>>> cmp.sim_score('ATCG', 'TAGC')
0.0
```

New in version 0.4.0.

class abydos.distance.**UnknownK**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Unknown K distance.

For two sets X and Y and a population N, Unknown K distance, which [Seq18] attributes to "Excoffier" but could not be located, is

$$dist_{UnknownK}(X, Y) = |N| \cdot (1 - \frac{|X \cap Y|}{|N|})$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$dist_{UnknownK} = n \cdot (1 - \frac{a}{n})$$

New in version 0.4.0.

Initialize UnknownK instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in _TokenDistance for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the soft and fuzzy variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the fuzzy variant.

New in version 0.4.0.

dist(*src*, *tar*)

Return the normalized Unknown K distance of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Normalized Unknown K distance

Return type float

Examples

```
>>> cmp = UnknownK()
>>> cmp.dist('cat', 'hat')
0.9974489795918368
>>> cmp.dist('Niall', 'Neil')
0.9974489795918368
>>> cmp.dist('aluminum', 'Catalan')
0.9987261146496815
>>> cmp.dist('ATCG', 'TAGC')
1.0
```

New in version 0.4.0.

dist_abs(*src*, *tar*)

Return the Unknown K distance of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Unknown K distance

Return type float

Examples

```
>>> cmp = UnknownK()
>>> cmp.dist_abs('cat', 'hat')
782.0
>>> cmp.dist_abs('Niall', 'Neil')
782.0
>>> cmp.dist_abs('aluminum', 'Catalan')
784.0
>>> cmp.dist_abs('ATCG', 'TAGC')
784.0
```

New in version 0.4.0.

class abydos.distance.**UnknownL**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Unknown L similarity.

For two sets X and Y and a population N, Unknown L similarity, which [Seq18] attributes to "Roux" but could not be located, is

$$sim_{UnknownL}(X, Y) = \frac{|X \cap Y| + |(N \setminus X) \setminus Y|}{min(|X \setminus Y|, |Y \setminus X|) + min(|N| - |X \setminus Y|, |N| - |Y \setminus X|)}$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$sim_{UnknownL} = \frac{a + d}{min(b, c) + min(n - b, n - c)}$$

New in version 0.4.0.

Initialize UnknownL instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in `_TokenDistance` for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in `_TokenDistance` for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

sim(*src*, *tar*)

Return the Unknown L similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Unknown L similarity

Return type float

Examples

```
>>> cmp = UnknownL()
>>> cmp.sim('cat', 'hat')
0.9948979591836735
>>> cmp.sim('Niall', 'Neil')
0.9923371647509579
>>> cmp.sim('aluminum', 'Catalan')
0.9821428571428571
>>> cmp.sim('ATCG', 'TAGC')
0.9872448979591837
```

New in version 0.4.0.

class abydos.distance.**UnknownM**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Unknown M similarity.

For two sets X and Y and a population N, Unknown < similarity, which [Seq18] attributes to "Roux" but could not be located, is

$$sim_{UnknownM}(X, Y) = \frac{|N| - |X \cap Y| \cdot |(N \setminus X) \setminus Y|}{\sqrt{|X| \cdot |N \setminus X| \cdot |Y| \cdot |N \setminus Y|}}$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$sim_{UnknownM} = \frac{n - ad}{\sqrt{(a+b)(c+d)(a+c)(b+d)}}$$

New in version 0.4.0.

Initialize UnknownM instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in _TokenDistance for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the soft and fuzzy variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the fuzzy variant.

New in version 0.4.0.

sim(*src*, *tar*)

Return the normalized Unknown M similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Normalized Unknown M similarity

Return type float

Examples

```
>>> cmp = UnknownM()
>>> cmp.sim('cat', 'hat')
0.6237179487179487
>>> cmp.sim('Niall', 'Neil')
0.5898213585061158
>>> cmp.sim('aluminum', 'Catalan')
0.49878582197419324
>>> cmp.sim('ATCG', 'TAGC')
0.3993581514762516
```

New in version 0.4.0.

sim_score(*src*, *tar*)

Return the Unknown M similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Unknown M similarity

Return type float

Examples

```
>>> cmp = UnknownM()
>>> cmp.sim_score('cat', 'hat')
-0.24743589743589745
>>> cmp.sim_score('Niall', 'Neil')
-0.17964271701223158
>>> cmp.sim_score('aluminum', 'Catalan')
0.0024283560516135103
>>> cmp.sim_score('ATCG', 'TAGC')
0.2012836970474968
```

New in version 0.4.0.

class abydos.distance.**Upholt**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Upholt similarity.

For two sets X and Y and a population N, Upholt similarity, Upholt's S, [Uph77] is

$$sim_{Upholt}(X, Y) = \frac{1}{2}\left(- \frac{2 \cdot |X \cap Y|}{|X| + |Y|} + \sqrt{\left(\frac{2 \cdot |X \cap Y|}{|X| + |Y|}\right)^2 + 8\frac{2 \cdot |X \cap Y|}{|X| + |Y|}}\right)$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$sim_{Upholt}(X, Y) = \frac{1}{2}\left(- \frac{2a}{2a + b + c} + \sqrt{\left(\frac{2a}{2a + b + c}\right)^2 + 8\frac{2a}{2a + b + c}}\right)$$

New in version 0.4.0.

Initialize Upholt instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in `_TokenDistance` for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in `_TokenDistance` for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

sim(*src*, *tar*)

Return the Upholt similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Upholt similarity

Return type float

Examples

```
>>> cmp = Upholt()
>>> cmp.sim('cat', 'hat')
0.7807764064044151
>>> cmp.sim('Niall', 'Neil')
0.6901511860568581
>>> cmp.sim('aluminum', 'Catalan')
0.42980140370106323
>>> cmp.sim('ATCG', 'TAGC')
0.0
```

New in version 0.4.0.

class abydos.distance.**WarrensI**(*tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Warrens I correlation.

For two sets X and Y, Warrens I correlation S_{NS1} [War08] is

$$corr_{WarrensI}(X, Y) = \frac{2|X \cap Y| - |X \setminus Y| - |Y \setminus X|}{2|X \cap Y| + |X \setminus Y| + |Y \setminus X|}$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$corr_{WarrensI} = \frac{2a - b - c}{2a + b + c}$$

New in version 0.4.0.

Initialize WarrensI instance.

Parameters

- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the soft and fuzzy variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the fuzzy variant.

New in version 0.4.0.

corr(*src*, *tar*)

Return the Warrens I correlation of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Warrens I correlation

Return type float

Examples

```
>>> cmp = WarrensI()
>>> cmp.corr('cat', 'hat')
0.0
>>> cmp.corr('Niall', 'Neil')
-0.2727272727272727
>>> cmp.corr('aluminum', 'Catalan')
-0.7647058823529411
>>> cmp.corr('ATCG', 'TAGC')
-1.0
```

New in version 0.4.0.

sim(*src*, *tar*)

Return the Warrens I similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Warrens I similarity

Return type float

Examples

```
>>> cmp = WarrensI()
>>> cmp.sim('cat', 'hat')
0.5
>>> cmp.sim('Niall', 'Neil')
0.36363636363636365
>>> cmp.sim('aluminum', 'Catalan')
0.11764705882352944
>>> cmp.sim('ATCG', 'TAGC')
0.0
```

New in version 0.4.0.

class abydos.distance.**WarrensII**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Warrens II similarity.

For two sets X and Y and a population N, Warrens II similarity S_{NS2} [War08] is

$$sim_{WarrensII}(X, Y) = \frac{2|(N \setminus X) \setminus Y|}{|N \setminus X| + |N \setminus Y|}$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$sim_{WarrensII} = \frac{2d}{b + c + 2d}$$

New in version 0.4.0.

Initialize WarrensII instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in `_TokenDistance` for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in `_TokenDistance` for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

sim(*src*, *tar*)

Return the Warrens II similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Warrens II similarity

Return type float

Examples

```
>>> cmp = WarrensII()
>>> cmp.sim('cat', 'hat')
0.9974358974358974
>>> cmp.sim('Niall', 'Neil')
0.9955041746949261
>>> cmp.sim('aluminum', 'Catalan')
0.9903412749517064
>>> cmp.sim('ATCG', 'TAGC')
0.993581514762516
```

New in version 0.4.0.

class abydos.distance.**WarrensIII**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Warrens III correlation.

For two sets X and Y and a population N, Warrens III correlation S_{NS3} [War08] is

$$corr_{WarrensIII}(X, Y) = \frac{2|(N \setminus X) \setminus Y| - |X \setminus Y| - |Y \setminus X|}{|N \setminus X| + |N \setminus Y|}$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$corr_{WarrensIII} = \frac{2d - b - c}{2d + b + c}$$

New in version 0.4.0.

Initialize WarrensIII instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in _TokenDistance for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the soft and fuzzy variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the fuzzy variant.

New in version 0.4.0.

corr(*src*, *tar*)

Return the Warrens III correlation of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Warrens III correlation

Return type float

Examples

```
>>> cmp = WarrensIII()
>>> cmp.corr('cat', 'hat')
0.9948717948717949
>>> cmp.corr('Niall', 'Neil')
0.9910083493898523
>>> cmp.corr('aluminum', 'Catalan')
0.9806825499034127
>>> cmp.corr('ATCG', 'TAGC')
0.9871630295250321
```

New in version 0.4.0.

sim(*src*, *tar*)

Return the Warrens III similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Warrens III similarity

Return type float

Examples

```
>>> cmp = WarrensIII()
>>> cmp.sim('cat', 'hat')
0.9974358974358974
>>> cmp.sim('Niall', 'Neil')
0.9955041746949261
>>> cmp.sim('aluminum', 'Catalan')
0.9903412749517064
>>> cmp.sim('ATCG', 'TAGC')
0.993581514762516
```

New in version 0.4.0.

class abydos.distance.**WarrensIV**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Warrens IV similarity.

For two sets X and Y and a population N, Warrens IV similarity [War08] is

$$sim_{WarrensIV}(X,Y) = \frac{4|X \cap Y| \cdot |(N \setminus X) \setminus Y|}{4|X \cap Y| \cdot |(N \setminus X) \setminus Y| + (|X \cap Y| + |(N \setminus X) \setminus Y|)(|X \setminus Y| + |Y \setminus X|)}$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$sim_{WarrensIV} = \frac{4ad}{4ad + (a+d)(b+c)}$$

New in version 0.4.0.

Initialize WarrensIV instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in `_TokenDistance` for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in `_TokenDistance` for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

sim(*src*, *tar*)

Return the Warrens IV similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Warrens IV similarity

Return type float

Examples

```
>>> cmp = WarrensIV()
>>> cmp.sim('cat', 'hat')
0.666095890410959
>>> cmp.sim('Niall', 'Neil')
0.5326918120113412
>>> cmp.sim('aluminum', 'Catalan')
0.21031040612607685
>>> cmp.sim('ATCG', 'TAGC')
0.0
```

New in version 0.4.0.

class abydos.distance.**WarrensV**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: `abydos.distance._token_distance._TokenDistance`

Warrens V similarity.

For two sets X and Y and a population N, Warrens V similarity [War08] is

$$sim_{WarrensV}(X, Y) = \frac{|X \cap Y| \cdot |(N \setminus X) \setminus Y| - |X \setminus Y| \cdot |Y \setminus X|}{min(|X| \cdot |Y|, |N \setminus X| \cdot |N \setminus Y|)}$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$sim_{WarrensV} = \frac{ad - bc}{min((a + b)(a + c), (b + d)(c + d))}$$

New in version 0.4.0.

Initialize WarrensV instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in `_TokenDistance` for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in `_TokenDistance` for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

sim(*src*, *tar*)

Return the normalized Warrens V similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Normalized Warrens V similarity

Return type float

Examples

```
>>> cmp = WarrensV()
>>> cmp.sim('cat', 'hat')
0.5
>>> cmp.sim('Niall', 'Neil')
0.3333333333333333
>>> cmp.sim('aluminum', 'Catalan')
0.11125283446712018
>>> cmp.sim('ATCG', 'TAGC')
0.0
```

New in version 0.4.0.

sim_score(*src*, *tar*)

Return the Warrens V similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Warrens V similarity

Return type float

Examples

```
>>> cmp = WarrensV()
>>> cmp.sim_score('cat', 'hat')
97.0
>>> cmp.sim_score('Niall', 'Neil')
51.266666666666666
>>> cmp.sim_score('aluminum', 'Catalan')
9.902777777777779
>>> cmp.sim_score('ATCG', 'TAGC')
-1.0
```

New in version 0.4.0.

class abydos.distance.**WeightedJaccard**(*tokenizer=None*, *intersection_type='crisp'*, *weight=3*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Weighted Jaccard similarity.

For two sets X and Y and a weight w, the Weighted Jaccard similarity [LL98] is

$$sim_{Jaccard_w}(X, Y) = \frac{w \cdot |X \cap Y|}{w \cdot |X \cap Y| + |X \setminus Y| + |Y \setminus X|}$$

Here, the intersection between the two sets is weighted by w. Compare to Jaccard similarity ($w = 1$), and to Dice similarity ($w = 2$). In the default case, the weight of the intersection is 3, following [LL98].

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$sim_{Jaccard_w} = \frac{w \cdot a}{w \cdot a + b + c}$$

New in version 0.4.0.

Initialize TripleWeightedJaccard instance.

Parameters

- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in `_TokenDistance` for details.
- **weight** (*int*) -- The weight to apply to the intersection cardinality. (3, by default.)
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

sim(*src*, *tar*)

Return the Triple Weighted Jaccard similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Weighted Jaccard similarity

Return type float

Examples

```
>>> cmp = WeightedJaccard()
>>> cmp.sim('cat', 'hat')
0.6
>>> cmp.sim('Niall', 'Neil')
0.46153846153846156
>>> cmp.sim('aluminum', 'Catalan')
0.16666666666666666
>>> cmp.sim('ATCG', 'TAGC')
0.0
```

New in version 0.4.0.

class abydos.distance.**Whittaker**(*tokenizer=None*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Whittaker distance.

For two multisets X and Y drawn from an alphabet S, Whittaker distance [Whi52] is

$$sim_{Whittaker}(X, Y) = 1 - \frac{1}{2} \sum_{i \in S} \left| \frac{|X_i|}{|X|} - \frac{|Y_i|}{|Y|} \right|$$

New in version 0.4.0.

Initialize Whittaker instance.

Parameters

- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.

New in version 0.4.0.

sim(*src*, *tar*)

Return the Whittaker distance of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Whittaker distance

Return type float

Examples

```
>>> cmp = Whittaker()
>>> cmp.sim('cat', 'hat')
0.5
>>> cmp.sim('Niall', 'Neil')
0.33333333333333
>>> cmp.sim('aluminum', 'Catalan')
0.11111111111111
>>> cmp.sim('ATCG', 'TAGC')
0.0
```

New in version 0.4.0.

class abydos.distance.**YatesChiSquared**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Yates's Chi-Squared similarity.

For two sets X and Y and a population N, Yates's χ^2 similarity [Yat34] is

$$sim_{Yates_{\chi^2}}(X, Y) = \frac{|N| \cdot (||X \cap Y| \cdot |(N \setminus X) \setminus Y| - |X \setminus Y| \cdot |Y \setminus X|| - \frac{|N|}{2})^2}{|X| \cdot |N \setminus X| \cdot |Y| \cdot |N \setminus Y|}$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$sim_{Yates_{\chi^2}} = \frac{n \cdot (|ad - bc| - \frac{n}{2})^2}{(a+b)(c+d)(a+c)(b+d)}$$

New in version 0.4.0.

Initialize YatesChiSquared instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in _TokenDistance for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the soft and fuzzy variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the fuzzy variant.

New in version 0.4.0.

sim(*src*, *tar*)

Return Yates's normalized Chi-Squared similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Normalized Yates's Chi-Squared similarity

Return type float

Examples

```
>>> cmp = YatesChiSquared()
>>> cmp.sim('cat', 'hat')
0.18081199852082455
>>> cmp.sim('Niall', 'Neil')
0.08608296705052738
>>> cmp.sim('aluminum', 'Catalan')
0.0026563223707532654
>>> cmp.sim('ATCG', 'TAGC')
0.0
```

New in version 0.4.0.

sim_score(*src*, *tar*, *signed=False*)

Return Yates's Chi-Squared similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison
- **signed** (*bool*) -- If True, negative correlations will carry a negative sign

Returns Yates's Chi-Squared similarity

Return type float

Examples

```
>>> cmp = YatesChiSquared()
>>> cmp.sim_score('cat', 'hat')
108.37343852728468
>>> cmp.sim_score('Niall', 'Neil')
56.630055670871954
>>> cmp.sim_score('aluminum', 'Catalan')
1.8574215841854373
>>> cmp.sim_score('ATCG', 'TAGC')
6.960385076156687
```

New in version 0.4.0.

class abydos.distance.**YuleQ**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Yule's Q correlation.

For two sets X and Y and a population N, Yule's Q correlation [Yul12] is

$$corr_{YuleQ}(X, Y) = \frac{|X \cap Y| \cdot |(N \setminus X) \setminus Y| - |X \setminus Y| \cdot |Y \setminus X|}{|X \cap Y| \cdot |(N \setminus X) \setminus Y| + |X \setminus Y| \cdot |Y \setminus X|}$$

Yule himself terms this the coefficient of association.

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$corr_{YuleQ} = \frac{ad - bc}{ad + bc}$$

New in version 0.4.0.

Initialize YuleQ instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in _TokenDistance for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the soft and fuzzy variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the fuzzy variant.

New in version 0.4.0.

corr(*src*, *tar*)

Return Yule's Q correlation of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Yule's Q correlation

Return type float

Examples

```
>>> cmp = YuleQ()
>>> cmp.corr('cat', 'hat')
0.9948717948717949
>>> cmp.corr('Niall', 'Neil')
0.9846350832266325
>>> cmp.corr('aluminum', 'Catalan')
0.8642424242424243
>>> cmp.corr('ATCG', 'TAGC')
-1.0
```

New in version 0.4.0.

sim(*src*, *tar*)
Return Yule's Q similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Yule's Q similarity

Return type float

Examples

```
>>> cmp = YuleQ()
>>> cmp.sim('cat', 'hat')
0.9974358974358974
>>> cmp.sim('Niall', 'Neil')
0.9923175416133163
>>> cmp.sim('aluminum', 'Catalan')
0.9321212121212121
>>> cmp.sim('ATCG', 'TAGC')
0.0
```

New in version 0.4.0.

class abydos.distance.**YuleQII**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)
Bases: abydos.distance._token_distance._TokenDistance

Yule's Q dissimilarity.

For two sets X and Y and a population N, Yule's Q dissimilarity [YK68] is

$$dist_{YuleQII}(X, Y) = \frac{2 \cdot |X \setminus Y| \cdot |Y \setminus X|}{|X \cap Y| \cdot |(N \setminus X) \setminus Y| + |X \setminus Y| \cdot |Y \setminus X|}$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$dist_{YuleQII} = \frac{2bc}{ad + bc}$$

New in version 0.4.0.

Initialize YuleQII instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in _TokenDistance for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

dist (*src*, *tar*)

Return normalized Yule's Q dissimilarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Normalized Yule's Q II distance

Return type float

Examples

```
>>> cmp = YuleQII()
>>> cmp.dist('cat', 'hat')
0.002564102564102564
>>> cmp.dist('Niall', 'Neil')
0.0076824583866837385
>>> cmp.dist('aluminum', 'Catalan')
0.06787878787878789
>>> cmp.dist('ATCG', 'TAGC')
1.0
```

New in version 0.4.0.

dist_abs (*src*, *tar*)

Return Yule's Q dissimilarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Yule's Q II distance

Return type float

Examples

```
>>> cmp = YuleQII()
>>> cmp.dist_abs('cat', 'hat')
0.005128205128205128
>>> cmp.dist_abs('Niall', 'Neil')
0.015364916773367477
>>> cmp.dist_abs('aluminum', 'Catalan')
0.13575757575757577
>>> cmp.dist_abs('ATCG', 'TAGC')
2.0
```

New in version 0.4.0.

class abydos.distance.**YuleY**(*alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Yule's Y correlation.

For two sets X and Y and a population N, Yule's Y correlation [Yul12] is

$$corr_{YuleY}(X,Y) = \frac{\sqrt{|X \cap Y| \cdot |(N \setminus X) \setminus Y|} - \sqrt{|X \setminus Y| \cdot |Y \setminus X|}}{\sqrt{|X \cap Y| \cdot |(N \setminus X) \setminus Y|} + \sqrt{|X \setminus Y| \cdot |Y \setminus X|}}$$

In [Yul12], this is labeled ω, so it is sometimes referred to as Yule's ω. Yule himself terms this the coefficient of colligation.

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$corr_{YuleY} = \frac{\sqrt{ad} - \sqrt{bc}}{\sqrt{ad} + \sqrt{bc}}$$

New in version 0.4.0.

Initialize YuleY instance.

Parameters

- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in _TokenDistance for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the soft and fuzzy variants.

- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

corr(*src*, *tar*)

Return Yule's Y correlation of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Yule's Y correlation

Return type float

Examples

```
>>> cmp = YuleY()
>>> cmp.corr('cat', 'hat')
0.9034892632818762
>>> cmp.corr('Niall', 'Neil')
0.8382551144735259
>>> cmp.corr('aluminum', 'Catalan')
0.5749826820237787
>>> cmp.corr('ATCG', 'TAGC')
-1.0
```

New in version 0.4.0.

sim(*src*, *tar*)

Return Yule's Y similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Yule's Y similarity

Return type float

Examples

```
>>> cmp = YuleY()
>>> cmp.sim('cat', 'hat')
0.9517446316409381
>>> cmp.sim('Niall', 'Neil')
0.919127557236763
>>> cmp.sim('aluminum', 'Catalan')
0.7874913410118893
>>> cmp.sim('ATCG', 'TAGC')
0.0
```

New in version 0.4.0.

class abydos.distance.**YJHHR**(*pval=1*, *alphabet=None*, *tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

YJHHR distance.

For two sets X and Y and a parameter p, YJHHR distance [YJH+16] is

$$dist_{YJHHR_p}(X, Y) = \sqrt[p]{|X \setminus Y|^p + |Y \setminus X|^p}$$

In 2x2 confusion table terms, where a+b+c+d=n, this is

$$dist_{YJHHR} = \sqrt[p]{b^p + c^p}$$

New in version 0.4.0.

Initialize YJHHR instance.

Parameters

- **pval** (*int*) -- The p-value of the L^p-space
- **alphabet** (*Counter, collection, int, or None*) -- This represents the alphabet of possible tokens. See alphabet description in _TokenDistance for details.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the soft and fuzzy variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the fuzzy variant.

New in version 0.4.0.

dist(*src*, *tar*)

Return the normalized YJHHR distance of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns normalized YJHHR distance

Return type float

Examples

```
>>> cmp = YJHHR()
>>> cmp.dist('cat', 'hat')
0.6666666666666666
>>> cmp.dist('Niall', 'Neil')
0.7777777777777778
>>> cmp.dist('aluminum', 'Catalan')
0.9375
>>> cmp.dist('ATCG', 'TAGC')
1.0
```

New in version 0.4.0.

dist_abs(*src*, *tar*)

Return the YJHHR distance of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns YJHHR distance

Return type float

Examples

```
>>> cmp = YJHHR()
>>> cmp.dist_abs('cat', 'hat')
4.0
>>> cmp.dist_abs('Niall', 'Neil')
7.0
>>> cmp.dist_abs('aluminum', 'Catalan')
15.0
>>> cmp.dist_abs('ATCG', 'TAGC')
10.0
```

New in version 0.4.0.

class abydos.distance.**Bhattacharyya**(*tokenizer=None*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Bhattacharyya distance.

For two multisets X and Y drawn from an alphabet S, Bhattacharyya distance [Bha46] is

$$dist_{Bhattacharyya}(X, Y) = -log(\sum_{i \in S} \sqrt{X_i Y_i})$$

New in version 0.4.0.

Initialize Bhattacharyya instance.

Parameters

- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package

- ****kwargs** -- Arbitrary keyword arguments

Other Parameters **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.

New in version 0.4.0.

dist(*src*, *tar*)

Return the Bhattacharyya coefficient of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Bhattacharyya distance

Return type float

Examples

```
>>> cmp = Bhattacharyya()
>>> cmp.dist('cat', 'hat')
0.5
>>> cmp.dist('Niall', 'Neil')
0.3651483716701107
>>> cmp.dist('aluminum', 'Catalan')
0.11785113019775792
>>> cmp.dist('ATCG', 'TAGC')
0.0
```

New in version 0.4.0.

dist_abs(*src*, *tar*)

Return the Bhattacharyya distance of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Bhattacharyya distance

Return type float

Examples

```
>>> cmp = Bhattacharyya()
>>> cmp.dist_abs('cat', 'hat')
0.6931471805599453
>>> cmp.dist_abs('Niall', 'Neil')
1.0074515102711326
>>> cmp.dist_abs('aluminum', 'Catalan')
2.1383330595080277
>>> cmp.dist_abs('ATCG', 'TAGC')
-inf
```

New in version 0.4.0.

class abydos.distance.**BrainerdRobinson**(*tokenizer=None*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Brainerd-Robinson similarity.

For two multisets X and Y drawn from an alphabet S, Brainerd-Robinson similarity [Rob51][Bra51] is

$$sim_{BrainerdRobinson}(X, Y) = 200 - 100 \cdot \sum_{i \in S} |\frac{X_i}{\sum_{i \in S} |X_i|} - \frac{Y_i}{\sum_{i \in S} |Y_i|}|$$

New in version 0.4.0.

Initialize BrainerdRobinson instance.

Parameters

- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.

New in version 0.4.0.

sim(*src*, *tar*)

Return the normalized Brainerd-Robinson similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Normalized Brainerd-Robinson similarity

Return type float

Examples

```
>>> cmp = BrainerdRobinson()
>>> cmp.sim('cat', 'hat')
0.5
>>> cmp.sim('Niall', 'Neil')
0.3333333333333334
>>> cmp.sim('aluminum', 'Catalan')
0.11111111111111
>>> cmp.sim('ATCG', 'TAGC')
0.0
```

New in version 0.4.0.

sim_score(*src*, *tar*)

Return the Brainerd-Robinson similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Brainerd-Robinson similarity

Return type float

Examples

```
>>> cmp = BrainerdRobinson()
>>> cmp.sim_score('cat', 'hat')
100.0
>>> cmp.sim_score('Niall', 'Neil')
66.66666666666669
>>> cmp.sim_score('aluminum', 'Catalan')
22.2222222222222
>>> cmp.sim_score('ATCG', 'TAGC')
0.0
```

New in version 0.4.0.

class abydos.distance.**QuantitativeCosine**(*tokenizer=None*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Quantitative Cosine similarity.

For two multisets X and Y drawn from an alphabet S, Quantitative Cosine similarity is

$$sim_{QuantitativeCosine}(X, Y) = \frac{\sum_{i \in S} X_i Y_i}{\sqrt{\sum_{i \in S} X_i^2} \sqrt{\sum_{i \in S} Y_i^2}}$$

New in version 0.4.0.

Initialize QuantitativeCosine instance.

Parameters

- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters qval (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.

New in version 0.4.0.

sim(*src*, *tar*)

Return the Quantitative Cosine similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Quantitative Cosine similarity

Return type float

Examples

```
>>> cmp = QuantitativeCosine()
>>> cmp.sim('cat', 'hat')
0.5
>>> cmp.sim('Niall', 'Neil')
0.3651483716701107
>>> cmp.sim('aluminum', 'Catalan')
0.10660035817780521
>>> cmp.sim('ATCG', 'TAGC')
0.0
```

New in version 0.4.0.

class abydos.distance.**QuantitativeDice**(*tokenizer=None*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Quantitative Dice similarity.

For two multisets X and Y drawn from an alphabet S, Quantitative Dice similarity is

$$sim_{QuantitativeDice}(X, Y) = \frac{2 \cdot \sum_{i \in S} X_i Y_i}{\sum_{i \in S} X_i^2 + \sum_{i \in S} Y_i^2}$$

New in version 0.4.0.

Initialize QuantitativeDice instance.

Parameters

- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.

New in version 0.4.0.

sim(*src*, *tar*)

Return the Quantitative Dice similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Quantitative Dice similarity

Return type float

Examples

```
>>> cmp = QuantitativeDice()
>>> cmp.sim('cat', 'hat')
0.5
>>> cmp.sim('Niall', 'Neil')
0.36363636363636365
>>> cmp.sim('aluminum', 'Catalan')
0.10526315789473684
>>> cmp.sim('ATCG', 'TAGC')
0.0
```

New in version 0.4.0.

class abydos.distance.**QuantitativeJaccard**(*tokenizer=None*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Quantitative Jaccard similarity.

For two multisets X and Y drawn from an alphabet S, Quantitative Jaccard similarity is

$$sim_{QuantitativeJaccard}(X, Y) = \frac{\sum_{i \in S} X_i Y_i}{\sum_{i \in S} X_i^2 + \sum_{i \in S} Y_i^2 - \sum_{i \in S} X_i Y_i}$$

New in version 0.4.0.

Initialize QuantitativeJaccard instance.

Parameters

- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.

New in version 0.4.0.

sim(*src*, *tar*)

Return the Quantitative Jaccard similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Quantitative Jaccard similarity

Return type float

Examples

```
>>> cmp = QuantitativeJaccard()
>>> cmp.sim('cat', 'hat')
0.3333333333333333
>>> cmp.sim('Niall', 'Neil')
0.2222222222222222
>>> cmp.sim('aluminum', 'Catalan')
0.05555555555555555
>>> cmp.sim('ATCG', 'TAGC')
0.0
```

New in version 0.4.0.

class abydos.distance.**Roberts**(*tokenizer=None*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Roberts similarity.

For two multisets X and Y drawn from an alphabet S, Roberts similarity [Rob86] is

$$sim_{Roberts}(X, Y) = \frac{\left[\sum_{i \in S}(X_i + Y_i) \cdot \frac{min(X_i, Y_i)}{max(X_i, Y_i)}\right]}{\sum_{i \in S}(X_i + Y_i)}$$

New in version 0.4.0.

Initialize Roberts instance.

Parameters

- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.

New in version 0.4.0.

sim(*src*, *tar*)

Return the Roberts similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Roberts similarity

Return type float

Examples

```
>>> cmp = Roberts()
>>> cmp.sim('cat', 'hat')
0.5
>>> cmp.sim('Niall', 'Neil')
0.36363636363636365
>>> cmp.sim('aluminum', 'Catalan')
0.11764705882352941
>>> cmp.sim('ATCG', 'TAGC')
0.0
```

New in version 0.4.0.

class abydos.distance.**AverageLinkage**(*tokenizer=None*, *metric=None*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Average linkage distance.

For two lists of tokens X and Y, average linkage distance [DD16] is

$$dist_{AverageLinkage}(X, Y) = \frac{\sum_{i \in X} \sum_{j \in Y} dist(X_i, Y_j)}{|X| \cdot |Y|}$$

New in version 0.4.0.

Initialize AverageLinkage instance.

Parameters

- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **metric** (*_Distance*) -- A string distance measure class for use in the soft and fuzzy variants. (Defaults to Levenshtein distance)
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.

New in version 0.4.0.

dist(*src*, *tar*)

Return the average linkage distance of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns average linkage distance

Return type float

Examples

```
>>> cmp = AverageLinkage()
>>> cmp.dist('cat', 'hat')
0.8125
>>> cmp.dist('Niall', 'Neil')
0.8333333333333334
>>> cmp.dist('aluminum', 'Catalan')
0.9166666666666666
>>> cmp.dist('ATCG', 'TAGC')
0.8
```

New in version 0.4.0.

class abydos.distance.**SingleLinkage**(*tokenizer=None*, *metric=None*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Single linkage distance.

For two multisets X and Y, single linkage distance [DD16] is

$$dist_{SingleLinkage}(X, Y) = min_{i \in X, j \in Y} dist(X_i, Y_j)$$

New in version 0.4.0.

Initialize SingleLinkage instance.

Parameters

- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants. (Defaults to Levenshtein distance)
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.

New in version 0.4.0.

dist(*src*, *tar*)

Return the normalized single linkage distance of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns normalized single linkage distance

Return type float

Examples

```
>>> cmp = SingleLinkage()
>>> cmp.dist('cat', 'hat')
0.0
>>> cmp.dist('Niall', 'Neil')
0.0
>>> cmp.dist('aluminum', 'Catalan')
0.0
>>> cmp.dist('ATCG', 'TAGC')
0.5
```

New in version 0.4.0.

dist_abs(*src*, *tar*)
Return the single linkage distance of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns single linkage distance

Return type float

Examples

```
>>> cmp = SingleLinkage()
>>> cmp.dist_abs('cat', 'hat')
0.0
>>> cmp.dist_abs('Niall', 'Neil')
0.0
>>> cmp.dist_abs('aluminum', 'Catalan')
0.0
>>> cmp.dist_abs('ATCG', 'TAGC')
1.0
```

New in version 0.4.0.

class abydos.distance.**CompleteLinkage**(*tokenizer=None*, *metric=None*, ***kwargs*)
Bases: abydos.distance._token_distance._TokenDistance

Complete linkage distance.

For two multisets X and Y, complete linkage distance [DD16] is

$$sim_{CompleteLinkage}(X, Y) = max_{i \in X, j \in Y} dist(X_i, Y_j)$$

New in version 0.4.0.

Initialize CompleteLinkage instance.

Parameters

- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package

- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants. (Defaults to Levenshtein distance)
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.

New in version 0.4.0.

dist(*src*, *tar*)

Return the normalized complete linkage distance of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns normalized complete linkage distance

Return type float

Examples

```
>>> cmp = CompleteLinkage()
>>> cmp.dist('cat', 'hat')
1.0
>>> cmp.dist('Niall', 'Neil')
1.0
>>> cmp.dist('aluminum', 'Catalan')
1.0
>>> cmp.dist('ATCG', 'TAGC')
1.0
```

New in version 0.4.0.

dist_abs(*src*, *tar*)

Return the complete linkage distance of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns complete linkage distance

Return type float

Examples

```
>>> cmp = CompleteLinkage()
>>> cmp.dist_abs('cat', 'hat')
2
>>> cmp.dist_abs('Niall', 'Neil')
2
>>> cmp.dist_abs('aluminum', 'Catalan')
2
>>> cmp.dist_abs('ATCG', 'TAGC')
2
```

New in version 0.4.0.

class abydos.distance.**Bag**(*tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Bag distance.

Bag distance is proposed in [BCP02]. It is defined as

$$dist_{bag}(src, tar) = max(|multiset(src) - multiset(tar)|, |multiset(tar) - multiset(src)|)$$

New in version 0.3.6.

Initialize Bag instance.

Parameters

- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the soft and fuzzy variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the fuzzy variant.

New in version 0.4.0.

dist(*src*, *tar*)

Return the normalized bag distance between two strings.

Bag distance is normalized by dividing by $max(|src|, |tar|)$.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns Normalized bag distance

Return type float

Examples

```
>>> cmp = Bag()
>>> cmp.dist('cat', 'hat')
0.3333333333333333
>>> cmp.dist('Niall', 'Neil')
0.4
>>> cmp.dist('aluminum', 'Catalan')
0.625
>>> cmp.dist('ATCG', 'TAGC')
0.0
```

New in version 0.1.0.

Changed in version 0.3.6: Encapsulated in class

dist_abs(*src*, *tar*, *normalized=False*)

Return the bag distance between two strings.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison
- **normalized** (*bool*) -- Normalizes to [0, 1] if True

Returns Bag distance

Return type int or float

Examples

```
>>> cmp = Bag()
>>> cmp.dist_abs('cat', 'hat')
1
>>> cmp.dist_abs('Niall', 'Neil')
2
>>> cmp.dist_abs('aluminum', 'Catalan')
5
>>> cmp.dist_abs('ATCG', 'TAGC')
0
>>> cmp.dist_abs('abcdefg', 'hijklm')
7
>>> cmp.dist_abs('abcdefg', 'hijklmno')
8
```

New in version 0.1.0.

Changed in version 0.3.6: Encapsulated in class

abydos.distance.**bag**(*src*, *tar*)

Return the bag distance between two strings.

This is a wrapper for *Bag.dist_abs()*.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns Bag distance

Return type int

Examples

```
>>> bag('cat', 'hat')
1
>>> bag('Niall', 'Neil')
2
>>> bag('aluminum', 'Catalan')
5
>>> bag('ATCG', 'TAGC')
0
>>> bag('abcdefg', 'hijklm')
7
>>> bag('abcdefg', 'hijklmno')
8
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the Bag.dist_abs method instead.

abydos.distance.**dist_bag**(*src*, *tar*)

Return the normalized bag distance between two strings.

This is a wrapper for *Bag.dist()*.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns Normalized bag distance

Return type float

Examples

```
>>> dist_bag('cat', 'hat')
0.3333333333333333
>>> dist_bag('Niall', 'Neil')
0.4
>>> dist_bag('aluminum', 'Catalan')
0.625
>>> dist_bag('ATCG', 'TAGC')
0.0
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the Bag.dist method instead.

abydos.distance.**sim_bag**(*src*, *tar*)

Return the normalized bag similarity of two strings.

This is a wrapper for `Bag.sim()`.

Parameters

- **src** (*str*) -- Source string for comparison

- **tar** (*str*) -- Target string for comparison

Returns Normalized bag similarity

Return type float

Examples

```
>>> round(sim_bag('cat', 'hat'), 12)
0.666666666667
>>> sim_bag('Niall', 'Neil')
0.6
>>> sim_bag('aluminum', 'Catalan')
0.375
>>> sim_bag('ATCG', 'TAGC')
1.0
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the Bag.sim method instead.

class abydos.distance.**SoftCosine**(*tokenizer=None*, *metric=None*, *sim_method='a'*, ***kwargs*)
Bases: abydos.distance._token_distance._TokenDistance

Soft Cosine similarity.

As described in [SGGomezAP14], soft cosine similarity of two multi-sets X and Y, drawn from an alphabet S, is

$$sim_{softcosine}(X, Y) = \frac{\sum_{i \in S} \sum_{j \in S} s_{ij} X_i Y_j}{\sqrt{\sum_{i \in S} \sum_{j \in S} s_{ij} X_i X_j} \sqrt{\sum_{i \in S} \sum_{j \in S} s_{ij} Y_i Y_j}}$$

where s_{ij} is the similarity of two tokens, by default a function of Levenshtein distance: $\frac{1}{1+Levenshtein\_distance(i,j)}$.

Notes

This class implements soft cosine similarity, as defined by [SGGomezAP14]. An alternative formulation of soft cosine similarity using soft (multi-)sets is provided by the *Cosine* class using intersection_type=``soft``, based on the soft intersection defined in [RHJF14].

New in version 0.4.0.

Initialize SoftCosine instance.

Parameters

- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package, defaulting to the QGrams tokenizer with q=4
- **threshold** (*float*) -- The minimum similarity for a pair of tokens to contribute to similarity
- **metric** (*_Distance*) -- A distance instance from the abydos.distance package, defaulting to Levenshtein distance

- **sim_method** (*str*) -- Selects the similarity method from the four given in [SGGomezAP14]:
  - a : $\frac{1}{1+d}$
  - b : $1 - \frac{d}{m}$
  - c : $\sqrt{1 - \frac{d}{m}}$
  - d : $\left(1 - \frac{d}{m}\right)^2$

  Where d is the distance (Levenshtein by default) and m is the maximum length of the two tokens. Option *a* is default, as suggested by the paper.
- ****kwargs** -- Arbitrary keyword arguments

Raises **ValueError** -- sim_method must be one of 'a', 'b', 'c', or 'd'

Other Parameters **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.

New in version 0.4.0.

sim(*src*, *tar*)

Return the Soft Cosine similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Fuzzy Cosine similarity

Return type float

Examples

```
>>> cmp = SoftCosine()
>>> cmp.sim('cat', 'hat')
0.8750000000000001
>>> cmp.sim('Niall', 'Neil')
0.8844691709074513
>>> cmp.sim('aluminum', 'Catalan')
0.831348688760277
>>> cmp.sim('ATCG', 'TAGC')
0.8571428571428572
```

New in version 0.4.0.

class abydos.distance.**MongeElkan**(*sim_func=None*, *symmetric=False*, ***kwargs*)

Bases: abydos.distance._distance._Distance

Monge-Elkan similarity.

Monge-Elkan is defined in [ME96].

Note: Monge-Elkan is NOT a symmetric similarity algorithm. Thus, the similarity of src to tar is not necessarily equal to the similarity of tar to src. If the symmetric argument is True, a symmetric value is calculated, at the cost of doubling the computation time (since $sim_{Monge-Elkan}(src, tar)$ and $sim_{Monge-Elkan}(tar, src)$ are both calculated and then averaged).

New in version 0.3.6.

Initialize MongeElkan instance.

Parameters

- **sim_func** (*function*) -- The internal similarity metric to employ
- **symmetric** (*bool*) -- Return a symmetric similarity measure
- ****kwargs** -- Arbitrary keyword arguments

New in version 0.4.0.

sim(*src*, *tar*)

Return the Monge-Elkan similarity of two strings.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns Monge-Elkan similarity

Return type float

Examples

```
>>> cmp = MongeElkan()
>>> cmp.sim('cat', 'hat')
0.75
>>> round(cmp.sim('Niall', 'Neil'), 12)
0.666666666667
>>> round(cmp.sim('aluminum', 'Catalan'), 12)
0.388888888889
>>> cmp.sim('ATCG', 'TAGC')
0.5
```

New in version 0.1.0.

Changed in version 0.3.6: Encapsulated in class

abydos.distance.**dist_monge_elkan**(*src*, *tar*, *sim_func=<function sim_levenshtein>*, *symmetric=False*)

Return the Monge-Elkan distance between two strings.

This is a wrapper for `MongeElkan.dist()`.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison
- **sim_func** (*function*) -- The internal similarity metric to employ
- **symmetric** (*bool*) -- Return a symmetric similarity measure

Returns Monge-Elkan distance

Return type float

Examples

```
>>> dist_monge_elkan('cat', 'hat')
0.25
>>> round(dist_monge_elkan('Niall', 'Neil'), 12)
0.333333333333
>>> round(dist_monge_elkan('aluminum', 'Catalan'), 12)
0.611111111111
>>> dist_monge_elkan('ATCG', 'TAGC')
0.5
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the MongeElkan.dist method instead.

abydos.distance.**sim_monge_elkan**(*src*, *tar*, *sim_func=<function sim_levenshtein>*, *symmetric=False*)

Return the Monge-Elkan similarity of two strings.

This is a wrapper for *MongeElkan.sim()*.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison
- **sim_func** (*function*) -- Rhe internal similarity metric to employ
- **symmetric** (*bool*) -- Return a symmetric similarity measure

Returns Monge-Elkan similarity

Return type float

Examples

```
>>> sim_monge_elkan('cat', 'hat')
0.75
>>> round(sim_monge_elkan('Niall', 'Neil'), 12)
0.666666666667
>>> round(sim_monge_elkan('aluminum', 'Catalan'), 12)
0.388888888889
>>> sim_monge_elkan('ATCG', 'TAGC')
0.5
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the MongeElkan.sim method instead.

class abydos.distance.**TFIDF**(*tokenizer=None*, *corpus=None*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

TF-IDF similarity.

For two sets X and Y and a population N, TF-IDF similarity [CRF03] is

$$sim_{TF-IDF}(X, Y) = \sum_{w \in X \cap Y} V(w, X) \cdot V(w, Y)$$

$$V(w, S) = \frac{V'(w, S)}{\sqrt{\sum_{w \in S} V'(w, S)^2}}$$

$$V'(w, S) = log(1 + TF_{w,S}) \cdot log(1 + IDF_w)$$

Notes

One is added to both the TF & IDF values before taking the logarithm to ensure the logarithms do not fall to 0, which will tend to result in 0.0 similarities even when there is a degree of matching.

New in version 0.4.0.

Initialize TFIDF instance.

Parameters

- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **corpus** (UnigramCorpus) -- A unigram corpus UnigramCorpus. If None, a corpus will be created from the two words when a similarity function is called.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.

New in version 0.4.0.

sim(*src*, *tar*)

Return the TF-IDF similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns TF-IDF similarity

Return type float

Examples

```
>>> cmp = TFIDF()
>>> cmp.sim('cat', 'hat')
0.30404449697373
>>> cmp.sim('Niall', 'Neil')
0.20108911303601
>>> cmp.sim('aluminum', 'Catalan')
0.05355175631194
>>> cmp.sim('ATCG', 'TAGC')
0.0
```

New in version 0.4.0.

class abydos.distance.**SoftTFIDF**(*tokenizer=None*, *corpus=None*, *metric=None*, *threshold=0.9*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

SoftTF-IDF similarity.

For two sets X and Y and a population N, SoftTF-IDF similarity [CRF03] is

$$sim_{SoftTF-IDF}(X,Y) = \sum_{w \in \{sim_{metric}(x,y) \geq \theta | x \in X, y \in Y\}} V(w,S) \cdot V(w,X) \cdot V(w,Y)$$

$$V(w,S) = \frac{V'(w,S)}{\sqrt{\sum_{w \in S} V'(w,S)^2}}$$

$$V'(w,S) = log(1 + TF_{w,S}) \cdot log(1 + IDF_w)$$

Notes

One is added to both the TF & IDF values before taking the logarithm to ensure the logarithms do not fall to 0, which will tend to result in 0.0 similarities even when there is a degree of matching.

Rather than needing to exceed the threshold value, as in [CRF03] the similarity must be greater than or equal to the threshold.

New in version 0.4.0.

Initialize SoftTFIDF instance.

Parameters

- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **corpus** (UnigramCorpus) -- A unigram corpus UnigramCorpus. If None, a corpus will be created from the two words when a similarity function is called.
- **metric** (*_Distance*) -- A string distance measure class for making soft matches, by default Jaro-Winkler.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as soft matches, by default 0.9.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.

New in version 0.4.0.

sim(*src*, *tar*)

Return the SoftTF-IDF similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns SoftTF-IDF similarity

Return type float

Examples

```
>>> cmp = SoftTFIDF()
>>> cmp.sim('cat', 'hat')
0.30404449697373
>>> cmp.sim('Niall', 'Neil')
0.20108911303601
>>> cmp.sim('aluminum', 'Catalan')
0.05355175631194
>>> cmp.sim('ATCG', 'TAGC')
0.0
```

New in version 0.4.0.

class abydos.distance.**JensenShannon**(*tokenizer=None*, *intersection_type='crisp'*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Jensen-Shannon divergence.

Jensen-Shannon divergence [DLP99] of two multi-sets X and Y is

$$\begin{aligned} dist_{JS}(X,Y) &= log2 + \tfrac{1}{2}\sum_{i \in X \cap Y} h(p(X_i) + p(Y_i)) - h(p(X_i)) - h(p(Y_i)) \\ h(x) &= -xlogx \\ p(X_i \in X) &= \tfrac{|X_i|}{|X|} \end{aligned}$$

New in version 0.4.0.

Initialize JensenShannon instance.

Parameters

- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.
- ****kwargs** -- Arbitrary keyword arguments

New in version 0.4.0.

dist(*src*, *tar*)

Return the normalized Jensen-Shannon distance of two strings.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns Normalized Jensen-Shannon distance

Return type float

Examples

```
>>> cmp = JensenShannon()
>>> cmp.dist('cat', 'hat')
0.49999999999999994
>>> cmp.dist('Niall', 'Neil')
0.6355222557917826
>>> cmp.dist('aluminum', 'Catalan')
0.8822392827203127
>>> cmp.dist('ATCG', 'TAGC')
1.0
```

New in version 0.4.0.

dist_abs(*src*, *tar*)

Return the Jensen-Shannon divergence of two strings.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns Jensen-Shannon divergence

Return type float

Examples

```
>>> cmp = JensenShannon()
>>> cmp.dist_abs('cat', 'hat')
0.3465735902799726
>>> cmp.dist_abs('Niall', 'Neil')
0.44051045978517045
>>> cmp.dist_abs('aluminum', 'Catalan')
0.6115216713968132
>>> cmp.dist_abs('ATCG', 'TAGC')
0.6931471805599453
```

New in version 0.4.0.

class abydos.distance.**FellegiSunter**(*tokenizer=None*, *intersection_type='crisp'*, *simplified=False*, *mismatch_factor=0.5*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

Fellegi-Sunter similarity.

Fellegi-Sunter similarity is based on the description in [CRF03] and implementation in [CRFR03].

New in version 0.4.0.

Initialize FellegiSunter instance.

Parameters

- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **intersection_type** (*str*) -- Specifies the intersection type, and set type as a result: See intersection_type description in _TokenDistance for details.
- **simplified** (*bool*) -- Specifies to use the simplified scoring variant

- **mismatch_factor** (*float*) -- Specifies the penalty factor for mismatches
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters

- **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.
- **metric** (*_Distance*) -- A string distance measure class for use in the `soft` and `fuzzy` variants.
- **threshold** (*float*) -- A threshold value, similarities above which are counted as members of the intersection for the `fuzzy` variant.

New in version 0.4.0.

sim(*src*, *tar*)

Return the normalized Fellegi-Sunter similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Normalized Fellegi-Sunter similarity

Return type float

Examples

```
>>> cmp = FellegiSunter()
>>> cmp.sim('cat', 'hat')
0.2934477792670495
>>> cmp.sim('Niall', 'Neil')
0.13917536933271363
>>> cmp.sim('aluminum', 'Catalan')
0.056763632331436484
>>> cmp.sim('ATCG', 'TAGC')
0.0
```

New in version 0.4.0.

sim_score(*src*, *tar*)

Return the Fellegi-Sunter similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Fellegi-Sunter similarity

Return type float

Examples

```
>>> cmp = FellegiSunter()
>>> cmp.sim_score('cat', 'hat')
0.8803433378011485
>>> cmp.sim_score('Niall', 'Neil')
0.6958768466635681
>>> cmp.sim_score('aluminum', 'Catalan')
0.45410905865149187
>>> cmp.sim_score('ATCG', 'TAGC')
0.0
```

New in version 0.4.0.

class abydos.distance.**MinHash**(*tokenizer=None*, *k=0*, *seed=10*, ***kwargs*)

Bases: abydos.distance._distance._Distance

MinHash similarity.

MinHash similarity [Bro97] is a method of approximating the intersection over the union of two sets. This implementation is based on [Kul15].

New in version 0.4.0.

Initialize MinHash instance.

Parameters

- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **k** (*int*) -- The number of hash functions to use for similarity estimation
- **seed** (*int*) -- A seed value for the random functions
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.

New in version 0.4.0.

sim(*src*, *tar*)

Return the MinHash similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns MinHash similarity

Return type float

Examples

```
>>> cmp = MinHash()
>>> cmp.sim('cat', 'hat')
0.75
>>> cmp.sim('Niall', 'Neil')
1.0
>>> cmp.sim('aluminum', 'Catalan')
0.5
>>> cmp.sim('ATCG', 'TAGC')
0.6
```

New in version 0.4.0.

class abydos.distance.**BLEU**(*n_min=1*, *n_max=4*, *tokenizers=None*, *weights=None*, ***kwargs*)

Bases: abydos.distance._distance._Distance

BLEU similarity.

BLEU similarity [PRWZ02] compares two strings for similarity using a set of tokenizers and a brevity penalty:

$$BP = \begin{cases} 1 & \text{if} \quad c > r \\ e^{(1-\frac{r}{c})} & \text{if} \quad c \leq r \end{cases}$$

The BLEU score is then:

$$\text{BLEU} = BP \cdot e^{\sum_{n=1}^{N} w_n log p_n}$$

For tokenizers 1 to N, by default q-gram tokenizers for q=1 to N in Abydos, weights w_n, which are uniformly $\frac{1}{N}$, and p_n:

$$p_n = \frac{\sum_{token \in tar} min(Count(token \in tar), Count(token \in src))}{|tar|}$$

New in version 0.4.0.

Initialize BLEU instance.

Parameters

- **n_min** (*int*) -- The minimum q-gram value for BLEU score calculation (1 by default)
- **n_max** (*int*) -- The maximum q-gram value for BLEU score calculation (4 by default)
- **tokenizers** (*list (_Tokenizer)*) -- A list of initialized tokenizers
- **weights** (*list (float)*) -- A list of floats representing the weights of the tokenizers. If tokenizers is set, this must have the same length. If n_min and n_max are used to set tokenizers, this must have length equal to n_max-n_min-1. Otherwise, uniform weights will be used.
- ****kwargs** -- Arbitrary keyword arguments

New in version 0.4.0.

sim(*src*, *tar*)
Return the BLEU similarity of two strings.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns BLEU similarity

Return type float

Examples

```
>>> cmp = BLEU()
>>> cmp.sim('cat', 'hat')
0.7598356856515925
>>> cmp.sim('Niall', 'Neil')
0.7247557929987696
>>> cmp.sim('aluminum', 'Catalan')
0.44815260192961937
>>> cmp.sim('ATCG', 'TAGC')
1.0
```

New in version 0.4.0.

class abydos.distance.**RougeL**(***kwargs*)
Bases: abydos.distance._distance._Distance

Rouge-L similarity.

Rouge-L similarity [Lin04]

New in version 0.4.0.

Initialize RougeL instance.

Parameters ****kwargs** -- Arbitrary keyword arguments

New in version 0.4.0.

sim(*src*, *tar*, *beta=8*)
Return the Rouge-L similarity of two strings.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison
- **beta** (*int or float*) -- A weighting factor to prejudice similarity towards src

Returns Rouge-L similarity

Return type float

Examples

```
>>> cmp = RougeL()
>>> cmp.sim('cat', 'hat')
0.6666666666666666
>>> cmp.sim('Niall', 'Neil')
0.6018518518518519
>>> cmp.sim('aluminum', 'Catalan')
0.3757225433526012
>>> cmp.sim('ATCG', 'TAGC')
0.5
```

New in version 0.4.0.

class abydos.distance.**RougeW**(*f_func=None*, *f_inv=None*, ***kwargs*)

Bases: abydos.distance._distance._Distance

Rouge-W similarity.

Rouge-W similarity [Lin04]

New in version 0.4.0.

Initialize RougeW instance.

Parameters

- **f_func** (*function*) -- A weighting function based on the value supplied to this function, such that f(x+y) > f(x) + f(y)
- **f_inv** (*function*) -- The close form inverse of f_func
- ****kwargs** -- Arbitrary keyword arguments

New in version 0.4.0.

sim(*src*, *tar*, *beta=8*)

Return the Rouge-W similarity of two strings.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison
- **beta** (*int or float*) -- A weighting factor to prejudice similarity towards src

Returns Rouge-W similarity

Return type float

Examples

```
>>> cmp = RougeW()
>>> cmp.sim('cat', 'hat')
0.6666666666666666
>>> cmp.sim('Niall', 'Neil')
0.34747932867894143
>>> cmp.sim('aluminum', 'Catalan')
0.280047049205176
>>> cmp.sim('ATCG', 'TAGC')
0.43301270189221935
```

New in version 0.4.0.

wlcs(*src*, *tar*)
Return the Rouge-W weighted longest common sub-sequence length.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns The Levenshtein distance between src & tar

Return type int (may return a float if cost has float values)

Examples

```
>>> cmp = RougeW()
>>> cmp.wlcs('cat', 'hat')
4
>>> cmp.wlcs('Niall', 'Neil')
3
>>> cmp.wlcs('aluminum', 'Catalan')
5
>>> cmp.wlcs('ATCG', 'TAGC')
3
```

New in version 0.4.0.

class abydos.distance.**RougeS**(*qval=2*, ***kwargs*)
Bases: abydos.distance._distance._Distance

Rouge-S similarity.

Rouge-S similarity [Lin04], operating on character-level skipgrams

New in version 0.4.0.

Initialize RougeS instance.

Parameters ****kwargs** -- Arbitrary keyword arguments

New in version 0.4.0.

sim(*src*, *tar*, *beta=8*)
Return the Rouge-S similarity of two strings.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison
- **beta** (*int or float*) -- A weighting factor to prejudice similarity towards src

Returns Rouge-S similarity

Return type float

Examples

```
>>> cmp = RougeS()
>>> cmp.sim('cat', 'hat')
0.3333333333333333
>>> cmp.sim('Niall', 'Neil')
0.30185758513931893
>>> cmp.sim('aluminum', 'Catalan')
0.10755653612796467
>>> cmp.sim('ATCG', 'TAGC')
0.6666666666666666
```

New in version 0.4.0.

class abydos.distance.**RougeSU**(*qval=2*, ***kwargs*)

Bases: abydos.distance._rouge_s.RougeS

Rouge-SU similarity.

Rouge-SU similarity [Lin04], operating on character-level skipgrams

New in version 0.4.0.

Initialize RougeSU instance.

Parameters ****kwargs** -- Arbitrary keyword arguments

New in version 0.4.0.

sim(*src*, *tar*, *beta=8*)

Return the Rouge-SU similarity of two strings.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison
- **beta** (*int or float*) -- A weighting factor to prejudice similarity towards src

Returns Rouge-SU similarity

Return type float

Examples

```
>>> cmp = RougeSU()
>>> cmp.sim('cat', 'hat')
0.5
>>> cmp.sim('Niall', 'Neil')
0.4020618556701031
>>> cmp.sim('aluminum', 'Catalan')
0.1672384219554031
>>> cmp.sim('ATCG', 'TAGC')
0.8
```

New in version 0.4.0.

class abydos.distance.**PositionalQGramDice**(*max_dist=1*, *tokenizer=None*, ***kwargs*)

Bases: abydos.distance._distance._Distance

Positional Q-Gram Dice coefficient.

Positional Q-Gram Dice coefficient [GIJ+01][Chr06]

New in version 0.4.0.

Initialize PositionalQGramDice instance.

Parameters

- **max_dist** (*int*) -- The maximum positional distance between to q-grams to count as a match.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.

New in version 0.4.0.

sim(*src*, *tar*)

Return the Positional Q-Gram Dice coefficient of two strings.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns Positional Q-Gram Dice coefficient

Return type float

Examples

```
>>> cmp = PositionalQGramDice()
>>> cmp.sim('cat', 'hat')
0.5
>>> cmp.sim('Niall', 'Neil')
0.36363636363636365
>>> cmp.sim('aluminum', 'Catalan')
0.0
>>> cmp.sim('ATCG', 'TAGC')
0.0
```

New in version 0.4.0.

class abydos.distance.**PositionalQGramJaccard**(*max_dist=1*, *tokenizer=None*, ***kwargs*)

Bases: abydos.distance._distance._Distance

Positional Q-Gram Jaccard coefficient.

Positional Q-Gram Jaccard coefficient [GIJ+01][Chr06]

New in version 0.4.0.

Initialize PositionalQGramJaccard instance.

Parameters

- **max_dist** (*int*) -- The maximum positional distance between to q-grams to count as a match.

- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.

New in version 0.4.0.

sim(*src*, *tar*)

Return the Positional Q-Gram Jaccard coefficient of two strings.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns Positional Q-Gram Jaccard coefficient

Return type float

Examples

```
>>> cmp = PositionalQGramJaccard()
>>> cmp.sim('cat', 'hat')
0.3333333333333333
>>> cmp.sim('Niall', 'Neil')
0.2222222222222222
>>> cmp.sim('aluminum', 'Catalan')
0.0
>>> cmp.sim('ATCG', 'TAGC')
0.0
```

New in version 0.4.0.

class abydos.distance.**PositionalQGramOverlap**(*max_dist=1*, *tokenizer=None*, ***kwargs*)

Bases: abydos.distance._distance._Distance

Positional Q-Gram Overlap coefficient.

Positional Q-Gram Overlap coefficient [GIJ+01][Chr06]

New in version 0.4.0.

Initialize PositionalQGramOverlap instance.

Parameters

- **max_dist** (*int*) -- The maximum positional distance between to q-grams to count as a match.
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.

New in version 0.4.0.

sim(*src*, *tar*)
Return the Positional Q-Gram Overlap coefficient of two strings.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns Positional Q-Gram Overlap coefficient

Return type float

Examples

```
>>> cmp = PositionalQGramOverlap()
>>> cmp.sim('cat', 'hat')
0.5
>>> cmp.sim('Niall', 'Neil')
0.4
>>> cmp.sim('aluminum', 'Catalan')
0.0
>>> cmp.sim('ATCG', 'TAGC')
0.0
```

New in version 0.4.0.

class abydos.distance.**NeedlemanWunsch**(*gap_cost=1*, *sim_func=None*, ***kwargs*)
Bases: abydos.distance._distance._Distance

Needleman-Wunsch score.

The Needleman-Wunsch score [NW70] is a standard edit distance measure.

New in version 0.3.6.

Initialize NeedlemanWunsch instance.

Parameters

- **gap_cost** (*float*) -- The cost of an alignment gap (1 by default)
- **sim_func** (*function*) -- A function that returns the similarity of two characters (identity similarity by default)
- ****kwargs** -- Arbitrary keyword arguments

New in version 0.4.0.

sim(*src*, *tar*)
Return the normalized Needleman-Wunsch score of two strings.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns Normalized Needleman-Wunsch score

Return type float

Examples

```
>>> cmp = NeedlemanWunsch()
>>> cmp.sim('cat', 'hat')
0.6666666666666667
>>> cmp.sim('Niall', 'Neil')
0.22360679774997896
>>> round(cmp.sim('aluminum', 'Catalan'), 12)
0.0
>>> cmp.sim('cat', 'hat')
0.6666666666666667
```

New in version 0.4.1.

static sim_matrix(*src*, *tar*, *mat=None*, *mismatch_cost=0*, *match_cost=1*, *symmetric=True*, *alphabet=None*)

Return the matrix similarity of two strings.

With the default parameters, this is identical to sim_ident. It is possible for sim_matrix to return values outside of the range $[0, 1]$, if values outside that range are present in mat, mismatch_cost, or match_cost.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison
- **mat** (*dict*) -- A dict mapping tuples to costs; the tuples are (src, tar) pairs of symbols from the alphabet parameter
- **mismatch_cost** (*float*) -- The value returned if (src, tar) is absent from mat when src does not equal tar
- **match_cost** (*float*) -- The value returned if (src, tar) is absent from mat when src equals tar
- **symmetric** (*bool*) -- True if the cost of src not matching tar is identical to the cost of tar not matching src; in this case, the values in mat need only contain (src, tar) or (tar, src), not both
- **alphabet** (*str*) -- A collection of tokens from which src and tar are drawn; if this is defined a ValueError is raised if either tar or src is not found in alphabet

Returns Matrix similarity

Return type float

Raises

- **ValueError** -- src value not in alphabet
- **ValueError** -- tar value not in alphabet

Examples

```
>>> NeedlemanWunsch.sim_matrix('cat', 'hat')
0
>>> NeedlemanWunsch.sim_matrix('hat', 'hat')
1
```

New in version 0.1.0.

Changed in version 0.3.6: Encapsulated in class

sim_score(*src*, *tar*)

Return the Needleman-Wunsch score of two strings.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns Needleman-Wunsch score

Return type float

Examples

```
>>> cmp = NeedlemanWunsch()
>>> cmp.sim_score('cat', 'hat')
2.0
>>> cmp.sim_score('Niall', 'Neil')
1.0
>>> cmp.sim_score('aluminum', 'Catalan')
-1.0
>>> cmp.sim_score('ATCG', 'TAGC')
0.0
```

New in version 0.1.0.

Changed in version 0.3.6: Encapsulated in class

abydos.distance.**needleman_wunsch**(*src*, *tar*, *gap_cost=1*, *sim_func=<function sim_ident>*)

Return the Needleman-Wunsch score of two strings.

This is a wrapper for `NeedlemanWunsch.dist_abs()`.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison
- **gap_cost** (*float*) -- The cost of an alignment gap (1 by default)
- **sim_func** (*function*) -- A function that returns the similarity of two characters (identity similarity by default)

Returns Needleman-Wunsch score

Return type float

Examples

```
>>> needleman_wunsch('cat', 'hat')
2.0
>>> needleman_wunsch('Niall', 'Neil')
1.0
>>> needleman_wunsch('aluminum', 'Catalan')
-1.0
>>> needleman_wunsch('ATCG', 'TAGC')
0.0
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the NeedlemanWunsch.dist_abs method instead.

class abydos.distance.**SmithWaterman**(*gap_cost=1*, *sim_func=None*, ***kwargs*)
Bases: abydos.distance._needleman_wunsch.NeedlemanWunsch

Smith-Waterman score.

The Smith-Waterman score [SW81] is a standard edit distance measure, differing from Needleman-Wunsch in that it focuses on local alignment and disallows negative scores.

New in version 0.3.6.

Initialize SmithWaterman instance.

Parameters

- **gap_cost** (*float*) -- The cost of an alignment gap (1 by default)
- **sim_func** (*function*) -- A function that returns the similarity of two characters (identity similarity by default)
- ****kwargs** -- Arbitrary keyword arguments

New in version 0.4.0.

sim(*src*, *tar*)
Return the normalized Smith-Waterman score of two strings.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns Normalized Smith-Waterman score

Return type float

Examples

```
>>> cmp = SmithWaterman()
>>> cmp.sim('cat', 'hat')
0.6666666666666667
>>> cmp.sim('Niall', 'Neil')
0.22360679774997896
>>> round(cmp.sim('aluminum', 'Catalan'), 12)
0.0
```

(continues on next page)

(continued from previous page)

```
>>> cmp.sim('cat', 'hat')
0.6666666666666667
```

New in version 0.4.1.

sim_score(*src*, *tar*)

Return the Smith-Waterman score of two strings.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns Smith-Waterman score

Return type float

Examples

```
>>> cmp = SmithWaterman()
>>> cmp.sim_score('cat', 'hat')
2.0
>>> cmp.sim_score('Niall', 'Neil')
1.0
>>> cmp.sim_score('aluminum', 'Catalan')
0.0
>>> cmp.sim_score('ATCG', 'TAGC')
1.0
```

New in version 0.1.0.

Changed in version 0.3.6: Encapsulated in class

abydos.distance.**smith_waterman**(*src*, *tar*, *gap_cost=1*, *sim_func=<function sim_ident>*)

Return the Smith-Waterman score of two strings.

This is a wrapper for `SmithWaterman.dist_abs()`.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison
- **gap_cost** (*float*) -- The cost of an alignment gap (1 by default)
- **sim_func** (*function*) -- A function that returns the similarity of two characters (identity similarity by default)

Returns Smith-Waterman score

Return type float

Examples

```
>>> smith_waterman('cat', 'hat')
2.0
>>> smith_waterman('Niall', 'Neil')
1.0
>>> smith_waterman('aluminum', 'Catalan')
0.0
>>> smith_waterman('ATCG', 'TAGC')
1.0
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the SmithWaterman.dist_abs method instead.

class abydos.distance.**Gotoh**(*gap_open=1*, *gap_ext=0.4*, *sim_func=None*, ***kwargs*)
Bases: abydos.distance._needleman_wunsch.NeedlemanWunsch

Gotoh score.

The Gotoh score [Got82] is essentially Needleman-Wunsch with affine gap penalties.

New in version 0.3.6.

Initialize Gotoh instance.

Parameters

- **gap_open** (*float*) -- The cost of an open alignment gap (1 by default)
- **gap_ext** (*float*) -- The cost of an alignment gap extension (0.4 by default)
- **sim_func** (*function*) -- A function that returns the similarity of two characters (identity similarity by default)
- ****kwargs** -- Arbitrary keyword arguments

New in version 0.4.0.

sim(*src*, *tar*)
Return the normalized Gotoh score of two strings.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns Normalized Gotoh score

Return type float

Examples

```
>>> cmp = Gotoh()
>>> cmp.sim('cat', 'hat')
0.6666666666666667
>>> cmp.sim('Niall', 'Neil')
0.22360679774997896
>>> round(cmp.sim('aluminum', 'Catalan'), 12)
0.0
>>> cmp.sim('cat', 'hat')
0.6666666666666667
```

New in version 0.4.1.

sim_score(*src*, *tar*)

Return the Gotoh score of two strings.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns Gotoh score

Return type float

Examples

```
>>> cmp = Gotoh()
>>> cmp.sim_score('cat', 'hat')
2.0
>>> cmp.sim_score('Niall', 'Neil')
1.0
>>> round(cmp.sim_score('aluminum', 'Catalan'), 12)
-0.4
>>> cmp.sim_score('cat', 'hat')
2.0
```

New in version 0.1.0.

Changed in version 0.3.6: Encapsulated in class

abydos.distance.**gotoh**(*src*, *tar*, *gap_open=1*, *gap_ext=0.4*, *sim_func=<function sim_ident>*)

Return the Gotoh score of two strings.

This is a wrapper for `Gotoh.dist_abs()`.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison
- **gap_open** (*float*) -- The cost of an open alignment gap (1 by default)
- **gap_ext** (*float*) -- The cost of an alignment gap extension (0.4 by default)
- **sim_func** (*function*) -- A function that returns the similarity of two characters (identity similarity by default)

Returns Gotoh score

Return type float

Examples

```
>>> gotoh('cat', 'hat')
2.0
>>> gotoh('Niall', 'Neil')
1.0
>>> round(gotoh('aluminum', 'Catalan'), 12)
-0.4
>>> gotoh('cat', 'hat')
2.0
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the Gotoh.dist_abs method instead.

class abydos.distance.**LCSseq**(*normalizer=<built-in function max>*, ***kwargs*)

Bases: abydos.distance._distance._Distance

Longest common subsequence.

Longest common subsequence (LCSseq) is the longest subsequence of characters that two strings have in common.

New in version 0.3.6.

Initialize LCSseq.

Parameters

- **normalizer** (*function*) -- A normalization function for the normalized similarity & distance. By default, the max of the lengths of the input strings. If lambda x: sum(x)/2.0 is supplied, the normalization proposed in [RTS+01] is used, i.e. $\frac{2|LCS(src,tar)|}{|src|+|tar|}$.
- ****kwargs** -- Arbitrary keyword arguments

New in version 0.4.0.

lcsseq(*src*, *tar*)

Return the longest common subsequence of two strings.

Based on the dynamic programming algorithm from http://rosettacode.org/wiki/Longest_common_subsequence [Cod18a]. This is licensed GFDL 1.2.

Modifications include: conversion to a numpy array in place of a list of lists

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns The longest common subsequence

Return type str

Examples

```
>>> sseq = LCSseq()
>>> sseq.lcsseq('cat', 'hat')
'at'
>>> sseq.lcsseq('Niall', 'Neil')
'Nil'
>>> sseq.lcsseq('aluminum', 'Catalan')
'aln'
>>> sseq.lcsseq('ATCG', 'TAGC')
'AC'
```

New in version 0.1.0.

Changed in version 0.3.6: Encapsulated in class

sim(*src*, *tar*)

Return the longest common subsequence similarity of two strings.

Longest common subsequence similarity (sim_{LCSseq}).

This employs the LCSseq function to derive a similarity metric: $sim_{LCSseq}(s,t) = \frac{|LCSseq(s,t)|}{max(|s|,|t|)}$

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns LCSseq similarity

Return type float

Examples

```
>>> sseq = LCSseq()
>>> sseq.sim('cat', 'hat')
0.6666666666666666
>>> sseq.sim('Niall', 'Neil')
0.6
>>> sseq.sim('aluminum', 'Catalan')
0.375
>>> sseq.sim('ATCG', 'TAGC')
0.5
```

New in version 0.1.0.

Changed in version 0.3.6: Encapsulated in class

Changed in version 0.4.0: Added normalization option

abydos.distance.**lcsseq**(*src*, *tar*)

Return the longest common subsequence of two strings.

This is a wrapper for *LCSseq.lcsseq()*.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns The longest common subsequence

Return type str

Examples

```
>>> lcsseq('cat', 'hat')
'at'
>>> lcsseq('Niall', 'Neil')
'Nil'
>>> lcsseq('aluminum', 'Catalan')
'aln'
>>> lcsseq('ATCG', 'TAGC')
'AC'
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the LCSseq.lcsseq method instead.

abydos.distance.**dist_lcsseq**(*src*, *tar*)

Return the longest common subsequence distance between two strings.

This is a wrapper for `LCSseq.dist()`.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns LCSseq distance

Return type float

Examples

```
>>> dist_lcsseq('cat', 'hat')
0.3333333333333333
>>> dist_lcsseq('Niall', 'Neil')
0.4
>>> dist_lcsseq('aluminum', 'Catalan')
0.625
>>> dist_lcsseq('ATCG', 'TAGC')
0.5
```

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the LCSseq.dist method instead.

abydos.distance.**sim_lcsseq**(*src*, *tar*)

Return the longest common subsequence similarity of two strings.

This is a wrapper for *LCSseq.sim()*.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns LCSseq similarity

Return type float

Examples

```
>>> sim_lcsseq('cat', 'hat')
0.6666666666666666
>>> sim_lcsseq('Niall', 'Neil')
0.6
>>> sim_lcsseq('aluminum', 'Catalan')
0.375
>>> sim_lcsseq('ATCG', 'TAGC')
0.5
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the LCSseq.sim method instead.

class abydos.distance.**LCSstr**(*normalizer=<built-in function max>*, ***kwargs*)

Bases: abydos.distance._distance._Distance

Longest common substring.

New in version 0.3.6.

Initialize LCSseq.

Parameters

- **normalizer** (*function*) -- A normalization function for the normalized similarity & distance. By default, the max of the lengths of the input strings. If lambda x: sum(x)/2.0 is supplied, the normalization proposed in [RTS+01] is used, i.e. $\frac{2|LCS(src,tar)|}{|src|+|tar|}$.
- ****kwargs** -- Arbitrary keyword arguments

New in version 0.4.0.

lcsstr(*src*, *tar*)

Return the longest common substring of two strings.

Longest common substring (LCSstr).

Based on the code from https://en.wikibooks.org/wiki/Algorithm_Implementation/Strings/Longest_common_substring [Wik18]. This is licensed Creative Commons: Attribution-ShareAlike 3.0.

Modifications include:

- conversion to a numpy array in place of a list of lists

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns The longest common substring

Return type str

Examples

```
>>> sstr = LCSstr()
>>> sstr.lcsstr('cat', 'hat')
'at'
>>> sstr.lcsstr('Niall', 'Neil')
'N'
>>> sstr.lcsstr('aluminum', 'Catalan')
'al'
>>> sstr.lcsstr('ATCG', 'TAGC')
'A'
```

New in version 0.1.0.

Changed in version 0.3.6: Encapsulated in class

sim(*src*, *tar*)

Return the longest common substring similarity of two strings.

Longest common substring similarity (sim_{LCSstr}).

This employs the LCS function to derive a similarity metric: $sim_{LCSstr}(s,t) = \frac{|LCSstr(s,t)|}{max(|s|,|t|)}$

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns LCSstr similarity

Return type float

Examples

```
>>> sstr = LCSstr()
>>> sstr.sim('cat', 'hat')
0.666666666666666
>>> sstr.sim('Niall', 'Neil')
0.2
>>> sstr.sim('aluminum', 'Catalan')
0.25
>>> sstr.sim('ATCG', 'TAGC')
0.25
```

New in version 0.1.0.

Changed in version 0.3.6: Encapsulated in class

Changed in version 0.4.0: Added normalization option

abydos.distance.**lcsstr**(*src*, *tar*)

Return the longest common substring of two strings.

This is a wrapper for *LCSstr.lcsstr()*.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns The longest common substring

Return type str

Examples

```
>>> lcsstr('cat', 'hat')
'at'
>>> lcsstr('Niall', 'Neil')
'N'
>>> lcsstr('aluminum', 'Catalan')
'al'
>>> lcsstr('ATCG', 'TAGC')
'A'
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the LCSstr.lcsstr method instead.

abydos.distance.**dist_lcsstr**(*src*, *tar*)

Return the longest common substring distance between two strings.

This is a wrapper for `LCSstr.dist()`.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns LCSstr distance

Return type float

Examples

```
>>> dist_lcsstr('cat', 'hat')
0.3333333333333333
>>> dist_lcsstr('Niall', 'Neil')
0.8
>>> dist_lcsstr('aluminum', 'Catalan')
0.75
>>> dist_lcsstr('ATCG', 'TAGC')
0.75
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the LCSstr.dist method instead.

abydos.distance.**sim_lcsstr**(*src*, *tar*)

Return the longest common substring similarity of two strings.

This is a wrapper for *LCSstr.sim()*.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns LCSstr similarity

Return type float

Examples

```
>>> sim_lcsstr('cat', 'hat')
0.6666666666666666
>>> sim_lcsstr('Niall', 'Neil')
0.2
>>> sim_lcsstr('aluminum', 'Catalan')
0.25
>>> sim_lcsstr('ATCG', 'TAGC')
0.25
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the LCSstr.sim method instead.

class abydos.distance.**LCPrefix**(***kwargs*)

Bases: abydos.distance._distance._Distance

Longest common prefix.

New in version 0.4.0.

Initialize _Distance instance.

Parameters ****kwargs** -- Arbitrary keyword arguments

New in version 0.4.0.

dist_abs(*src*, *tar*, **args*)

Return the length of the longest common prefix of the strings.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison
- ***args** (*strs*) -- Additional strings for comparison

Raises **ValueError** -- All arguments must be of type str

Returns The length of the longest common prefix

Return type int

Examples

```
>>> pfx = LCPrefix()
>>> pfx.dist_abs('cat', 'hat')
0
>>> pfx.dist_abs('Niall', 'Neil')
1
>>> pfx.dist_abs('aluminum', 'Catalan')
0
>>> pfx.dist_abs('ATCG', 'TAGC')
0
```

New in version 0.4.0.

lcprefix(*strings*)

Return the longest common prefix of a list of strings.

Longest common prefix (LCPrefix).

Parameters **strings** (*list of strings*) -- Strings for comparison

Returns The longest common prefix

Return type str

Examples

```
>>> pfx = LCPrefix()
>>> pfx.lcprefix(['cat', 'hat'])
''
>>> pfx.lcprefix(['Niall', 'Neil'])
'N'
>>> pfx.lcprefix(['aluminum', 'Catalan'])
''
>>> pfx.lcprefix(['ATCG', 'TAGC'])
''
```

New in version 0.4.0.

sim(*src*, *tar*, **args*)

Return the longest common prefix similarity of two or more strings.

Longest common prefix similarity ($sim_{LCPrefix}$).

This employs the LCPrefix function to derive a similarity metric: $sim_{LCPrefix}(s,t) = \frac{|LCPrefix(s,t)|}{max(|s|,|t|)}$

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison
- ***args** (*strs*) -- Additional strings for comparison

Returns LCPrefix similarity

Return type float

Examples

```
>>> pfx = LCPrefix()
>>> pfx.sim('cat', 'hat')
0.0
>>> pfx.sim('Niall', 'Neil')
0.2
>>> pfx.sim('aluminum', 'Catalan')
0.0
>>> pfx.sim('ATCG', 'TAGC')
0.0
```

New in version 0.4.0.

class abydos.distance.**LCSuffix**(***kwargs*)

Bases: abydos.distance._lcprefix.LCPrefix

Longest common suffix.

New in version 0.4.0.

Initialize _Distance instance.

Parameters ****kwargs** -- Arbitrary keyword arguments

New in version 0.4.0.

dist_abs(*src*, *tar*, **args*)

Return the length of the longest common suffix of the strings.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison
- ***args** (*strs*) -- Additional strings for comparison

Raises **ValueError** -- All arguments must be of type str

Returns The length of the longest common suffix

Return type int

Examples

```
>>> sfx = LCSuffix()
>>> sfx.dist_abs('cat', 'hat')
2
>>> sfx.dist_abs('Niall', 'Neil')
1
>>> sfx.dist_abs('aluminum', 'Catalan')
0
>>> sfx.dist_abs('ATCG', 'TAGC')
0
```

New in version 0.4.0.

lcsuffix(*strings*)

Return the longest common suffix of a list of strings.

Longest common suffix (LCSuffix).

Parameters **strings** (*list of strings*) -- Strings for comparison

Returns The longest common suffix

Return type str

Examples

```
>>> sfx = LCSuffix()
>>> sfx.lcsuffix(['cat', 'hat'])
'at'
>>> sfx.lcsuffix(['Niall', 'Neil'])
'l'
>>> sfx.lcsuffix(['aluminum', 'Catalan'])
''
>>> sfx.lcsuffix(['ATCG', 'TAGC'])
''
```

New in version 0.4.0.

sim(*src*, *tar*, **args*)

Return the longest common suffix similarity of two or more strings.

Longest common prefix similarity ($sim_{LCPrefix}$).

This employs the LCSuffix function to derive a similarity metric: $sim_{LCSuffix}(s,t) = \frac{|LCSuffix(s,t)|}{max(|s|,|t|)}$

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison
- ***args** (*strs*) -- Additional strings for comparison

Returns LCSuffix similarity

Return type float

Examples

```
>>> pfx = LCPrefix()
>>> pfx.sim('cat', 'hat')
0.0
>>> pfx.sim('Niall', 'Neil')
0.2
>>> pfx.sim('aluminum', 'Catalan')
0.0
>>> pfx.sim('ATCG', 'TAGC')
0.0
```

New in version 0.4.0.

class abydos.distance.**RatcliffObershelp**(***kwargs*)

Bases: abydos.distance._distance._Distance

Ratcliff-Obershelp similarity.

This follows the Ratcliff-Obershelp algorithm [RM88] to derive a similarity measure:

1. Find the length of the longest common substring in src & tar.
2. Recurse on the strings to the left & right of each this substring in src & tar. The base case is a 0 length common substring, in which case, return 0. Otherwise, return the sum of the current longest common substring and the left & right recursed sums.
3. Multiply this length by 2 and divide by the sum of the lengths of src & tar.

Cf. http://www.drdobbs.com/database/pattern-matching-the-gestalt-approach/184407970

New in version 0.3.6.

Initialize _Distance instance.

Parameters ****kwargs** -- Arbitrary keyword arguments

New in version 0.4.0.

sim(*src*, *tar*)

Return the Ratcliff-Obershelp similarity of two strings.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns Ratcliff-Obershelp similarity

Return type float

Examples

```
>>> cmp = RatcliffObershelp()
>>> round(cmp.sim('cat', 'hat'), 12)
0.666666666667
>>> round(cmp.sim('Niall', 'Neil'), 12)
0.666666666667
>>> round(cmp.sim('aluminum', 'Catalan'), 12)
0.4
>>> cmp.sim('ATCG', 'TAGC')
0.5
```

New in version 0.1.0.

Changed in version 0.3.6: Encapsulated in class

abydos.distance.**dist_ratcliff_obershelp**(*src*, *tar*)

Return the Ratcliff-Obershelp distance between two strings.

This is a wrapper for `RatcliffObershelp.dist()`.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns Ratcliff-Obershelp distance

Return type float

Examples

```
>>> round(dist_ratcliff_obershelp('cat', 'hat'), 12)
0.333333333333
>>> round(dist_ratcliff_obershelp('Niall', 'Neil'), 12)
0.333333333333
>>> round(dist_ratcliff_obershelp('aluminum', 'Catalan'), 12)
0.6
>>> dist_ratcliff_obershelp('ATCG', 'TAGC')
0.5
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the RatcliffObershelp.dist method instead.

abydos.distance.**sim_ratcliff_obershelp**(*src*, *tar*)

Return the Ratcliff-Obershelp similarity of two strings.

This is a wrapper for *RatcliffObershelp.sim()*.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns Ratcliff-Obershelp similarity

Return type float

Examples

```
>>> round(sim_ratcliff_obershelp('cat', 'hat'), 12)
0.666666666667
>>> round(sim_ratcliff_obershelp('Niall', 'Neil'), 12)
0.666666666667
>>> round(sim_ratcliff_obershelp('aluminum', 'Catalan'), 12)
0.4
>>> sim_ratcliff_obershelp('ATCG', 'TAGC')
0.5
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the RatcliffObershelp.sim method instead.

class abydos.distance.**Ident**(***kwargs*)

Bases: abydos.distance._distance._Distance

Identity distance and similarity.

New in version 0.3.6.

Initialize _Distance instance.

Parameters ****kwargs** -- Arbitrary keyword arguments

New in version 0.4.0.

sim(*src*, *tar*)

Return the identity similarity of two strings.

Identity similarity is 1.0 if the two strings are identical, otherwise 0.0

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns Identity similarity

Return type float

Examples

```
>>> cmp = Ident()
>>> cmp.sim('cat', 'hat')
0.0
>>> cmp.sim('cat', 'cat')
1.0
```

New in version 0.1.0.

Changed in version 0.3.6: Encapsulated in class

abydos.distance.**dist_ident**(*src*, *tar*)

Return the identity distance between two strings.

This is a wrapper for `Ident.dist()`.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns Identity distance

Return type float

Examples

```
>>> dist_ident('cat', 'hat')
1.0
>>> dist_ident('cat', 'cat')
0.0
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the Ident.dist method instead.

abydos.distance.**sim_ident**(*src*, *tar*)

Return the identity similarity of two strings.

This is a wrapper for *Ident.sim()*.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns Identity similarity

Return type float

Examples

```
>>> sim_ident('cat', 'hat')
0.0
>>> sim_ident('cat', 'cat')
1.0
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the Ident.sim method instead.

class abydos.distance.**Length**(***kwargs*)
Bases: abydos.distance._distance._Distance

Length similarity and distance.

New in version 0.3.6.

Initialize _Distance instance.

Parameters ****kwargs** -- Arbitrary keyword arguments

New in version 0.4.0.

sim(*src*, *tar*)
Return the length similarity of two strings.

Length similarity is the ratio of the length of the shorter string to the longer.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns Length similarity

Return type float

Examples

```
>>> cmp = Length()
>>> cmp.sim('cat', 'hat')
1.0
>>> cmp.sim('Niall', 'Neil')
0.8
>>> cmp.sim('aluminum', 'Catalan')
0.875
>>> cmp.sim('ATCG', 'TAGC')
1.0
```

New in version 0.1.0.

Changed in version 0.3.6: Encapsulated in class

abydos.distance.**dist_length**(*src*, *tar*)
Return the length distance between two strings.

This is a wrapper for Length.dist().

Parameters

- **src** (*str*) -- Source string for comparison

- **tar** (*str*) -- Target string for comparison

Returns Length distance

Return type float

Examples

```
>>> dist_length('cat', 'hat')
0.0
>>> dist_length('Niall', 'Neil')
0.19999999999999996
>>> dist_length('aluminum', 'Catalan')
0.125
>>> dist_length('ATCG', 'TAGC')
0.0
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the Length.dist method instead.

abydos.distance.**sim_length**(*src*, *tar*)

Return the length similarity of two strings.

This is a wrapper for *Length.sim()*.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns Length similarity

Return type float

Examples

```
>>> sim_length('cat', 'hat')
1.0
>>> sim_length('Niall', 'Neil')
0.8
>>> sim_length('aluminum', 'Catalan')
0.875
>>> sim_length('ATCG', 'TAGC')
1.0
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the Length.sim method instead.

class abydos.distance.**Prefix**(***kwargs*)

Bases: abydos.distance._distance._Distance

Prefix similiarity and distance.

New in version 0.3.6.

Initialize _Distance instance.

Parameters ****kwargs** -- Arbitrary keyword arguments

New in version 0.4.0.

sim(*src*, *tar*)

Return the prefix similarity of two strings.

Prefix similarity is the ratio of the length of the shorter term that exactly matches the longer term to the length of the shorter term, beginning at the start of both terms.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns Prefix similarity

Return type float

Examples

```
>>> cmp = Prefix()
>>> cmp.sim('cat', 'hat')
0.0
>>> cmp.sim('Niall', 'Neil')
0.25
>>> cmp.sim('aluminum', 'Catalan')
0.0
>>> cmp.sim('ATCG', 'TAGC')
0.0
```

New in version 0.1.0.

Changed in version 0.3.6: Encapsulated in class

abydos.distance.**dist_prefix**(*src*, *tar*)

Return the prefix distance between two strings.

This is a wrapper for `Prefix.dist()`.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns Prefix distance

Return type float

Examples

```
>>> dist_prefix('cat', 'hat')
1.0
>>> dist_prefix('Niall', 'Neil')
0.75
>>> dist_prefix('aluminum', 'Catalan')
1.0
>>> dist_prefix('ATCG', 'TAGC')
1.0
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the Prefix.dist method instead.

abydos.distance.**sim_prefix**(*src*, *tar*)

Return the prefix similarity of two strings.

This is a wrapper for *Prefix.sim()*.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns Prefix similarity

Return type float

Examples

```
>>> sim_prefix('cat', 'hat')
0.0
>>> sim_prefix('Niall', 'Neil')
0.25
>>> sim_prefix('aluminum', 'Catalan')
0.0
>>> sim_prefix('ATCG', 'TAGC')
0.0
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the Prefix.sim method instead.

class abydos.distance.**Suffix**(***kwargs*)

Bases: abydos.distance._distance._Distance

Suffix similarity and distance.

New in version 0.3.6.

Initialize _Distance instance.

Parameters ****kwargs** -- Arbitrary keyword arguments

New in version 0.4.0.

sim(*src*, *tar*)

Return the suffix similarity of two strings.

Suffix similarity is the ratio of the length of the shorter term that exactly matches the longer term to the length of the shorter term, beginning at the end of both terms.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns Suffix similarity

Return type float

Examples

```
>>> cmp = Suffix()
>>> cmp.sim('cat', 'hat')
0.6666666666666666
>>> cmp.sim('Niall', 'Neil')
0.25
>>> cmp.sim('aluminum', 'Catalan')
0.0
>>> cmp.sim('ATCG', 'TAGC')
0.0
```

New in version 0.1.0.

Changed in version 0.3.6: Encapsulated in class

abydos.distance.**dist_suffix**(*src*, *tar*)

Return the suffix distance between two strings.

This is a wrapper for `Suffix.dist()`.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns Suffix distance

Return type float

Examples

```
>>> dist_suffix('cat', 'hat')
0.33333333333333337
>>> dist_suffix('Niall', 'Neil')
0.75
>>> dist_suffix('aluminum', 'Catalan')
1.0
>>> dist_suffix('ATCG', 'TAGC')
1.0
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the Suffix.dist method instead.

abydos.distance.**sim_suffix**(*src*, *tar*)

Return the suffix similarity of two strings.

This is a wrapper for *Suffix.sim()*.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns Suffix similarity

Return type float

Examples

```
>>> sim_suffix('cat', 'hat')
0.6666666666666666
>>> sim_suffix('Niall', 'Neil')
0.25
>>> sim_suffix('aluminum', 'Catalan')
0.0
>>> sim_suffix('ATCG', 'TAGC')
0.0
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the Suffix.sim method instead.

class abydos.distance.**NCDzlib**(*level=-1*, ***kwargs*)

Bases: abydos.distance._distance._Distance

Normalized Compression Distance using zlib compression.

Cf. https://zlib.net/

Normalized compression distance (NCD) [CV05].

New in version 0.3.6.

Initialize zlib compressor.

Parameters **level** (*int*) -- The compression level (0 to 9)

New in version 0.3.6.

dist(*src*, *tar*)

Return the NCD between two strings using zlib compression.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns Compression distance

Return type float

Examples

```
>>> cmp = NCDzlib()
>>> cmp.dist('cat', 'hat')
0.3333333333333333
>>> cmp.dist('Niall', 'Neil')
0.45454545454545453
>>> cmp.dist('aluminum', 'Catalan')
0.5714285714285714
>>> cmp.dist('ATCG', 'TAGC')
0.4
```

New in version 0.3.5.

Changed in version 0.3.6: Encapsulated in class

abydos.distance.**dist_ncd_zlib**(*src*, *tar*)

Return the NCD between two strings using zlib compression.

This is a wrapper for *NCDzlib.dist()*.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns Compression distance

Return type float

Examples

```
>>> dist_ncd_zlib('cat', 'hat')
0.3333333333333333
>>> dist_ncd_zlib('Niall', 'Neil')
0.45454545454545453
>>> dist_ncd_zlib('aluminum', 'Catalan')
0.5714285714285714
>>> dist_ncd_zlib('ATCG', 'TAGC')
0.4
```

New in version 0.3.5.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the NCDzlib.dist method instead.

abydos.distance.**sim_ncd_zlib**(*src*, *tar*)

Return the NCD similarity between two strings using zlib compression.

This is a wrapper for `NCDzlib.sim()`.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns **float**

Return type Compression similarity

Examples

```
>>> sim_ncd_zlib('cat', 'hat')
0.6666666666666667
>>> sim_ncd_zlib('Niall', 'Neil')
0.5454545454545454
>>> sim_ncd_zlib('aluminum', 'Catalan')
0.4285714285714286
>>> sim_ncd_zlib('ATCG', 'TAGC')
0.6
```

New in version 0.3.5.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the NCDzlib.sim method instead.

class abydos.distance.**NCDbz2**(*level=9*, ***kwargs*)

Bases: abydos.distance._distance._Distance

Normalized Compression Distance using bzip2 compression.

Cf. https://en.wikipedia.org/wiki/Bzip2

Normalized compression distance (NCD) [CV05].

New in version 0.3.6.

Initialize bzip2 compressor.

Parameters **level** (*int*) -- The compression level (0 to 9)

New in version 0.3.6.

Changed in version 0.3.6: Encapsulated in class

dist(*src*, *tar*)

Return the NCD between two strings using bzip2 compression.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns Compression distance

Return type float

Examples

```
>>> cmp = NCDbz2()
>>> cmp.dist('cat', 'hat')
0.06666666666666667
>>> cmp.dist('Niall', 'Neil')
0.03125
>>> cmp.dist('aluminum', 'Catalan')
0.17647058823529413
>>> cmp.dist('ATCG', 'TAGC')
0.03125
```

New in version 0.3.5.

Changed in version 0.3.6: Encapsulated in class

abydos.distance.**dist_ncd_bz2**(*src*, *tar*)

Return the NCD between two strings using bzip2 compression.

This is a wrapper for *NCDbz2.dist()*.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns Compression distance

Return type float

Examples

```
>>> dist_ncd_bz2('cat', 'hat')
0.0666666666666667
>>> dist_ncd_bz2('Niall', 'Neil')
0.03125
>>> dist_ncd_bz2('aluminum', 'Catalan')
0.17647058823529413
>>> dist_ncd_bz2('ATCG', 'TAGC')
0.03125
```

New in version 0.3.5.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the NCDbz2.dist method instead.

abydos.distance.**sim_ncd_bz2**(*src*, *tar*)

Return the NCD similarity between two strings using bzip2 compression.

This is a wrapper for `NCDbz2.sim()`.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns Compression similarity

Return type float

Examples

```
>>> sim_ncd_bz2('cat', 'hat')
0.9333333333333333
>>> sim_ncd_bz2('Niall', 'Neil')
0.96875
>>> sim_ncd_bz2('aluminum', 'Catalan')
0.8235294117647058
>>> sim_ncd_bz2('ATCG', 'TAGC')
0.96875
```

New in version 0.3.5.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the NCDbz2.sim method instead.

class abydos.distance.**NCDlzma**(*level=6*, ***kwargs*)

Bases: abydos.distance._distance._Distance

Normalized Compression Distance using LZMA compression.

Cf. https://en.wikipedia.org/wiki/Lempel-Ziv-Markov_chain_algorithm

Normalized compression distance (NCD) [CV05].

New in version 0.3.6.

Initialize LZMA compressor.

Parameters **level** (*int*) -- The compression level (0 to 9)

New in version 0.5.0.

dist(*src*, *tar*)

Return the NCD between two strings using LZMA compression.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns Compression distance

Return type float

Examples

```
>>> cmp = NCDlzma()
>>> cmp.dist('cat', 'hat')
0.08695652173913043
>>> cmp.dist('Niall', 'Neil')
0.16
>>> cmp.dist('aluminum', 'Catalan')
0.16
>>> cmp.dist('ATCG', 'TAGC')
0.08695652173913043
```

New in version 0.3.5.

Changed in version 0.3.6: Encapsulated in class

abydos.distance.**dist_ncd_lzma**(*src*, *tar*)

Return the NCD between two strings using LZMA compression.

This is a wrapper for *NCDlzma.dist()*.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns Compression distance

Return type float

Examples

```
>>> dist_ncd_lzma('cat', 'hat')
0.08695652173913043
>>> dist_ncd_lzma('Niall', 'Neil')
0.16
>>> dist_ncd_lzma('aluminum', 'Catalan')
0.16
>>> dist_ncd_lzma('ATCG', 'TAGC')
0.08695652173913043
```

New in version 0.3.5.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the NCDlzma.dist method instead.

abydos.distance.**sim_ncd_lzma**(*src*, *tar*)

Return the NCD similarity between two strings using LZMA compression.

This is a wrapper for `NCDlzma.sim()`.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns Compression similarity

Return type float

Examples

```
>>> sim_ncd_lzma('cat', 'hat')
0.9130434782608696
>>> sim_ncd_lzma('Niall', 'Neil')
0.84
>>> sim_ncd_lzma('aluminum', 'Catalan')
0.84
>>> sim_ncd_lzma('ATCG', 'TAGC')
0.9130434782608696
```

New in version 0.3.5.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the NCDlzma.sim method instead.

class abydos.distance.**NCDarith**(*probs=None*, ***kwargs*)

Bases: `abydos.distance._distance._Distance`

Normalized Compression Distance using arithmetic coding.

Cf. https://en.wikipedia.org/wiki/Arithmetic_coding

Normalized compression distance (NCD) [CV05].

New in version 0.3.6.

Initialize the arithmetic coder object.

Parameters **probs** (*dict*) -- A dictionary trained with `Arithmetic.train()`

New in version 0.3.6.

Changed in version 0.3.6: Encapsulated in class

dist(*src*, *tar*)

Return the NCD between two strings using arithmetic coding.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns Compression distance

Return type float

Examples

```
>>> cmp = NCDarith()
>>> cmp.dist('cat', 'hat')
0.5454545454545454
>>> cmp.dist('Niall', 'Neil')
0.6875
>>> cmp.dist('aluminum', 'Catalan')
0.8275862068965517
>>> cmp.dist('ATCG', 'TAGC')
0.6923076923076923
```

New in version 0.3.5.

Changed in version 0.3.6: Encapsulated in class

abydos.distance.**dist_ncd_arith**(*src*, *tar*, *probs=None*)

Return the NCD between two strings using arithmetic coding.

This is a wrapper for *NCDarith.dist()*.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison
- **probs** (*dict*) -- A dictionary trained with `Arithmetic.train()`

Returns Compression distance

Return type float

Examples

```
>>> dist_ncd_arith('cat', 'hat')
0.5454545454545454
>>> dist_ncd_arith('Niall', 'Neil')
0.6875
>>> dist_ncd_arith('aluminum', 'Catalan')
0.8275862068965517
>>> dist_ncd_arith('ATCG', 'TAGC')
0.6923076923076923
```

New in version 0.3.5.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the NCDarith.dist method instead.

abydos.distance.**sim_ncd_arith**(*src*, *tar*, *probs=None*)

Return the NCD similarity between two strings using arithmetic coding.

This is a wrapper for `NCDarith.sim()`.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison
- **probs** (*dict*) -- A dictionary trained with `Arithmetic.train()`

Returns Compression similarity

Return type float

Examples

```
>>> sim_ncd_arith('cat', 'hat')
0.4545454545454546
>>> sim_ncd_arith('Niall', 'Neil')
0.3125
>>> sim_ncd_arith('aluminum', 'Catalan')
0.1724137931034483
>>> sim_ncd_arith('ATCG', 'TAGC')
0.3076923076923077
```

New in version 0.3.5.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the NCDarith.sim method instead.

class abydos.distance.**NCDbwtrle**(***kwargs*)

Bases: abydos.distance._ncd_rle.NCDrle

Normalized Compression Distance using BWT plus RLE.

Cf. https://en.wikipedia.org/wiki/Burrows-Wheeler_transform

Normalized compression distance (NCD) [CV05].

New in version 0.3.6.

Initialize _Distance instance.

Parameters ****kwargs** -- Arbitrary keyword arguments

New in version 0.4.0.

dist(*src*, *tar*)

Return the NCD between two strings using BWT plus RLE.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns Compression distance

Return type float

Examples

```
>>> cmp = NCDbwtrle()
>>> cmp.dist('cat', 'hat')
0.75
>>> cmp.dist('Niall', 'Neil')
0.8333333333333334
>>> cmp.dist('aluminum', 'Catalan')
1.0
>>> cmp.dist('ATCG', 'TAGC')
0.8
```

New in version 0.3.5.

Changed in version 0.3.6: Encapsulated in class

abydos.distance.**dist_ncd_bwtrle**(*src*, *tar*)

Return the NCD between two strings using BWT plus RLE.

This is a wrapper for *NCDbwtrle.dist()*.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns Compression distance

Return type float

Examples

```
>>> dist_ncd_bwtrle('cat', 'hat')
0.75
>>> dist_ncd_bwtrle('Niall', 'Neil')
0.8333333333333334
>>> dist_ncd_bwtrle('aluminum', 'Catalan')
1.0
>>> dist_ncd_bwtrle('ATCG', 'TAGC')
0.8
```

New in version 0.3.5.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the NCDbwtrle.dist method instead.

abydos.distance.**sim_ncd_bwtrle**(*src*, *tar*)

Return the NCD similarity between two strings using BWT plus RLE.

This is a wrapper for `NCDbwtrle.sim()`.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns Compression similarity

Return type float

Examples

```
>>> sim_ncd_bwtrle('cat', 'hat')
0.25
>>> sim_ncd_bwtrle('Niall', 'Neil')
0.1666666666666663
>>> sim_ncd_bwtrle('aluminum', 'Catalan')
0.0
>>> sim_ncd_bwtrle('ATCG', 'TAGC')
0.19999999999999996
```

New in version 0.3.5.

class abydos.distance.**NCDrle**(***kwargs*)

Bases: abydos.distance._distance._Distance

Normalized Compression Distance using RLE.

Cf. https://en.wikipedia.org/wiki/Run-length_encoding

Normalized compression distance (NCD) [CV05].

New in version 0.3.6.

Initialize _Distance instance.

Parameters ****kwargs** -- Arbitrary keyword arguments

New in version 0.4.0.

dist(*src*, *tar*)

Return the NCD between two strings using RLE.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns Compression distance

Return type float

Examples

```
>>> cmp = NCDrle()
>>> cmp.dist('cat', 'hat')
1.0
>>> cmp.dist('Niall', 'Neil')
1.0
>>> cmp.dist('aluminum', 'Catalan')
1.0
>>> cmp.dist('ATCG', 'TAGC')
1.0
```

New in version 0.3.5.

Changed in version 0.3.6: Encapsulated in class

abydos.distance.**dist_ncd_rle**(*src*, *tar*)

Return the NCD between two strings using RLE.

This is a wrapper for *NCDrle.dist()*.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns Compression distance

Return type float

Examples

```
>>> dist_ncd_rle('cat', 'hat')
1.0
>>> dist_ncd_rle('Niall', 'Neil')
1.0
>>> dist_ncd_rle('aluminum', 'Catalan')
1.0
>>> dist_ncd_rle('ATCG', 'TAGC')
1.0
```

New in version 0.3.5.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the NCDrle.dist method instead.

abydos.distance.**sim_ncd_rle**(*src*, *tar*)

Return the NCD similarity between two strings using RLE.

This is a wrapper for `NCDrle.sim()`.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns Compression similarity

Return type float

Examples

```
>>> sim_ncd_rle('cat', 'hat')
0.0
>>> sim_ncd_rle('Niall', 'Neil')
0.0
>>> sim_ncd_rle('aluminum', 'Catalan')
0.0
>>> sim_ncd_rle('ATCG', 'TAGC')
0.0
```

New in version 0.3.5.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the NCDrle.sim method instead.

class abydos.distance.**NCDpaq9a**(***kwargs*)

Bases: abydos.distance._distance._Distance

Normalized Compression Distance using PAQ9A compression.

Cf. http://mattmahoney.net/dc/#paq9a

Normalized compression distance (NCD) [CV05].

New in version 0.4.0.

Initialize _Distance instance.

Parameters ****kwargs** -- Arbitrary keyword arguments

New in version 0.4.0.

dist(*src*, *tar*)

Return the NCD between two strings using PAQ9A compression.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns Compression distance

Return type float

Raises **ValueError** -- Install the paq module in order to use PAQ9A

Examples

```
>>> cmp = NCDpaq9a()
>>> cmp.dist('cat', 'hat')
0.42857142857142855
>>> cmp.dist('Niall', 'Neil')
0.5555555555555556
>>> cmp.dist('aluminum', 'Catalan')
0.5833333333333334
>>> cmp.dist('ATCG', 'TAGC')
0.5
```

New in version 0.4.0.

class abydos.distance.**NCDlzss**(***kwargs*)

Bases: abydos.distance._distance._Distance

Normalized Compression Distance using LZSS compression.

Cf. https://en.wikipedia.org/wiki/Lempel-Ziv-Storer-Szymanski

Normalized compression distance (NCD) [CV05].

New in version 0.4.0.

Initialize _Distance instance.

Parameters ****kwargs** -- Arbitrary keyword arguments

New in version 0.4.0.

dist(*src*, *tar*)

Return the NCD between two strings using LZSS compression.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns Compression distance

Return type float

Raises **ValueError** -- Install the PyLZSS module in order to use LZSS

Examples

```
>>> cmp = NCDlzss()
>>> cmp.dist('cat', 'hat')
0.75
>>> cmp.dist('Niall', 'Neil')
1.0
>>> cmp.dist('aluminum', 'Catalan')
1.0
>>> cmp.dist('ATCG', 'TAGC')
0.8
```

New in version 0.4.0.

class abydos.distance.**FuzzyWuzzyPartialString**(***kwargs*)

Bases: abydos.distance._distance._Distance

FuzzyWuzzy Partial String similarity.

This follows the FuzzyWuzzy Partial String similarity algorithm [Coh11]. Rather than returning an integer in the range [0, 100], as demonstrated in the blog post, this implementation returns a float in the range [0.0, 1.0].

New in version 0.4.0.

Initialize _Distance instance.

Parameters ****kwargs** -- Arbitrary keyword arguments

New in version 0.4.0.

sim(*src*, *tar*)

Return the FuzzyWuzzy Partial String similarity of two strings.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns FuzzyWuzzy Partial String similarity

Return type float

Examples

```
>>> cmp = FuzzyWuzzyPartialString()
>>> round(cmp.sim('cat', 'hat'), 12)
0.666666666667
>>> round(cmp.sim('Niall', 'Neil'), 12)
0.75
>>> round(cmp.sim('aluminum', 'Catalan'), 12)
0.428571428571
>>> cmp.sim('ATCG', 'TAGC')
0.5
```

New in version 0.4.0.

class abydos.distance.**FuzzyWuzzyTokenSort**(*tokenizer=None*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

FuzzyWuzzy Token Sort similarity.

This follows the FuzzyWuzzy Token Sort similarity algorithm [Coh11]. Rather than returning an integer in the range [0, 100], as demonstrated in the blog post, this implementation returns a float in the range [0.0, 1.0].

New in version 0.4.0.

Initialize FuzzyWuzzyTokenSort instance.

Parameters

- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package. By default, the regexp tokenizer is employed, matching only letters.
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.

New in version 0.4.0.

sim(*src*, *tar*)

Return the FuzzyWuzzy Token Sort similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns FuzzyWuzzy Token Sort similarity

Return type float

Examples

```
>>> cmp = FuzzyWuzzyTokenSort()
>>> cmp.sim('cat', 'hat')
0.666666666666666
>>> cmp.sim('Niall', 'Neil')
0.666666666666666
>>> cmp.sim('aluminum', 'Catalan')
0.4
>>> cmp.sim('ATCG', 'TAGC')
0.5
```

New in version 0.4.0.

class abydos.distance.**FuzzyWuzzyTokenSet**(*tokenizer=None*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

FuzzyWuzzy Token Set similarity.

This follows the FuzzyWuzzy Token Set similarity algorithm [Coh11]. Rather than returning an integer in the range [0, 100], as demonstrated in the blog post, this implementation returns a float in the range [0.0, 1.0]. Distinct from the

New in version 0.4.0.

Initialize FuzzyWuzzyTokenSet instance.

Parameters

- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package. By default, the regexp tokenizer is employed, matching only letters.

- ****kwargs** -- Arbitrary keyword arguments

Other Parameters **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.

New in version 0.4.0.

sim(*src*, *tar*)

Return the FuzzyWuzzy Token Set similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns FuzzyWuzzy Token Set similarity

Return type float

Examples

```
>>> cmp = FuzzyWuzzyTokenSet()
>>> cmp.sim('cat', 'hat')
0.75
>>> cmp.sim('Niall', 'Neil')
0.7272727272727273
>>> cmp.sim('aluminum', 'Catalan')
0.47058823529411764
>>> cmp.sim('ATCG', 'TAGC')
0.6
```

New in version 0.4.0.

class abydos.distance.**PhoneticDistance**(*transforms=None*, *metric=None*, *encode_alpha=False*, ***kwargs*)

Bases: abydos.distance._distance._Distance

Phonetic distance.

Phonetic distance applies one or more supplied string transformations to words and compares the resulting transformed strings using a supplied distance measure.

A simple example would be to create a 'Soundex distance':

```
>>> from abydos.phonetic import Soundex
>>> soundex = PhoneticDistance(transforms=Soundex())
>>> soundex.dist('Ashcraft', 'Ashcroft')
0.0
>>> soundex.dist('Robert', 'Ashcraft')
1.0
```

New in version 0.4.1.

Initialize PhoneticDistance instance.

Parameters

- **transforms** (*list or _Phonetic or _Stemmer or _Fingerprint or type*) -- An instance of a subclass of _Phonetic, _Stemmer, or _Fingerprint, or a list (or other iterable) of such instances to apply to each input word before computing their distance or similarity. If omitted, no transformations will be performed.

- **metric** (*_Distance or type*) -- An instance of a subclass of _Distance, used for computing the inputs' distance or similarity after being transformed. If omitted, the strings will be compared for identify (returning 0.0 if identical, otherwise 1.0, when distance is computed).
- **encode_alpha** (*bool*) -- Set to true to use the encode_alpha method of phonetic algoritms whenever possible.
- ****kwargs** -- Arbitrary keyword arguments

New in version 0.4.1.

dist (*src*, *tar*)
Return the normalized Phonetic distance.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns The normalized Phonetic distance

Return type float

Examples

```
>>> from abydos.phonetic import Soundex
>>> cmp = PhoneticDistance(Soundex())
>>> cmp.dist('cat', 'hat')
1.0
>>> cmp.dist('Niall', 'Neil')
0.0
>>> cmp.dist('Colin', 'Cuilen')
0.0
>>> cmp.dist('ATCG', 'TAGC')
1.0
```

```
>>> from abydos.distance import Levenshtein
>>> cmp = PhoneticDistance(transforms=[Soundex], metric=Levenshtein)
>>> cmp.dist('cat', 'hat')
0.25
>>> cmp.dist('Niall', 'Neil')
0.0
>>> cmp.dist('Colin', 'Cuilen')
0.0
>>> cmp.dist('ATCG', 'TAGC')
0.75
```

New in version 0.4.1.

dist_abs (*src*, *tar*)
Return the Phonetic distance.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns The Phonetic distance

Return type float or int

Examples

```
>>> from abydos.phonetic import Soundex
>>> cmp = PhoneticDistance(Soundex())
>>> cmp.dist_abs('cat', 'hat')
1
>>> cmp.dist_abs('Niall', 'Neil')
0
>>> cmp.dist_abs('Colin', 'Cuilen')
0
>>> cmp.dist_abs('ATCG', 'TAGC')
1
```

```
>>> from abydos.distance import Levenshtein
>>> cmp = PhoneticDistance(transforms=[Soundex], metric=Levenshtein)
>>> cmp.dist_abs('cat', 'hat')
1
>>> cmp.dist_abs('Niall', 'Neil')
0
>>> cmp.dist_abs('Colin', 'Cuilen')
0
>>> cmp.dist_abs('ATCG', 'TAGC')
3
```

New in version 0.4.1.

class abydos.distance.**MRA**(***kwargs*)

Bases: abydos.distance._distance._Distance

Match Rating Algorithm comparison rating.

The Western Airlines Surname Match Rating Algorithm comparison rating, as presented on page 18 of [MKTM77].

New in version 0.3.6.

Initialize _Distance instance.

Parameters ****kwargs** -- Arbitrary keyword arguments

New in version 0.4.0.

dist_abs(*src*, *tar*)

Return the MRA comparison rating of two strings.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns MRA comparison rating

Return type int

Examples

```
>>> cmp = MRA()
>>> cmp.dist_abs('cat', 'hat')
5
>>> cmp.dist_abs('Niall', 'Neil')
6
>>> cmp.dist_abs('aluminum', 'Catalan')
0
>>> cmp.dist_abs('ATCG', 'TAGC')
5
```

New in version 0.1.0.

Changed in version 0.3.6: Encapsulated in class

sim(*src*, *tar*)

Return the normalized MRA similarity of two strings.

This is the MRA normalized to $[0, 1]$, given that MRA itself is constrained to the range $[0, 6]$.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns Normalized MRA similarity

Return type float

Examples

```
>>> cmp = MRA()
>>> cmp.sim('cat', 'hat')
0.8333333333333334
>>> cmp.sim('Niall', 'Neil')
1.0
>>> cmp.sim('aluminum', 'Catalan')
0.0
>>> cmp.sim('ATCG', 'TAGC')
0.8333333333333334
```

New in version 0.1.0.

Changed in version 0.3.6: Encapsulated in class

abydos.distance.**mra_compare**(*src*, *tar*)

Return the MRA comparison rating of two strings.

This is a wrapper for *MRA.dist_abs()*.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns MRA comparison rating

Return type int

Examples

```
>>> mra_compare('cat', 'hat')
5
>>> mra_compare('Niall', 'Neil')
6
>>> mra_compare('aluminum', 'Catalan')
0
>>> mra_compare('ATCG', 'TAGC')
5
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the MRA.dist_abs method instead.

abydos.distance.**dist_mra**(*src*, *tar*)

Return the normalized MRA distance between two strings.

This is a wrapper for `MRA.dist()`.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns Normalized MRA distance

Return type float

Examples

```
>>> dist_mra('cat', 'hat')
0.1666666666666663
>>> dist_mra('Niall', 'Neil')
0.0
>>> dist_mra('aluminum', 'Catalan')
1.0
>>> dist_mra('ATCG', 'TAGC')
0.1666666666666663
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the MRA.dist method instead.

abydos.distance.**sim_mra**(*src*, *tar*)

Return the normalized MRA similarity of two strings.

This is a wrapper for *MRA.sim()*.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns Normalized MRA similarity

Return type float

Examples

```
>>> sim_mra('cat', 'hat')
0.8333333333333334
>>> sim_mra('Niall', 'Neil')
1.0
>>> sim_mra('aluminum', 'Catalan')
0.0
>>> sim_mra('ATCG', 'TAGC')
0.8333333333333334
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the MRA.sim method instead.

class abydos.distance.**Editex**(*cost=(0, 1, 2)*, *local=False*, *taper=False*, ***kwargs*)

Bases: abydos.distance._distance._Distance

Editex.

As described on pages 3 & 4 of [ZD96].

The local variant is based on [RU09].

New in version 0.3.6.

Changed in version 0.4.0: Added taper option

Initialize Editex instance.

Parameters

- **cost** (*tuple*) -- A 3-tuple representing the cost of the four possible edits: match, same-group, and mismatch respectively (by default: (0, 1, 2))
- **local** (*bool*) -- If True, the local variant of Editex is used
- **taper** (*bool*) -- Enables cost tapering. Following [ZD96], it causes edits at the start of the string to "just [exceed] twice the minimum penalty for replacement or deletion at the end of the string".
- ****kwargs** -- Arbitrary keyword arguments

New in version 0.4.0.

dist(*src*, *tar*)

Return the normalized Editex distance between two strings.

The Editex distance is normalized by dividing the Editex distance (calculated by any of the three supported methods) by the greater of the number of characters in src times the cost of a delete and the number of characters in tar times the cost of an insert. For the case in which all operations have $cost = 1$, this is equivalent to the greater of the length of the two strings src & tar.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns Normalized Editex distance

Return type int

Examples

```
>>> cmp = Editex()
>>> round(cmp.dist('cat', 'hat'), 12)
0.333333333333
>>> round(cmp.dist('Niall', 'Neil'), 12)
0.2
>>> cmp.dist('aluminum', 'Catalan')
0.75
>>> cmp.dist('ATCG', 'TAGC')
0.75
```

New in version 0.1.0.

Changed in version 0.3.6: Encapsulated in class

dist_abs(*src*, *tar*)

Return the Editex distance between two strings.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns Editex distance

Return type int

Examples

```
>>> cmp = Editex()
>>> cmp.dist_abs('cat', 'hat')
2
>>> cmp.dist_abs('Niall', 'Neil')
2
>>> cmp.dist_abs('aluminum', 'Catalan')
12
>>> cmp.dist_abs('ATCG', 'TAGC')
6
```

New in version 0.1.0.

Changed in version 0.3.6: Encapsulated in class

abydos.distance.**editex**(*src*, *tar*, *cost=(0, 1, 2)*, *local=False*)

Return the Editex distance between two strings.

This is a wrapper for *Editex.dist_abs()*.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison
- **cost** (*tuple*) -- A 3-tuple representing the cost of the four possible edits: match, same-group, and mismatch respectively (by default: (0, 1, 2))
- **local** (*bool*) -- If True, the local variant of Editex is used

Returns Editex distance

Return type int

Examples

```
>>> editex('cat', 'hat')
2
>>> editex('Niall', 'Neil')
2
>>> editex('aluminum', 'Catalan')
12
>>> editex('ATCG', 'TAGC')
6
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the Editex.dist_abs method instead.

abydos.distance.**dist_editex**(*src*, *tar*, *cost=(0, 1, 2)*, *local=False*)

Return the normalized Editex distance between two strings.

This is a wrapper for *Editex.dist()*.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison
- **cost** (*tuple*) -- A 3-tuple representing the cost of the four possible edits: match, same-group, and mismatch respectively (by default: (0, 1, 2))
- **local** (*bool*) -- If True, the local variant of Editex is used

Returns Normalized Editex distance

Return type int

Examples

```
>>> round(dist_editex('cat', 'hat'), 12)
0.333333333333
>>> round(dist_editex('Niall', 'Neil'), 12)
0.2
>>> dist_editex('aluminum', 'Catalan')
0.75
>>> dist_editex('ATCG', 'TAGC')
0.75
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the Editex.dist method instead.

abydos.distance.**sim_editex**(*src*, *tar*, *cost=(0, 1, 2)*, *local=False*)

Return the normalized Editex similarity of two strings.

This is a wrapper for `Editex.sim()`.

Parameters

- **src** (*str*) -- Source string for comparison

- **tar** (*str*) -- Target string for comparison
- **cost** (*tuple*) -- A 3-tuple representing the cost of the four possible edits: match, same-group, and mismatch respectively (by default: (0, 1, 2))
- **local** (*bool*) -- If True, the local variant of Editex is used

Returns Normalized Editex similarity

Return type int

Examples

```
>>> round(sim_editex('cat', 'hat'), 12)
0.666666666667
>>> round(sim_editex('Niall', 'Neil'), 12)
0.8
>>> sim_editex('aluminum', 'Catalan')
0.25
>>> sim_editex('ATCG', 'TAGC')
0.25
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the Editex.sim method instead.

class abydos.distance.**Baystat**(*min_ss_len=None*, *left_ext=None*, *right_ext=None*, ***kwargs*)
Bases: abydos.distance._distance._Distance

Baystat similarity and distance.

Good results for shorter words are reported when setting min_ss_len to 1 and either left_ext OR right_ext to 1.

The Baystat similarity is defined in [FurnrohrRvR02].

This is ostensibly a port of the R module PPRL's implementation: https://github.com/cran/PPRL/blob/master/src/MTB_Baystat.cpp [Ruk18]. As such, this could be made more pythonic.

New in version 0.3.6.

Initialize Levenshtein instance.

Parameters

- **min_ss_len** (*int*) -- Minimum substring length to be considered
- **left_ext** (*int*) -- Left-side extension length
- **right_ext** (*int*) -- Right-side extension length
- ****kwargs** -- Arbitrary keyword arguments

New in version 0.4.0.

sim(*src*, *tar*)
Return the Baystat similarity.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns The Baystat similarity

Return type float

Examples

```
>>> cmp = Baystat()
>>> round(cmp.sim('cat', 'hat'), 12)
0.666666666667
>>> cmp.sim('Niall', 'Neil')
0.4
>>> round(cmp.sim('Colin', 'Cuilen'), 12)
0.166666666667
>>> cmp.sim('ATCG', 'TAGC')
0.0
```

New in version 0.3.0.

Changed in version 0.3.6: Encapsulated in class

abydos.distance.**dist_baystat**(*src*, *tar*, *min_ss_len=None*, *left_ext=None*, *right_ext=None*)

Return the Baystat distance.

This is a wrapper for `Baystat.dist()`.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison
- **min_ss_len** (*int*) -- Minimum substring length to be considered
- **left_ext** (*int*) -- Left-side extension length
- **right_ext** (*int*) -- Right-side extension length

Returns The Baystat distance

Return type float

Examples

```
>>> round(dist_baystat('cat', 'hat'), 12)
0.333333333333
>>> dist_baystat('Niall', 'Neil')
0.6
>>> round(dist_baystat('Colin', 'Cuilen'), 12)
0.833333333333
>>> dist_baystat('ATCG', 'TAGC')
1.0
```

New in version 0.3.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the Baystat.dist method instead.

abydos.distance.**sim_baystat**(*src*, *tar*, *min_ss_len=None*, *left_ext=None*, *right_ext=None*)

Return the Baystat similarity.

This is a wrapper for *Baystat.sim()*.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

- **min_ss_len** (*int*) -- Minimum substring length to be considered
- **left_ext** (*int*) -- Left-side extension length
- **right_ext** (*int*) -- Right-side extension length

Returns The Baystat similarity

Return type float

Examples

```
>>> round(sim_baystat('cat', 'hat'), 12)
0.666666666667
>>> sim_baystat('Niall', 'Neil')
0.4
>>> round(sim_baystat('Colin', 'Cuilen'), 12)
0.166666666667
>>> sim_baystat('ATCG', 'TAGC')
0.0
```

New in version 0.3.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the Baystat.sim method instead.

class abydos.distance.**Eudex**(*weights='exponential'*, *max_length=8*, ***kwargs*)
Bases: abydos.distance._distance._Distance

Distance between the Eudex hashes of two terms.

Cf. [Tic].

New in version 0.3.6.

Initialize Eudex instance.

Parameters

- **weights** (*str, iterable, or generator function*) -- The weights or weights generator function
  - If set to None, a simple Hamming distance is calculated.
  - If set to exponential, weight decays by powers of 2, as proposed in the eudex specification: https://github.com/ticki/eudex.
  - If set to fibonacci, weight decays through the Fibonacci series, as in the eudex reference implementation.
  - If set to a callable function, this assumes it creates a generator and the generator is used to populate a series of weights.
  - If set to an iterable, the iterable's values should be integers and will be used as the weights.

  In all cases, the weights should be ordered or generated from least significant to most significant, so larger values should generally come first.
- **max_length** (*int*) -- The number of characters to encode as a eudex hash
- ****kwargs** -- Arbitrary keyword arguments

New in version 0.4.0.

dist (*src*, *tar*)

Return normalized distance between the Eudex hashes of two terms.

This is Eudex distance normalized to [0, 1].

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns The normalized Eudex Hamming distance

Return type int

Examples

```
>>> cmp = Eudex()
>>> round(cmp.dist('cat', 'hat'), 12)
0.062745098039
>>> round(cmp.dist('Niall', 'Neil'), 12)
0.000980392157
>>> round(cmp.dist('Colin', 'Cuilen'), 12)
0.004901960784
>>> round(cmp.dist('ATCG', 'TAGC'), 12)
0.197549019608
```

New in version 0.3.0.

Changed in version 0.3.6: Encapsulated in class

dist_abs (*src*, *tar*, *normalized=False*)

Calculate the distance between the Eudex hashes of two terms.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison
- **normalized** (*bool*) -- Normalizes to [0, 1] if True

Returns The Eudex Hamming distance

Return type int

Examples

```
>>> cmp = Eudex()
>>> cmp.dist_abs('cat', 'hat')
128
>>> cmp.dist_abs('Niall', 'Neil')
2
>>> cmp.dist_abs('Colin', 'Cuilen')
10
>>> cmp.dist_abs('ATCG', 'TAGC')
403
```

```
>>> cmp = Eudex(weights='fibonacci')
>>> cmp.dist_abs('cat', 'hat')
34
>>> cmp.dist_abs('Niall', 'Neil')
2
>>> cmp.dist_abs('Colin', 'Cuilen')
7
>>> cmp.dist_abs('ATCG', 'TAGC')
117
```

```
>>> cmp = Eudex(weights=None)
>>> cmp.dist_abs('cat', 'hat')
1
>>> cmp.dist_abs('Niall', 'Neil')
1
>>> cmp.dist_abs('Colin', 'Cuilen')
2
>>> cmp.dist_abs('ATCG', 'TAGC')
9
```

```
>>> # Using the OEIS A000142:
>>> cmp = Eudex(weights=[1, 1, 2, 6, 24, 120, 720, 5040])
>>> cmp.dist_abs('cat', 'hat')
5040
>>> cmp.dist_abs('Niall', 'Neil')
1
>>> cmp.dist_abs('Colin', 'Cuilen')
7
>>> cmp.dist_abs('ATCG', 'TAGC')
15130
```

New in version 0.3.0.

Changed in version 0.3.6: Encapsulated in class

static gen_exponential(*base=2*)

Yield the next value in an exponential series of the base.

Starts at base**0

Parameters **base** (*int*) -- The base to exponentiate

Yields *int* -- The next power of *base*

New in version 0.3.0.

Changed in version 0.3.6: Encapsulated in class

static gen_fibonacci()

Yield the next Fibonacci number.

Based on https://www.python-course.eu/generators.php Starts at Fibonacci number 3 (the second 1)

Yields *int* -- The next Fibonacci number

New in version 0.3.0.

Changed in version 0.3.6: Encapsulated in class

abydos.distance.**eudex_hamming**(*src*, *tar*, *weights='exponential'*, *max_length=8*, *normalized=False*)

Calculate the Hamming distance between the Eudex hashes of two terms.

This is a wrapper for `Eudex.eudex_hamming()`.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison
- **weights** (*str, iterable, or generator function*) -- The weights or weights generator function
- **max_length** (*int*) -- The number of characters to encode as a eudex hash
- **normalized** (*bool*) -- Normalizes to [0, 1] if True

Returns The Eudex Hamming distance

Return type int

Examples

```
>>> eudex_hamming('cat', 'hat')
128
>>> eudex_hamming('Niall', 'Neil')
2
>>> eudex_hamming('Colin', 'Cuilen')
10
>>> eudex_hamming('ATCG', 'TAGC')
403
```

```
>>> eudex_hamming('cat', 'hat', weights='fibonacci')
34
>>> eudex_hamming('Niall', 'Neil', weights='fibonacci')
2
>>> eudex_hamming('Colin', 'Cuilen', weights='fibonacci')
7
>>> eudex_hamming('ATCG', 'TAGC', weights='fibonacci')
117
```

```
>>> eudex_hamming('cat', 'hat', weights=None)
1
>>> eudex_hamming('Niall', 'Neil', weights=None)
1
>>> eudex_hamming('Colin', 'Cuilen', weights=None)
2
>>> eudex_hamming('ATCG', 'TAGC', weights=None)
9
```

```
>>> # Using the OEIS A000142:
>>> eudex_hamming('cat', 'hat', [1, 1, 2, 6, 24, 120, 720, 5040])
5040
>>> eudex_hamming('Niall', 'Neil', [1, 1, 2, 6, 24, 120, 720, 5040])
1
>>> eudex_hamming('Colin', 'Cuilen', [1, 1, 2, 6, 24, 120, 720, 5040])
7
>>> eudex_hamming('ATCG', 'TAGC', [1, 1, 2, 6, 24, 120, 720, 5040])
15130
```

New in version 0.3.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the Eudex.dist_abs method instead.

abydos.distance.**dist_eudex**(*src*, *tar*, *weights='exponential'*, *max_length=8*)

Return normalized Hamming distance between Eudex hashes of two terms.

This is a wrapper for *Eudex.dist()*.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison
- **weights** (*str, iterable, or generator function*) -- The weights or weights generator function
- **max_length** (*int*) -- The number of characters to encode as a eudex hash

Returns The normalized Eudex Hamming distance

Return type int

Examples

```
>>> round(dist_eudex('cat', 'hat'), 12)
0.062745098039
>>> round(dist_eudex('Niall', 'Neil'), 12)
0.000980392157
>>> round(dist_eudex('Colin', 'Cuilen'), 12)
0.004901960784
>>> round(dist_eudex('ATCG', 'TAGC'), 12)
0.197549019608
```

New in version 0.3.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the Eudex.dist method instead.

abydos.distance.**sim_eudex**(*src*, *tar*, *weights='exponential'*, *max_length=8*)

Return normalized Hamming similarity between Eudex hashes of two terms.

This is a wrapper for `Eudex.sim()`.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison
- **weights** (*str, iterable, or generator function*) -- The weights or weights generator function
- **max_length** (*int*) -- The number of characters to encode as a eudex hash

Returns The normalized Eudex Hamming similarity

Return type int

Examples

```
>>> round(sim_eudex('cat', 'hat'), 12)
0.937254901961
>>> round(sim_eudex('Niall', 'Neil'), 12)
0.999019607843
>>> round(sim_eudex('Colin', 'Cuilen'), 12)
0.995098039216
>>> round(sim_eudex('ATCG', 'TAGC'), 12)
0.802450980392
```

New in version 0.3.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the Eudex.sim method instead.

class abydos.distance.**Sift4**(*max_offset=5*, *max_distance=0*, ***kwargs*)
Bases: abydos.distance._distance._Distance

Sift4 Common version.

This is an approximation of edit distance, described in [Zac14].

New in version 0.3.6.

Initialize Sift4 instance.

Parameters

- **max_offset** (*int*) -- The number of characters to search for matching letters
- **max_distance** (*int*) -- The distance at which to stop and exit
- ****kwargs** -- Arbitrary keyword arguments

New in version 0.4.0.

dist(*src*, *tar*)
Return the normalized "common" Sift4 distance between two terms.

This is Sift4 distance, normalized to [0, 1].

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns The normalized Sift4 distance

Return type float

Examples

```
>>> cmp = Sift4()
>>> round(cmp.dist('cat', 'hat'), 12)
0.333333333333
>>> cmp.dist('Niall', 'Neil')
0.4
>>> cmp.dist('Colin', 'Cuilen')
0.5
>>> cmp.dist('ATCG', 'TAGC')
0.5
```

New in version 0.3.0.

Changed in version 0.3.6: Encapsulated in class

dist_abs(*src*, *tar*)
Return the "common" Sift4 distance between two terms.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns The Sift4 distance according to the common formula

Return type int

Examples

```
>>> cmp = Sift4()
>>> cmp.dist_abs('cat', 'hat')
1
>>> cmp.dist_abs('Niall', 'Neil')
2
>>> cmp.dist_abs('Colin', 'Cuilen')
3
>>> cmp.dist_abs('ATCG', 'TAGC')
2
```

New in version 0.3.0.

Changed in version 0.3.6: Encapsulated in class

class abydos.distance.**Sift4Simplest**(*max_offset=5*, ***kwargs*)
Bases: abydos.distance._sift4.Sift4

Sift4 Simplest version.

This is an approximation of edit distance, described in [Zac14].

New in version 0.3.6.

Initialize Sift4Simplest instance.

Parameters

- **max_offset** (*int*) -- The number of characters to search for matching letters
- ****kwargs** -- Arbitrary keyword arguments

New in version 0.4.0.

dist_abs(*src*, *tar*)
Return the "simplest" Sift4 distance between two terms.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns The Sift4 distance according to the simplest formula

Return type int

Examples

```
>>> cmp = Sift4Simplest()
>>> cmp.dist_abs('cat', 'hat')
1
>>> cmp.dist_abs('Niall', 'Neil')
2
>>> cmp.dist_abs('Colin', 'Cuilen')
3
>>> cmp.dist_abs('ATCG', 'TAGC')
2
```

New in version 0.3.0.

Changed in version 0.3.6: Encapsulated in class

class abydos.distance.**Sift4Extended**(*max_offset=5*, *max_distance=0*, *tokenizer=None*, *token_matcher=None*, *matching_evaluator=None*, *local_length_evaluator=None*, *transposition_cost_evaluator=None*, *transpositions_evaluator=None*, ***kwargs*)

Bases: abydos.distance._distance._Distance

Sift4 Extended version.

This is an approximation of edit distance, described in [Zac14].

New in version 0.4.0.

Initialize Sift4Extended instance.

Parameters

- **max_offset** (*int*) -- The number of characters to search for matching letters
- **max_distance** (*int*) -- The distance at which to stop and exit
- **tokenizer** (*_Tokenizer*) -- A tokenizer instance (character tokenization by default)
- **token_matcher** (*function*) -- A token matcher function of two parameters (equality by default). $Sift4Extended.sift4_token_matcher$ is also supplied.
- **matching_evaluator** (*function*) -- A token match quality function of two parameters (1 by default). $Sift4Extended.sift4_matching_evaluator$ is also supplied.
- **local_length_evaluator** (*function*) -- A local length evaluator function (its single parameter by default). $Sift4Extended.reward_length_evaluator$ and $Sift4Extended.reward_length_evaluator_exp$ are also supplied.
- **transposition_cost_evaluator** (*function*) -- A transposition cost evaluator function of two parameters (1 by default). $Sift4Extended.longer_transpositions_are_more_costly$ is also supplied.
- **transpositions_evaluator** (*function*) -- A transpositions evaluator function of two parameters (the second parameter subtracted from the first, by default).
- ****kwargs** -- Arbitrary keyword arguments

New in version 0.4.0.

dist_abs(*src*, *tar*)

Return the Sift4 Extended distance between two strings.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns The Sift4 distance according to the extended formula

Return type int

Examples

```
>>> cmp = Sift4Extended()
>>> cmp.dist_abs('cat', 'hat')
1
>>> cmp.dist_abs('Niall', 'Neil')
2
>>> cmp.dist_abs('aluminum', 'Catalan')
5
>>> cmp.dist_abs('ATCG', 'TAGC')
2
```

New in version 0.4.0.

static longer_transpositions_are_more_costly(*pos1*, *pos2*)
Longer Transpositions Are More Costly.

Parameters

- **pos1** (*int*) -- The position of the first transposition
- **pos2** (*int*) -- The position of the second transposition

Returns

- *float* -- A cost that grows as difference in the positions increases
- *.. versionadded:: 0.4.0*

static reward_length_evaluator(*length*)
Reward Length Evaluator.

Parameters **length** (*int*) -- The length of a local match

Returns

- *float* -- A reward value that grows sub-linearly
- *.. versionadded:: 0.4.0*

static reward_length_evaluator_exp(*length*)
Reward Length Evaluator.

Parameters **length** (*int*) -- The length of a local match

Returns

- *float* -- A reward value that grows exponentially
- *.. versionadded:: 0.4.0*

static sift4_matching_evaluator(*src*, *tar*)
Sift4 Matching Evaluator.

Parameters

- **src** (*str*) -- Source string for comparison

- **tar** (*str*) -- Target string for comparison

Returns

- *float* -- The Sift4 similarity of the two tokens
- *.. versionadded:: 0.4.0*

static sift4_token_matcher(*src*, *tar*)
Sift4 Token Matcher.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns

- *bool* -- Whether the Sift4 similarity of the two tokens is over 0.7
- *.. versionadded:: 0.4.0*

abydos.distance.**sift4_common**(*src*, *tar*, *max_offset=5*, *max_distance=0*)
Return the "common" Sift4 distance between two terms.

This is a wrapper for *Sift4.dist_abs()*.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison
- **max_offset** (*int*) -- The number of characters to search for matching letters
- **max_distance** (*int*) -- The distance at which to stop and exit

Returns The Sift4 distance according to the common formula

Return type int

Examples

```
>>> sift4_common('cat', 'hat')
1
>>> sift4_common('Niall', 'Neil')
2
>>> sift4_common('Colin', 'Cuilen')
3
>>> sift4_common('ATCG', 'TAGC')
2
```

New in version 0.3.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the Sift4.dist_abs method instead.

abydos.distance.**sift4_simplest**(*src*, *tar*, *max_offset=5*)
Return the "simplest" Sift4 distance between two terms.

This is a wrapper for *Sift4Simplest.dist_abs()*.

Parameters

- **src** (*str*) -- Source string for comparison

- **tar** (*str*) -- Target string for comparison
- **max_offset** (*int*) -- The number of characters to search for matching letters

Returns The Sift4 distance according to the simplest formula

Return type int

Examples

```
>>> sift4_simplest('cat', 'hat')
1
>>> sift4_simplest('Niall', 'Neil')
2
>>> sift4_simplest('Colin', 'Cuilen')
3
>>> sift4_simplest('ATCG', 'TAGC')
2
```

New in version 0.3.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the Sift4Simplest.dist_abs method instead.

abydos.distance.**dist_sift4**(*src*, *tar*, *max_offset=5*, *max_distance=0*)

Return the normalized "common" Sift4 distance between two terms.

This is a wrapper for *Sift4.dist()*.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison
- **max_offset** (*int*) -- The number of characters to search for matching letters
- **max_distance** (*int*) -- The distance at which to stop and exit

Returns The normalized Sift4 distance

Return type float

Examples

```
>>> round(dist_sift4('cat', 'hat'), 12)
0.333333333333
>>> dist_sift4('Niall', 'Neil')
0.4
>>> dist_sift4('Colin', 'Cuilen')
0.5
>>> dist_sift4('ATCG', 'TAGC')
0.5
```

New in version 0.3.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the Sift4.dist method instead.

abydos.distance.**sim_sift4**(*src*, *tar*, *max_offset=5*, *max_distance=0*)

Return the normalized "common" Sift4 similarity of two terms.

This is a wrapper for `Sift4.sim()`.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison
- **max_offset** (*int*) -- The number of characters to search for matching letters
- **max_distance** (*int*) -- The distance at which to stop and exit

Returns The normalized Sift4 similarity

Return type float

Examples

```
>>> round(sim_sift4('cat', 'hat'), 12)
0.666666666667
>>> sim_sift4('Niall', 'Neil')
0.6
>>> sim_sift4('Colin', 'Cuilen')
0.5
>>> sim_sift4('ATCG', 'TAGC')
0.5
```

New in version 0.3.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the Sift4.sim method instead.

class abydos.distance.**Typo**(*metric='euclidean'*, *cost=(1*, *1*, *0.5*, *0.5)*, *layout='QWERTY'*, *failsafe=False*, ***kwargs*)

Bases: abydos.distance._distance._Distance

Typo distance.

This is inspired by Typo-Distance [Son11], and a fair bit of this was copied from that module. Compared to the original, this supports different metrics for substitution.

New in version 0.3.6.

Initialize Typo instance.

Parameters

- **metric** (*str*) -- Supported values include: euclidean, manhattan, log-euclidean, and log-manhattan
- **cost** (*tuple*) -- A 4-tuple representing the cost of the four possible edits: inserts, deletes, substitutions, and shift, respectively (by default: (1, 1, 0.5, 0.5)) The substitution & shift costs should be significantly less than the cost of an insertion & deletion unless a log metric is used.
- **layout** (*str*) -- Name of the keyboard layout to use (Currently supported: QWERTY, Dvorak, AZERTY, QWERTZ, auto). If auto is selected, the class will attempt to determine an appropriate keyboard based on the supplied words.
- **failsafe** (*bool*) -- If True, substitution of an unknown character (one not present on the selected keyboard) will incur a cost equal to an insertion plus a deletion.
- ****kwargs** -- Arbitrary keyword arguments

New in version 0.4.0.

dist (*src*, *tar*)

Return the normalized typo distance between two strings.

This is typo distance, normalized to [0, 1].

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns Normalized typo distance

Return type float

Examples

```
>>> cmp = Typo()
>>> round(cmp.dist('cat', 'hat'), 12)
0.527046283086
>>> round(cmp.dist('Niall', 'Neil'), 12)
0.565028142929
>>> round(cmp.dist('Colin', 'Cuilen'), 12)
0.569035609563
>>> cmp.dist('ATCG', 'TAGC')
0.625
```

New in version 0.3.0.

Changed in version 0.3.6: Encapsulated in class

dist_abs (*src*, *tar*)

Return the typo distance between two strings.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns Typo distance

Return type float

Raises **ValueError** -- char not found in any keyboard layouts

Examples

```
>>> cmp = Typo()
>>> cmp.dist_abs('cat', 'hat')
1.5811388
>>> cmp.dist_abs('Niall', 'Neil')
2.8251407
>>> cmp.dist_abs('Colin', 'Cuilen')
3.4142137
>>> cmp.dist_abs('ATCG', 'TAGC')
2.5
```

```
>>> cmp = Typo(metric='manhattan')
>>> cmp.dist_abs('cat', 'hat')
2.0
>>> cmp.dist_abs('Niall', 'Neil')
3.0
>>> cmp.dist_abs('Colin', 'Cuilen')
3.5
>>> cmp.dist_abs('ATCG', 'TAGC')
2.5
```

```
>>> cmp = Typo(metric='log-manhattan')
>>> cmp.dist_abs('cat', 'hat')
0.804719
>>> cmp.dist_abs('Niall', 'Neil')
2.2424533
>>> cmp.dist_abs('Colin', 'Cuilen')
2.2424533
>>> cmp.dist_abs('ATCG', 'TAGC')
2.3465736
```

New in version 0.3.0.

Changed in version 0.3.6: Encapsulated in class

abydos.distance.**typo**(*src*, *tar*, *metric='euclidean'*, *cost=(1*, *1*, *0.5*, *0.5)*, *layout='QWERTY'*)

Return the typo distance between two strings.

This is a wrapper for `Typo.typo()`.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison
- **metric** (*str*) -- Supported values include: `euclidean`, `manhattan`, `log-euclidean`, and `log-manhattan`
- **cost** (*tuple*) -- A 4-tuple representing the cost of the four possible edits: inserts, deletes, substitutions, and shift, respectively (by default: (1, 1, 0.5, 0.5)) The substitution & shift costs should be significantly less than the cost of an insertion & deletion unless a log metric is used.
- **layout** (*str*) -- Name of the keyboard layout to use (Currently supported: `QWERTY`, `Dvorak`, `AZERTY`, `QWERTZ`)

Returns Typo distance

Return type float

Examples

```
>>> typo('cat', 'hat')
1.5811388
>>> typo('Niall', 'Neil')
2.8251407
>>> typo('Colin', 'Cuilen')
3.4142137
>>> typo('ATCG', 'TAGC')
2.5
```

```
>>> typo('cat', 'hat', metric='manhattan')
2.0
>>> typo('Niall', 'Neil', metric='manhattan')
3.0
>>> typo('Colin', 'Cuilen', metric='manhattan')
3.5
>>> typo('ATCG', 'TAGC', metric='manhattan')
2.5
```

```
>>> typo('cat', 'hat', metric='log-manhattan')
0.804719
>>> typo('Niall', 'Neil', metric='log-manhattan')
2.2424533
>>> typo('Colin', 'Cuilen', metric='log-manhattan')
2.2424533
>>> typo('ATCG', 'TAGC', metric='log-manhattan')
2.3465736
```

New in version 0.3.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the Typo.dist_abs method instead.

abydos.distance.**dist_typo**(*src*, *tar*, *metric='euclidean'*, *cost=(1, 1, 0.5, 0.5)*, *layout='QWERTY'*)

Return the normalized typo distance between two strings.

This is a wrapper for *Typo.dist()*.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison
- **metric** (*str*) -- Supported values include: `euclidean`, `manhattan`, `log-euclidean`, and `log-manhattan`
- **cost** (*tuple*) -- A 4-tuple representing the cost of the four possible edits: inserts, deletes, substitutions, and shift, respectively (by default: (1, 1, 0.5, 0.5)) The substitution & shift costs should be significantly less than the cost of an insertion & deletion unless a log metric is used.
- **layout** (*str*) -- Name of the keyboard layout to use (Currently supported: `QWERTY`, `Dvorak`, `AZERTY`, `QWERTZ`)

Returns Normalized typo distance

Return type float

Examples

```
>>> round(dist_typo('cat', 'hat'), 12)
0.527046283086
>>> round(dist_typo('Niall', 'Neil'), 12)
0.565028142929
>>> round(dist_typo('Colin', 'Cuilen'), 12)
0.569035609563
>>> dist_typo('ATCG', 'TAGC')
0.625
```

New in version 0.3.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the Typo.dist method instead.

abydos.distance.**sim_typo**(*src*, *tar*, *metric='euclidean'*, *cost=(1, 1, 0.5, 0.5)*, *layout='QWERTY'*)

Return the normalized typo similarity between two strings.

This is a wrapper for `Typo.sim()`.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison
- **metric** (*str*) -- Supported values include: `euclidean`, `manhattan`, `log-euclidean`, and `log-manhattan`
- **cost** (*tuple*) -- A 4-tuple representing the cost of the four possible edits: inserts, deletes, substitutions, and shift, respectively (by default: (1, 1, 0.5, 0.5)) The substitution & shift costs should be significantly less than the cost of an insertion & deletion unless a log metric is used.
- **layout** (*str*) -- Name of the keyboard layout to use (Currently supported: `QWERTY`, `Dvorak`, `AZERTY`, `QWERTZ`)

Returns Normalized typo similarity

Return type float

Examples

```
>>> round(sim_typo('cat', 'hat'), 12)
0.472953716914
>>> round(sim_typo('Niall', 'Neil'), 12)
0.434971857071
>>> round(sim_typo('Colin', 'Cuilen'), 12)
0.430964390437
>>> sim_typo('ATCG', 'TAGC')
0.375
```

New in version 0.3.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the Typo.sim method instead.

class abydos.distance.**Synoname**(*word_approx_min=0.3*, *char_approx_min=0.73*, *tests=4095*, *ret_name=False*, ***kwargs*)

Bases: `abydos.distance._distance._Distance`

Synoname.

Cf. [JPGTrust91][Gro91]

New in version 0.3.6.

Initialize Synoname instance.

Parameters

- **word_approx_min** (*float*) -- The minimum word approximation value to signal a 'word_approx' match
- **char_approx_min** (*float*) -- The minimum character approximation value to signal a 'char_approx' match
- **tests** (*int or Iterable*) -- Either an integer indicating tests to perform or a list of test names to perform (defaults to performing all tests)
- **ret_name** (*bool*) -- If True, returns the match name rather than its integer equivalent
- ****kwargs** -- Arbitrary keyword arguments

New in version 0.4.0.

dist(*src*, *tar*)

Return the normalized Synoname distance between two words.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns Normalized Synoname distance

Return type float

New in version 0.3.0.

Changed in version 0.3.6: Encapsulated in class

dist_abs(*src*, *tar*, *force_numeric=False*)

Return the Synoname similarity type of two words.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison
- **force_numeric** (*bool*) -- Overrides the instance's ret_name setting

Returns Synoname value

Return type int (or str if ret_name is True)

Examples

```
>>> cmp = Synoname()
>>> cmp.dist_abs(('Breghel', 'Pieter', ''), ('Brueghel', 'Pieter', ''))
2
```

```
>>> cmp = Synoname(ret_name=True)
>>> cmp.dist_abs(('Breghel', 'Pieter', ''), ('Brueghel', 'Pieter', ''))
'omission'
```

(continues on next page)

(continued from previous page)

```
>>> cmp.dist_abs(('Dore', 'Gustave', ''),
... ('Dore', 'Paul Gustave Louis Christophe', ''))
'inclusion'
>>> cmp.dist_abs(('Pereira', 'I. R.', ''), ('Pereira', 'I. Smith', ''))
'word_approx'
```

New in version 0.3.0.

Changed in version 0.3.6: Encapsulated in class

abydos.distance.**synoname**(*src*, *tar*, *word_approx_min=0.3*, *char_approx_min=0.73*, *tests=4095*, *ret_name=False*)

Return the Synoname similarity type of two words.

This is a wrapper for *Synoname.dist_abs()*.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison
- **word_approx_min** (*float*) -- The minimum word approximation value to signal a 'word_approx' match
- **char_approx_min** (*float*) -- The minimum character approximation value to signal a 'char_approx' match
- **tests** (*int or Iterable*) -- Either an integer indicating tests to perform or a list of test names to perform (defaults to performing all tests)
- **ret_name** (*bool*) -- If True, returns the match name rather than its integer equivalent

Returns Synoname value

Return type int (or str if ret_name is True)

Examples

```
>>> synoname(('Breghel', 'Pieter', ''), ('Brueghel', 'Pieter', ''))
2
>>> synoname(('Breghel', 'Pieter', ''), ('Brueghel', 'Pieter', ''),
... ret_name=True)
'omission'
>>> synoname(('Dore', 'Gustave', ''),
... ('Dore', 'Paul Gustave Louis Christophe', ''), ret_name=True)
'inclusion'
>>> synoname(('Pereira', 'I. R.', ''), ('Pereira', 'I. Smith', ''),
... ret_name=True)
'word_approx'
```

New in version 0.3.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the Synoname.dist_abs method instead.

class abydos.distance.**Ozbay**(***kwargs*)

Bases: abydos.distance._distance._Distance

Ozbay metric.

The Ozbay metric [Ozb15] is a string distance measure developed by Hakan Ozbay, which combines Jaccard distance, Levenshtein distance, and longest common substring distance.

The normalized variant should be considered experimental.

New in version 0.4.0.

Initialize _Distance instance.

Parameters ****kwargs** -- Arbitrary keyword arguments

New in version 0.4.0.

dist(*src*, *tar*)

Return the normalized Ozbay distance.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns Normalized Ozbay distance

Return type float

Examples

```
>>> cmp = Ozbay()
>>> round(cmp.dist('cat', 'hat'), 12)
0.027777777778
>>> round(cmp.dist('Niall', 'Neil'), 12)
0.24
>>> round(cmp.dist('Colin', 'Cuilen'), 12)
0.214285714286
>>> cmp.dist('ATCG', 'TAGC')
0.140625
```

New in version 0.4.0.

dist_abs(*src*, *tar*)

Return the Ozbay metric.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns Ozbay metric

Return type float

Examples

```
>>> cmp = Ozbay()
>>> round(cmp.dist_abs('cat', 'hat'), 12)
0.75
>>> round(cmp.dist_abs('Niall', 'Neil'), 12)
6.0
>>> round(cmp.dist_abs('Colin', 'Cuilen'), 12)
7.714285714286
>>> cmp.dist_abs('ATCG', 'TAGC')
3.0
```

New in version 0.4.0.

class abydos.distance.**ISG**(*full_guth=False*, *symmetric=True*, ***kwargs*)

Bases: abydos.distance._distance._Distance

Indice de Similitude-Guth (ISG) similarity.

This is an implementation of Bouchard & Pouyez's Indice de Similitude-Guth (ISG) [BP80]. At its heart, ISG is Jaccard similarity, but limits on token matching are added according to part of Guth's matching criteria [Gut76].

[BP80] is limited in its implementation details. Based on the examples given in the paper, it appears that only the first 4 of Guth's rules are considered (a letter in the first string must match a letter in the second string appearing in the same position, an adjacent position, or two positions ahead). It also appears that the distance in the paper is the greater of the distance from string 1 to string 2 and the distance from string 2 to string 1.

These qualities can be specified as parameters. At initialization, specify `full_guth=True` to apply all of Guth's rules and `symmetric=False` to calculate only the distance from string 1 to string 2.

New in version 0.4.1.

Initialize ISG instance.

Parameters

- **full_guth** (*bool*) -- Whether to apply all of Guth's matching rules
- **symmetric** (*bool*) -- Whether to calculate the symmetric distance
- ****kwargs** -- Arbitrary keyword arguments

New in version 0.4.1.

sim(*src*, *tar*)

Return the Indice de Similitude-Guth (ISG) similarity of two words.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns The ISG similarity

Return type float

Examples

```
>>> cmp = ISG()
>>> cmp.sim('cat', 'hat')
0.5
>>> cmp.sim('Niall', 'Neil')
0.5
>>> cmp.sim('aluminum', 'Catalan')
0.15384615384615385
>>> cmp.sim('ATCG', 'TAGC')
1.0
```

New in version 0.4.1.

class abydos.distance.**Inclusion**(***kwargs*)

Bases: abydos.distance._distance._Distance

Inclusion distance.

The INC Programme, developed by [BP80] designates two terms as being "included" when:

- One name is shorter than the other
- There are at least 3 common characters
- There is at most one difference, disregarding unmatching prefixes and suffixes

In addition to these rules, this implementation considers two terms as being "included" if they are identical.

The return value, though a float, can only take one of two values: 0.0, indicating inclusion, or 1.0, indication non-inclusion.

New in version 0.4.1.

Initialize _Distance instance.

Parameters ****kwargs** -- Arbitrary keyword arguments

New in version 0.4.0.

dist(*src*, *tar*)

Return the INClusion Programme value of two words.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns The INC Programme distance

Return type float

Examples

```
>>> cmp = Inclusion()
>>> cmp.dist('cat', 'hat')
1.0
>>> cmp.dist('Niall', 'Neil')
1.0
>>> cmp.dist('aluminum', 'Catalan')
1.0
>>> cmp.dist('ATCG', 'TAGC')
1.0
```

New in version 0.4.1.

class abydos.distance.**Guth**(*tokenizer=None*, ***kwargs*)

Bases: abydos.distance._distance._Distance

Guth matching.

Guth matching [Gut76] uses a simple positional matching rule list to determine whether two names match. Following the original, the *sim_score()* method returns only 1.0 for matching or 0.0 for non-matching.

The *.sim* mathod instead penalizes more distant matches and never outrightly declares two names a non-matching unless no matches can be made in the two strings.

Tokens other than single characters can be matched by specifying a tokenizer during initialization or setting the qval parameter.

New in version 0.4.1.

Initialize Guth instance.

Parameters

- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters **qval** (*int*) -- The length of each q-gram. Using this parameter and tokenizer=None will cause the instance to use the QGram tokenizer with this q value.

New in version 0.4.1.

sim(*src*, *tar*)

Return the relative Guth similarity of two strings.

This deviates from the algorithm described in [Gut76] in that more distant matches are penalized, so that less similar terms score lower that more similar terms.

If no match is found for a particular token in the source string, this does not result in an automatic 0.0 score. Rather, the score is further penalized towards 0.0.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns Relative Guth matching score

Return type float

Examples

```
>>> cmp = Guth()
>>> cmp.sim('cat', 'hat')
0.8666666666666667
>>> cmp.sim('Niall', 'Neil')
0.8800000000000001
>>> cmp.sim('aluminum', 'Catalan')
0.4
>>> cmp.sim('ATCG', 'TAGC')
0.8
```

New in version 0.4.1.

sim_score(*src*, *tar*)

Return the Guth matching score of two strings.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns Guth matching score (1.0 if matching, otherwise 0.0)

Return type float

Examples

```
>>> cmp = Guth()
>>> cmp.sim_score('cat', 'hat')
1.0
>>> cmp.sim_score('Niall', 'Neil')
1.0
>>> cmp.sim_score('aluminum', 'Catalan')
0.0
>>> cmp.sim_score('ATCG', 'TAGC')
1.0
```

New in version 0.4.1.

class abydos.distance.**VPS**(***kwargs*)

Bases: abydos.distance._distance._Distance

Victorian Panel Study (VPS) score.

VPS score is presented in [Schurer07].

New in version 0.4.1.

Initialize _Distance instance.

Parameters ****kwargs** -- Arbitrary keyword arguments

New in version 0.4.0.

sim(*src*, *tar*)

Return the Victorian Panel Study score of two words.

Parameters

- **src** (*str*) -- Source string for comparison

- **tar** (*str*) -- Target string for comparison

Returns The VPS score

Return type float

Examples

```
>>> cmp = VPS()
>>> cmp.sim('cat', 'hat')
0.5
>>> cmp.sim('Niall', 'Neil')
0.3
>>> cmp.sim('aluminum', 'Catalan')
0.14285714285714285
>>> cmp.sim('ATCG', 'TAGC')
0.3333333333333333
```

New in version 0.4.1.

class abydos.distance.**LIG3**(***kwargs*)

Bases: abydos.distance._distance._Distance

LIG3 similarity.

[SD02] proposes three Levenshtein-ISG-Guth hybrid similarity measures: LIG1, LIG2, and LIG3. Of these, LIG1 is identical to ISG and LIG2 is identical to normalized Levenshtein similarity. Only LIG3 is a novel measure, defined as:

$$sim_{LIG3}(X, Y) = \frac{2I}{2I + C}$$

Here, I is the number of exact matches between the two words, truncated to the length of the shorter word, and C is the Levenshtein distance between the two words.

New in version 0.4.1.

Initialize _Distance instance.

Parameters ****kwargs** -- Arbitrary keyword arguments

New in version 0.4.0.

sim(*src*, *tar*)

Return the LIG3 similarity of two words.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns The LIG3 similarity

Return type float

Examples

```
>>> cmp = LIG3()
>>> cmp.sim('cat', 'hat')
0.8
>>> cmp.sim('Niall', 'Neil')
0.5714285714285714
>>> cmp.sim('aluminum', 'Catalan')
0.0
>>> cmp.sim('ATCG', 'TAGC')
0.0
```

New in version 0.4.1.

class abydos.distance.**SSK**(*tokenizer=None*, *ssk_lambda=0.9*, ***kwargs*)

Bases: abydos.distance._token_distance._TokenDistance

String subsequence kernel (SSK) similarity.

This is based on [LSShaweTaylor+02].

New in version 0.4.1.

Initialize SSK instance.

Parameters

- **tokenizer** (*_Tokenizer*) -- A tokenizer instance from the *abydos.tokenizer* package
- **ssk_lambda** (*float or Iterable*) -- A value in the range (0.0, 1.0) used for discouting gaps between characters according to the method described in [LSShaweTaylor+02]. To supply multiple values of lambda, provide an Iterable of numeric values, such as (0.5, 0.05) or np.arange(0.05, 0.5, 0.05)
- ****kwargs** -- Arbitrary keyword arguments

Other Parameters **qval** (*int*) -- The length of each q-skipgram. Using this parameter and tokenizer=None will cause the instance to use the QGramskipgrams tokenizer with this q value.

New in version 0.4.1.

sim(*src*, *tar*)

Return the normalized SSK similarity of two strings.

Parameters

- **src** (*str*) -- Source string (or QGrams/Counter objects) for comparison
- **tar** (*str*) -- Target string (or QGrams/Counter objects) for comparison

Returns Normalized string subsequence kernel similarity

Return type float

Examples

```
>>> cmp = SSK()
>>> cmp.sim('cat', 'hat')
0.3558718861209964
>>> cmp.sim('Niall', 'Neil')
0.4709007822130597
>>> cmp.sim('aluminum', 'Catalan')
0.13760157193822603
>>> cmp.sim('ATCG', 'TAGC')
0.6140899528060498
```

New in version 0.4.1.

sim_score(*src*, *tar*)

Return the SSK similarity of two strings.

Parameters

- **src** (*str*) -- Source string for comparison
- **tar** (*str*) -- Target string for comparison

Returns String subsequence kernel similarity

Return type float

Examples

```
>>> cmp = SSK()
>>> cmp.dist_abs('cat', 'hat')
0.6441281138790036
>>> cmp.dist_abs('Niall', 'Neil')
0.5290992177869402
>>> cmp.dist_abs('aluminum', 'Catalan')
0.862398428061774
>>> cmp.dist_abs('ATCG', 'TAGC')
0.38591004719395017
```

New in version 0.4.1.

3.1.1.4 abydos.fingerprint package

abydos.fingerprint.

The fingerprint package implements string fingerprints such as:

- Basic fingerprinters originating in *OpenRefine <http://openrefine.org>*:
  - String (*String*)
  - Phonetic, which applies a phonetic algorithm and returns the string fingerprint of the result (*Phonetic*)
  - QGram, which applies Q-gram tokenization and returns the string fingerprint of the result (*QGram*)
- Fingerprints developed by Pollock & Zomora:
  - Skeleton key (*SkeletonKey*)
  - Omission key (*OmissionKey*)

- Fingerprints developed by Cisłak & Grabowski:
  - Occurrence (*Occurrence*)
  - Occurrence halved (*OccurrenceHalved*)
  - Count (*Count*)
  - Position (*Position*)
- The Synoname toolcode (*SynonameToolcode*)
- Taft's codings:
  - Consonant coding (*Consonant*)
  - Extract - letter list (*Extract*)
  - Extract - position & frequency (*ExtractPositionFrequency*)
- L.A. County Sheriff's System (*LACSS*)
- Library of Congress Cutter table encoding (*LCCutter*)
- Burrows-Wheeler transform (*BWTF*) and run-length encoded Burrows-Wheeler transform (*BWTRLEF*)

Each fingerprint class has a `fingerprint` method that takes a string and returns the string's fingerprint:

```
>>> sk = SkeletonKey()
>>> sk.fingerprint('orange')
'ORNGAE'
>>> sk.fingerprint('strange')
'STRNGAE'
```

---

class abydos.fingerprint.**String**(*joiner=' '*)

Bases: abydos.fingerprint._fingerprint._Fingerprint

String Fingerprint.

The fingerprint of a string is a string consisting of all of the unique words in a string, alphabetized & concatenated with intervening joiners. This fingerprint is described at [Ope12].

New in version 0.3.6.

Initialize String instance.

Parameters **joiner** (*str*) -- The string that will be placed between each word

New in version 0.4.0.

fingerprint(*phrase*)

Return string fingerprint.

Parameters **phrase** (*str*) -- The string from which to calculate the fingerprint

Returns The fingerprint of the phrase

Return type str

Example

```
>>> sf = String()
>>> sf.fingerprint('The quick brown fox jumped over the lazy dog.')
'brown dog fox jumped lazy over quick the'
```

New in version 0.1.0.

Changed in version 0.3.6: Encapsulated in class

abydos.fingerprint.**str_fingerprint**(*phrase*, *joiner=' '*)

Return string fingerprint.

This is a wrapper for *String.fingerprint()*.

Parameters

- **phrase** (*str*) -- The string from which to calculate the fingerprint
- **joiner** (*str*) -- The string that will be placed between each word

Returns The fingerprint of the phrase

Return type str

Example

```
>>> str_fingerprint('The quick brown fox jumped over the lazy dog.')
'brown dog fox jumped lazy over quick the'
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the String.fingerprint method instead.

class abydos.fingerprint.**QGram**(*qval=2*, *start_stop=''*, *joiner=''*, *skip=0*)

Bases: abydos.fingerprint._fingerprint._Fingerprint

Q-Gram Fingerprint.

A q-gram fingerprint is a string consisting of all of the unique q-grams in a string, alphabetized & concatenated. This fingerprint is described at [Ope12].

New in version 0.3.6.

Initialize Q-Gram fingerprinter.

qval [int] The length of each q-gram (by default 2)

start_stop [str] The start & stop symbol(s) to concatenate on either end of the phrase, as defined in tokenizer.QGrams

joiner [str] The string that will be placed between each word

skip [int or Iterable] The number of characters to skip, can be an integer, range object, or list

New in version 0.4.0.

fingerprint(*phrase*)

Return Q-Gram fingerprint.

Parameters **phrase** (*str*) -- The string from which to calculate the q-gram fingerprint

Returns The q-gram fingerprint of the phrase

Return type str

Examples

```
>>> qf = QGram()
>>> qf.fingerprint('The quick brown fox jumped over the lazy dog.')
'azbrckdoedeleqerfoheicjukblampnfogovowoxpequrortthuiumvewnxjydzy'
>>> qf.fingerprint('Christopher')
'cherhehrisopphristto'
>>> qf.fingerprint('Niall')
'aliallni'
```

New in version 0.1.0.

Changed in version 0.3.6: Encapsulated in class

abydos.fingerprint.**qgram_fingerprint**(*phrase*, *qval=2*, *start_stop=''*, *joiner=''*)

Return Q-Gram fingerprint.

This is a wrapper for *QGram.fingerprint()*.

Parameters

- **phrase** (*str*) -- The string from which to calculate the q-gram fingerprint
- **qval** (*int*) -- The length of each q-gram (by default 2)
- **start_stop** (*str*) -- The start & stop symbol(s) to concatenate on either end of the phrase, as defined in tokenizer.QGrams
- **joiner** (*str*) -- The string that will be placed between each word

Returns The q-gram fingerprint of the phrase

Return type str

Examples

```
>>> qgram_fingerprint('The quick brown fox jumped over the lazy dog.')
'azbrckdoedeleqerfoheicjukblampnfogovowoxpequrortthuiumvewnxjydzy'
>>> qgram_fingerprint('Christopher')
'cherhehrisopphristto'
>>> qgram_fingerprint('Niall')
'aliallni'
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the QGram.fingerprint method instead.

class abydos.fingerprint.**Phonetic**(*phonetic_algorithm=None*, *joiner=' '*)

Bases: abydos.fingerprint._string.String

Phonetic Fingerprint.

A phonetic fingerprint is identical to a standard string fingerprint, as implemented in *String*, but performs the fingerprinting function after converting the string to its phonetic form, as determined by some phonetic algorithm. This fingerprint is described at [Ope12].

New in version 0.3.6.

Initialize Phonetic instance.

phonetic_algorithm [function] A phonetic algorithm that takes a string and returns a string (presumably a phonetic representation of the original string). By default, this function uses *double_metaphone()*.

joiner [str] The string that will be placed between each word

New in version 0.4.0.

fingerprint(*phrase*)

Return the phonetic fingerprint of a phrase.

Parameters **phrase** (*str*) -- The string from which to calculate the phonetic fingerprint

Returns The phonetic fingerprint of the phrase

Return type str

Examples

```
>>> pf = Phonetic()
>>> pf.fingerprint('The quick brown fox jumped over the lazy dog.')
'0 afr fks jmpt kk ls prn tk'
```

```
>>> from abydos.phonetic import Soundex
>>> pf = Phonetic(Soundex())
>>> pf.fingerprint('The quick brown fox jumped over the lazy dog.')
'b650 d200 f200 j513 l200 o160 q200 t000'
```

New in version 0.1.0.

Changed in version 0.3.6: Encapsulated in class

abydos.fingerprint.**phonetic_fingerprint**(*phrase*, *phonetic_algorithm=<function double_metaphone>*, *joiner=' '*, **args*, ***kwargs*)

Return the phonetic fingerprint of a phrase.

This is a wrapper for *Phonetic.fingerprint()*.

Parameters

- **phrase** (*str*) -- The string from which to calculate the phonetic fingerprint
- **phonetic_algorithm** (*function*) -- A phonetic algorithm that takes a string and returns a string (presumably a phonetic representation of the original string). By default, this function uses *double_metaphone()*.
- **joiner** (*str*) -- The string that will be placed between each word
- ***args** -- Variable length argument list
- ****kwargs** -- Arbitrary keyword arguments

Returns The phonetic fingerprint of the phrase

Return type str

Examples

```
>>> phonetic_fingerprint('The quick brown fox jumped over the lazy dog.')
'0 afr fks jmpt kk ls prn tk'
```

```
>>> from abydos.phonetic import soundex
>>> phonetic_fingerprint('The quick brown fox jumped over the lazy dog.',
... phonetic_algorithm=soundex)
'b650 d200 f200 j513 l200 o160 q200 t000'
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the Phonetic.fingerprint method instead.

class abydos.fingerprint.**OmissionKey**

Bases: abydos.fingerprint._fingerprint._Fingerprint

Omission Key.

The omission key of a word is defined in [PZ84].

New in version 0.3.6.

fingerprint(*word*)

Return the omission key.

Parameters **word** (*str*) -- The word to transform into its omission key

Returns The omission key

Return type str

Examples

```
>>> ok = OmissionKey()
>>> ok.fingerprint('The quick brown fox jumped over the lazy dog.')
'JKQXZVWYBFMGPDHCLNTREUIOA'
>>> ok.fingerprint('Christopher')
'PHCTSRIOE'
>>> ok.fingerprint('Niall')
'LNIA'
```

New in version 0.1.0.

Changed in version 0.3.6: Encapsulated in class

abydos.fingerprint.**omission_key**(*word*)

Return the omission key.

This is a wrapper for *OmissionKey.fingerprint()*.

Parameters **word** (*str*) -- The word to transform into its omission key

Returns The omission key

Return type str

Examples

```
>>> omission_key('The quick brown fox jumped over the lazy dog.')
'JKQXZVWYBFMGPDHCLNTREUIOA'
>>> omission_key('Christopher')
'PHCTSRIOE'
>>> omission_key('Niall')
'LNIA'
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the OmissionKey.fingerprint method instead.

class abydos.fingerprint.**SkeletonKey**

Bases: abydos.fingerprint._fingerprint._Fingerprint

Skeleton Key.

The skeleton key of a word is defined in [PZ84].

New in version 0.3.6.

fingerprint(*word*)

Return the skeleton key.

Parameters **word** (*str*) -- The word to transform into its skeleton key

Returns The skeleton key

Return type str

Examples

```
>>> sk = SkeletonKey()
>>> sk.fingerprint('The quick brown fox jumped over the lazy dog.')
'THQCKBRWNFXJMPDVLZYGEUIOA'
>>> sk.fingerprint('Christopher')
'CHRSTPIOE'
>>> sk.fingerprint('Niall')
'NLIA'
```

New in version 0.1.0.

Changed in version 0.3.6: Encapsulated in class

abydos.fingerprint.**skeleton_key**(*word*)

Return the skeleton key.

This is a wrapper for *SkeletonKey.fingerprint()*.

Parameters **word** (*str*) -- The word to transform into its skeleton key

Returns The skeleton key

Return type str

Examples

```
>>> skeleton_key('The quick brown fox jumped over the lazy dog.')
'THQCKBRWNFXJMPDVLZYGEUIOA'
>>> skeleton_key('Christopher')
'CHRSTPIOE'
>>> skeleton_key('Niall')
'NLIA'
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the SkeletonKey.fingerprint method instead.

class abydos.fingerprint.**Occurrence**(*n_bits=16*, *most_common=('e', 't', 'a', 'o', 'i', 'n', 's', 'h', 'r', 'd', 'l', 'c', 'u', 'm', 'w', 'f')*)

Bases: abydos.fingerprint._fingerprint._Fingerprint

Occurrence Fingerprint.

Based on the occurrence fingerprint from [CislakG17].

New in version 0.3.6.

Initialize Count instance.

Parameters

- **n_bits** (*int*) -- Number of bits in the fingerprint returned
- **most_common** (*list*) -- The most common tokens in the target language, ordered by frequency

New in version 0.4.0.

fingerprint(*word*)

Return the occurrence fingerprint.

Parameters **word** (*str*) -- The word to fingerprint

Returns The occurrence fingerprint

Return type int

Examples

```
>>> of = Occurrence()
>>> bin(of.fingerprint('hat'))
'0b1100000100000000'
>>> bin(of.fingerprint('niall'))
'0b10110000100000'
>>> bin(of.fingerprint('colin'))
'0b1110000110000'
>>> bin(of.fingerprint('atcg'))
'0b1100000000010000'
>>> bin(of.fingerprint('entreatment'))
'0b1110010010000100'
```

New in version 0.3.0.

Changed in version 0.3.6: Encapsulated in class

abydos.fingerprint.**occurrence_fingerprint**(*word*, *n_bits=16*, *most_common=('e', 't', 'a', 'o', 'i', 'n', 's', 'h', 'r', 'd', 'l', 'c', 'u', 'm', 'w', 'f')*)

Return the occurrence fingerprint.

This is a wrapper for *Occurrence.fingerprint()*.

Parameters

- **word** (*str*) -- The word to fingerprint
- **n_bits** (*int*) -- Number of bits in the fingerprint returned
- **most_common** (*list*) -- The most common tokens in the target language, ordered by frequency

Returns The occurrence fingerprint

Return type int

Examples

```
>>> bin(occurrence_fingerprint('hat'))
'0b110000100000000'
>>> bin(occurrence_fingerprint('niall'))
'0b10110000100000'
>>> bin(occurrence_fingerprint('colin'))
'0b11100000110000'
>>> bin(occurrence_fingerprint('atcg'))
'0b1100000000010000'
>>> bin(occurrence_fingerprint('entreatment'))
'0b11100100100000100'
```

New in version 0.3.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the Occurrence.fingerprint method instead.

class abydos.fingerprint.**OccurrenceHalved**(*n_bits=16*, *most_common=('e', 't', 'a', 'o', 'i', 'n', 's', 'h', 'r', 'd', 'l', 'c', 'u', 'm', 'w', 'f')*)

Bases: abydos.fingerprint._fingerprint._Fingerprint

Occurrence Halved Fingerprint.

Based on the occurrence halved fingerprint from [CislakG17].

New in version 0.3.6.

Initialize Count instance.

Parameters

- **n_bits** (*int*) -- Number of bits in the fingerprint returned
- **most_common** (*list*) -- The most common tokens in the target language, ordered by frequency

New in version 0.4.0.

fingerprint(*word*)

Return the occurrence halved fingerprint.

Based on the occurrence halved fingerprint from [CislakG17].

Parameters

- **word** (*str*) -- The word to fingerprint

- **n_bits** (*int*) -- Number of bits in the fingerprint returned
- **most_common** (*list*) -- The most common tokens in the target language, ordered by frequency

Returns The occurrence halved fingerprint

Return type int

Examples

```
>>> ohf = OccurrenceHalved()
>>> bin(ohf.fingerprint('hat'))
'0b1010000000010'
>>> bin(ohf.fingerprint('niall'))
'0b10010100000'
>>> bin(ohf.fingerprint('colin'))
'0b1001010000'
>>> bin(ohf.fingerprint('atcg'))
'0b10100000000000'
>>> bin(ohf.fingerprint('entreatment'))
'0b1111010000110000'
```

New in version 0.3.0.

Changed in version 0.3.6: Encapsulated in class

abydos.fingerprint.**occurrence_halved_fingerprint**(*word*, *n_bits=16*, *most_common=('e', 't', 'a', 'o', 'i', 'n', 's', 'h', 'r', 'd', 'l', 'c', 'u', 'm', 'w', 'f')*)

Return the occurrence halved fingerprint.

This is a wrapper for *OccurrenceHalved.fingerprint()*.

Parameters

- **word** (*str*) -- The word to fingerprint
- **n_bits** (*int*) -- Number of bits in the fingerprint returned
- **most_common** (*list*) -- The most common tokens in the target language, ordered by frequency

Returns The occurrence halved fingerprint

Return type int

Examples

```
>>> bin(occurrence_halved_fingerprint('hat'))
'0b1010000000010'
>>> bin(occurrence_halved_fingerprint('niall'))
'0b10010100000'
>>> bin(occurrence_halved_fingerprint('colin'))
'0b1001010000'
>>> bin(occurrence_halved_fingerprint('atcg'))
'0b10100000000000'
>>> bin(occurrence_halved_fingerprint('entreatment'))
'0b1111010000110000'
```

New in version 0.3.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the OccurrenceHalved.fingerprint method instead.

class abydos.fingerprint.**Count**(*n_bits=16*, *most_common=('e', 't', 'a', 'o', 'i', 'n', 's', 'h', 'r', 'd', 'l', 'c', 'u', 'm', 'w', 'f')*)

Bases: abydos.fingerprint._fingerprint._Fingerprint

Count Fingerprint.

Based on the count fingerprint from [CislakG17].

New in version 0.3.6.

Initialize Count instance.

Parameters

- **n_bits** (*int*) -- Number of bits in the fingerprint returned
- **most_common** (*list*) -- The most common tokens in the target language, ordered by frequency

New in version 0.4.0.

fingerprint(*word*)

Return the count fingerprint.

Parameters **word** (*str*) -- The word to fingerprint

Returns The count fingerprint

Return type int

Examples

```
>>> cf = Count()
>>> bin(cf.fingerprint('hat'))
'0b1010000000001'
>>> bin(cf.fingerprint('niall'))
'0b10001010000'
>>> bin(cf.fingerprint('colin'))
'0b101010000'
>>> bin(cf.fingerprint('atcg'))
'0b1010000000000'
>>> bin(cf.fingerprint('entreatment'))
'0b1111010000100000'
```

New in version 0.3.0.

Changed in version 0.3.6: Encapsulated in class

abydos.fingerprint.**count_fingerprint**(*word*, *n_bits=16*, *most_common=('e', 't', 'a', 'o', 'i', 'n', 's', 'h', 'r', 'd', 'l', 'c', 'u', 'm', 'w', 'f')*)

Return the count fingerprint.

This is a wrapper for *Count.fingerprint()*.

Parameters

- **word** (*str*) -- The word to fingerprint
- **n_bits** (*int*) -- Number of bits in the fingerprint returned

- **most_common** (*list*) -- The most common tokens in the target language, ordered by frequency

Returns The count fingerprint

Return type int

Examples

```
>>> bin(count_fingerprint('hat'))
'0b1010000000001'
>>> bin(count_fingerprint('niall'))
'0b10001010000'
>>> bin(count_fingerprint('colin'))
'0b101010000'
>>> bin(count_fingerprint('atcg'))
'0b1010000000000'
>>> bin(count_fingerprint('entreatment'))
'0b1111010000100000'
```

New in version 0.3.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the Count.fingerprint method instead.

class abydos.fingerprint.**Position**(*n_bits=16*, *most_common=('e', 't', 'a', 'o', 'i', 'n', 's', 'h', 'r', 'd', 'l', 'c', 'u', 'm', 'w', 'f')*, *bits_per_letter=3*)

Bases: abydos.fingerprint._fingerprint._Fingerprint

Position Fingerprint.

Based on the position fingerprint from [CislakG17].

New in version 0.3.6.

Initialize Count instance.

Parameters

- **n_bits** (*int*) -- Number of bits in the fingerprint returned
- **most_common** (*list*) -- The most common tokens in the target language, ordered by frequency

New in version 0.4.0.

fingerprint(*word*)

Return the position fingerprint.

Parameters **word** (*str*) -- The word to fingerprint

Returns The position fingerprint

Return type int

Examples

```
>>> bin(position_fingerprint('hat'))
'0b1110100011111111'
>>> bin(position_fingerprint('niall'))
'0b1111110101110010'
>>> bin(position_fingerprint('colin'))
'0b1111111110010111'
>>> bin(position_fingerprint('atcg'))
'0b1110010001111111'
>>> bin(position_fingerprint('entreatment'))
'0b101011111111'
```

New in version 0.3.0.

Changed in version 0.3.6: Encapsulated in class

abydos.fingerprint.**position_fingerprint**(*word*, *n_bits=16*, *most_common=('e', 't', 'a', 'o', 'i', 'n', 's', 'h', 'r', 'd', 'l', 'c', 'u', 'm', 'w', 'f')*, *bits_per_letter=3*)

Return the position fingerprint.

This is a wrapper for *Position.fingerprint()*.

Parameters

- **word** (*str*) -- The word to fingerprint
- **n_bits** (*int*) -- Number of bits in the fingerprint returned
- **most_common** (*list*) -- The most common tokens in the target language, ordered by frequency
- **bits_per_letter** (*int*) -- The bits to assign for letter position

Returns The position fingerprint

Return type int

Examples

```
>>> bin(position_fingerprint('hat'))
'0b1110100011111111'
>>> bin(position_fingerprint('niall'))
'0b1111110101110010'
>>> bin(position_fingerprint('colin'))
'0b1111111110010111'
>>> bin(position_fingerprint('atcg'))
'0b1110010001111111'
>>> bin(position_fingerprint('entreatment'))
'0b101011111111'
```

New in version 0.3.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the Position.fingerprint method instead.

class abydos.fingerprint.**SynonameToolcode**

Bases: abydos.fingerprint._fingerprint._Fingerprint

Synoname Toolcode.

Cf. [JPGTrust91][Gro91].

New in version 0.3.6.

fingerprint(*lname*, *fname=''*, *qual=''*, *normalize=0*)

Build the Synoname toolcode.

Parameters

- **lname** (*str*) -- Last name
- **fname** (*str*) -- First name (can be blank)
- **qual** (*str*) -- Qualifier
- **normalize** (*int*) -- Normalization mode (0, 1, or 2)

Returns The transformed names and the synoname toolcode

Return type tuple

Examples

```
>>> st = SynonameToolcode()
>>> st.fingerprint('hat')
('hat', '', '0000000003$$h')
>>> st.fingerprint('niall')
('niall', '', '0000000005$$n')
>>> st.fingerprint('colin')
('colin', '', '0000000005$$c')
>>> st.fingerprint('atcg')
('atcg', '', '0000000004$$a')
>>> st.fingerprint('entreatment')
('entreatment', '', '0000000011$$e')
```

```
>>> st.fingerprint('Ste.-Marie', 'Count John II', normalize=2)
('ste.-marie ii', 'count john', '0200491310$015b049a127c$smcji')
>>> st.fingerprint('Michelangelo IV', '', 'Workshop of')
('michelangelo iv', '', '3000550015$055b$mi')
```

New in version 0.3.0.

Changed in version 0.3.6: Encapsulated in class

abydos.fingerprint.**synoname_toolcode**(*lname*, *fname=''*, *qual=''*, *normalize=0*)

Build the Synoname toolcode.

This is a wrapper for *SynonameToolcode.fingerprint()*.

Parameters

- **lname** (*str*) -- Last name
- **fname** (*str*) -- First name (can be blank)
- **qual** (*str*) -- Qualifier
- **normalize** (*int*) -- Normalization mode (0, 1, or 2)

Returns The transformed names and the synoname toolcode

Return type tuple

Examples

```
>>> synoname_toolcode('hat')
('hat', '', '0000000003$$h')
>>> synoname_toolcode('niall')
('niall', '', '0000000005$$n')
>>> synoname_toolcode('colin')
('colin', '', '0000000005$$c')
>>> synoname_toolcode('atcg')
('atcg', '', '0000000004$$a')
>>> synoname_toolcode('entreatment')
('entreatment', '', '0000000011$$e')
```

```
>>> synoname_toolcode('Ste.-Marie', 'Count John II', normalize=2)
('ste.-marie ii', 'count john', '0200491310$015b049a127c$smcji')
>>> synoname_toolcode('Michelangelo IV', '', 'Workshop of')
('michelangelo iv', '', '3000550015$055b$mi')
```

New in version 0.3.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the SynonameToolcode.fingerprint method instead.

class abydos.fingerprint.**Consonant**(*variant=1*, *doubles=True*, *vowels=None*)

Bases: abydos.fingerprint._fingerprint._Fingerprint

Consonant Coding Fingerprint.

Based on the consonant coding from [Taf70], variants 1, 2, 3, 1-D, 2-D, and 3-D.

New in version 0.4.1.

Initialize Consonant instance.

Parameters

- **variant** (*int*) -- Selects between Taft's 3 variants, which assign to the vowel set one of:
  1. A, E, I, O, & U
  2. A, E, I, O, U, W, & Y
  3. A, E, I, O, U, W, H, & Y
- **doubles** (*bool*) -- If set to False, multiple consonants in a row are conflated to a single instance.
- **vowels** (*list, set, or str*) -- Setting vowels to a non-None value overrides the variant setting and defines the set of letters to be removed from the input.

New in version 0.4.1.

fingerprint(*word*)

Return the consonant coding.

Parameters **word** (*str*) -- The word to fingerprint

Returns The consonant coding

Return type int

Examples

```
>>> cf = Consonant()
>>> cf.fingerprint('hat')
'HT'
>>> cf.fingerprint('niall')
'NLL'
>>> cf.fingerprint('colin')
'CLN'
>>> cf.fingerprint('atcg')
'ATCG'
>>> cf.fingerprint('entreatment')
'ENTRTMNT'
```

New in version 0.4.1.

class abydos.fingerprint.**Extract**(*letter_list=1*)

Bases: abydos.fingerprint._fingerprint._Fingerprint

Extract Letter List fingerprint.

Based on the extract letter list coding from [Taf70], for lists 1, 2, 3, & 4.

New in version 0.4.1.

Initialize Extract instance.

Parameters **letter_list** (*int or iterable*) -- If an integer (1-4) is supplied, Taft's specified letter lists are used. If an iterable is supplied, its values will be used as the list of letters to remove (in order).

New in version 0.4.1.

fingerprint(*word*)

Return the extract letter list coding.

Parameters **word** (*str*) -- The word to fingerprint

Returns The extract letter list coding

Return type int

Examples

```
>>> fp = Extract()
>>> fp.fingerprint('hat')
'HAT'
>>> fp.fingerprint('niall')
'NILL'
>>> fp.fingerprint('colin')
'CLIN'
>>> fp.fingerprint('atcg')
'ATCG'
>>> fp.fingerprint('entreatment')
'NRMN'
```

New in version 0.4.1.

class abydos.fingerprint.**ExtractPositionFrequency**

Bases: abydos.fingerprint._fingerprint._Fingerprint

Extract - Position & Frequency fingerprint.

Based on the extract - position & frequency coding from [Taf70].

New in version 0.4.1.

fingerprint(*word*)

Return the extract - position & frequency coding.

Parameters **word** (*str*) -- The word to fingerprint

Returns The extract - position & frequency coding

Return type int

Examples

```
>>> fp = ExtractPositionFrequency()
>>> fp.fingerprint('hat')
'HAT'
>>> fp.fingerprint('niall')
'NILL'
>>> fp.fingerprint('colin')
'COLN'
>>> fp.fingerprint('atcg')
'ATCG'
>>> fp.fingerprint('entreatment')
'NMNT'
```

New in version 0.4.1.

class abydos.fingerprint.**LACSS**

Bases: abydos.fingerprint._fingerprint._Fingerprint

L.A. County Sheriff's System fingerprint.

Based on the description from [Taf70].

New in version 0.4.1.

fingerprint(*word*)

Return the LACSS coding.

Parameters **word** (*str*) -- The word to fingerprint

Returns The L.A. County Sheriff's System fingerprint

Return type int

Examples

```
>>> cf = LACSS()
>>> cf.fingerprint('hat')
'4911211'
>>> cf.fingerprint('niall')
'6488374'
>>> cf.fingerprint('colin')
'3015957'
>>> cf.fingerprint('atcg')
'1772371'
```

(continues on next page)

(continued from previous page)

```
>>> cf.fingerprint('entreatment')
'3882324'
```

New in version 0.4.1.

class abydos.fingerprint.**LCCutter**(*max_length=64*)
Bases: abydos.fingerprint._fingerprint._Fingerprint

Library of Congress Cutter table encoding.

This is based on the Library of Congress Cutter table encoding scheme, as described at https://www.loc.gov/aba/pcc/053/table.html [oC13]. Handling for numerals is not included.

New in version 0.4.1.

Initialize LCCutter instance.

Parameters **max_length** (*int*) -- The length of the code returned (defaults to 64)

New in version 0.4.1.

fingerprint(*word*)
Return the Library of Congress Cutter table encoding of a word.

Parameters **word** (*str*) -- The word to fingerprint

Returns The Library of Congress Cutter table encoding

Return type str

Examples

```
>>> cf = LCCutter()
>>> cf.fingerprint('hat')
'H38'
>>> cf.fingerprint('niall')
'N5355'
>>> cf.fingerprint('colin')
'C6556'
>>> cf.fingerprint('atcg')
'A834'
>>> cf.fingerprint('entreatment')
'E5874386468'
```

New in version 0.4.1.

class abydos.fingerprint.**BWTF**(*terminator='x00'*)
Bases: abydos.fingerprint._fingerprint._Fingerprint

Burrows-Wheeler transform fingerprint.

This is a wrapper of the BWT class in abydos.compression, which provides the same interface as other descendants of _Fingerprint.

New in version 0.4.1.

Initialize BWTF instance.

Parameters **terminator** (*str*) -- A character added to signal the end of the string

New in version 0.4.1.

fingerprint(*word*)

Return the Burrows-Wheeler transform of a word.

Parameters **word** (*str*) -- The word to fingerprint

Returns The Burrows-Wheeler transform of a word

Return type str

Examples

```
>>> fp = BWTF()
>>> fp.fingerprint('hat')
'th\x00a'
>>> fp.fingerprint('niall')
'linla\x00'
>>> fp.fingerprint('colin')
'n\x00loic'
>>> fp.fingerprint('atcg')
'g\x00tca'
>>> fp.fingerprint('entreatment')
'term\x00teetnan'
```

New in version 0.4.1.

class abydos.fingerprint.**BWTRLEF**(*terminator='x00'*)

Bases: abydos.fingerprint._fingerprint._Fingerprint

Burrows-Wheeler transform plus run-length encoding fingerprint.

This is a wrapper of the BWT and RLE classes in abydos.compression, which provides the same interface as other descendants of _Fingerprint.

New in version 0.4.1.

Initialize BWTRLEF instance.

Parameters **terminator** (*str*) -- A character added to signal the end of the string

New in version 0.4.1.

fingerprint(*word*)

Return the run-length encoded Burrows-Wheeler transform of a word.

Parameters **word** (*str*) -- The word to fingerprint

Returns The run-length encoded Burrows-Wheeler transform of a word

Return type str

Examples

```
>>> fp = BWTRLEF()
>>> fp.fingerprint('hat')
'th\x00a'
>>> fp.fingerprint('niall')
'linla\x00'
>>> fp.fingerprint('colin')
'n\x00loic'
>>> fp.fingerprint('atcg')
```

(continues on next page)

(continued from previous page)

```
'g\x00tca'
>>> fp.fingerprint('entreatment')
'term\x00teetnan'
```

New in version 0.4.1.

3.1.1.5 abydos.phones package

abydos.phones.

The phones module implements phonetic feature coding, decoding, and comparison functions. It has three functions:

- *ipa_to_features()* takes a string of IPA symbols and returns list of integers that represent the phonetic features bundled in the phone that the symbols represents.
- *ipa_to_feature_dicts()* takes a string of IPA symbols and returns list of human-readable dicts that represent the phonetic features bundled in the phone that the symbols represents.
- *get_feature()* takes a list of feature bundles produced by *ipa_to_features()* and a feature name and returns a list representing whether that feature is present in each component of the list.
- *cmp_features()* takes two phonetic feature bundles, such as the components of the lists returned by *ipa_to_features()*, and returns a measure of their similarity.

An example using these functions on two different pronunciations of the word 'international':

```
>>> int1 = 'ntnæn'
>>> int2 = 'nnæn'
>>> feat1 = ipa_to_features(int1)
>>> feat1
[1826957413067434410,
 2711173160463936106,
 2783230754502126250,
 1828083331160779178,
 2711173160463936106,
 1826957425885227434,
 2783231556184615322,
 1828083331160779178,
 2711173160463936106,
 1828083331160779178,
 2693158721554917798]
>>> feat2 = ipa_to_features(int2)
>>> feat2
[1826957413067434410,
 2711173160463936106,
 2711173160463935914,
 1828083331160779178,
 2711173160463936106,
 1826957425885227434,
 2783231556184615322,
 1826957414069873066,
 2711173160463936106,
 1828083331160779178,
 2693158721554917798]
>>> ipa_to_feature_dicts('n')
[{'syllabic': '-',
  'consonantal': '+',
```

(continues on next page)

(continued from previous page)

```
  'sonorant': '-',
  'approximant': '-',
  'labial': '-',
  'round': '0',
  'protruded': '0',
  'compressed': '0',
  'labiodental': '0',
  'coronal': '+',
  'anterior': '-',
  'distributed': '+',
  'dorsal': '+',
  'high': '-',
  'low': '-',
  'front': '-',
  'back': '-',
  'tense': '-',
  'pharyngeal': '-',
  'atr': '0',
  'rtr': '0',
  'voice': '+',
  'spread_glottis': '-',
  'constricted_glottis': '-',
  'glottalic_suction': '-',
  'velaric_suction': '-',
  'continuant': '+/-',
  'nasal': '-',
  'strident': '+',
  'lateral': '-',
  'delayed_release': '+'},
 {'syllabic': '+',
  'consonantal': '-',
  'sonorant': '+',
  'approximant': '+',
  'labial': '+',
  'round': '-',
  'protruded': '-',
  'compressed': '-',
  'labiodental': '-',
  'coronal': '-',
  'anterior': '0',
  'distributed': '0',
  'dorsal': '+',
  'high': '+',
  'low': '-',
  'front': '+',
  'back': '-',
  'tense': '-',
  'pharyngeal': '+',
  'atr': '-',
  'rtr': '-',
  'voice': '+',
  'spread_glottis': '-',
  'constricted_glottis': '-',
  'glottalic_suction': '-',
  'velaric_suction': '-',
  'continuant': '+',
  'nasal': '-',
```

(continues on next page)

(continued from previous page)

```
  'strident': '-',
  'lateral': '-',
  'delayed_release': '-'},
 {'syllabic': '-',
  'consonantal': '+',
  'sonorant': '+',
  'approximant': '-',
  'labial': '-',
  'round': '0',
  'protruded': '0',
  'compressed': '0',
  'labiodental': '0',
  'coronal': '+',
  'anterior': '+',
  'distributed': '-',
  'dorsal': '-',
  'high': '0',
  'low': '0',
  'front': '0',
  'back': '0',
  'tense': '0',
  'pharyngeal': '-',
  'atr': '0',
  'rtr': '0',
  'voice': '+',
  'spread_glottis': '-',
  'constricted_glottis': '-',
  'glottalic_suction': '-',
  'velaric_suction': '-',
  'continuant': '-',
  'nasal': '+',
  'strident': '-',
  'lateral': '-',
  'delayed_release': '-'}]
>>> get_feature(feat1, 'consonantal')
[-1, 1, 1, -1, 1, -1, 1, -1, 1, -1, 1]
>>> get_feature(feat1, 'nasal')
[-1, 1, -1, -1, 1, -1, -1, -1, 1, -1, -1]
>>> [cmp_features(f1, f2) for f1, f2 in zip(feat1, feat2)]
[1.0,
 1.0,
 0.9032258064516129,
 1.0,
 1.0,
 1.0,
 1.0,
 0.9193548387096774,
 1.0,
 1.0,
 1.0]
>>> sum(cmp_features(f1, f2) for f1, f2 in zip(feat1, feat2))/len(feat1)
0.9838709677419355
```

abydos.phones.**ipa_to_features**(*ipa*)

Convert IPA to features.

This translates an IPA string of one or more phones to a list of ints representing the features of the string.

Parameters **ipa** (*str*) -- The IPA representation of a phone or series of phones

Returns A representation of the features of the input string

Return type list of ints

Examples

```
>>> ipa_to_features('mut')
[2709662981243185770, 1825831513894594986, 2783230754502126250]
>>> ipa_to_features('fon')
[2781702983095331242, 1825831531074464170, 2711173160463936106]
>>> ipa_to_features('telz')
[2783230754502126250, 1826957430176000426, 2693158761954453926,
2783230754501863834]
```

New in version 0.1.0.

abydos.phones.**ipa_to_feature_dicts**(*ipa*)

Convert IPA to a feature dict list.

This translates an IPA string of one or more phones to a list of dicts representing the features of the string.

Parameters **ipa** (*str*) -- The IPA representation of a phone or series of phones

Returns A representation of the features of the input string

Return type list of dicts

Examples

```
>>> ipa_to_feature_dicts('mut')
[{'syllabic': '-',
  'consonantal': '+',
  'sonorant': '+',
  'approximant': '-',
  'labial': '+',
  'round': '-',
  'protruded': '-',
  'compressed': '-',
  'labiodental': '-',
  'coronal': '-',
  'anterior': '0',
  'distributed': '0',
  'dorsal': '-',
  'high': '0',
  'low': '0',
  'front': '0',
  'back': '0',
  'tense': '0',
  'pharyngeal': '-',
  'atr': '0',
  'rtr': '0',
  'voice': '+',
  'spread_glottis': '-',
  'constricted_glottis': '-',
  'glottalic_suction': '-',
```

(continues on next page)

(continued from previous page)

```
  'velaric_suction': '-',
  'continuant': '-',
  'nasal': '+',
  'strident': '-',
  'lateral': '-',
  'delayed_release': '-'},
 {'syllabic': '+',
  'consonantal': '-',
  'sonorant': '+',
  'approximant': '+',
  'labial': '+',
  'round': '+',
  'protruded': '-',
  'compressed': '-',
  'labiodental': '-',
  'coronal': '-',
  'anterior': '0',
  'distributed': '0',
  'dorsal': '+',
  'high': '+',
  'low': '-',
  'front': '-',
  'back': '+',
  'tense': '+',
  'pharyngeal': '+',
  'atr': '+',
  'rtr': '-',
  'voice': '+',
  'spread_glottis': '-',
  'constricted_glottis': '-',
  'glottalic_suction': '-',
  'velaric_suction': '-',
  'continuant': '+',
  'nasal': '-',
  'strident': '-',
  'lateral': '-',
  'delayed_release': '-'},
 {'syllabic': '-',
  'consonantal': '+',
  'sonorant': '-',
  'approximant': '-',
  'labial': '-',
  'round': '0',
  'protruded': '0',
  'compressed': '0',
  'labiodental': '0',
  'coronal': '+',
  'anterior': '+',
  'distributed': '-',
  'dorsal': '-',
  'high': '0',
  'low': '0',
  'front': '0',
  'back': '0',
  'tense': '0',
  'pharyngeal': '-',
  'atr': '0',
```

(continues on next page)

(continued from previous page)

```
  'rtr': '0',
  'voice': '-',
  'spread_glottis': '-',
  'constricted_glottis': '-',
  'glottalic_suction': '-',
  'velaric_suction': '-',
  'continuant': '-',
  'nasal': '-',
  'strident': '-',
  'lateral': '-',
  'delayed_release': '-'}]
```

New in version 0.4.1.

abydos.phones.**get_feature**(*vector*, *feature*)

Get a feature vector.

This returns a list of ints, equal in length to the vector input, representing presence/absence/neutrality with respect to a particular phonetic feature.

Parameters

- **vector** (*list*) -- A tuple or list of ints representing the phonetic features of a phone or series of phones (such as is returned by the ipa_to_features function)
- **feature** (*str*) -- A feature name from the set:
  - syllabic
  - consonantal
  - sonorant
  - approximant
  - labial
  - round
  - protruded
  - compressed
  - labiodental
  - coronal
  - anterior
  - distributed
  - dorsal
  - high
  - low
  - front
  - back
  - tense
  - pharyngeal
  - atr

- rtr
- voice
- spread_glottis
- constricted_glottis
- glottalic_suction
- velaric_suction
- continuant
- nasal
- strident
- lateral
- delayed_release

Returns A list indicating presence/absence/neutrality with respect to the feature

Return type list of ints

Raises **AttributeError** -- feature must be one of ...

Examples

```
>>> tails = ipa_to_features('telz')
>>> get_feature(tails, 'consonantal')
[1, -1, 1, 1]
>>> get_feature(tails, 'sonorant')
[-1, 1, 1, -1]
>>> get_feature(tails, 'nasal')
[-1, -1, -1, -1]
>>> get_feature(tails, 'coronal')
[1, -1, 1, 1]
```

New in version 0.1.0.

abydos.phones.**cmp_features**(*feat1*, *feat2*, *weights=None*)

Compare features.

This returns a number in the range [0, 1] representing a comparison of two feature bundles.

If one of the bundles is negative, -1 is returned (for unknown values)

If the bundles are identical, 1 is returned.

If they are inverses of one another, 0 is returned.

Otherwise, a float representing their similarity is returned.

Parameters

- **feat1** (*int*) -- A feature bundle
- **feat2** (*int*) -- A feature bundle
- **weights** (*None or list or tuple or dict*) -- If None, all features are of equal significance and a simple normalized hamming distance of the features is calculated. If a list or tuple of numeric values is supplied, the values are inferred as the weights for each feature, in order of the features listed in _FEATURE_MASK. If a dict is supplied, its key

values should match keys in _FEATURE_MASK to which each weight (value) should be assigned. Missing values in all cases are assigned a weight of 0 and will be omitted from the comparison.

Returns A comparison of the feature bundles

Return type float

Examples

```
>>> cmp_features(ipa_to_features('l')[0], ipa_to_features('l')[0])
1.0
>>> cmp_features(ipa_to_features('l')[0], ipa_to_features('n')[0])
0.8709677419354839
>>> cmp_features(ipa_to_features('l')[0], ipa_to_features('z')[0])
0.8709677419354839
>>> cmp_features(ipa_to_features('l')[0], ipa_to_features('i')[0])
0.564516129032258
```

New in version 0.1.0.

Changed in version 0.4.1: Added weights parameter for modifiable feature weighting

3.1.1.6 abydos.phonetic package

abydos.phonetic.

The phonetic package includes classes for phonetic algorithms, including:

- Robert C. Russell's Index (*RussellIndex*)
- American Soundex (*Soundex*)
- Refined Soundex (*RefinedSoundex*)
- Daitch-Mokotoff Soundex (*DaitchMokotoff*)
- NYSIIS (*NYSIIS*)
- Match Rating Algorithm (*phonetic.MRA*)
- Metaphone (*Metaphone*)
- Double Metaphone (*DoubleMetaphone*)
- Caverphone (*Caverphone*)
- Alpha Search Inquiry System (*AlphaSIS*)
- Fuzzy Soundex (*FuzzySoundex*)
- Phonex (*Phonex*)
- Phonem (*Phonem*)
- Phonix (*Phonix*)
- PHONIC (*PHONIC*)
- Standardized Phonetic Frequency Code (*SPFC*)
- Statistics Canada (*StatisticsCanada*)
- LEIN (*LEIN*)

- Roger Root (*RogerRoot*)
- Eudex phonetic hash (*phonetic.Eudex*)
- Parmar-Kumbharana (*ParmarKumbharana*)
- Davidson's Consonant Code (*Davidson*)
- SoundD (*SoundD*)
- PSHP Soundex/Viewex Coding (*PSHPSoundexFirst* and *PSHPSoundexLast*)
- Dolby Code (*Dolby*)
- NRL English-to-phoneme (*NRL*)
- Ainsworth grapheme to phoneme (*Ainsworth*)
- Beider-Morse Phonetic Matching (*BeiderMorse*)

There are also language-specific phonetic algorithms for German:

- Kölner Phonetik (*Koelner*)
- phonet (*Phonet*)
- Haase Phonetik (*Haase*)
- Reth-Schek Phonetik (*RethSchek*)

For French:

- FONEM (*FONEM*)
- an early version of Henry Code (*HenryEarly*)

For Spanish:

- Phonetic Spanish (*PhoneticSpanish*)
- Spanish Metaphone (*SpanishMetaphone*)

For Swedish:

- SfinxBis (*SfinxBis*)
- Wåhlin (*Waahlin*)

For Norwegian:

- Norphone (*Norphone*)

For Brazilian Portuguese:

- SoundexBR (*SoundexBR*)

And there are some hybrid phonetic algorithms that employ multiple underlying phonetic algorithms:

- Oxford Name Compression Algorithm (ONCA) (*ONCA*)
- MetaSoundex (*MetaSoundex*)

Each class has an `encode` method to return the phonetically encoded string. Classes for which `encode` returns a numeric value generally have an `encode_alpha` method that returns an alphabetic version of the phonetic encoding, as demonstrated below:

```
>>> rus = RussellIndex()
>>> rus.encode('Abramson')
128637
>>> rus.encode_alpha('Abramson')
'ABRMCN'
```

---

class abydos.phonetic.**RussellIndex**

Bases: abydos.phonetic._phonetic._Phonetic

Russell Index.

This follows Robert C. Russell's Index algorithm, as described in [Rus18].

New in version 0.3.6.

encode(*word*)

Return the Russell Index (integer output) of a word.

Parameters **word** (*str*) -- The word to transform

Returns The Russell Index value

Return type int

Examples

```
>>> pe = RussellIndex()
>>> pe.encode('Christopher')
3813428
>>> pe.encode('Niall')
715
>>> pe.encode('Smith')
3614
>>> pe.encode('Schmidt')
3614
```

New in version 0.1.0.

Changed in version 0.3.6: Encapsulated in class

encode_alpha(*word*)

Return the Russell Index (alphabetic output) for the word.

This follows Robert C. Russell's Index algorithm, as described in [Rus18].

Parameters **word** (*str*) -- The word to transform

Returns The Russell Index value as an alphabetic string

Return type str

Examples

```
>>> pe = RussellIndex()
>>> pe.encode_alpha('Christopher')
'CRACDBR'
>>> pe.encode_alpha('Niall')
'NAL'
>>> pe.encode_alpha('Smith')
'CMAD'
>>> pe.encode_alpha('Schmidt')
'CMAD'
```

New in version 0.1.0.

Changed in version 0.3.6: Encapsulated in class

abydos.phonetic.**russell_index**(*word*)

Return the Russell Index (integer output) of a word.

This is a wrapper for *RussellIndex.encode()*.

Parameters **word** (*str*) -- The word to transform

Returns The Russell Index value

Return type int

Examples

```
>>> russell_index('Christopher')
3813428
>>> russell_index('Niall')
715
>>> russell_index('Smith')
3614
>>> russell_index('Schmidt')
3614
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the RussellIndex.encode method instead.

abydos.phonetic.**russell_index_num_to_alpha**(*num*)

Convert the Russell Index integer to an alphabetic string.

This is a wrapper for RussellIndex._to_alpha().

Parameters **num** (*int*) -- A Russell Index integer value

Returns The Russell Index as an alphabetic string

Return type str

Examples

```
>>> russell_index_num_to_alpha(3813428)
'CRACDBR'
>>> russell_index_num_to_alpha(715)
'NAL'
>>> russell_index_num_to_alpha(3614)
'CMAD'
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the RussellIndex._to_alpha method instead.

abydos.phonetic.**russell_index_alpha**(*word*)

Return the Russell Index (alphabetic output) for the word.

This is a wrapper for *RussellIndex.encode_alpha()*.

Parameters **word** (*str*) -- The word to transform

Returns The Russell Index value as an alphabetic string

Return type str

Examples

```
>>> russell_index_alpha('Christopher')
'CRACDBR'
>>> russell_index_alpha('Niall')
'NAL'
>>> russell_index_alpha('Smith')
'CMAD'
>>> russell_index_alpha('Schmidt')
'CMAD'
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the RussellIndex.encode_alpha method instead.

class abydos.phonetic.**Soundex**(*max_length=4*, *var='American'*, *reverse=False*, *zero_pad=True*)

Bases: abydos.phonetic._phonetic._Phonetic

Soundex.

Three variants of Soundex are implemented:

- 'American' follows the American Soundex algorithm, as described at [Sta07] and in [Knu98]; this is also called Miracode
- 'special' follows the rules from the 1880-1910 US Census retrospective re-analysis, in which h & w are not treated as blocking consonants but as vowels. Cf. [Rep13].
- 'Census' follows the rules laid out in GIL 55 [Sta97] by the US Census, including coding prefixed and unprefixed versions of some names

New in version 0.3.6.

Initialize Soundex instance.

Parameters

- **max_length** (*int*) -- The length of the code returned (defaults to 4)
- **var** (*str*) -- The variant of the algorithm to employ (defaults to `American`):
  - `American` follows the American Soundex algorithm, as described at [Sta07] and in [Knu98]; this is also called Miracode
  - `special` follows the rules from the 1880-1910 US Census retrospective re-analysis, in which h & w are not treated as blocking consonants but as vowels. Cf. [Rep13].
  - `Census` follows the rules laid out in GIL 55 [Sta97] by the US Census, including coding prefixed and unprefixed versions of some names
- **reverse** (*bool*) -- Reverse the word before computing the selected Soundex (defaults to False); This results in "Reverse Soundex", which is useful for blocking in cases where the initial elements may be in error.
- **zero_pad** (*bool*) -- Pad the end of the return value with 0s to achieve a max_length string

New in version 0.4.0.

encode(*word*)

Return the Soundex code for a word.

Parameters **word** (*str*) -- The word to transform

Returns The Soundex value

Return type str

Examples

```
>>> pe = Soundex()
>>> pe.encode("Christopher")
'C623'
>>> pe.encode("Niall")
'N400'
>>> pe.encode('Smith')
'S530'
>>> pe.encode('Schmidt')
'S530'
```

```
>>> Soundex(max_length=-1).encode('Christopher')
'C623160000000000000000000000000000000000000000000000000000000000000000000000'
>>> Soundex(max_length=-1, zero_pad=False).encode('Christopher')
'C62316'
```

```
>>> Soundex(reverse=True).encode('Christopher')
'R132'
```

```
>>> pe.encode('Ashcroft')
'A261'
>>> pe.encode('Asicroft')
'A226'
```

```
>>> pe_special = Soundex(var='special')
>>> pe_special.encode('Ashcroft')
'A226'
```

(continues on next page)

(continued from previous page)

```
>>> pe_special.encode('Asicroft')
'A226'
```

New in version 0.1.0.

Changed in version 0.3.6: Encapsulated in class

encode_alpha(*word*)

Return the alphabetic Soundex code for a word.

Parameters **word** (*str*) -- The word to transform

Returns The alphabetic Soundex value

Return type str

Examples

```
>>> pe = Soundex()
>>> pe.encode_alpha("Christopher")
'CRKT'
>>> pe.encode_alpha("Niall")
'NL'
>>> pe.encode_alpha('Smith')
'SNT'
>>> pe.encode_alpha('Schmidt')
'SNT'
```

New in version 0.4.0.

abydos.phonetic.**soundex**(*word*, *max_length=4*, *var='American'*, *reverse=False*, *zero_pad=True*)

Return the Soundex code for a word.

This is a wrapper for *Soundex.encode()*.

Parameters

- **word** (*str*) -- The word to transform
- **max_length** (*int*) -- The length of the code returned (defaults to 4)
- **var** (*str*) -- The variant of the algorithm to employ (defaults to `American`):
  - `American` follows the American Soundex algorithm, as described at [Sta07] and in [Knu98]; this is also called Miracode
  - `special` follows the rules from the 1880-1910 US Census retrospective re-analysis, in which h & w are not treated as blocking consonants but as vowels. Cf. [Rep13].
  - `Census` follows the rules laid out in GIL 55 [Sta97] by the US Census, including coding prefixed and unprefixed versions of some names
- **reverse** (*bool*) -- Reverse the word before computing the selected Soundex (defaults to False); This results in "Reverse Soundex", which is useful for blocking in cases where the initial elements may be in error.
- **zero_pad** (*bool*) -- Pad the end of the return value with 0s to achieve a max_length string

Returns The Soundex value

Return type str

Examples

```
>>> soundex("Christopher")
'C623'
>>> soundex("Niall")
'N400'
>>> soundex('Smith')
'S530'
>>> soundex('Schmidt')
'S530'
```

```
>>> soundex('Christopher', max_length=-1)
'C623160000000000000000000000000000000000000000000000000000000000000000000000'
>>> soundex('Christopher', max_length=-1, zero_pad=False)
'C62316'
```

```
>>> soundex('Christopher', reverse=True)
'R132'
```

```
>>> soundex('Ashcroft')
'A261'
>>> soundex('Asicroft')
'A226'
>>> soundex('Ashcroft', var='special')
'A226'
>>> soundex('Asicroft', var='special')
'A226'
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the Soundex.encode method instead.

class abydos.phonetic.**RefinedSoundex**(*max_length=-1*, *zero_pad=False*, *retain_vowels=False*)

Bases: abydos.phonetic._phonetic._Phonetic

Refined Soundex.

This is Soundex, but with more character classes. It was defined at [Boy98].

New in version 0.3.6.

Initialize RefinedSoundex instance.

Parameters

- **max_length** (*int*) -- The length of the code returned (defaults to unlimited)
- **zero_pad** (*bool*) -- Pad the end of the return value with 0s to achieve a max_length string
- **retain_vowels** (*bool*) -- Retain vowels (as 0) in the resulting code

New in version 0.4.0.

encode(*word*)

Return the Refined Soundex code for a word.

Parameters **word** (*str*) -- The word to transform

Returns The Refined Soundex value

Return type str

Examples

```
>>> pe = RefinedSoundex()
>>> pe.encode('Christopher')
'C93619'
>>> pe.encode('Niall')
'N7'
>>> pe.encode('Smith')
'S86'
>>> pe.encode('Schmidt')
'S386'
```

New in version 0.3.0.

Changed in version 0.3.6: Encapsulated in class

encode_alpha(*word*)

Return the alphabetic Refined Soundex code for a word.

Parameters **word** (*str*) -- The word to transform

Returns The alphabetic Refined Soundex value

Return type str

Examples

```
>>> pe = RefinedSoundex()
>>> pe.encode_alpha('Christopher')
'CRKTPR'
>>> pe.encode_alpha('Niall')
'NL'
>>> pe.encode_alpha('Smith')
'SNT'
>>> pe.encode_alpha('Schmidt')
'SKNT'
```

New in version 0.4.0.

abydos.phonetic.**refined_soundex**(*word*, *max_length=-1*, *zero_pad=False*, *retain_vowels=False*)

Return the Refined Soundex code for a word.

This is a wrapper for *RefinedSoundex.encode()*.

Parameters

- **word** (*str*) -- The word to transform
- **max_length** (*int*) -- The length of the code returned (defaults to unlimited)
- **zero_pad** (*bool*) -- Pad the end of the return value with 0s to achieve a max_length string
- **retain_vowels** (*bool*) -- Retain vowels (as 0) in the resulting code

Returns The Refined Soundex value

Return type str

Examples

```
>>> refined_soundex('Christopher')
'C93619'
>>> refined_soundex('Niall')
'N7'
>>> refined_soundex('Smith')
'S86'
>>> refined_soundex('Schmidt')
'S386'
```

New in version 0.3.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the RefinedSoundex.encode method instead.

class abydos.phonetic.**DaitchMokotoff**(*max_length=6*, *zero_pad=True*)

Bases: abydos.phonetic._phonetic._Phonetic

Daitch-Mokotoff Soundex.

Based on Daitch-Mokotoff Soundex [Mok97], this returns values of a word as a set. A collection is necessary since there can be multiple values for a single word.

New in version 0.3.6.

Initialize DaitchMokotoff instance.

Parameters

- **max_length** (*int*) -- The length of the code returned (defaults to 6; must be between 6 and 64)
- **zero_pad** (*bool*) -- Pad the end of the return value with 0s to achieve a max_length string

New in version 0.4.0.

encode(*word*)

Return the Daitch-Mokotoff Soundex code for a word.

Parameters **word** (*str*) -- The word to transform

Returns The Daitch-Mokotoff Soundex value

Return type str

Examples

```
>>> pe = DaitchMokotoff()
>>> sorted(pe.encode('Christopher'))
['494379', '594379']
>>> pe.encode('Niall')
{'680000'}
>>> pe.encode('Smith')
{'463000'}
>>> pe.encode('Schmidt')
{'463000'}
```

```
>>> sorted(DaitchMokotoff(max_length=20,
... zero_pad=False).encode('The quick brown fox'))
['35457976754', '3557976754']
```

New in version 0.1.0.

Changed in version 0.3.6: Encapsulated in class

encode_alpha(*word*)

Return the alphabetic Daitch-Mokotoff Soundex code for a word.

Parameters **word** (*str*) -- The word to transform

Returns The alphabetic Daitch-Mokotoff Soundex value

Return type str

Examples

```
>>> pe = DaitchMokotoff()
>>> sorted(pe.encode_alpha('Christopher'))
['KRSTPR', 'SRSTPR']
>>> pe.encode_alpha('Niall')
{'NL'}
>>> pe.encode_alpha('Smith')
{'SNT'}
>>> pe.encode_alpha('Schmidt')
{'SNT'}
```

```
>>> sorted(DaitchMokotoff(max_length=20,
... zero_pad=False).encode_alpha('The quick brown fox'))
['TKKPRPNPKS', 'TKSKPRPNPKS']
```

New in version 0.4.0.

abydos.phonetic.**dm_soundex**(*word*, *max_length=6*, *zero_pad=True*)

Return the Daitch-Mokotoff Soundex code for a word.

This is a wrapper for *DaitchMokotoff.encode()*.

Parameters

- **word** (*str*) -- The word to transform
- **max_length** (*int*) -- The length of the code returned (defaults to 6; must be between 6 and 64)
- **zero_pad** (*bool*) -- Pad the end of the return value with 0s to achieve a max_length string

Returns The Daitch-Mokotoff Soundex value

Return type str

Examples

```
>>> sorted(dm_soundex('Christopher'))
['494379', '594379']
>>> dm_soundex('Niall')
{'680000'}
>>> dm_soundex('Smith')
{'463000'}
>>> dm_soundex('Schmidt')
{'463000'}
```

```
>>> sorted(dm_soundex('The quick brown fox', max_length=20,
... zero_pad=False))
['35457976754', '3557976754']
```

New in version 0.1.0.

Changed in version 0.3.6: Encapsulated in class

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the DaitchMokotoff.encode method instead.

class abydos.phonetic.**FuzzySoundex**(*max_length=5*, *zero_pad=True*)

Bases: abydos.phonetic._phonetic._Phonetic

Fuzzy Soundex.

Fuzzy Soundex is an algorithm derived from Soundex, defined in [HM02].

New in version 0.3.6.

Initialize FuzzySoundex instance.

Parameters

- **max_length** (*int*) -- The length of the code returned (defaults to 4)
- **zero_pad** (*bool*) -- Pad the end of the return value with 0s to achieve a max_length string

New in version 0.4.0.

encode(*word*)

Return the Fuzzy Soundex code for a word.

Parameters **word** (*str*) -- The word to transform

Returns The Fuzzy Soundex value

Return type str

Examples

```
>>> pe = FuzzySoundex()
>>> pe.encode('Christopher')
'K6931'
>>> pe.encode('Niall')
'N4000'
>>> pe.encode('Smith')
'S5300'
>>> pe.encode('Smith')
'S5300'
```

New in version 0.1.0.

Changed in version 0.3.6: Encapsulated in class

encode_alpha(*word*)

Return the alphabetic Fuzzy Soundex code for a word.

Parameters **word** (*str*) -- The word to transform

Returns The alphabetic Fuzzy Soundex value

Return type str

Examples

```
>>> pe = FuzzySoundex()
>>> pe.encode_alpha('Christopher')
'KRSTP'
>>> pe.encode_alpha('Niall')
'NL'
>>> pe.encode_alpha('Smith')
'SNT'
>>> pe.encode_alpha('Schmidt')
'SNT'
```

New in version 0.4.0.

abydos.phonetic.**fuzzy_soundex**(*word*, *max_length=5*, *zero_pad=True*)

Return the Fuzzy Soundex code for a word.

This is a wrapper for *FuzzySoundex.encode()*.

Parameters

- **word** (*str*) -- The word to transform
- **max_length** (*int*) -- The length of the code returned (defaults to 4)
- **zero_pad** (*bool*) -- Pad the end of the return value with 0s to achieve a max_length string

Returns The Fuzzy Soundex value

Return type str

Examples

```
>>> fuzzy_soundex('Christopher')
'K6931'
>>> fuzzy_soundex('Niall')
'N4000'
>>> fuzzy_soundex('Smith')
'S5300'
>>> fuzzy_soundex('Smith')
'S5300'
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the FuzzySoundex.encode method instead.

class abydos.phonetic.**LEIN**(*max_length=4*, *zero_pad=True*)

Bases: abydos.phonetic._phonetic._Phonetic

LEIN code.

This is Michigan LEIN (Law Enforcement Information Network) name coding, described in [MKTM77].

New in version 0.3.6.

Initialize LEIN instance.

Parameters

- **max_length** (*int*) -- The length of the code returned (defaults to 4)
- **zero_pad** (*bool*) -- Pad the end of the return value with 0s to achieve a max_length string

New in version 0.4.0.

encode(*word*)

Return the LEIN code for a word.

Parameters **word** (*str*) -- The word to transform

Returns The LEIN code

Return type str

Examples

```
>>> pe = LEIN()
>>> pe.encode('Christopher')
'C351'
>>> pe.encode('Niall')
'N300'
>>> pe.encode('Smith')
'S210'
>>> pe.encode('Schmidt')
'S521'
```

New in version 0.3.0.

Changed in version 0.3.6: Encapsulated in class

encode_alpha(*word*)

Return the alphabetic LEIN code for a word.

Parameters **word** (*str*) -- The word to transform

Returns The alphabetic LEIN code

Return type str

Examples

```
>>> pe = LEIN()
>>> pe.encode_alpha('Christopher')
'CLKT'
>>> pe.encode_alpha('Niall')
'NL'
>>> pe.encode_alpha('Smith')
'SNT'
>>> pe.encode_alpha('Schmidt')
'SKNT'
```

New in version 0.4.0.

abydos.phonetic.**lein**(*word*, *max_length=4*, *zero_pad=True*)

Return the LEIN code for a word.

This is a wrapper for *LEIN.encode()*.

Parameters

- **word** (*str*) -- The word to transform
- **max_length** (*int*) -- The length of the code returned (defaults to 4)

- **zero_pad** (*bool*) -- Pad the end of the return value with 0s to achieve a max_length string

Returns The LEIN code

Return type str

Examples

```
>>> lein('Christopher')
'C351'
>>> lein('Niall')
'N300'
>>> lein('Smith')
'S210'
>>> lein('Schmidt')
'S521'
```

New in version 0.3.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the LEIN.encode method instead.

class abydos.phonetic.**Phonex**(*max_length=4*, *zero_pad=True*)

Bases: abydos.phonetic._phonetic._Phonetic

Phonex code.

Phonex is an algorithm derived from Soundex, defined in [LR96].

New in version 0.3.6.

Initialize Phonex instance.

Parameters

- **max_length** (*int*) -- The length of the code returned (defaults to 4)
- **zero_pad** (*bool*) -- Pad the end of the return value with 0s to achieve a max_length string

New in version 0.4.0.

encode(*word*)

Return the Phonex code for a word.

Parameters **word** (*str*) -- The word to transform

Returns The Phonex value

Return type str

Examples

```
>>> pe = Phonex()
>>> pe.encode('Christopher')
'C623'
>>> pe.encode('Niall')
'N400'
>>> pe.encode('Schmidt')
'S253'
>>> pe.encode('Smith')
'S530'
```

New in version 0.1.0.

Changed in version 0.3.6: Encapsulated in class

encode_alpha(*word*)

Return the alphabetic Phonex code for a word.

Parameters **word** (*str*) -- The word to transform

Returns The alphabetic Phonex value

Return type str

Examples

```
>>> pe = Phonex()
>>> pe.encode_alpha('Christopher')
'CRST'
>>> pe.encode_alpha('Niall')
'NL'
>>> pe.encode_alpha('Smith')
'SNT'
>>> pe.encode_alpha('Schmidt')
'SSNT'
```

New in version 0.4.0.

abydos.phonetic.**phonex**(*word*, *max_length=4*, *zero_pad=True*)

Return the Phonex code for a word.

This is a wrapper for *Phonex.encode()*.

Parameters

- **word** (*str*) -- The word to transform
- **max_length** (*int*) -- The length of the code returned (defaults to 4)
- **zero_pad** (*bool*) -- Pad the end of the return value with 0s to achieve a max_length string

Returns The Phonex value

Return type str

Examples

```
>>> phonex('Christopher')
'C623'
>>> phonex('Niall')
'N400'
>>> phonex('Schmidt')
'S253'
>>> phonex('Smith')
'S530'
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the Phonex.encode method instead.

class abydos.phonetic.**PHONIC**(*max_length=5*, *zero_pad=True*, *extended=False*)

Bases: abydos.phonetic._phonetic._Phonetic

PHONIC code.

PHONIC is a Soundex-like algorithm defined in [Taf70].

New in version 0.4.1.

Initialize PHONIC instance.

Parameters

- **max_length** (*int*) -- The length of the code returned (defaults to 5)
- **zero_pad** (*bool*) -- Pad the end of the return value with 0s to achieve a max_length string
- **extended** (*bool*) -- If True, this uses Taft's 'Extended PHONIC coding' mode, which simply omits the first character of the code.

New in version 0.4.1.

encode(*word*)

Return the PHONIC code for a word.

Parameters **word** (*str*) -- The word to transform

Returns The PHONIC code

Return type str

Examples

```
>>> pe = PHONIC()
>>> pe.encode('Christopher')
'C6401'
>>> pe.encode('Niall')
'N2500'
>>> pe.encode('Smith')
'S0310'
>>> pe.encode('Schmidt')
'S0631'
```

New in version 0.4.1.

encode_alpha(*word*)

Return the alphabetic PHONIC code for a word.

Parameters **word** (*str*) -- The word to transform

Returns The alphabetic PHONIC value

Return type str

Examples

```
>>> pe = PHONIC()
>>> pe.encode_alpha('Christopher')
'JRSTF'
>>> pe.encode_alpha('Niall')
'NL'
>>> pe.encode_alpha('Smith')
'SMT'
>>> pe.encode_alpha('Schmidt')
'SJMT'
```

New in version 0.4.1.

class abydos.phonetic.**Phonix**(*max_length=4*, *zero_pad=True*)

Bases: abydos.phonetic._phonetic._Phonetic

Phonix code.

Phonix is a Soundex-like algorithm defined in [Gad90].

This implementation is based on: - [Pfe00] - [Chr11] - [Kollar]

New in version 0.3.6.

Initialize Phonix instance.

Parameters

- **max_length** (*int*) -- The length of the code returned (defaults to 4)
- **zero_pad** (*bool*) -- Pad the end of the return value with 0s to achieve a max_length string

New in version 0.3.6.

encode(*word*)

Return the Phonix code for a word.

Parameters **word** (*str*) -- The word to transform

Returns The Phonix value

Return type str

Examples

```
>>> pe = Phonix()
>>> pe.encode('Christopher')
'K683'
>>> pe.encode('Niall')
'N400'
>>> pe.encode('Smith')
'S530'
>>> pe.encode('Schmidt')
'S530'
```

New in version 0.1.0.

Changed in version 0.3.6: Encapsulated in class

encode_alpha(*word*)

Return the alphabetic Phonix code for a word.

Parameters **word** (*str*) -- The word to transform

Returns The alphabetic Phonix value

Return type str

Examples

```
>>> pe = Phonix()
>>> pe.encode_alpha('Christopher')
'KRST'
>>> pe.encode_alpha('Niall')
'NL'
>>> pe.encode_alpha('Smith')
'SNT'
>>> pe.encode_alpha('Schmidt')
'SNT'
```

New in version 0.4.0.

abydos.phonetic.**phonix**(*word*, *max_length=4*, *zero_pad=True*)

Return the Phonix code for a word.

This is a wrapper for *Phonix.encode()*.

Parameters

- **word** (*str*) -- The word to transform
- **max_length** (*int*) -- The length of the code returned (defaults to 4)
- **zero_pad** (*bool*) -- Pad the end of the return value with 0s to achieve a max_length string

Returns The Phonix value

Return type str

Examples

```
>>> phonix('Christopher')
'K683'
>>> phonix('Niall')
'N400'
>>> phonix('Smith')
'S530'
>>> phonix('Schmidt')
'S530'
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the Phonix.encode method instead.

class abydos.phonetic.**PSHPSoundexFirst**(*max_length=4*, *german=False*)

Bases: abydos.phonetic._phonetic._Phonetic

PSHP Soundex/Viewex Coding of a first name.

This coding is based on [HBD76].

Reference was also made to the German version of the same: [HBD79].

A separate class, *PSHPSoundexLast* is used for last names.

New in version 0.3.6.

Initialize PSHPSoundexFirst instance.

Parameters

- **max_length** (*int*) -- The length of the code returned (defaults to 4)
- **german** (*bool*) -- Set to True if the name is German (different rules apply)

New in version 0.4.0.

encode(*fname*)

Calculate the PSHP Soundex/Viewex Coding of a first name.

Parameters **fname** (*str*) -- The first name to encode

Returns The PSHP Soundex/Viewex Coding

Return type str

Examples

```
>>> pe = PSHPSoundexFirst()
>>> pe.encode('Smith')
'S530'
>>> pe.encode('Waters')
'W352'
>>> pe.encode('James')
'J700'
>>> pe.encode('Schmidt')
'S500'
>>> pe.encode('Ashcroft')
'A220'
>>> pe.encode('John')
'J500'
>>> pe.encode('Colin')
'K400'
>>> pe.encode('Niall')
'N400'
>>> pe.encode('Sally')
'S400'
>>> pe.encode('Jane')
'J500'
```

New in version 0.3.0.

Changed in version 0.3.6: Encapsulated in class

encode_alpha(*fname*)

Calculate the alphabetic PSHP Soundex/Viewex Coding of a first name.

Parameters **fname** (*str*) -- The first name to encode

Returns The alphabetic PSHP Soundex/Viewex Coding

Return type str

Examples

```
>>> pe = PSHPSoundexFirst()
>>> pe.encode_alpha('Smith')
'SNT'
>>> pe.encode_alpha('Waters')
'WTNK'
>>> pe.encode_alpha('James')
'JN'
>>> pe.encode_alpha('Schmidt')
'SN'
>>> pe.encode_alpha('Ashcroft')
'AKK'
>>> pe.encode_alpha('John')
'JN'
>>> pe.encode_alpha('Colin')
'KL'
>>> pe.encode_alpha('Niall')
'NL'
>>> pe.encode_alpha('Sally')
'SL'
>>> pe.encode_alpha('Jane')
'JN'
```

New in version 0.4.0.

abydos.phonetic.**pshp_soundex_first**(*fname*, *max_length=4*, *german=False*)

Calculate the PSHP Soundex/Viewex Coding of a first name.

This is a wrapper for *PSHPSoundexFirst.encode()*.

Parameters

- **fname** (*str*) -- The first name to encode
- **max_length** (*int*) -- The length of the code returned (defaults to 4)
- **german** (*bool*) -- Set to True if the name is German (different rules apply)

Returns The PSHP Soundex/Viewex Coding

Return type str

Examples

```
>>> pshp_soundex_first('Smith')
'S530'
>>> pshp_soundex_first('Waters')
'W352'
>>> pshp_soundex_first('James')
'J700'
>>> pshp_soundex_first('Schmidt')
'S500'
>>> pshp_soundex_first('Ashcroft')
'A220'
>>> pshp_soundex_first('John')
'J500'
>>> pshp_soundex_first('Colin')
'K400'
```

(continues on next page)

(continued from previous page)

```
>>> pshp_soundex_first('Niall')
'N400'
>>> pshp_soundex_first('Sally')
'S400'
>>> pshp_soundex_first('Jane')
'J500'
```

New in version 0.3.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the PSHPSoundexFirst.encode method instead.

class abydos.phonetic.**PSHPSoundexLast**(*max_length=4*, *german=False*)

Bases: abydos.phonetic._phonetic._Phonetic

PSHP Soundex/Viewex Coding of a last name.

This coding is based on [HBD76].

Reference was also made to the German version of the same: [HBD79].

A separate function, *PSHPSoundexFirst* is used for first names.

New in version 0.3.6.

Initialize PSHPSoundexLast instance.

Parameters

- **max_length** (*int*) -- The length of the code returned (defaults to 4)
- **german** (*bool*) -- Set to True if the name is German (different rules apply)

New in version 0.4.0.

encode(*lname*)

Calculate the PSHP Soundex/Viewex Coding of a last name.

Parameters **lname** (*str*) -- The last name to encode

Returns The PSHP Soundex/Viewex Coding

Return type str

Examples

```
>>> pe = PSHPSoundexLast()
>>> pe.encode('Smith')
'S530'
>>> pe.encode('Waters')
'W350'
>>> pe.encode('James')
'J500'
>>> pe.encode('Schmidt')
'S530'
>>> pe.encode('Ashcroft')
'A225'
```

New in version 0.3.0.

Changed in version 0.3.6: Encapsulated in class

encode_alpha(*lname*)

Calculate the alphabetic PSHP Soundex/Viewex Coding of a last name.

Parameters **lname** (*str*) -- The last name to encode

Returns The PSHP alphabetic Soundex/Viewex Coding

Return type str

Examples

```
>>> pe = PSHPSoundexLast()
>>> pe.encode_alpha('Smith')
'SNT'
>>> pe.encode_alpha('Waters')
'WTN'
>>> pe.encode_alpha('James')
'JN'
>>> pe.encode_alpha('Schmidt')
'SNT'
>>> pe.encode_alpha('Ashcroft')
'AKKN'
```

New in version 0.4.0.

abydos.phonetic.**pshp_soundex_last**(*lname*, *max_length=4*, *german=False*)

Calculate the PSHP Soundex/Viewex Coding of a last name.

This is a wrapper for *PSHPSoundexLast.encode()*.

Parameters

- **lname** (*str*) -- The last name to encode
- **max_length** (*int*) -- The length of the code returned (defaults to 4)
- **german** (*bool*) -- Set to True if the name is German (different rules apply)

Returns The PSHP Soundex/Viewex Coding

Return type str

Examples

```
>>> pshp_soundex_last('Smith')
'S530'
>>> pshp_soundex_last('Waters')
'W350'
>>> pshp_soundex_last('James')
'J500'
>>> pshp_soundex_last('Schmidt')
'S530'
>>> pshp_soundex_last('Ashcroft')
'A225'
```

New in version 0.3.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the PSHPSoundexLast.encode method instead.

class abydos.phonetic.**NYSIIS**(*max_length=6*, *modified=False*)

Bases: abydos.phonetic._phonetic._Phonetic

NYSIIS Code.

The New York State Identification and Intelligence System algorithm is defined in [Taf70].

The modified version of this algorithm is described in Appendix B of [LA77].

New in version 0.3.6.

Initialize AlphaSIS instance.

Parameters

- **max_length** (*int*) -- The maximum length (default 6) of the code to return
- **modified** (*bool*) -- Indicates whether to use USDA modified NYSIIS

New in version 0.4.0.

encode(*word*)

Return the NYSIIS code for a word.

Parameters **word** (*str*) -- The word to transform

Returns The NYSIIS value

Return type str

Examples

```
>>> pe = NYSIIS()
>>> pe.encode('Christopher')
'CRASTA'
>>> pe.encode('Niall')
'NAL'
>>> pe.encode('Smith')
'SNAT'
>>> pe.encode('Schmidt')
'SNAD'
```

```
>>> NYSIIS(max_length=-1).encode('Christopher')
'CRASTAFAR'
```

```
>>> pe_8m = NYSIIS(max_length=8, modified=True)
>>> pe_8m.encode('Christopher')
'CRASTAFA'
>>> pe_8m.encode('Niall')
'NAL'
>>> pe_8m.encode('Smith')
'SNAT'
>>> pe_8m.encode('Schmidt')
'SNAD'
```

New in version 0.1.0.

Changed in version 0.3.6: Encapsulated in class

abydos.phonetic.**nysiis**(*word*, *max_length=6*, *modified=False*)

Return the NYSIIS code for a word.

This is a wrapper for *NYSIIS.encode()*.

Parameters

- **word** (*str*) -- The word to transform
- **max_length** (*int*) -- The maximum length (default 6) of the code to return
- **modified** (*bool*) -- Indicates whether to use USDA modified NYSIIS

Returns The NYSIIS value

Return type str

Examples

```
>>> nysiis('Christopher')
'CRASTA'
>>> nysiis('Niall')
'NAL'
>>> nysiis('Smith')
'SNAT'
>>> nysiis('Schmidt')
'SNAD'
```

```
>>> nysiis('Christopher', max_length=-1)
'CRASTAFAR'
```

```
>>> nysiis('Christopher', max_length=8, modified=True)
'CRASTAFA'
>>> nysiis('Niall', max_length=8, modified=True)
'NAL'
>>> nysiis('Smith', max_length=8, modified=True)
'SNAT'
>>> nysiis('Schmidt', max_length=8, modified=True)
'SNAD'
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the NYSIIS.encode method instead.

class abydos.phonetic.**MRA**

Bases: abydos.phonetic._phonetic._Phonetic

Western Airlines Surname Match Rating Algorithm.

A description of the Western Airlines Surname Match Rating Algorithm can be found on page 18 of [MKTM77].

New in version 0.3.6.

encode(*word*)

Return the MRA personal numeric identifier (PNI) for a word.

Parameters **word** (*str*) -- The word to transform

Returns The MRA PNI

Return type str

Examples

```
>>> pe = MRA()
>>> pe.encode('Christopher')
'CHRPHR'
>>> pe.encode('Niall')
'NL'
>>> pe.encode('Smith')
'SMTH'
>>> pe.encode('Schmidt')
'SCHMDT'
```

New in version 0.1.0.

Changed in version 0.3.6: Encapsulated in class

abydos.phonetic.**mra**(*word*)

Return the MRA personal numeric identifier (PNI) for a word.

This is a wrapper for *MRA.encode()*.

Parameters **word** (*str*) -- The word to transform

Returns The MRA PNI

Return type str

Examples

```
>>> mra('Christopher')
'CHRPHR'
>>> mra('Niall')
'NL'
>>> mra('Smith')
'SMTH'
>>> mra('Schmidt')
'SCHMDT'
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the MRA.encode method instead.

class abydos.phonetic.**Caverphone**(*version=2*)

Bases: abydos.phonetic._phonetic._Phonetic

Caverphone.

A description of version 1 of the algorithm can be found in [Hoo02].

A description of version 2 of the algorithm can be found in [Hoo04].

New in version 0.3.6.

Initialize Caverphone instance.

Parameters **version** (*int*) -- The version of Caverphone to employ for encoding (defaults to 2)

New in version 0.4.0.

encode(*word*)

Return the Caverphone code for a word.

Parameters **word** (*str*) -- The word to transform

Returns The Caverphone value

Return type str

Examples

```
>>> pe = Caverphone()
>>> pe.encode('Christopher')
'KRSTFA1111'
>>> pe.encode('Niall')
'NA11111111'
>>> pe.encode('Smith')
'SMT1111111'
>>> pe.encode('Schmidt')
'SKMT111111'
```

```
>>> pe_1 = Caverphone(version=1)
>>> pe_1.encode('Christopher')
'KRSTF1'
>>> pe_1.encode('Niall')
'N11111'
>>> pe_1.encode('Smith')
'SMT111'
>>> pe_1.encode('Schmidt')
'SKMT11'
```

New in version 0.1.0.

Changed in version 0.3.6: Encapsulated in class

encode_alpha(*word*)

Return the alphabetic Caverphone code for a word.

Parameters **word** (*str*) -- The word to transform

Returns The alphabetic Caverphone value

Return type str

Examples

```
>>> pe = Caverphone()
>>> pe.encode_alpha('Christopher')
'KRSTFA'
>>> pe.encode_alpha('Niall')
'NA'
>>> pe.encode_alpha('Smith')
'SMT'
>>> pe.encode_alpha('Schmidt')
'SKMT'
```

```
>>> pe_1 = Caverphone(version=1)
>>> pe_1.encode_alpha('Christopher')
'KRSTF'
>>> pe_1.encode_alpha('Niall')
'N'
```

(continues on next page)

(continued from previous page)

```
>>> pe_1.encode_alpha('Smith')
'SMT'
>>> pe_1.encode_alpha('Schmidt')
'SKMT'
```

New in version 0.4.0.

abydos.phonetic.**caverphone**(*word*, *version=2*)

Return the Caverphone code for a word.

This is a wrapper for *Caverphone.encode()*.

Parameters

- **word** (*str*) -- The word to transform
- **version** (*int*) -- The version of Caverphone to employ for encoding (defaults to 2)

Returns The Caverphone value

Return type str

Examples

```
>>> caverphone('Christopher')
'KRSTFA1111'
>>> caverphone('Niall')
'NA11111111'
>>> caverphone('Smith')
'SMT1111111'
>>> caverphone('Schmidt')
'SKMT111111'
```

```
>>> caverphone('Christopher', 1)
'KRSTF1'
>>> caverphone('Niall', 1)
'N11111'
>>> caverphone('Smith', 1)
'SMT111'
>>> caverphone('Schmidt', 1)
'SKMT11'
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the Caverphone.encode method instead.

class abydos.phonetic.**AlphaSIS**(*max_length=14*)

Bases: abydos.phonetic._phonetic._Phonetic

Alpha-SIS.

The Alpha Search Inquiry System code is defined in [Cor73]. This implementation is based on the description in [MKTM77].

New in version 0.3.6.

Initialize AlphaSIS instance.

Parameters **max_length** (*int*) -- The length of the code returned (defaults to 14)

New in version 0.4.0.

encode(*word*)

Return the IBM Alpha Search Inquiry System code for a word.

A collection is necessary as the return type since there can be multiple values for a single word. But the collection must be ordered since the first value is the primary coding.

Parameters **word** (*str*) -- The word to transform

Returns The Alpha-SIS value

Return type tuple

Examples

```
>>> pe = AlphaSIS()
>>> pe.encode('Christopher')
('06401840000000', '07040184000000', '04018400000000')
>>> pe.encode('Niall')
('02500000000000',)
>>> pe.encode('Smith')
('03100000000000',)
>>> pe.encode('Schmidt')
('06310000000000',)
```

New in version 0.1.0.

Changed in version 0.3.6: Encapsulated in class

encode_alpha(*word*)

Return the alphabetic Alpha-SIS code for a word.

Parameters **word** (*str*) -- The word to transform

Returns The alphabetic Alpha-SIS value

Return type tuple

Examples

```
>>> pe = AlphaSIS()
>>> pe.encode_alpha('Christopher')
('JRSTFR', 'KSRSTFR', 'RSTFR')
>>> pe.encode_alpha('Niall')
('NL',)
>>> pe.encode_alpha('Smith')
('MT',)
>>> pe.encode_alpha('Schmidt')
('JMT',)
```

New in version 0.4.0.

abydos.phonetic.**alpha_sis**(*word*, *max_length=14*)

Return the IBM Alpha Search Inquiry System code for a word.

This is a wrapper for *AlphaSIS.encode()*.

Parameters

- **word** (*str*) -- The word to transform

- **max_length** (*int*) -- The length of the code returned (defaults to 14)

Returns The Alpha-SIS value

Return type tuple

Examples

```
>>> alpha_sis('Christopher')
('06401840000000', '07040184000000', '04018400000000')
>>> alpha_sis('Niall')
('02500000000000',)
>>> alpha_sis('Smith')
('03100000000000',)
>>> alpha_sis('Schmidt')
('06310000000000',)
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the AlphaSIS.encode method instead.

class abydos.phonetic.**Davidson**(*omit_fname=False*)

Bases: abydos.phonetic._phonetic._Phonetic

Davidson Consonant Code.

This is based on the name compression system described in [Dav62].

[Dol70] identifies this as having been the name compression algorithm used by SABRE.

New in version 0.3.6.

Initialize Davidson instance.

Parameters **omit_fname** (*bool*) -- Set to True to completely omit the first character of the first name

New in version 0.4.0.

encode(*lname*, *fname='.'*)

Return Davidson's Consonant Code.

Parameters

- **lname** (*str*) -- Last name (or word) to be encoded
- **fname** (*str*) -- First name (optional), of which the first character is included in the code.

Returns Davidson's Consonant Code

Return type str

Example

```
>>> pe = Davidson()
>>> pe.encode('Gough')
'G   .'
>>> pe.encode('pneuma')
'PNM .'
>>> pe.encode('knight')
'KNGT.'
>>> pe.encode('trice')
'TRC .'
>>> pe.encode('judge')
'JDG .'
>>> pe.encode('Smith', 'James')
'SMT J'
>>> pe.encode('Wasserman', 'Tabitha')
'WSRMT'
```

New in version 0.3.0.

Changed in version 0.3.6: Encapsulated in class

abydos.phonetic.**davidson**(*lname*, *fname='.'*, *omit_fname=False*)

Return Davidson's Consonant Code.

This is a wrapper for *Davidson.encode()*.

Parameters

- **lname** (*str*) -- Last name (or word) to be encoded
- **fname** (*str*) -- First name (optional), of which the first character is included in the code.
- **omit_fname** (*bool*) -- Set to True to completely omit the first character of the first name

Returns Davidson's Consonant Code

Return type str

Example

```
>>> davidson('Gough')
'G   .'
>>> davidson('pneuma')
'PNM .'
>>> davidson('knight')
'KNGT.'
>>> davidson('trice')
'TRC .'
>>> davidson('judge')
'JDG .'
>>> davidson('Smith', 'James')
'SMT J'
>>> davidson('Wasserman', 'Tabitha')
'WSRMT'
```

New in version 0.3.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the Davidson.encode method instead.

class abydos.phonetic.**Dolby**(*max_length=-1*, *keep_vowels=False*, *vowel_char='*'*)

Bases: abydos.phonetic._phonetic._Phonetic

Dolby Code.

This follows "A Spelling Equivalent Abbreviation Algorithm For Personal Names" from [Dol70] and [C+69].

New in version 0.3.6.

Initialize Dolby instance.

Parameters

- **max_length** (*int*) -- Maximum length of the returned Dolby code -- this also activates the fixed-length code mode if it is greater than 0
- **keep_vowels** (*bool*) -- If True, retains all vowel markers
- **vowel_char** (*str*) -- The vowel marker character (default to *)

New in version 0.4.0.

encode(*word*)

Return the Dolby Code of a name.

Parameters **word** (*str*) -- The word to transform

Returns The Dolby Code

Return type str

Examples

```
>>> pe = Dolby()
>>> pe.encode('Hansen')
'H*NSN'
>>> pe.encode('Larsen')
'L*RSN'
>>> pe.encode('Aagaard')
'*GR'
>>> pe.encode('Braaten')
'BR*DN'
>>> pe.encode('Sandvik')
'S*NVK'
```

```
>>> pe_6 = Dolby(max_length=6)
>>> pe_6.encode('Hansen')
'H*NS*N'
>>> pe_6.encode('Larsen')
'L*RS*N'
>>> pe_6.encode('Aagaard')
'*G*R  '
>>> pe_6.encode('Braaten')
'BR*D*N'
>>> pe_6.encode('Sandvik')
'S*NF*K'
```

```
>>> pe.encode('Smith')
'SM*D'
>>> pe.encode('Waters')
```

(continues on next page)

(continued from previous page)

```
'W*DRS'
>>> pe.encode('James')
'J*MS'
>>> pe.encode('Schmidt')
'SM*D'
>>> pe.encode('Ashcroft')
'*SKRFD'
```

```
>>> pe_6.encode('Smith')
'SM*D  '
>>> pe_6.encode('Waters')
'W*D*RS'
>>> pe_6.encode('James')
'J*M*S '
>>> pe_6.encode('Schmidt')
'SM*D  '
>>> pe_6.encode('Ashcroft')
'*SKRFD'
```

New in version 0.3.0.

Changed in version 0.3.6: Encapsulated in class

encode_alpha(*word*)

Return the alphabetic Dolby Code of a name.

Parameters **word** (*str*) -- The word to transform

Returns The alphabetic Dolby Code

Return type str

Examples

```
>>> pe = Dolby()
>>> pe.encode_alpha('Hansen')
'HANSN'
>>> pe.encode_alpha('Larsen')
'LARSN'
>>> pe.encode_alpha('Aagaard')
'AGR'
>>> pe.encode_alpha('Braaten')
'BRADN'
>>> pe.encode_alpha('Sandvik')
'SANVK'
```

New in version 0.4.0.

abydos.phonetic.**dolby**(*word*, *max_length=-1*, *keep_vowels=False*, *vowel_char='*'*)

Return the Dolby Code of a name.

This is a wrapper for *Dolby.encode()*.

Parameters

- **word** (*str*) -- The word to transform
- **max_length** (*int*) -- Maximum length of the returned Dolby code -- this also activates the fixed-length code mode if it is greater than 0

- **keep_vowels** (*bool*) -- If True, retains all vowel markers
- **vowel_char** (*str*) -- The vowel marker character (default to *)

Returns The Dolby Code

Return type str

Examples

```
>>> dolby('Hansen')
'H*NSN'
>>> dolby('Larsen')
'L*RSN'
>>> dolby('Aagaard')
'*GR'
>>> dolby('Braaten')
'BR*DN'
>>> dolby('Sandvik')
'S*NVK'
>>> dolby('Hansen', max_length=6)
'H*NS*N'
>>> dolby('Larsen', max_length=6)
'L*RS*N'
>>> dolby('Aagaard', max_length=6)
'*G*R  '
>>> dolby('Braaten', max_length=6)
'BR*D*N'
>>> dolby('Sandvik', max_length=6)
'S*NF*K'
```

```
>>> dolby('Smith')
'SM*D'
>>> dolby('Waters')
'W*DRS'
>>> dolby('James')
'J*MS'
>>> dolby('Schmidt')
'SM*D'
>>> dolby('Ashcroft')
'*SKRFD'
>>> dolby('Smith', max_length=6)
'SM*D  '
>>> dolby('Waters', max_length=6)
'W*D*RS'
>>> dolby('James', max_length=6)
'J*M*S '
>>> dolby('Schmidt', max_length=6)
'SM*D  '
>>> dolby('Ashcroft', max_length=6)
'*SKRFD'
```

New in version 0.3.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the Dolby.encode method instead.

class abydos.phonetic.**SPFC**

Bases: abydos.phonetic._phonetic._Phonetic

Standardized Phonetic Frequency Code (SPFC).

Standardized Phonetic Frequency Code is roughly Soundex-like. This implementation is based on page 19-21 of [MKTM77].

New in version 0.3.6.

encode(*word*)

Return the Standardized Phonetic Frequency Code (SPFC) of a word.

Parameters **word** (*str*) -- The word to transform

Returns The SPFC value

Return type str

Raises **AttributeError** -- Word attribute must be a string with a space or period dividing the first and last names or a tuple/list consisting of the first and last names

Examples

```
>>> pe = SPFC()
>>> pe.encode('Christopher Smith')
'01160'
>>> pe.encode('Christopher Schmidt')
'01160'
>>> pe.encode('Niall Smith')
'01660'
>>> pe.encode('Niall Schmidt')
'01660'
```

```
>>> pe.encode('L.Smith')
'01960'
>>> pe.encode('R.Miller')
'65490'
```

```
>>> pe.encode(('L', 'Smith'))
'01960'
>>> pe.encode(('R', 'Miller'))
'65490'
```

New in version 0.1.0.

Changed in version 0.3.6: Encapsulated in class

encode_alpha(*word*)

Return the alphabetic SPFC of a word.

Parameters **word** (*str*) -- The word to transform

Returns The alphabetic SPFC value

Return type str

Examples

```
>>> pe = SPFC()
>>> pe.encode_alpha('Christopher Smith')
'SDCMS'
>>> pe.encode_alpha('Christopher Schmidt')
'SDCMS'
>>> pe.encode_alpha('Niall Smith')
'SDMMS'
>>> pe.encode_alpha('Niall Schmidt')
'SDMMS'
```

```
>>> pe.encode_alpha('L.Smith')
'SDEMS'
>>> pe.encode_alpha('R.Miller')
'EROES'
```

```
>>> pe.encode_alpha(('L', 'Smith'))
'SDEMS'
>>> pe.encode_alpha(('R', 'Miller'))
'EROES'
```

New in version 0.4.0.

abydos.phonetic.**spfc**(*word*)

Return the Standardized Phonetic Frequency Code (SPFC) of a word.

This is a wrapper for *SPFC.encode()*.

Parameters **word** (*str*) -- The word to transform

Returns The SPFC value

Return type str

Examples

```
>>> spfc('Christopher Smith')
'01160'
>>> spfc('Christopher Schmidt')
'01160'
>>> spfc('Niall Smith')
'01660'
>>> spfc('Niall Schmidt')
'01660'
```

```
>>> spfc('L.Smith')
'01960'
>>> spfc('R.Miller')
'65490'
```

```
>>> spfc(('L', 'Smith'))
'01960'
>>> spfc(('R', 'Miller'))
'65490'
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the SPFC.encode method instead.

class abydos.phonetic.**RogerRoot**(*max_length=5*, *zero_pad=True*)
Bases: abydos.phonetic._phonetic._Phonetic

Roger Root code.

This is Roger Root name coding, described in [MKTM77].

New in version 0.3.6.

Initialize RogerRoot instance.

Parameters

- **max_length** (*int*) -- The maximum length (default 5) of the code to return
- **zero_pad** (*bool*) -- Pad the end of the return value with 0s to achieve a max_length string

New in version 0.4.0.

encode(*word*)
Return the Roger Root code for a word.

Parameters **word** (*str*) -- The word to transform

Returns The Roger Root code

Return type str

Examples

```
>>> pe = RogerRoot()
>>> pe.encode('Christopher')
'06401'
>>> pe.encode('Niall')
'02500'
>>> pe.encode('Smith')
'00310'
>>> pe.encode('Schmidt')
'06310'
```

New in version 0.3.0.

Changed in version 0.3.6: Encapsulated in class

encode_alpha(*word*)
Return the alphabetic Roger Root code for a word.

Parameters **word** (*str*) -- The word to transform

Returns The alphabetic Roger Root code

Return type str

Examples

```
>>> pe = RogerRoot()
>>> pe.encode_alpha('Christopher')
'JRST'
>>> pe.encode_alpha('Niall')
'NL'
>>> pe.encode_alpha('Smith')
'SMT'
>>> pe.encode_alpha('Schmidt')
'JMT'
```

New in version 0.4.0.

abydos.phonetic.**roger_root**(*word*, *max_length=5*, *zero_pad=True*)

Return the Roger Root code for a word.

This is a wrapper for *RogerRoot.encode()*.

Parameters

- **word** (*str*) -- The word to transform
- **max_length** (*int*) -- The maximum length (default 5) of the code to return
- **zero_pad** (*bool*) -- Pad the end of the return value with 0s to achieve a max_length string

Returns The Roger Root code

Return type str

Examples

```
>>> roger_root('Christopher')
'06401'
>>> roger_root('Niall')
'02500'
>>> roger_root('Smith')
'00310'
>>> roger_root('Schmidt')
'06310'
```

New in version 0.3.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the RogerRoot.encode method instead.

class abydos.phonetic.**StatisticsCanada**(*max_length=4*)

Bases: abydos.phonetic._phonetic._Phonetic

Statistics Canada code.

The original description of this algorithm could not be located, and may only have been specified in an unpublished TR. The coding does not appear to be in use by Statistics Canada any longer. In its place, this is an implementation of the "Census modified Statistics Canada name coding procedure".

The modified version of this algorithm is described in Appendix B of [MKTM77].

New in version 0.3.6.

Initialize StatisticsCanada instance.

Parameters **max_length** (*int*) -- The length of the code returned (defaults to 4)

New in version 0.4.0.

encode(*word*)

Return the Statistics Canada code for a word.

Parameters **word** (*str*) -- The word to transform

Returns The Statistics Canada name code value

Return type str

Examples

```
>>> pe = StatisticsCanada()
>>> pe.encode('Christopher')
'CHRS'
>>> pe.encode('Niall')
'NL'
>>> pe.encode('Smith')
'SMTH'
>>> pe.encode('Schmidt')
'SCHM'
```

New in version 0.3.0.

Changed in version 0.3.6: Encapsulated in class

abydos.phonetic.**statistics_canada**(*word*, *max_length=4*)

Return the Statistics Canada code for a word.

This is a wrapper for *StatisticsCanada.encode()*.

Parameters

- **word** (*str*) -- The word to transform
- **max_length** (*int*) -- The maximum length (default 4) of the code to return

Returns The Statistics Canada name code value

Return type str

Examples

```
>>> statistics_canada('Christopher')
'CHRS'
>>> statistics_canada('Niall')
'NL'
>>> statistics_canada('Smith')
'SMTH'
>>> statistics_canada('Schmidt')
'SCHM'
```

New in version 0.3.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the StatisticsCanada.encode method instead.

class abydos.phonetic.**SoundD**(*max_length=4*)

Bases: abydos.phonetic._phonetic._Phonetic

SoundD code.

SoundD is defined in [VB12].

New in version 0.3.6.

Initialize SoundD instance.

Parameters **max_length** (*int*) -- The length of the code returned (defaults to 4)

New in version 0.4.0.

encode(*word*)

Return the SoundD code.

Parameters **word** (*str*) -- The word to transform

Returns The SoundD code

Return type str

Examples

```
>>> pe = SoundD()
>>> pe.encode('Gough')
'2000'
>>> pe.encode('pneuma')
'5500'
>>> pe.encode('knight')
'5300'
>>> pe.encode('trice')
'3620'
>>> pe.encode('judge')
'2200'
```

New in version 0.3.0.

Changed in version 0.3.6: Encapsulated in class

encode_alpha(*word*)

Return the alphabetic SoundD code.

Parameters **word** (*str*) -- The word to transform

Returns The alphabetic SoundD code

Return type str

Examples

```
>>> pe = SoundD()
>>> pe.encode_alpha('Gough')
'K'
>>> pe.encode_alpha('pneuma')
'NN'
>>> pe.encode_alpha('knight')
'NT'
>>> pe.encode_alpha('trice')
'TRK'
>>> pe.encode_alpha('judge')
'KK'
```

New in version 0.4.0.

abydos.phonetic.**sound_d**(*word*, *max_length=4*)

Return the SoundD code.

Parameters

- **word** (*str*) -- The word to transform
- **max_length** (*int*) -- The length of the code returned (defaults to 4)

Returns The SoundD code

Return type str

Examples

```
>>> sound_d('Gough')
'2000'
>>> sound_d('pneuma')
'5500'
>>> sound_d('knight')
'5300'
>>> sound_d('trice')
'3620'
>>> sound_d('judge')
'2200'
```

New in version 0.3.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the SoundD.encode method instead.

class abydos.phonetic.**ParmarKumbharana**

Bases: abydos.phonetic._phonetic._Phonetic

Parmar-Kumbharana code.

This is based on the phonetic algorithm proposed in [PK14].

New in version 0.3.6.

encode(*word*)

Return the Parmar-Kumbharana encoding of a word.

Parameters **word** (*str*) -- The word to transform

Returns The Parmar-Kumbharana encoding

Return type str

Examples

```
>>> pe = ParmarKumbharana()
>>> pe.encode('Gough')
'GF'
>>> pe.encode('pneuma')
'NM'
>>> pe.encode('knight')
'NT'
>>> pe.encode('trice')
```

(continues on next page)

(continued from previous page)

```
'TRS'
>>> pe.encode('judge')
'JJ'
```

New in version 0.3.0.

Changed in version 0.3.6: Encapsulated in class

abydos.phonetic.**parmar_kumbharana**(*word*)

Return the Parmar-Kumbharana encoding of a word.

This is a wrapper for *ParmarKumbharana.encode()*.

Parameters **word** (*str*) -- The word to transform

Returns The Parmar-Kumbharana encoding

Return type str

Examples

```
>>> parmar_kumbharana('Gough')
'GF'
>>> parmar_kumbharana('pneuma')
'NM'
>>> parmar_kumbharana('knight')
'NT'
>>> parmar_kumbharana('trice')
'TRS'
>>> parmar_kumbharana('judge')
'JJ'
```

New in version 0.3.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the ParmarKumbharana.encode method instead.

class abydos.phonetic.**Metaphone**(*max_length=-1*)

Bases: abydos.phonetic._phonetic._Phonetic

Metaphone.

Based on Lawrence Philips' Pick BASIC code from 1990 [Phi90b], as described in [Phi90a]. This incorporates some corrections to the above code, particularly some of those suggested by Michael Kuhn in [Kuh95].

New in version 0.3.6.

Initialize AlphaSIS instance.

Parameters **max_length** (*int*) -- The maximum length of the returned Metaphone code (defaults to 64, but in Philips' original implementation this was 4)

New in version 0.4.0.

encode(*word*)

Return the Metaphone code for a word.

Based on Lawrence Philips' Pick BASIC code from 1990 [Phi90b], as described in [Phi90a]. This incorporates some corrections to the above code, particularly some of those suggested by Michael Kuhn in [Kuh95].

Parameters **word** (*str*) -- The word to transform

Returns The Metaphone value

Return type str

Examples

```
>>> pe = Metaphone()
>>> pe.encode('Christopher')
'KRSTFR'
>>> pe.encode('Niall')
'NL'
>>> pe.encode('Smith')
'SM0'
>>> pe.encode('Schmidt')
'SKMTT'
```

New in version 0.1.0.

Changed in version 0.3.6: Encapsulated in class

abydos.phonetic.**metaphone**(*word*, *max_length=-1*)

Return the Metaphone code for a word.

This is a wrapper for *Metaphone.encode()*.

Parameters

- **word** (*str*) -- The word to transform
- **max_length** (*int*) -- The maximum length of the returned Metaphone code (defaults to 64, but in Philips' original implementation this was 4)

Returns The Metaphone value

Return type str

Examples

```
>>> metaphone('Christopher')
'KRSTFR'
>>> metaphone('Niall')
'NL'
>>> metaphone('Smith')
'SM0'
>>> metaphone('Schmidt')
'SKMTT'
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the Metaphone.encode method instead.

class abydos.phonetic.**DoubleMetaphone**(*max_length=-1*)

Bases: abydos.phonetic._phonetic._Phonetic

Double Metaphone.

Based on Lawrence Philips' (Visual) C++ code from 1999 [Phi00].

New in version 0.3.6.

Initialize DoubleMetaphone instance.

Parameters **max_length** (*int*) -- Maximum length of the returned Dolby code -- this also activates the fixed-length code mode if it is greater than 0

New in version 0.4.0.

encode(*word*)

Return the Double Metaphone code for a word.

Parameters **word** (*str*) -- The word to transform

Returns The Double Metaphone value(s)

Return type tuple

Examples

```
>>> pe = DoubleMetaphone()
>>> pe.encode('Christopher')
('KRSTFR', '')
>>> pe.encode('Niall')
('NL', '')
>>> pe.encode('Smith')
('SM0', 'XMT')
>>> pe.encode('Schmidt')
('XMT', 'SMT')
```

New in version 0.1.0.

Changed in version 0.3.6: Encapsulated in class

encode_alpha(*word*)

Return the alphabetic Double Metaphone code for a word.

Parameters **word** (*str*) -- The word to transform

Returns The alphabetic Double Metaphone value(s)

Return type tuple

Examples

```
>>> pe = DoubleMetaphone()
>>> pe.encode_alpha('Christopher')
('KRSTFR', '')
>>> pe.encode_alpha('Niall')
('NL', '')
>>> pe.encode_alpha('Smith')
('SMÞ', 'XMT')
>>> pe.encode_alpha('Schmidt')
('XMT', 'SMT')
```

New in version 0.4.0.

abydos.phonetic.**double_metaphone**(*word*, *max_length=-1*)

Return the Double Metaphone code for a word.

This is a wrapper for *DoubleMetaphone.encode()*.

Parameters

- **word** (*str*) -- The word to transform
- **max_length** (*int*) -- The maximum length of the returned Double Metaphone codes (defaults to unlimited, but in Philips' original implementation this was 4)

Returns The Double Metaphone value(s)

Return type tuple

Examples

```
>>> double_metaphone('Christopher')
('KRSTFR', '')
>>> double_metaphone('Niall')
('NL', '')
>>> double_metaphone('Smith')
('SM0', 'XMT')
>>> double_metaphone('Schmidt')
('XMT', 'SMT')
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the DoubleMetaphone.encode method instead.

class abydos.phonetic.**Eudex**(*max_length=8*)

Bases: abydos.phonetic._phonetic._Phonetic

Eudex hash.

This implementation of eudex phonetic hashing is based on the specification (not the reference implementation) at [Tic].

Further details can be found at [Tic16].

New in version 0.3.6.

Initialize Eudex instance.

Parameters **max_length** (*int*) -- The length in bits of the code returned (default 8)

New in version 0.4.0.

encode(*word*)

Return the eudex phonetic hash of a word.

Parameters **word** (*str*) -- The word to transform

Returns The eudex hash

Return type int

Examples

```
>>> pe = Eudex()
>>> pe.encode('Colin')
432345564238053650
>>> pe.encode('Christopher')
433648490138894409
>>> pe.encode('Niall')
648518346341351840
>>> pe.encode('Smith')
720575940412906756
>>> pe.encode('Schmidt')
720589151732307997
```

New in version 0.3.0.

Changed in version 0.3.6: Encapsulated in class

abydos.phonetic.**eudex**(*word*, *max_length=8*)

Return the eudex phonetic hash of a word.

This is a wrapper for *Eudex.encode()*.

Parameters

- **word** (*str*) -- The word to transform
- **max_length** (*int*) -- The length in bits of the code returned (default 8)

Returns The eudex hash

Return type int

Examples

```
>>> eudex('Colin')
432345564238053650
>>> eudex('Christopher')
433648490138894409
>>> eudex('Niall')
648518346341351840
>>> eudex('Smith')
720575940412906756
>>> eudex('Schmidt')
720589151732307997
```

New in version 0.3.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the Eudex.encode method instead.

class abydos.phonetic.**BeiderMorse**(*language_arg=0*, *name_mode='gen'*, *match_mode='approx'*, *concat=False*, *filter_langs=False*)

Bases: abydos.phonetic._phonetic._Phonetic

Beider-Morse Phonetic Matching.

The Beider-Morse Phonetic Matching algorithm is described in [BM08]. The reference implementation is licensed under GPLv3.

New in version 0.3.6.

Initialize BeiderMorse instance.

Parameters

- **language_arg** (*str or int*) -- The language of the term; supported values include:
  - any
  - arabic
  - cyrillic
  - czech
  - dutch
  - english
  - french
  - german
  - greek
  - greeklatin
  - hebrew
  - hungarian
  - italian
  - latvian
  - polish
  - portuguese
  - romanian
  - russian
  - spanish
  - turkish
- **name_mode** (*str*) -- The name mode of the algorithm:
  - gen -- general (default)
  - ash -- Ashkenazi
  - sep -- Sephardic
- **match_mode** (*str*) -- Matching mode: approx or exact
- **concat** (*bool*) -- Concatenation mode
- **filter_langs** (*bool*) -- Filter out incompatible languages

New in version 0.4.0.

encode(*word*)

Return the Beider-Morse Phonetic Matching encoding(s) of a term.

Parameters **word** (*str*) -- The word to transform

Returns The Beider-Morse phonetic value(s)

Return type tuple

Raises **ValueError** -- Unknown language

Examples

```
>>> pe = BeiderMorse()
>>> pe.encode('Christopher')
'xrQstopir xrQstYpir xristopir xristYpir xrQstofir xrQstYfir
xristofir xristYfir xristopi xritopir xritopi xristofi xritofir
xritofi tzristopir tzristofir zristopir zristopi zritopir zritopi
zristofir zristofi zritofir zritofi'
>>> pe.encode('Niall')
'nial niol'
>>> pe.encode('Smith')
'zmit'
>>> pe.encode('Schmidt')
'zmit stzmit'
```

```
>>> BeiderMorse(language_arg='German').encode('Christopher')
'xrQstopir xrQstYpir xristopir xristYpir xrQstofir xrQstYfir
xristofir xristYfir'
>>> BeiderMorse(language_arg='English').encode('Christopher')
'tzristofir tzrQstofir tzristafir tzrQstafir xristofir xrQstofir
xristafir xrQstafir'
>>> BeiderMorse(language_arg='German',
... name_mode='ash').encode('Christopher')
'xrQstopir xrQstYpir xristopir xristYpir xrQstofir xrQstYfir
xristofir xristYfir'
```

```
>>> BeiderMorse(language_arg='German',
... match_mode='exact').encode('Christopher')
'xriStopher xriStofer xristopher xristofer'
```

New in version 0.1.0.

Changed in version 0.3.6: Encapsulated in class

abydos.phonetic.**bmpm**(*word*, *language_arg=0*, *name_mode='gen'*, *match_mode='approx'*, *concat=False*, *filter_langs=False*)

Return the Beider-Morse Phonetic Matching encoding(s) of a term.

This is a wrapper for *BeiderMorse.encode()*.

Parameters

- **word** (*str*) -- The word to transform
- **language_arg** (*str*) -- The language of the term; supported values include:
  - any
  - arabic
  - cyrillic
  - czech
  - dutch
  - english
  - french
  - german
  - greek

  - greeklatin
  - hebrew
  - hungarian
  - italian
  - latvian
  - polish
  - portuguese
  - romanian
  - russian
  - spanish
  - turkish
- **name_mode** (*str*) -- The name mode of the algorithm:
  - gen -- general (default)
  - ash -- Ashkenazi
  - sep -- Sephardic
- **match_mode** (*str*) -- Matching mode: approx or exact
- **concat** (*bool*) -- Concatenation mode
- **filter_langs** (*bool*) -- Filter out incompatible languages

Returns The Beider-Morse phonetic value(s)

Return type tuple

Examples

```
>>> bmpm('Christopher')
'xrQstopir xrQstYpir xristopir xristYpir xrQstofir xrQstYfir xristofir
xristYfir xristopi xritopir xritopi xristofi xritofir xritofi
tzristopir tzristofir zristopir zristopi zritopir zritopi zristofir
zristofi zritofir zritofi'
>>> bmpm('Niall')
'nial niol'
>>> bmpm('Smith')
'zmit'
>>> bmpm('Schmidt')
'zmit stzmit'
```

```
>>> bmpm('Christopher', language_arg='German')
'xrQstopir xrQstYpir xristopir xristYpir xrQstofir xrQstYfir xristofir
xristYfir'
>>> bmpm('Christopher', language_arg='English')
'tzristofir tzrQstofir tzristafir tzrQstafir xristofir xrQstofir
xristafir xrQstafir'
>>> bmpm('Christopher', language_arg='German', name_mode='ash')
'xrQstopir xrQstYpir xristopir xristYpir xrQstofir xrQstYfir xristofir
xristYfir'
```

```
>>> bmpm('Christopher', language_arg='German', match_mode='exact')
'xriStopher xriStofer xristopher xristofer'
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the BeiderMorse.encode method instead.

class abydos.phonetic.**NRL**

Bases: abydos.phonetic._phonetic._Phonetic

Naval Research Laboratory English-to-phoneme encoder.

This is defined by [EJMS76].

New in version 0.3.6.

encode(*word*)

Return the Naval Research Laboratory phonetic encoding of a word.

Parameters **word** (*str*) -- The word to transform

Returns The NRL phonetic encoding

Return type str

Examples

```
>>> pe = NRL()
>>> pe.encode('the')
'DHAX'
>>> pe.encode('round')
'rAWnd'
>>> pe.encode('quick')
'kwIHk'
>>> pe.encode('eaten')
'IYtEHn'
>>> pe.encode('Smith')
'smIHTH'
>>> pe.encode('Larsen')
'lAArsEHn'
```

New in version 0.3.0.

Changed in version 0.3.6: Encapsulated in class

abydos.phonetic.**nrl**(*word*)

Return the Naval Research Laboratory phonetic encoding of a word.

This is a wrapper for *NRL.encode()*.

Parameters **word** (*str*) -- The word to transform

Returns The NRL phonetic encoding

Return type str

Examples

```
>>> nrl('the')
'DHAX'
>>> nrl('round')
'rAWnd'
>>> nrl('quick')
'kwIHk'
>>> nrl('eaten')
'IYtEHn'
>>> nrl('Smith')
'smIHTH'
>>> nrl('Larsen')
'lAArsEHn'
```

New in version 0.3.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the NRL.encode method instead.

class abydos.phonetic.**MetaSoundex**(*lang='en'*)

Bases: abydos.phonetic._phonetic._Phonetic

MetaSoundex.

This is based on [KV17]. Only English ('en') and Spanish ('es') languages are supported, as in the original.

New in version 0.3.6.

Initialize MetaSoundex instance.

Parameters **lang** (*str*) -- Either en for English or es for Spanish

New in version 0.4.0.

encode(*word*)

Return the MetaSoundex code for a word.

Parameters **word** (*str*) -- The word to transform

Returns The MetaSoundex code

Return type str

Examples

```
>>> pe = MetaSoundex()
>>> pe.encode('Smith')
'4500'
>>> pe.encode('Waters')
'7362'
>>> pe.encode('James')
'1520'
>>> pe.encode('Schmidt')
'4530'
>>> pe.encode('Ashcroft')
'0261'
```

```
>>> pe = MetaSoundex(lang='es')
>>> pe.encode('Perez')
'094'
```

(continues on next page)

(continued from previous page)

```
>>> pe.encode('Martinez')
'69364'
>>> pe.encode('Gutierrez')
'83994'
>>> pe.encode('Santiago')
'4638'
>>> pe.encode('Nicolás')
'6754'
```

New in version 0.3.0.

Changed in version 0.3.6: Encapsulated in class

encode_alpha(*word*)

Return the MetaSoundex code for a word.

Parameters **word** (*str*) -- The word to transform

Returns The MetaSoundex code

Return type str

Examples

```
>>> pe = MetaSoundex()
>>> pe.encode_alpha('Smith')
'SN'
>>> pe.encode_alpha('Waters')
'WTRK'
>>> pe.encode_alpha('James')
'JNK'
>>> pe.encode_alpha('Schmidt')
'SNT'
>>> pe.encode_alpha('Ashcroft')
'AKRP'
```

```
>>> pe = MetaSoundex(lang='es')
>>> pe.encode_alpha('Perez')
'PRS'
>>> pe.encode_alpha('Martinez')
'NRTNS'
>>> pe.encode_alpha('Gutierrez')
'GTRRS'
>>> pe.encode_alpha('Santiago')
'SNTG'
>>> pe.encode_alpha('Nicolás')
'NKLS'
```

New in version 0.4.0.

abydos.phonetic.**metasoundex**(*word*, *lang='en'*)

Return the MetaSoundex code for a word.

This is a wrapper for *MetaSoundex.encode()*.

Parameters

- **word** (*str*) -- The word to transform

- **lang** (*str*) -- Either `en` for English or `es` for Spanish

Returns The MetaSoundex code

Return type str

Examples

```
>>> metasoundex('Smith')
'4500'
>>> metasoundex('Waters')
'7362'
>>> metasoundex('James')
'1520'
>>> metasoundex('Schmidt')
'4530'
>>> metasoundex('Ashcroft')
'0261'
>>> metasoundex('Perez', lang='es')
'094'
>>> metasoundex('Martinez', lang='es')
'69364'
>>> metasoundex('Gutierrez', lang='es')
'83994'
>>> metasoundex('Santiago', lang='es')
'4638'
>>> metasoundex('Nicolás', lang='es')
'6754'
```

New in version 0.3.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the MetaSoundex.encode method instead.

class abydos.phonetic.**ONCA**(*max_length=4*, *zero_pad=True*)

Bases: abydos.phonetic._phonetic._Phonetic

Oxford Name Compression Algorithm (ONCA).

This is the Oxford Name Compression Algorithm, based on [Gil97].

I can find no complete description of the "anglicised version of the NYSIIS method" identified as the first step in this algorithm, so this is likely not a precisely correct implementation, in that it employs the standard NYSIIS algorithm.

New in version 0.3.6.

Initialize ONCA instance.

Parameters

- **max_length** (*int*) -- The maximum length (default 5) of the code to return
- **zero_pad** (*bool*) -- Pad the end of the return value with 0s to achieve a max_length string

New in version 0.4.0.

encode(*word*)

Return the Oxford Name Compression Algorithm (ONCA) code for a word.

Parameters **word** (*str*) -- The word to transform

Returns The ONCA code

Return type str

Examples

```
>>> pe = ONCA()
>>> pe.encode('Christopher')
'C623'
>>> pe.encode('Niall')
'N400'
>>> pe.encode('Smith')
'S530'
>>> pe.encode('Schmidt')
'S530'
```

New in version 0.3.0.

Changed in version 0.3.6: Encapsulated in class

encode_alpha(*word*)

Return the alphabetic ONCA code for a word.

Parameters **word** (*str*) -- The word to transform

Returns The alphabetic ONCA code

Return type str

Examples

```
>>> pe = ONCA()
>>> pe.encode_alpha('Christopher')
'CRKT'
>>> pe.encode_alpha('Niall')
'NL'
>>> pe.encode_alpha('Smith')
'SNT'
>>> pe.encode_alpha('Schmidt')
'SNT'
```

New in version 0.4.0.

abydos.phonetic.**onca**(*word*, *max_length=4*, *zero_pad=True*)

Return the Oxford Name Compression Algorithm (ONCA) code for a word.

This is a wrapper for *ONCA.encode()*.

Parameters

- **word** (*str*) -- The word to transform
- **max_length** (*int*) -- The maximum length (default 5) of the code to return
- **zero_pad** (*bool*) -- Pad the end of the return value with 0s to achieve a max_length string

Returns The ONCA code

Return type str

Examples

```
>>> onca('Christopher')
'C623'
>>> onca('Niall')
'N400'
>>> onca('Smith')
'S530'
>>> onca('Schmidt')
'S530'
```

New in version 0.3.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the ONCA.encode method instead.

class abydos.phonetic.**FONEM**

Bases: abydos.phonetic._phonetic._Phonetic

FONEM.

FONEM is a phonetic algorithm designed for French (particularly surnames in Saguenay, Canada), defined in [BBL81].

Guillaume Plique's Javascript implementation [Pli18] at https://github.com/Yomguithereal/talisman/blob/master/src/phonetics/french/fonem.js was also consulted for this implementation.

New in version 0.3.6.

encode(*word*)

Return the FONEM code of a word.

Parameters **word** (*str*) -- The word to transform

Returns The FONEM code

Return type str

Examples

```
>>> pe = FONEM()
>>> pe.encode('Marchand')
'MARCHEN'
>>> pe.encode('Beaulieu')
'BOLIEU'
>>> pe.encode('Beaumont')
'BOMON'
>>> pe.encode('Legrand')
'LEGREN'
>>> pe.encode('Pelletier')
'PELETIER'
```

New in version 0.3.0.

Changed in version 0.3.6: Encapsulated in class

abydos.phonetic.**fonem**(*word*)

Return the FONEM code of a word.

This is a wrapper for *FONEM.encode()*.

Parameters **word** (*str*) -- The word to transform

Returns The FONEM code

Return type str

Examples

```
>>> fonem('Marchand')
'MARCHEN'
>>> fonem('Beaulieu')
'BOLIEU'
>>> fonem('Beaumont')
'BOMON'
>>> fonem('Legrand')
'LEGREN'
>>> fonem('Pelletier')
'PELETIER'
```

New in version 0.3.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the FONEM.encode method instead.

class abydos.phonetic.**HenryEarly**(*max_length=3*)
Bases: abydos.phonetic._phonetic._Phonetic

Henry code, early version.

The early version of Henry coding is given in [LegareLC72]. This is different from the later version defined in [Hen76].

New in version 0.3.6.

Initialize HenryEarly instance.

Parameters **max_length** (*int*) -- The length of the code returned (defaults to 3)

New in version 0.4.0.

encode(*word*)
Calculate the early version of the Henry code for a word.

Parameters **word** (*str*) -- The word to transform

Returns The early Henry code

Return type str

Examples

```
>>> pe = HenryEarly()
>>> pe.encode('Marchand')
'MRC'
>>> pe.encode('Beaulieu')
'BL'
>>> pe.encode('Beaumont')
'BM'
>>> pe.encode('Legrand')
'LGR'
>>> pe.encode('Pelletier')
'PLT'
```

New in version 0.3.0.

Changed in version 0.3.6: Encapsulated in class

abydos.phonetic.**henry_early**(*word*, *max_length=3*)

Calculate the early version of the Henry code for a word.

This is a wrapper for *HenryEarly.encode()*.

Parameters

- **word** (*str*) -- The word to transform
- **max_length** (*int*) -- The length of the code returned (defaults to 3)

Returns The early Henry code

Return type str

Examples

```
>>> henry_early('Marchand')
'MRC'
>>> henry_early('Beaulieu')
'BL'
>>> henry_early('Beaumont')
'BM'
>>> henry_early('Legrand')
'LGR'
>>> henry_early('Pelletier')
'PLT'
```

New in version 0.3.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the HenryEarly.encode method instead.

class abydos.phonetic.**Koelner**

Bases: abydos.phonetic._phonetic._Phonetic

Kölner Phonetik.

Based on the algorithm defined by [Pos69].

New in version 0.3.6.

encode(*word*)

Return the Kölner Phonetik (numeric output) code for a word.

While the output code is numeric, it is still a str because 0s can lead the code.

Parameters **word** (*str*) -- The word to transform

Returns The Kölner Phonetik value as a numeric string

Return type str

Example

```
>>> pe = Koelner()
>>> pe.encode('Christopher')
'478237'
>>> pe.encode('Niall')
'65'
>>> pe.encode('Smith')
'862'
>>> pe.encode('Schmidt')
'862'
>>> pe.encode('Müller')
'657'
>>> pe.encode('Zimmermann')
'86766'
```

New in version 0.1.0.

Changed in version 0.3.6: Encapsulated in class

encode_alpha(*word*)

Return the Kölner Phonetik (alphabetic output) code for a word.

Parameters **word** (*str*) -- The word to transform

Returns The Kölner Phonetik value as an alphabetic string

Return type str

Examples

```
>>> pe = Koelner()
>>> pe.encode_alpha('Smith')
'SNT'
>>> pe.encode_alpha('Schmidt')
'SNT'
>>> pe.encode_alpha('Müller')
'NLR'
>>> pe.encode_alpha('Zimmermann')
'SNRNN'
```

New in version 0.1.0.

Changed in version 0.3.6: Encapsulated in class

abydos.phonetic.**koelner_phonetik**(*word*)

Return the Kölner Phonetik (numeric output) code for a word.

This is a wrapper for *Koelner.encode()*.

Parameters **word** (*str*) -- The word to transform

Returns The Kölner Phonetik value as a numeric string

Return type str

Example

```
>>> koelner_phonetik('Christopher')
'478237'
>>> koelner_phonetik('Niall')
'65'
>>> koelner_phonetik('Smith')
'862'
>>> koelner_phonetik('Schmidt')
'862'
>>> koelner_phonetik('Müller')
'657'
>>> koelner_phonetik('Zimmermann')
'86766'
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the Koelner.encode method instead.

abydos.phonetic.**koelner_phonetik_num_to_alpha**(*num*)
Convert a Kölner Phonetik code from numeric to alphabetic.

This is a wrapper for `Koelner._to_alpha()`.

Parameters **num** (*str or int*) -- A numeric Kölner Phonetik representation

Returns An alphabetic representation of the same word

Return type str

Examples

```
>>> koelner_phonetik_num_to_alpha('862')
'SNT'
>>> koelner_phonetik_num_to_alpha('657')
'NLR'
>>> koelner_phonetik_num_to_alpha('86766')
'SNRNN'
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the Koelner._to_alpha method instead.

abydos.phonetic.**koelner_phonetik_alpha**(*word*)
Return the Kölner Phonetik (alphabetic output) code for a word.

This is a wrapper for *Koelner.encode_alpha()*.

Parameters **word** (*str*) -- The word to transform

Returns The Kölner Phonetik value as an alphabetic string

Return type str

Examples

```
>>> koelner_phonetik_alpha('Smith')
'SNT'
>>> koelner_phonetik_alpha('Schmidt')
'SNT'
>>> koelner_phonetik_alpha('Müller')
'NLR'
>>> koelner_phonetik_alpha('Zimmermann')
'SNRNN'
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the Koelner.encode_alpha method instead.

class abydos.phonetic.**Haase**(*primary_only=False*)

Bases: abydos.phonetic._phonetic._Phonetic

Haase Phonetik.

Based on the algorithm described at [Pra15].

Based on the original [HH00].

New in version 0.3.6.

Initialize Haase instance.

Parameters **primary_only** (*bool*) -- If True, only the primary code is returned

New in version 0.4.0.

encode(*word*)

Return the Haase Phonetik (numeric output) code for a word.

While the output code is numeric, it is nevertheless a str.

Parameters **word** (*str*) -- The word to transform

Returns The Haase Phonetik value as a numeric string

Return type tuple

Examples

```
>>> pe = Haase()
>>> pe.encode('Joachim')
('9496',)
>>> pe.encode('Christoph')
('4798293', '8798293')
>>> pe.encode('Jörg')
('974',)
>>> pe.encode('Smith')
('8692',)
>>> pe.encode('Schmidt')
('8692', '4692')
```

New in version 0.3.0.

Changed in version 0.3.6: Encapsulated in class

encode_alpha(*word*)

Return the alphabetic Haase Phonetik code for a word.

Parameters **word** (*str*) -- The word to transform

Returns The alphabetic Haase Phonetik value

Return type tuple

Examples

```
>>> pe = Haase()
>>> pe.encode_alpha('Joachim')
('AKAN',)
>>> pe.encode_alpha('Christoph')
('KRASTAF', 'SRASTAF')
>>> pe.encode_alpha('Jörg')
('ARK',)
>>> pe.encode_alpha('Smith')
('SNAT',)
>>> pe.encode_alpha('Schmidt')
('SNAT', 'KNAT')
```

New in version 0.4.0.

abydos.phonetic.**haase_phonetik**(*word*, *primary_only=False*)

Return the Haase Phonetik code for a word.

This is a wrapper for *Haase.encode()*.

Parameters

- **word** (*str*) -- The word to transform
- **primary_only** (*bool*) -- If True, only the primary code is returned

Returns The Haase Phonetik value as a numeric string

Return type tuple

Examples

```
>>> haase_phonetik('Joachim')
('9496',)
>>> haase_phonetik('Christoph')
('4798293', '8798293')
>>> haase_phonetik('Jörg')
('974',)
>>> haase_phonetik('Smith')
('8692',)
>>> haase_phonetik('Schmidt')
('8692', '4692')
```

New in version 0.3.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the Haase.encode method instead.

class abydos.phonetic.**RethSchek**

Bases: abydos.phonetic._phonetic._Phonetic

Reth-Schek Phonetik.

This algorithm is proposed in [vonRethS77].

Since I couldn't secure a copy of that document (maybe I'll look for it next time I'm in Germany), this implementation is based on what I could glean from the implementations published by German Record Linkage Center (www.record-linkage.de):

- Privacy-preserving Record Linkage (PPRL) (in R) [Ruk18]
- Merge ToolBox (in Java) [SBB04]

Rules that are unclear:

- Should 'C' become 'G' or 'Z'? (PPRL has both, 'Z' rule blocked)
- Should 'CC' become 'G'? (PPRL has blocked 'CK' that may be typo)
- Should 'TUI' -> 'ZUI' rule exist? (PPRL has rule, but I can't think of a German word with '-tui-' in it.)
- Should we really change 'SCH' -> 'CH' and then 'CH' -> 'SCH'?

New in version 0.3.6.

encode(*word*)

Return Reth-Schek Phonetik code for a word.

Parameters **word** (*str*) -- The word to transform

Returns The Reth-Schek Phonetik code

Return type str

Examples

```
>>> pe = RethSchek()
>>> pe.encode('Joachim')
'JOAGHIM'
>>> pe.encode('Christoph')
'GHRISDOF'
>>> pe.encode('Jörg')
'JOERG'
>>> pe.encode('Smith')
'SMID'
>>> pe.encode('Schmidt')
'SCHMID'
```

New in version 0.3.0.

Changed in version 0.3.6: Encapsulated in class

abydos.phonetic.**reth_schek_phonetik**(*word*)

Return Reth-Schek Phonetik code for a word.

This is a wrapper for *RethSchek.encode()*.

Parameters **word** (*str*) -- The word to transform

Returns The Reth-Schek Phonetik code

Return type str

Examples

```
>>> reth_schek_phonetik('Joachim')
'JOAGHIM'
>>> reth_schek_phonetik('Christoph')
'GHRISDOF'
>>> reth_schek_phonetik('Jörg')
'JOERG'
>>> reth_schek_phonetik('Smith')
'SMID'
>>> reth_schek_phonetik('Schmidt')
'SCHMID'
```

New in version 0.3.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the RethSchek.encode method instead.

class abydos.phonetic.**Phonem**

Bases: abydos.phonetic._phonetic._Phonetic

Phonem.

Phonem is defined in [GM88].

This version is based on the Perl implementation documented at [Wil05]. It includes some enhancements presented in the Java port at [dcm4che].

Phonem is intended chiefly for German names/words.

New in version 0.3.6.

encode(*word*)

Return the Phonem code for a word.

Parameters

- **word**(*str*) --
- **word to transform**(*The*) --

Returns The Phonem value

Return type str

Examples

```
>>> pe = Phonem()
>>> pe.encode('Christopher')
'CRYSDOVR'
>>> pe.encode('Niall')
'NYAL'
>>> pe.encode('Smith')
'SMYD'
>>> pe.encode('Schmidt')
'CMYD'
```

New in version 0.1.0.

Changed in version 0.3.6: Encapsulated in class

abydos.phonetic.**phonem**(*word*)

Return the Phonem code for a word.

This is a wrapper for *Phonem.encode()*.

Parameters **word** (*str*) -- The word to transform

Returns The Phonem value

Return type str

Examples

```
>>> phonem('Christopher')
'CRYSDOVR'
>>> phonem('Niall')
'NYAL'
>>> phonem('Smith')
'SMYD'
>>> phonem('Schmidt')
'CMYD'
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the Phonem.encode method instead.

class abydos.phonetic.**Phonet**(*mode=1*, *lang='de'*)

Bases: abydos.phonetic._phonetic._Phonetic

Phonet code.

phonet ("Hannoveraner Phonetik") was developed by Jörg Michael and documented in [Mic99].

This is a port of Jesper Zedlitz's code, which is licensed LGPL [Zed15].

That is, in turn, based on Michael's C code, which is also licensed LGPL [Mic07].

New in version 0.3.6.

Initialize AlphaSIS instance.

Parameters

- **mode** (*int*) -- The ponet variant to employ (1 or 2)
- **lang** (*str*) -- de (default) for German, none for no language

New in version 0.4.0.

encode(*word*)

Return the phonet code for a word.

Parameters **word** (*str*) -- The word to transform

Returns The phonet value

Return type str

Examples

```
>>> pe = Phonet()
>>> pe.encode('Christopher')
'KRISTOFA'
>>> pe.encode('Niall')
'NIAL'
>>> pe.encode('Smith')
'SMIT'
>>> pe.encode('Schmidt')
'SHMIT'
```

```
>>> pe2 = Phonet(mode=2)
>>> pe2.encode('Christopher')
'KRIZTUFA'
>>> pe2.encode('Niall')
'NIAL'
>>> pe2.encode('Smith')
'ZNIT'
>>> pe2.encode('Schmidt')
'ZNIT'
```

```
>>> pe_none = Phonet(lang='none')
>>> pe_none.encode('Christopher')
'CHRISTOPHER'
>>> pe_none.encode('Niall')
'NIAL'
>>> pe_none.encode('Smith')
'SMITH'
>>> pe_none.encode('Schmidt')
'SCHMIDT'
```

New in version 0.1.0.

Changed in version 0.3.6: Encapsulated in class

abydos.phonetic.**phonet**(*word*, *mode=1*, *lang='de'*)

Return the phonet code for a word.

This is a wrapper for *Phonet.encode()*.

Parameters

- **word** (*str*) -- The word to transform
- **mode** (*int*) -- The ponet variant to employ (1 or 2)
- **lang** (*str*) -- de (default) for German, none for no language

Returns The phonet value

Return type str

Examples

```
>>> phonet('Christopher')
'KRISTOFA'
>>> phonet('Niall')
'NIAL'
>>> phonet('Smith')
'SMIT'
>>> phonet('Schmidt')
'SHMIT'
```

```
>>> phonet('Christopher', mode=2)
'KRIZTUFA'
>>> phonet('Niall', mode=2)
'NIAL'
>>> phonet('Smith', mode=2)
'ZNIT'
>>> phonet('Schmidt', mode=2)
'ZNIT'
```

```
>>> phonet('Christopher', lang='none')
'CHRISTOPHER'
>>> phonet('Niall', lang='none')
'NIAL'
>>> phonet('Smith', lang='none')
'SMITH'
>>> phonet('Schmidt', lang='none')
'SCHMIDT'
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the Phonet.encode method instead.

class abydos.phonetic.**SoundexBR**(*max_length=4*, *zero_pad=True*)

Bases: abydos.phonetic._phonetic._Phonetic

SoundexBR.

This is based on [Mar15].

New in version 0.3.6.

Initialize SoundexBR instance.

Parameters

- **max_length** (*int*) -- The length of the code returned (defaults to 4)
- **zero_pad** (*bool*) -- Pad the end of the return value with 0s to achieve a max_length string

New in version 0.4.0.

encode(*word*)

Return the SoundexBR encoding of a word.

Parameters **word** (*str*) -- The word to transform

Returns The SoundexBR code

Return type str

Examples

```
>>> pe = SoundexBR()
>>> pe.encode('Oliveira')
'O416'
>>> pe.encode('Almeida')
'A453'
>>> pe.encode('Barbosa')
'B612'
>>> pe.encode('Araújo')
'A620'
>>> pe.encode('Gonçalves')
'G524'
>>> pe.encode('Goncalves')
'G524'
```

New in version 0.3.0.

Changed in version 0.3.6: Encapsulated in class

encode_alpha(*word*)

Return the alphabetic SoundexBR encoding of a word.

Parameters **word** (*str*) -- The word to transform

Returns The alphabetic SoundexBR code

Return type str

Examples

```
>>> pe = SoundexBR()
>>> pe.encode_alpha('Oliveira')
'OLPR'
>>> pe.encode_alpha('Almeida')
'ALNT'
>>> pe.encode_alpha('Barbosa')
'BRPK'
>>> pe.encode_alpha('Araújo')
'ARK'
>>> pe.encode_alpha('Gonçalves')
'GNKL'
>>> pe.encode_alpha('Goncalves')
'GNKL'
```

New in version 0.4.0.

abydos.phonetic.**soundex_br**(*word*, *max_length=4*, *zero_pad=True*)

Return the SoundexBR encoding of a word.

This is a wrapper for *SoundexBR.encode()*.

Parameters

- **word** (*str*) -- The word to transform
- **max_length** (*int*) -- The length of the code returned (defaults to 4)
- **zero_pad** (*bool*) -- Pad the end of the return value with 0s to achieve a max_length string

Returns The SoundexBR code

Return type str

Examples

```
>>> soundex_br('Oliveira')
'O416'
>>> soundex_br('Almeida')
'A453'
>>> soundex_br('Barbosa')
'B612'
>>> soundex_br('Araújo')
'A620'
>>> soundex_br('Gonçalves')
'G524'
>>> soundex_br('Goncalves')
'G524'
```

New in version 0.3.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the SoundexBR.encode method instead.

class abydos.phonetic.**PhoneticSpanish**(*max_length=-1*)
Bases: abydos.phonetic._phonetic._Phonetic

PhoneticSpanish.

This follows the coding described in [AmonME12] and [delPAngelesEGGM15].

New in version 0.3.6.

Initialize PhoneticSpanish instance.

Parameters **max_length** (*int*) -- The length of the code returned (defaults to unlimited)

New in version 0.4.0.

encode(*word*)
Return the PhoneticSpanish coding of word.

Parameters **word** (*str*) -- The word to transform

Returns The PhoneticSpanish code

Return type str

Examples

```
>>> pe = PhoneticSpanish()
>>> pe.encode('Perez')
'094'
>>> pe.encode('Martinez')
'69364'
>>> pe.encode('Gutierrez')
'83994'
>>> pe.encode('Santiago')
'4638'
>>> pe.encode('Nicolás')
'6454'
```

New in version 0.3.0.

Changed in version 0.3.6: Encapsulated in class

encode_alpha(*word*)

Return the alphabetic PhoneticSpanish coding of word.

Parameters **word** (*str*) -- The word to transform

Returns The alphabetic PhoneticSpanish code

Return type str

Examples

```
>>> pe = PhoneticSpanish()
>>> pe.encode_alpha('Perez')
'PRS'
>>> pe.encode_alpha('Martinez')
'NRTNS'
>>> pe.encode_alpha('Gutierrez')
'GTRRS'
>>> pe.encode_alpha('Santiago')
'SNTG'
>>> pe.encode_alpha('Nicolás')
'NSLS'
```

New in version 0.4.0.

abydos.phonetic.**phonetic_spanish**(*word*, *max_length=-1*)

Return the PhoneticSpanish coding of word.

This is a wrapper for *PhoneticSpanish.encode()*.

Parameters

- **word** (*str*) -- The word to transform
- **max_length** (*int*) -- The length of the code returned (defaults to unlimited)

Returns The PhoneticSpanish code

Return type str

Examples

```
>>> phonetic_spanish('Perez')
'094'
>>> phonetic_spanish('Martinez')
'69364'
>>> phonetic_spanish('Gutierrez')
'83994'
>>> phonetic_spanish('Santiago')
'4638'
>>> phonetic_spanish('Nicolás')
'6454'
```

New in version 0.3.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the PhoneticSpanish.encode method instead.

class abydos.phonetic.**SpanishMetaphone**(*max_length=6*, *modified=False*)
Bases: abydos.phonetic._phonetic._Phonetic

Spanish Metaphone.

This is a quick rewrite of the Spanish Metaphone Algorithm, as presented at https://github.com/amsqr/Spanish-Metaphone and discussed in [MLM12].

Modified version based on [delPAngelesBailonM16].

New in version 0.3.6.

Initialize AlphaSIS instance.

Parameters

- **max_length** (*int*) -- The length of the code returned (defaults to 6)
- **modified** (*bool*) -- Set to True to use del Pilar Angeles & Bailón-Miguel's modified version of the algorithm

New in version 0.4.0.

encode(*word*)
Return the Spanish Metaphone of a word.

Parameters **word** (*str*) -- The word to transform

Returns The Spanish Metaphone code

Return type str

Examples

```
>>> pe = SpanishMetaphone()
>>> pe.encode('Perez')
'PRZ'
>>> pe.encode('Martinez')
'MRTNZ'
>>> pe.encode('Gutierrez')
'GTRRZ'
>>> pe.encode('Santiago')
'SNTG'
>>> pe.encode('Nicolás')
'NKLS'
```

New in version 0.3.0.

Changed in version 0.3.6: Encapsulated in class

abydos.phonetic.**spanish_metaphone**(*word*, *max_length=6*, *modified=False*)
Return the Spanish Metaphone of a word.

This is a wrapper for *SpanishMetaphone.encode()*.

Parameters

- **word** (*str*) -- The word to transform
- **max_length** (*int*) -- The length of the code returned (defaults to 6)
- **modified** (*bool*) -- Set to True to use del Pilar Angeles & Bailón-Miguel's modified version of the algorithm

Returns The Spanish Metaphone code

Return type str

Examples

```
>>> spanish_metaphone('Perez')
'PRZ'
>>> spanish_metaphone('Martinez')
'MRTNZ'
>>> spanish_metaphone('Gutierrez')
'GTRRZ'
>>> spanish_metaphone('Santiago')
'SNTG'
>>> spanish_metaphone('Nicolás')
'NKLS'
```

New in version 0.3.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the SpanishMetaphone.encode method instead.

class abydos.phonetic.**SfinxBis**(*max_length=-1*)
Bases: abydos.phonetic._phonetic._Phonetic

SfinxBis code.

SfinxBis is a Soundex-like algorithm defined in [Axe09].

This implementation follows the reference implementation: [Sjoo09].

SfinxBis is intended chiefly for Swedish names.

New in version 0.3.6.

Initialize SfinxBis instance.

Parameters **max_length** (*int*) -- The length of the code returned (defaults to unlimited)

New in version 0.4.0.

encode(*word*)
Return the SfinxBis code for a word.

Parameters **word** (*str*) -- The word to transform

Returns The SfinxBis value

Return type tuple

Examples

```
>>> pe = SfinxBis()
>>> pe.encode('Christopher')
('K68376',)
>>> pe.encode('Niall')
('N4',)
>>> pe.encode('Smith')
('S53',)
>>> pe.encode('Schmidt')
('S53',)
```

```
>>> pe.encode('Johansson')
('J585',)
>>> pe.encode('Sjöberg')
('#162',)
```

New in version 0.1.0.

Changed in version 0.3.6: Encapsulated in class

encode_alpha(*word*)

Return the alphabetic SfinxBis code for a word.

Parameters **word** (*str*) -- The word to transform

Returns The alphabetic SfinxBis value

Return type tuple

Examples

```
>>> pe = SfinxBis()
>>> pe.encode_alpha('Christopher')
('KRSTFR',)
>>> pe.encode_alpha('Niall')
('NL',)
>>> pe.encode_alpha('Smith')
('SNT',)
>>> pe.encode_alpha('Schmidt')
('SNT',)
```

```
>>> pe.encode_alpha('Johansson')
('JNSN',)
>>> pe.encode_alpha('Sjöberg')
('ŠPRK',)
```

New in version 0.4.0.

abydos.phonetic.**sfinxbis**(*word*, *max_length=-1*)

Return the SfinxBis code for a word.

This is a wrapper for *SfinxBis.encode()*.

Parameters

- **word** (*str*) -- The word to transform
- **max_length** (*int*) -- The length of the code returned (defaults to unlimited)

Returns The SfinxBis value

Return type tuple

Examples

```
>>> sfinxbis('Christopher')
('K68376',)
>>> sfinxbis('Niall')
('N4',)
>>> sfinxbis('Smith')
('S53',)
>>> sfinxbis('Schmidt')
('S53',)
```

```
>>> sfinxbis('Johansson')
('J585',)
>>> sfinxbis('Sjöberg')
('#162',)
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the SfinxBis.encode method instead.

class abydos.phonetic.**Waahlin**(*encoder=None*)

Bases: abydos.phonetic._phonetic._Phonetic

Wåhlin code.

Wåhlin's first-letter coding is based on the description in [Eri97].

New in version 0.3.6.

Initialize Waahlin instance.

Parameters **encoder** (*_Phonetic*) -- An initialized phonetic algorithm object

New in version 0.4.0.

encode(*word*, *alphabetic=False*)

Return the Wåhlin code for a word.

Parameters

- **word** (*str*) -- The word to transform
- **alphabetic** (*bool*) -- If True, the encoder will apply its alphabetic form (.encode_alpha rather than .encode)

Returns The Wåhlin code value

Return type str

Examples

```
>>> pe = Waahlin()
>>> pe.encode('Christopher')
'KRISTOFER'
>>> pe.encode('Niall')
'NJALL'
>>> pe.encode('Smith')
'SMITH'
>>> pe.encode('Schmidt')
'*MIDT'
```

New in version 0.4.0.

encode_alpha(*word*)

Return the alphabetic Wåhlin code for a word.

Parameters **word** (*str*) -- The word to transform

Returns The alphabetic Wåhlin code value

Return type str

Examples

```
>>> pe = Waahlin()
>>> pe.encode_alpha('Christopher')
'KRISTOFER'
>>> pe.encode_alpha('Niall')
'NJALL'
>>> pe.encode_alpha('Smith')
'SMITH'
>>> pe.encode_alpha('Schmidt')
'ŠMIDT'
```

New in version 0.4.0.

class abydos.phonetic.**Norphone**

Bases: abydos.phonetic._phonetic._Phonetic

Norphone.

The reference implementation by Lars Marius Garshol is available in [Gar15].

Norphone was designed for Norwegian, but this implementation has been extended to support Swedish vowels as well. This function incorporates the "not implemented" rules from the above file's rule set.

New in version 0.3.6.

encode(*word*)

Return the Norphone code.

Parameters **word** (*str*) -- The word to transform

Returns The Norphone code

Return type str

Examples

```
>>> pe = Norphone()
>>> pe.encode('Hansen')
'HNSN'
>>> pe.encode('Larsen')
'LRSN'
>>> pe.encode('Aagaard')
'ÅKRT'
>>> pe.encode('Braaten')
'BRTN'
>>> pe.encode('Sandvik')
'SNVK'
```

New in version 0.3.0.

Changed in version 0.3.6: Encapsulated in class

abydos.phonetic.**norphone**(*word*)

Return the Norphone code.

This is a wrapper for *Norphone.encode()*.

Parameters **word** (*str*) -- The word to transform

Returns The Norphone code

Return type str

Examples

```
>>> norphone('Hansen')
'HNSN'
>>> norphone('Larsen')
'LRSN'
>>> norphone('Aagaard')
'ÅKRT'
>>> norphone('Braaten')
'BRTN'
>>> norphone('Sandvik')
'SNVK'
```

New in version 0.3.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the Norphone.encode method instead.

class abydos.phonetic.**Ainsworth**

Bases: abydos.phonetic._phonetic._Phonetic

Ainsworth's grapheme to phoneme converter.

Based on the ruleset listed in [Ain73].

New in version 0.4.1.

encode(*word*)

Return the phonemic representation of a word.

Parameters **word** (*str*) -- The word to transform

Returns The phonemic representation in IPA

Return type str

Examples

```
>>> pe = Ainsworth()
>>> pe.encode('Christopher')
'trstof'
>>> pe.encode('Niall')
'nl'
>>> pe.encode('Smith')
'smð'
>>> pe.encode('Schmidt')
'skmdt'
```

New in version 0.4.1.

3.1.1.7 abydos.stats package

abydos.stats.

The stats module defines functions for calculating various statistical data about linguistic objects.

Functions are provided for calculating the following means:

- arithmetic mean (*amean()*)
- geometric mean (*gmean()*)
- harmonic mean (*hmean()*)
- quadratic mean (*qmean()*)
- contraharmonic mean (*cmean()*)
- logarithmic mean (*lmean()*)
- identric (exponential) mean (*imean()*)
- Seiffert's mean (*seiffert_mean()*)
- Lehmer mean (*lehmer_mean()*)
- Heronian mean (*heronian_mean()*)
- Hölder (power/generalized) mean (*hoelder_mean()*)
- arithmetic-geometric mean (*agmean()*)
- geometric-harmonic mean (*ghmean()*)
- arithmetic-geometric-harmonic mean (*aghmean()*)

And for calculating:

- midrange (*midrange()*)
- median (*median()*)
- mode (*mode()*)
- variance (*var()*)
- standard deviation (*std()*)

Some examples of the basic functions:

```
>>> nums = [16, 49, 55, 49, 6, 40, 23, 47, 29, 85, 76, 20]
>>> amean(nums)
41.25
>>> aghmean(nums)
32.42167170892585
>>> heronian_mean(nums)
37.931508950381925
>>> mode(nums)
49
>>> std(nums)
22.876935255113754
```

Two pairwise functions are provided:

- mean pairwise similarity (*mean_pairwise_similarity()*), which returns the mean similarity (using a supplied similarity function) among each item in a collection
- pairwise similarity statistics (*pairwise_similarity_statistics()*), which returns the max, min, mean, and standard deviation of pairwise similarities between two collections

The confusion table class (*ConfusionTable*) can be constructed in a number of ways:

- four values, representing true positives, true negatives, false positives, and false negatives, can be passed to the constructor
- a list or tuple with four values, representing true positives, true negatives, false positives, and false negatives, can be passed to the constructor
- a dict with keys 'tp', 'tn', 'fp', 'fn', each assigned to the values for true positives, true negatives, false positives, and false negatives can be passed to the constructor

The *ConfusionTable* class has methods:

- *to_tuple()* extracts the *ConfusionTable* values as a tuple: (w, x, y, z)
- *to_dict()* extracts the *ConfusionTable* values as a dict: {'tp':w, 'tn':x, 'fp':y, 'fn':z}
- *true_pos()* returns the number of true positives
- *true_neg()* returns the number of true negatives
- *false_pos()* returns the number of false positives
- *false_neg()* returns the number of false negatives
- *correct_pop()* returns the correct population
- *error_pop()* returns the error population
- *pred_pos_pop()* returns the test positive population
- *pred_neg_pop()* returns the test negative population
- *cond_pos_pop()* returns the condition positive population
- *cond_neg_pop()* returns the condition negative population
- *population()* returns the total population
- *precision()* returns the precision
- *precision_gain()* returns the precision gain
- *recall()* returns the recall
- *specificity()* returns the specificity
- *npv()* returns the negative predictive value
- *fallout()* returns the fallout
- *fdr()* returns the false discovery rate
- *accuracy()* returns the accuracy
- *accuracy_gain()* returns the accuracy gain
- *balanced_accuracy()* returns the balanced accuracy
- *informedness()* returns the informedness
- *markedness()* returns the markedness
- *pr_amean()* returns the arithmetic mean of precision & recall

- *pr_gmean()* returns the geometric mean of precision & recall
- *pr_hmean()* returns the harmonic mean of precision & recall
- *pr_qmean()* returns the quadratic mean of precision & recall
- *pr_cmean()* returns the contraharmonic mean of precision & recall
- *pr_lmean()* returns the logarithmic mean of precision & recall
- *pr_imean()* returns the identric mean of precision & recall
- *pr_seiffert_mean()* returns Seiffert's mean of precision & recall
- *pr_lehmer_mean()* returns the Lehmer mean of precision & recall
- *pr_heronian_mean()* returns the Heronian mean of precision & recall
- *pr_hoelder_mean()* returns the Hölder mean of precision & recall
- *pr_agmean()* returns the arithmetic-geometric mean of precision & recall
- *pr_ghmean()* returns the geometric-harmonic mean of precision & recall
- *pr_aghmean()* returns the arithmetic-geometric-harmonic mean of precision & recall
- *fbeta_score()* returns the F_{beta} score
- *f2_score()* returns the F_2 score
- *fhalf_score()* returns the $F_{\frac{1}{2}}$ score
- *e_score()* returns the E score
- *f1_score()* returns the F_1 score
- *f_measure()* returns the F measure
- *g_measure()* returns the G measure
- *mcc()* returns Matthews correlation coefficient
- *significance()* returns the significance
- *kappa_statistic()* returns the Kappa statistic

```
>>> ct = ConfusionTable(120, 60, 20, 30)
>>> ct.f1_score()
0.8275862068965518
>>> ct.mcc()
0.5367450401216932
>>> ct.specificity()
0.75
>>> ct.significance()
66.26190476190476
```

The *ConfusionTable* class also supports checking for equality with another *ConfusionTable* and casting to string with `str()`:

```
>>> (ConfusionTable({'tp':120, 'tn':60, 'fp':20, 'fn':30}) ==
... ConfusionTable(120, 60, 20, 30))
True
>>> str(ConfusionTable(120, 60, 20, 30))
'tp:120, tn:60, fp:20, fn:30'
```

class abydos.stats.**ConfusionTable**(*tp=0*, *tn=0*, *fp=0*, *fn=0*)

Bases: object

ConfusionTable object.

This object is initialized by passing either four integers (or a tuple of four integers) representing the squares of a confusion table: true positives, true negatives, false positives, and false negatives

The object possesses methods for the calculation of various statistics based on the confusion table.

Initialize ConfusionTable.

Parameters

- **tp** (*int or a tuple, list, or dict*) -- True positives; If a tuple or list is supplied, it must include 4 values in the order [tp, tn, fp, fn]. If a dict is supplied, it must have 4 keys, namely 'tp', 'tn', 'fp', & 'fn'.
- **tn** (*int*) -- True negatives
- **fp** (*int*) -- False positives
- **fn** (*int*) -- False negatives

Raises **AttributeError** -- ConfusionTable requires a 4-tuple when being created from a tuple.

Examples

```
>>> ct = ConfusionTable(120, 60, 20, 30)
>>> ct == ConfusionTable((120, 60, 20, 30))
True
>>> ct == ConfusionTable([120, 60, 20, 30])
True
>>> ct == ConfusionTable({'tp': 120, 'tn': 60, 'fp': 20, 'fn': 30})
True
```

New in version 0.1.0.

accuracy()

Return accuracy.

Accuracy is defined as

$$\frac{tp + tn}{population}$$

Cf. https://en.wikipedia.org/wiki/Accuracy

Returns The accuracy of the confusion table

Return type float

Example

```
>>> ct = ConfusionTable(120, 60, 20, 30)
>>> ct.accuracy()
0.782608695652174
```

New in version 0.1.0.

accuracy_gain()
Return gain in accuracy.

The gain in accuracy is defined as

$$G(accuracy) = \frac{accuracy}{random\ accuracy}$$

Cf. https://en.wikipedia.org/wiki/Gain_(information_retrieval)

Returns The gain in accuracy of the confusion table

Return type float

Example

```
>>> ct = ConfusionTable(120, 60, 20, 30)
>>> ct.accuracy_gain()
1.4325259515570934
```

New in version 0.1.0.

actual_entropy()
Return the actual entropy.

Implementation based on https://github.com/Magnetic/proficiency-metric

Returns The actual entropy of the confusion table

Return type float

Example

```
>>> ct = ConfusionTable(120, 60, 20, 30)
>>> ct.actual_entropy()
0.6460905050608101
```

New in version 0.4.0.

balanced_accuracy()
Return balanced accuracy.

Balanced accuracy is defined as

$$\frac{sensitivity + specificity}{2}$$

Cf. https://en.wikipedia.org/wiki/Accuracy

Returns The balanced accuracy of the confusion table

Return type float

Example

```
>>> ct = ConfusionTable(120, 60, 20, 30)
>>> ct.balanced_accuracy()
0.775
```

New in version 0.1.0.

cond_neg_pop()
Return condition negative population.

Returns The condition negative population of the confusion table

Return type int

Example

```
>>> ct = ConfusionTable(120, 60, 20, 30)
>>> ct.cond_neg_pop()
80
```

New in version 0.1.0.

cond_pos_pop()
Return condition positive population.

Returns The condition positive population of the confusion table

Return type int

Example

```
>>> ct = ConfusionTable(120, 60, 20, 30)
>>> ct.cond_pos_pop()
150
```

New in version 0.1.0.

correct_pop()
Return correct population.

Returns The correct population of the confusion table

Return type int

Example

```
>>> ct = ConfusionTable(120, 60, 20, 30)
>>> ct.correct_pop()
180
```

New in version 0.1.0.

d_measure()
Return D-measure.

D-measure is defined as

$$1 - \frac{1}{\frac{1}{precision} + \frac{1}{recall} - 1}$$

Returns The D-measure of the confusion table

Return type float

Examples

```
>>> ct = ConfusionTable(120, 60, 20, 30)
>>> ct.d_measure()
0.2941176470588237
```

New in version 0.4.0.

dependency()
Return dependency.

Implementation based on https://github.com/Magnetic/proficiency-metric

Returns The dependency of the confusion table

Return type float

Example

```
>>> ct = ConfusionTable(120, 60, 20, 30)
>>> ct.dependency()
0.12618094145262454
```

New in version 0.4.0.

diagnostic_odds_ratio()
Return diagnostic odds ratio.

Diagnostic odds ratio is defined as

$$\frac{tp \cdot tn}{fp \cdot fn}$$

Cf. https://en.wikipedia.org/wiki/Diagnostic_odds_ratio

Returns The negative likelihood ratio of the confusion table

Return type float

Example

```
>>> ct = ConfusionTable(120, 60, 20, 30)
>>> ct.diagnostic_odds_ratio()
12.0
```

New in version 0.4.0.

e_score(*beta=1.0*)

Return E-score.

This is Van Rijsbergen's effectiveness measure: $E = 1 - F_\beta$.

Cf. https://en.wikipedia.org/wiki/Information_retrieval#F-measure

Parameters **beta** (*float*) -- The β parameter in the above formula

Returns The E-score of the confusion table

Return type float

Example

```
>>> ct = ConfusionTable(120, 60, 20, 30)
>>> ct.e_score()
0.17241379310344818
```

New in version 0.1.0.

error_pop()

Return error population.

Returns The error population of the confusion table

Return type int

Example

```
>>> ct = ConfusionTable(120, 60, 20, 30)
>>> ct.error_pop()
50
```

New in version 0.1.0.

error_rate()

Return error rate.

Error rate is defined as

$$\frac{fp + fn}{population}$$

Returns The error rate of the confusion table

Return type float

Example

```
>>> ct = ConfusionTable(120, 60, 20, 30)
>>> ct.error_rate()
0.21739130434782608
```

New in version 0.4.0.

f1_score()
Return F_1 score.

F_1 score is the harmonic mean of precision and recall

$$2 \cdot \frac{precision \cdot recall}{precision + recall}$$

Cf. https://en.wikipedia.org/wiki/F1_score

Returns The F_1 of the confusion table

Return type float

Example

```
>>> ct = ConfusionTable(120, 60, 20, 30)
>>> ct.f1_score()
0.8275862068965518
```

New in version 0.1.0.

f2_score()
Return F_2.

The F_2 score emphasizes recall over precision in comparison to the F_1 score

Cf. https://en.wikipedia.org/wiki/F1_score

Returns The F_2 of the confusion table

Return type float

Example

```
>>> ct = ConfusionTable(120, 60, 20, 30)
>>> ct.f2_score()
0.8108108108108109
```

New in version 0.1.0.

f_measure()
Return F-measure.

F-measure is the harmonic mean of precision and recall

$$2 \cdot \frac{precision \cdot recall}{precision + recall}$$

Cf. https://en.wikipedia.org/wiki/F1_score

Returns The math:F-measure of the confusion table

Return type float

Example

```
>>> ct = ConfusionTable(120, 60, 20, 30)
>>> ct.f_measure()
0.8275862068965516
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the ConfusionTable.pr_hmean method instead.

fallout()
Return fall-out.

Fall-out is defined as

$$\frac{fp}{fp + tn}$$

AKA false positive rate (FPR)

Cf. https://en.wikipedia.org/wiki/Information_retrieval#Fall-out

Returns The fall-out of the confusion table

Return type float

Example

```
>>> ct = ConfusionTable(120, 60, 20, 30)
>>> ct.fallout()
0.25
```

New in version 0.1.0.

false_neg()
Return false negatives.

AKA Type II error

Returns The false negatives of the confusion table

Return type int

Example

```
>>> ct = ConfusionTable(120, 60, 20, 30)
>>> ct.false_neg()
30
```

New in version 0.1.0.

false_omission_rate()
Return false omission rate (FOR).

FOR is defined as

$$\frac{fn}{tn + fn}$$

Cf. https://en.wikipedia.org/wiki/False_omission_rate

Returns The false omission rate of the confusion table

Return type float

Example

```
>>> ct = ConfusionTable(120, 60, 20, 30)
>>> ct.false_omission_rate()
0.3333333333333333
```

New in version 0.4.0.

false_pos()
Return false positives.

AKA Type I error

Returns The false positives of the confusion table

Return type int

Example

```
>>> ct = ConfusionTable(120, 60, 20, 30)
>>> ct.false_pos()
20
```

New in version 0.1.0.

fbeta_score(*beta=1.0*)
Return F_β score.

F_β for a positive real value β "measures the effectiveness of retrieval with respect to a user who attaches β times as much importance to recall as precision" (van Rijsbergen 1979)

F_β score is defined as

$$(1 + \beta^2) \cdot \frac{precision \cdot recall}{((\beta^2 \cdot precision) + recall)}$$

Cf. https://en.wikipedia.org/wiki/F1_score

Parameters **beta** (*float*) -- The β parameter in the above formula

Returns The F_β of the confusion table

Return type float

Raises **AttributeError** -- Beta must be a positive real value

Examples

```
>>> ct = ConfusionTable(120, 60, 20, 30)
>>> ct.fbeta_score()
0.8275862068965518
>>> ct.fbeta_score(beta=0.1)
0.8565371024734982
```

New in version 0.1.0.

fdr()

Return false discovery rate (FDR).

False discovery rate is defined as

$$\frac{fp}{fp + tp}$$

Cf. https://en.wikipedia.org/wiki/False_discovery_rate

Returns The false discovery rate of the confusion table

Return type float

Example

```
>>> ct = ConfusionTable(120, 60, 20, 30)
>>> ct.fdr()
0.14285714285714285
```

New in version 0.1.0.

fhalf_score()

Return $F_{0.5}$ score.

The $F_{0.5}$ score emphasizes precision over recall in comparison to the F_1 score

Cf. https://en.wikipedia.org/wiki/F1_score

Returns The $F_{0.5}$ score of the confusion table

Return type float

Example

```
>>> ct = ConfusionTable(120, 60, 20, 30)
>>> ct.fhalf_score()
0.8450704225352114
```

New in version 0.1.0.

fnr()
Return false negative rate.

False negative rate is defined as

$$\frac{fn}{tp + fn}$$

AKA miss rate

Cf. https://en.wikipedia.org/wiki/False_negative_rate

Returns The false negative rate of the confusion table

Return type float

Example

```
>>> ct = ConfusionTable(120, 60, 20, 30)
>>> round(ct.fnr(), 8)
0.2
```

New in version 0.4.0.

g_measure()
Return G-measure.

G-measure is the geometric mean of precision and recall:

$$\sqrt{precision \cdot recall}$$

This is identical to the Fowlkes–Mallows (FM) index for two clusters.

Cf. https://en.wikipedia.org/wiki/F1_score#G-measure

Cf. https://en.wikipedia.org/wiki/Fowlkes%E2%80%93Mallows_index

Returns The G-measure of the confusion table

Return type float

Example

```
>>> ct = ConfusionTable(120, 60, 20, 30)
>>> ct.g_measure()
0.828078671210825
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the ConfusionTable.pr_gmean method instead.

igr()

Return information gain ratio.

Implementation based on https://github.com/Magnetic/proficiency-metric

Cf. https://en.wikipedia.org/wiki/Information_gain_ratio

Returns The information gain ratio of the confusion table

Return type float

Example

```
>>> ct = ConfusionTable(120, 60, 20, 30)
>>> ct.igr()
0.22019657299448012
```

New in version 0.4.0.

informedness()

Return informedness.

Informedness is defined as

$$sensitivity + specificity - 1$$

AKA Youden's J statistic ([You50])

AKA DeltaP'

Cf. https://en.wikipedia.org/wiki/Youden%27s_J_statistic

Returns The informedness of the confusion table

Return type float

Example

```
>>> ct = ConfusionTable(120, 60, 20, 30)
>>> ct.informedness()
0.55
```

New in version 0.1.0.

jaccard()
Return Jaccard index.

The Jaccard index of a confusion table is

$$\frac{tp}{tp + fp + fn}$$

Returns The Jaccard index of the confusion table

Return type float

Example

```
>>> ct = ConfusionTable(120, 60, 20, 30)
>>> ct.jaccard()
0.7058823529411765
```

New in version 0.4.0.

joint_entropy()
Return the joint entropy.

Implementation based on https://github.com/Magnetic/proficiency-metric

Returns The joint entropy of the confusion table

Return type float

Example

```
>>> ct = ConfusionTable(120, 60, 20, 30)
>>> ct.joint_entropy()
1.1680347446270396
```

New in version 0.4.0.

kappa_statistic()
Return statistic.

The statistic is defined as

$$\kappa = \frac{accuracy - random\ accuracy}{1 - random\ accuracy},$$

The statistic compares the performance of the classifier relative to the performance of a random classifier. κ = 0 indicates performance identical to random. κ = 1 indicates perfect predictive success. κ = -1 indicates perfect predictive failure.

Returns The statistic of the confusion table

Return type float

Example

```
>>> ct = ConfusionTable(120, 60, 20, 30)
>>> ct.kappa_statistic()
0.5344129554655871
```

New in version 0.1.0.

lift()
Return lift.

Implementation based on https://github.com/Magnetic/proficiency-metric

Returns The lift of the confusion table

Return type float

Example

```
>>> ct = ConfusionTable(120, 60, 20, 30)
>>> ct.lift()
1.3142857142857143
```

New in version 0.4.0.

markedness()
Return markedness.

Markedness is defined as

$$precision + npv - 1$$

AKA DeltaP

Returns The markedness of the confusion table

Return type float

Example

```
>>> ct = ConfusionTable(120, 60, 20, 30)
>>> ct.markedness()
0.5238095238095237
```

New in version 0.1.0.

mcc()
Return Matthews correlation coefficient (MCC).

The Matthews correlation coefficient is defined in [Mat75] as:

$$\frac{(tp \cdot tn) - (fp \cdot fn)}{\sqrt{(tp + fp)(tp + fn)(tn + fp)(tn + fn)}}$$

This is equivalent to the geometric mean of informedness and markedness, defined above.

Cf. https://en.wikipedia.org/wiki/Matthews_correlation_coefficient

Returns The Matthews correlation coefficient of the confusion table

Return type float

Example

```
>>> ct = ConfusionTable(120, 60, 20, 30)
>>> ct.mcc()
0.5367450401216932
```

New in version 0.1.0.

mutual_information()

Return the mutual information.

Implementation based on https://github.com/Magnetic/proficiency-metric

Returns

- *float* -- The mutual information of the confusion table
- **Cf. https** (*//en.wikipedia.org/wiki/Mutual_information*)

Example

```
>>> ct = ConfusionTable(120, 60, 20, 30)
>>> ct.mutual_information()
0.14738372372641576
```

New in version 0.4.0.

neg_likelihood_ratio()

Return negative likelihood ratio.

Negative likelihood ratio is defined as

$$\frac{1 - recall}{specificity}$$

Cf. https://en.wikipedia.org/wiki/Likelihood_ratios_in_diagnostic_testing

Returns The negative likelihood ratio of the confusion table

Return type float

Example

```
>>> ct = ConfusionTable(120, 60, 20, 30)
>>> ct.neg_likelihood_ratio()
0.2666666666666666
```

New in version 0.4.0.

npv()
Return negative predictive value (NPV).

NPV is defined as

$$\frac{tn}{tn + fn}$$

AKA inverse precision

Cf. https://en.wikipedia.org/wiki/Negative_predictive_value

Returns The negative predictive value of the confusion table

Return type float

Example

```
>>> ct = ConfusionTable(120, 60, 20, 30)
>>> ct.npv()
0.6666666666666666
```

New in version 0.1.0.

phi_coefficient()
Return coefficient.

The ϕ coefficient is defined as

$$\phi = \frac{tp \cdot tn - fp \cdot tn}{\sqrt{(tp + fp) \cdot (tp + fn) \cdot (tn + fp) \cdot (tn + fn)}}$$

Returns The coefficient of the confusion table

Return type float

Example

```
>>> ct = ConfusionTable(120, 60, 20, 30)
>>> ct.phi_coefficient()
0.5367450401216932
```

New in version 0.4.0.

population()
Return population, N.

Returns The population (N) of the confusion table

Return type int

Example

```
>>> ct = ConfusionTable(120, 60, 20, 30)
>>> ct.population()
230
```

New in version 0.1.0.

pos_likelihood_ratio()
Return positive likelihood ratio.

Positive likelihood ratio is defined as

$$\frac{recall}{1 - specificity}$$

Cf. https://en.wikipedia.org/wiki/Likelihood_ratios_in_diagnostic_testing

Returns The positive likelihood ratio of the confusion table

Return type float

Example

```
>>> ct = ConfusionTable(120, 60, 20, 30)
>>> ct.pos_likelihood_ratio()
3.2
```

New in version 0.4.0.

pr_aghmean()
Return arithmetic-geometric-harmonic mean of precision & recall.

Iterates over arithmetic, geometric, & harmonic means until they converge to a single value (rounded to 12 digits), following the method described in [RaissouliLC09].

Returns The arithmetic-geometric-harmonic mean of the confusion table's precision & recall

Return type float

Example

```
>>> ct = ConfusionTable(120, 60, 20, 30)
>>> ct.pr_aghmean()
0.8280786712108288
```

New in version 0.1.0.

pr_agmean()
Return arithmetic-geometric mean of precision & recall.

Iterates between arithmetic & geometric means until they converge to a single value (rounded to 12 digits)

Cf. https://en.wikipedia.org/wiki/Arithmetic-geometric_mean

Returns The arithmetic-geometric mean of the confusion table's precision & recall

Return type float

Example

```
>>> ct = ConfusionTable(120, 60, 20, 30)
>>> ct.pr_agmean()
0.8283250315702829
```

New in version 0.1.0.

pr_amean()

Return arithmetic mean of precision & recall.

The arithmetic mean of precision and recall is defined as

$$\frac{precision \cdot recall}{2}$$

Cf. https://en.wikipedia.org/wiki/Arithmetic_mean

Returns The arithmetic mean of the confusion table's precision & recall

Return type float

Example

```
>>> ct = ConfusionTable(120, 60, 20, 30)
>>> ct.pr_amean()
0.8285714285714285
```

New in version 0.1.0.

pr_cmean()

Return contraharmonic mean of precision & recall.

The contraharmonic mean is

$$\frac{precision^2 + recall^2}{precision + recall}$$

Cf. https://en.wikipedia.org/wiki/Contraharmonic_mean

Returns The contraharmonic mean of the confusion table's precision & recall

Return type float

Example

```
>>> ct = ConfusionTable(120, 60, 20, 30)
>>> ct.pr_cmean()
0.8295566502463055
```

New in version 0.1.0.

pr_ghmean()
Return geometric-harmonic mean of precision & recall.

Iterates between geometric & harmonic means until they converge to a single value (rounded to 12 digits)

Cf. https://en.wikipedia.org/wiki/Geometric-harmonic_mean

Returns The geometric-harmonic mean of the confusion table's precision & recall

Return type float

Example

```
>>> ct = ConfusionTable(120, 60, 20, 30)
>>> ct.pr_ghmean()
0.8278323841238441
```

New in version 0.1.0.

pr_gmean()
Return geometric mean of precision & recall.

The geometric mean of precision and recall is defined as:

$$\sqrt{precision \cdot recall}$$

Cf. https://en.wikipedia.org/wiki/Geometric_mean

Returns The geometric mean of the confusion table's precision & recall

Return type float

Example

```
>>> ct = ConfusionTable(120, 60, 20, 30)
>>> ct.pr_gmean()
0.828078671210825
```

New in version 0.1.0.

pr_heronian_mean()
Return Heronian mean of precision & recall.

The Heronian mean of precision and recall is defined as

$$\frac{precision + \sqrt{precision \cdot recall} + recall}{3}$$

Cf. https://en.wikipedia.org/wiki/Heronian_mean

Returns The Heronian mean of the confusion table's precision & recall

Return type float

Example

```
>>> ct = ConfusionTable(120, 60, 20, 30)
>>> ct.pr_heronian_mean()
0.8284071761178939
```

New in version 0.1.0.

pr_hmean()

Return harmonic mean of precision & recall.

The harmonic mean of precision and recall is defined as

$$\frac{2 \cdot precision \cdot recall}{precision + recall}$$

Cf. https://en.wikipedia.org/wiki/Harmonic_mean

Returns The harmonic mean of the confusion table's precision & recall

Return type float

Example

```
>>> ct = ConfusionTable(120, 60, 20, 30)
>>> ct.pr_hmean()
0.8275862068965516
```

New in version 0.1.0.

pr_hoelder_mean(*exp=2*)

Return Hölder (power/generalized) mean of precision & recall.

The power mean of precision and recall is defined as

$$\frac{1}{2} \cdot \sqrt[exp]{precision^{exp} + recall^{exp}}$$

for $exp \neq 0$, and the geometric mean for $exp = 0$

Cf. https://en.wikipedia.org/wiki/Generalized_mean

Parameters **exp** (*float*) -- The exponent of the Hölder mean

Returns The Hölder mean for the given exponent of the confusion table's precision & recall

Return type float

Example

```
>>> ct = ConfusionTable(120, 60, 20, 30)
>>> ct.pr_hoelder_mean()
0.8290638930598233
```

New in version 0.1.0.

pr_imean()
Return identric (exponential) mean of precision & recall.

The identric mean is: precision if precision = recall, otherwise

$$\frac{1}{e} \cdot \sqrt[precision-recall]{\frac{precision^{precision}}{recall^{recall}}}$$

Cf. https://en.wikipedia.org/wiki/Identric_mean

Returns The identric mean of the confusion table's precision & recall

Return type float

Example

```
>>> ct = ConfusionTable(120, 60, 20, 30)
>>> ct.pr_imean()
0.8284071826325543
```

New in version 0.1.0.

pr_lehmer_mean(*exp=2.0*)
Return Lehmer mean of precision & recall.

The Lehmer mean is

$$\frac{precision^{exp} + recall^{exp}}{precision^{exp-1} + recall^{exp-1}}$$

Cf. https://en.wikipedia.org/wiki/Lehmer_mean

Parameters **exp** (*float*) -- The exponent of the Lehmer mean

Returns The Lehmer mean for the given exponent of the confusion table's precision & recall

Return type float

Example

```
>>> ct = ConfusionTable(120, 60, 20, 30)
>>> ct.pr_lehmer_mean()
0.8295566502463055
```

New in version 0.1.0.

pr_lmean()
Return logarithmic mean of precision & recall.

The logarithmic mean is: 0 if either precision or recall is 0, the precision if they are equal, otherwise

$$\frac{precision - recall}{ln(precision) - ln(recall)}$$

Cf. https://en.wikipedia.org/wiki/Logarithmic_mean

Returns The logarithmic mean of the confusion table's precision & recall

Return type float

Example

```
>>> ct = ConfusionTable(120, 60, 20, 30)
>>> ct.pr_lmean()
0.8282429171492667
```

New in version 0.1.0.

pr_qmean()
Return quadratic mean of precision & recall.

The quadratic mean of precision and recall is defined as

$$\sqrt{\frac{precision^2 + recall^2}{2}}$$

Cf. https://en.wikipedia.org/wiki/Quadratic_mean

Returns The quadratic mean of the confusion table's precision & recall

Return type float

Example

```
>>> ct = ConfusionTable(120, 60, 20, 30)
>>> ct.pr_qmean()
0.8290638930598233
```

New in version 0.1.0.

pr_seiffert_mean()
Return Seiffert's mean of precision & recall.

Seiffert's mean of precision and recall is

$$\frac{precision - recall}{4 \cdot arctan\sqrt{\frac{precision}{recall}} - \pi}$$

It is defined in [Sei93].

Returns Seiffert's mean of the confusion table's precision & recall

Return type float

Example

```
>>> ct = ConfusionTable(120, 60, 20, 30)
>>> ct.pr_seiffert_mean()
0.8284071696048312
```

New in version 0.1.0.

precision()
Return precision.

Precision is defined as

$$\frac{tp}{tp + fp}$$

AKA positive predictive value (PPV)

Cf. https://en.wikipedia.org/wiki/Precision_and_recall

Cf. https://en.wikipedia.org/wiki/Information_retrieval#Precision

Returns The precision of the confusion table

Return type float

Example

```
>>> ct = ConfusionTable(120, 60, 20, 30)
>>> ct.precision()
0.8571428571428571
```

New in version 0.1.0.

precision_gain()
Return gain in precision.

The gain in precision is defined as

$$G(precision) = \frac{precision}{random\ precision}$$

Cf. https://en.wikipedia.org/wiki/Gain_(information_retrieval)

Returns The gain in precision of the confusion table

Return type float

Example

```
>>> ct = ConfusionTable(120, 60, 20, 30)
>>> ct.precision_gain()
1.3142857142857143
```

New in version 0.1.0.

pred_neg_pop()
Return predicted negative population.

Returns The predicted negative population of the confusion table

Return type int

Example

```
>>> ct = ConfusionTable(120, 60, 20, 30)
>>> ct.pred_neg_pop()
90
```

New in version 0.1.0.

Changed in version 0.4.0: renamed from test_neg_pop

New in version 0.1.0.

pred_pos_pop()
Return predicted positive population.

Returns The predicted positive population of the confusion table

Return type int

Example

```
>>> ct = ConfusionTable(120, 60, 20, 30)
>>> ct.pred_pos_pop()
140
```

New in version 0.1.0.

Changed in version 0.4.0: renamed from test_pos_pop

New in version 0.1.0.

predicted_entropy()
Return the predicted entropy.

Implementation based on https://github.com/Magnetic/proficiency-metric

Returns The predicted entropy of the confusion table

Return type float

Example

```
>>> ct = ConfusionTable(120, 60, 20, 30)
>>> ct.predicted_entropy()
0.6693279632926457
```

New in version 0.4.0.

prevalence()
Return prevalence.

Prevalence is defined as

$$\frac{condition positive}{population}$$

Cf. https://en.wikipedia.org/wiki/Prevalence

Returns The prevelence of the confusion table

Return type float

Example

```
>>> ct = ConfusionTable(120, 60, 20, 30)
>>> ct.prevalence()
0.6521739130434783
```

New in version 0.4.0.

proficiency()
Return the proficiency.

Implementation based on https://github.com/Magnetic/proficiency-metric [SLaclavik15]

AKA uncertainty coefficient

Cf. https://en.wikipedia.org/wiki/Uncertainty_coefficient

Returns The proficiency of the confusion table

Return type float

Example

```
>>> ct = ConfusionTable(120, 60, 20, 30)
>>> ct.proficiency()
0.228116219897929
```

New in version 0.4.0.

recall()
Return recall.

Recall is defined as

$$\frac{tp}{tp + fn}$$

AKA sensitivity

AKA true positive rate (TPR)

Cf. https://en.wikipedia.org/wiki/Precision_and_recall

Cf. https://en.wikipedia.org/wiki/Sensitivity_(test)

Cf. https://en.wikipedia.org/wiki/Information_retrieval#Recall

Returns The recall of the confusion table

Return type float

Example

```
>>> ct = ConfusionTable(120, 60, 20, 30)
>>> ct.recall()
0.8
```

New in version 0.1.0.

significance()

Return the significance, χ^2.

Significance is defined as

$$\chi^2 = \frac{(tp \cdot tn - fp \cdot fn)^2 (tp + tn + fp + fn)}{((tp + fp)(tp + fn)(tn + fp)(tn + fn)}.$$

Also: $\chi^2 = MCC^2 \cdot n$

Cf. https://en.wikipedia.org/wiki/Pearson%27s_chi-square_test

Returns The significance of the confusion table

Return type float

Example

```
>>> ct = ConfusionTable(120, 60, 20, 30)
>>> ct.significance()
66.26190476190476
```

New in version 0.1.0.

specificity()

Return specificity.

Specificity is defined as

$$\frac{tn}{tn + fp}$$

AKA true negative rate (TNR)

AKA inverse recall

Cf. https://en.wikipedia.org/wiki/Specificity_(tests)

Returns The specificity of the confusion table

Return type float

Example

```
>>> ct = ConfusionTable(120, 60, 20, 30)
>>> ct.specificity()
0.75
```

New in version 0.1.0.

to_dict()
Cast to dict.

Returns The confusion table as a dict

Return type dict

Example

```
>>> ct = ConfusionTable(120, 60, 20, 30)
>>> import pprint
>>> pprint.pprint(ct.to_dict())
{'fn': 30, 'fp': 20, 'tn': 60, 'tp': 120}
```

New in version 0.1.0.

to_tuple()
Cast to tuple.

Returns The confusion table as a 4-tuple (tp, tn, fp, fn)

Return type tuple

Example

```
>>> ct = ConfusionTable(120, 60, 20, 30)
>>> ct.to_tuple()
(120, 60, 20, 30)
```

New in version 0.1.0.

true_neg()
Return true negatives.

Returns The true negatives of the confusion table

Return type int

Example

```
>>> ct = ConfusionTable(120, 60, 20, 30)
>>> ct.true_neg()
60
```

New in version 0.1.0.

true_pos()
Return true positives.

Returns The true positives of the confusion table

Return type int

Example

```
>>> ct = ConfusionTable(120, 60, 20, 30)
>>> ct.true_pos()
120
```

New in version 0.1.0.

abydos.stats.**amean**(*nums*)
Return arithmetic mean.

The arithmetic mean is defined as

$$\frac{\sum nums}{|nums|}$$

Cf. https://en.wikipedia.org/wiki/Arithmetic_mean

Parameters **nums** (*list*) -- A series of numbers

Returns The arithmetic mean of nums

Return type float

Examples

```
>>> amean([1, 2, 3, 4])
2.5
>>> amean([1, 2])
1.5
>>> amean([0, 5, 1000])
335.0
```

New in version 0.1.0.

abydos.stats.**gmean**(*nums*)
Return geometric mean.

The geometric mean is defined as

$$\sqrt[|nums|]{\prod_i nums_i}$$

Cf. https://en.wikipedia.org/wiki/Geometric_mean

Parameters **nums** (*list*) -- A series of numbers

Returns The geometric mean of nums

Return type float

Examples

```
>>> gmean([1, 2, 3, 4])
2.213363839400643
>>> gmean([1, 2])
1.4142135623730951
>>> gmean([0, 5, 1000])
0.0
```

New in version 0.1.0.

abydos.stats.**hmean**(*nums*)

Return harmonic mean.

The harmonic mean is defined as

$$\frac{|nums|}{\sum_i \frac{1}{nums_i}}$$

Following the behavior of Wolfram|Alpha: - If one of the values in nums is 0, return 0. - If more than one value in nums is 0, return NaN.

Cf. https://en.wikipedia.org/wiki/Harmonic_mean

Parameters **nums** (*list*) -- A series of numbers

Returns The harmonic mean of nums

Return type float

Raises **ValueError** -- hmean requires at least one value

Examples

```
>>> hmean([1, 2, 3, 4])
1.9200000000000004
>>> hmean([1, 2])
1.3333333333333333
>>> hmean([0, 5, 1000])
0
```

New in version 0.1.0.

abydos.stats.**agmean**(*nums*, *prec=12*)

Return arithmetic-geometric mean.

Iterates between arithmetic & geometric means until they converge to a single value (rounded to 10 digits).

Cf. https://en.wikipedia.org/wiki/Arithmetic-geometric_mean

Parameters **nums** (*list*) -- A series of numbers

Returns

- *float* -- The arithmetic-geometric mean of nums
- **prec** (*int*) -- Digits of precision when testing convergeance

Examples

```
>>> agmean([1, 2, 3, 4])
2.3545004777751077
>>> agmean([1, 2])
1.4567910310469068
>>> agmean([0, 5, 1000])
2.9753977059954195e-13
```

New in version 0.1.0.

abydos.stats.**ghmean**(*nums*, *prec=12*)

Return geometric-harmonic mean.

Iterates between geometric & harmonic means until they converge to a single value (rounded to 10 digits).

Cf. https://en.wikipedia.org/wiki/Geometric-harmonic_mean

Parameters

- **nums** (*list*) -- A series of numbers
- **prec** (*int*) -- Digits of precision when testing convergeance

Returns The geometric-harmonic mean of nums

Return type float

Examples

```
>>> ghmean([1, 2, 3, 4])
2.058868154613003
>>> ghmean([1, 2])
1.3728805006183502
>>> ghmean([0, 5, 1000])
0.0
```

```
>>> ghmean([0, 0])
0.0
>>> ghmean([0, 0, 5])
nan
```

New in version 0.1.0.

abydos.stats.**aghmean**(*nums*, *prec=12*)

Return arithmetic-geometric-harmonic mean.

Iterates over arithmetic, geometric, & harmonic means until they converge to a single value (rounded to 10 digits), following the method described in [RaissouliLC09].

Parameters

- **nums** (*list*) -- A series of numbers
- **prec** (*int*) -- Digits of precision when testing convergeance

Returns The arithmetic-geometric-harmonic mean of nums

Return type float

Examples

```
>>> aghmean([1, 2, 3, 4])
2.198327159900212
>>> aghmean([1, 2])
1.4142135623731884
>>> aghmean([0, 5, 1000])
335.0
```

New in version 0.1.0.

abydos.stats.**cmean**(*nums*)

Return contraharmonic mean.

The contraharmonic mean is

$$\frac{\sum_i x_i^2}{\sum_i x_i}$$

Cf. https://en.wikipedia.org/wiki/Contraharmonic_mean

Parameters **nums** (*list*) -- A series of numbers

Returns The contraharmonic mean of nums

Return type float

Examples

```
>>> cmean([1, 2, 3, 4])
3.0
>>> cmean([1, 2])
1.6666666666666667
>>> cmean([0, 5, 1000])
995.0497512437811
```

New in version 0.1.0.

abydos.stats.**imean**(*nums*)

Return identric (exponential) mean.

The identric mean of two numbers x and y is: x if x = y otherwise

$$\frac{1}{e}\sqrt[x-y]{\frac{x^x}{y^y}}$$

Cf. https://en.wikipedia.org/wiki/Identric_mean

Parameters **nums** (*list*) -- A series of numbers

Returns The identric mean of nums

Return type float

Raises **ValueError** -- imean supports no more than two values

Examples

```
>>> imean([1, 2])
1.4715177646857693
>>> imean([1, 0])
nan
>>> imean([2, 4])
2.9430355293715387
```

New in version 0.1.0.

abydos.stats.**lmean**(*nums*)

Return logarithmic mean.

The logarithmic mean of an arbitrarily long series is defined by http://www.survo.fi/papers/logmean.pdf as

$$L(x_1, x_2, ..., x_n) = (n-1)! \sum_{i=1}^{n} \frac{x_i}{\prod\limits_{\substack{j=1 \\ j \neq i}}^{n} ln \frac{x_i}{x_j}}$$

Cf. https://en.wikipedia.org/wiki/Logarithmic_mean

Parameters **nums** (*list*) -- A series of numbers

Returns The logarithmic mean of nums

Return type float

Raises **ValueError** -- No two values in the nums list may be equal

Examples

```
>>> lmean([1, 2, 3, 4])
2.2724242417489258
>>> lmean([1, 2])
1.4426950408889634
```

New in version 0.1.0.

abydos.stats.**qmean**(*nums*)

Return quadratic mean.

The quadratic mean is defined as

$$\sqrt{\sum_i \frac{num_i^2}{|nums|}}$$

Cf. https://en.wikipedia.org/wiki/Quadratic_mean

Parameters **nums** (*list*) -- A series of numbers

Returns The quadratic mean of nums

Return type float

Examples

```
>>> qmean([1, 2, 3, 4])
2.7386127875258306
>>> qmean([1, 2])
1.5811388300841898
>>> qmean([0, 5, 1000])
577.3574860228857
```

New in version 0.1.0.

abydos.stats.**heronian_mean**(*nums*)

Return Heronian mean.

The Heronian mean is:

$$\frac{\sum_{i,j} \sqrt{x_i \cdot x_j}}{|nums| \cdot \frac{|nums|+1}{2}}$$

for $j \geq i$

Cf. https://en.wikipedia.org/wiki/Heronian_mean

Parameters **nums** (*list*) -- A series of numbers

Returns The Heronian mean of nums

Return type float

Examples

```
>>> heronian_mean([1, 2, 3, 4])
2.3888282852609093
>>> heronian_mean([1, 2])
1.4714045207910316
>>> heronian_mean([0, 5, 1000])
179.28511301977582
```

New in version 0.1.0.

abydos.stats.**hoelder_mean**(*nums*, *exp=2*)

Return Hölder (power/generalized) mean.

The Hölder mean is defined as:

$$\sqrt[p]{\frac{1}{|nums|} \cdot \sum_i x_i^p}$$

for $p \neq 0$, and the geometric mean for $p = 0$

Cf. https://en.wikipedia.org/wiki/Generalized_mean

Parameters

- **nums** (*list*) -- A series of numbers
- **exp** (*numeric*) -- The exponent of the Hölder mean

Returns The Hölder mean of nums for the given exponent

Return type float

Examples

```
>>> hoelder_mean([1, 2, 3, 4])
2.7386127875258306
>>> hoelder_mean([1, 2])
1.5811388300841898
>>> hoelder_mean([0, 5, 1000])
577.3574860228857
```

New in version 0.1.0.

abydos.stats.**lehmer_mean**(*nums*, *exp=2*)

Return Lehmer mean.

The Lehmer mean is

$$\frac{\sum_i x_i^p}{\sum_i x_i^(p-1)}$$

Cf. https://en.wikipedia.org/wiki/Lehmer_mean

Parameters

- **nums** (*list*) -- A series of numbers
- **exp** (*numeric*) -- The exponent of the Lehmer mean

Returns The Lehmer mean of nums for the given exponent

Return type float

Examples

```
>>> lehmer_mean([1, 2, 3, 4])
3.0
>>> lehmer_mean([1, 2])
1.6666666666666667
>>> lehmer_mean([0, 5, 1000])
995.0497512437811
```

New in version 0.1.0.

abydos.stats.**seiffert_mean**(*nums*)

Return Seiffert's mean.

Seiffert's mean of two numbers x and y is

$$\frac{x - y}{4 \cdot arctan\sqrt{\frac{x}{y}} - \pi}$$

It is defined in [Sei93].

Parameters **nums** (*list*) -- A series of numbers

Returns Sieffert's mean of nums

Return type float

Raises **ValueError** -- seiffert_mean supports no more than two values

Examples

```
>>> seiffert_mean([1, 2])
1.4712939827611637
>>> seiffert_mean([1, 0])
0.3183098861837907
>>> seiffert_mean([2, 4])
2.9425879655223275
>>> seiffert_mean([2, 1000])
336.84053300118825
```

New in version 0.1.0.

abydos.stats.**median**(*nums*)

Return median.

With numbers sorted by value, the median is the middle value (if there is an odd number of values) or the arithmetic mean of the two middle values (if there is an even number of values).

Cf. https://en.wikipedia.org/wiki/Median

Parameters **nums** (*list*) -- A series of numbers

Returns The median of nums

Return type int or float

Examples

```
>>> median([1, 2, 3])
2
>>> median([1, 2, 3, 4])
2.5
>>> median([1, 2, 2, 4])
2
```

New in version 0.1.0.

abydos.stats.**midrange**(*nums*)

Return midrange.

The midrange is the arithmetic mean of the maximum & minimum of a series.

Cf. https://en.wikipedia.org/wiki/Midrange

Parameters **nums** (*list*) -- A series of numbers

Returns The midrange of nums

Return type float

Examples

```
>>> midrange([1, 2, 3])
2.0
>>> midrange([1, 2, 2, 3])
2.0
>>> midrange([1, 2, 1000, 3])
500.5
```

New in version 0.1.0.

abydos.stats.**mode**(*nums*)

Return the mode.

The mode of a series is the most common element of that series

Cf. https://en.wikipedia.org/wiki/Mode_(statistics)

Parameters **nums** (*list*) -- A series of numbers

Returns The mode of nums

Return type int or float

Example

```
>>> mode([1, 2, 2, 3])
2
```

New in version 0.1.0.

abydos.stats.**std**(*nums*, *mean_func=<function amean>*, *ddof=0*)

Return the standard deviation.

The standard deviation of a series of values is the square root of the variance.

Cf. https://en.wikipedia.org/wiki/Standard_deviation

Parameters

- **nums** (*list*) -- A series of numbers
- **mean_func** (*function*) -- A mean function (amean by default)
- **ddof** (*int*) -- The degrees of freedom (0 by default)

Returns The standard deviation of the values in the series

Return type float

Examples

```
>>> std([1, 1, 1, 1])
0.0
>>> round(std([1, 2, 3, 4]), 12)
1.11803398875
>>> round(std([1, 2, 3, 4], ddof=1), 12)
1.290994448736
```

New in version 0.3.0.

abydos.stats.**var**(*nums*, *mean_func=<function amean>*, *ddof=0*)

Calculate the variance.

The variance (σ^2) of a series of numbers (x_i) with mean μ and population N is:

$$\sigma^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2$$

Cf. https://en.wikipedia.org/wiki/Variance

Parameters

- **nums** (*list*) -- A series of numbers
- **mean_func** (*function*) -- A mean function (amean by default)
- **ddof** (*int*) -- The degrees of freedom (0 by default)

Returns The variance of the values in the series

Return type float

Examples

```
>>> var([1, 1, 1, 1])
0.0
>>> var([1, 2, 3, 4])
1.25
>>> round(var([1, 2, 3, 4], ddof=1), 12)
1.666666666667
```

New in version 0.3.0.

abydos.stats.**mean_pairwise_similarity**(*collection*, *metric=<function sim_levenshtein>*, *mean_func=<function hmean>*, *symmetric=False*)

Calculate the mean pairwise similarity of a collection of strings.

Takes the mean of the pairwise similarity between each member of a collection, optionally in both directions (for asymmetric similarity metrics.

Parameters

- **collection** (*list*) -- A collection of terms or a string that can be split
- **metric** (*function*) -- A similarity metric function
- **mean_func** (*function*) -- A mean function that takes a list of values and returns a float
- **symmetric** (*bool*) -- Set to True if all pairwise similarities should be calculated in both directions

Returns The mean pairwise similarity of a collection of strings

Return type float

Raises

- **ValueError** -- mean_func must be a function
- **ValueError** -- metric must be a function
- **ValueError** -- collection is neither a string nor iterable type
- **ValueError** -- collection has fewer than two members

Examples

```
>>> round(mean_pairwise_similarity(['Christopher', 'Kristof',
... 'Christobal']), 12)
0.519801980198
>>> round(mean_pairwise_similarity(['Niall', 'Neal', 'Neil']), 12)
0.545454545455
```

New in version 0.1.0.

abydos.stats.**pairwise_similarity_statistics**(*src_collection*, *tar_collection*, *metric=<function sim_levenshtein>*, *mean_func=<function amean>*, *symmetric=False*)

Calculate the pairwise similarity statistics a collection of strings.

Calculate pairwise similarities among members of two collections, returning the maximum, minimum, mean (according to a supplied function, arithmetic mean, by default), and (population) standard deviation of those similarities.

Parameters

- **src_collection** (*list*) -- A collection of terms or a string that can be split
- **tar_collection** (*list*) -- A collection of terms or a string that can be split
- **metric** (*function*) -- A similarity metric function
- **mean_func** (*function*) -- A mean function that takes a list of values and returns a float
- **symmetric** (*bool*) -- Set to True if all pairwise similarities should be calculated in both directions

Returns The max, min, mean, and standard deviation of similarities

Return type tuple

Raises

- **ValueError** -- mean_func must be a function
- **ValueError** -- metric must be a function
- **ValueError** -- src_collection is neither a string nor iterable
- **ValueError** -- tar_collection is neither a string nor iterable

Example

```
>>> tuple(round(_, 12) for _ in pairwise_similarity_statistics(
... ['Christopher', 'Kristof', 'Christobal'], ['Niall', 'Neal', 'Neil']))
(0.2, 0.0, 0.118614718615, 0.075070477184)
```

New in version 0.3.0.

3.1.1.8 abydos.stemmer package

abydos.stemmer.

The stemmer package collects stemmer classes for a number of languages including:

- English stemmers:
  - Lovins' (*Lovins*)
  - Porter (*Porter*)
  - Porter2 (i.e. Snowball English) (*Porter2*)
  - UEA-Lite (*UEALite*)
  - Paice-Husk (*PaiceHusk*)
  - S-stemmer (*SStemmer*)
- German stemmers:
  - Caumanns' (*Caumanns*)
  - CLEF German (*CLEFGerman*)
  - CLEF German Plus (*CLEFGermanPlus*)
  - Snowball German (*SnowballGerman*)
- Swedish stemmers:
  - CLEF Swedish (*CLEFSwedish*)
  - Snowball Swedish (*SnowballSwedish*)
- Latin stemmer:
  - Schinke (*Schinke*)
- Danish stemmer:
  - Snowball Danish (*SnowballDanish*)

- Dutch stemmer:
  - Snowball Dutch (*SnowballDutch*)
- Norwegian stemmer:
  - Snowball Norwegian (*SnowballNorwegian*)

Each stemmer has a `stem` method, which takes a word and returns its stemmed form:

```
>>> stmr = Porter()
>>> stmr.stem('democracy')
'democraci'
>>> stmr.stem('trusted')
'trust'
```

---

class abydos.stemmer.**Lovins**

Bases: abydos.stemmer._stemmer._Stemmer

Lovins stemmer.

The Lovins stemmer is described in Julie Beth Lovins's article [Lov68].

New in version 0.3.6.

Initialize the stemmer.

New in version 0.3.6.

stem(*word*)

Return Lovins stem.

Parameters **word** (*str*) -- The word to stem

Returns Word stem

Return type str

Examples

```
>>> stmr = Lovins()
>>> stmr.stem('reading')
'read'
>>> stmr.stem('suspension')
'suspens'
>>> stmr.stem('elusiveness')
'elus'
```

New in version 0.2.0.

Changed in version 0.3.6: Encapsulated in class

abydos.stemmer.**lovins**(*word*)

Return Lovins stem.

This is a wrapper for *Lovins.stem()*.

Parameters **word** (*str*) -- The word to stem

Returns str

Return type Word stem

Examples

```
>>> lovins('reading')
'read'
>>> lovins('suspension')
'suspens'
>>> lovins('elusiveness')
'elus'
```

New in version 0.2.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the Lovins.stem method instead.

class abydos.stemmer.**PaiceHusk**

Bases: abydos.stemmer._stemmer._Stemmer

Paice-Husk stemmer.

Implementation of the Paice-Husk Stemmer, also known as the Lancaster Stemmer, developed by Chris Paice, with the assistance of Gareth Husk

This is based on the algorithm's description in [Pai90].

New in version 0.3.6.

stem(*word*)

Return Paice-Husk stem.

Parameters **word** (*str*) -- The word to stem

Returns Word stem

Return type str

Examples

```
>>> stmr = PaiceHusk()
>>> stmr.stem('assumption')
'assum'
>>> stmr.stem('verifiable')
'ver'
>>> stmr.stem('fancies')
'fant'
>>> stmr.stem('fanciful')
'fancy'
>>> stmr.stem('torment')
'tor'
```

New in version 0.3.0.

Changed in version 0.3.6: Encapsulated in class

abydos.stemmer.**paice_husk**(*word*)

Return Paice-Husk stem.

This is a wrapper for *PaiceHusk.stem()*.

Parameters **word** (*str*) -- The word to stem

Returns Word stem

Return type str

Examples

```
>>> paice_husk('assumption')
'assum'
>>> paice_husk('verifiable')
'ver'
>>> paice_husk('fancies')
'fant'
>>> paice_husk('fanciful')
'fancy'
>>> paice_husk('torment')
'tor'
```

New in version 0.3.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the PaiceHusk.stem method instead.

class abydos.stemmer.**UEALite**(*max_word_length=20*, *max_acro_length=8*, *return_rule_no=False*, *var='standard'*)

Bases: abydos.stemmer._stemmer._Stemmer

UEA-Lite stemmer.

The UEA-Lite stemmer is discussed in [JS05].

This is chiefly based on the Java implementation of the algorithm, with variants based on the Perl implementation and Jason Adams' Ruby port.

Java version: [Chu] Perl version: [JS05] Ruby version: [Ada17]

New in version 0.3.6.

Initialize UEALite instance.

Parameters

- **max_word_length** (*int*) -- The maximum word length allowed
- **max_acro_length** (*int*) -- The maximum acronym length allowed
- **return_rule_no** (*bool*) -- If True, returns the stem along with rule number
- **var** (*str*) -- Variant rules to use:
  - `standard` to use the original (Java-version) rules
  - `Adams` to use Jason Adams' rules
  - `Perl` to use the original Perl rules

New in version 0.4.0.

stem(*word*)

Return UEA-Lite stem.

Parameters **word** (*str*) -- The word to stem

Returns Word stem

Return type str or (str, int)

Examples

```
>>> uealite('readings')
'read'
>>> uealite('insulted')
'insult'
>>> uealite('cussed')
'cuss'
>>> uealite('fancies')
'fancy'
>>> uealite('eroded')
'erode'
```

New in version 0.1.0.

Changed in version 0.3.6: Encapsulated in class

abydos.stemmer.**uealite**(*word*, *max_word_length=20*, *max_acro_length=8*, *return_rule_no=False*, *var='standard'*)

Return UEA-Lite stem.

This is a wrapper for *UEALite.stem()*.

Parameters

- **word** (*str*) -- The word to stem
- **max_word_length** (*int*) -- The maximum word length allowed
- **max_acro_length** (*int*) -- The maximum acronym length allowed
- **return_rule_no** (*bool*) -- If True, returns the stem along with rule number
- **var** (*str*) -- Variant rules to use:
  - `Adams` to use Jason Adams' rules
  - `Perl` to use the original Perl rules

Returns Word stem

Return type str or (str, int)

Examples

```
>>> uealite('readings')
'read'
>>> uealite('insulted')
'insult'
>>> uealite('cussed')
'cuss'
>>> uealite('fancies')
'fancy'
>>> uealite('eroded')
'erode'
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the UEALite.stem method instead.

class abydos.stemmer.**SStemmer**

Bases: abydos.stemmer._stemmer._Stemmer

S-stemmer.

The S stemmer is defined in [Har91].

New in version 0.3.6.

stem(*word*)

Return the S-stemmed form of a word.

Parameters **word** (*str*) -- The word to stem

Returns Word stem

Return type str

Examples

```
>>> stmr = SStemmer()
>>> stmr.stem('summaries')
'summary'
>>> stmr.stem('summary')
'summary'
>>> stmr.stem('towers')
'tower'
>>> stmr.stem('reading')
'reading'
>>> stmr.stem('census')
'census'
```

New in version 0.3.0.

Changed in version 0.3.6: Encapsulated in class

abydos.stemmer.**s_stemmer**(*word*)

Return the S-stemmed form of a word.

This is a wrapper for *SStemmer.stem()*.

Parameters **word** (*str*) -- The word to stem

Returns Word stem

Return type str

Examples

```
>>> s_stemmer('summaries')
'summary'
>>> s_stemmer('summary')
'summary'
>>> s_stemmer('towers')
'tower'
>>> s_stemmer('reading')
'reading'
>>> s_stemmer('census')
'census'
```

New in version 0.3.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the SStemmer.stem method instead.

class abydos.stemmer.**Caumanns**
Bases: abydos.stemmer._stemmer._Stemmer

Caumanns stemmer.

Jörg Caumanns' stemmer is described in his article in [Cau99].

This implementation is based on the GermanStemFilter described at [Lan13].

New in version 0.3.6.

stem(*word*)
Return Caumanns German stem.

Parameters **word** (*str*) -- The word to stem

Returns Word stem

Return type str

Examples

```
>>> stmr = Caumanns()
>>> stmr.stem('lesen')
'les'
>>> stmr.stem('graues')
'grau'
>>> stmr.stem('buchstabieren')
'buchstabier'
```

New in version 0.2.0.

Changed in version 0.3.6: Encapsulated in class

abydos.stemmer.**caumanns**(*word*)
Return Caumanns German stem.

This is a wrapper for *Caumanns.stem()*.

Parameters **word** (*str*) -- The word to stem

Returns Word stem

Return type str

Examples

```
>>> caumanns('lesen')
'les'
>>> caumanns('graues')
'grau'
>>> caumanns('buchstabieren')
'buchstabier'
```

New in version 0.2.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the Caumanns.stem method instead.

class abydos.stemmer.**Schinke**

Bases: abydos.stemmer._stemmer._Stemmer

Schinke stemmer.

This is defined in [SGRW96].

New in version 0.3.6.

stem(*word*)

Return the stem of a word according to the Schinke stemmer.

Parameters **word** (*str*) -- The word to stem

Returns Word stem

Return type str

Examples

```
>>> stmr = Schinke()
>>> stmr.stem('atque')
{'n': 'atque', 'v': 'atque'}
>>> stmr.stem('census')
{'n': 'cens', 'v': 'censu'}
>>> stmr.stem('virum')
{'n': 'uir', 'v': 'uiru'}
>>> stmr.stem('populusque')
{'n': 'popul', 'v': 'populu'}
>>> stmr.stem('senatus')
{'n': 'senat', 'v': 'senatu'}
```

New in version 0.3.0.

Changed in version 0.3.6: Encapsulated in class

abydos.stemmer.**schinke**(*word*)

Return the stem of a word according to the Schinke stemmer.

This is a wrapper for *Schinke.stem()*.

Parameters **word** (*str*) -- The word to stem

Returns Word stem

Return type str

Examples

```
>>> schinke('atque')
{'n': 'atque', 'v': 'atque'}
>>> schinke('census')
{'n': 'cens', 'v': 'censu'}
>>> schinke('virum')
{'n': 'uir', 'v': 'uiru'}
>>> schinke('populusque')
{'n': 'popul', 'v': 'populu'}
>>> schinke('senatus')
{'n': 'senat', 'v': 'senatu'}
```

New in version 0.3.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the Schinke.stem method instead.

class abydos.stemmer.**Porter**(*early_english=False*)
Bases: abydos.stemmer._stemmer._Stemmer

Porter stemmer.

The Porter stemmer is described in [Por80].

New in version 0.3.6.

Initialize Porter instance.

Parameters **early_english** (*bool*) -- Set to True in order to remove -eth & -est (2nd & 3rd person singular verbal agreement suffixes)

New in version 0.4.0.

stem(*word*)
Return Porter stem.

Parameters **word** (*str*) -- The word to stem

Returns Word stem

Return type str

Examples

```
>>> stmr = Porter()
>>> stmr.stem('reading')
'read'
>>> stmr.stem('suspension')
'suspens'
>>> stmr.stem('elusiveness')
'elus'
```

```
>>> stmr = Porter(early_english=True)
>>> stmr.stem('eateth')
'eat'
```

New in version 0.1.0.

Changed in version 0.3.6: Encapsulated in class

abydos.stemmer.**porter**(*word*, *early_english=False*)
Return Porter stem.

This is a wrapper for *Porter.stem()*.

Parameters

- **word** (*str*) -- The word to stem
- **early_english** (*bool*) -- Set to True in order to remove -eth & -est (2nd & 3rd person singular verbal agreement suffixes)

Returns Word stem

Return type str

Examples

```
>>> porter('reading')
'read'
>>> porter('suspension')
'suspens'
>>> porter('elusiveness')
'elus'
```

```
>>> porter('eateth', early_english=True)
'eat'
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the Porter.stem method instead.

class abydos.stemmer.**Porter2**(*early_english=False*)

Bases: abydos.stemmer._snowball._Snowball

Porter2 (Snowball English) stemmer.

The Porter2 (Snowball English) stemmer is defined in [Por02].

New in version 0.3.6.

Initialize Porter2 instance.

Parameters **early_english** (*bool*) -- Set to True in order to remove -eth & -est (2nd & 3rd person singular verbal agreement suffixes)

New in version 0.4.0.

stem(*word*)

Return the Porter2 (Snowball English) stem.

Parameters **word** (*str*) -- The word to stem

Returns Word stem

Return type str

Examples

```
>>> stmr = Porter2()
>>> stmr.stem('reading')
'read'
>>> stmr.stem('suspension')
'suspens'
>>> stmr.stem('elusiveness')
'elus'
```

```
>>> stmr = Porter2(early_english=True)
>>> stmr.stem('eateth')
'eat'
```

New in version 0.1.0.

Changed in version 0.3.6: Encapsulated in class

abydos.stemmer.**porter2**(*word*, *early_english=False*)

Return the Porter2 (Snowball English) stem.

This is a wrapper for *Porter2.stem()*.

Parameters

- **word** (*str*) -- The word to stem
- **early_english** (*bool*) -- Set to True in order to remove -eth & -est (2nd & 3rd person singular verbal agreement suffixes)

Returns Word stem

Return type str

Examples

```
>>> porter2('reading')
'read'
>>> porter2('suspension')
'suspens'
>>> porter2('elusiveness')
'elus'
```

```
>>> porter2('eateth', early_english=True)
'eat'
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the Porter2.stem method instead.

class abydos.stemmer.**SnowballDanish**

Bases: abydos.stemmer._snowball._Snowball

Snowball Danish stemmer.

The Snowball Danish stemmer is defined at: http://snowball.tartarus.org/algorithms/danish/stemmer.html

New in version 0.3.6.

stem(*word*)

Return Snowball Danish stem.

Parameters **word** (*str*) -- The word to stem

Returns Word stem

Return type str

Examples

```
>>> stmr = SnowballDanish()
>>> stmr.stem('underviser')
'undervis'
>>> stmr.stem('suspension')
'suspension'
>>> stmr.stem('sikkerhed')
'sikker'
```

New in version 0.1.0.

Changed in version 0.3.6: Encapsulated in class

abydos.stemmer.**sb_danish**(*word*)

Return Snowball Danish stem.

This is a wrapper for *SnowballDanish.stem()*.

Parameters **word** (*str*) -- The word to stem

Returns Word stem

Return type str

Examples

```
>>> sb_danish('underviser')
'undervis'
>>> sb_danish('suspension')
'suspension'
>>> sb_danish('sikkerhed')
'sikker'
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the SnowballDanish.stem method instead.

class abydos.stemmer.**SnowballDutch**

Bases: abydos.stemmer._snowball._Snowball

Snowball Dutch stemmer.

The Snowball Dutch stemmer is defined at: http://snowball.tartarus.org/algorithms/dutch/stemmer.html

New in version 0.3.6.

stem(*word*)

Return Snowball Dutch stem.

Parameters **word** (*str*) -- The word to stem

Returns Word stem

Return type str

Examples

```
>>> stmr = SnowballDutch()
>>> stmr.stem('lezen')
'lez'
>>> stmr.stem('opschorting')
'opschort'
>>> stmr.stem('ongrijpbaarheid')
'ongrijp'
```

New in version 0.1.0.

Changed in version 0.3.6: Encapsulated in class

abydos.stemmer.**sb_dutch**(*word*)

Return Snowball Dutch stem.

This is a wrapper for *SnowballDutch.stem()*.

Parameters **word** (*str*) -- The word to stem

Returns Word stem

Return type str

Examples

```
>>> sb_dutch('lezen')
'lez'
>>> sb_dutch('opschorting')
'opschort'
>>> sb_dutch('ongrijpbaarheid')
'ongrijp'
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the SnowballDutch.stem method instead.

class abydos.stemmer.**SnowballGerman**(*alternate_vowels=False*)

Bases: abydos.stemmer._snowball._Snowball

Snowball German stemmer.

The Snowball German stemmer is defined at: http://snowball.tartarus.org/algorithms/german/stemmer.html

New in version 0.3.6.

Initialize SnowballGerman instance.

Parameters **alternate_vowels** (*bool*) -- Composes ae as ä, oe as ö, and ue as ü before running the algorithm

New in version 0.4.0.

stem(*word*)

Return Snowball German stem.

Parameters **word** (*str*) -- The word to stem

Returns Word stem

Return type str

Examples

```
>>> stmr = SnowballGerman()
>>> stmr.stem('lesen')
'les'
>>> stmr.stem('graues')
'grau'
>>> stmr.stem('buchstabieren')
'buchstabi'
```

New in version 0.1.0.

Changed in version 0.3.6: Encapsulated in class

abydos.stemmer.**sb_german**(*word*, *alternate_vowels=False*)
Return Snowball German stem.

This is a wrapper for *SnowballGerman.stem()*.

Parameters

- **word** (*str*) -- The word to stem
- **alternate_vowels** (*bool*) -- Composes ae as ä, oe as ö, and ue as ü before running the algorithm

Returns Word stem

Return type str

Examples

```
>>> sb_german('lesen')
'les'
>>> sb_german('graues')
'grau'
>>> sb_german('buchstabieren')
'buchstabi'
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the SnowballGerman.stem method instead.

class abydos.stemmer.**SnowballNorwegian**
Bases: abydos.stemmer._snowball._Snowball

Snowball Norwegian stemmer.

The Snowball Norwegian stemmer is defined at: http://snowball.tartarus.org/algorithms/norwegian/stemmer.html

New in version 0.3.6.

stem(*word*)
Return Snowball Norwegian stem.

Parameters **word** (*str*) -- The word to stem

Returns Word stem

Return type str

Examples

```
>>> stmr = SnowballNorwegian()
>>> stmr.stem('lese')
'les'
>>> stmr.stem('suspensjon')
'suspensjon'
>>> stmr.stem('sikkerhet')
'sikker'
```

New in version 0.1.0.

Changed in version 0.3.6: Encapsulated in class

abydos.stemmer.**sb_norwegian**(*word*)
Return Snowball Norwegian stem.

This is a wrapper for *SnowballNorwegian.stem()*.

Parameters **word** (*str*) -- The word to stem

Returns Word stem

Return type str

Examples

```
>>> sb_norwegian('lese')
'les'
>>> sb_norwegian('suspensjon')
'suspensjon'
>>> sb_norwegian('sikkerhet')
'sikker'
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the SnowballNorwegian.stem method instead.

class abydos.stemmer.**SnowballSwedish**
Bases: abydos.stemmer._snowball._Snowball

Snowball Swedish stemmer.

The Snowball Swedish stemmer is defined at: http://snowball.tartarus.org/algorithms/swedish/stemmer.html

New in version 0.3.6.

stem(*word*)
Return Snowball Swedish stem.

Parameters **word** (*str*) -- The word to stem

Returns Word stem

Return type str

Examples

```
>>> stmr = SnowballSwedish()
>>> stmr.stem('undervisa')
'undervis'
>>> stmr.stem('suspension')
'suspension'
>>> stmr.stem('visshet')
'viss'
```

New in version 0.1.0.

Changed in version 0.3.6: Encapsulated in class

abydos.stemmer.**sb_swedish**(*word*)
Return Snowball Swedish stem.

This is a wrapper for *SnowballSwedish.stem()*.

Parameters **word** (*str*) -- The word to stem

Returns Word stem

Return type str

Examples

```
>>> sb_swedish('undervisa')
'undervis'
>>> sb_swedish('suspension')
'suspension'
>>> sb_swedish('visshet')
'viss'
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the SnowballSwedish.stem method instead.

class abydos.stemmer.**CLEFGerman**

Bases: abydos.stemmer._stemmer._Stemmer

CLEF German stemmer.

The CLEF German stemmer is defined at [Sav05].

New in version 0.3.6.

stem(*word*)

Return CLEF German stem.

Parameters **word** (*str*) -- The word to stem

Returns Word stem

Return type str

Examples

```
>>> stmr = CLEFGerman()
>>> stmr.stem('lesen')
'lese'
>>> stmr.stem('graues')
'grau'
>>> stmr.stem('buchstabieren')
'buchstabier'
```

New in version 0.1.0.

Changed in version 0.3.6: Encapsulated in class

abydos.stemmer.**clef_german**(*word*)

Return CLEF German stem.

This is a wrapper for *CLEFGerman.stem()*.

Parameters **word** (*str*) -- The word to stem

Returns Word stem

Return type str

Examples

```
>>> clef_german('lesen')
'lese'
>>> clef_german('graues')
'grau'
>>> clef_german('buchstabieren')
'buchstabier'
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the CLEFGerman.stem method instead.

class abydos.stemmer.**CLEFGermanPlus**

Bases: abydos.stemmer._stemmer._Stemmer

CLEF German stemmer plus.

The CLEF German stemmer plus is defined at [Sav05].

New in version 0.3.6.

stem(*word*)

Return 'CLEF German stemmer plus' stem.

Parameters **word** (*str*) -- The word to stem

Returns Word stem

Return type str

Examples

```
>>> stmr = CLEFGermanPlus()
>>> clef_german_plus('lesen')
'les'
>>> clef_german_plus('graues')
'grau'
>>> clef_german_plus('buchstabieren')
'buchstabi'
```

New in version 0.1.0.

Changed in version 0.3.6: Encapsulated in class

abydos.stemmer.**clef_german_plus**(*word*)

Return 'CLEF German stemmer plus' stem.

This is a wrapper for *CLEFGermanPlus.stem()*.

Parameters **word** (*str*) -- The word to stem

Returns Word stem

Return type str

Examples

```
>>> stmr = CLEFGermanPlus()
>>> clef_german_plus('lesen')
'les'
>>> clef_german_plus('graues')
'grau'
>>> clef_german_plus('buchstabieren')
'buchstabi'
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the CLEFGermanPlus.stem method instead.

class abydos.stemmer.**CLEFSwedish**

Bases: abydos.stemmer._stemmer._Stemmer

CLEF Swedish stemmer.

The CLEF Swedish stemmer is defined at [Sav05].

New in version 0.3.6.

stem(*word*)

Return CLEF Swedish stem.

Parameters **word** (*str*) -- The word to stem

Returns Word stem

Return type str

Examples

```
>>> clef_swedish('undervisa')
'undervis'
>>> clef_swedish('suspension')
'suspensio'
>>> clef_swedish('visshet')
'viss'
```

New in version 0.1.0.

Changed in version 0.3.6: Encapsulated in class

abydos.stemmer.**clef_swedish**(*word*)

Return CLEF Swedish stem.

This is a wrapper for *CLEFSwedish.stem()*.

Parameters **word** (*str*) -- The word to stem

Returns Word stem

Return type str

Examples

```
>>> clef_swedish('undervisa')
'undervis'
>>> clef_swedish('suspension')
'suspensio'
>>> clef_swedish('visshet')
'viss'
```

New in version 0.1.0.

Deprecated since version 0.4.0: This will be removed in 0.6.0. Use the CLEFSwedish.stem method instead.

3.1.1.9 abydos.tokenizer package

abydos.tokenizer.

The tokenizer package collects classes whose purpose is to tokenize text or individual words. Each tokenizer also supports a scaler attribute when constructed, which adjusts count scaling. The scaler defaults to None, which performs no scaling. Setting scaler to 'set' is used to convert token counters from multi-sets to sets, so even if multiple instances of a token are present, they will be counted as one. Additionally, a callable function (of one argument, such as log, exp, or lambda x: x + 1) may be passed to scaler and this function will be applied to each count value.

The following general tokenizers are provided:

- *QGrams* tokenizes a string into q-grams, substrings of length q. The class supports different values of q, the addition of start and stop symbols, and skip values. It even supports multiple values for q and skip, using lists or ranges.
- *QSkipgrams* tokenizes a string into skipgrams of length q. A skipgram is a sequence of letters from a string with q, often discontinuous, characters. For example, the string 'ABCD' has the following 2-skipgrams: 'AB', 'AC', 'AD', 'BC', 'BD', 'CD'.
- *CharacterTokenizer* tokenizes a string into individual characters.
- *RegexpTokenizer* tokenizes a string according to a supplied regular expression.
- *WhitespaceTokenizer* tokenizes a string by dividing it at instances of whitespace.
- *WordpunctTokenizer* tokenizes a string by dividing it into strings of letters and strings of punctuation.

Six syllable-oriented tokenizers are provided:

- *COrVClusterTokenizer* tokenizes a string by dividing it into strings of consonants, vowels, or other characters:
- *COrVClusterTokenizer* tokenizes a string by dividing it into strings of consonants (C* clusters), vowels (V* clusters, or non-letter characters:
- *CVClusterTokenizer* tokenizes a string by dividing it into strings of consonants then vowels (C*V* clusters) or non-letter characters:
- *VCClusterTokenizer* tokenizes a string by dividing it into strings of vowels then characters (V*C* clusters) or non-letter characters:
- *SAPSTokenizer* tokenizes a string according to the rules specified by the SAPS syllabification algorithm [RY05]:
- *SonoriPyTokenizer* does syllabification according to the sonority sequencing principle, using SyllabiPy. It requires that SyllabiPy be installed.

- *LegaliPyTokenizer* does syllabification according to the onset maximization principle (principle of legality), using SyllabiPy. It requires that SyllabiPy be installed, and works best if it has been trained on a corpus of text.

Finally, an NLTK tokenizer is provided:

- *NLTKTokenizer* does tokenization using an instantiated NLTK tokenizer. Accordingly, NLTK needs to be installed.

---

class abydos.tokenizer.**QGrams**(*qval=2*, *start_stop='$#'*, *skip=0*, *scaler=None*)
Bases: abydos.tokenizer._tokenizer._Tokenizer

A q-gram class, which functions like a bag/multiset.

A q-gram is here defined as all sequences of q characters. Q-grams are also known as k-grams and n-grams, but the term n-gram more typically refers to sequences of whitespace-delimited words in a string, where q-gram refers to sequences of characters in a word or string.

New in version 0.1.0.

Initialize QGrams.

Parameters

- **qval** (*int or Iterable*) -- The q-gram length (defaults to 2), can be an integer, range object, or list
- **start_stop** (*str*) -- A string of length >= 0 indicating start & stop symbols. If the string is '', q-grams will be calculated without start & stop symbols appended to each end. Otherwise, the first character of start_stop will pad the beginning of the string and the last character of start_stop will pad the end of the string before q-grams are calculated. (In the case that start_stop is only 1 character long, the same symbol will be used for both.)
- **skip** (*int or Iterable*) -- The number of characters to skip, can be an integer, range object, or list
- **scaler** (*None, str, or function*) -- A scaling function for the Counter:
  - None : no scaling
  - 'set' : All non-zero values are set to 1.
  - 'length' : Each token has weight equal to its length.
  - **'length-log'** [Each token has weight equal to the log of its] length + 1.
  - **'length-exp'** [Each token has weight equal to e raised to its] length.
  - a callable function : The function is applied to each value in the Counter. Some useful functions include math.exp, math.log1p, math.sqrt, and indexes into interesting integer sequences such as the Fibonacci sequence.

Raises **ValueError** -- Use WhitespaceTokenizer instead of qval=0.

Examples

```
>>> qg = QGrams().tokenize('AATTATAT')
>>> qg
QGrams({'AT': 3, 'TA': 2, '$A': 1, 'AA': 1, 'TT': 1, 'T#': 1})
```

```
>>> qg = QGrams(qval=1, start_stop='').tokenize('AATTATAT')
>>> qg
QGrams({'A': 4, 'T': 4})
```

```
>>> qg = QGrams(qval=3, start_stop='').tokenize('AATTATAT')
>>> qg
QGrams({'TAT': 2, 'AAT': 1, 'ATT': 1, 'TTA': 1, 'ATA': 1})
```

```
>>> QGrams(qval=2, start_stop='$#').tokenize('interning')
QGrams({'in': 2, '$i': 1, 'nt': 1, 'te': 1, 'er': 1, 'rn': 1,
'ni': 1, 'ng': 1, 'g#': 1})
```

```
>>> QGrams(start_stop='', skip=1).tokenize('AACTAGAAC')
QGrams({'AC': 2, 'AT': 1, 'CA': 1, 'TG': 1, 'AA': 1, 'GA': 1, 'A': 1})
```

```
>>> QGrams(start_stop='', skip=[0, 1]).tokenize('AACTAGAAC')
QGrams({'AC': 4, 'AA': 3, 'GA': 2, 'CT': 1, 'TA': 1, 'AG': 1,
'AT': 1, 'CA': 1, 'TG': 1, 'A': 1})
```

```
>>> QGrams(qval=range(3), skip=[0, 1]).tokenize('interdisciplinarian')
QGrams({'i': 10, 'n': 7, 'r': 4, 'a': 4, 'in': 3, 't': 2, 'e': 2,
'd': 2, 's': 2, 'c': 2, 'p': 2, 'l': 2, 'ri': 2, 'ia': 2, '$i': 1,
'nt': 1, 'te': 1, 'er': 1, 'rd': 1, 'di': 1, 'is': 1, 'sc': 1, 'ci': 1,
'ip': 1, 'pl': 1, 'li': 1, 'na': 1, 'ar': 1, 'an': 1, 'n#': 1, '$n': 1,
'it': 1, 'ne': 1, 'tr': 1, 'ed': 1, 'ds': 1, 'ic': 1, 'si': 1, 'cp': 1,
'il': 1, 'pi': 1, 'ln': 1, 'nr': 1, 'ai': 1, 'ra': 1, 'a#': 1})
```

New in version 0.1.0.

Changed in version 0.4.0: Broke tokenization functions out into tokenize method

tokenize(*string*)

Tokenize the term and store it.

The tokenized term is stored as an ordered list and as a Counter object.

Parameters **string** (*str*) -- The string to tokenize

New in version 0.4.0.

class abydos.tokenizer.**QSkipgrams**(*qval=2*, *start_stop='$#'*, *scaler=None*, *ssk_lambda=0.9*)

Bases: abydos.tokenizer._tokenizer._Tokenizer

A q-skipgram class, which functions like a bag/multiset.

A q-gram is here defined as all sequences of q characters. Q-grams are also known as k-grams and n-grams, but the term n-gram more typically refers to sequences of whitespace-delimited words in a string, where q-gram refers to sequences of characters in a word or string.

New in version 0.4.0.

Initialize QSkipgrams.

Parameters

- **qval** (*int or Iterable*) -- The q-gram length (defaults to 2), can be an integer, range object, or list
- **start_stop** (*str*) -- A string of length >= 0 indicating start & stop symbols. If the string is '', q-grams will be calculated without start & stop symbols appended to each end. Otherwise, the first character of start_stop will pad the beginning of the string and the last character of start_stop will pad the end of the string before q-grams are calculated. (In the case that start_stop is only 1 character long, the same symbol will be used for both.)
- **scaler** (*None, str, or function*) -- A scaling function for the Counter:
  - None : no scaling
  - 'set' : All non-zero values are set to 1.
  - 'length' : Each token has weight equal to its length.
  - **'length-log'** [Each token has weight equal to the log of its] length + 1.
  - **'length-exp'** [Each token has weight equal to e raised to its] length.
  - a callable function : The function is applied to each value in the Counter. Some useful functions include math.exp, math.log1p, math.sqrt, and indexes into interesting integer sequences such as the Fibonacci sequence.
  - 'SSK' : Applies weighting according to the substring kernel rules of [LSShaweTaylor+02].
- **ssk_lambda** (*float or Iterable*) -- A value in the range (0.0, 1.0) used for discouting gaps between characters according to the method described in [LSShaweTaylor+02]. To supply multiple values of lambda, provide an Iterable of numeric values, such as (0.5, 0.05) or np.arange(0.05, 0.5, 0.05)

Raises **ValueError** -- Use WhitespaceTokenizer instead of qval=0.

Examples

```
>>> QSkipgrams().tokenize('AATTAT')
QSkipgrams({'AT': 7, '$A': 3, '$T': 3, 'AA': 3, 'A#': 3, 'TT': 3,
'T#': 3, 'TA': 2, '$#': 1})
```

```
>>> QSkipgrams(qval=1, start_stop='').tokenize('AATTAT')
QSkipgrams({'A': 3, 'T': 3})
```

```
>>> QSkipgrams(qval=3, start_stop='').tokenize('AATTAT')
QSkipgrams({'ATT': 6, 'AAT': 5, 'ATA': 4, 'TAT': 2, 'AAA': 1,
'TTA': 1, 'TTT': 1})
```

```
>>> QSkipgrams(start_stop='').tokenize('ABCD')
QSkipgrams({'AB': 1, 'AC': 1, 'AD': 1, 'BC': 1, 'BD': 1, 'CD': 1})
```

```
>>> QSkipgrams().tokenize('Colin')
QSkipgrams({'$C': 1, '$o': 1, '$l': 1, '$i': 1, '$n': 1, '$#': 1,
'Co': 1, 'Cl': 1, 'Ci': 1, 'Cn': 1, 'C#': 1, 'ol': 1, 'oi': 1, 'on': 1,
'o#': 1, 'li': 1, 'ln': 1, 'l#': 1, 'in': 1, 'i#': 1, 'n#': 1})
```

```
>>> QSkipgrams(qval=3).tokenize('AACTAGAAC')
QSkipgrams({'$AA': 20, '$A#': 20, 'AA#': 20, '$AC': 14, 'AC#': 14,
'AAC': 11, 'AAA': 10, '$C#': 8, '$AG': 6, '$CA': 6, '$TA': 6, 'ACA': 6,
'ATA': 6, 'AGA': 6, 'AG#': 6, 'CA#': 6, 'TA#': 6, '$$A': 5, 'A##': 5,
'$AT': 4, '$T#': 4, '$GA': 4, '$G#': 4, 'AT#': 4, 'GA#': 4, 'AAG': 3,
'AGC': 3, 'CTA': 3, 'CAA': 3, 'CAC': 3, 'TAA': 3, 'TAC': 3, '$$C': 2,
'$$#': 2, '$CT': 2, '$CG': 2, '$CC': 2, '$TG': 2, '$TC': 2, '$GC': 2,
'$##': 2, 'ACT': 2, 'ACG': 2, 'ACC': 2, 'ATG': 2, 'ATC': 2, 'CT#': 2,
'CGA': 2, 'CG#': 2, 'CC#': 2, 'C##': 2, 'TGA': 2, 'TG#': 2, 'TC#': 2,
'GAC': 2, 'GC#': 2, '$$T': 1, '$$G': 1, 'AAT': 1, 'CTG': 1, 'CTC': 1,
'CAG': 1, 'CGC': 1, 'TAG': 1, 'TGC': 1, 'T##': 1, 'GAA': 1, 'G##': 1})
```

QSkipgrams may also be used to produce weights in accordance with the substring kernel rules of [LSShaweTaylor+02] by passing the scaler value `'SSK'`:

```
>>> QSkipgrams(scaler='SSK').tokenize('AACTAGAAC')
QSkipgrams({'AA': 6.170192010000001, 'AC': 4.486377699,
'$A': 2.8883286990000006, 'A#': 2.6526399291000002, 'TA': 2.05659,
'AG': 1.931931, 'CA': 1.850931, 'GA': 1.5390000000000001, 'AT': 1.3851,
'C#': 1.2404672100000003, '$C': 1.0047784401000002, 'CT': 0.81,
'TG': 0.7290000000000001, 'CG': 0.6561, 'GC': 0.6561,
'$T': 0.5904900000000001, 'G#': 0.5904900000000001, 'TC': 0.531441,
'$G': 0.4782969000000001, 'CC': 0.4782969000000001,
'T#': 0.4782969000000001, '$#': 0.31381059609000006})
```

New in version 0.4.0.

tokenize(*string*)

Tokenize the term and store it.

The tokenized term is stored as an ordered list and as a Counter object.

Parameters **string** (*str*) -- The string to tokenize

New in version 0.4.0.

class abydos.tokenizer.**CharacterTokenizer**(*scaler=None*)

Bases: abydos.tokenizer._tokenizer._Tokenizer

A character tokenizer.

New in version 0.4.0.

Initialize tokenizer.

Parameters **scaler** (*None, str, or function*) -- A scaling function for the Counter:

- None : no scaling
- 'set' : All non-zero values are set to 1.
- a callable function : The function is applied to each value in the Counter. Some useful functions include math.exp, math.log1p, math.sqrt, and indexes into interesting integer sequences such as the Fibonacci sequence.

New in version 0.4.0.

tokenize(*string*)

Tokenize the term and store it.

The tokenized term is stored as an ordered list and as a Counter object.

Parameters **string** (*str*) -- The string to tokenize

Examples

```
>>> CharacterTokenizer().tokenize('AACTAGAAC')
CharacterTokenizer({'A': 5, 'C': 2, 'T': 1, 'G': 1})
```

New in version 0.4.0.

class abydos.tokenizer.**RegexpTokenizer**(*scaler=None*, *regexp='\w+'*, *flags=0*)

Bases: abydos.tokenizer._tokenizer._Tokenizer

A regexp tokenizer.

New in version 0.4.0.

Initialize tokenizer.

Parameters **scaler** (*None, str, or function*) -- A scaling function for the Counter:

- None : no scaling
- 'set' : All non-zero values are set to 1.
- 'length' : Each token has weight equal to its length.
- **'length-log'** [Each token has weight equal to the log of its] length + 1.
- **'length-exp'** [Each token has weight equal to e raised to its] length.
- a callable function : The function is applied to each value in the Counter. Some useful functions include math.exp, math.log1p, math.sqrt, and indexes into interesting integer sequences such as the Fibonacci sequence.

New in version 0.4.0.

tokenize(*string*)

Tokenize the term and store it.

The tokenized term is stored as an ordered list and as a Counter object.

Parameters **string** (*str*) -- The string to tokenize

Examples

```
>>> RegexpTokenizer(regexp=r'[^-]+').tokenize('AA-CT-AG-AA-CD')
RegexpTokenizer({'AA': 2, 'CT': 1, 'AG': 1, 'CD': 1})
```

New in version 0.4.0.

class abydos.tokenizer.**WhitespaceTokenizer**(*scaler=None*, *flags=0*)

Bases: abydos.tokenizer._regexp.RegexpTokenizer

A whitespace tokenizer.

Examples

```
>>> WhitespaceTokenizer().tokenize('a b c f a c g e a b')
WhitespaceTokenizer({'a': 3, 'b': 2, 'c': 2, 'f': 1, 'g': 1, 'e': 1})
```

New in version 0.4.0.

Initialize tokenizer.

Parameters **scaler** (*None, str, or function*) -- A scaling function for the Counter:

- None : no scaling
- 'set' : All non-zero values are set to 1.
- 'length' : Each token has weight equal to its length.
- **'length-log'** [Each token has weight equal to the log of its] length + 1.
- **'length-exp'** [Each token has weight equal to e raised to its] length.
- a callable function : The function is applied to each value in the Counter. Some useful functions include math.exp, math.log1p, math.sqrt, and indexes into interesting integer sequences such as the Fibonacci sequence.

New in version 0.4.0.

class abydos.tokenizer.**WordpunctTokenizer**(*scaler=None*, *flags=0*)

Bases: abydos.tokenizer._regexp.RegexpTokenizer

A wordpunct tokenizer.

Examples

```
>>> WordpunctTokenizer().tokenize("Can't stop the feelin'!")
WordpunctTokenizer({'Can': 1, "'": 1, 't': 1, 'stop': 1, 'the': 1,
'feelin': 1, "'!": 1})
```

New in version 0.4.0.

Initialize tokenizer.

Parameters **scaler** (*None, str, or function*) -- A scaling function for the Counter:

- None : no scaling
- 'set' : All non-zero values are set to 1.
- 'length' : Each token has weight equal to its length.
- **'length-log'** [Each token has weight equal to the log of its] length + 1.
- **'length-exp'** [Each token has weight equal to e raised to its] length.
- a callable function : The function is applied to each value in the Counter. Some useful functions include math.exp, math.log1p, math.sqrt, and indexes into interesting integer sequences such as the Fibonacci sequence.

New in version 0.4.0.

class abydos.tokenizer.**COrVClusterTokenizer**(*scaler=None*, *consonants=None*, *vowels=None*)

Bases: abydos.tokenizer._tokenizer._Tokenizer

A C- or V-cluster tokenizer.

New in version 0.4.0.

Initialize tokenizer.

Parameters **scaler** (*None, str, or function*) -- A scaling function for the Counter:

- None : no scaling
- 'set' : All non-zero values are set to 1.
- 'length' : Each token has weight equal to its length.
- **'length-log'** [Each token has weight equal to the log of its] length + 1.
- **'length-exp'** [Each token has weight equal to e raised to its] length.
- a callable function : The function is applied to each value in the Counter. Some useful functions include math.exp, math.log1p, math.sqrt, and indexes into interesting integer sequences such as the Fibonacci sequence.

New in version 0.4.0.

tokenize(*string*)

Tokenize the term and store it.

The tokenized term is stored as an ordered list and as a Counter object.

Parameters **string** (*str*) -- The string to tokenize

Examples

```
>>> COrVClusterTokenizer().tokenize('seven-twelfths')
COrVClusterTokenizer({'e': 3, 's': 1, 'v': 1, 'n': 1, '-': 1,
'tw': 1, 'lfths': 1})
```

```
>>> COrVClusterTokenizer().tokenize('character')
COrVClusterTokenizer({'a': 2, 'r': 2, 'ch': 1, 'ct': 1, 'e': 1})
```

New in version 0.4.0.

class abydos.tokenizer.**CVClusterTokenizer**(*scaler=None, consonants=None, vowels=None*)

Bases: abydos.tokenizer._tokenizer._Tokenizer

A C*V*-cluster tokenizer.

New in version 0.4.0.

Initialize tokenizer.

Parameters **scaler** (*None, str, or function*) -- A scaling function for the Counter:

- None : no scaling
- 'set' : All non-zero values are set to 1.
- 'length' : Each token has weight equal to its length.
- **'length-log'** [Each token has weight equal to the log of its] length + 1.
- **'length-exp'** [Each token has weight equal to e raised to its] length.

- a callable function : The function is applied to each value in the Counter. Some useful functions include math.exp, math.log1p, math.sqrt, and indexes into interesting integer sequences such as the Fibonacci sequence.

New in version 0.4.0.

tokenize(*string*)

Tokenize the term and store it.

The tokenized term is stored as an ordered list and as a Counter object.

Parameters **string** (*str*) -- The string to tokenize

Examples

```
>>> CVClusterTokenizer().tokenize('seven-twelfths')
CVClusterTokenizer({'se': 1, 've': 1, 'n': 1, '-': 1, 'twe': 1,
'lfths': 1})
```

```
>>> CVClusterTokenizer().tokenize('character')
CVClusterTokenizer({'cha': 1, 'ra': 1, 'cte': 1, 'r': 1})
```

New in version 0.4.0.

class abydos.tokenizer.**VCClusterTokenizer**(*scaler=None*, *consonants=None*, *vowels=None*)

Bases: abydos.tokenizer._tokenizer._Tokenizer

A V*C*-cluster tokenizer.

New in version 0.4.0.

Initialize tokenizer.

Parameters **scaler** (*None, str, or function*) -- A scaling function for the Counter:

- None : no scaling
- 'set' : All non-zero values are set to 1.
- 'length' : Each token has weight equal to its length.
- **'length-log'** [Each token has weight equal to the log of its] length + 1.
- **'length-exp'** [Each token has weight equal to e raised to its] length.
- a callable function : The function is applied to each value in the Counter. Some useful functions include math.exp, math.log1p, math.sqrt, and indexes into interesting integer sequences such as the Fibonacci sequence.

New in version 0.4.0.

tokenize(*string*)

Tokenize the term and store it.

The tokenized term is stored as an ordered list and as a Counter object.

Parameters **string** (*str*) -- The string to tokenize

Examples

```
>>> VCClusterTokenizer().tokenize('seven-twelfths')
VCClusterTokenizer({'s': 1, 'ev': 1, 'en': 1, '-': 1, 'tw': 1,
'elfths': 1})
```

```
>>> VCClusterTokenizer().tokenize('character')
VCClusterTokenizer({'ch': 1, 'ar': 1, 'act': 1, 'er': 1})
```

New in version 0.4.0.

class abydos.tokenizer.**SAPSTokenizer**(*scaler=None*)

Bases: abydos.tokenizer._tokenizer._Tokenizer

Syllable Alignment Pattern Searching tokenizer.

This is the syllabifier described on p. 917 of [RY05].

New in version 0.4.0.

Initialize Tokenizer.

Parameters **scaler** (*None, str, or function*) -- A scaling function for the Counter:

- None : no scaling
- 'set' : All non-zero values are set to 1.
- 'length' : Each token has weight equal to its length.
- **'length-log'** [Each token has weight equal to the log of its] length + 1.
- **'length-exp'** [Each token has weight equal to e raised to its] length.
- a callable function : The function is applied to each value in the Counter. Some useful functions include math.exp, math.log1p, math.sqrt, and indexes into interesting integer sequences such as the Fibonacci sequence.

New in version 0.4.0.

tokenize(*string*)

Tokenize the term and store it.

The tokenized term is stored as an ordered list and as a Counter object.

Parameters **string** (*str*) -- The string to tokenize

Examples

```
>>> SAPSTokenizer().tokenize('seven-twelfths')
SAPSTokenizer({'t': 2, 'se': 1, 'ven': 1, '-': 1, 'wel': 1, 'f': 1,
'h': 1, 's': 1})
```

```
>>> SAPSTokenizer().tokenize('character')
SAPSTokenizer({'c': 1, 'ha': 1, 'rac': 1, 'ter': 1})
```

New in version 0.4.0.

class abydos.tokenizer.**SonoriPyTokenizer**(*scaler=None*)

Bases: abydos.tokenizer._tokenizer._Tokenizer

SonoriPy tokenizer.

New in version 0.4.0.

Initialize Tokenizer.

Parameters **scaler** (*None, str, or function*) -- A scaling function for the Counter:

- None : no scaling
- 'set' : All non-zero values are set to 1.
- 'length' : Each token has weight equal to its length.
- **'length-log'** [Each token has weight equal to the log of its] length + 1.
- **'length-exp'** [Each token has weight equal to e raised to its] length.
- a callable function : The function is applied to each value in the Counter. Some useful functions include math.exp, math.log1p, math.sqrt, and indexes into interesting integer sequences such as the Fibonacci sequence.

New in version 0.4.0.

tokenize(*string*)

Tokenize the term and store it.

The tokenized term is stored as an ordered list and as a Counter object.

Parameters **string** (*str*) -- The string to tokenize

Examples

```
>>> SonoriPyTokenizer().tokenize('seven-twelfths')
SonoriPyTokenizer({'se': 1, 'ven-': 1, 'twelfths': 1})
```

```
>>> SonoriPyTokenizer().tokenize('character')
SonoriPyTokenizer({'cha': 1, 'rac': 1, 'ter': 1})
```

New in version 0.4.0.

class abydos.tokenizer.**LegaliPyTokenizer**(*scaler=None*)

Bases: abydos.tokenizer._tokenizer._Tokenizer

LegaliPy tokenizer.

New in version 0.4.0.

Initialize Tokenizer.

Parameters **scaler** (*None, str, or function*) -- A scaling function for the Counter:

- None : no scaling
- 'set' : All non-zero values are set to 1.
- 'length' : Each token has weight equal to its length.
- **'length-log'** [Each token has weight equal to the log of its] length + 1.
- **'length-exp'** [Each token has weight equal to e raised to its] length.
- a callable function : The function is applied to each value in the Counter. Some useful functions include math.exp, math.log1p, math.sqrt, and indexes into interesting integer sequences such as the Fibonacci sequence.

New in version 0.4.0.

tokenize(*string*, *ipa=False*)
Tokenize the term and store it.

The tokenized term is stored as an ordered list and as a Counter object.

Parameters

- **string** (*str*) -- The string to tokenize
- **ipa** (*bool*) -- If True, indicates that the string is in IPA

Examples

```
>>> LegaliPyTokenizer().tokenize('seven-twelfths')
LegaliPyTokenizer({'s': 1, 'ev': 1, 'en-tw': 1, 'elfths': 1})
```

```
>>> LegaliPyTokenizer().tokenize('character')
LegaliPyTokenizer({'ch': 1, 'ar': 1, 'act': 1, 'er': 1})
```

New in version 0.4.0.

train_onsets(*text*, *threshold=0.0002*, *clean=True*, *append=False*)
Train the onsets on a text.

Parameters

- **text** (*str*) -- The text on which to train
- **threshold** (*float*) -- Threshold proportion above which to include onset into onset list
- **clean** (*bool*) -- If True, the text is stripped of numerals and punctuation
- **append** (*bool*) -- If True, the current onset list is extended

New in version 0.4.0.

class abydos.tokenizer.**NLTKTokenizer**(*nltk_tokenizer=None*, *scaler=None*)
Bases: abydos.tokenizer._tokenizer._Tokenizer

NLTK tokenizer wrapper class.

New in version 0.4.0.

Initialize Tokenizer.

Parameters

- **scaler** (*None, str, or function*) -- A scaling function for the Counter:
  - None : no scaling
  - 'set' : All non-zero values are set to 1.
  - 'length' : Each token has weight equal to its length.
  - **'length-log'** [Each token has weight equal to the log of its] length + 1.
  - **'length-exp'** [Each token has weight equal to e raised to its] length.
  - a callable function : The function is applied to each value in the Counter. Some useful functions include math.exp, math.log1p, math.sqrt, and indexes into interesting integer sequences such as the Fibonacci sequence.
- **nltk_tokenizer** (*Object*) -- An instantiated tokenizer from NLTK.

New in version 0.4.0.

tokenize(*string*)

Tokenize the term and store it.

The tokenized term is stored as an ordered list and as a Counter object.

Parameters **string** (*str*) -- The string to tokenize

Examples

```
>>> from nltk.tokenize.casual import TweetTokenizer
>>> nltk_tok = TweetTokenizer()
>>> NLTKTokenizer(nltk_tokenizer=nltk_tok).tokenize(
... '.@Twitter Today is #lit!')
NLTKTokenizer({'.': 1, '@Twitter': 1, 'Today': 1, 'is': 1, '#lit': 1,
'!': 1})
```

New in version 0.4.0.

3.1.1.10 abydos.util package

abydos.util.

The util module defines various utility functions for other modules within Abydos, including:

- _prod -- computes the product of a collection of numbers (akin to sum)

These functions are not intended for use by users.

abydos.util.**download_package**(*resource_name*, *url=None*, *data_path=None*, *force=False*, *silent=False*)

Download and install a package or collection.

abydos.util.**list_available_packages**(*url=None*)

List all data packages available for install.

abydos.util.**list_installed_packages**(*path=None*)

List all installed data packages.

abydos.util.**package_path**(*resource_name*)

Given a resource name, returns the path to the package.

CHAPTER

FOUR

RELEASE HISTORY

4.1 0.5.0 (2020-01-10) *ecgtheow*

doi:10.5281/zenodo.3603514

Changes:

- Support for Python 2.7 was removed.

4.2 0.4.1 (2020-01-07) *distant dietrich*

doi:10.5281/zenodo.3600548

Changes:

- Support for Python 3.4 was removed. (3.4 reached end-of-life on March 18, 2019)
- Fuzzy intersections were corrected to avoid over-counting partial intersection instances.
- Levenshtein can now return an optimal alignment
- **Added the following distance measures:**
  - Indice de Similitude-Guth (ISG)
  - INClusion Programme
  - Guth
  - Victorian Panel Study (VPS) score
  - LIG3 similarity
  - Discounted Levenshtein
  - Relaxed Hamming
  - String subsequence kernel (SSK) similarity
  - Phonetic edit distance
  - Henderson-Heron dissimilarity
  - Raup-Crick similarity
  - Millar's binomial deviance dissimilarity
  - Morisita similarity
  - Horn-Morisita similarity

  - Clark's coefficient of divergence
  - Chao's Jaccard similarity
  - Chao's Dice similarity
  - Cao's CY similarity (CYs) and dissimilarity (CYd)
- **Added the following fingerprint classes:**
  - Taft's Consonant coding
  - Taft's Extract - letter list
  - Taft's Extract - position & frequency
  - L.A. County Sheriff's System
  - Library of Congres Cutter table encoding
- **Added the following phonetic algorithms:**
  - Ainsworth's grapheme-to-phoneme
  - PHONIC

4.3 0.4.0 (2019-05-30) *dietrich*

doi:10.5281/zenodo.3235034

Version 0.4.0 focuses on distance measures, adding 211 new measures. Attempts were made to provide normalized version for measure that did not inherently range from 0 to 1. The other major focus was the addition of 12 tokenizers, in service of expanding distance measure options.

Changes:

- Support for Python 3.3 was dropped.
- Deprecated functions that merely wrap class methods to maintain API compatibility, for removal in 0.6.0
- **Added methods to ConfusionTable to return:**
  - its internal representation
  - false negative rate
  - false omission rate
  - positive & negative likelihood ratios
  - diagnostic odds ratio
  - error rate
  - prevalence
  - Jaccard index
  - D-measure
  - Phi coefficient
  - joint, actual, & predicted entropies
  - mutual information
  - proficiency (uncertainty coefficient)

    - information gain ratio
    - dependency
    - lift
- Deprecated f-measure & g-measure from ConfusionTable for removal in 0.6.0
- Added notes to indicate when functions, classes, & methods were added
- **Added the following 12 tokenizers:**
    - QSkipgrams
    - CharacterTokenizer
    - RegexpTokenizer, WhitespaceTokenizer, & WordpunctTokenizer
    - COrVClusterTokenizer, CVClusterTokenizer, & VCClusterTokenizer
    - SonoriPyTokenizer & LegaliPyTokenizer
    - NLTKTokenizer
    - SAPSTokenizer
- Added the UnigramCorpus class & a facility for downloading data, such as pre-processed/trained data, from storage on GitHub
- Added the Wåhlin phonetic encoding
- **Added the following 211 similarity/distance/correlation measures:**
    - ALINE
    - AMPLE
    - Anderberg
    - Andres & Marzo's Delta
    - Average Linkage
    - AZZOO
    - Baroni-Urbani & Buser I & II
    - Batagelj & Bren
    - Baulieu I-XV
    - Benini I & II
    - Bennet
    - Bhattacharyya
    - BI-SIM
    - BLEU
    - Block Levenshtein
    - Brainerd-Robinson
    - Braun-Blanquet
    - Canberra
    - Chord

- Clement
- Cohen's Kappa
- Cole
- Complete Linkage
- Consonni & Todeschini I-V
- Cormode's LZ
- Covington
- Dennis
- Dice Asymmetric I & II
- Digby
- Dispersion
- Doolittle
- Dunning
- Eyraud
- Fager & McGowan
- Faith
- Fellegi-Sunter
- Fidelity
- Fleiss
- Fleiss-Levin-Paik
- FlexMetric
- Forbes I & II
- Fossum
- FuzzyWuzzy Partial String
- FuzzyWuzzy Token Set
- FuzzyWuzzy Token Sort
- Generalized Fleiss
- Gilbert
- Gilbert & Wells
- Gini I & II
- Goodall
- Goodman & Kruskal's Lambda
- Goodman & Kruskal's Lambda-r
- Goodman & Kruskal's Tau A & B
- Gower & Legendre
- Guttman's Lambda A & B

- Gwet's AC
- Hamann
- Harris & Lahey
- Hassanat
- Hawkins & Dotson
- Hellinger
- Higuera & Mico
- Hurlbert
- Iterative SubString
- Jaccard-NM
- Jensen-Shannon
- Johnson
- Kendall's Tau
- Kent & Foster I & II
- Koppen I & II
- Kuder & Richardson
- Kuhns I-XII
- Kulczynski I & II
- Longest Common Prefix
- Longest Common Suffix
- Lorentzian
- Maarel
- Marking
- Marking Metric
- MASI
- Matusita
- Maxwell & Pilliner
- McConnaughey
- McEwen & Michael
- MetaLevenshtein
- Michelet
- MinHash
- Mountford
- Mean Squared Contingency
- Mutual Information
- NCD with LZSS

- NCD with PAQ9a
- Ozbay
- Pattern
- Pearson's Chi-Squared
- Pearson & Heron II
- Pearson II & III
- Pearson's Phi
- Peirce
- Positional Q-Gram Dice, Jaccard, & Overlap
- Q-Gram
- Quantitative Cosine, Dice, & Jaccard
- Rees-Levenshtein
- Roberts
- Rogers & Tanimoto
- Rogot & Goldberg
- Rouge-L, -S, -SU, & -W
- Russell & Rao
- SAPS
- Scott's Pi
- Shape
- Shapira & Storer I
- Sift4 Extended
- Single Linkage
- Size
- Soft Cosine
- SoftTF-IDF
- Sokal & Michener
- Sokal & Sneath I-V
- Sorgenfrei
- Steffensen
- Stiles
- Stuart's Tau
- Tarantula
- Tarwid
- Tetrachoric
- TF-IDF

  - Tichy
  - Tulloss's R, S, T, & U
  - Unigram Subtuple
  - Unknown A-M
  - Upholt
  - Warrens I-V
  - Weighted Jaccard
  - Whittaker
  - Yates' Chi-Squared
  - YJHHR
  - Yujian & Bo
  - Yule's Q, Q II, & Y
- Four intersection types are now supported for all distance measure that are based on _TokenDistance. In addition to basic crisp intersections, soft, fuzzy, and group linkage intersections have been provided.

4.4 0.3.6 (2018-11-17) *classy carl*

doi:10.5281/zenodo.1490537

Changes:

- Most functions were encapsulated into classes.
- Each class is broken out into its own file, with test files paralleling library files.
- Documentation was converted from Sphinx markup to Numpy style.
- A tutorial was written for each subpackage.
- Documentation was cleaned up, with math markup corrections and many additional links.

4.5 0.3.5 (2018-10-31) *cantankerous carl*

doi:10.5281/zenodo.1463204

Version 0.3.5 focuses on refactoring the whole project. The API itself remains largely the same as in previous versions, but underlyingly modules have been split up. Essentially no new features are added (bugfixes aside) in this version.

Changes:

- Refactored library and tests into smaller modules
- Broke compression distances (NCD) out into separate functions
- Adopted Black code style
- Added pyproject.toml to use Poetry for packaging (but will continue using setuptools and setup.py for the present)
- Minor bug fixes

4.6 0.3.0 (2018-10-15) *carl*

doi:10.5281/zenodo.1462443

Version 0.3.0 focuses on additional phonetic algorithms, but does add numerous distance measures, fingerprints, and even a few stemmers. Another focus was getting everything to build again (including docs) and to move to more standard modern tools (flake8, tox, etc.).

Changes:

- Fixed implementation of Bag distance
- Updated BMPM to version 3.10
- Fixed Sphinx documentation on readthedocs.org
- Split string fingerprints out of clustering into their own module
- Added support for q-grams to skip-n characters
- **New phonetic algorithms:**
  - Statistics Canada
  - Lein
  - Roger Root
  - Oxford Name Compression Algorithm (ONCA)
  - Eudex phonetic hash
  - Haase Phonetik
  - Reth-Schek Phonetik
  - FONEM
  - Parmar-Kumbharana
  - Davidson's Consonant Code
  - SoundD
  - PSHP Soundex/Viewex Coding
  - an early version of Henry Code
  - Norphone
  - Dolby Code
  - Phonetic Spanish
  - Spanish Metaphone
  - MetaSoundex
  - SoundexBR
  - NRL English-to-phoneme
- **New string fingerprints:**
  - Cisłak & Grabowski's occurrence fingerprint
  - Cisłak & Grabowski's occurrence halved fingerprint
  - Cisłak & Grabowski's count fingerprint

    - Cisłak & Grabowski's position fingerprint
    - Synoname Toolcode
- **New distance measures:**
    - Minkowski distance & similarity
    - Manhattan distance & similarity
    - Euclidean distance & similarity
    - Chebyshev distance & similarity
    - Eudex distances
    - Sift4 distance
    - Baystat distance & similarity
    - Typo distance
    - Indel distance
    - Synoname
- **New stemmers:**
    - UEA-Lite Stemmer
    - Paice-Husk Stemmer
    - Schinke Latin stemmer
    - S stemmer
- Eliminated ._compat submodule in favor of six
- Transitioned from PEP8 to flake8, etc.
- Phonetic algorithms now consistently use max_length=-1 to indicate that there should be no length limit
- Added example notebooks in binder directory

4.7 0.2.0 (2015-05-27) *berthold*

- Added Caumanns' German stemmer
- Added Lovins' English stemmer
- Updated Beider-Morse Phonetic Matching to 3.04
- Added Sphinx documentation

4.8 0.1.1 (2015-05-12) *albrecht*

- First Beta release to PyPI

CHAPTER

FIVE

INDICES

- genindex
- modindex
- search

BIBLIOGRAPHY

[AZvanGemund07] Rui Abreu, Peter Zoeteweij, and Arjan J. C. van Gemund. An evaluation of similarity coefficients for software fault localization. In *2006 12th Pacific Rim International Symposium on Dependable Computing (PRDC'06)*. 2007. doi:10.1109/PRDC.2006.18.

[Ada17] Jason Adams. Ruby port of uealite stemmer. 2017. URL: https://github.com/ealdent/uea-stemmer.

[Ain73] William A. Ainsworth. A system for converting text into speech. *IEEE Transactions on Audio and Electroacoustics*, AU-21(3):288–290, June 1973. doi:10.1109/TAU.1973.1162452.

[AmonME12] Iván Amón, Francisco Moreno, and Jaime Echeverri. Algoritmo fonético para detección de cadenas de texto duplicadas en el idioma español. *Revista Ingenier\'ıas Universidad de Medell\'ın*, 11(20):127–138, June 2012. URL: http://www.scielo.org.co/scielo.php?pid=S1692-33242012000100011\T1\textbackslash{}&script=sci\T1\textbackslash{}_abstract\T1\textbackslash{}&tlng=es.

[And73] Michael R. Anderberg. *Cluster Analysis for Applications*. Academic Press, New York, 1973. doi:10.1016/C2013-0-06161-0.

[AM04] Marti J. Anderson and Russell B. Millar. Spatial variation and effects of habitat on temperate reef fish assemblages in northeastern new zealand. *Journal of Experimental Marine Biology and Ecology*, 305:191–221, 2004. doi:10.1016/j.jembe.2003.12.011.

[AndresM04] A. Martín Andrés and P. Femia Marzo. Delta: a new measure of agreement between two raters. *British Journal of Mathematical and Statistical Psychology*, 57(1):1–20, May 2004. doi:10.1348/000711004849268.

[AC77] Brian Austin and Rita R. Colwell. Evaluation of some coefficients for use in numerical taxonomy of microorganisms. *International Journal of Systematic Bacteriology*, 27(3):204–210, July 1977. doi:10.1099/00207713-27-3-204.

[Axe09] Pål Axelsson. Sfinxbis. Technical Report, Swedish Alliance for Middleware Infrastructure, April 2009. URL: http://www.swami.se/download/18.248ad5af12aa8136533800091/SfinxBis.pdf.

[BUB76] Cesare Baroni-Urbani and Mauro W. Buser. Similarity of binary data. *Systematic Biology*, 25(3):251–259, September 1976. doi:10.2307/2412493.

[BCP02] Ilaria Bartolini, Paolo Ciaccia, and Marco Patella. String matching with metric trees using an approximate distance. In Alberto H. F. Laender and Arlindo L. Oliveira, editors, *SPIRE 2002: String Processing and Information Retrieval*, 271–283. Berlin, Heidelberg, 2002. Springer Berlin Heidelberg. URL: http://www-db.disi.unibo.it/research/papers/SPIRE02.pdf, doi:10.1007/3-540-45735-6_24.

[BB95] Vladimir Batagelj and Matevž Bren. Comparing resemblance measures. *Journal of Classification*, 12(1):73–90, March 1995. doi:10.1007/BF01202268.

[Bau89] Forrest B. Baulieu. A classification of presence/absence based dissimilarity coefficients. *Journal of Classification*, 6(1):233–246, 1989. doi:10.1007/BF01908601.

[Bau97] Forrest B. Baulieu. Two variant axiom systems for presence/absence based dissimilarity coefficients. *Journal of Classification*, 14(1):159–170, 1997. doi:10.1007/s003579900009.

[BM08] Alexander Beider and Stephen P. Morse. Beider-morse phonetic matching: an alternative to soundex with fewer false hits. *International Review of Jewish Genealogy*, Summer 2008. URL: https://stevemorse.org/phonetics/bmpm.htm.

[Ben01] Rudolfo Benini. *Principii di Demografia*. Number 29 in Manuali Barbera di Scienze Giuridiche Sociali e Politiche. G. Barbera, Firenze, 1901. URL: http://www.archive.org/stream/principiididemo00benigoog.

[BAG54] E. M. Bennet, R. Alpert, and A. C. Goldstein. Communications through limited-response questioning. *Public Opinion Quarterly*, 18(3):303–308, 1954. doi:10.1086/266520.

[Bha46] Anil Kumar Bhattacharyya. On a measure of divergence between two multinomial populations. *Sankhyā: The Indian Journal of Statistics (1933-1960)*, 7(4):401–406, July 1946. doi:10.2307/25047882.

[BP80] Gerard Bouchard and Christian Pouyez. Name variations and computerized record linkage. *Historical Methods: A Journal of Quantitative and Interdisciplinary History*, 13(2):119–125, 1980. doi:10.1080/01615440.1980.10594037.

[BBL81] Gérard Bouchard, Patrick Brard, and Yolande Lavoie. Fonem: un code de transcription phonétique pour la reconstitution automatique des familles saguenayennes. *Population*, 1981. URL: http://www.persee.fr/doc/pop\T1\textbackslash{}_0032-4663\T1\textbackslash{}_1981\T1\textbackslash{}_num\T1\textbackslash{}_36\T1\textbackslash{}_6\T1\textbackslash{}_17248, doi:10.2307/1532326.

[Boy98] Carolyn B. Boyce. Information on the refined soundex algorithm. November 1998. URL: https://web.archive.org/web/20010513121003/http://www.bluepoof.com:80/Soundex/info2.html.

[Boy11] Leonid Boytsov. Indexing methods for approximate dictionary searching: comparative analysis. *Journal of Experimental Algorithmics*, 16:1.1:1.1–1.1:1.91, May 2011. doi:10.1145/1963190.1963191.

[Bra51] George W. Brainerd. The place of chronological ordering in archaeological analysis. *American Antiquity*, 16(4):301–313, April 1951. doi:10.2307/276979.

[BB32] Josias Braun-Blanquet. *Plant Sociology: The Study of Plant Communities*. McGraw-Hill Book Company, New York, 1932. URL: https://archive.org/details/plantsociologyst00brau.

[BC57] J. Roger Bray and John T. Curtis. An ordination of upland forest communities of southern wisconsin. *Ecological Monographs*, 27(4):325–349, February 1957. URL: http://cescos.fau.edu/gawliklab/papers/BrayJRandJTCurtis1957.pdf, doi:10.2307/1942268.

[Bro97] Andrei Z. Broder. On the resemblance and containment of documents. In *Compression and Complexity of Sequences: Proceedings, Positano, Amalfitan Coast, Salerno, Italy, June 11-13, 1997*, 21–29. 1997. doi:10.1109/SEQUEN.1997.666900.

[BW94] Michael Burrows and David J. Wheeler. A block sorting lossless data compression algorithm. SRC Research Report 124, Digital Equipment Corporation, Palo Alto, May 1994. URL: http://www.hpl.hp.com/techreports/Compaq-DEC/SRC-RR-124.html.

[CBW97] Yong Cao, Anthony W. Bark, and W. Peter Williams. Similarity measure bias in river benthic aufwuchs community analysis. *Water Environment Research*, 69(1):95–106, 1997. doi:10.2175/106143097x125227.

[Cau99] Jörg Caumanns. A fast and simple stemming algorithm for german words. Technical Report, Free University of Berlin, 1999. URL: https://refubium.fu-berlin.de/bitstream/handle/fub188/18405/tr-b-99-16.pdf.

[Cha08] Sung-Hyuk Cha. Taxonomy of nominal type histogram distance measures. In *Proceedings of the American Conference on Applied Mathematics (MATH '08)*. 2008. URL: http://www.wseas.us/e-library/conferences/2008/harvard/math/49-577-887.pdf.

[CTY06] Sung-Hyuk Cha, Charles C. Tappert, and Sungsoo Yoon. Enhancing binary feature vector similarity measures. *Journal of Pattern Recognition Research*, 1(1):63–77, 2006. doi:10.13176/11.20.

[CCCS04] Anne Chao, Robin L. Chazdon, Robert K. Colwell, and Tsung-Jen Shen. A new statistical approach for assessing similarity of species composition with incidence and abundance data. *Ecology Letters*, 8(2):148–159, 2004. doi:10.1111/j.1461-0248.2004.00707.x.

[CCT10] Seung-Seok Choi, Sung-Hyuk Cha, and Charles C. Tappert. A survey of binary similarity and distance measures. *Systemics, Cybernetics and Informatics*, 8(1):43–48, 2010.

[Chr06] Peter Christen. A comparison of personal name matching: techniques and practical issues. Technical Report TR-CS-06-02, Australian National University, Canberra, Australia, 2006. URL: https://openresearch-repository.anu.edu.au/bitstream/1885/44521/3/TR-CS-06-02.pdf.

[Chr11] Peter Christen. Febrl (freely extensible biomedical record linkage) – encode.py. December 2011. URL: https://sourceforge.net/projects/febrl/.

[CGHH91] Kenneth Church, William Gale, Patrick Hanks, and Donald Hindle. Using statistics in lexical analysis. In *Lexical Acquisition: Exploiting On-Line Resources to Build up a Lexicon*, pages 115–164. Lawrence Erlbaum, Hillsdale, NJ, 1991.

[Chu] Richard Churchill. Ueastem.java. URL: http://lemur.cmp.uea.ac.uk/Research/stemmer/UEAstem.java.

[CV05] Rudi Cilibrasi and Paul Michael Béla Vitanyi. Clustering by compression. *IEEE Transactions on Information Theory*, 51(4):1523–1545, April 2005. URL: https://ieeexplore.ieee.org/document/1412045, doi:10.1109/TIT.2005.844059.

[CislakG17] Aleksander Cisłak and Szymon Grabowski. Lightweight fingerprints for fast approximate keyword matching using bitwise operations. *CoRR*, 2017. URL: http://arxiv.org/abs/1711.08475.

[Cla52] Philip J. Clark. An extension of the coefficient of divergence for use with multiple characters. *Copeia*, 1952(2):61–64, June 1952. doi:10.2307/1438532.

[Cle76] Paul W. Clement. A formula for computing inter-observer agreement. *Psychological Reports*, 39(1):257–258, 1976. doi:10.2466/pr0.1976.39.1.257.

[Cod18a] Rosetta Code. Longest common subsequence. 2018. URL: http://rosettacode.org/wiki/Longest\T1\textbackslash{}_common\T1\textbackslash{}_subsequence\T1\textbackslash{}#Dynamic\T1\textbackslash{}_Programming\T1\textbackslash{}_6.

[Cod18b] Rosetta Code. Run-length encoding. 2018. URL: https://rosettacode.org/wiki/Run-length\T1\textbackslash{}_encoding\T1\textbackslash{}#Python.

[Coh11] Adam Cohen. Fuzzywuzzy: fuzzy string matching in python. July 2011. URL: https://chairnerd.seatgeek.com/fuzzywuzzy-fuzzy-string-matching-in-python/.

[Coh60] Jacob Cohen. A coefficient of agreement for nominal scales. *Educational and Psychological Measurement*, 20(1):37–46, 1960. doi:10.1177/001316446002000104.

[CRF03] William A. Cohen, Pradeep Ravikumar, and Stephen E. Fienberg. A comparison of string distance metrics for name-matching tasks. In *IIWEB'03 Proceedings of the 2003 International Conference on Information*, 73–78. 2003. URL: http://www.cs.cmu.edu/~wcohen/postscript/ijcai-ws-2003.pdf.

[CRFR03] William W. Cohen, Pradeep Ravikumar, Stephen E. Fienberg, and Kathryn Rivard. Secondstring. 2003. URL: https://github.com/TeamCohen/secondstring.

[Col49] Lamont C. Cole. The measurement of interspecific association. *Ecology*, 30(4):411–424, 1949. doi:10.2307/1932444.

[CT12] Viviana Consonni and Roberto Todeschini. New similarity coefficients for binary data. *MATCH Communications in Mathematical and in Computer Chemistry*, 68:581–592, 2012.

[Cor03] Graham Cormode. *Seuqnce Distance Embeddings*. PhD thesis, The University of Warwick, 2003. URL: http://wrap.warwick.ac.uk/61310/7/WRAP\T1\textbackslash{}_THESIS\T1\textbackslash{}_Cormode\T1\textbackslash{}_2003.pdf.

[CPSV00] Graham Cormode, Mike Paterson, Süleyman Cenk Sahinalp, and Uzi Vishkin. Communication complexity of document exchange. In *SODA '00 Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms*, 197–200. 2000.

[Cor73] IBM Corporation. *Alpha Search Inquiry System, General Information Manual*. White Plains, NY, 1973.

[Cor17] IBM Corporation. *IBM SPSS Statistics Algorithms*. IBM Corporation, 25 edition, 2017. URL: ftp://public.dhe.ibm.com/software/analytics/spss/documentation/statistics/subscription/en/client/Manuals/IBM\T1\textbackslash{}_SPSS\T1\textbackslash{}_Statistics\T1\textbackslash{}_Algorithms.pdf.

[Cov96] Michael A. Covington. An algorithm to align words for historical comparison. *Computational Linguistics*, 22(4):481–496, December 1996.

[Cro51] Lee J. Cronbach. Coefficient alpha and the internal structure of tests. *Psychometrika*, 16(3):297–334, September 1951. doi:10.1007/BF02310555.

[C+69] Jay L. Cunningham and others. A study of the organization and search of bibliographic holdings in online computer systems: phase i. Technical Report, University of California, Berkleley, Institute of Library Research, March 1969. URL: https://files.eric.ed.gov/fulltext/ED029679.pdf.

[Cze09] Jan Czekanowski. Zur differentialdiagnose der neandertalgruppe. *Korrespondenz-Blatt der Deutschen Gesellschaft für Anthropologie, Ethnologie und Urgeschichte*, 40:44–47, 1909.

[DLP99] Ido Dagan, Lillian Lee, and Fernando C. N. Pereire. Similarity-based models of word cooccurrence probabilities. *Machine Learning*, 34(1–3):43–69, February 1999. doi:10.1023/A:1007537716579.

[Dal05] Andrew Dalke. Arithmetic coder (python recipe). 2005. URL: http://code.activestate.com/recipes/306626/.

[DLZ05] Valentin Dallmeier, Christian Lindig, and Andreas Zeller. Lightweight. In *ECOOP'05 Proceedings of the 19th European conference on Object-Oriented Programming*. 2005. URL: https://www.st.cs.uni-saarland.de/papers/dlz2004/dlz2004.pdf, doi:10.1007/11531142_23.

[Dam64] Fred J. Damerau. A technique for computer detection and correction of spelling errors. *Communications of the ACM*, 7(3):171–176, March 1964. doi:10.1145/363958.363994.

[Dav62] Leon Davidson. Retrieval of misspelled names in an airlines passenger record system. *Communications of the ACM*, 5(3):169–171, March 1962. doi:10.1145/366862.366913.

[dcm4che] dcm4che. DICOM toolkit & library: phonem.java. URL: https://github.com/dcm4che/dcm4che/blob/master/dcm4che-soundex/src/main/java/org/dcm4che3/soundex/Phonem.java.

[Den65] Sally F. Dennis. The construction of a thesaurus automatic from a sample of text. In Mary Elizabeth Stevens, Vincent E. Giuliano, and Laurence B. Heilprin, editors, *Statistical Association Techniques for Mechanized Documentation: Symposium Proceedings*, number 269 in National Bureau of Standards Miscellaneous Publication, 61–148. Washington, D.C., December 1965. United States Department of Commerce. URL: https://archive.org/details/statisticalassoc269stev.

[DD16] Michel Marie Deza and Elena Deza. *Encyclopedia of Distances*. Springer-Verlag, Berlin, 4 edition, 2016.

[Dic45] Lee R. Dice. Measures of the amount of ecologic association between species. *Ecology*, 26(3):297–302, 1945. URL: https://www.jstor.org/stable/1932409, doi:10.2307/1932409.

[Dig83] P. G. N. Digby. Approximating the tetrachoric correlation coefficient. *Biometrics*, 39(3):753–757, September 1983. doi:10.2307/2531104.

[Dol70] James L. Dolby. An algorithm for variable-length proper-name compression. *Journal of Library Automation*, 3(4):257–275, 1970. URL: https://ejournals.bc.edu/ojs/index.php/ital/article/download/5259/4734, doi:10.6017/ital.v3i4.5259.

[Doo84] Mayrick H. Doolittle. The verification of predictions. *The American Meteorological Journal*, 2:327–329, 1884. URL: https://books.google.com/books?id=2f0wAQAAMAAJ&pg=PA327.

[DHC+08] Sean S. Downey, Brian Hallmark, Murray P. Cox, Peter Norquest, and J. Stephen Lansing. Computational featuresensitive reconstruction of language relationships: developing the aline distance for comparative historical linguistic reconstruction. *Journal of Quantitative Linguistics*, 15(4):340–369, November 2008. doi:10.1080/09296170802326681.

[DK32] Harold E. Driver and Alfred L. Kroeber. Quantitative expression of cultural relationships. *University of California Publications in American Archaeology and Ethnology*, 31(4):211–256, 1932. URL: http://digitalassets.lib.berkeley.edu/anthpubs/ucb/text/ucp031-005.pdf.

[Dun93] Ted Dunning. Accurate methods for the statistics of surprise and coincidence. *Computational Linguistics*, 19(1):61–74, 1993. URL: http://www.aclweb.org/anthology/J93-1003.

[EH88] Andrzej Ehrenfeucht and David Haussler. A new distance metric on strings computable in linear time. *Discrete Applied Mathematics*, 20(3):191–203, 1988. doi:10.1016/0166-218X(88)90076-5.

[Eid14] Horst Eidenberger. *Categorization and Machine Learning: The ModModel of Human Understanding in Computers*. atpress, 2014.

[Ell56] Heinz Ellenberg. *Grundlagen Der Vegetationsgliederung. Teil 1. Aufgaben Und Methoden Der Vegetationskunde*. Verlag Eugen Ulmer, Stuttgart, 1956.

[EJMS76] Honey S. Elovitz, Rodney W. Johnson, Astrid McHugh, and John E. Shore. Automatic translation of english text to pphonetic by means of letter-to-sound rules. NRL Report 7948, document AD/A021 929, Naval Research Laboratory, Washington, D.C., 1976.

[Eri97] Klas Erikson. Approximate swedish name matching - survey and test of different algorithms. Nada report TRITA-NA-E9721, KTH, Royal Institute of Technology, Stockholm, Sweden, 1997. URL: ftp://ftp.nada.kth.se/pub/documents/Theory/Viggo-Kann/NADA-E9721.pdf.

[Eyr38] Henri Eyraud. Les principes de la mesure des corrélations. *Annales de l'Universit/e de Lyon, III Series, Section A*, 1:30–47, 1938.

[Fag57] Edward W. Fager. Determination and analysis of recurrent groups. *Ecology*, 38(4):586–595, October 1957. doi:10.2307/1943124.

[FM63] Edward W. Fager and John A. McGowan. Zooplankton species groups in the north pacific. *Science*, 140(3566):453–460, 1963. doi:10.1126/science.140.3566.453.

[Fai83] Daniel P. Faith. Asymmetric binary similarity measures. *Oecologia*, 57(3):287–290, March 1983. doi:10.1007/BF00377169.

[Fle75] Joseph L. Fleiss. Measuring agreement between two judges on the presence or absence of a trait. *Biometrics*, 31(3):651–659, 1975. doi:10.2307/2529549.

[FLP03] Joseph L. Fleiss, Bruce Levin, and Myunghee Cho Paik. *Statistical Methods for Rates and Proportions*. Wiley Series in Probability and Statistics. John Wiley & Sons, Hoboken, 3rd edition, 2003.

[For07] Stephen A. Forbes. On the local distribution of certain illinois fishes: an essay in statistical ecology. *Bulletin of the Illinois State Laboratory of Natural History*, 7:273–303, 1907.

[For25] Stephen A. Forbes. Method of determining and measuring the associative relations of species. *Science*, 61(1585):518–524, 1925.

[FK66] Earl G. Fossum and Gilbert Kaskey. Optimization and standardization of information retrieval language and systems. Technical Report, Directorate of Information Sciences, Air Force Office of Scientific Research, Office of Aerospace Research, United States Air Force, Washington, D.C., 1966. URL: https://archive.org/details/DTIC\T1\textbackslash{}_AD0630797.

[FM83] E. B. Fowlkes and Colin L. Mallows. A method for comparing two hierarchical clusterings. *Journal of the American Statistical Association*, 78(383):553–569, 1983. doi:10.1080/01621459.1983.10478008.

[FurnrohrRvR02] Michael Fürnrohr, Birgit Rimmelspacher, and Tilman von Roncador. Zusammenführung von datenbeständen ohne numerische identifikatoren: ein verfahren im rahmen der testuntersuchungen zu einem registergestützten zensus. *Bayern in Zahlen*, 2002(7):308–321, 2002. URL: https://www.statistik.bayern.de/medien/statistik/zensus/zusammenf\T1\textbackslash{}\T1\textbackslash{}_hrung\T1\textbackslash{}_von\T1\textbackslash{}_datenbest\T1\textbackslash{}\T1\textbackslash{}_nden\T1\textbackslash{}_ohne\T1\textbackslash{}_numerische\T1\textbackslash{}_identifikatoren.pdf.

[Gad90] T. N. Gadd. Phonix: the algorithm. *Program*, 24(4):363–366, 1990. doi:10.1108/eb047069.

[Gar15] Lars Marius Garshol. Norphone comparator. 2015. URL: https://github.com/larsga/Duke/blob/master/duke-core/src/main/java/no/priv/garshol/duke/comparators/NorphoneComparator.java.

[GM88] Wilde Georg and Carsten Meyer. Nicht wörtlich genommen, 'schreibweisentolerante' suchroutine in dbase implementiert. *c't Magazin für Computer Technik*, pages 126–131, October 1988.

[GW66] N. Gilbert and Terry C. E. Wells. Analysis of quadrat data. *Journal of Ecology*, 54(3):675–685, November 1966. doi:10.2307/2257810.

[Gil84] Grove K. Gilbert. Finley's tornado predictions. *American Meteorological Journal*, 1:166–172, 1884.

[Gil97] Leicester E. Gill. Ox-link: the oxford medical record linkage system. In *Record Linkage Techniques*. Washington, D.C., March 1997. Federal Committee on Statistical Methodology, Office of Management and Budget. URL: https://pdfs.semanticscholar.org/fff7/02a3322e05c282a84064ee085e589ef74584.pdf.

[Gin12] Corrado Gini. *Variabilità e mutabilità*. Contributo allo Studio delle Distribuzioni e delle Relazioni Statistiche. C. Cuppini, Bologna, 1912.

[Gin15] Corrado Gini. Nuovi contributi all teoria delle relazioni statistiche. *Atti del Reale Istituto Veneto di Scienze, Lettere ed Arti, Series 8*, 74(2):1903–1942, 1915.

[Gle20] Henry Allan Gleason. Some applications of the quadrat method. *Bulletin of the Torrey Botanical Club*, 47(1):21–33, January 1920. doi:10.2307/2480223.

[Goo67] David W. Goodall. The distribution of the matching coefficient. *Biometrics*, 23(4):647–656, December 1967. doi:10.2307/2528419.

[GK54] Leo A. Goodman and William H. Kruskal. Measures of association for cross classification i. *Journal of the American Statistical Association*, 49(268):732–764, 1954. doi:10.2307/2281536.

[GK59] Leo A. Goodman and William H. Kruskal. Measures of association for cross classification ii: further discussion and references. *Journal of the American Statistical Association*, 54(285):123–163, March 1959. doi:10.2307/2282143.

[Got82] Osamu Gotoh. An improved algorithm for matching biological sequences. *Journal of Molecular Biology*, 162(3):705–708, 1982. URL: http://www.sciencedirect.com/science/article/pii/0022283682903989, doi:10.1016/0022-2836(82)90398-9.

[Gow71] John C. Gower. A general coefficient of similarities and some of its properties. *Biometrics*, 27(4):857–871, December 1971. doi:10.2307/2528823.

[GL86] John C. Gower and Pierre Legendre. Metric and Euclidean properties of dissimilarity coefficients. *Journal of Classification*, 3(1):5–48, February 1986. doi:10.1007/BF01896809.

[GIJ+01] Luis Gravano, Panagiotis G. Ipeirotis, H. V. Jagadish, Nick Koudas, S. Muthukrishman, and Divesh Srivastava. Approximate string joins in a database (almost) for free. In *Proceedings of the 27th VLDB Conference, Roma, Italy, 2001*. 2001.

[Gro91] Aaron D. Gross. Getty synoname: the development of software for personal name pattern matching. In *Intelligent Text and Image Handling - Volume 2*, RIAO '91, 754–763. Paris, France, France, 1991. LE CENTRE DE HAUTES ETUDES INTERNATIONALES D'INFORMATIQUE DOCUMENTAIRE. URL: http://dl.acm.org/citation.cfm?id=3171004.3171021.

[Guirk] J. P. Guildford. *Fundamental Statistics in Psychology and Education*. McGraw-Hill Book Company, New York, New York. URL: https://archive.org/details/in.ernet.dli.2015.228996.

[Gut76] Gloria J. A. Guth. Surname spellings and computerized record linkage. *Historical Methods Newsletter*, 10(1):10–19, 1976. doi:10.1080/00182494.1976.10112645.

[Gut41] Louis Guttman. An outline of the statistical theory of prediction. In Paul Horst, editor, *The Prediction of Personal Adjustment*, number 48, pages 253–311. Social Science Research Council, 1941. URL: https://babel.hathitrust.org/cgi/pt?id=uc1.b4579784;view=1up;seq=271.

[Gwe08] Kilem Li Gwet. Computing inter-rater reliability and its variance in the presence of high agreement. *British Journal of Mathematical and Statistical Psychology*, 61(1):29–48, 2008. doi:10.1348/000711006X126600.

[HH00] Martin Haase and Kai Heitmann. Die erweiterte kölner phonetik. 2000.

[Ham61] Ulrich Hamann. Merkmalbestand und verwandtschaftsbeziehungen der farinosae: ein beitrag zum system der monokotyledonen. *Willdenowia*, 2:639–768, 1961.

[Ham50] R. W. Hamming. Error detecting and error correcting codes. *The Bell System Technical Journal*, 29(2):147–160, April 1950. URL: https://ieeexplore.ieee.org/document/6772729/, doi:10.1002/j.1538-7305.1950.tb00463.x.

[Har91] Donna Harman. How effective is stemming? *Journal of the American Society for Information Science*, 42(1):7–15, 1991. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.104.9828\T1\textbackslash{}&rep=rep1\T1\textbackslash{}&type=pdf, doi:10.1002/(SICI)1097-4571(199101)42:1\%3C7::AID-ASI2\%3E3.0.CO;2-P.

[HL78] Francis C. Harris and Benjamin B. Lahey. A method for combining occurrence and nonoccurrence interobserver agreement scores. *Journal of Applied Behavior Analysis*, 11(4):523–527, 1978. doi:10.1901/jaba.1978.11-523.

[Has14] Ahmad Basheer Hassanat. Dimensionality invariant similarity measure. *Journal of American Science*, 10(8):221–226, 2014. URL: https://arxiv.org/abs/1409.0923.

[HD73] Robert P. Hawkins and Victor A. Dotson. Reliability scores that delude: an alice in wonderland trip through the misleading characteristics of inter-observer agreement scores in interval recording. Technical Report, Western Michigan University, 1973. URL: https://eric.ed.gov/?id=ED094277.

[Hel09] Ernst Hellinger. Neue begründung der theorie quadratischer formen von unendlichvielen veränderlichen. *Journal Für Die Reine Und Angewandte Mathematik*, 1909(136):210–271, 1909. doi:10.1515/crll.1909.136.210.

[HH77] Robert A. Henderson and Malcolm L. Heron. A probabilistic method of paleobiogeographic analysis. *Lethaia*, 10(1):1–15, 1977. doi:10.1111/j.1502-3931.1977.tb00584.x.

[Hen76] Louis Henry. Projet de transcription phonétique des noms de famille. *Annales de Démographie Historique*, 1976:201–214, 1976. URL: https://www.persee.fr/doc/adh\T1\textbackslash{}_0066-2062\T1\textbackslash{}_1976\T1\textbackslash{}_num\T1\textbackslash{}_1976\T1\textbackslash{}_1\T1\textbackslash{}_1313.

[HBD76] Theodore Hershberg, Alan Burstein, and Robert Dockhorn. Record linkage. *Historical Methods Newsletter*, 9(2–3):137–163, 1976. doi:10.1080/00182494.1976.10112639.

[HBD79] Theodore Hershberg, Alan Burstein, and Robert Dockhorn. Verkettung von daten: record linkage am beispiel des philadelphia social history project. In Wilhelm Heinz Schröder, editor, *Moderne Stadtgeschichte*, volume 8, pages 35–73. Klett-Cotta, 1979. URL: https://www.ssoar.info/ssoar/handle/document/32782.

[HM02] David Holmes and M. Catherine McCabe. Improving precision and recall for soundex retrieval. In *Proceedings. International Conference on Information Technology: Coding and Computing*, 22–26. April 2002. URL: https://ieeexplore.ieee.org/document/1000354/, doi:10.1109/ITCC.2002.1000354.

[Hoo02] David Hood. Cavesystem: phonetic matching algorithm. Technical Report CTP060902, University of Otago, Dunedin, New Zealand, September 2002. URL: https://caversham.otago.ac.nz/files/working/ctp060902.pdf.

[Hoo04] David Hood. Caverphone revisited. Technical Report CTP150804, University of Otago, Dunedin, New Zealand, December 2004. URL: https://caversham.otago.ac.nz/files/working/ctp150804.pdf.

[Hor66] Henry S. Horn. Measurement of "overlap" in comparative ecological studies. *The American Naturalist*, 100(914):419–424, September 1966. doi:10.2307/2459242.

[Hubalek08] Zdenek Hubálek. Coefficients of association and similarity, based on binary (presence-absence) data: an evaluation. *Biological Reviews*, 57(4):669–689, February 2008. doi:10.1111/j.1469-185X.1982.tb00376.x.

[Hur69] Stuart H. Hurlbert. A coefficient of interspecific assciation. *Ecology*, 50(1):1–9, January 1969. doi:10.2307/1934657.

[Jac01] Paul Jaccard. Distribution de la flore alpine dans le bassin des dranses et dans quelques régions voisines. *Bulletin de la Société Vaudoise des Sciences Naturelles*, 37:241–272, 1901. URL: https://core.ac.uk/download/pdf/20654241.pdf.

[Jar89] Matthew A. Jaro. Advances in record linkage methodology as applied to the 1985 census of tampa florida. *Journal of the American Statistical Association*, 84(406):414–420, 1989. doi:10.1080/01621459.1989.10478785.

[JS05] Marie-Claire Jenkins and Dan Smith. Conservative stemming for search and indexing. Technical Report, University of East-Anglia, Norwich, UK, 2005. URL: http://lemur.cmp.uea.ac.uk/Research/stemmer/stemmer25feb.pdf.

[JBG13] Sergio Jiminez, Claudio Becerra, and Alexander Gelbukh. SOFTCARDINALITY-CORE: improving text overlap with distributional measures for semantic textual similarity. In *Second Joint Conference on Lexical and Computational Semantics (\textasteriskcenteredSEM), Volume 1: Proceedings of the Main Conference and the Shared Task*, 194–201. Atlanta, GA, June 2013. Association for Computational Linguistics. URL: http://www.aclweb.org/anthology/S13-1028.

[Joh67] Stephen C. Johnson. Hierarchical clustering schemes. *Psychometrika*, 32(3):241–254, September 1967. doi:10.1007/BF02289588.

[JH05] James A. Jones and Mary Jean Harrold. Empirical evaluation of the tarantula automatic fault-localization technique. In *ASE '05 Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*, 273–282. New York, November 2005. ACM, ACM. doi:10.1145/1101908.1101949.

[Kem05] Sebastian Kempken. Bewertung historischer und regionaler schreibvarianten mit hilfe von abstandsmaßen. Master's thesis, Universität Duisburg-Essen, December 2005. URL: https://duepublico.uni-duisburg-essen.de/servlets/DerivateServlet/Derivate-17252/BewertungSchreibvarianten.pdf.

[Ken38] Maurice G. Kendall. A new measure of rank correlation. *Biometrika*, 30(1/2):81–93, June 1938. doi:10.2307/2332226.

[KF77] Ronald N. Kent and Sharon L. Foster. Direct observational procedure: methodological issues in naturalistic settings. In Anthony R. Ciminero, Karen, S. Calhoun, and Henry E. Adams, editors, *Handbook of Behavioral Assessment*, chapter 9, pages 279–328. John Wiley & Sons, New York, 1977. URL: https://archive.org/details/handbookofbehavi00cimi.

[Knu98] Donald E. Knuth. *The Art of Computer Programming: Volume 3, Sorting and Searching*, pages 394. Addison-Wesley, 1998.

[Kollar] Maroš Kollár. Text::phonetic::phonix. URL: https://github.com/maros/Text-Phonetic/blob/master/lib/Text/Phonetic/Phonix.pm.

[Kon00] Grzegorz Kondrak. A new algorithm for the alignment of phonetic sequences. In *NAACL 2000 Proceedings of the 1st North American chapter of the Association for Computational Linguistics conference*. 2000. doi:10.0000/dl.acm.org/974343.

[Kon02] Grzegorz Kondrak. *Algorithms for Language Reconstruction*. PhD thesis, University of Toronto, 2002. URL: https://webdocs.cs.ualberta.ca/~kondrak/papers/thesis.pdf.

[KD03] Grzegorz Kondrak and Bonnie J. Dorr. A similarity-based approach and evaluation methodology for reduction of drug name confusion. Technical Report, University of Maryland, Institute for Advanced Computer Studies, 2003. URL: https://apps.dtic.mil/dtic/tr/fulltext/u2/a452242.pdf.

[KV17] Kerrthi Koneru and Cihan Varol. Privacy preserving record linkage using metasoundex algorithm. In *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*, 443–447. December 2017. URL: https://ieeexplore.ieee.org/document/8260671/, doi:10.1109/ICMLA.2017.0-121.

[KR37] G. Frederic Kuder and Marion Webster Richardson. The theory of the estimation of test reliability. *Psychometrika*, 2(3):151–160, September 1937. doi:10.1007/bf02288391.

[Kuh95] Michael Kuhn. Metaphone searches. November 1995. URL: http://aspell.net/metaphone/metaphone-kuhn.txt.

[Kuh64] John L. Kuhns. The continuum of coefficients of association. In Mary Elizabeth Stevens, Vincent E. Giuliano, and Laurence B. Heilprin, editors, *Statistical Association Methods for Mechanized Documentation*, number 269 in National Bureau of Standards Miscellaneous Publication, 33–40. 1964.

[Kul15] Maciej Kula. Simple minhash implementation in python. June 2015. URL: https://maciejkula.github.io/2015/06/01/simple-minhash-implementation-in-python/.

[Kulczynski27] Stanisław Kulczyński. Die pflanzenassoziationen der pieninen. *Bulletin International de l'Academie Polonaise des Sciences et des Lettres, Classe des Sciences Mathematiques et Naturelles, B (Sciences Naturelles)*, pages 57–203, 1927.

[Koppen70] Wladimir Köppen. Die aufeinanderfolge der periodischen witterungserscheinungen nach den grundsätzen der wahrscheinlichkeitsrechnung. In *Repertorium für Meteorologie*, volume 2, pages 189–238. Akademiia Nauk, 1870. URL: https://books.google.com/books?id=1ww0AQAAMAAJ\T1\textbackslash{}&pg=RA1-PA187\T1\textbackslash{}#v=onepage\T1\textbackslash{}&q\T1\textbackslash{}&f=false.

[LR96] Andrew J. Lait and Brian Randell. An assessment of name matching algorithms. Technical Report, University of Newcastle upon Tyne, Newcastle upon Tyne, UK, 1996. URL: http://homepages.cs.ncl.ac.uk/brian.randell/Genealogy/NameMatching.pdf.

[LW66] Godfrey N. Lance and William T. Williams. Computer programs for hierarchical polythetic classification ("similarity analysis"). *Computer Journal*, 1966. doi:10.1093/comjnl/9.1.60.

[LW67a] Godfrey N. Lance and William T. Williams. A general theory of classificatory sorting strategies. ii. clustering systems. *Computer Journal*, 10(3):271–277, January 1967. URL: https://academic.oup.com/comjnl/article-pdf/10/3/271/1333425/100271.pdf, doi:10.1093/comjnl/10.3.271.

[LW67b] Godfrey N. Lance and William T. Williams. Mixed-data classificatory programs i. agglomerative systems. *Australian Computer Journal*, 1:15–20, 1967.

[Lan13] Joerg Lang. Inner wworking of the german analyzer in lucene. November 2013. URL: http://www.evelix.ch/unternehmen/Blog/evelix/2013/11/11/inner-workings-of-the-german-analyzer-in-lucene.

[LL98] Pierre Legendre and Louis Legendre. *Numerical Ecology*. Number 20 in Developments in Environmental Modelling. Elsevier, Amsterdam, 2nd edition, 1998.

[Lev65] Vladimir I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Doklady Akademii Nauk SSSR*, 163(4):845–848, 1965. URL: http://mi.mathnet.ru/dan31411.

[Lev66] Vladimir I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10(8):707–710, February 1966. URL: https://nymity.ch/sybilhunting/pdf/Levenshtein1966a.pdf.

[Lin04] Chin-Yew Lin. Rouge: a package for automatic evaluation of summaries. In *Text Summarization Branches Out*. 2004. URL: http://aclweb.org/anthology/W04-1013.

[LSShaweTaylor+02] Huma Lodhi, Craig Saunders, John Shawe-Taylor, Nello Cristianini, and Chris Watkins. Text classification using string kernels. *Journal of Machine Learning Research*, 2:419–444, 2002. doi:10.1162/153244302760200687.

[Lov68] Julie Beth Lovins. Development of a stemming algorithm. *Mechanical Translation and Computational Linguistics*, 11(1–2):22–31, June 1968. URL: http://www.mt-archive.info/MT-1968-Lovins.pdf.

[LA77] Billy T. Lynch and William L. Arends. Selection of a surname coding procedure for the srs record linkage system. Technical Report, Statistical Reporting Service, US Department of Agriculture, Washington, D.C., February 1977. URL: https://naldc.nal.usda.gov/download/27833/PDF.

[LegareLC72] Jacques Légaré, Yolande Lavoie, and Hubert Charbonneau. The early canadian population: problems in automatic record linkage. *Canadian Historical Review*, 53(4):427–442, December 1972. doi:10.3138/CHR-053-04-03.

[Mar15] Daniel Marcelino. Soundexbr: soundex (phonetic) algorithm for Brazilian portuguese. July 2015. URL: https://github.com/danielmarcelino/SoundexBR.

[Mat75] Brian W. Matthews. Comparison of the predicted and observed secondary structure of t4 phage lysozyme. *Biochimica et Biophysica Acta (BBA) - Protein Structure*, 405(2):442–451, 1975.

[Mat55] Kameo Matusita. Decision rules, based on the distance, for problems of fit, two samples, and estimation. *The Annals of Mathematical Statistics*, 26(4):631–640, December 1955. doi:10.2307/2236376.

[MP68] A. E. Maxwell and A. E. G. Pilliner. Deriving coefficients of reliability and agreement for ratings. *The British Journal of Mathematical and Statistical Psychology*, 21(1):105–116, May 1968. doi:10.1111/j.2044-8317.1968.tb00401.x.

[McC64] Bayard H. McConnaughey. The determination and analysis of plankton communities. *Lembaga Penelitian Laut*, pages 1–40, 1964.

[Mic99] Jörg Michael. Doppelgänger gesucht – ein programm für die kontextsensitive phonetische stringumwandlung. *c't Magazin für Computer Technik*, pages 252, 1999. URL: http://www.heise.de/ct/ftp/99/25/252/.

[Mic07] Jörg Michael. Phonet.c. August 2007. URL: ftp://ftp.heise.de/pub/ct/listings/phonet.zip.

[Mic20] Ellis L. Michael. Marine ecology and the coefficient of association: a plea in behalf of quantitative biology. *The Journal of Ecology*, 8(1):54–59, 1920. doi:10.2307/2255213.

[Min10] Hermann Minkowski. *Geometrie der Zahlen*. R. G. Teubner, Leipzig, 1910. URL: https://archive.org/stream/geometriederzahl00minkrich.

[Mok97] Gary Mokotoff. Soundexing and genealogy. 1997. URL: http://www.avotaynu.com/soundex.htm.

[ME96] Alvaro E. Monge and Charles P. Elkan. The field matching problem: algorithms and applications. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD'96, 267–270. AAAI Press, 1996. URL: http://dl.acm.org/citation.cfm?id=3001460.3001516.

[MKTM77] Gwendolyn B. Moore, John L. Kuhns, Jeffrey L. Trefftzs, and Christine A. Montgomery. *Accessing Individual Records from Personal Data Files Using Non-Unique Identifiers*. Number 500-2 in Special Publication. National Bureau of Standards, Washington, D.C., February 1977. URL: https://archive.org/details/accessingindivid00moor.

[MYCappe08] Erwan Moreau, François Yvon, and Olivier Cappé. Robust similarity measures for named entities matching. In *COLING '08 Proceedings of the 22nd International Conference on Computational Linguistics - Volume 1*, 593–600. August 2008.

[Mor59] Masaaki Morisita. Measuring of interspecific association and similarity between communities. In *Memoirs of the Faculty of Science*, volume 3 of Series E (Biology), pages 65–80. Kyushu University, 1959.

[Mor12] James F. Morris. *A Quantitative MethoMethod for Vetting "Dark Network" Intelligence Sources for Social Network Analysis*. PhD thesis, Air Force Institute of Technology, 2012. URL: https://apps.dtic.mil/dtic/tr/fulltext/u2/a561702.pdf.

[MLM12] Alejandro Mosquera, Elena Lloret, and Paloma Moreda. Towards facilitating the accessibility of web 2.0 Texts through text normalisation. In *Proceedings of the LREC workshop: Natural Language Processing for Improving Textual Accessibility (NLP4ITA) ; Istanbul, Turkey.*, 9–14. 2012. URL: http://www.taln.upf.edu/pages/nlp4ita/pdfs/mosquera-nlp4ita2012.pdf.

[MDobrzanskiZ50] J. Motyka, B. Dobrzański, and S. Zawadzki. Wstçpne badania nad lakami paludniowo-wschodnilj lubel-szczyzny (preliminary studies on meadows in the south-east of the province lublin). *Annales Universitatis Mariae Curie-Skłodowska, Sectio E*, 5(13):367–447, 1950.

[Mou62] M. D. Mountford. An index of similarity and its application to classificatory problems. In P. W. Murphy, editor, *Progress in Soil Zoology: Papers from a Colloquium on Research Methods Organized by the Soil Zoology Committee of the International Society of Soil Science*, 43–50. London, July 1962. Butterworths. URL: https://openlibrary.org/books/OL5908681M/Progress\T1\textbackslash{}_in\T1\textbackslash{}_soil\T1\textbackslash{}_zoology.

[Moz36] Alan Mozley. The statistical analysis of the distribution of pond molluscs in western Canada. *The American Naturalist*, 1936. doi:10.1086/280660.

[NMM11] Rashid Naseem, Onaiza Maqbool, and Siraj Muhammad. Improved similarity measures for software clustering. In *Proceedings of the Euromicro Conference on Software Maintenance and Reengineering, CSMR*. March 2011. doi:10.1109/CSMR.2011.9.

[NW70] Saul B. Needleman and Christian D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3):443–453, 1970. URL: http://www.sciencedirect.com/science/article/pii/0022283670900574, doi:10.1016/0022-2836(70)90057-4.

[Och57] Akira Ochiai. Zoogeographical studies on the soleoid fishes found in Japan and its neighhouring regions-ii. *Bulletin of the Japanese Society of Scientific Fisheries*, 22(9):526–530, 1957. URL: https://www.jstage.jst.go.jp/article/suisan1932/22/9/22\T1\textbackslash{}_9\T1\textbackslash{}_526/\T1\textbackslash{}_pdf/-char/en, doi:10.2331/suisan.22.526.

[oC13] Library of Congress. *Classification and Shelflisting Manual*. Library of Congress, 2013. URL: https://www.loc.gov/aba/publications/FreeCSM/freecsm.html.

[Ope12] OpenRefine. Clustering in depth. 2012. URL: https://github.com/OpenRefine/OpenRefine/wiki/Clustering-In-Depth.

[Orloci67] Laszlo Orlóci. An agllomerative method for classification of plant communities. *The Journal of Ecology*, 55(1):193–206, March 1967. doi:10.2307/2257725.

[Ots36] Yanosuke Otsuka. The faunal character of the Japanese pleistocene marine mollusca, as evidence of the climate having become colder during the pleistocene in Japan. *Bulletin of the Biogeographical Society of Japan*, 6(16):165–170, 1936.

[Ozb15] Hakan Ozbay. Ozbay metric. 2015. URL: https://github.com/hakanozbay/ozbay-metric.

[Pai90] Chris D. Paice. Another stemmer. In *ACM SIGIR Forum*, volume 24, 56–61. Fall 1990. URL: https://dl.acm.org/citation.cfm?id=101310, doi:10.1145/101306.101310.

[PRWZ02] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL), Philadelphia, July 2002*, 311–318. 2002. URL: https://www.aclweb.org/anthology/P02-1040.pdf.

[PK14] Vimal P. Parmar and CK Kumbharana. Study existing various phonetic algorithms and designing and development of a working model for the new developed algorithm and comparison by implementing ti with existing algorithm(s). *International Journal of Computer Applications*, 98(19):45–49, 2014. doi:10.5120/17295-7795.

[Pas06] Rebecca Passonneau. Measuring agreement on set-valued items (masi) for semantic and pragmatic annotation. In *Proceedings of the Fifth International Conference on Language Resources and Evaluation (LREC'06)*, 831–836. May 2006.

[Pea00] Karl Pearson. Mathematical contributions to the theory of evolution. vii. on the correlation of characters not quantitatively measurable. *Philosophical Transactions of the Royal Society*, 195 A:1–47, 1900. doi:10.1098/rsta.1900.0022.

[PH13] Karl Pearson and David Heron. On theories of association. *Biometrika*, 9(1/2):159–315, 1913. doi:10.2307/2331805.

[Pec10] Pavel Pecina. Lexical association measures and collocation extraction. *Language Resources & Evaluation*, 44(1/2):137–158, 2010. doi:10.2307/40666353.

[Pei84] Charles S. Peirce. The numerical measure of the success of predictions. *Science*, 4(93):453–454, 1884. doi:10.1126/science.ns-4.93.453-a.

[Pen52] Lionel S. Penrose. Distance, size and shape. *Annals of Eugenics*, 17(1):337–343, January 1952. doi:10.1111/j.1469-1809.1952.tb02527.x.

[Pfe00] Ulrich Pfeifer. Wait 1.8 - soundex.c. 2000. URL: https://fastapi.metacpan.org/source/ULPFR/WAIT-1.800/soundex.c.

[Phi90a] Lawrence Philips. Hanging on the metaphone. *Computer Language*, 7(12):39–44, December 1990.

[Phi90b] Lawrence Philips. Metaphone. December 1990. URL: http://aspell.net/metaphone/metaphone.basic.

[Phi00] Lawrence Philips. The double metaphone search algorithm. *C/C++ Users Journal*, 18(6):38–43, June 2000.

[Pli18] Guillaume Plique. Talisman. 2018. URL: https://github.com/Yomguithereal/talisman.

[PZ84] Joseph J. Pollock and Antonio Zamora. Automatic spelling correction in scientific and scholarly text. *Communications of the ACM*, 27(4):358–368, April 1984. URL: http://dl.acm.org/citation.cfm?id=358048, doi:10.1145/358027.358048.

[Por80] Martin F. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, July 1980. URL: http://snowball.tartarus.org/algorithms/porter/stemmer.html, doi:10.1108/eb046814.

[Por02] Martin F. Porter. The english (porter2) stemming algorithm. September 2002. URL: http://snowball.tartarus.org/algorithms/english/stemmer.html.

[Pos69] Hans Joachim Postel. Die kölner phonetik: ein verfahren zur identifizierung von personennamen auf der grundlage der gestaltanalyse. *IBM-Nachrichten*, 19:925–931, 1969.

[Pra15] Jörg Prante. Elasticsearch – haasephonetik.java. 2015. URL: https://github.com/elastic/elasticsearch/blob/master/plugins/analysis-phonetic/src/main/java/org/elasticsearch/index/analysis/phonetic/HaasePhonetik.java.

[Rruvzivcka58] M. Růžička. Anwendung mathematische-statistischer methoden in der geobotanik (synthetische bearbeitung von aufnahmen). *Biologia, Bratislava*, 13:647–661, 1958.

[RTS+01] Dragomir Radev, Simone Teufel, Horacio Saggion, Wai Lam, John Blitzer, Arda Çelebi, Hong Qi, Elliott Drabek, and Danyu Liu. Evaluation of text summarization in a cross-lingual information retrieval framework. Technical Report, Johns Hopkins, 2001. URL: https://pdfs.semanticscholar.org/44a1/df62a1c815fc84aa42788283655a38c85550.pdf.

[Ran71] William M. Rand. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical Association*, 66(336):846–850, December 1971. doi:10.2307/2284239.

[RM88] John W. Ratcliff and David E. Metzener. Pattern matching: the gestalt approach. *Dr. Dobbs Journal*, 1988. URL: http://www.drdobbs.com/database/pattern-matching-the-gestalt-approach/184407970.

[RC79] David M. Raup and Rex E. Crick. Measurement of faunal similarity in paleontology. *Journal of Paleontology*, 53(5):1213–1227, September 1979. doi:10.2307/1304099.

[RaissouliLC09] Mustapha Raïssouli, Fatima Leazizi, and Mohamed Chergui. Arithmetic-geometric-harmonic mean of three positive operators. *Journal of Inequalities in Pure and Applied Mathematics*, 2009. URL: http://www.emis.de/journals/JIPAM/images/014\T1\textbackslash{}_08\T1\textbackslash{}_JIPAM/014\T1\textbackslash{}_08.pdf.

[Ree14] Tony Rees. Taxamatch, an algorithm for near ('fuzzy') matching on scientific names in taxonomic databases. *PLoS ONE*, 9(9):1–27, September 2014. doi:10.1371/journal.pone.0107510.

[RB13] Tony Rees and Barbara Boehmer. The mdld (modified damerau-levenshtein distance) algorithm. November 2013. URL: https://confluence.csiro.au/public/taxamatch/the-mdld-modified-damerau-levenshtein-distance-algorithm.

[Rep13] Dominic John Repici. Understanding classic soundex algorithms. 2013. URL: http://creativyst.com/Doc/Articles/SoundEx1/SoundEx1.htm\T1\textbackslash{}#SoundExAndCensus.

[RU09] Nicholas Ring and Alexandra L. Uitdenbogerd. Finding `lucy in disguise': the misheard lyric matching problem. In Gary Geunbae Lee, Dawei Song, Chin-Yew Lin, Akiko Aizawa, Kazuko Kuriyama, Masaharu Yoshioka, and Tetsuya Sakai, editors, *Information Retrieval Technology*, 157–167. Berlin, Heidelberg, 2009. Springer Berlin Heidelberg. doi:10.1007/978-3-642-04769-5_14.

[Rob86] David W. Roberts. Ordination on the basis of fuzzy set theory. *Vegetatio*, 66(3):123–131, 1986. doi:10.1007/BF00039905.

[RC67] A. H. Robinson and Colin Cherry. Results of a prototype television bandwidth compression scheme. In *Proceedings of the IEEE*, volume 55, 356–364. IEEE, 1967. doi:10.1109/PROC.1967.5493.

[Rob51] W. S. Robinson. A method for chronologically ordering archaeological deposits. *American Antiquity*, 16(4):293–301, April 1951. doi:10.2307/276978.

[RT60] David J. Rogers and Taffee T. Tanimoto. A computer program for classifying plants. *Science*, 132(3434):1115–1118, October 1960. doi:10.1126/science.132.3434.1115.

[RG66] Eugene Rogot and Irving D. Goldberg. A proposed index for measuring agreement in test-retest studies. *Journal of Chronic Diseases*, 1966. doi:10.1016/0021-9681(66)90032-4.

[RY05] Gong Ruibin and Chan Kai Yun. An adaptive model for phonetic string search. In *Knowledge-Based Intelligent Information and Engineering Systems, 9th International Conference, KES 2005 Melbourne, Australia, September 14-16, 2005 Proceedings, Part III*, volume 3683 of Lecture Notes in Artificial Intelligence, 915–921. 2005.

[Ruk18] Dorothea Rukasz. Pprl – privacy preserving record linkage. 2018. URL: https://github.com/cran/PPRL.

[RHJF14] Daniel E. Russ, Kwan-Yuet Ho, Calvin A. Johnson, and Melissa C. Friesen. Computer-based coding of occupation codes for epidemiological analysis. In *2014 IEEE 27th International Symposium on Computer-Based Medical Systems*, 347–350. 2014. doi:10.1109/CBMS.2014.79.

[RR40] Paul F. Russell and T. Ramachandra Rao. On habitat and association of species of anopheline larvae in south-eastern madras. *Journal of the Malaria Institute of India*, 3(1):153–178, 1940.

[Rus18] Robert C. Russell. Index. 1918. URL: https://patentimages.storage.googleapis.com/31/35/a1/f697a3ab85ced6/US1261167.pdf.

[Sav05] Jacques Savoy. IR multilingual resources at unine. 2005. URL: http://members.unine.ch/jacques.savoy/clef/.

[Schurer07] Kevin Schürer. Creating a nationally representative individual and household sample for great britain, 1851 to 1901 - the victorian panel study (vps). *Historical Social Research / Historische Sozialforschung*, 32(2):211–331, 2007. doi:10.2307/20762213.

[SGRW96] Robyn Schinke, Mark Greengrass, Alexander M. Robertson, and Peter Willett. A stemming algorithm for latin text databases. *Journal of Documentation*, 52(2):172–187, 1996. doi:10.1108/eb026966.

[SBB04] Rainer Schnell, Tobias Bachteler, and Stefan Bender. A toolbox for record linkage. *Australian Journal of Statistics*, 33(1-2):125–133, 2004. URL: https://pdfs.semanticscholar.org/2353/21c24ed0401cd05d7752c2c8a8da5b7a4dc0.pdf.

[Sco55] William A. Scott. Reliability of content analysis: the case of nominal scale coding. *Public Opinion Quarterly*, 19(3):321–325, 1955. doi:10.1086/266577.

[Sei93] Heinz-Jürgen Seiffert. Problem 887. *Nieuw Archief voor Wiskunde*, 11(4):176, 1993.

[Seq18] SequentiX. Distance measures. 2018. URL: https://www.sequentix.de/gelquest/help/distance\T1\textbackslash{}_measures.htm.

[SA10] Boumedyen A. N. Shannaq and Victor V. Alexandrov. Using product similarity for adding business. *Global Journal of Computer Science and Technology*, 10(12):2–8, October 2010. URL: https://www.sial.iias.spb.su/files/386-386-1-PB.pdf.

[SS07] Dana Shapira and James A. Storer. Edit distance with move operations. *Journal of Discrete Algorithms*, 5(2):380–392, June 2007. doi:10.1016/j.jda.2005.01.010.

[Shi93] Guang R. Shi. Multivariate data analysis in palaeoecology and palaeobiogeography—a review. *Palaeogeography, Palaeoclimatology, Palaeoecology*, 105(3-4):199–234, 1993. doi:10.1016/0031-0182(93)90084-v.

[SGGomezAP14] Grigori Sidorov, Alexander Gelbukh, Helena Gómez-Adorno, and David Pinto. Soft similarity and soft cosine measure: similarity of features in vector space model. *Computación y Sistemas*, 2014. URL: http://www.scielo.org.mx/pdf/cys/v18n3/v18n3a7.pdf, doi:10.13053/CyS-18-3-2043.

[Sim49] Edward H. Simpson. Measurement of diversity. *Nature*, 163:688, April 1949. URL: https://www.nature.com/articles/163688a0, doi:10.1038/163688a0.

[Sjoo09] Allan Sjöö. Swamisfinxbix. 2009. URL: http://www.swami.se/download/18.248ad5af12aa8136533800093/swamiSfinxBis.java.txt.

[SW81] Temple F. Smith and Michael S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1):195–197, 1981. URL: http://www.sciencedirect.com/science/article/pii/0022283681900875, doi:10.1016/0022-2836(81)90087-5.

[SD02] Chakkrit Snae and Bernard Diaz. An interface for mining genealogical nominal data using the concept of linkage and a hybrid name matching algorithm. *Journal of 3D-Forum Society*, 16(1):142–147, 2002. URL: https://web.archive.org/web/20050329140715/www.csc.liv.ac.uk/~chakkrit/Publications/hc2001\T1\textbackslash{}_Journal.pdf.

[SM58] Robert R. Sokal and Charles D. Michener. A statistical method for evaluating systematic relationships. *The University of Kansas Science Bulletin*, 38, part 2(22):1409–1438, March 1958. URL: https://archive.org/details/cbarchive\T1\textbackslash{}_133648\T1\textbackslash{}_astatisticalmethodforevaluatin1902.

[SS63] Robert R. Sokal and Peter H. A. Sneath. *Principles of Numerical Taxonomy*. W. H. Freeman and Company, San Francisco, 1963.

[Son11] Wayne Song. Typo-distance. 2011. URL: https://github.com/wsong/Typo-Distance.

[Sor58] Theodor Sorgenfrei. *Molluscan Assemblages from the Marine Middle Miocene of South Jutland and Their Environments*. Number 79 in 2. Danmarks Geologiske Undersøgelse, 1–503, 1958.

[Sta97] United States. *Using the Census Soundex*. Number 55 in General Information Leaflet. National Archives and Records Administration, Washington, D.C., 1997. URL: https://hdl.handle.net/2027/pur1.32754067050041.

[Sta07] United States. Soundex system: the soundex indexing system. 2007. URL: https://www.archives.gov/research/census/soundex.html.

[Ste34] J. F. Steffensen. On certain measures of dependence between statistical variables. *Biometrika*, 26(1/2):251–255, May 1934. doi:10.2307/2332058.

[SLaclavik15] Sam Steingold and Michal Laclavík. An information theoretic metric for multi-class categorization. Technical Report, Magnetic Media Online, 2015. URL: https://github.com/Magnetic/proficiency-metric/blob/master/paper/predeval.pdf.

[Ste14] Kevin L. Stern. Dameraulevenshteinalgorithm.java. 2014. URL: https://github.com/KevinStern/software-and-algorithms/blob/master/src/main/java/blogspot/software\T1\textbackslash{}_and\T1\textbackslash{}_algorithms/stern\T1\textbackslash{}_library/string/DamerauLevenshteinAlgorithm.java.

[Sti61] H. Edmund Stiles. The association factor in information retrieval. *Journal of the ACM*, 8(2):271–279, April 1961. doi:10.1145/321062.321074.

[SSK05] Giorgos Stoilos, Giorgos Stamou, and Stefanos Kollias. A string metric for ontology alignment. In *ISWC'05 Proceedings of the 4th international conference on The Semantic Web*, 624–637. Galway, Ireland, November 2005. doi:10.1007/11574620_45.

[Stu53] A. Stuart. The estimation and comparison of strengths of association in contingency tables. *Biometrika*, 40(1/2):105–110, June 1953. doi:10.2307/2333101.

[Szy34] Dezydery Szymkiewicz. Une contribution statistique à la géographie floristique. *Acta Societatis Botanicorum Poloniae*, 11(3):249–265, 1934. URL: https://pbsociety.org.pl/journals/index.php/asbp/article/download/asbp.1934.012/6710, doi:10.5586/asbp.1934.012.

[Sorensen48] Thorvald Sørensen. A method of establishing groups of equal amplitude in plant sociology based on similarity of species and its application to analyses of the vegetation on danish commons. *Kongelige Danske Videnskabernes Selskab*, 5(4):1–34, 1948. URL: http://www.royalacademy.dk/Publications/High/295\T1\textbackslash{}_S\T1\textbackslash{}%C3\T1\textbackslash{}%B8rensen,\T1\textbackslash{}%20Thorvald.pdf.

[Taf70] Robert L. Taft. *Name Search Techniques*. Special report (New York State Identification and Intelligence System). Bureau of Systems Development, New York State Identification and Intelligence System, 1970.

[Tan58] T. T. Tanimoto. An elementary mathematical theory of classification and prediction. Technical Report, IBM, 1958.

[Tar60] Kazimierz Tarwid. Szacowanie zbieznosci nisz ekologicznych gatunkow droga oceny prawdopodobienstwa spotykania sie ich w polowach. *Ekologia Polska, Seria B*, pages 115–130, 1960.

[Tic84] Walter F. Tichy. The string-to-string correction problem with block moves. *ACM Transactions on Computer Systems*, 2(4):309–321, November 1984. doi:10.1145/357401.357404.

[Tic] Ticki. Eudex: a blazingly fast phonetic reduction/hashing algorithm. URL: https://github.com/ticki/eudex.

[Tic16] Ticki. The eudex algorithm. December 2016. URL: http://ticki.github.io/blog/the-eudex-algorithm/.

[Tul97] Rodham E. Tulloss. Assessment of similarity indices for undesirable properties and a new tripartite similarity index based on cost functions. In Mary E. Palm and Ignacio H. Chapela, editors, *Mycology in Sustainable Development: Expanding Concepts, Vanishing Borders*, pages 122–143. Parkway Publishers, Inc., Boone, NC, 1997.

[TCLM88] W. A. Turner, G. Charton, F. Laville, and B. Michelet. Packaging information for peer review: new co-word analysis techniques. In *Handbook of Quantitative Studies of Science and Technology*. New Holland, 1988.

[Tve77] Amos Tversky. Features of similarity. *Psychological Review*, 84(4):327–352, 1977. URL: http://www.cogsci.ucsd.edu/~coulson/203/tversky-features.pdf, doi:10.1037/0033-295x.84.4.327.

[Ukk92] Esko Ukkonen. Approximate string-matching with q-grams and maximal matches. *Theoretical Computer Science*, 92(1):191–211, 1992. doi:10.1016/0304-3975(92)90143-4.

[Uph77] William B. Upholt. Estimation of DNA sequence divergence from comparison of restriction endonuclease digests. *Nucleic Acids Research*, 4(5):1257–1265, January 1977. doi:10.1093/nar/4.5.1257.

[VB12] Cihan Varol and Coskun Bayrak. Hybrid matching algorithm for personal names. *Journal of Data and Information Quality*, 3(4):8:1–8:18, September 2012. doi:10.1145/2348828.2348830.

[WF74] Robert A. Wagner and Michael J. Fischer. The string-to-string correction problem. *Journal of the ACM*, 21(1):168–173, January 1974. doi:10.1145/321796.321811.

[War08] Matthijs J. Warrens. *Similarity Coefficients for Binary Data: Properties of Coefficients, Coefficient Matrices, Multi-way Metrics and Multivariate Coefficients*. PhD thesis, Universiteit Leiden, Leiden, June 2008. URL: https://openaccess.leidenuniv.nl/bitstream/handle/1887/12987/Full\T1\textbackslash{}_thesis.pdf.

[Whid.] Simon White. How to strike a match. Web, Nd. The oldest version on Internet Archive was archived in 2004. URL: http://www.catalysoft.com/articles/StrikeAMatch.html.

[Whi52] R. H. Whittaker. A study of summer foliage insect communities in the great smoky mountains. *Ecological Monographs*, 22(1):1–44, January 1952. doi:10.2307/1948527.

[Whi82] Robert H. Whittaker. *Ordination of Plant Communities*. Volume 5 of Handbook of Vegetation Sciecne. Springer Netherlands, 1982.

[Wik18] Wikibooks. Algorithm implementation/strings/longest common substring. 2018. URL: https://en.wikibooks.org/wiki/Algorithm\T1\textbackslash{}_Implementation/Strings/Longest\T1\textbackslash{}_common\T1\textbackslash{}_substring\T1\textbackslash{}#Python.

[Wil05] Martin Wilz. Aspekte der kodierung phonetischer Ähnlichkeiten in deutschen eigennamen. Master's thesis, Universität zu Köln, Köln, 2005. URL: http://ifl.phil-fak.uni-koeln.de/sites/linguistik/Phonetik/import/Phonetik\T1\textbackslash{}_Files/Allgemeine\T1\textbackslash{}_Dateien/Martin\T1\textbackslash{}_Wilz.pdf.

[Win90] William E. Winkler. String comparator metrics and enhanced decision rules in the fellegi-sunter model of record linkage. Technical Report, U.S. Bureau of the Census, Statistical Research Division, Washington, D.C., 1990. URL: https://files.eric.ed.gov/fulltext/ED325505.pdf.

[WMJL94] William E. Winkler, George McLaughlin, Matthew A. Jaro, and Maureen Lync. Strcmp95.c. January 1994. URL: https://web.archive.org/web/20110629121242/http://www.census.gov/geo/msb/stand/strcmp.c.

[Xia13] Hua Xiang. *Similarity-based Virtual Screening: Effect of the Choice of Similarity Measure*. PhD thesis, The University of Sheffield, 2013. URL: http://etheses.whiterose.ac.uk/5662/1/Thesis\T1\textbackslash{}_Final.pdf.

[YJH+16] Ruiyu Yang, Yuxiang Jiang, Matthew W. Hahn, Elizabeth A. Houseworth, and Predrag Radivojac. New metrics for learning and inference on sets, ontologies, and functions. March 2016. URL: https://arxiv.org/abs/1603.06846v1.

[Yat34] Frank Yates. Contingency tables involving small numbers and the \$\chi \$2 Test. *Supplement to the Journal of the Royal Statistical Society*, 1(2):217–235, 1934. doi:10.2307/2983604.

[You50] William John Youden. Index for rating diagnostic tests. *Cancer*, 3(1):32–35, 1950. doi:10.1002/1097-0142(1950)3:1<32::aid-cncr2820030106>3.0.co;2-3.

[YB07] Li Yujian and Liu Bo. A normalized levenshtein distance metric. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(6):1091–1095, 2007. doi:10.1109/TPAMI.2007.1078.

[Yul12] G. Udny Yule. On the methods of measuring association between two attributes. *Journal of the Royal Statistical Society*, 1912. doi:10.2307/2340126.

[YK68] G. Udny Yule and Maurice G. Kendall. *An Introduction to the Theory of Statistics*. Griffin, London, 14 edition, 1968.

[Zac14] Siderite Zackwehdex. Super fast and accurate string distance algorithm: sift4. 2014. URL: https://siderite.blogspot.com/2014/11/super-fast-and-accurate-string-distance.html.

[Zed15] Jesper Zedlitz. Phonet4java phonet.java. 2015. URL: https://github.com/jze/phonet4java/blob/master/src/main/java/de/zedlitz/phonet4java/Phonet.java.

[ZD96] Justin Zobel and Philip Dart. Phonetic string matching: lessons from information retrieval. In *Proceedings of the 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '96, 166–172. New York, NY, USA, 1996. ACM. doi:10.1145/243199.243258.

[delHigueraMico08] Colin de la Higuera and Luisa Micó. A contextual normalised edit distance. In *First International Workshop on Similarity Search and Applications (sisap 2008)*. 2008. doi:10.1109/SISAP.2008.17.

[delPAngelesBailonM16] María del Pilar Angeles and Noemi Bailón-Miguel. Performance of spanish encoding functions during record linkage. In *DATA ANALYTICS 2016: The Fifth International Conference on Data Analysis*, 1–7. 2016. URL: https://core.ac.uk/download/pdf/55855695.pdf\T1\textbackslash{}#page=14.

[delPAngelesEGGM15] María del Pilar Angeles, Adrián Espino-Gamez, and Jonathan Gil-Moncada. Comparison of a modified spanish phonetic, soundex, and phonex coding functions during data matching process. In *2015 International Conference on Informatics, Electronics Vision (ICIEV)*, 1–5. June 2015. URL: https://www.researchgate.net/publication/285589803\T1\textbackslash{}_Comparison\T1\textbackslash{}_of\T1\textbackslash{}_a\T1\textbackslash{}_Modified\T1\textbackslash{}_Spanish\T1\textbackslash{}_Phonetic\T1\textbackslash{}_Soundex\T1\textbackslash{}_and\T1\textbackslash{}_Phonex\T1\textbackslash{}_coding\T1\textbackslash{}_functions\T1\textbackslash{}_during\T1\textbackslash{}_data\T1\textbackslash{}_matching\T1\textbackslash{}_process, doi:10.1109/ICIEV.2015.7334028.

[JPGTrust91] The J. Paul Getty Trust. Synoname. 1991. URL: http://www.cs.cmu.edu/Groups/AI/areas/nlp/misc/synoname/synoname.zip.

[vandMaarel69] Eddy van der Maarel. On the use of ordination model in phytosociology. *Vegetatio Acta Geobotanica*, 19(1–6):21–46, January 1969.

[vonRethS77] Hans-Peter von Reth and Hans-Jörg Schek. Eine zugriffsmethode für die phonetische Ähnlichkeitssuche. Technical Report 77.03.002, IBM Deutschland GmbH., 1977.

PYTHON MODULE INDEX

a

INDEX

A

B

C

D

E

F

G

H

I

N

O

P

Q

R

S

T

U

V

W

Y