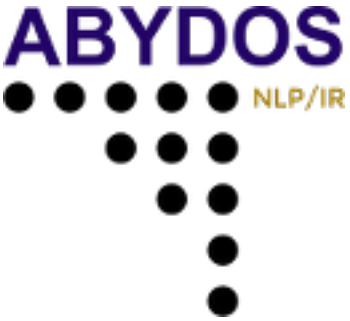
Abydos Documentation

Release 0.2.0

Chris Little

May 28, 2015

Contents



Contents:

CHAPTER 1

abydos package

1.1 Submodules

1.2 abydos.clustering module

abydos.clustering

The clustering module implements clustering algorithms such as:

- string fingerprinting

abydos.clustering.**fingerprint**(*phrase*)

String fingerprint

The fingerprint of a string is a string consisting of all of the unique words in a string, alphabetized & concatenated with intervening spaces

Parameters **phrase** (*str*) – the string from which to calculate the fingerprint

Returns the fingerprint of the phrase

Return type str

abydos.clustering.**mean_pairwise_similarity**(*collection*, *metric=<function sim>*, *mean_func=<function hmean>*, *symmetric=False*)

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** (*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 str

abydos.clustering.**omission_key**(*word*)
Omission key

The omission key of a word is defined in: Pollock, Joseph J. and Antonio Zamora. 1984. "Automatic Spelling Correction in Scientific and Scholarly Text." Communications of the ACM, 27(4). 358–368. <http://dl.acm.org/citation.cfm?id=358048>

Parameters **word** (*str*) – the word to transform into its omission key

Returns the omission key

Return type str

abydos.clustering.**phonetic_fingerprint**(*phrase*, *phonetic_algorithm=<function double_metaphone>*, **args*)
Return the phonetic fingerprint of a phrase

A phonetic fingerprint is identical to a standard string fingerprint, as implemented in abydos.clustering.fingerprint(), but performs the fingerprinting function after converting the string to its phonetic form, as determined by some phonetic algorithm.

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 abydos.phonetic.double_metaphone()
- **args** – additional arguments to pass to the phonetic algorithm, along with the phrase itself

Returns the phonetic fingerprint of the phrase

Return type str

abydos.clustering.**qgram_fingerprint**(*phrase*, *qval=2*, *start_stop=u''*)
Q-Gram fingerprint

A q-gram fingerprint is a string consisting of all of the unique q-grams in a string, alphabetized & concatenated.

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 abydos.util.qgram()

Returns the q-gram fingerprint of the phrase

Return type str

abydos.clustering.**skeleton_key**(*word*)
Skeleton key

The skeleton key of a word is defined in: Pollock, Joseph J. and Antonio Zamora. 1984. "Automatic Spelling Correction in Scientific and Scholarly Text." Communications of the ACM, 27(4). 358–368. <http://dl.acm.org/citation.cfm?id=358048>

Parameters **word** (*str*) – the word to transform into its skeleton key

Returns the skeleton key

Return type str

1.3 abydos.compression module

abydos.compression

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

- arithmetic coding functions (ac_train, ac_encode, & ac_decode)
- Burrows-Wheeler transform encoder/decoder (bwt_encode & bwt_decode)
- Run-Length Encoding encoder/decoder (rle_encode & rle_decode)

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

Decode the number to a string using the given statistics

This is based on Andrew Dalke's public domain implementation: http://code.activestate.com/recipes/306626/ It has been ported to use the abydos.util.Rational class.

Parameters

- **longval** – The first part of an encoded tuple from ac_encode
- **nbits** – The second part of an encoded tuple from ac_encode
- **probs** – A probability statistics dictionary generated by ac_train

Returns The arithmetically decoded text

Return type str

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

Encode a text using arithmetic coding with the provided probabilities

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

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

This is based on Andrew Dalke's public domain implementation: http://code.activestate.com/recipes/306626/ It has been ported to use the abydos.util.Rational class.

Parameters

- **text** – A string to encode
- **probs** – A probability statistics dictionary generated by ac_train

Returns The arithmetically coded text

Return type *tuple*

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

Generate a probability dict from the provided text

Text -> 0-order probability statistics as a dict

This is based on Andrew Dalke's public domain implementation: http://code.activestate.com/recipes/306626/ It has been ported to use the abydos.util.Rational class.

Parameters **text** – 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*

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

Return a word decoded from BWT form

The Burrows-Wheeler transform is an attempt at placing similar characters together to improve compression. This function reverses the transform. Cf. https://en.wikipedia.org/wiki/Burrows%E2%80%93Wheeler_transform

Parameters

- **code** – the word to transform from BWT form
- **terminator** – a character added to word to signal the end of the string

Returns word decoded by BWT

Return type str

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

Return the Burrows-Wheeler transformed form of a word

The Burrows-Wheeler transform is an attempt at placing similar characters together to improve compression. Cf. https://en.wikipedia.org/wiki/Burrows%E2%80%93Wheeler_transform

Parameters

- **word** – the word to transform using BWT
- **terminator** – a character to add to word to signal the end of the string

Returns word encoded by BWT

Return type str

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

Simple, crude run-length-encoding (RLE) decoder

Based on http://rosettacode.org/wiki/Run-length_encoding#Python

Digits 0-9 cannot have been in the original text.

Parameters

- **text** – a text string to decode
- **use_bwt** – boolean indicating whether to perform BWT decoding after RLE decoding

Returns word decoded by BWT

Return type str

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

Simple, crude run-length-encoding (RLE) encoder

Based on http://rosettacode.org/wiki/Run-length_encoding#Python

Digits 0-9 cannot be in text.

Parameters

- **text** – a text string to encode
- **use_bwt** – boolean indicating whether to perform BWT encoding before RLE encoding

Returns word decoded by BWT

Return type str

1.4 abydos.corpus module

abydos.corpus

The corpus class is a container for linguistic corpora and includes various functions for corpus statistics, language modeling, etc.

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

Bases: object

The 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.

docs()

Get 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 paragraphs in the corpus as a list of lists of lists of strs

Return type list(list(list(str)))

docs_of_words()

Get 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 list(list(str))

idf(*term*, *transform=None*)

Calculates the Inverse Document Frequency (IDF) of a term in the corpus.

Parameters

- **term** – the term to calculate the IDF of
- **transform** – a function to apply to each document term before checking for the presence of term

Returns the IDF

Return type float

paras()

Get 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 list(list(list(str)))

raw()

Get the raw corpus

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

Returns the raw corpus

Return type str

sents()
Get 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 list(list(str))

words()
Get the words in the corpus as a single list

Returns the words in the corpus as a list of strs

Return type list(str)

1.5 abydos.distance module

abydos.distance

The distance module implements string edit distance functions including:

- Levenshtein distance (incl. a [0, 1] normalized variant)
- Optimal String Alignment distance (incl. a [0, 1] normalized variant)
- Levenshtein-Damerau distance (incl. a [0, 1] normalized variant)
- Hamming distance (incl. a [0, 1] normalized variant)
- Tversky index
- Sørensen–Dice coefficient & distance
- Jaccard similarity coefficient & distance
- overlap similarity & distance
- Tanimoto coefficient & distance
- cosine similarity & distance
- Jaro distance
- Jaro-Winkler distance (incl. the strcmp95 algorithm variant)
- Longest common substring
- Ratcliff-Obershelp similarity & distance
- Match Rating Algorithm similarity
- Normalized Compression Distance (NCD) & similarity
- Monge-Elkan similarity & distance
- Matrix similarity
- Needleman-Wunsch score
- Smither-Waterman score
- Gotoh score

- Length similarity
- Prefix, Suffix, and Identity similarity & distance
- Modified Language-Independent Product Name Search (MLIPNS) similarity & distance
- Bag distance (incl. a [0, 1] normalized variant)
- Editex distance (incl. a [0, 1] normalized variant)
- TF-IDF similarity

Functions beginning with the prefixes 'sim' and 'dist' are guaranteed to be in the range [0, 1], and sim_X = 1 - dist_X since the two are complements. If a sim_X function is supplied identical src & tar arguments, it is guaranteed to return 1; the corresponding dist_X function is guaranteed to return 0.

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

Bag distance is: $max(|multiset(src) - multiset(tar)|, |multiset(tar) - multiset(src)|)$

Parameters **src, tar** (*str*) – two strings to be compared

Returns bag distance

Return type int

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

This computes the Damerau-Levenshtein distance. Cf. https://en.wikipedia.org/wiki/Damerau%E2%80%93Levenshtein_distance

Damerau-Levenshtein code based on Java code by Kevin L. Stern, under the MIT license: https://github.com/KevinStern/software-and-algorithms/blob/master/src/main/java/blogspot/software_and_algorithms/stern_libra

Parameters

- **src, tar** (*str*) – two strings to be compared
- **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)

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

This is a generalized function for calling other distance functions.

Parameters

- **src, tar** (*str*) – two strings to be compared
- **method** (*function*) – specifies the similarity metric (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

abydos.distance.**dist_bag**(*src*, *tar*)
Return the normalized bag distance of two strings.

Bag distance is normalized by dividing by $max(|src|, |tar|)$.

Parameters **src, tar** (*str*) – two strings to be compared

Returns normalized bag distance

Return type float

abydos.distance.**dist_compression**(*src*, *tar*, *compressor=u'bz2'*, *probs=None*)
normalized compression distance (NCD)

Cf. https://en.wikipedia.org/wiki/Normalized_compression_distance#Normalized_compression_distance

Parameters

- **src, tar** (*str*) – two strings to be compared
- **compressor** (*str*) – a compression scheme to use for the similarity calculation, from the following:
  - *zlib* – standard zlib/gzip
  - *bz2* – bzip2
  - *lzma* – Lempel–Ziv–Markov chain algorithm
  - *arith* – arithmetic coding
  - *rle* – run-length encoding
  - *bwtrle* – Burrows-Wheeler transform followed by run-length encoding
- **probs** (*doct*) – a dictionary trained with ac_train (for the arith compressor only)

Returns compression distance

Return type float

abydos.distance.**dist_cosine**(*src*, *tar*, *qval=2*)
cosine distance

Cosine distance is the complement of cosine similarity: $dist_{cosine} = 1 - sim_{cosine}$

Parameters

- **src, tar** (*str*) – two strings to be compared (or QGrams/Counter objects)
- **qval** (*int*) – the length of each q-gram; 0 or None for non-q-gram version

Returns cosine distance

Return type float

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

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

Parameters

- **src, tar** (*str*) – two strings to be compared
- **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 normalized Damerau-Levenshtein distance

Return type float

abydos.distance.**dist_dice**(*src*, *tar*, *qval=2*)
Sørensen–Dice distance

Sørensen–Dice distance is the complemenjt of the Sørensen–Dice coefficient: $dist_{dice} = 1 - sim_{dice}$

Parameters

- **src, tar** (*str*) – two strings to be compared (or QGrams/Counter objects)
- **qval** (*int*) – the length of each q-gram; 0 or None for non-q-gram version

Returns Sørensen–Dice distance

Return type float

abydos.distance.**dist_editex**(*src*, *tar*, *cost=(0, 1, 2)*, *local=False*)
Editex distance normalized to the interval [0, 1]

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, tar** (*str*) – two strings to be compared
- **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 float

abydos.distance.**dist_hamming**(*src*, *tar*, *difflens=True*)
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 difflens is set to False, in which case an exception is raised).

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

Parameters

- **src, tar** (*str*) – two strings to be compared
- **allow_different_lengths** (*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 normalized Hamming distance

Return type float

abydos.distance.**dist_ident**(*src*, *tar*)
identity distance

This is 0 if the two strings are identical, otherwise 1, i.e. $dist_{identity} = 1 - sim_{identity}$

Parameters **src, tar** (*str*) – two strings to be compared

Returns indentity distance

Return type int

abydos.distance.**dist_jaccard**(*src*, *tar*, *qval=2*)

Jaccard distance

Jaccard distance is the complement of the Jaccard similarity coefficient: $dist_{Jaccard} = 1 - sim_{Jaccard}$

Parameters

- **src, tar** (*str*) – two strings to be compared (or QGrams/Counter objects)
- **qval** (*int*) – the length of each q-gram; 0 or None for non-q-gram version

Returns Jaccard distance

Return type float

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

Jaro(-Winkler) distance

Jaro-Winkler distance is 1 - the Jaro-Winkler similarity

Parameters

- **src, tar** (*str*) – two strings to be compared
- **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

The following arguments apply only when mode is 'winkler':

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."
- **boost_threshold** (*float*) – a value between 0 and 1, below which the Winkler boost is not applied (defaults to 0.7)
- **scaling_factor** (*float*) – a value between 0 and 0.25, indicating by how much to boost scores for matching prefixes (defaults to 0.1)

Returns Jaro or Jaro-Winkler distance

Return type float

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

longest common subsequence distance ($dist_{LCSseq}$)

This employs the LCSseq function to derive a similarity metric: $dist_{LCSseq}(s,t) = 1 - sim_{LCSseq}(s,t)$

Parameters **src, tar** (*str*) – two strings to be compared

Returns LCSseq distance

Return type float

abydos.distance.**dist_lcsstr**(*src*, *tar*)
longest common substring distance ($dist_{LCSstr}$)

This employs the LCS function to derive a similarity metric: $dist_{LCSstr}(s,t) = 1 - sim_{LCSstr}(s,t)$

Parameters **src, tar** (*str*) – two strings to be compared

Returns LCSstr distance

Return type float

abydos.distance.**dist_length**(*src*, *tar*)
length distance

Length distance is the complement of length similarity: $dist_{length} = 1 - sim_{length}$

Parameters **src, tar** (*str*) – two strings to be compared

Returns length distance

Return type float

abydos.distance.**dist_levenshtein**(*src*, *tar*, *mode=u'lev'*, *cost=(1, 1, 1, 1)*)
Levenshtein distance normalized to the interval [0, 1]

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, tar** (*str*) – two strings to be compared
- **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
  - 'dam' computes the Damerau-Levenshtein distance, in which edits may include inserts, deletes, substitutions, and transpositions and substrings may undergo repeated edits
- **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 normalized Levenshtein distance

Return type float

abydos.distance.**dist_mlipns**(*src*, *tar*, *threshold=0.25*, *maxmismatches=2*)
Modified Language-Independent Product Name Search (MLIPNS)

MLIPNS distance is the complement of MLIPNS similarity: $dist_{MLIPNS} = 1 - sim_{MLIPNS}$

This function returns only 0.0 (distant) or 1.0 (not distant)

Parameters

- **src, tar** (*str*) – two strings to be compared
- **threshold** (*float*) – a number [0, 1] indicating the maximum similarity score, below which the strings are considered 'similar' (0.25 by default)

- **maxmismatches** (*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

abydos.distance.**dist_monge_elkan**(*src*, *tar*, *sim_func=<function sim_levenshtein>*, *symmetric=False*)

Monge-Elkan distance

Monge-Elkan is defined in: Monge, Alvaro E. and Charles P. Elkan. 1996. "The field matching problem: Algorithms and applications." KDD-9 Proceedings. http://www.aaai.org/Papers/KDD/1996/KDD96-044.pdf

Note: Monge-Elkan is NOT a symmetric similarity algoritm. Thus, the distance between src and tar is not necessarily equal to the distance between tar and src. If the sym argument is True, a symmetric value is calculated, at the cost of doubling the computation time (since the $sim_{Monge-Elkan}(src, tar)$ and $sim_{Monge-Elkan}(tar, src)$ are both calculated and then averaged).

Parameters

- **src, tar** (*str*) – two strings to be compared
- **sim_func** (*function*) – the internal similarity metric to emply
- **symmetric** (*bool*) – return a symmetric similarity measure

Returns Monge-Elkan distance

Return type float

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

normalized Match Rating Algorithm distance

MRA distance is the complement of MRA similarity: $dist_{MRA} = 1 - sim_{MRA}$

Parameters **src, tar** (*str*) – two strings to be compared

Returns normalized MRA distance

Return type float

abydos.distance.**dist_overlap**(*src*, *tar*, *qval=2*)

overlap distance

Overlap distance is the complement of the overlap coefficient: $sim_{overlap} = 1 - dist_{overlap}$

Parameters

- **src, tar** (*str*) – two strings to be compared (or QGrams/Counter objects)
- **qval** (*int*) – the length of each q-gram; 0 or None for non-q-gram version

Returns overlap distance

Return type float

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

prefix distance

Prefix distance is the complement of prefix similarity: $dist_{prefix} = 1 - sim_{prefix}$

Parameters **src, tar** (*str*) – two strings to be compared

Returns prefix distance

Return type float

abydos.distance.**dist_ratcliff_obershelp**(*src*, *tar*)
Ratcliff-Obershelp distance

Ratcliff-Obershelp distance the complement of Ratcliff-Obershelp similarity: $dist_{Ratcliff-Obershelp} = 1 - sim_{Ratcliff-Obershelp}$

Parameters **src, tar** (*str*) – two strings to be compared

Returns Ratcliffe-Obershelp distance

Return type float

abydos.distance.**dist_strcmp95**(*src*, *tar*, *long_strings=False*)
strcmp95 distance

strcmp95 distance is 1 - strcmp95 similarity

Parameters

- **src, tar** (*str*) – two strings to be compared
- **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

abydos.distance.**dist_suffix**(*src*, *tar*)
suffix distance

Suffix distance is the complement of suffix similarity: $dist_{suffix} = 1 - sim_{suffix}$

Parameters **src, tar** (*str*) – two strings to be compared

Returns suffix distance

Return type float

abydos.distance.**dist_tversky**(*src*, *tar*, *qval=2*, *alpha=1*, *beta=1*, *bias=None*)
Tversky distance

Tversky distance is the complement of the Tvesrsky index (similarity): $dist_{Tversky} = 1 - sim_{Tversky}$

The symmetric variant is defined in Jiminez, Sergio, Claudio Becerra, and Alexander Gelbukh. 2013. SOFTCARDINALITY-CORE: Improving Text Overlap with Distributional Measures for Semantic Textual Similarity. This is activated by specifying a bias parameter. Cf. http://aclweb.org/anthology/S/S13/S13-1028.pdf

Parameters

- **src, tar** (*str*) – two strings to be compared (or QGrams/Counter objects)
- **qval** (*int*) – the length of each q-gram; 0 or None for non-q-gram version
- **alpha, beta** (*float*) – two Tversky index parameters as indicated in the description below

Returns Tversky distance

Return type float

abydos.distance.**editex**(*src*, *tar*, *cost=(0, 1, 2)*, *local=False*)
Editex distance

As described on pages 3 & 4 of Zobel, Justin and Philip Dart. 1996. Phonetic string matching: Lessons from information retrieval. In: Proceedings of the ACM-SIGIR Conference on Research and Development in Information Retrieval, Zurich, Switzerland. 166–173. http://goanna.cs.rmit.edu.au/~jz/fulltext/sigir96.pdf

The local variant is based on Ring, Nicholas and Alexandra L. Uitdenbogerd. 2009. Finding 'Lucy in Disguise': The Misheard Lyric Matching Problem. In: Proceedings of the 5th Asia Information Retrieval Symposium, Sapporo, Japan. 157-167. http://www.seg.rmit.edu.au/research/download.php?manuscript=404

Parameters

- **src, tar** (*str*) – two strings to be compared
- **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

abydos.distance.**gotoh**(*src*, *tar*, *gap_open=1*, *gap_ext=0.4*, *sim_func=<function sim_ident>*)
Gotoh score

Gotoh's algorithm is essentially Needleman-Wunsch with affine gap penalties: https://www.cs.umd.edu/class/spring2003/cmsc838t/papers/gotoh1982.pdf

Parameters

- **src, tar** (*str*) – two strings to be compared
- **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 (in fact dependent on the gap_cost & return value of sim_func)

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

Hamming distance 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.

Parameters

- **src, tar** (*str*) – two strings to be compared
- **allow_different_lengths** (*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

abydos.distance.**lcsseq**(*src*, *tar*)
longest common subsequence (LCSseq)

Based on the dynamic programming algorithm from http://rosettacode.org/wiki/Longest_common_subsequence#Dynamic_Progra
This is licensed GFDL 1.2

Modifications include: conversion to a numpy array in place of a list of lists

Parameters **src, tar** (*str*) – two strings to be compared

Returns the longes common subsequence

Return type str

abydos.distance.**lcsstr**(*src*, *tar*)
longest common substring (LCSstr)

Based on the code from https://en.wikibooks.org/wiki/Algorithm_Implementation/Strings/Longest_common_substring#Python
This is licensed Creative Commons: Attribution-ShareAlike 3.0

Modifications include:

•conversion to a numpy array in place of a list of lists

•conversion to Python 2/3-safe _range from xrange

Parameters **src, tar** (*str*) – two strings to be compared

Returns the longes common substring

Return type float

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

This is the standard edit distance measure. Cf. https://en.wikipedia.org/wiki/Levenshtein_distance

Two additional variants: optimal string alignment (aka restricted Damerau-Levenshtein distance) and the Damerau-Levenshtein distance are also supported. Cf. https://en.wikipedia.org/wiki/Damerau%E2%80%93Levenshtein_distance

The ordinary Levenshtein & Optimal String Alignment distance both employ the Wagner-Fischer dynamic programming algorithm. Cf. https://en.wikipedia.org/wiki/Wagner%E2%80%93Fischer_algorithm

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

Parameters

- **src, tar** (*str*) – two strings to be compared
- **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
  - 'dam' computes the Damerau-Levenshtein distance, in which edits may include inserts, deletes, substitutions, and transpositions and substrings may undergo repeated edits
- **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)

abydos.distance.**mra_compare**(*src*, *tar*)
Western Airlines Surname Match Rating Algorithm comparison rating

A description of the algorithm can be found on page 18 of https://archive.org/details/accessingindivid00moor

Parameters **src, tar** (*str*) – two strings to be compared

Returns MRA comparison rating

Return type int

abydos.distance.**needleman_wunsch**(*src*, *tar*, *gap_cost=1*, *sim_func=<function sim_ident>*)
Needleman-Wunsch score

This is the standard edit distance measure.

Cf. https://en.wikipedia.org/wiki/Needleman–Wunsch_algorithm

Cf. http://csb.stanford.edu/class/public/readings/Bioinformatics_I_Lecture6/Needleman_Wunsch_JMB_70_Global_alignment.pd

Parameters

- **src, tar** (*str*) – two strings to be compared
- **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 int (in fact dependent on the gap_cost & return value of sim_func)

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

This is a generalized function for calling other similarity functions.

Parameters

- **src, tar** (*str*) – two strings to be compared
- **method** (*function*) – specifies the similarity metric (Levenshtein by default)

Returns similarity according to the specified function

Return type float

abydos.distance.**sim_bag**(*src*, *tar*)
normalized bag similarity

Normalized bag similarity is the complement of normalized bag distance: $sim_{bag} = 1 - dist_{bag}$

Parameters **src, tar** (*str*) – two strings to be compared

Returns normalized bag similarity

Return type float

abydos.distance.**sim_compression**(*src*, *tar*, *compressor=u'bz2'*, *probs=None*)
normalized compression similarity (NCS)

Normalized compression similarity is the complement of normalized compression distance: $sim_{NCS} = 1 - dist_{NCD}$

Parameters

- **src, tar** (*str*) – two strings to be compared

- **compressor** (*str*) – a compression scheme to use for the similarity calculation:
  - *zlib* – standard zlib/gzip
  - *bz2* – bzip2
  - *lzma* – Lempel–Ziv–Markov chain algorithm
  - *arith* – arithmetic coding
  - *rle* – run-length encoding
  - *bwtrle* – Burrows-Wheeler transform followed by run-length encoding
- **probs** (dict) – a dictionary trained with ac_train (for the arith compressor only)

Returns compression similarity

Return type float

abydos.distance.**sim_cosine**(*src*, *tar*, *qval=2*)

cosine similarity (Ochiai coefficient)

For two sets X and Y, the cosine similarity (Ochiai coefficient) is: $sim_{cosine}(X, Y) = \frac{|X \cap Y|}{\sqrt{|X| \cdot |Y|}}$

Parameters

- **src, tar** (*str*) – two strings to be compared (or QGrams/Counter objects)
- **qval** (*int*) – the length of each q-gram; 0 or None for non-q-gram version

Returns cosine similarity

Return type float

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

Levenshtein similarity normalized to the interval [0, 1]

Damerau-Levenshtein similarity the complement of Damerau-Levenshtein distance: $sim_{Damerau} = 1 - dist_{Damerau}$

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

Parameters

- **src, tar** (*str*) – two strings to be compared
- **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 normalized Damerau-Levenshtein similarity

Return type float

abydos.distance.**sim_dice**(*src*, *tar*, *qval=2*)

Sørensen–Dice coefficient

For two sets X and Y, the Sørensen–Dice coefficient is $sim_{dice}(X, Y) = \frac{2 \cdot |X \cap Y|}{|X| + |Y|}$

This is identical to the Tanimoto similarity coefficient and the Tversky index for $\alpha = \beta = 0.5$

Parameters

- **src, tar** (*str*) – two strings to be compared (or QGrams/Counter objects)
- **qval** (*int*) – the length of each q-gram; 0 or None for non-q-gram version

Returns Sørensen–Dice similarity

Return type float

abydos.distance.**sim_editex**(*src*, *tar*, *cost=(0, 1, 2)*, *local=False*)
Editex similarity normalized to the interval [0, 1]

The Editex similarity is the complement of Editex distance $sim_{Editex} = 1 - dist_{Editex}$

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

Parameters

- **src, tar** (*str*) – two strings to be compared
- **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 float

abydos.distance.**sim_hamming**(*src*, *tar*, *difflens=True*)
Hamming similarity normalized to the interval [0, 1]

Hamming similarity is the complement of normalized Hamming distance: $sim_{Hamming} = 1 - distHamming$

Provided that difflens==True, the Hamming similarity is identical to the Language-Independent Product Name Search (LIPNS) similarity score. For further information, see the sim_mlipns documentation.

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

Parameters

- **src, tar** (*str*) – two strings to be compared
- **allow_different_lengths** (*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 normalized Hamming similarity

Return type float

abydos.distance.**sim_ident**(*src*, *tar*)
identity similarity

This is 1 if the two strings are identical, otherwise 0.

Parameters **src, tar** (*str*) – two strings to be compared

Returns identity similarity

Return type int

abydos.distance.**sim_jaccard**(*src*, *tar*, *qval=2*)
Jaccard similarity coefficient

For two sets X and Y, the Jaccard similarity coefficient is $sim_{jaccard}(X, Y) = \frac{|X \cap Y|}{|X \cup Y|}$

This is identical to the Tanimoto similarity coefficient and the Tversky index for $\alpha = \beta = 1$

Parameters

- **src, tar** (*str*) – two strings to be compared (or QGrams/Counter objects)

- **qval** (*int*) – the length of each q-gram; 0 or None for non-q-gram version

Returns Jaccard similarity

Return type float

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

Jaro(-Winkler) distance

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

Parameters

- **src, tar** (*str*) – two strings to be compared
- **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

The following arguments apply only when mode is 'winkler':

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."
- **boost_threshold** (*float*) – a value between 0 and 1, below which the Winkler boost is not applied (defaults to 0.7)
- **scaling_factor** (*float*) – a value between 0 and 0.25, indicating by how much to boost scores for matching prefixes (defaults to 0.1)

Returns Jaro or Jaro-Winkler similarity

Return type float

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

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, tar** (*str*) – two strings to be compared

Returns LCSseq similarity

Return type float

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

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, tar** (*str*) – two strings to be compared

Returns LCSstr similarity

Return type float

abydos.distance.**sim_length**(*src*, *tar*)
length similarity

This is the ratio of the length of the shorter string to the longer.

Parameters **src, tar** (*str*) – two strings to be compared

Returns length similarity

Return type float

abydos.distance.**sim_levenshtein**(*src*, *tar*, *mode=u'lev'*, *cost=(1, 1, 1, 1)*)
Levenshtein similarity normalized to the interval [0, 1]

Levenshtein similarity the complement of Levenshtein distance: $sim_{Levenshtein} = 1 - dist_{Levenshtein}$

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

Parameters

- **src, tar** (*str*) – two strings to be compared
- **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
  - 'dam' computes the Damerau-Levenshtein distance, in which edits may include inserts, deletes, substitutions, and transpositions and substrings may undergo repeated edits
- **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 normalized Levenshtein similarity

Return type float

abydos.distance.**sim_matrix**(*src*, *tar*, *mat=None*, *mismatch_cost=0*, *match_cost=1*, *symmetric=True*, *alphabet=None*)
matrix similarity

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, tar** (*str*) – two strings to be compared
- **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

abydos.distance.**sim_mlipns**(*src*, *tar*, *threshold=0.25*, *maxmismatches=2*)
Modified Language-Independent Product Name Search (MLIPNS)

The MLIPNS algorithm is described in Shannaq, Boumedyen A. N. and Victor V. Alexandrov. 2010. "Using Product Similarity for Adding Business." Global Journal of Computer Science and Technology. 10(12). 2-8. http://www.sial.iias.spb.su/files/386-386-1-PB.pdf

This function returns only 1.0 (similar) or 0.0 (not similar).

LIPNS similarity is identical to normalized Hamming similarity.

Parameters

- **src, tar** (*str*) – two strings to be compared
- **threshold** (*float*) – a number [0, 1] indicating the maximum similarity score, below which the strings are considered 'similar' (0.25 by default)
- **maxmismatches** (*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

abydos.distance.**sim_monge_elkan**(*src*, *tar*, *sim_func=<function sim_levenshtein>*, *symmetric=False*)
Monge-Elkan similarity

Monge-Elkan is defined in: Monge, Alvaro E. and Charles P. Elkan. 1996. "The field matching problem: Algorithms and applications." KDD-9 Proceedings. http://www.aaai.org/Papers/KDD/1996/KDD96-044.pdf

Note: Monge-Elkan is NOT a symmetric similarity algoritm. Thus, the similarity of src to tar is not necessarily equal to the similarity of tar to src. If the sym argument is True, a symmetric value is calculated, at the cost of doubling the computation time (since the $sim_{Monge-Elkan}(src, tar)$ and $sim_{Monge-Elkan}(tar, src)$ are both calculated and then averaged).

Parameters

- **src, tar** (*str*) – two strings to be compared
- **sim_func** (*function*) – the internal similarity metric to emply
- **symmetric** (*bool*) – return a symmetric similarity measure

Returns Monge-Elkan similarity

Return type float

abydos.distance.**sim_mra**(*src*, *tar*)
normalized Match Rating Algorithm similarity

This is the MRA normalized to $[0, 1]$, given that MRA itself is constrained to the range $[0, 6]$.

Parameters **src, tar** (*str*) – two strings to be compared

Returns normalized MRA similarity

Return type float

abydos.distance.**sim_overlap**(*src*, *tar*, *qval=2*)
overlap coefficient

For two sets X and Y, the overlap coefficient is $sim_{overlap}(X, Y) = \frac{|X \cap Y|}{min(|X|, |Y|)}$

Parameters

- **src, tar** (*str*) – two strings to be compared (or QGrams/Counter objects)
- **qval** (*int*) – the length of each q-gram; 0 or None for non-q-gram version

Returns overlap similarity

Return type float

abydos.distance.**sim_prefix**(*src*, *tar*)
prefix similarity

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, tar** (*str*) – two strings to be compared

Returns prefix similarity

Return type float

abydos.distance.**sim_ratcliff_obershelp**(*src*, *tar*)
Ratcliff-Obershelp similarity

This follows the Ratcliff-Obershelp algorithm 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

Parameters **src, tar** (*str*) – two strings to be compared

Returns Ratcliff-Obserhelp similarity

Return type float

abydos.distance.**sim_strcmp95**(*src*, *tar*, *long_strings=False*)
strcmp95 similarity

This is a Python translation of the C code for strcmp95: http://web.archive.org/web/20110629121242/http://www.census.gov/geo/n
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.

Parameters

- **src, tar** (*str*) – two strings to be compared
- **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

abydos.distance.**sim_suffix**(*src*, *tar*)
suffix similarity

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, tar** (*str*) – two strings to be compared

Returns suffix similarity

Return type float

abydos.distance.**sim_tanimoto**(*src*, *tar*, *qval=2*)
Tanimoto similarity

For two sets X and Y, the Tanimoto similarity coefficient is $sim_{Tanimoto}(X, Y) = \frac{|X \cap Y|}{|X \cup Y|}$ This is identical to the Jaccard similarity coefficient and the Tversky index for $\alpha = \beta = 1$

Parameters

- **src, tar** (*str*) – two strings to be compared (or QGrams/Counter objects)
- **qval** (*int*) – the length of each q-gram; 0 or None for non-q-gram version

Returns Tanimoto similarity

Return type float

abydos.distance.**sim_tfidf**(*src*, *tar*, *qval=2*, *docs_src=None*, *docs_tar=None*)
TF-IDF similarity

This is chiefly based on the "Formal Definition of TF/IDF Distance" at: http://alias-i.com/lingpipe/docs/api/com/aliasi/spell/TfIdfDistance.html

Parameters

- **src, tar** (*str*) – two strings to be compared (or QGrams/Counter objects)
- **qval** (*int*) – the length of each q-gram; 0 or None for non-q-gram version
- **docs_src** (*Counter*) – a Counter object or string representing the document corpus for the src string
- **docs_tar** (*Counter*) – a Counter object or string representing the document corpus for the tar string (or set to None to use the docs_src for both)

Returns TF-IDF similarity

Return type float

abydos.distance.**sim_tversky**(*src*, *tar*, *qval=2*, *alpha=1*, *beta=1*, *bias=None*)
Tversky index

The Tversky index is defined as: For two sets X and Y: $sim_{Tversky}(X, Y) = \frac{|X \cap Y|}{|X \cap Y| + \alpha|X - Y| + \beta|Y - X|}$

Cf. https://en.wikipedia.org/wiki/Tversky_index

$\alpha = \beta = 1$ is equivalent to the Jaccard & Tanimoto similarity coefficients.

$\alpha = \beta = 0.5$ is equivalent to the Sørensen-Dice similarity coefficient.

Unequal α and β 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:

- $\alpha and \beta > 1$ emphsize unique contributions over the intersection
- $\alpha and \beta < 1$ emphsize the intersection over unique contributions

The symmetric variant is defined in Jiminez, Sergio, Claudio Becerra, and Alexander Gelbukh. 2013. SOFTCARDINALITY-CORE: Improving Text Overlap with Distributional Measures for Semantic Textual Similarity. This is activated by specifying a bias parameter. Cf. http://aclweb.org/anthology/S/S13/S13-1028.pdf

Parameters

- **src, tar** (*str*) – two strings to be compared (or QGrams/Counter objects)
- **qval** (*int*) – the length of each q-gram; 0 or None for non-q-gram version
- **alpha, beta** (*float*) – two Tversky index parameters as indicated in the description below

Returns Tversky similarity

Return type float

abydos.distance.**smith_waterman**(*src*, *tar*, *gap_cost=1*, *sim_func=<function sim_ident>*)

Smith-Waterman score

This is the standard edit distance measure.

Cf. https://en.wikipedia.org/wiki/Smith–Waterman_algorithm

Parameters

- **src, tar** (*str*) – two strings to be compared
- **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 int (in fact dependent on the gap_cost & return value of sim_func)

abydos.distance.**tanimoto**(*src*, *tar*, *qval=2*)

Tanimoto distance

Tanimoto distance is $-log_2 sim_{Tanimoto}$

Parameters

- **src, tar** (*str*) – two strings to be compared (or QGrams/Counter objects)
- **qval** (*int*) – the length of each q-gram; 0 or None for non-q-gram version

Returns Tanimoto distance

Return type float

1.6 abydos.ngram module

abydos.ngram

The NGram class is a container for an n-gram corpus

class abydos.ngram.**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'].

corpus_importer(*corpus*)
Fill in self.ngcorpus from a Corpus argument

Parameters **corpus** – The Corpus from which to initialize the n-gram corpus

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

ngcorpus = Counter()

1.7 abydos.phones module

abydos.phones

The phones module implements ...

abydos.phones.**cmp_features**(*feat1*, *feat2*)
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, feat2** (*int*) – Two feature bundles to compare

abydos.phones.**get_feature**(*vector*, *feature*)
Get feature

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: 'consonantal', 'sonorant', 'syllabic', 'labial', 'round', 'coronal', 'anterior', 'distributed', 'dorsal', 'high', 'low', 'back', 'tense', 'pharyngeal', 'ATR', 'voice', 'spread_glottis', 'constricted_glottis', 'continuant', 'strident', 'lateral', 'delayed_release', 'nasal'

Returns a list indicating presence/absence/neutrality with respect to the feature

Return type list(int)

abydos.phones.**ipa_to_features**(*ipa*)

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(int)

1.8 abydos.phonetic module

abydos.phonetic

The phonetic module implements phonetic algorithms including:

- Robert C. Russell's Index
- American Soundex
- Daitch-Mokotoff Soundex
- Kölner Phonetik
- NYSIIS
- Match Rating Algorithm
- Metaphone
- Double Metaphone
- Caverphone
- Alpha Search Inquiry System
- Fuzzy Soundex
- Phonex
- Phonem
- Phonix
- SfinxBis
- phonet
- Standardized Phonetic Frequency Code
- Beider-Morse Phonetic Matching

abydos.phonetic.**alpha_sis**(*word*, *maxlength=14*)

IBM Alpha Search Inquiry System

Based on the algorithm described in "Accessing individual records from personal data files using non-unique identifiers" / Gwendolyn B. Moore, et al.; prepared for the Institute for Computer Sciences and Technology, National Bureau of Standards, Washington, D.C (1977): https://archive.org/stream/accessingindivid00moor#page/15/mode/1up

A collection is necessary 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
- **maxlength** (*int*) – the length of the code returned (defaults to 14)

Returns the Alpha SIS value

Return type *tuple*

abydos.phonetic.**bmpm**(*word*, *language_arg=0*, *name_mode=u'gen'*, *match_mode=u'approx'*, *concat=False*, *filter_langs=False*)

Beider-Morse Phonetic Matching algorithm

The Beider-Morse Phonetic Matching algorithm is described at: http://stevemorse.org/phonetics/bmpm.htm The reference implementation is licensed under GPLv3 and available at: http://stevemorse.org/phoneticinfo.htm

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'
  - 'polish'
  - 'portuguese'
  - 'romanian'
  - 'russian'
  - 'spanish'
  - 'turkish'
  - 'germandjsg'
  - 'polishdjskp'
  - 'russiandjsre'
- **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 BMPM value(s)

Return type *tuple*

abydos.phonetic.**caverphone**(*word*, *version=2*)
Caverphone

A description of version 1 of the algorithm can be found at: http://caversham.otago.ac.nz/files/working/ctp060902.pdf

A description of version 2 of the algorithm can be found at: http://caversham.otago.ac.nz/files/working/ctp150804.pdf

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

abydos.phonetic.**dm_soundex**(*word*, *maxlength=6*, *reverse=False*, *zero_pad=True*)
Daitch-Mokotoff Soundex

Returns values of a word as a set. A collection is necessary since there can be multiple values for a single word.

Parameters

- **word** – the word to transform
- **maxlength** – the length of the code returned (defaults to 6)
- **reverse** – reverse the word before computing the selected Soundex (defaults to False); This results in "Reverse Soundex"
- **zero_pad** – pad the end of the return value with 0s to achieve a maxlength string

Returns the Daitch-Mokotoff Soundex value

Return type str

abydos.phonetic.**double_metaphone**(*word*, *maxlength=inf*)
Double Metaphone

Based on Lawrence Philips' (Visual) C++ code from 1999: http://aspell.net/metaphone/dmetaph.cpp

Parameters

- **word** – the word to transform
- **maxlength** – 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*

abydos.phonetic.**fuzzy_soundex**(*word*, *maxlength=5*, *zero_pad=True*)
Fuzzy Soundex

Fuzzy Soundex is an algorithm derived from Soundex, defined in: Holmes, David and M. Catherine McCabe. "Improving Precision and Recall for Soundex Retrieval." http://wayback.archive.org/web/20100629121128/http://www.ir.iit.edu/publications/downloads/IEEESoundexV5.pdf

Parameters

- **word** (*str*) – the word to transform
- **maxlength** (*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 maxlength string

Returns the Fuzzy Soundex value

Return type str

abydos.phonetic.**koelner_phonetik**(*word*)
Kölner Phonetik (numeric output)

Based on the algorithm described at https://de.wikipedia.org/wiki/Kölner_Phonetik

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

abydos.phonetic.**koelner_phonetik_alpha**(*word*)
Kölner Phonetik (alphabetic output)

Parameters **word** (*str*) – the word to transform

Returns the Kölner Phonetik value as an alphabetic string

Return type str

abydos.phonetic.**koelner_phonetik_num_to_alpha**(*num*)
Convert Kölner Phonetik from numeric to alphabetic

Parameters **num** (*str*) – a numeric Kölner Phonetik representation

Returns an alphabetic representation of the same word

Return type str

abydos.phonetic.**metaphone**(*word*, *maxlength=inf*)
Metaphone

Based on Lawrence Philips' Pick BASIC code from 1990: http://aspell.net/metaphone/metaphone.basic This incorporates some corrections to the above code, particularly some of those suggested by Michael Kuhn in: http://aspell.net/metaphone/metaphone-kuhn.txt

Parameters

- **word** (*str*) – the word to transform
- **maxlength** (*int*) – the maximum length of the returned Metaphone code (defaults to unlimited, but in Philips' original implementation this was 4)

Returns the Metaphone value

Return type str

abydos.phonetic.**mra**(*word*)
Western Airlines Surname Match Rating Algorithm personal numeric identifier (PNI)

A description of the algorithm can be found on page 18 of https://archive.org/details/accessingindivid00moor

Parameters **word** (*str*) – the word to transform

Returns the MRA PNI

Return type str

abydos.phonetic.**nysiis**(*word*, *maxlength=6*)

New York State Identification and Intelligence System (NYSIIS)

A description of the algorithm can be found at https://en.wikipedia.org/wiki/New_York_State_Identification_and_Intelligence_Sys

Parameters

- **word** (*str*) – the word to transform
- **maxlength** (*int*) – the maximum length (default 6) of the code to return

Returns the NYSIIS value

Return type str

abydos.phonetic.**phonem**(*word*)

Phonem

Phonem is defined in Wilde, Georg and Carsten Meyer. 1999. "Doppelgaenger gesucht - Ein Programm fuer kontextsensitive phonetische Textumwandlung." ct Magazin fuer Computer & Technik 25/1999.

This version is based on the Perl implementation documented at: http://phonetik.phil-fak.uni-koeln.de/fileadmin/home/ritters/Allgemeine_Dateien/Martin_Wilz.pdf It includes some enhancements presented in the Java port at: https://github.com/dcm4che/dcm4che/blob/master/dcm4che-soundex/src/main/java/org/dcm4che3/soundex/Phonem.java

Phonem is intended chiefly for German names/words.

Parameters **word** (*str*) – the word to transform

Returns the Phonem value

Return type str

abydos.phonetic.**phonet**(*word*, *mode=1*, *lang='u'de'*, *trace=False*)

phonet was developed by Jörg Michael and documented in c't magazine vol. 25/1999, p. 252. It is a phonetic algorithm designed primarily for German. Cf. http://www.heise.de/ct/ftp/99/25/252/

This is a port of Jesper Zedlitz's code, which is licensed LGPL: https://code.google.com/p/phonet4java/source/browse/trunk/src/main/java/com/googlecode/phonet4java/Phonet.java

That is, in turn, based on Michael's C code, which is also licensed LGPL: ftp://ftp.heise.de/pub/ct/listings/phonet.zip

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
- **trace** (*bool*) – prints debugging info if True

Returns the phonet value

Return type str

abydos.phonetic.**phonex**(*word*, *maxlength=4*, *zero_pad=True*)
Phonex

Phonex is an algorithm derived from Soundex, defined in: Lait, A. J. and B. Randell. "An Assessment of Name Matching Algorithms". http://homepages.cs.ncl.ac.uk/brian.randell/Genealogy/NameMatching.pdf

Parameters

- **word** (*str*) – the word to transform
- **maxlength** (*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 maxlength string

Returns the Phonex value

Return type str

abydos.phonetic.**phonix**(*word*, *maxlength=4*, *zero_pad=True*)
Phonix

Phonix is a Soundex-like algorithm defined in: T.N. Gadd: PHONIX — The Algorithm, Program 24/4, 1990, p.363-366.

This implementation is based on http://cpansearch.perl.org/src/ULPFR/WAIT-1.800/soundex.c http://cs.anu.edu.au/people/Peter.Christen/Febrl/febrl-0.4.01/encode.py and https://metacpan.org/pod/Text::Phonetic::Phonix

Parameters

- **word** (*str*) – the word to transform
- **maxlength** (*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 maxlength string

Returns the Phonix value

Return type str

abydos.phonetic.**russell_index**(*word*)
Russell Index (integer output)

This follows Robert C. Russell's Index algorithm, as described in US Patent 1,261,167 (1917)

Parameters **word** (*str*) – the word to transform

Returns the Russell Index value

Return type int

abydos.phonetic.**russell_index_alpha**(*word*)
Russell Index (alphabetic output)

This follows Robert C. Russell's Index algorithm, as described in US Patent 1,261,167 (1917)

Parameters **word** (*str*) – the word to transform

Returns the Russell Index value as an alphabetic string

Return type str

abydos.phonetic.**russell_index_num_to_alpha**(*num*)
Russell Index integer to alphabetic string

This follows Robert C. Russell's Index algorithm, as described in US Patent 1,261,167 (1917)

Parameters **num** (*int*) – a Russell Index integer value

Returns the Russell Index as an alphabetic string

Return type str

abydos.phonetic.**sfinxbis**(*word*, *maxlength=None*)
SfinxBis

SfinxBis is a Soundex-like algorithm defined in: http://www.swami.se/download/18.248ad5af12aa8136533800091/SfinxBis.pdf

This implementation follows the reference implementation: http://www.swami.se/download/18.248ad5af12aa8136533800093/swa

SfinxBis is intended chiefly for Swedish names.

Parameters

- **word** (*str*) – the word to transform
- **maxlength** (*int*) – the length of the code returned (defaults to unlimited)

Returns the SfinxBis value

Return type *tuple*

abydos.phonetic.**soundex**(*word*, *maxlength=4*, *var=u'American'*, *reverse=False*, *zero_pad=True*)
Soundex

Parameters

- **word** (*str*) – the word to transform
- **maxlength** (*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 http://www.archives.gov/publications/general-info-leaflets/55-census.html and in Knuth(1998:394); this is also called Miracode
  - 'special' follows the rules from the 1880-1910 US Census, in which h & w are not treated as blocking consonants but as vowels
  - 'dm' computes the Daitch-Mokotoff Soundex
- **reverse** (*bool*) – reverse the word before computing the selected Soundex (defaults to False); This results in "Reverse Soundex"
- **zero_pad** (*bool*) – pad the end of the return value with 0s to achieve a maxlength string

Returns the Soundex value

Return type str

abydos.phonetic.**spfc**(*word*)
Standardized Phonetic Frequency Code (SPFC)

Standardized Phonetic Frequency Code is roughly Soundex-like. This implementation is based on page 19-21 of https://archive.org/stream/accessingindivid00moor#page/19/mode/1up

Parameters **word** (*str*) – the word to transform

Returns the SPFC value

Return type str

1.9 abydos.qgram module

abydos.qgram

The qgram module defines the QGrams multi-set class

class abydos.qgram.**QGrams**(*term*, *qval=2*, *start_stop=u'$#'*)
Bases: collections.Counter

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.

count()
q-grams count

Returns the total count of q-grams in a QGrams object

Return type int

ordered_list = []

term = u''

term_ss = u''

1.10 abydos.stats module

abydos.stats

The stats module defines functions for calculating various statistical data about linguistic objects.

This includes the ConfusionTable object, which includes members cable of calculating the following data based on a confusion table:

- population counts
- precision, recall, specificity, negative predictive value, fall-out, false discovery rate, accuracy, balanced accuracy, informedness, and markedness
- various means of the precision & recall, including: arithmetic, geometric, harmonic, quadratic, logarithmic, contraharmonic, identric (exponential), & Hölder (power/generalized) means
- F_β-scores, E-scores, G-measures, along with special functions for F_1, $F_{0.5}$, & F_2 scores
- significance & Matthews correlation coefficient calculation

Functions are provided for calculating the following means:

- arithmetic
- geometric
- harmonic
- quadratic
- contraharmonic
- logarithmic,
- identric (exponential)

- Seiffert's
- Lehmer
- Heronian
- Hölder (power/generalized)
- arithmetic-geometric
- geometric-harmonic
- arithmetic-geometric-harmonic

And for calculating:

- midrange
- median
- mode

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.

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

accuracy_gain()
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

balanced_accuracy()
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

cond_neg_pop()
condition negative population

Returns The condition negative population of the confusion table

Return type int

cond_pos_pop()
condition positive population

Returns The condition positive population of the confusion table

Return type int

correct_pop()
correct population

Returns the correct population of the confusion table

Return type int

dict()
cast to dict

Returns the confusion table as a dict

Return type *dict*

e_score(*beta=1*)
E-score

This is Van Rijsbergen's effectiveness measure

Cf. https://en.wikipedia.org/wiki/Information_retrieval#F-measure

Returns The E-score of the confusion table

Return type float

error_pop()
error population

Returns The error population of the confusion table

Return type int

f1_score()
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

f2_score()
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

f_measure()
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

fallout()
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

false_neg()
false negatives

Returns the false negatives of the confusion table

Return type int

false_pos()
false positives

Returns the false positives of the confusion table

Return type int

fbeta_score(*beta=1*)
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

Params numeric beta The β parameter in the above formula

Returns The F_β of the confusion table

Return type float

fdr()
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

fhalf_score()
$F_{0.5}$ 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

g_measure()
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

informedness()
Informedness is defined as $sensitivity + specificity - 1$.

AKA Youden's J statistic

AKA DeltaP'

Cf. https://en.wikipedia.org/wiki/Youden%27s_J_statistic

Cf. http://dspace.flinders.edu.au/xmlui/bitstream/handle/2328/27165/Powers%20Evaluation.pdf

Returns The informedness of the confusion table

Return type float

kappa_statistic()
κ statistic

The κ statistic is defined as: $\kappa = \frac{accuracy - random\ accuracy}{1 - random\ accuracy}$

The κ 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 κ statistic of the confusion table

Return type float

markedness()
Markedness is defined as $precision + npv - 1$

AKA DeltaP

Cf. https://en.wikipedia.org/wiki/Youden%27s_J_statistic

Cf. http://dspace.flinders.edu.au/xmlui/bitstream/handle/2328/27165/Powers%20Evaluation.pdf

Returns The markedness of the confusion table

Return type float

mcc()
Matthews correlation coefficient (MCC)

The Matthews correlation coefficient is defined 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

npv()
negative predictive value (NPV)

NPV is defined as $\frac{tn}{tn+fn}$

Cf. https://en.wikipedia.org/wiki/Negative_predictive_value

Returns The negative predictive value of the confusion table

Return type float

population(*N*)

Returns The population (N) of the confusion table

Return type int

pr_aghmean()
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 by Raïssouli, Leazizi, & Chergui: http://www.emis.de/journals/JIPAM/images/014_08_JIPAM/014_08.pdf

Returns The arithmetic-geometric-harmonic mean of the confusion table's precision & recall

Return type float

pr_agmean()
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

pr_amean()
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

pr_cmean()
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

pr_ghmean()
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

pr_gmean()
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

pr_heronian_mean()
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

pr_hmean()
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

pr_hoelder_mean(*exp=2*)
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** (*numeric*) – The exponent of the Hölder mean

Returns The Hölder mean for the given exponent of the confusion table's precision & recall

Return type float

pr_imean()
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

pr_lehmer_mean(*exp=2*)
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** (*numeric*) – The exponent of the Lehmer mean

Returns The Lehmer mean for the given exponent of the confusion table's precision & recall

Return type float

pr_lmean()
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

pr_qmean()
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

pr_seiffert_mean()
Seiffert's mean of precision & recall

Seiffert's mean of precision and recall is: $\frac{precision-recall}{4 \cdot arctan\sqrt{\frac{precision}{recall}}-\pi}$

Cf. http://www.helsinki.fi/~hasto/pp/miaPreprint.pdf

Returns Seiffer's mean of the confusion table's precision & recall

Return type float

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

precision_gain()
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

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

significance()
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

specificity()
Specificity is defined as $\frac{tn}{tn+fp}$

AKA true negative rate (TNR)

Cf. https://en.wikipedia.org/wiki/Specificity_(tests)

Returns The specificity of the confusion table

Return type float

test_neg_pop()
test negative population

Returns The test negative population of the confusion table

Return type int

test_pos_pop()
test positive population

Returns The test positive population of the confusion table

Return type int

true_neg()
true negatives

Returns the true negatives of the confusion table

Return type int

true_pos()
true positives

Returns the true positives of the confusion table

Return type int

tuple()
cast to tuple

Returns the confusion table as a 4-tuple (tp, tn, fp, fn)

Return type *tuple*

abydos.stats.**aghmean**(*nums*)
arithmetic-geometric-harmonic mean

Iterates over arithmetic, geometric, & harmonic means until they converge to a single value (rounded to 12 digits), following the method described by Raïssouli, Leazizi, & Chergui: http://www.emis.de/journals/JIPAM/images/014_08_JIPAM/014_08.pdf

Parameters **nums** (*list*) – A series of numbers

Returns The arithmetic-geometric-harmonic mean of nums

Return type float

abydos.stats.**agmean**(*nums*)
arithmetic-geometric mean

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

Parameters **nums** (*list*) – A series of numbers

Returns The arithmetic-geometric mean of nums

Return type float

abydos.stats.**amean**(*nums*)
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

abydos.stats.**cmean**(*nums*)
contraharmonic mean

The contraharmonic mean is: $\frac{\sum\limits_i x_i^2}{\sum\limits_i x_i}$

Cf. https://en.wikipedia.org/wiki/Contraharmonic_mean

Parameters **nums** (*list*) – A series of numbers

Returns The contraharmonic mean of nums

Return type float

abydos.stats.**ghmean**(*nums*)
geometric-harmonic mean

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

Parameters **nums** (*list*) – A series of numbers

Returns The geometric-harmonic mean of nums

Return type float

abydos.stats.**gmean**(*nums*)
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

abydos.stats.**heronian_mean**(*nums*)

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

abydos.stats.**hmean**(*nums*)

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

abydos.stats.**hoelder_mean**(*nums*, *exp=2*)

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

abydos.stats.**imean**(*nums*)

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

abydos.stats.**lehmer_mean**(*nums*, *exp=2*)

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

abydos.stats.**lmean**(*nums*)

logarithmic mean

The logarithmic mean of an arbitrarily long series is defined by http://www.survo.fi/papers/logmean.pdf as: $L(x_1, x_2, ..., x_n) = (n-1)! \sum_{i=1}^{n} \frac{x_i}{\prod_{\substack{j=1 \\ j \neq i}}^{n} ln \frac{x_i}{x_j}}$

Cf. https://en.wikipedia.org/wiki/Logarithmic_mean

Parameters **nums** (*list*) – A series of numbers

Returns The logarithmic mean of nums

Return type float

abydos.stats.**median**(*nums*)

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 float

abydos.stats.**midrange**(*nums*)

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

abydos.stats.**mode**(*nums*)

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 float

abydos.stats.**qmean**(*nums*)
quadratic mean

The quadratic mean of precision and recall 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

abydos.stats.**seiffert_mean**(*nums*)
Seiffert's mean

Seiffert's mean of two numbers x and y is: $\frac{x-y}{4 \cdot arctan\sqrt{\frac{x}{y}}-\pi}$

Cf. http://www.helsinki.fi/~hasto/pp/miaPreprint.pdf

Parameters **nums** (*list*) – A series of numbers

Returns Sieffert's mean of nums

Return type float

1.11 abydos.stemmer module

abydos.stemmer

The stemmer module defines word stemmers including:

- the Lovins stemmer
- the Porter and Porter2 (Snowball English) stemmers
- Snowball stemmers for German, Dutch, Norwegian, Swedish, and Danish
- CLEF German, German plus, and Swedish stemmers
- Caumann's German stemmer

abydos.stemmer.**caumanns**(*word*)
Caumanns German stemmer

Jörg Caumanns' stemmer is described in his article at: http://edocs.fu-berlin.de/docs/servlets/MCRFileNodeServlet/FUDOCS_derivate_000000000350/tr-b-99-16.pdf

This implementation is based on the GermanStemFilter described at: http://www.evelix.ch/unternehmen/Blog/evelix/2013/11/11/inner-workings-of-the-german-analyzer-in-lucene

Parameters **word** – the word to calculate the stem of

Returns word stem

Return type str

abydos.stemmer.**clef_german**(*word*)
CLEF German stemmer

The CLEF German stemmer is defined at: http://members.unine.ch/jacques.savoy/clef/germanStemmer.txt

Parameters **word** – the word to calculate the stem of

Returns word stem

Return type str

abydos.stemmer.**clef_german_plus**(*word*)
CLEF German stemmer plus

The CLEF German stemmer plus is defined at: http://members.unine.ch/jacques.savoy/clef/germanStemmerPlus.txt

Parameters **word** – the word to calculate the stem of

Returns word stem

Return type str

abydos.stemmer.**clef_swedish**(*word*)
CLEF Swedish stemmer

The CLEF Swedish stemmer is defined at: http://members.unine.ch/jacques.savoy/clef/swedishStemmer.txt

Parameters **word** – the word to calculate the stem of

Returns word stem

Return type str

abydos.stemmer.**lovins**(*word*)
Lovins stemmer

The Lovins stemmer is described in Julie Beth Lovins's article at: http://www.mt-archive.info/MT-1968-Lovins.pdf

Parameters **word** – the word to stem

Returns word stem

Return type string

abydos.stemmer.**porter**(*word*, *early_english=False*)
Porter stemmer

The Porter stemmer is defined at: http://snowball.tartarus.org/algorithms/porter/stemmer.html

Parameters

- **word** – the word to calculate the stem of
- **early_english** – set to True in order to remove -eth & -est (2nd & 3rd person singular verbal agreement suffixes)

Returns word stem

Return type str

abydos.stemmer.**porter2**(*word*, *early_english=False*)
Porter2 (Snowball English) stemmer:

The Porter2 (Snowball English) stemmer is defined at: http://snowball.tartarus.org/algorithms/english/stemmer.html

Parameters

- **word** – the word to calculate the stem of

- **early_english** – set to True in order to remove -eth & -est (2nd & 3rd person singular verbal agreement suffixes)

Returns word stem

Return type str

abydos.stemmer.**sb_danish**(*word*)
Snowball Danish stemmer

The Snowball Danish stemmer is defined at: http://snowball.tartarus.org/algorithms/danish/stemmer.html

Parameters **word** – the word to calculate the stem of

Returns word stem

Return type str

abydos.stemmer.**sb_dutch**(*word*)
Snowball Dutch stemmer

The Snowball Dutch stemmer is defined at: http://snowball.tartarus.org/algorithms/dutch/stemmer.html

Parameters **word** – the word to calculate the stem of

Returns word stem

Return type str

abydos.stemmer.**sb_german**(*word*, *alternate_vowels=False*)
Snowball German stemmer

The Snowball German stemmer is defined at: http://snowball.tartarus.org/algorithms/german/stemmer.html

Parameters

- **word** – the word to calculate the stem of
- **alternate_vowels** – composes ae as ä, oe as ö, and ue as ü before running the algorithm

Returns word stem

Return type str

abydos.stemmer.**sb_norwegian**(*word*)
Snowball Norwegian stemmer

The Snowball Norwegian stemmer is defined at: http://snowball.tartarus.org/algorithms/norwegian/stemmer.html

Parameters **word** – the word to calculate the stem of

Returns word stem

Return type str

abydos.stemmer.**sb_swedish**(*word*)
Snowball Swedish stemmer

The Snowball Swedish stemmer is defined at: http://snowball.tartarus.org/algorithms/swedish/stemmer.html

Parameters **word** – the word to calculate the stem of

Returns word stem

Return type str

1.12 abydos.util module

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)
- **jitter – adds small random noise to each member of a list of numbers** (ported from R's jitter function)
- Rational – a rational number class

class abydos.util.**Rational**(*p=0*, *q=1*)
Bases: object

Rational number object supporting arithmetic and comparison operations

denominator()
Get the denominator (q)

Returns the denominator q

Return type int

numerator()
Get the numerator (p)

Returns the numerator p

Return type int

p = 0L

q = 1L

abydos.util.**jitter**(*nums*, *factor=1*, *amount=None*, *min_val=None*, *max_val=None*, *rfunc=u'normal'*)
Jitter

Adapted from R documentation as this is ported directly from the R code:

The result, say r, is r = x + numpy.random.uniform(-a, a) where n = len(x) and a is the amount argument (if specified).

Let z = max(x) - min(x) (assuming the usual case). The amount a to be added is either provided as positive argument amount or otherwise computed from z, as follows:

If amount == 0, we set a = factor * z/50 (same as S).

If amount is None (default), we set a = factor * d/5 where d is the smallest difference between adjacent unique x values.

Based on: http://svn.r-project.org/R/trunk/src/library/base/R/jitter.R

Parameters

- **x** – numeric collection to which jitter should be added
- **factor** – numeric
- **amount** – numeric; if positive, used as amount (see below), otherwise, if = 0 the default is factor * z/50. Default (NULL): factor * d/5 where d is about the smallest difference between x values.
- **min_val** – the minimum permitted value in the returned list
- **max_val** – the maximum permitted value in the returned list

- **rand** – a string or function to indicate the random distribution used: 'normal' (default), 'uniform' (standard in the R version), or 'laplace' (requires Numpy) If a function is supplied, it should take one argument (the value passed as amount).

Returns a list of numbers with random noise added, according to the R jitter function

abydos.util.**prod**(*nums*)

Product

The product is Π(nums).

Cf. https://en.wikipedia.org/wiki/Product_(mathematics)

Parameters **nums** – a collection (list, tuple, set, etc.) of numbers

Returns the product of a nums

1.13 Module contents

abydos

Abydos NLP/IR library by Christopher C. Little

This library contains code I'm using for research, in particular dissertation research & experimentation.

Further documentation to come...

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

a

A

B

C

D

Q

R

S

T

W