nltk.stem package

Submodules

nltk.stem.api module

class nltk.stem.api.**StemmerI** [source]

Bases: object

A processing interface for removing morphological affixes from words. This process is known as stemming.

abstract **stem**(*token*) [source]

Strip affixes from the token and return the stem.

Parameters

token (*str*) – The token that should be stemmed.

nltk.stem.arlstem module

ARLSTem Arabic Stemmer The details about the implementation of this algorithm are described in: K. Abainia, S. Ouamour and H. Sayoud, A Novel Robust Arabic Light Stemmer , Journal of Experimental & Theoretical Artificial Intelligence (JETAI'17), Vol. 29, No. 3, 2017, pp. 557-573. The ARLSTem is a light Arabic stemmer that is based on removing the affixes from the word (i.e. prefixes, suffixes and infixes). It was evaluated and compared to several other stemmers using Paice's parameters (under-stemming index, over-stemming index and stemming weight), and the results showed that ARLSTem is promising and producing high performances. This stemmer is not based on any dictionary and can be used on-line effectively.

class nltk.stem.arlstem.**ARLSTem** [source]

Bases: nltk.stem.api.StemmerI

ARLSTem stemmer : a light Arabic Stemming algorithm without any dictionary. Department of Telecommunication & Information Processing. USTHB University, Algiers, Algeria. ARLSTem.stem(token) returns the Arabic stem for the input token. The ARLSTem Stemmer requires that all tokens are encoded using Unicode encoding.

fem2masc(*token*) [source]

transform the word from the feminine form to the masculine form.

norm(*token*) [source]

normalize the word by removing diacritics, replacing hamzated Alif with Alif replacing AlifMaqsura with Yaa and removing Waaw at the beginning.

plur2sing(*token*) [source]

transform the word from the plural form to the singular form.

pref(*token*) [source]

remove prefixes from the words' beginning.

stem(*token*) [source]

call this function to get the word's stem based on ARLSTem .

suff(*token*) [source]

remove suffixes from the word's end.

verb(*token*) [source]

stem the verb prefixes and suffixes or both

verb_t1(*token*) [source]

stem the present prefixes and suffixes

verb_t2(*token*) [source]

stem the future prefixes and suffixes

verb_t3(*token*) [source]

stem the present suffixes

verb_t4(*token*) [source]

stem the present prefixes

verb_t5(*token*) [source]

stem the future prefixes

verb_t6(*token*) [source]

stem the order prefixes

nltk.stem.cistem module

class nltk.stem.cistem.**Cistem**(*case_insensitive=False*) [source]

Bases: nltk.stem.api.StemmerI

CISTEM Stemmer for German

This is the official Python implementation of the CISTEM stemmer. It is based on the paper Leonie Weissweiler, Alexander Fraser (2017). Developing a Stemmer for German Based on a Comparative Analysis of Publicly Available Stemmers. In Proceedings of the German Society for Computational Linguistics and Language Technology (GSCL) which can be read here: http://www.cis.lmu.de/~weissweiler/cistem/

In the paper, we conducted an analysis of publicly available stemmers, developed two gold standards for German stemming and evaluated the stemmers based on the two gold standards. We then proposed the stemmer implemented here and show that it achieves slightly better f-measure than the other stemmers and is thrice as fast as the Snowball stemmer for German while being about as fast as most other stemmers.

case_insensitive is a a boolean specifying if case-insensitive stemming should be used. Case insensitivity improves performance only if words in the text may be incorrectly upper case. For all-lowercase and correctly cased text, best performance is achieved by setting case_insensitive for false.

Parameters

case_insensitive (*bool*) – if True, the stemming is case insensitive. False by default.

repl_xx = *re.compile('(.)\\1')*

repl_xx_back = *re.compile('(.)*')*

static **replace_back**(*word*) [source]

static **replace_to**(*word*) [source]

segment(*word*) [source]

This method works very similarly to stem (:func:'cistem.stem'). The difference is that in addition to returning the stem, it also returns the rest that was removed at the end. To be able to return the stem unchanged so the stem and the rest can be concatenated to form the original word, all subsitutions that altered the stem in any other way than by removing letters at the end were left out.

Parameters

word (*unicode*) – the word that is to be stemmed

Return word

the stemmed word

Return type

unicode

Return word

the removed suffix

Return type

unicode

```
>>> from nltk.stem.cistem import Cistem
>>> stemmer = Cistem()
>>> s1 = "Speicherbehältern"
>>> print("('" + stemmer.segment(s1)[0] + "', '" + stemmer.segment(s1)[1] + "')")
('speicherbehält', 'ern')
>>> s2 = "Grenzpostens"
>>> stemmer.segment(s2)
('grenzpost', 'ens')
>>> s3 = "Ausgefeiltere"
>>> stemmer.segment(s3)
('ausgefeilt', 'ere')
>>> stemmer = Cistem(True)
>>> print("('" + stemmer.segment(s1)[0] + "', '" + stemmer.segment(s1)[1] + "')")
('speicherbehäl', 'tern')
>>> stemmer.segment(s2)
('grenzpo', 'stens')
>>> stemmer.segment(s3)
('ausgefeil', 'tere')
```

stem(*word*) [source]

This method takes the word to be stemmed and returns the stemmed word.

Parameters

word (*unicode*) – the word that is to be stemmed

Return word

the stemmed word

Return type

unicode

```
>>> from nltk.stem.cistem import Cistem
>>> stemmer = Cistem()
>>> s1 = "Speicherbehältern"
>>> stemmer.stem(s1)
'speicherbehalt'
>>> s2 = "Grenzpostens"
>>> stemmer.stem(s2)
'grenzpost'
>>> s3 = "Ausgefeiltere"
>>> stemmer.stem(s3)
'ausgefeilt'
>>> stemmer = Cistem(True)
>>> stemmer.stem(s1)
'speicherbehal'
>>> stemmer.stem(s2)
'grenzpo'
>>> stemmer.stem(s3)
'ausgefeil'
```

strip_emr = *re.compile('e[mr]$')*

strip_esn = *re.compile('[esn]$')*

strip_ge = *re.compile('^ge(.{4,})')*

strip_nd = *re.compile('nd$')*

strip_t = *re.compile('t$')*

nltk.stem.isri module

ISRI Arabic Stemmer

The algorithm for this stemmer is described in:

Taghva, K., Elkoury, R., and Coombs, J. 2005. Arabic Stemming without a root dictionary. Information Science Research Institute. University of Nevada, Las Vegas, USA.

The Information Science Research Institute's (ISRI) Arabic stemmer shares many features with the Khoja stemmer. However, the main difference is that ISRI stemmer does not use root dictionary. Also, if a root is not found, ISRI stemmer returned normalized form, rather than returning the original unmodified word.

Additional adjustments were made to improve the algorithm:

1- Adding 60 stop words. 2- Adding the pattern (تفاعيل) to ISRI pattern set. 3- The step 2 in the original algorithm was normalizing all hamza. This step is discarded because it increases the word ambiguities and changes the original root.

class `nltk.stem.isri.`**`ISRIStemmer`** [source]

Bases: `nltk.stem.api.StemmerI`

ISRI Arabic stemmer based on algorithm: Arabic Stemming without a root dictionary. Information Science Research Institute. University of Nevada, Las Vegas, USA.

A few minor modifications have been made to ISRI basic algorithm. See the source code of this module for more information.

isri.stem(token) returns Arabic root for the given token.

The ISRI Stemmer requires that all tokens have Unicode string types. If you use Python IDLE on Arabic Windows you have to decode text first using Arabic '1256' coding.

`end_w5`(*word*) [source]

ending step (word of length five)

`end_w6`(*word*) [source]

ending step (word of length six)

`norm`(*word*, *num=3*) [source]

normalization: num=1 normalize diacritics num=2 normalize initial hamza num=3 both 1&2

`pre1`(*word*) [source]

normalize short prefix

`pre32`(*word*) [source]

remove length three and length two prefixes in this order

`pro_w4`(*word*) [source]

process length four patterns and extract length three roots

`pro_w53`(*word*) [source]

process length five patterns and extract length three roots

`pro_w54`(*word*) [source]

process length five patterns and extract length four roots

`pro_w6`(*word*) [source]

process length six patterns and extract length three roots

`pro_w64`(*word*) [source]

process length six patterns and extract length four roots

`stem`(*token*) [source]

Stemming a word token using the ISRI stemmer.

`suf1`(*word*) [source]

normalize short sufix

suf32(*word*) [source]

remove length three and length two suffixes in this order

waw(*word*) [source]

remove connective ‘و’ if it precedes a word beginning with ‘و’

nltk.stem.lancaster module

A word stemmer based on the Lancaster (Paice/Husk) stemming algorithm. Paice, Chris D. “Another Stemmer.” ACM SIGIR Forum 24.3 (1990): 56-61.

class nltk.stem.lancaster.**LancasterStemmer**(*rule_tuple=None*, *strip_prefix_flag=False*) [source]

Bases: nltk.stem.api.StemmerI

Lancaster Stemmer

```
>>> from nltk.stem.lancaster import LancasterStemmer
>>> st = LancasterStemmer()
>>> st.stem('maximum')     # Remove "-um" when word is intact
'maxim'
>>> st.stem('presumably')  # Don't remove "-um" when word is not intact
'presum'
>>> st.stem('multiply')    # No action taken if word ends with "-ply"
'multiply'
>>> st.stem('provision')   # Replace "-sion" with "-j" to trigger "j" set of rules
'provid'
>>> st.stem('owed')        # Word starting with vowel must contain at least 2 letters
'ow'
>>> st.stem('ear')         # ditto
'ear'
>>> st.stem('saying')      # Words starting with consonant must contain at least 3
'say'
>>> st.stem('crying')      #     letters and one of those letters must be a vowel
'cry'
>>> st.stem('string')      # ditto
'string'
>>> st.stem('meant')       # ditto
'meant'
>>> st.stem('cement')      # ditto
'cem'
>>> st_pre = LancasterStemmer(strip_prefix_flag=True)
>>> st_pre.stem('kilometer') # Test Prefix
'met'
>>> st_custom = LancasterStemmer(rule_tuple=("ssen4>", "s1t."))
>>> st_custom.stem("ness") # Change s to t
'nest'
```

default_rule_tuple = *('ai*2.', 'a*1.', 'bb1.', 'city3s.', 'ci2>', 'cn1t>', 'dd1.', 'dei3y>', 'deec2ss.', 'dee1.', 'de2>', 'dooh4>', 'e1>', 'feil1v.', 'fi2>', 'gni3>', 'gai3y.', 'ga2>', 'gg1.', 'ht*2.', 'hsiug5ct.', 'hsi3>', 'i*1.', 'i1y>', 'ji1d.', 'juf1s.', 'ju1d.', 'jo1d.', 'jeh1r.', 'jrev1t.', 'jsim2t.', 'jn1d.', 'j1s.', 'lbaifi6.', 'lbai4y.', 'lba3>', 'lbi3.', 'lib2l>', 'lc1.', 'lufi4y.', 'luf3>', 'lu2.', 'lai3>', 'lau3>', 'la2>', 'll1.', 'mui3.', 'mu*2.', 'msi3>', 'mm1.', 'nois4j>', 'noix4ct.', 'noi3>', 'nai3>', 'na2>', 'nee0.', 'ne2>', 'nn1.', 'pihs4>', 'pp1.', 're2>', 'rae0.', 'ra2.', 'ro2>', 'ru2>', 'rr1.', 'rt1>', 'rei3y>', 'sei3y>', 'sis2.', 'si2>', 'ssen4>', 'ss0.', 'suo3>', 'su*2.', 's*1>', 's0.', 'tacilp4y.', 'ta2>', 'tnem4>', 'tne3>', 'tna3>', 'tpir2b.', 'tpro2b.', 'tcud1.', 'tpmus2.', 'tpec2iv.', 'tulo2v.', 'tsis0.', 'tsi3>', 'tt1.', 'uqi3.', 'ugo1.', 'vis3j>', 'vie0.', 'vi2>', 'ylb1>', 'yli3y>', 'ylp0.', 'yl2>', 'ygo1.', 'yhp1.', 'ymo1.', 'ypo1.', 'yti3>', 'yte3>', 'ytl2.', 'yrtsi5.', 'yra3>', 'yro3>', 'yfi3.', 'ycn2t>', 'yca3>', 'zi2>', 'zy1s.')*

parseRules(*rule_tuple=None*) [source]

Validate the set of rules used in this stemmer.

If this function is called as an individual method, without using stem method, rule_tuple argument will be compiled into self.rule_dictionary. If this function is called within stem, self._rule_tuple will be used.

stem(*word*) [source]

Stem a word using the Lancaster stemmer.

nltk.stem.porter module

Porter Stemmer

This is the Porter stemming algorithm. It follows the algorithm presented in

Porter, M. "An algorithm for suffix stripping." Program 14.3 (1980): 130-137.

with some optional deviations that can be turned on or off with the *mode* argument to the constructor.

Martin Porter, the algorithm's inventor, maintains a web page about the algorithm at

http://www.tartarus.org/~martin/PorterStemmer/

which includes another Python implementation and other implementations in many languages.

class `nltk.stem.porter.`**`PorterStemmer`**(*mode='NLTK_EXTENSIONS'*) [source]

Bases: `nltk.stem.api.StemmerI`

A word stemmer based on the Porter stemming algorithm.

Porter, M. "An algorithm for suffix stripping." Program 14.3 (1980): 130-137.

See http://www.tartarus.org/~martin/PorterStemmer/ for the homepage of the algorithm.

Martin Porter has endorsed several modifications to the Porter algorithm since writing his original paper, and those extensions are included in the implementations on his website. Additionally, others have proposed further improvements to the algorithm, including NLTK contributors. There are thus three modes that can be selected by passing the appropriate constant to the class constructor's *mode* attribute:

PorterStemmer.ORIGINAL_ALGORITHM - Implementation that is faithful to the original paper.

Note that Martin Porter has deprecated this version of the algorithm. Martin distributes implementations of the Porter Stemmer in many languages, hosted at:

http://www.tartarus.org/~martin/PorterStemmer/

and all of these implementations include his extensions. He strongly recommends against using the original, published version of the algorithm; only use this mode if you clearly understand why you are choosing to do so.

PorterStemmer.MARTIN_EXTENSIONS - Implementation that only uses the modifications to the

algorithm that are included in the implementations on Martin Porter's website. He has declared Porter frozen, so the behaviour of those implementations should never change.

PorterStemmer.NLTK_EXTENSIONS (default) - Implementation that includes further improvements devised by

NLTK contributors or taken from other modified implementations found on the web.

For the best stemming, you should use the default NLTK_EXTENSIONS version. However, if you need to get the same results as either the original algorithm or one of Martin Porter's hosted versions for compatibility with an existing implementation or dataset, you can use one of the other modes instead.

`MARTIN_EXTENSIONS` = *'MARTIN_EXTENSIONS'*

`NLTK_EXTENSIONS` = *'NLTK_EXTENSIONS'*

`ORIGINAL_ALGORITHM` = *'ORIGINAL_ALGORITHM'*

`stem`(*word*) [source]

Strip affixes from the token and return the stem.

Parameters

token (*str*) – The token that should be stemmed.

nltk.stem.porter.**demo**() [source]

A demonstration of the porter stemmer on a sample from the Penn Treebank corpus.

nltk.stem.regexp module

class nltk.stem.regexp.**RegexpStemmer**(*regexp*, *min=0*) [source]

Bases: nltk.stem.api.StemmerI

A stemmer that uses regular expressions to identify morphological affixes. Any substrings that match the regular expressions will be removed.

```
>>> from nltk.stem import RegexpStemmer
>>> st = RegexpStemmer('ing$|s$|e$|able$', min=4)
>>> st.stem('cars')
'car'
>>> st.stem('mass')
'mas'
>>> st.stem('was')
'was'
>>> st.stem('bee')
'bee'
>>> st.stem('compute')
'comput'
>>> st.stem('advisable')
'advis'
```

Parameters

- **regexp** (*str or regexp*) – The regular expression that should be used to identify morphological affixes.
- **min** (*int*) – The minimum length of string to stem

stem(*word*) [source]

Strip affixes from the token and return the stem.

Parameters

token (*str*) – The token that should be stemmed.

nltk.stem.rslp module

class nltk.stem.rslp.**RSLPStemmer** [source]

Bases: nltk.stem.api.StemmerI

A stemmer for Portuguese.

```
>>> from nltk.stem import RSLPStemmer
>>> st = RSLPStemmer()
>>> # opening lines of Erico Verissimo's "Música ao Longe"
>>> text = '''
... Clarissa risca com giz no quadro-negro a paisagem que os alunos
... devem copiar . Uma casinha de porta e janela , em cima duma
... coxilha .'''
>>> for token in text.split():
...     print(st.stem(token))
clariss risc com giz no quadro-negr a pais que os alun dev copi .
uma cas de port e janel , em cim dum coxilh .
```

apply_rule(*word*, *rule_index*) [source]

read_rule(*filename*) [source]

stem(*word*) [source]

Strip affixes from the token and return the stem.

Parameters

token (*str*) – The token that should be stemmed.

nltk.stem.snowball module

Snowball stemmers

This module provides a port of the Snowball stemmers developed by Martin Porter.

There is also a demo function: *snowball.demo()*.

class nltk.stem.snowball.**ArabicStemmer**(*ignore_stopwords=False*) [source]

Bases: nltk.stem.snowball._StandardStemmer

https://github.com/snowballstem/snowball/blob/master/algorithms/arabic/stem_Unicode.sbl
(Original Algorithm) The Snowball Arabic light Stemmer Algorithm : Assem Chelli

Abdelkrim Aries Lakhdar Benzahia

Nltk Version Author : Lakhdar Benzahia

is_defined = *False*

is_noun = *True*

is_verb = *True*

prefix_step2a_success = *False*

prefix_step3a_noun_success = *False*

prefix_step3b_noun_success = *False*

stem(*word*) [source]

Stem an Arabic word and return the stemmed form.

Parameters

word – string

Returns

string

suffix_noun_step1a_success = *False*

suffix_noun_step2a_success = *False*

suffix_noun_step2b_success = *False*

suffix_noun_step2c2_success = *False*

suffix_verb_step2a_success = *False*

suffix_verb_step2b_success = *False*

suffixe_noun_step1b_success = *False*

suffixes_verb_step1_success = *False*

class nltk.stem.snowball.**DanishStemmer**(*ignore_stopwords=False*) [source]

Bases: nltk.stem.snowball._ScandinavianStemmer

The Danish Snowball stemmer.

Variables

- **__vowels** – The Danish vowels.
- **__consonants** – The Danish consonants.
- **__double_consonants** – The Danish double consonants.
- **__s_ending** – Letters that may directly appear before a word final 's'.
- **__step1_suffixes** – Suffixes to be deleted in step 1 of the algorithm.
- **__step2_suffixes** – Suffixes to be deleted in step 2 of the algorithm.

- **__step3_suffixes** – Suffixes to be deleted in step 3 of the algorithm.

Note

A detailed description of the Danish stemming algorithm can be found under http://snowball.tartarus.org/algorithms/danish/stemmer.html

stem(*word*) [source]

Stem a Danish word and return the stemmed form.

Parameters

word (*str or unicode*) – The word that is stemmed.

Returns

The stemmed form.

Return type

unicode

class nltk.stem.snowball.**DutchStemmer**(*ignore_stopwords=False*) [source]

Bases: nltk.stem.snowball._StandardStemmer

The Dutch Snowball stemmer.

Variables

- **__vowels** – The Dutch vowels.
- **__step1_suffixes** – Suffixes to be deleted in step 1 of the algorithm.
- **__step3b_suffixes** – Suffixes to be deleted in step 3b of the algorithm.

Note

A detailed description of the Dutch stemming algorithm can be found under http://snowball.tartarus.org/algorithms/dutch/stemmer.html

stem(*word*) [source]

Stem a Dutch word and return the stemmed form.

Parameters

word (*str or unicode*) – The word that is stemmed.

Returns

The stemmed form.

Return type

unicode

class nltk.stem.snowball.**EnglishStemmer**(*ignore_stopwords=False*) [source]

Bases: nltk.stem.snowball._StandardStemmer

The English Snowball stemmer.

Variables

- **__vowels** – The English vowels.
- **__double_consonants** – The English double consonants.
- **__li_ending** – Letters that may directly appear before a word final 'li'.
- **__step0_suffixes** – Suffixes to be deleted in step 0 of the algorithm.
- **__step1a_suffixes** – Suffixes to be deleted in step 1a of the algorithm.
- **__step1b_suffixes** – Suffixes to be deleted in step 1b of the algorithm.
- **__step2_suffixes** – Suffixes to be deleted in step 2 of the algorithm.
- **__step3_suffixes** – Suffixes to be deleted in step 3 of the algorithm.
- **__step4_suffixes** – Suffixes to be deleted in step 4 of the algorithm.
- **__step5_suffixes** – Suffixes to be deleted in step 5 of the algorithm.

- **__special_words** – A dictionary containing words which have to be stemmed specially.

Note

A detailed description of the English stemming algorithm can be found under http://snowball.tartarus.org/algorithms/english/stemmer.html

`stem`(*word*) [source]

Stem an English word and return the stemmed form.

Parameters

word (*str or unicode*) – The word that is stemmed.

Returns

The stemmed form.

Return type

unicode

class `nltk.stem.snowball.FinnishStemmer`(*ignore_stopwords=False*) [source]

Bases: `nltk.stem.snowball._StandardStemmer`

The Finnish Snowball stemmer.

Variables

- **__vowels** – The Finnish vowels.
- **__restricted_vowels** – A subset of the Finnish vowels.
- **__long_vowels** – The Finnish vowels in their long forms.
- **__consonants** – The Finnish consonants.
- **__double_consonants** – The Finnish double consonants.
- **__step1_suffixes** – Suffixes to be deleted in step 1 of the algorithm.
- **__step2_suffixes** – Suffixes to be deleted in step 2 of the algorithm.
- **__step3_suffixes** – Suffixes to be deleted in step 3 of the algorithm.
- **__step4_suffixes** – Suffixes to be deleted in step 4 of the algorithm.

Note

A detailed description of the Finnish stemming algorithm can be found under http://snowball.tartarus.org/algorithms/finnish/stemmer.html

`stem`(*word*) [source]

Stem a Finnish word and return the stemmed form.

Parameters

word (*str or unicode*) – The word that is stemmed.

Returns

The stemmed form.

Return type

unicode

class `nltk.stem.snowball.FrenchStemmer`(*ignore_stopwords=False*) [source]

Bases: `nltk.stem.snowball._StandardStemmer`

The French Snowball stemmer.

Variables

- **__vowels** – The French vowels.
- **__step1_suffixes** – Suffixes to be deleted in step 1 of the algorithm.
- **__step2a_suffixes** – Suffixes to be deleted in step 2a of the algorithm.
- **__step2b_suffixes** – Suffixes to be deleted in step 2b of the algorithm.

- **__step4_suffixes** – Suffixes to be deleted in step 4 of the algorithm.

Note

A detailed description of the French stemming algorithm can be found under http://snowball.tartarus.org/algorithms/french/stemmer.html

`stem`(*word*) [source]

Stem a French word and return the stemmed form.

Parameters

word (*str or unicode*) – The word that is stemmed.

Returns

The stemmed form.

Return type

unicode

class `nltk.stem.snowball.GermanStemmer`(*ignore_stopwords=False*) [source]

Bases: `nltk.stem.snowball._StandardStemmer`

The German Snowball stemmer.

Variables

- **__vowels** – The German vowels.
- **__s_ending** – Letters that may directly appear before a word final 's'.
- **__st_ending** – Letter that may directly appear before a word final 'st'.
- **__step1_suffixes** – Suffixes to be deleted in step 1 of the algorithm.
- **__step2_suffixes** – Suffixes to be deleted in step 2 of the algorithm.
- **__step3_suffixes** – Suffixes to be deleted in step 3 of the algorithm.

Note

A detailed description of the German stemming algorithm can be found under http://snowball.tartarus.org/algorithms/german/stemmer.html

`stem`(*word*) [source]

Stem a German word and return the stemmed form.

Parameters

word (*str or unicode*) – The word that is stemmed.

Returns

The stemmed form.

Return type

unicode

class `nltk.stem.snowball.HungarianStemmer`(*ignore_stopwords=False*) [source]

Bases: `nltk.stem.snowball._LanguageSpecificStemmer`

The Hungarian Snowball stemmer.

Variables

- **__vowels** – The Hungarian vowels.
- **__digraphs** – The Hungarian digraphs.
- **__double_consonants** – The Hungarian double consonants.
- **__step1_suffixes** – Suffixes to be deleted in step 1 of the algorithm.
- **__step2_suffixes** – Suffixes to be deleted in step 2 of the algorithm.
- **__step3_suffixes** – Suffixes to be deleted in step 3 of the algorithm.
- **__step4_suffixes** – Suffixes to be deleted in step 4 of the algorithm.
- **__step5_suffixes** – Suffixes to be deleted in step 5 of the algorithm.

- **__step6_suffixes** – Suffixes to be deleted in step 6 of the algorithm.
- **__step7_suffixes** – Suffixes to be deleted in step 7 of the algorithm.
- **__step8_suffixes** – Suffixes to be deleted in step 8 of the algorithm.
- **__step9_suffixes** – Suffixes to be deleted in step 9 of the algorithm.

Note

A detailed description of the Hungarian stemming algorithm can be found under http://snowball.tartarus.org/algorithms/hungarian/stemmer.html

`stem`(*word*) [source]

Stem an Hungarian word and return the stemmed form.

Parameters

word (*str or unicode*) – The word that is stemmed.

Returns

The stemmed form.

Return type

unicode

class `nltk.stem.snowball.ItalianStemmer`(*ignore_stopwords=False*) [source]

Bases: `nltk.stem.snowball._StandardStemmer`

The Italian Snowball stemmer.

Variables

- **__vowels** – The Italian vowels.
- **__step0_suffixes** – Suffixes to be deleted in step 0 of the algorithm.
- **__step1_suffixes** – Suffixes to be deleted in step 1 of the algorithm.
- **__step2_suffixes** – Suffixes to be deleted in step 2 of the algorithm.

Note

A detailed description of the Italian stemming algorithm can be found under http://snowball.tartarus.org/algorithms/italian/stemmer.html

`stem`(*word*) [source]

Stem an Italian word and return the stemmed form.

Parameters

word (*str or unicode*) – The word that is stemmed.

Returns

The stemmed form.

Return type

unicode

class `nltk.stem.snowball.NorwegianStemmer`(*ignore_stopwords=False*) [source]

Bases: `nltk.stem.snowball._ScandinavianStemmer`

The Norwegian Snowball stemmer.

Variables

- **__vowels** – The Norwegian vowels.
- **__s_ending** – Letters that may directly appear before a word final 's'.
- **__step1_suffixes** – Suffixes to be deleted in step 1 of the algorithm.
- **__step2_suffixes** – Suffixes to be deleted in step 2 of the algorithm.
- **__step3_suffixes** – Suffixes to be deleted in step 3 of the algorithm.

Note

A detailed description of the Norwegian stemming algorithm can be found under http://snowball.tartarus.org/algorithms/norwegian/stemmer.html

stem(*word*) [source]

Stem a Norwegian word and return the stemmed form.

Parameters

word (*str or unicode*) – The word that is stemmed.

Returns

The stemmed form.

Return type

unicode

class nltk.stem.snowball.**PorterStemmer**(*ignore_stopwords=False*) [source]

Bases: nltk.stem.snowball._LanguageSpecificStemmer, nltk.stem.porter.PorterStemmer

A word stemmer based on the original Porter stemming algorithm.

Porter, M. "An algorithm for suffix stripping." Program 14.3 (1980): 130-137.

A few minor modifications have been made to Porter's basic algorithm. See the source code of the module nltk.stem.porter for more information.

class nltk.stem.snowball.**PortugueseStemmer**(*ignore_stopwords=False*) [source]

Bases: nltk.stem.snowball._StandardStemmer

The Portuguese Snowball stemmer.

Variables

- **__vowels** – The Portuguese vowels.
- **__step1_suffixes** – Suffixes to be deleted in step 1 of the algorithm.
- **__step2_suffixes** – Suffixes to be deleted in step 2 of the algorithm.
- **__step4_suffixes** – Suffixes to be deleted in step 4 of the algorithm.

Note

A detailed description of the Portuguese stemming algorithm can be found under http://snowball.tartarus.org/algorithms/portuguese/stemmer.html

stem(*word*) [source]

Stem a Portuguese word and return the stemmed form.

Parameters

word (*str or unicode*) – The word that is stemmed.

Returns

The stemmed form.

Return type

unicode

class nltk.stem.snowball.**RomanianStemmer**(*ignore_stopwords=False*) [source]

Bases: nltk.stem.snowball._StandardStemmer

The Romanian Snowball stemmer.

Variables

- **__vowels** – The Romanian vowels.
- **__step0_suffixes** – Suffixes to be deleted in step 0 of the algorithm.
- **__step1_suffixes** – Suffixes to be deleted in step 1 of the algorithm.
- **__step2_suffixes** – Suffixes to be deleted in step 2 of the algorithm.
- **__step3_suffixes** – Suffixes to be deleted in step 3 of the algorithm.

Note

A detailed description of the Romanian stemming algorithm can be found under http://snowball.tartarus.org/algorithms/romanian/stemmer.html

`stem`(*word*) [source]

Stem a Romanian word and return the stemmed form.

Parameters

word (*str or unicode*) – The word that is stemmed.

Returns

The stemmed form.

Return type

unicode

class `nltk.stem.snowball.RussianStemmer`(*ignore_stopwords=False*) [source]

Bases: `nltk.stem.snowball._LanguageSpecificStemmer`

The Russian Snowball stemmer.

Variables

- **__perfective_gerund_suffixes** – Suffixes to be deleted.
- **__adjectival_suffixes** – Suffixes to be deleted.
- **__reflexive_suffixes** – Suffixes to be deleted.
- **__verb_suffixes** – Suffixes to be deleted.
- **__noun_suffixes** – Suffixes to be deleted.
- **__superlative_suffixes** – Suffixes to be deleted.
- **__derivational_suffixes** – Suffixes to be deleted.

Note

A detailed description of the Russian stemming algorithm can be found under http://snowball.tartarus.org/algorithms/russian/stemmer.html

`stem`(*word*) [source]

Stem a Russian word and return the stemmed form.

Parameters

word (*str or unicode*) – The word that is stemmed.

Returns

The stemmed form.

Return type

unicode

class `nltk.stem.snowball.SnowballStemmer`(*language*, *ignore_stopwords=False*) [source]

Bases: `nltk.stem.api.StemmerI`

Snowball Stemmer

The following languages are supported: Arabic, Danish, Dutch, English, Finnish, French, German, Hungarian, Italian, Norwegian, Portuguese, Romanian, Russian, Spanish and Swedish.

The algorithm for English is documented here:

Porter, M. “An algorithm for suffix stripping.” Program 14.3 (1980): 130-137.

The algorithms have been developed by Martin Porter. These stemmers are called Snowball, because Porter created a programming language with this name for creating new stemming algorithms. There is more information available at http://snowball.tartarus.org/

The stemmer is invoked as shown below:

```
>>> from nltk.stem import SnowballStemmer
>>> print(" ".join(SnowballStemmer.languages)) # See which languages are supported
```

```
arabic danish dutch english finnish french german hungarian
italian norwegian porter portuguese romanian russian
spanish swedish
>>> stemmer = SnowballStemmer("german") # Choose a language
>>> stemmer.stem("Autobahnen") # Stem a word
'autobahn'
```

Invoking the stemmers that way is useful if you do not know the language to be stemmed at runtime. Alternatively, if you already know the language, then you can invoke the language specific stemmer directly:

```
>>> from nltk.stem.snowball import GermanStemmer
>>> stemmer = GermanStemmer()
>>> stemmer.stem("Autobahnen")
'autobahn'
```

Parameters

- **language** (*str or unicode*) – The language whose subclass is instantiated.
- **ignore_stopwords** (*bool*) – If set to True, stopwords are not stemmed and returned unchanged. Set to False by default.

Raises

ValueError – If there is no stemmer for the specified language, a ValueError is raised.

`languages` = *('arabic', 'danish', 'dutch', 'english', 'finnish', 'french', 'german', 'hungarian', 'italian', 'norwegian', 'porter', 'portuguese', 'romanian', 'russian', 'spanish', 'swedish')*

`stem`(*token*) [source]

Strip affixes from the token and return the stem.

Parameters

token (*str*) – The token that should be stemmed.

class `nltk.stem.snowball.SpanishStemmer`(*ignore_stopwords=False*) [source]

Bases: `nltk.stem.snowball._StandardStemmer`

The Spanish Snowball stemmer.

Variables

- **__vowels** – The Spanish vowels.
- **__step0_suffixes** – Suffixes to be deleted in step 0 of the algorithm.
- **__step1_suffixes** – Suffixes to be deleted in step 1 of the algorithm.
- **__step2a_suffixes** – Suffixes to be deleted in step 2a of the algorithm.
- **__step2b_suffixes** – Suffixes to be deleted in step 2b of the algorithm.
- **__step3_suffixes** – Suffixes to be deleted in step 3 of the algorithm.

Note

A detailed description of the Spanish stemming algorithm can be found under http://snowball.tartarus.org/algorithms/spanish/stemmer.html

`stem`(*word*) [source]

Stem a Spanish word and return the stemmed form.

Parameters

word (*str or unicode*) – The word that is stemmed.

Returns

The stemmed form.

Return type

unicode

class `nltk.stem.snowball.SwedishStemmer`(*ignore_stopwords=False*) [source]

Bases: `nltk.stem.snowball._ScandinavianStemmer`

The Swedish Snowball stemmer.

Variables

- **__vowels** – The Swedish vowels.
- **__s_ending** – Letters that may directly appear before a word final 's'.
- **__step1_suffixes** – Suffixes to be deleted in step 1 of the algorithm.
- **__step2_suffixes** – Suffixes to be deleted in step 2 of the algorithm.
- **__step3_suffixes** – Suffixes to be deleted in step 3 of the algorithm.

Note

A detailed description of the Swedish stemming algorithm can be found under http://snowball.tartarus.org/algorithms/swedish/stemmer.html

`stem`(*word*) [source]

Stem a Swedish word and return the stemmed form.

Parameters

word (*str or unicode*) – The word that is stemmed.

Returns

The stemmed form.

Return type

unicode

`nltk.stem.snowball.demo`() [source]

This function provides a demonstration of the Snowball stemmers.

After invoking this function and specifying a language, it stems an excerpt of the Universal Declaration of Human Rights (which is a part of the NLTK corpus collection) and then prints out the original and the stemmed text.

nltk.stem.util module

`nltk.stem.util.prefix_replace`(*original*, *old*, *new*) [source]

Replaces the old prefix of the original string by a new suffix

Parameters

- **original** – string
- **old** – string
- **new** – string

Returns

string

`nltk.stem.util.suffix_replace`(*original*, *old*, *new*) [source]

Replaces the old suffix of the original string by a new suffix

nltk.stem.wordnet module

class `nltk.stem.wordnet.WordNetLemmatizer` [source]

Bases: `object`

WordNet Lemmatizer

Lemmatize using WordNet's built-in morphy function. Returns the input word unchanged if it cannot be found in WordNet.

```
>>> from nltk.stem import WordNetLemmatizer
>>> wnl = WordNetLemmatizer()
```

```
>>> print(wnl.lemmatize('dogs'))
dog
>>> print(wnl.lemmatize('churches'))
church
>>> print(wnl.lemmatize('aardwolves'))
aardwolf
>>> print(wnl.lemmatize('abaci'))
abacus
>>> print(wnl.lemmatize('hardrock'))
hardrock
```

lemmatize(*word*, *pos='n'*) [source]

nltk.stem.wordnet.**teardown_module**(*module=None*) [source]

Module contents

NLTK Stemmers

Interfaces used to remove morphological affixes from words, leaving only the word stem. Stemming algorithms aim to remove those affixes required for eg. grammatical role, tense, derivational morphology leaving only the stem of the word. This is a difficult problem due to irregular words (eg. common verbs in English), complicated morphological rules, and part-of-speech and sense ambiguities (eg. `ceil-` is not the stem of `ceiling`).

StemmerI defines a standard interface for stemmers.