NLTK

The Natural Language Toolkit (NLTK) is a popular library in Python for working with human language data, widely used for natural language processing (NLP) tasks. It provides easy-to-use interfaces for numerous corpora and lexical resources like WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and more .

Sample Useage for Tokenize

Regression Test : NLTKWordTokenizer

In [1]:
```
from nltk.tokenize import word_tokenize
```

In [2]:
```
s1 = "Natural language processing allows computers to understand and generate hu
```

In [3]:
```
word_tokenize(s1)
```

Out[3]:
```
['Natural',
 'language',
 'processing',
 'allows',
 'computers',
 'to',
 'understand',
 'and',
 'generate',
 'human',
 'language',
 ',',
 'bridging',
 'the',
 'gap',
 'between',
 'technology',
 'and',
 'communication',
 '.']
```

In [4]:
```
s2 = "Life teaches us that every setback is an opportunity to learn, grow, and b
```

In [5]:
```
word_tokenize(s2)
```

Out[5]:
```
['Life',
 'teaches',
 'us',
 'that',
 'every',
 'setback',
 'is',
 'an',
 'opportunity',
 'to',
 'learn',
 ',',
 'grow',
 ',',
 'and',
 'become',
 'stronger',
 'than',
 'we',
 'were',
 'before',
 '.']
```

In [6]:
```
s3 = "Motivation is the spark that ignites action, but it’s discipline that fuel
```

In [7]:
```
word_tokenize(s3)
```

Out[7]:
```
['Motivation',
 'is',
 'the',
 'spark',
 'that',
 'ignites',
 'action',
 ',',
 'but',
 'it',
 '’',
 's',
 'discipline',
 'that',
 'fuels',
 'the',
 'journey',
 'toward',
 'achieving',
 'your',
 'dreams',
 '.']
```

In [8]:
```
s4 = "I cannot cannot work under these conditions!"
```

In [ ]:

In [ ]:

In [ ]:

In [9]:
```
s6 =  "The company spent $30,000,000 last year."
```

In [10]:
```
word_tokenize(s6)
```

Out[10]:
```
['The', 'company', 'spent', '$', '30,000,000', 'last', 'year', '.']
```

Gathering the spans of the tokenized strings.

In [11]:
```
from nltk.tokenize import NLTKWordTokenizer
```

In [12]:
```
s = '''Good muffins cost $3.88\nin New (York).  Please (buy) me\ntwo of them.\n(
```

In [13]:
```
expected = [(0, 4), (5, 12), (13, 17), (18, 19), (19, 23),
... (24, 26), (27, 30), (31, 32), (32, 36), (36, 37), (37, 38),
... (40, 46), (47, 48), (48, 51), (51, 52), (53, 55), (56, 59),
... (60, 62), (63, 68), (69, 70), (70, 76), (76, 77), (77, 78)]
```

In [14]:
```
list(NLTKWordTokenizer().span_tokenize(s)) == expected
```

Out[14]:
```
True
```

In [15]:
```
expected = ['Good', 'muffins', 'cost', '$', '3.88', 'in',
... 'New', '(', 'York', ')', '.', 'Please', '(', 'buy', ')',
... 'me', 'two', 'of', 'them.', '(', 'Thanks', ')', '.']
```

In [16]:
```
[s[start:end] for start, end in NLTKWordTokenizer().span_tokenize(s)] == expecte
```

Out[16]:
```
True
```

In [17]:
```
 s = '''I said, "I'd like to buy some ''good muffins" which cost $3.88\n each in
```

In [18]:
```
expected = [(0, 1), (2, 6), (6, 7), (8, 9), (9, 10), (10, 12),
... (13, 17), (18, 20), (21, 24), (25, 29), (30, 32), (32, 36),
... (37, 44), (44, 45), (46, 51), (52, 56), (57, 58), (58, 62),
... (64, 68), (69, 71), (72, 75), (76, 77), (77, 81), (81, 82),
... (82, 83), (83, 84)]
```

In [19]:
```
list(NLTKWordTokenizer().span_tokenize(s)) == expected
```

Out[19]:
```
True
```

In [20]:
```
expected = ['I', 'said', ',', '"', 'I', "'d", 'like', 'to',
... 'buy', 'some', "''", "good", 'muffins', '"', 'which', 'cost',
... '$', '3.88', 'each', 'in', 'New', '(', 'York', ')', '.', '"']
```

In [21]:
```
[s[start:end] for start, end in NLTKWordTokenizer().span_tokenize(s)] == expecte
```

Out[21]:
```
True
```

Testing treebank's detokenizer

In [22]:
```
from nltk.tokenize.treebank import TreebankWordDetokenizer
```

In [23]:
```
detokenizer = TreebankWordDetokenizer()
```

In [24]:
```
detokenizer.detokenize(word_tokenize(s1))
```

Out[24]:
```
'Natural language processing allows computers to understand and generate human
language, bridging the gap between technology and communication.'
```

In [25]:
```
s = "Well, we couldn't have this predictable, cliche-ridden, \"Touched by an Ang
```

In [26]:
```
detokenizer.detokenize(word_tokenize(s))
```

Out[26]:
```
'Well, we couldn\'t have this predictable, cliche-ridden, "Touched by an Angel"
(a show creator John Masius worked on) wanna-be if she didn\'t.'
```

In [27]:
```
s = '<A sentence> with (many) [kinds] of {parentheses}. "Sometimes it\'s inside
```

In [28]:
```
detokenizer.detokenize(word_tokenize(s))
```

Out[28]:
```
'<A sentence> with (many) [kinds] of {parentheses}. "Sometimes it\'s inside (qu
otes)". ("Sometimes the otherway around").'
```

In [29]:
```
 s = "The company spent $30,000,000 last year."
```

In [30]:
```
detokenizer.detokenize(word_tokenize(s))
```

Out[30]:
```
'The company spent $30,000,000 last year.'
```

In [31]:
```
s = "I've"
```

In [32]:
```
detokenizer.detokenize(word_tokenize(s))
```

Out[32]:
```
"I've"
```

In [33]:
```
s = "Don't"
```

In [34]:
```
detokenizer.detokenize(word_tokenize(s))
```

Out[34]:
```
"Don't"
```

In [35]:
```
s = "I'd"
```

In [36]:
```
detokenizer.detokenize(word_tokenize(s))
```

Out[36]:
```
"I'd"
```

Sentence tokenization in word_tokenize:

In [37]:
```
s7 = "I called Dr. Jones. I called Dr. Jones."
```

In [38]:
```
word_tokenize(s7)
```

Out[38]:
```
['I', 'called', 'Dr.', 'Jones', '.', 'I', 'called', 'Dr.', 'Jones', '.']
```

In [39]:
```
s8 = ("Ich muss unbedingt daran denken, Mehl, usw. fur einen "
...       "Kuchen einzukaufen. Ich muss.")
```

In [40]:
```
word_tokenize(s8)
```

Out[40]:
```
['Ich',
 'muss',
 'unbedingt',
 'daran',
 'denken',
 ',',
 'Mehl',
 ',',
 'usw',
 '.',
 'fur',
 'einen',
 'Kuchen',
 'einzukaufen',
 '.',
 'Ich',
 'muss',
 '.']
```

In [41]:
```
word_tokenize(s8, 'german')
```

Out[41]:
```
['Ich',
 'muss',
 'unbedingt',
 'daran',
 'denken',
 ',',
 'Mehl',
 ',',
 'usw.',
 'fur',
 'einen',
 'Kuchen',
 'einzukaufen',
 '.',
 'Ich',
 'muss',
 '.']
```

Regression Tests: Regexp Tokenizer

In [42]:
```
s = ("Good muffins cost $3.88\nin New York.  Please buy me\n"
...      "two of them.\n\nThanks.")
```

In [43]:
```
s2 = ("Alas, it has not rained today. When, do you think, "
...       "will it rain again?")
```

In [44]:
```
s3 = ("<p>Although this is <b>not</b> the case here, we must "
...       "not relax our vigilance!</p>")
```

In [45]:
```
from nltk.tokenize import regexp_tokenize
```

In [46]:
```
regexp_tokenize(s2, r'[,\.\?!"]\s*', gaps=False)
[', ', '. ', ', ', ', ', ', '?']
```

Out[46]:
```
[', ', '. ', ', ', ', ', ', '?']
```

In [47]:
```
regexp_tokenize(s2, r'[,\.\?!"]\s*', gaps=True)
```

Out[47]:
```
['Alas',
 'it has not rained today',
 'When',
 'do you think',
 'will it rain again']
```

Take care to avoid using capturing groups

In [48]:
```
regexp_tokenize(s3, r'</?[bp]>', gaps=False)
```

Out[48]:
```
['<p>', '<b>', '</b>', '</p>']
```

In [49]:
```
regexp_tokenize(s3, r'</?(?:b|p)>', gaps=False)
```

Out[49]:
```
['<p>', '<b>', '</b>', '</p>']
```

In [50]:
```
regexp_tokenize(s3, r'</?(?:b|p)>', gaps=True)
```

Out[50]:
```
['Although this is ',
 'not',
 ' the case here, we must not relax our vigilance!']
```

Named groups are capturing groups, and confuse the tokenizer:

In [51]:
```
regexp_tokenize(s3, r'</?(?P<named>b|p)>', gaps=False)
```

Out[51]:
```
['p', 'b', 'b', 'p']
```

In [52]:
```
regexp_tokenize(s3, r'</?(?P<named>b|p)>', gaps=True)
```

Out[52]:
```
['p',
 'Although this is ',
 'b',
 'not',
 'b',
 ' the case here, we must not relax our vigilance!',
 'p']
```

Make sure that nested groups don't confuse the tokenizer:

In [53]:
```
regexp_tokenize(s2, r'(?:h|r|l)a(?:s|(?:i|n0))', gaps=False)
```

Out[53]:
```
['las', 'has', 'rai', 'rai']
```

In [54]:
```
regexp_tokenize(s2, r'(?:h|r|l)a(?:s|(?:i|n0))', gaps=True)
```

Out[54]:
```
['A', ', it ', ' not ', 'ned today. When, do you think, will it ', 'n again?']
```

Back-references require capturing groups, and these are not supported:

In [55]:
```
regexp_tokenize("aabbbcccc", r'(.)\1')
```

Out[55]:
```
['a', 'b', 'c', 'c']
```

A simple sentence tokenizer '.(s+|$)'

In [56]:
```
regexp_tokenize(s, pattern=r'\.(?:\s+|$)', gaps=True)
```

Out[56]:
```
['Good muffins cost $3.88\nin New York',
 'Please buy me\ntwo of them',
 'Thanks']
```

Regression Tests: TweetTokenizer

TweetTokenizer is a tokenizer specifically designed for micro-blogging tokenization tasks

In [57]:
```
from nltk.tokenize import TweetTokenizer
```

In [58]:
```
tknzr = TweetTokenizer()
```

In [59]:
```
s0 = "This is a cooool #dummysmiley: :-) :-P <3 and some arrows < > -> <--"
```

In [60]:
```
tknzr.tokenize(s0)
```

Out[60]:
```
['This',
 'is',
 'a',
 'cooool',
 '#dummysmiley',
 ':',
 ':-)',
 ':-P',
 '<3',
 'and',
 'some',
 'arrows',
 '<',
 '>',
 '->',
 '<--']
```

In [61]:
```
s1 = "@Joyster2012 @CathStaincliffe Good for you, girl!! Best wishes :-)"
```

In [62]:
```
tknzr.tokenize(s1)
```

Out[62]:
```
['@Joyster2012',
 '@CathStaincliffe',
 'Good',
 'for',
 'you',
 ',',
 'girl',
 '!',
 '!',
 'Best',
 'wishes',
 ':-)']
```

In [63]:
```
s2 = "3Points for #DreamTeam Gooo BAILEY! :) #PBB737Gold @PBBabscbn"
```

In [64]:
```
tknzr.tokenize(s2)
```

Out[64]:
```
['3Points',
 'for',
 '#DreamTeam',
 'Gooo',
 'BAILEY',
 '!',
 ':)',
 '#PBB737Gold',
 '@PBBabscbn']
```

In [65]:
```
s3 = "@Insanomania They do... Their mentality doesn't :("
```

In [66]:
```
tknzr.tokenize(s3)
```

Out[66]:
```
['@Insanomania', 'They', 'do', '...', 'Their', 'mentality', "doesn't", ':(']
```

In [67]:
```
s4 = "RT @facugambande: Ya por arrancar a grabar !!! #TirenTirenTiren vamoo !!"
```

In [68]:
```
tknzr.tokenize(s4)
```

Out[68]:
```
['RT',
 '@facugambande',
 ':',
 'Ya',
 'por',
 'arrancar',
 'a',
 'grabar',
 '!',
 '!',
 '!',
 '#TirenTirenTiren',
 'vamoo',
 '!',
 '!']
```

In [69]:
```
tknzr = TweetTokenizer(reduce_len=True)
```

In [70]:
```
s5 = "@crushinghes the summer holidays are great but I'm so bored already :("
```

In [71]:
```
tknzr.tokenize(s5)
```

Out[71]:
```
['@crushinghes',
 'the',
 'summer',
 'holidays',
 'are',
 'great',
 'but',
 "I'm",
 'so',
 'bored',
 'already',
 ':(']
```

In [72]:
```
tknzr = TweetTokenizer(strip_handles=True, reduce_len=True)
```

In [73]:
```
s6 = '@remy: This is waaaaayyyy too much for you!!!!!!'
```

In [74]:
```
tknzr.tokenize(s6)
```

Out[74]:
```
[':', 'This', 'is', 'waaayyy', 'too', 'much', 'for', 'you', '!', '!', '!']
```

In [75]:
```
s7 = '@_willy65: No place for @chuck tonight. Sorry.'
```

In [76]:
```
tknzr.tokenize(s7)
```

Out[76]:
```
[':', 'No', 'place', 'for', 'tonight', '.', 'Sorry', '.']
```

In [77]:
```
tknzr = TweetTokenizer()
```

In [78]:
```
sentences = [
...     "This is a cooool #dummysmiley: :-) :-P <3 and some arrows < > -> <--",
...     "@jrmy: I'm REALLY HAPPYYY about that! NICEEEE :D :P",
...     "@_willy65: No place for @chuck tonight. Sorry."
... ]
```

In [79]:
```
tknzr.tokenize_sents(sentences)
```

Out[79]:
```
[['This',
  'is',
  'a',
  'cooool',
  '#dummysmiley',
  ':',
  ':-)',
  ':-P',
  '<3',
  'and',
  'some',
  'arrows',
  '<',
  '>',
  '->',
  '<--'],
 ['@jrmy',
  ':',
  "I'm",
  'REALLY',
  'HAPPYYY',
  'about',
  'that',
  '!',
  'NICEEEE',
  ':D',
  ':P'],
 ['@_willy65',
  ':',
  'No',
  'place',
  'for',
  '@chuck',
  'tonight',
  '.',
  'Sorry',
  '.']]
```

Regression Tests: PunktSentenceTokenizer

In [80]:
```
from nltk.tokenize import PunktSentenceTokenizer
```

In [81]:
```
pst = PunktSentenceTokenizer()
```

In [82]:
```
pst.tokenize('See Section 3).  Or Section 2).  ')
```

Out[82]:
```
['See Section 3).', 'Or Section 2).']
```

In [83]:
```
pst.tokenize('See Section 3.)  Or Section 2.)  ')
```

Out[83]:
```
['See Section 3.)', 'Or Section 2.)']
```

In [84]:
```
pst.tokenize('See Section 3.)  Or Section 2.)  ', realign_boundaries=False)
```

Out[84]:
```
['See Section 3.', ')  Or Section 2.', ')']
```

Two instances of PunktSentenceTokenizer should not share PunktParameters.

In [85]:
```
pst = PunktSentenceTokenizer()
```

In [86]:
```
pst2 = PunktSentenceTokenizer()
```

In [87]:
```
pst._params is pst2._params
```

Out[87]:
```
False
```

Testing mutable default arguments

In [88]:
```
from nltk.tokenize.punkt import PunktBaseClass, PunktTrainer, PunktSentenceToken
```

In [89]:
```
from nltk.tokenize.punkt import PunktLanguageVars, PunktParameters
```

In [90]:
```
pbc = PunktBaseClass(lang_vars = None, params = None)
```

In [91]:
```
type(pbc._params)
```

Out[91]:
```
nltk.tokenize.punkt.PunktParameters
```

In [92]:
```
type(pbc._lang_vars)
```

Out[92]:
```
nltk.tokenize.punkt.PunktLanguageVars
```

In [93]:
```
pt = PunktTrainer(lang_vars = None)
```

In [94]:
```
type(pt._lang_vars)
```

Out[94]:
```
nltk.tokenize.punkt.PunktLanguageVars
```

In [95]:
```
pst = PunktSentenceTokenizer(lang_vars=None)
```

In [96]:
```
type(pst._lang_vars)
```

Out[96]:
```
nltk.tokenize.punkt.PunktLanguageVars
```

In [97]:
```
pst = PunktSentenceTokenizer(lang_vars=None)
```

In [98]:
```
pst.tokenize(". This input starts with a dot. This used to cause issues.")
```

Out[98]:
```
['.', 'This input starts with a dot.', 'This used to cause issues.']
```

Regression Tests: align_tokens

Post-hoc alignment of tokens with a source string

In [99]:
```
from nltk.tokenize.util import align_tokens
```

In [100…
```
list(align_tokens([''],""))
```

Out[100…
```
[(0, 0)]
```

In [101…
```
list(align_tokens([''], " "))
```

Out[101…
```
[(0, 0)]
```

In [102…
```
list(align_tokens([], ""))
```

Out[102…
```
[]
```

In [103…
```
list(align_tokens([], " "))
```

Out[103…
```
[]
```

In [104…
```
list(align_tokens(['a'], "a"))
```

Out[104…
```
[(0, 1)]
```

In [105…
```
list(align_tokens(['abc', 'def'], "abcdef"))
```

Out[105…
```
[(0, 3), (3, 6)]
```

In [106…
```
list(align_tokens(['abc', 'def'], "abc def"))
```

Out[106…
```
[(0, 3), (4, 7)]
```

In [107…
```
list(align_tokens(['ab', 'cd'], "ab cd ef"))
```

Out[107…
```
[(0, 2), (3, 5)]
```

In [108…
```
list(align_tokens(['ab', 'cd', 'ef'], "ab cd ef"))
```

Out[108…
```
[(0, 2), (3, 5), (6, 8)]
```

Sentence tokenization in word_tokenize:

In [109…
```
s11 = "I called Dr. Jones. I called Dr. Jones."
```

In [110…
```
word_tokenize(s11)
```

Out[110…
```
['I', 'called', 'Dr.', 'Jones', '.', 'I', 'called', 'Dr.', 'Jones', '.']
```

In [111…
```
 s12 = ("Ich muss unbedingt daran denken, Mehl, usw. fur einen "
...        "Kuchen einzukaufen. Ich muss.")
```

In [112…
```
word_tokenize(s12)
```

Out[112…
```
['Ich',
 'muss',
 'unbedingt',
 'daran',
 'denken',
 ',',
 'Mehl',
 ',',
 'usw',
 '.',
 'fur',
 'einen',
 'Kuchen',
 'einzukaufen',
 '.',
 'Ich',
 'muss',
 '.']
```

In [113…
```
word_tokenize(s12, 'german')
```

Out[113…
```
['Ich',
 'muss',
 'unbedingt',
 'daran',
 'denken',
 ',',
 'Mehl',
 ',',
 'usw.',
 'fur',
 'einen',
 'Kuchen',
 'einzukaufen',
 '.',
 'Ich',
 'muss',
 '.']
```

In [ ]: