Word and Sentence Tokenization

WHAT IS TOKENIZATION?

Tokenization may be defined as the process of splitting the text into smaller parts called tokens, and is considered a crucial step in Natural Language Processing.

The tokens may be words or number or punctuation mark. Tokenization does this task by locating word boundaries. Ending point of a word and beginning of the next word is called word boundaries. Tokenization is also known as word segmentation.

CHALLENGES IN TOKENIZATION

Challenges in tokenization depends on the type of language. Languages such as English and French are referred to as space delimited as most of the words are separated from each other by white spaces. Languages such as Chinese and Thai are referred to as unsegmented as words do not have clear boundaries. Tokenising unsegmented language sentences requires additional lexical and morphological information.

Difficulties

- e-mails
- Phone number
- Address
- Abbreviations
- Sentence Boarder
- Period
- Ordinal numbers
- Date
- Time
- Names
- De-hyphenation
- Company name
-

Word Tokenisation

The word_tokenizer tokenise each words. Following python code explains about word_tokenizer.

```
import nltk
sentence = """I am . Amir! I want to become an actor. I am working hard for carrier."""
tokens = nltk.word_tokenize(sentence)
tokens
```

```
['I',
 'am',
 '.',
 'Amir',
 '!',
 'I',
 'want',
 'to',
 'become',
 'an',
 'actor',
 '.',
 'I',
 'am',
 'working',
 'hard',
 'for',
 'carrier',
 '.']
```

Sentence Tokenisation

The sent_tokenizer tokenise each sentences. Following python code explains about sent_tokenizer.

```
import nltk
sentence = """I am Amir. I want to become an actor. I am working hard for carrier."""
tokens = nltk.sent_tokenize(sentence)
tokens
```

```
['I am Amir.', 'I want to become an actor.', 'I am working hard for carrier.']
```

Multi-Word Expression Tokenizer

A `MWETokenizer` takes a string which has already been divided into tokens and retokenizes it, merging multi-word expressions into single tokens, using a lexicon of MWEs.

Following python code explains about `MWETokenizer`.

```
from nltk.tokenize import MWETokenizer
tokenizer = MWETokenizer([('a', 'little'), ('a', 'little', 'bit'), ('a', 'lot')])
tokenizer.add_mwe(('in', 'spite', 'of'))
tokenizer.tokenize('In a little or a little bit or a lot in spite of'.split())
```

```
['In', 'a_little', 'or', 'a_little_bit', 'or', 'a_lot', 'in_spite_of']
```

Regular-Expression Tokenizers

A `RegexpTokenizer` splits a string into substrings using a regular expression. For example, the following tokenizer forms tokens out of alphabetic sequences, money expressions, and any other non-whitespace sequences.

Following python code explains about `MWETokenizer`.

```
from nltk.tokenize import RegexpTokenizer
s = "Good muffins cost $38.8\nin New York.  Please buy me\ntwo of them.\n\nThanks."
tokenizer = RegexpTokenizer('\w+|\$[\f]+|\S+')
tokenizer.tokenize(s)
```

```
['Good',
 'muffins',
 'cost',
 '$38.8',
 'in',
 'New',
 'York',
 '.',
 'Please',
 'buy',
 'me',
 'two',
 'of',
 'them',
 '.',
 'Thanks',
 '.']
```

To identify Capitalised words

```
from nltk.tokenize import RegexpTokenizer
s = "Good muffins cost $38.8\nin New York.  Please buy me\ntwo of them.\n\nThanks."
tokenizer = RegexpTokenizer('[A-Z]\w+')
tokenizer.tokenize(s)
```

```
['Good', 'New', 'York', 'Please', 'Thanks']
```

Parameters:

pattern (*str*) – The pattern used to build this tokenizer. (This pattern must not contain capturing parentheses; Use non-capturing parentheses, e.g. (?:...), instead)

gaps (*bool*) – True if this tokenizer's pattern should be used to find separators between tokens; False if this tokenizer's pattern should be used to find the tokens themselves.

discard_empty (*bool*) – True if any empty tokens '' generated by the tokenizer should be discarded. Empty tokens can only be generated if *_gaps == True*.

White space Tokenizer

Tokenize a string on whitespace (space, tab, newline). In general, users should use the string `split()` method instead.

```
s = "Good muffins cost $3.88\nin New York. Please buy me\ntwo of them.\n\nThanks."
nltk.tokenize.WhitespaceTokenizer().tokenize(s)
```

```
['Good',
 'muffins',
 'cost',
 '$3.88',
 'in',
 'New',
 'York.',
 'Please',
 'buy',
 'me',
 'two',
 'of',
 'them.',
 'Thanks.']
```

Twitter-aware tokenizer

Twitter-aware tokenizer, designed to be flexible and easy to adapt to new domains and tasks. The basic logic is this:

1. The tuple regex_strings defines a list of regular expression strings.
2. The regex_strings strings are put, in order, into a compiled regular expression object called word_re.
3. The tokenization is done by word_re.findall(s), where s is the user-supplied string, inside the tokenize() method of the class Tokenizer.
4. When instantiating Tokenizer objects, there is a single option: preserve_case. By default, it is set to True. If it is set to False, then the tokenizer will downcase everything except for emoticons.

```
from nltk.tokenize import TweetTokenizer
tknzr = TweetTokenizer()
s = "This is a cooool #dummysmiley: :-) :-P <3 and some arrows < > -> <--"
tknzr.tokenize(s)
```

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

Examples using *strip_handles* and *reduce_len parameters*:
reduce_len: removes the extra letters in a word.

```
tknzr = TweetTokenizer(strip_handles=True, reduce_len=True)
s = '@remy: This is waaaaayyyy too much for you!!!!!!'
tknzr.tokenize(s)
```

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

S-Expression Tokenizer

SExprTokenizer is used to find parenthesized expressions in a string. In particular, it divides a string into a sequence of substrings that are either parenthesized expressions (including any nested parenthesized expressions), or other whitespace-separated tokens.

```
from nltk.tokenize import SExprTokenizer
tok = SExprTokenizer()
s = "(Good muffins) (cost $3.88)\nin (New York). Please buy me\ntwo of them.\n\nThanks."
tok.tokenize(s)
```

```
['(Good muffins)',
 '(cost $3.88)',
 'in',
 '(New York)',
 '. Please buy me\ntwo of them.\n\nThanks.']
```

Line Tokenizer

Tokenize a string into its lines optionally discarding the plank lines. This is noting but split("\n").

```
s = "Good muffins cost $3.88\nin New York. Please buy me\ntwo of them.\n\nThanks."
from nltk.tokenize import LineTokenizer
ltok = LineTokenizer()
ltok.tokenize(s)
```

```
['Good muffins cost $3.88',
 'in New York. Please buy me',
 'two of them.',
 'Thanks.']
```

Tab Tokenizer

```
s = "Good muffins \t cost $3.88\nin New York. Please \t buy me\ntwo of them.\n\nThanks."
import nltk
tok = nltk.tokenize.TabTokenizer()
tok.tokenize(s)
```

```
['Good muffins ',
 ' cost $3.88\nin New York. Please ',
 ' buy me\ntwo of them.\n\nThanks.']
```

Reg Exp Span Tokenizer

Return the offsets of the tokens in *s*, as a sequence of `(start, end)` tuples, by splitting the string at each successive match of *regexp*.

```
s = "Good muffins cost $3.88\nin New York. Please buy me\ntwo of them.\n\nThanks."
from nltk.tokenize.util import regexp_span_tokenize
import nltk
list(regexp_span_tokenize(s, r'\s'))
```

```
[(0, 4),
 (5, 12),
 (13, 17),
 (18, 23),
 (24, 26),
 (27, 30),
 (31, 36),
 (37, 43),
 (44, 47),
 (48, 50),
 (51, 54),
 (55, 57),
 (58, 63),
 (65, 72)]
```