NLTK :

Natural Language Toolkit

NLTK is a leading platform for building Python programs to work with human language data. It provides easy-to-use interfaces to [over 50 corpora and lexical resources](https://www.nltk.org/nltk_data/) such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning, wrappers for industrial-strength NLP libraries, and an active [discussion forum](https://groups.google.com/group/nltk-users).

Thanks to a hands-on guide introducing programming fundamentals alongside topics in computational linguistics, plus comprehensive API documentation, NLTK is suitable for linguists, engineers, students, educators, researchers, and industry users alike. NLTK is available for Windows, Mac OS X, and Linux. Best of all, NLTK is a free, open source, community-driven project.

NLTK has been called “a wonderful tool for teaching, and working in, computational linguistics using Python,” and “an amazing library to play with natural language.”

[Natural Language Processing with Python](https://www.nltk.org/book/) provides a practical introduction to programming for language processing. Written by the creators of NLTK, it guides the reader through the fundamentals of writing Python programs, working with corpora, categorizing text, analyzing linguistic structure, and more

Exemples:

Tokenize and tag some text: **>>> import** **nltk**

**>>>** sentence = """At eight o'clock on Thursday morning

**...** Arthur didn't feel very good."""

**>>>** tokens = nltk.word\_tokenize(sentence)

**>>>** tokens

['At', 'eight', "o'clock", 'on', 'Thursday', 'morning',

'Arthur', 'did', "n't", 'feel', 'very', 'good', '.']

**>>>** tagged = nltk.pos\_tag(tokens)

**>>>** tagged[0:6]

[('At', 'IN'), ('eight', 'CD'), ("o'clock", 'JJ'), ('on', 'IN'),

('Thursday', 'NNP'), ('morning', 'NN')]

Identify named entities:

**>>>** entities = nltk.chunk.ne\_chunk(tagged)

**>>>** entities

Tree('S', [('At', 'IN'), ('eight', 'CD'), ("o'clock", 'JJ'),

('on', 'IN'), ('Thursday', 'NNP'), ('morning', 'NN'),

Tree('PERSON', [('Arthur', 'NNP')]),

('did', 'VBD'), ("n't", 'RB'), ('feel', 'VB'),

('very', 'RB'), ('good', 'JJ'), ('.', '.')])

Display a parse tree:

**>>> from** **nltk.corpus** **import** treebank

**>>>** t = treebank.parsed\_sents('wsj\_0001.mrg')[0]

**>>>** t.draw()

*Natural Language Toolkit: Punkt sentence tokenizer*

*Punkt Sentence Tokenizer*

*This tokenizer divides a text into a list of sentences*

*by using an unsupervised algorithm to build a model for abbreviation*

*words, collocations, and words that start sentences. It must be*

*trained on a large collection of plaintext in the target language*

*before it can be used.*

*The NLTK data package includes a pre-trained Punkt tokenizer for*

*English.*

*>>> import nltk.data*

*>>> text = '''*

*... Punkt knows that the periods in Mr. Smith and Johann S. Bach*

*... do not mark sentence boundaries. And sometimes sentences*

*... can start with non-capitalized words. i is a good variable*

*... name.*

*... '''*

*>>> sent\_detector = nltk.data.load('tokenizers/punkt/english.pickle')*

*>>> print('\n-----\n'.join(sent\_detector.tokenize(text.strip())))*

*Punkt knows that the periods in Mr. Smith and Johann S. Bach*

*do not mark sentence boundaries.*

*-----*

*And sometimes sentences*

*can start with non-capitalized words.*

*-----*

*i is a good variable*

*name.*

*(Note that whitespace from the original text, including newlines, is*

*retained in the output.)*

*Punctuation following sentences is also included by default*

*(from NLTK 3.0 onwards). It can be excluded with the realign\_boundaries*

*flag.*

*>>> text = '''*

*... (How does it deal with this parenthesis?) "It should be part of the*

*... previous sentence." "(And the same with this one.)" ('And this one!')*

*... "('(And (this)) '?)" [(and this. )]*

*... '''*

*>>> print('\n-----\n'.join(*

*... sent\_detector.tokenize(text.strip())))*

*(How does it deal with this parenthesis?)*

*-----*

*"It should be part of the*

*previous sentence."*

*-----*

*"(And the same with this one.)"*

*-----*

*('And this one!')*

*-----*

*"('(And (this)) '?)"*

*-----*

*[(and this. )]*

*>>> print('\n-----\n'.join(*

*... sent\_detector.tokenize(text.strip(), realign\_boundaries=False)))*

*(How does it deal with this parenthesis?*

*-----*

*) "It should be part of the*

*previous sentence.*

*-----*

*" "(And the same with this one.*

*-----*

*)" ('And this one!*

*-----*

*')*

*"('(And (this)) '?*

*-----*

*)" [(and this.*

*-----*

*)]*

*However, Punkt is designed to learn parameters (a list of abbreviations, etc.)*

*unsupervised from a corpus similar to the target domain. The pre-packaged models*

*may therefore be unsuitable: use ``PunktSentenceTokenizer(text)`` to learn*

*parameters from the given text.*

*:class:`.PunktTrainer` learns parameters such as a list of abbreviations*

*(without supervision) from portions of text. Using a ``PunktTrainer`` directly*

*allows for incremental training and modification of the hyper-parameters used*

*to decide what is considered an abbreviation, etc.*

*The algorithm for this tokenizer is described in::*

*Kiss, Tibor and Strunk, Jan (2006): Unsupervised Multilingual Sentence*

*Boundary Detection. Computational Linguistics 32: 485-525.*

*"""*

*# TODO: Make orthographic heuristic less susceptible to overtraining*

*# TODO: Frequent sentence starters optionally exclude always-capitalised words*

*# FIXME: Problem with ending string with e.g. '!!!' -> '!! !'*

Tensorflow: