

# Chapter 11

## Workshop#2 N-grams in NLTK and Tokenization in SpaCy (Hour 3–4)

### 11.1 Introduction

Workshop 2 consists of two parts:

Part 1 will introduce N-gram language model using NLTK in Python and N-grams class to generate N-gram statistics on any sentence, text objects, whole document, literature to provide a foundation technique for text analysis, parsing and semantic analysis in subsequent workshops.

Part 2 will introduce spaCy, the second important NLP Python implementation tools not only for teaching and learning (like NLTK) but also widely used for NLP applications including text summarization, information extraction, and Q&A chat-bot. It is a critical mass to integrate with Transformer Technology in subsequent workshops.

### 11.2 What Is N-Gram?

N-gram is an algorithm based on statistical language model (Bird et al. 2009; Perkins 2014; Arumugam and Shanmugamani 2018), its basic idea is that contents such as phonemes, syllables, letters, words, or base pairs in texts are operated by a sliding window of size  $N$  to form a byte fragments sequence of length  $N$  (Sidorov 2019).

$N$  can be 1, 2, or other positive integer, although usually large  $N$  is not considered because they rarely occur.

Each byte fragment is called a gram, and the frequency of all grams is counted and filtered according to a pre-set threshold to form a list of key grams, which is the text's vector feature space, and each kind of gram in the list is a feature vector dimension.

## 11.3 Applications of N-Grams in NLP

N-gram models are widely used (Albrecht et al. 2020; Arumugam and Shanmugamani 2018; Hardeniya et al. 2016; Kedia and Rasu 2020) in:

- Speech recognition where phonemes and sequences of phonemes are modeled using a N-gram distribution.
- Parsing on words are modeled so that each N-gram is composed of N words. For language identification, sequences of characters/graphemes (e.g. letters of the alphabet) are modeled for different languages.
- Auto sentences completion
- Auto spell-check
- Semantic analysis

## 11.4 Generation of N-Grams in NLTK

NLTK (NLTK 2022; Bird et al. 2009; Perkins 2014) offers useful tools in NLP processing.

`Ngrams()` function in NLTK facilitates N-gram operation.

Python code uses N-grams in NLTK to generate N-grams for any text string. Try it and study how it works.

The following example is the first sentence of *A Scandal in Bohemia* from *The Adventures of Sherlock Holmes* (Doyle 2019): *To Sherlock Holmes she is always “The Woman.” I have seldom heard him mention her under any other name*, demonstrating how N-gram generator works in NLTK.

In[1] ▶

```
import nltk
from nltk import ngrams
sentence = input( "Enter the sentence: " )
n = int(input( "Enter the value of n: " ))
n_grams = ngrams(sentence.split(), n)
for grams in n_grams:
    print(grams)
```

Out[1]

```
Enter the sentence: To Sherlock Holmes she is always "The Woman". I
have seldom heard him mention her under any other name.
Enter the value of n: 2
('To', 'Sherlock') ('Sherlock', 'Holmes') ('Holmes', 'she') ('she', 'is')
('is', 'always') ('always', '"The') ('"The', 'Woman".') ('Woman"', 'I')
('I', 'have') ('have', 'seldom') ('seldom', 'heard') ('heard', 'him')
('him', 'mention') ('mention', 'her') ('her', 'under') ('under', 'any')
('any', 'other') ('other', 'name.')
```

Here are the Bigrams. Let's try Trigrams N=3.

In[2] 

```
import nltk
from nltk import ngrams
sentence = input( "Enter the sentence: " )
n = int(input( "Enter the value of n: " ))
n_grams = ngrams(sentence.split(), n)
for grams in n_grams:
    print(grams)
```

Out[2]

```
Enter the sentence: To Sherlock Holmes she is always "The Woman". I
have seldom heard him mention her under any other name.
Enter the value of n: 3
('To', 'Sherlock', 'Holmes') ('Sherlock', 'Holmes', 'she')
('Holmes', 'she', 'is') ('she', 'is', 'always') ('is', 'always', '"The')
('always', '"The', 'Woman".') ('"The', 'Woman".', 'I')
('Woman".', 'I', 'have') ('I', 'have', 'seldom') ('have', 'seldom', 'heard')
('seldom', 'heard', 'him') ('heard', 'him', 'mention') ('him', 'mention', 'her')
('mention', 'her', 'under') ('her', 'under', 'any') ('under', 'any', 'other')
('any', 'other', 'name.')
```

How about Quadrigram N=4? Let's use the same sentence.

In[3] 

```
import nltk
from nltk import ngrams
sentence = input( "Enter the sentence: " )
n = int(input( "Enter the value of n: " ))
n_grams = ngrams(sentence.split(), n)
for grams in n_grams:
    print(grams)
```

Out[3]

```
Enter the sentence: To Sherlock Holmes she is always "The Woman". I
have seldom heard him mention her under any other name.
Enter the value of n: 4
('To', 'Sherlock', 'Holmes', 'she') ('Sherlock', 'Holmes', 'she', 'is')
('Holmes', 'she', 'is', 'always') ('she', 'is', 'always', '"The')
('is', 'always', '"The', 'Woman".') ('always', '"The', 'Woman".', 'I')
('"The', 'Woman".', 'I', 'have') ('Woman".', 'I', 'have', 'seldom')
('I', 'have', 'seldom', 'heard') ('have', 'seldom', 'heard', 'him')
('seldom', 'heard', 'him', 'mention') ('heard', 'him', 'mention', 'her')
('him', 'mention', 'her', 'under') ('mention', 'her', 'under', 'any')
('her', 'under', 'any', 'other') ('under', 'any', 'other', 'name.')
```



NLTK offers an easy solution to generate N-gram of any N-number which are useful in N-gram probability calculations and text analysis



### Workshop 2.1 N-Grams on The Adventures of Sherlock Holmes

1. Read Adventures\_Holmes.txt text file.
2. Save contents into a string object "holmes\_doc".
3. Extract favorite paragraph from "holmes\_doc" into "holmes\_para".
4. Use above Python code to generate N-grams for N=3, N=4, and N=5.

## 11.5 Generation of N-Grams Statistics

Once N-grams are generated, the next step is to calculate term frequency (TF) of each N-grams from a document to list out top items.

NLTK-based Python codes extend previous example to create N-grams statistics to list out top 10 N-grams.

Let us try first two sentences of *A Scandal in Bohemia* from *The Adventures of Sherlock Holmes*.

In[4] ▶

sentence

Out[4]

'To Sherlock Holmes she is always "The Woman". I have seldom heard him mention her under any other name.'

Import RE package to do some simple text pre-processing:

In[5] ▶

```
import re, string


# get rid of all the XML markup
sentence = re.sub('<.*>', '', sentence)

# get rid of punctuation (except periods!)
punctuationNoPeriod = "[" + re.sub("\.", "", string.punctuation) + "]"
sentence = re.sub(punctuationNoPeriod, "", sentence)


# first get individual words
tokenized = sentence.split()

# and get a list of all the bi-grams
Bigrams = ngrams(tokenized, 2)
```

Review N-grams to see how they work:

In[6]  ngrams?

To generate N-gram statistics, first import “collections” class and invoke Counter() method over Bigrams to perform N-gram statistics analysis.

In[7] 

```
import collections
# get the frequency of each bigram in our corpus
BigramFreq = collections.Counter(Bigrams)

# what are the ten most popular ngrams in this corpus?
BigramFreq.most_common(10)
```

Out[7] 

```
[(('To', 'Sherlock'), 1),
 (('Sherlock', 'Holmes'), 1),
 (('Holmes', 'she'), 1),
 (('she', 'is'), 1),
 (('is', 'always'), 1),
 (('always', 'The'), 1),
 (('The', 'Woman'), 1),
 (('Woman', 'T'), 1),
 (('T', 'have'), 1),
 (('have', 'seldom'), 1)]
```



It is noted that top 10 bigram frequency are all with count 1.

This is because the sample sentence is short and does not contain any bigram(s) with a frequent bigram statistic. To sort out this problem, let us try a longer text.

The following example uses the whole first paragraph of *A Scandal in Bohemia* from *The Adventures of Sherlock Holmes* and see whether it has a preferable result.

The first paragraph looks like this:

In[8] ▶

```
first_para = "To Sherlock Holmes she is always the woman I have seldom
heard him mention her under any other name In his eyes she eclipses and
predominates the whole of her sex It was not that he felt any emotion akin
to love for Irene Adler All emotions and that one particularly were abhor-
rent to his cold precise but admirably balanced mind He was I take it the
most perfect reasoning and observing machine that the world has seen but
as a lover he would have placed himself in a false position He never spoke
of the softer passions save with a gibe and a sneer They were admirable
things for the observer—excellent for drawing the veil from men's mo-
tives and actions But for the trained reasoner to admit such intrusions into
his own delicate and finely adjusted temperament was to introduce a dis-
tracting factor which might throw a doubt upon all his mental results Grit
in a sensitive instrument or a crack in one of his own high power lenses
would not be more disturbing than a strong emotion in a nature such as
his And yet there was but one woman to him and that woman was the late
Irene Adler of dubious and questionable memory."
```

Let's review this first paragraph:

In[9] ▶

```
first_para
```

Use Python script to remove punctuation marks and tokenize the first\_para object:

In[10] ▶

```
import re, string
# get rid of all the XML markup
first_para = re.sub('<.*>', '', first_para)

# get rid of punctuation (except periods!)
punctuationNoPeriod = "[" + re.sub("\.", "", string.punctuation) + "]"
first_para = re.sub(punctuationNoPeriod, "", first_para)
# first get individual words
tokenized = first_para.split()

# and get a list of all the bi-grams
Bigrams = ngrams(tokenized, 2)
```

Use Counter() method of Collections class to calculate Bigram statistics of first\_para:

In[11] ▶

```
import collections
# get the frequency of each bigram in our corpus
BigramFreq = collections.Counter(Bigrams)

# what are the ten most popular ngrams in this corpus?
BigramFreq.most_common(10)
```

Out[11]

```
[(('in', 'a'), 3),
 (('Irene', 'Adler'), 2),
 (('and', 'that'), 2),
 (('for', 'the'), 2),
 (('his', 'own'), 2),
 (('To', 'Sherlock'), 1),
 (('Sherlock', 'Holmes'), 1),
 (('Holmes', 'she'), 1),
 (('she', 'is'), 1),
 (('is', 'always'), 1)]
```



The results are satisfactory. It is noted that bigram *in a* has the most occurrence frequency, i.e. three times while four other bigrams: *Irene Adler*, *and that*, *for the*, *his own* have occurred twice each within the paragraph. Bigram *in a*, *and that* and *for the* are frequently used English phrases which occurred in almost every text document. How about *To Sherlock* and *Irene Adler*? There are two N-gram types frequently used in N-gram Language Model studied in Chap. 2. One is the frequently used N-gram phrase in English like *in a*, *and that* and *for that* in our case. These bigrams are common phrases in other documents and literature writings. Another is domain-specific N-grams. These types are only frequently used in specific domain, documents, and genre of literatures. Hence, *To Sherlock* and *Irene Adler* are frequently used related to this story only and not in other situations.



### Workshop 2.2 N-grams Statistics on The Adventures of Sherlock Holmes

1. Read Adventures\_Holmes.txt text file.
2. Save contents into a string object "holmes\_doc".
3. Generate a more representative N-gram statistic using the whole holmes\_doc.
4. Generate top 10 N-grams summary for N=3, N=4, and N=5.
5. Review results and comments on pattern(s) found.

Bigram analysis is required to examine which bigrams are commonly used not only on single paragraph but also for the whole document or literature. Remember in Workshop 1 NLTK has a built-in list of tokenized sample literatures in `nltk.book`. Let us refer them first by using the `nltk.book` import statement.

```
In[12] # Let's load some sample books from the nltk databank
import nltk
from nltk.book import *
```

```
Out[12] *** Introductory Examples for the NLTK Book ***
Loading text1, ..., text9 and sent1, ..., sent9
Type the name of the text or sentence to view it.
Type: 'texts()' or 'sents()' to list the materials.
text1: Moby Dick by Herman Melville 1851
text2: Sense and Sensibility by Jane Austen 1811
text3: The Book of Genesis
text4: Inaugural Address Corpus
text5: Chat Corpus
text6: Monty Python and the Holy Grail
text7: Wall Street Journal
text8: Personals Corpus
text9: The Man Who Was Thursday by G . K . Chesterton 1908
```

Check with text1 to see what they are:

```
In[13] text1
```

```
Out[13] <Text: Moby Dick by Herman Melville 1851>
```

or download using `nltk.corpus.gutenberg.words()` from Project Gutenberg of copyright clearance classic literature (Gutenberg, 2022). Let's use this method to download *Moby Dick* (Melville, 2006).

```
In[14] import nltk.corpus
from nltk.text import Text
moby = Text(nltk.corpus.gutenberg.words('melville-moby_dick.txt'))
```

```
In[15] moby
```

```
Out[15] <Text: Moby Dick by Herman Melville 1851>
```


Review the first 50 elements of *Moby Dick* text object to see whether they are tokenized.

```
In[16] moby[1:50]
```



Out[16] ['Moby', 'Dick', 'by', 'Herman', 'Melville', '1851', ''], 'ETYMOLOGY',  
'', '(', 'Supplied', 'by', 'a', 'Late', 'Consumptive', 'Usher', 'to', 'a',  
'Grammar', 'School', ')', 'The', 'pale', 'Usher', '--', 'threadbare', 'in',  
'coat', ',', 'heart', ',', 'body', ',', 'and', 'brain', ',', 'T', 'see', 'him', 'now',  
'', 'He', 'was', 'ever', 'dusting', 'his', 'old', 'lexicons', 'and']

Use *Collections* class and *ngrams()* method for Bigram statistics to identify top 20 most frequently bigrams occurred for the entire *Moby Dick* literature.

In[17]  **import** collections  
# and get a list of all the bi-grams  
Bigrams = ngrams(moby, 2)  
  
# get the frequency of each bigram in our corpus  
BigramFreq = collections.Counter(Bigrams)  
  
# what are the 20 most popular ngrams in this corpus?  
BigramFreq.most\_common(20)

Out[17] [((',', 'and'), 2607), (('of', 'the'), 1847), (('"', 's'), 1737), (('in', 'the'), 1120),  
((',', 'the'), 908), ((';', 'and'), 853), (('to', 'the'), 712), (('(', 'But'), 596),  
((',', 'that'), 584), (('(', '""'), 557), (('(', 'as'), 523), (('(', 'T'), 461),  
((',', 'he'), 446), (('from', 'the'), 428), (('(', 'in'), 402), (('of', 'his'), 371),  
(('the', 'whale'), 369), (('(', 'The'), 369), (('and', 'the'), 357),  
((',', 'but'), 340)]



### Workshop 2.3 N-grams Statistics with removal of unnecessary punctuations

The results are average and unsatisfactory. It is noted that *and*, *of the*, *s* and *in the* are top 4 bigrams occurred in the entire *Moby Dick* literature. It is average since these bigrams are common English usage but original bigram statistics in simple sentences required to remove all punctuations by:

1. List out all punctuations required to remove.
2. Revise above Python script to remove these punctuation symbol from the token list.
3. Generate a top 20 Bigram summary for *Moby Dick* without punctuations.
4. Use sample method to generate (cleaned) Bigram statistics from *Moby Dick*, *Adventures of Sherlock Holmes*, *Sense and Sensibility*, *Book of Genesis*, *Inaugural Address Corpus*, and *Wall Street Journal*.
5. Verify results and comments of any pattern(s) found.
6. Try the same analysis for Trigram (N=3) and Quadrigram (N=4) to find any pattern(s).

## 11.6 spaCy in NLP

### 11.6.1 What Is spaCy?

SpaCy (2022) is a free, open-source library for advanced NLP written in Python and Cython programming languages.

The library is published under MIT license developed by Dr. Matthew Honnibal and Dr. Ines Montani, founders of software company Explosion.

SpaCy is designed specifically for production use and build NLP applications to process large volumes of text (Altinok 2021; Srinivasa-Desikan 2018; Vasiliev 2020) different from NLTK focused on teaching and learning perspective.

It also provides workflow pipelines for machine learning and deep learning tools that can integrate with common platforms such as PyTorch, MXNet, and TensorFlow with its machine learning library called *Thinc*. spaCy provides recurrent neural models such as convolution neural networks (CNN) by adopting Thinc for NLP implementation such as Dependency Parsing (DP), NER (Named Entity Recognition), POS Tagging, and Text Classification and other advanced NLP applications such as Natural Language Understanding (NLU) systems, Information Retrieval (IR), Information Extraction (IE) systems, and Question-and-Answer Chatbot systems.

A spaCy system architecture is shown in Fig. 11.1, its major features support:

- NLP-based statistical models for over 19 commonly used languages,
- tokenization tools implementation for over 60 international languages,
- NLP pipeline components include NER, POS Tagging, DP, Text Classification, and Chatbot implementation,
- integration with common Python platforms such as TensorFlow, PyTorch and other high-level frameworks,
- integration with the latest Transformer and BERT technologies,
- user-friendly modular system packaging, evaluation, and deployment tools.

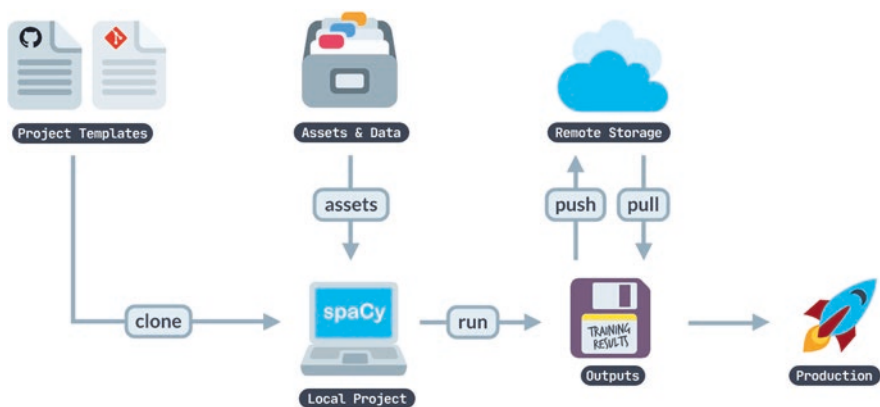


Fig. 11.1 System architecture of spaCy

## 11.7 How to Install spaCy?

SpaCy can be installed in MacOS/OSX, MS Windows, and Linux platforms (SpaCy 2022) as per other Python-based development tools like NLTK.

spaCy.io provides a one-stop-process for users to select own spaCy (1) language(s) as trained pipelines, (2) optimal target in system efficiency vs accuracy for NLP applications development based on a large dataset and lexical database, and (3) download appropriate APIs and modules to maximize efficiency under CPU and GPU hardware configuration. Figure 11.2 shows a Windows-based PIP download environment using CUDA 11.3 GPU in English as trained pipelines and target for speed efficiency over accuracy.

The screenshot displays the spaCy configuration selection interface. It features several sections with dropdown menus and checkboxes:

- Operating system:** macOS / OSX, **Windows**, Linux
- Platform:** **x86**, ARM / M1
- Package manager:** **pip**, conda, from source
- Hardware:** CPU, **GPU**, CUDA 11.3
- Configuration:** ☐ virtual env, ☐ train models
- Trained pipelines:** ☐ Catalan, ☐ Chinese, ☐ Danish, ☐ Dutch, ☒ English, ☐ French, ☐ German, ☐ Greek, ☐ Italian, ☐ Japanese, ☐ Lithuanian, ☐ Macedonian, ☐ Multi-language, ☐ Norwegian Bokmål, ☐ Polish, ☐ Portuguese, ☐ Romanian, ☐ Russian, ☐ Spanish
- Select pipeline for:** **efficiency**, accuracy

Below the configuration section, a terminal window shows the following commands:

```
$ pip install -U pip setuptools wheel
$ pip install -U 'spacy[cuda113]'
$ python -m spacy download en_core_web_sm
```

Fig. 11.2 Screenshot of spaCy configuration selection

## 11.8 Tokenization using spaCy

Tokenization is an operation in NLP. spaCy provides an easy-to-use scheme to tokenize any text document into sentences like NLTK and further tokenize sentences into words.

This section uses *Adventures\_Holmes.txt* as example to demonstrate tokenization in spaCy.

### 11.8.1 Step 1: Import spaCy Module

```
In[18] ▶ import spacy
```

### 11.8.2 Step 2: Load spaCy Module "en\_core\_web\_sm".

Use en\_core\_web\_md-3.2.0 package for English pipeline optimized for CPU in the current platform with components including: tok2vec, tagger, parser, sender, ner, attribute\_ruler, lemmatizer.

```
In[19] ▶ nlp = spacy.load( "en_core_web_sm" )
```


### 11.8.3 Step 3: Open and Read Text File "Adventures\_Holmes.txt" Into file\_handler "fholmes"

Note: Since text file already exists, skip try-except module to save programming steps.

```
In[20] ▶ fholmes = open( "Adventures_Holmes.txt", "r", encoding="utf-8" )
```

### 11.8.4 Step 4: Read Adventures of Sherlock Holmes

Use read() method to read whole text document as a complex string object "holmes".

In[21]	 <pre>holmes = fholmes.read() holmes</pre>
Out[21]	<pre>'\uffeffThe Adventures of Sherlock Holmes\n\u00aby Arthur Conan Doyle\n\n\nContents\n\n I. A Scandal in Bohemia\n II. The Red-Headed League\n III. A Case of Identity\n IV. The Boscombe Valley Mystery\n V. The Five Orange Pips\n VI. The Man with the Twisted Lip\n VII. The Adventure of the Blue Carbuncle\n VIII. The Adventure of the Speckled Band\n IX. The Adventure of the Engineer's Thumb\n X. The Adventure of the Noble Bachelor\n XI. The Adventure of the Beryl Coronet\n XII.\n\n... '</pre>

### 11.8.5 Step 5: Replace All Newline Symbols

Replace all newline characters "\n" into space characters.

```
In[22] holmes = holmes.replace( "\n", " " )
```

### 11.8.6 Step 6: Simple Counting

Review total number of characters in *The Adventures of Sherlock Holmes* and examine the result document.

In[23]	len(holmes)
Out[23]	580632
In[24]	holmes
Out[24]	<pre>"u00ffThe Adventures of Sherlock Holmes by Arthur Conan Doyle  Contents  I. A Scandal in Bohemia  II. The Red-Headed League  III. A Case of Identity  IV. The Boscombe Valley Mystery  V. The Five Orange Pips  VI. The Man with the Twisted Lip  VII. The Adventure of the Blue Carbuncle  VIII. The Adventure of the Speckled Band  IX. The Adventure of the Engineer's Thumb  X. The Adventure of the Noble Bachelor  XI. The Adventure of the Beryl Coronet  XII. The Adventure of the Copper Beeches  I. A SCANDAL IN BOHEMIA  I. To Sherlock Holmes she is always _the_ woman. ... "</pre>

### 11.8.7 Step 7: Invoke `nlp()` Method in `spaCy`

SpaCy `nlp()` method is an important Text Processing Pipeline to initialize `nlp` object (English in our case) for NLP processing such as tokenization. It will convert any text string object into a `nlp` object.

Study `nlp()` docstring to see how it works.

```
In[25]  ➤ nlp?
```

```
In[26]  ➤ holmes_doc = nlp(holmes)
```


```
In[27]  ➤ holmes_doc
```

```
Out[27] The Adventures of Sherlock Holmes by Arthur Conan Doyle Contents I.
A Scandal in Bohemia II. The Red-Headed League III. A Case of
Identity IV. The Boscombe Valley Mystery V. The Five Orange Pips
VI. The Man with the Twisted Lip VII. The Adventure of the Blue
Carbuncle VIII. The Adventure of the Speckled Band IX. The Adven-
ture of the Engineer's Thumb X. The Adventure of the Noble Bachelor
XI. The Adventure of the Beryl Coronet XII. The Adventure of the
Copper Beeches I. A SCANDAL IN BOHEMIA I. To Sherlock Holmes
she is always _the_ woman. I have seldom heard him mention her under any
other name. In his eyes she eclipses and predominates the whole of her sex.
It was not that he felt any emotion akin to love for Irene Adler. All emotions,
and that one particularly, were abhorrent to his cold, precise but admirably
balanced mind. ...
```

### 11.8.8 Step 8: Convert Text Document Into Sentence Object


SpaCy is practical for text document tokenization to convert text document object into (1) sentence objects and (2) tokens.

This example uses *for-in* statement to convert the whole Sherlock Holmes document into `holmes_sentences`.

In[28]  holmes\_sentences = [sentence.text for sentence in holmes\_doc.sents]  
holmes\_sentences

Out[28] ['\uffeffThe Adventures of Sherlock Holmes by Arthur Conan Doyle Con-  
tents I. A Scandal in Bohemia II.,  
' The Red-Headed League III.,  
' A Case of Identity IV.,  
' The Boscombe Valley Mystery V. The Five Orange Pips VI.,  
' The Man with the Twisted Lip VII.,  
' The Adventure of the Blue Carbuncle VIII.,  
' The Adventure of the Speckled Band IX.,  
' The Adventure of the Engineer's Thumb X. The Adventure of the  
Noble Bachelor XI.,  
' The Adventure of the Beryl Coronet XII.,  
' The Adventure of the Copper Beeches I. A SCANDAL IN BOHEMIA  
I. To Sherlock Holmes she is always \_the\_ woman.'... ']

Examine the structure of spaCy sentences and see what can be found.

In[29]  holmes\_sentences?

Study the numbers of sentences contained in *The Adventures of Sherlock Holmes*.

In[30]  len(holmes\_sentences)

Out[30] 6830

List out sentence numbers 50<sup>th</sup> – 59<sup>th</sup> to review.

```
In[31]  holmes_sentences[50:60]
```

```
Out[31] ['You would certainly have been burned, had you lived a few centuries ago.',
        'It is true that I had a country walk on Thursday and came home in a dreadful
        mess, but as I have changed my clothes I can't imagine how you deduce it.',
        'As to Mary Jane, she is incorrigible, and my wife has given her notice, but
        there, again, I fail to see how you work it out.',
        'He chuckled to himself and rubbed his long, nervous hands together.',
        '"It is simplicity itself," said he; "my eyes tell me that on the inside of your
        left shoe, just where the firelight strikes it, the leather is scored by six almost
        parallel cuts.',
        'Obviously they have been caused by someone who has very carelessly
        scraped round the edges of the sole in order to remove crusted mud from it.',
        'Hence, you see, my double deduction that you had been out in vile weather,
        and that you had a particularly malignant boot-slitting specimen of the Lon-
        don slavey.',
        'As to your practice, if a gentleman walks into my rooms smelling of iodo-
        form, with a black mark of nitrate of silver upon his right forefinger, and a
        bulge on the right side of his top-hat to show where he has secreted his stetho-
        scope, I must be dull, indeed, if I do not pronounce him to be an active
        member of the medical profession.',
        'I could not help laughing at the ease with which he explained his process
        of deduction.',
        '"When I hear you give your reasons," I remarked, "the thing always appears
        to me to be so ridiculously simple that I could easily do it myself, though at
        each successive instance of your reasoning I am baffled until you explain
        your process.']
```

### 11.8.9 Step 9: Directly Tokenize Text Document

Tokenize text document into word tokens by using “token” object in spaCy instead of text document object extraction into sentence list object. Study how it operates.

```
In[32]  holmes_words = [token.text for token in holmes_doc]
        holmes_words [130:180]
```

```
Out[32] ['To', 'Sherlock', 'Holmes', 'she', 'is', 'always', '_', 'the', '_', 'woman',
        '.', 'I', 'have', 'seldom', 'heard', 'him', 'mention', 'her', 'under', 'any',
        'other', 'name', '.', 'In', 'his', 'eyes', 'she', 'eclipses', 'and', 'predominates',
        'the', 'whole', 'of', 'her', 'sex', '.', 'It', 'was', 'not', 'that', 'he', 'felt', 'any',
        'emotion', 'akin', 'to', 'love', 'for', 'Irene', 'Adler']
```

```
In[33]  holmes_words?
```



In[34]  `len(holmes_words)`

Out[34] 133749

In[35]  `nltk_homles_tokens = nltk.word_tokenize(holmes)`

In[36]  `nltk_homles_tokens [104:153]`

Out[36] ['To', 'Sherlock', 'Holmes', 'she', 'is', 'always', 'the\_', 'woman',  
, 'I', 'have', 'seldom', 'heard', 'him', 'mention', 'her', 'under', 'any',  
, 'other', 'name', 'In', 'his', 'eyes', 'she', 'eclipses', 'and', 'predominates',  
, 'the', 'whole', 'of', 'her', 'sex', 'It', 'was', 'not', 'that', 'he', 'felt', 'any',  
, 'emotion', 'akin', 'to', 'love', 'for', 'Irene', 'Adler', '.']



According to extracted tokens, they seem to be identical.

1. Are they 100% identical?
2. What is/are the difference(s)?
3. Which one is better?



#### Workshop 2.4 SpaCy or NLTK: Which one is Faster?

In many applications, especially in AI and NLP application, *speed* (i.e. efficiency) is one of the most important consideration because:

1. Many AI and NLP applications involve a huge data/database/databank for system training with huge population size, e.g. Lexical database of English and Chinese. So, whether an NLP engine/application is fast enough in every NLP operation such as tokenization, tagging and POS tagging, and parsing is an important factor.
2. In many AI-based related NLP application such as Deep Learning for real-time Information Extraction, it involves tedious network training and learning process, how efficient of every NLP operation is a critical process to decide whether NLP application can be used in real-world scenario.

This workshop studies how efficient NLTK vs spaCy in terms of text document Tokenization.

To achieve this, integrate Python codes of NTLK/spaCy document tokenization with Timer object - time.

1. Implement tokenization codes in NTLK and spaCy to time tokenization time by using time object, the following codes can be used as reference.
2. Examine time taken for Tokenization process of "Adventures\_Holmes.txt" using NLTK vs spaCy methods.
3. Which one is faster? or are they similar? Why?
4. How about Document->Text efficiency? Compare NTLK vs spaCy on Doc->Text efficiency.

Hint: Like spaCy, NLTK can also implement Document->Text by two simple codes:

```
nltk_tokenizer = nltk.data.load("tokenizers/punkt/english.pickle")
nltk_sentences = tokenizer.tokenize(holmes) # holmes is the text document string object
```

In[37]  *# Sample code for Efficiency Performance of the NLP Engine*

```
import nltk # or spacy
import time
start = time.time()
#
# YOUR NTLK or spaCy Tokenization codes
#
print( "Time taken: %s s" % (time.time() - start))
```

Out[37] 0.0s

## References

- Albrecht, J., Ramachandran, S. and Winkler, C. (2020) Blueprints for Text Analytics Using Python: Machine Learning-Based Solutions for Common Real World (NLP) Applications. O'Reilly Media.
- Altinok, D. (2021) Mastering spaCy: An end-to-end practical guide to implementing NLP applications using the Python ecosystem. Packt Publishing.
- Arumugam, R., & Shanmugamani, R. (2018). Hands-on natural language processing with python. Packt Publishing.
- Bird, S., Klein, E., and Loper, E. (2009). Natural language processing with python. O'Reilly.
- Doyle, A. C. (2019) The Adventures of Sherlock Holmes (AmazonClassics Edition). AmazonClassics.
- Gutenberg (2022) Project Gutenberg official site. <https://www.gutenberg.org/> Accessed 16 June 2022.
- Hardeniya, N., Perkins, J. and Chopra, D. (2016) Natural Language Processing: Python and NLTK. Packt Publishing.
- Melville, H. (2006) Moby Dick. Hard Press.
- Kedia, A. and Rasu, M. (2020) Hands-On Python Natural Language Processing: Explore tools and techniques to analyze and process text with a view to building real-world NLP applications. Packt Publishing.
- NLTK (2022) NLTK official site. <https://www.nltk.org/>. Accessed 16 June 2022.
- Perkins, J. (2014). Python 3 text processing with NLTK 3 cookbook. Packt Publishing Ltd.
- SpaCy (2022) spaCy official site. <https://spacy.io/>. Accessed 16 June 2022.
- Sidorov, G. (2019) Syntactic n-grams in Computational Linguistics (SpringerBriefs in Computer Science). Springer.
- Srinivasa-Desikan, B. (2018). Natural language processing and computational linguistics: A practical guide to text analysis with python, gensim, SpaCy, and keras. Packt Publishing, Limited.
- Vasiliev, Y. (2020) Natural Language Processing with Python and spaCy: A Practical Introduction. No Starch Press.