

# NLTK 자연어 처리 패키지

NLTK(Natural Language Toolkit) 패키지는 교육용으로 개발된 자연어 처리 및 문서 분석용 파이썬 패키지다. 다양한 기능 및 예제를 가지고 있으며 실무 및 연구에서도 많이 사용된다.

NLTK 패키지가 제공하는 주요 기능은 다음과 같다.

- 말뭉치
- 토큰 생성
- 형태소 분석
- 품사 태깅

## 말뭉치

말뭉치(corpus)는 자연어 분석 작업을 위해 만든 샘플 문서 집합을 말한다. 단순히 소설, 신문 등의 문서를 모아놓은 것도 있지만 품사, 형태소, 등의 보조적 의미를 추가하고 쉬운 분석을 위해 구조적인 형태로 정리해 놓은 것을 포함한다. NLTK 패키지의 corpus 서브패키지에서는 다양한 연구용 말뭉치를 제공한다. 이 목록은 전체 corpus의 일부일 뿐이다. 말뭉치 자료는 설치시에 제공되지 않고 download 명령으로 사용자가 다운로드 받아야 한다. nltk.download("book") 명령을 실행하면 NLTK 패키지 사용자 설명서에서 요구하는 대부분의 말뭉치를 다운로드 받아준다.

In [1]:

```
import nltk
nltk.download("book", quiet=True)
from nltk.book import *
```

```
*** 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
```

예를 들어 저작권이 말소된 문학작품을 포함하는 gutenberg 말뭉치에는 다음과 같은 작품이 샘플로 포함되어 있다.

In [2]:

```
nltk.corpus.gutenberg.fileids()
```

Out [2]:

```
['austen-emma.txt',  
'austen-persuasion.txt',  
'austen-sense.txt',  
'bible-kjv.txt',  
'blake-poems.txt',  
'bryant-stories.txt',  
'burgess-busterbrown.txt',  
'carroll-alice.txt',  
'chester-tonball.txt',  
'chester-tonbrown.txt',  
'chester-tonthursday.txt',  
'edgeworth-parents.txt',  
'melville-moby_dick.txt',  
'milton-paradise.txt',  
'shakespeare-caesar.txt',  
'shakespeare-hamlet.txt',  
'shakespeare-macbeth.txt',  
'whitman-leaves.txt']
```

이 중 제인 오스틴의 엠마 문서를 살펴보면 다음과 같이 원문 형태 그대로를 포함하고 있다.

In [3]:

```
emma_raw = nltk.corpus.gutenberg.raw("austen-emma.txt")
print(emma_raw[:1302])
```

[Emma by Jane Austen 1816]

VOLUME I

CHAPTER I

Emma Woodhouse, handsome, clever, and rich, with a comfortable home and happy disposition, seemed to unite some of the best blessings of existence; and had lived nearly twenty-one years in the world with very little to distress or vex her.

She was the youngest of the two daughters of a most affectionate, indulgent father; and had, in consequence of her sister's marriage, been mistress of his house from a very early period. Her mother had died too long ago for her to have more than an indistinct remembrance of her caresses; and her place had been supplied by an excellent woman as governess, who had fallen little short of a mother in affection.

Sixteen years had Miss Taylor been in Mr. Woodhouse's family, less as a governess than a friend, very fond of both daughters, but particularly of Emma. Between \_them\_ it was more the intimacy of sisters. Even before Miss Taylor had ceased to hold the nominal office of governess, the mildness of her temper had hardly allowed her to impose any restraint; and the shadow of authority being now long passed away, they had been living together as friend and friend very mutually attached, and Emma doing just what she liked; highly esteeming Miss Taylor's judgment, but directed chiefly by her own.

## 토큰 생성

자연어 문서를 분석하기 위해서는 우선 긴 문자열을 분석을 위한 작은 단위로 나누어야 한다. 이 문자열 단위를 토큰(token)이라고 하고 이렇게 문자열을 토큰으로 나누는 작업을 토큰 생성(tokenizing)이라고 한다. 영문의 경우에는 문장, 단어 등을 토큰으로 사용하거나 정규 표현식을 쓸 수 있다.

문자열을 토큰으로 분리하는 함수를 토큰 생성 함수(tokenizer)라고 한다. 토큰 생성 함수는 문자열을 입력받아 토큰 문자열의 리스트를 출력한다.

In [4]:

```
from nltk.tokenize import sent_tokenize
print(sent_tokenize(emma_raw[:1000])[3])
```

Sixteen years had Miss Taylor been in Mr. Woodhouse's family, less as a governess than a friend, very fond of both daughters, but particularly of Emma.

In [5]:

```
from nltk.tokenize import word_tokenize
word_tokenize(emma_raw[50:100])
```

Out[5]:

```
['Emma',
 'Woodhouse',
 ',',
 ',',
 'handsome',
 ',',
 'clever',
 ',',
 'and',
 'rich',
 ',',
 'with',
 'a']
```

In [6]:

```
from nltk.tokenize import RegexpTokenizer
retokenize = RegexpTokenizer("[Ww]+")
retokenize.tokenize(emma_raw[50:100])
```

Out[6]:

```
['Emma', 'Woodhouse', 'handsome', 'clever', 'and', 'rich', 'with', 'a']
```

## 형태소 분석

형태소(morpheme)는 언어학에서 일정한 의미가 있는 가장 작은 말의 단위를 뜻한다. 보통 자연어 처리에서는 토큰으로 형태소를 이용한다. 형태소 분석(morphological analysis)이란 단어로부터 어근, 접두사, 접미사, 품사 등 다양한 언어적 속성을 파악하고 이를 이용하여 형태소를 찾아내거나 처리하는 작업이다. 형태소 분석의 예로는 다음과 같은 작업이 있다.

- 어간 추출(stemming)
- 원형 복원(lemmatizing)
- 품사 부착(Part-Of-Speech tagging)

## 어간 추출과 원형 복원

어간 추출(stemming)은 변화된 단어의 접미사나 어미를 제거하여 같은 의미를 가지는 형태소의 기본형을 찾는 방법이다. NLTK는 PorterStemmer LancasterStemmer 등을 제공한다. 어간 추출법은 단순히 어미를 제거할 뿐이므로 단어의 원형의 정확히 찾아주지는 않는다.

In [7]:

```
from nltk.stem import PorterStemmer, LancasterStemmer

st1 = PorterStemmer()
st2 = LancasterStemmer()

words = ["fly", "flies", "flying", "flew", "flown"]

print("Porter Stemmer  :", [st1.stem(w) for w in words])
print("Lancaster Stemmer:", [st2.stem(w) for w in words])
```

```
Porter Stemmer  : ['fli', 'fli', 'fli', 'flew', 'flown']
Lancaster Stemmer: ['fly', 'fli', 'fly', 'flew', 'flown']
```

원형 복원(lemmatizing)은 같은 의미를 가지는 여러 단어를 사전형으로 통일하는 작업이다. 품사(part of speech)를 지정하는 경우 좀 더 정확한 원형을 찾을 수 있다.

In [8]:

```
from nltk.stem import WordNetLemmatizer

lm = WordNetLemmatizer()

[lm.lemmatize(w, pos="v") for w in words]
```

Out[8]:

```
['fly', 'fly', 'fly', 'fly', 'fly']
```

## 품사 부착

품사(POS, part-of-speech)는 낱말을 문법적인 기능이나 형태, 뜻에 따라 구분한 것이다. 품사의 구분은 언어마다 그리고 학자마다 다르다. 예를 들어 NLTK에서는 펜 트리뱅크 태그세트(Penn Treebank Tagset)라는 것을 이용한다. 다음은 펜 트리뱅크 태그세트에서 사용하는 품사의 예이다.

- NNP: 단수 고유명사
- VB: 동사
- VBP: 동사 현재형
- TO: to 전치사
- NN: 명사(단수형 혹은 집합형)
- DT: 관형사

nltk.help.upenn\_tagset 명령으로 자세한 설명을 볼 수 있다.

In [9]:

```
nltk.help.upenn_tagset("VB")
```

```
VB: verb, base form
ask assemble assess assign assume atone attention avoid bake balkanize
bank begin behold believe bend benefit bevel beware bless boil bomb
boost brace break bring broil brush build ...
```

pos\_tag 명령을 사용하면 단어 토큰에 품사를 부착하여 튜플로 출력한다. 다음 예문에서 refuse, permit이라는 같은 철자의 단어가 각각 동사와 명사로 다르게 품사 부착된 것을 볼 수 있다.

In [10]:

```
from nltk.tag import pos_tag
sentence = "Emma refused to permit us to obtain the refuse permit"
tagged_list = pos_tag(word_tokenize(sentence))
tagged_list
```

Out[10]:

```
[('Emma', 'NNP'),
 ('refused', 'VBD'),
 ('to', 'TO'),
 ('permit', 'VB'),
 ('us', 'PRP'),
 ('to', 'TO'),
 ('obtain', 'VB'),
 ('the', 'DT'),
 ('refuse', 'NN'),
 ('permit', 'NN')]
```

품사 태그 정보를 사용하면 명사인 토큰만 선택할 수 있다.

In [11]:

```
nouns_list = [t[0] for t in tagged_list if t[1] == "NN"]
nouns_list
```

Out[11]:

```
['refuse', 'permit']
```

untag 명령을 사용하면 태그 튜플을 제거할 수 있다.

In [12]:

```
from nltk.tag import untag
untag(tagged_list)
```

Out[12]:

```
['Emma',
 'refused',
 'to',
 'permit',
 'us',
 'to',
 'obtain',
 'the',
 'refuse',
 'permit']
```

Scikit-Learn 등에서 자연어 분석을 할 때는 같은 토큰이라도 품사가 다르면 다른 토큰으로 처리해야 하는 경우가 많은데 이 때는 원래의 토큰과 품사를 붙여서 새로운 토큰 이름을 만들어 사용하면 철자가 같고 품사가 다른 단어를 구분할 수 있다.

In [13]:

```
def tokenizer(doc):  
    return ["/".join(p) for p in tagged_list]  
  
tokenizer(sentence)
```

Out[13]:

```
['Emma/NNP',  
'refused/VBD',  
'to/TO',  
'permit/VB',  
'us/PRP',  
'to/TO',  
'obtain/VB',  
'the/DT',  
'refuse/NN',  
'permit/NN']
```

## Text 클래스

NLTK의 `Text` 클래스는 문서 분석에 유용한 여러가지 메서드를 제공한다. 토큰열을 입력하여 생성한다.

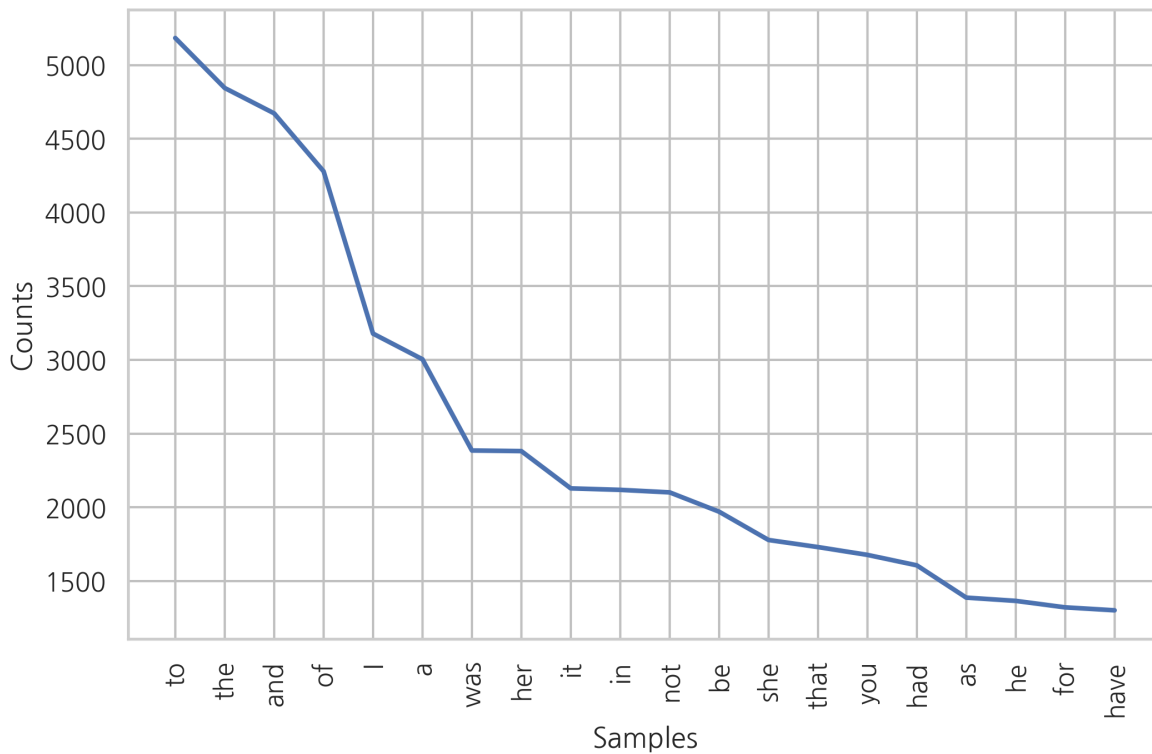
In [14]:

```
from nltk import Text  
  
text = Text(retokenize.tokenize(emma_raw))
```

`plot` 메소드를 사용하면 각 단어(토큰)의 사용 빈도를 그래프로 그려준다.

In [15]:

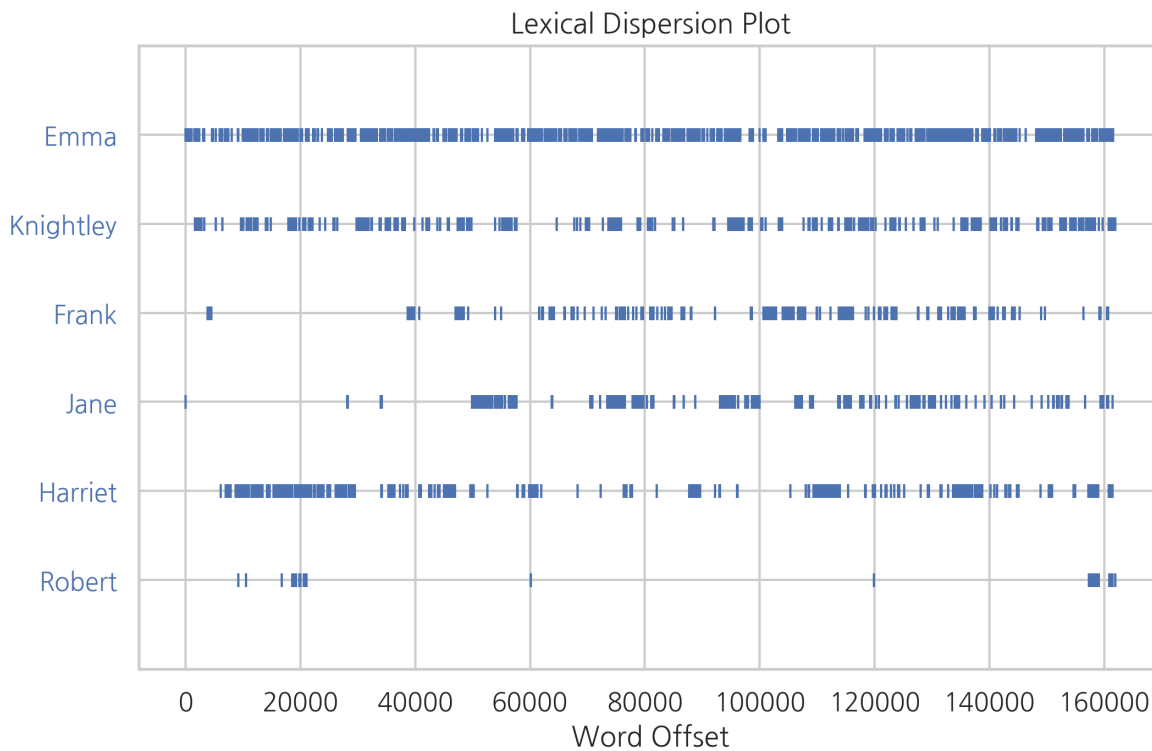
```
text.plot(20)
plt.show()
```



dispersion\_plot 메서드는 단어가 사용된 위치를 시각화한다. 소설 엠마의 각 등장인물에 대해 적용하면 다음과 같은 결과를 얻는다.

In [16]:

```
text.dispersion_plot(["Emma", "Knightley", "Frank", "Jane", "Harriet", "Robert"])
```





concordance 메서드로 단어가 사용된 위치를 직접 표시하면 문맥(context)이 어떤지 볼 수 있다. 여기에서 문맥은 해당 단어의 앞과 뒤에 사용된 단어를 뜻한다.

In [17]:

```
text.concordance("Emma")
```

Displaying 25 of 865 matches:

```
Emma by Jane Austen 1816 VOLUME I CHAPTER
Jane Austen 1816 VOLUME I CHAPTER I Emma Woodhouse handsome clever and rich w
f both daughters but particularly of Emma Between _them_ it was more the intim
nd friend very mutually attached and Emma doing just what she liked highly est
by her own The real evils indeed of Emma s situation were the power of having
ding day of this beloved friend that Emma first sat in mournful thought of any
going only half a mile from them but Emma was aware that great must be the dif
a day It was a melancholy change and Emma could not but sigh over it and wish
ll the rest of her life at Hartfield Emma smiled and chatted as cheerfully as
l be able to tell her how we all are Emma spared no exertions to maintain this
or l have a great regard for you and Emma but when it comes to the question of
a fanciful troublesome creature said Emma playfully That is what you have in y
e few people who could see faults in Emma Woodhouse and the only one who ever
is was not particularly agreeable to Emma herself she knew it would be so much
being thought perfect by every body Emma knows l never flatter her said Mr Kn
that she must be a gainer Well said Emma willing to let it pass you want to h
were sure of meeting every day Dear Emma bears every thing so well said her f
l_ miss her more than she thinks for Emma turned away her head divided between
ars and smiles It is impossible that Emma should not miss such a companion sai
rgotten one matter of joy to me said Emma and a very considerable one that l m
lf than good to them by interference Emma never thinks of herself if she can d
ch better thing Invite him to dinner Emma and help him to the best of the fish
and could not think without pain of Emma s losing a single pleasure or suffer
nt of her companionableness but dear Emma was of no feeble character she was m
ent was so just and so apparent that Emma well as she knew her father was some
```

similar 메서드는 같은 문맥에서 주어진 단어 대신 사용된 횟수가 높은 단어들을 찾는다.

In [18]:

```
text.similar("Emma")
```

```
she it he i harriet you her jane him that me and all they them herself
there but be isabella
```

두 단어의 공통 문맥을 보려면 common\_contexts 메서드를 사용한다.

In [19]:

```
text.common_contexts(["Emma", "she"])
```

```
but_was and_could that_should said_and which_could whom_knew
which_particularly and_imagined that_could said_i that_began
and_thought do_was but_could than_had said_but manner_was this_could
as_saw possible_could
```

## FreqDist

FreqDist 클래스는 문서에 사용된 단어(토큰)의 사용빈도 정보를 담은 클래스이다. Text 클래스의 vocab 메서드로 추출할 수 있다.

In [20]:

```
fd = text.vocab()
type(fd)
```

Out[20]:

```
nltk.probability.FreqDist
```

또는 다음처럼 토큰 리스트를 넣어서 직접 만들 수도 있다. 다음 코드에서는 Emma 말뭉치에서 사람의 이름만 모아서 FreqDist 클래스 객체를 만들었다. 품사 태그에서 NNP (고유대명사)이면서 필요없는 단어(stop words)는 제거하였다.

In [21]:

```
from nltk import FreqDist

stopwords = ["Mr.", "Mrs.", "Miss", "Mr", "Mrs", "Dear"]
emma_tokens = pos_tag(retokenize.tokenize(emma_raw))
names_list = [t[0] for t in emma_tokens if t[1] == "NNP" and t[0] not in stopwords]
fd_names = FreqDist(names_list)
```

FreqDist 클래스는 단어를 키(key), 출현빈도를 값(value)으로 가지는 사전 자료형과 유사하다. 다음 코드는 전체 단어의 수, "Emma"라는 단어의 출현 횟수, 확률을 각각 계산한다.

In [22]:

```
fd_names.N(), fd_names["Emma"], fd_names.freq("Emma")
```

Out[22]:

```
(7863, 830, 0.10555767518758744)
```

most\_common 메서드를 사용하면 가장 출현 횟수가 높은 단어를 찾는다.

In [23]:

```
fd_names.most_common(5)
```

Out[23]:

```
[('Emma', 830),
 ('Harriet', 491),
 ('Weston', 439),
 ('Knightley', 389),
 ('Elton', 385)]
```

## 워드클라우드

wordcloud 패키지를 사용하면 단어의 사용 빈도수에 따라 워드클라우드(Word Cloud) 시각화를 할 수 있다.

In [24]:

```
from wordcloud import WordCloud
```

```
wc = WordCloud(width=1000, height=600, background_color="white", random_state=0)
plt.imshow(wc.generate_from_frequencies(fd_names))
plt.axis("off")
plt.show()
```

