영어 텍스트 처리: nltk

tokenize

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

In [2]:
```
example_string = """
... Muad'Dib learned rapidly because his first training was in how to learn.
... And the first lesson of all was the basic trust that he could learn.
... It's shocking to find how many people do not believe they can learn,
... and how many more believe learning to be difficult."""
```

In [3]:
```
# 문장 단위 분할
sent_tokenize(example_string)
```

Out[3]:
```
["\nMuad'Dib learned rapidly because his first training was in how to learn.",
 'And the first lesson of all was the basic trust that he could learn.',
 "It's shocking to find how many people do not believe they can learn,\nand how many more believe learning to be difficult."]
```

In [5]:
```
# 단어 단위 분할
word_tokenize(example_string)[:5]
```

Out[5]:
```
["Muad'Dib", 'learned', 'rapidly', 'because', 'his']
```

filtering stopwords

In [7]:
```
import nltk
nltk.download("stopwords")
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\Gilseung\AppData\Roaming\nltk_data...
[nltk_data]   Unzipping corpora\stopwords.zip.
```

Out[7]:
```
True
```

In [14]:
```
from nltk.corpus import stopwords
stop_words = set(stopwords.words("english"))
list(stop_words)[:5]
```

```
Out[14]: ['i', 'haven', 'those', 'itself', 'you']
```

Stemming

```
In [15]: from nltk.stem import PorterStemmer
         from nltk.tokenize import word_tokenize
```

```
In [16]: stemmer = PorterStemmer()
```

```
In [17]: string_for_stemming = """
         ... The crew of the USS Discovery discovered many discoveries.
         ... Discovering is what explorers do."""
```

```
In [18]: words = word_tokenize(string_for_stemming)
```

```
In [19]: stemmed_words = [stemmer.stem(word) for word in words]
```

```
In [21]: stemmed_words[:5]
```

```
Out[21]: ['the', 'crew', 'of', 'the', 'uss']
```

POS

```
In [23]: nltk.pos_tag(stemmed_words)[:5]
```

```
Out[23]: [('the', 'DT'), ('crew', 'NN'), ('of', 'IN'), ('the', 'DT'), ('uss', 'JJ')]
```

```
In [27]: # 태그 목록 확인
         #nltk.help.upenn_tagset()
```

Lemmatization

```
In [35]:
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
string_for_lemmatizing = "The friends of DeSoto love scarves."
words = word_tokenize(string_for_lemmatizing)
lemmatized_words = [lemmatizer.lemmatize(word) for word in words]
lemmatized_words
```

```
Out[35]: ['The', 'friend', 'of', 'DeSoto', 'love', 'scarf', '.']
```

한국어 텍스트 처리: konlpy

데이터 준비

```
In [46]:
from konlpy.corpus import kolaw
c = kolaw.open('constitution.txt').read()
print(c[:40])
```

```
대한민국헌법

유구한 역사와 전통에 빛나는 우리 대한국민은 3·1운동으로
```

형태소 분석

```
In [34]:
from konlpy.tag import *

hannanum = Hannanum()
kkma = Kkma()
komoran = Komoran()
#mecab = Mecab()
#okt = Okt()
```

명사 추출

```
In [48]:
print(hannanum.nouns(c[:40]))
print(kkma.nouns(c[:40]))
print(komoran.nouns(c[:40]))
```

```
['대한민국헌법', '유구', '역사', '전통', '빛', '우리', '대한국민', '3·1운동']
['대한', '대한민국', '대한민국헌법', '민국', '헌법', '유구', '역사', '전통', '우리', '국민', '3', '1', '1운동', '운동']
['대한민국헌법', '역사', '전통', '한국민', '3·1운동']
```

형태소 추출

In [51]:
```
print(hannanum.morphs(c[:40]))
print(kkma.morphs(c[:40]))
```

```
['대한민국헌법', '유구', '하', 'ㄴ', '역사', '와', '전통', '에', '빛', '나는', '우리', '대한국민', '은', '3·1운동', '으로']
['대한민국', '헌법', '유구', '하', 'ㄴ', '역사', '와', '전통', '에', '빛나', '는', '우리', '대하', 'ㄴ', '국민', '은', '3', '·', '1', '운동', '으
로']
```

품사태깅

In [53]:
```
print(hannanum.pos(c[:40]))
print(kkma.pos(c[:40]))
```

```
[('대한민국헌법', 'N'), ('유구', 'N'), ('하', 'X'), ('ㄴ', 'E'), ('역사', 'N'), ('와', 'J'), ('전통', 'N'), ('에', 'J'), ('빛', 'N'), ('나
는', 'J'), ('우리', 'N'), ('대한국민', 'N'), ('은', 'J'), ('3·1운동', 'N'), ('으로', 'J')]
[('대한민국', 'NNG'), ('헌법', 'NNG'), ('유구', 'NNG'), ('하', 'XSV'), ('ㄴ', 'ETD'), ('역사', 'NNG'), ('와', 'JC'), ('전통', 'NNG'),
('에', 'JKM'), ('빛나', 'VV'), ('는', 'ETD'), ('우리', 'NNM'), ('대하', 'VV'), ('ㄴ', 'ETD'), ('국민', 'NNG'), ('은', 'JX'), ('3', 'N
R'), ('·', 'SP'), ('1', 'NR'), ('운동', 'NNG'), ('으로', 'JKM')]
```

특수문자 제거

In [57]:
```
import re
def clean_text(text):
    """ 한글, 영문, 숫자만 남기고 제거한다. :param text: :return: """
    text = text.replace(".", " ").strip()
    text = text.replace("·", " ").strip()
    pattern = '[^ ㄱ-ㅣ가-힣|0-9|a-zA-Z]+'
    text = re.sub(pattern=pattern, repl='', string=text)
    return text
```

In [58]:
```
clean_text("하하하 @@@@ ㅋㅋㅋ !!")
```

Out[58]:
```
'하하하  ㅋㅋㅋ '
```

Term - document matrix

CountVectorizer

class sklearn.feature_extraction.text.CountVectorizer(*, input='content', encoding='utf-8', decode_error='strict', strip_accents=None, lowercase=True, preprocessor=None, tokenizer=None, stop_words=None, token_pattern='(?u)\b\w\w+\b', ngram_range=(1, 1), analyzer='word', max_df=1.0, min_df=1, max_features=None, vocabulary=None, binary=False, dtype=<class 'numpy.int64'>)

Parameters

- decode_error: {'strict', 'ignore', 'replace'}, default='strict'
- ngram_range: tuple (min_n, max_n), default=(1, 1)
- binary: bool, default=False (count vs occurence)

In [30]:
```
from sklearn.feature_extraction.text import CountVectorizer
corpus = ['This is the first document.',
          'This document is the second document.',
          'And this is the third one.',
          'Is this the first document?']

vectorizer = CountVectorizer()
X = vectorizer.fit_transform(corpus)
print(vectorizer.get_feature_names())
print(X.toarray()) # ndarray
```

```
['and', 'document', 'first', 'is', 'one', 'second', 'the', 'third', 'this']
[[0 1 1 1 0 0 1 0 1]
 [0 2 0 1 0 1 1 0 1]
 [1 0 0 1 1 0 1 1 1]
 [0 1 1 1 0 0 1 0 1]]
<class 'numpy.ndarray'>
```

tf-idf

In [31]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
corpus = ['This is the first document.',
         'This document is the second document.',
         'And this is the third one.',
         'Is this the first document?']
vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(corpus)
print(vectorizer.get_feature_names())
print(X.toarray())
```

```
['and', 'document', 'first', 'is', 'one', 'second', 'the', 'third', 'this']
[[0.         0.46979139 0.58028582 0.38408524 0.         0.
```

```
  0.38408524 0.         0.38408524]
 [0.         0.6876236  0.         0.28108867 0.         0.53864762
  0.28108867 0.         0.28108867]
 [0.51184851 0.         0.         0.26710379 0.51184851 0.
  0.26710379 0.51184851 0.26710379]
 [0.         0.46979139 0.58028582 0.38408524 0.         0.
  0.38408524 0.         0.38408524]]
```

토픽모델링

데이터 불러오기

In [2]:

```python
import pandas as pd
import gensim
from sklearn.feature_extraction.text import CountVectorizer

documents = pd.read_csv('C:/Users/Gilseung/Downloads/news-data.csv/news-data.csv',
                        error_bad_lines=False, nrows = 1000)

documents.head()
```

Out[2]:

	publish_date	headline_text
0	20030219	aba decides against community broadcasting lic...
1	20030219	act fire witnesses must be aware of defamation
2	20030219	a g calls for infrastructure protection summit
3	20030219	air nz staff in aust strike for pay rise
4	20030219	air nz strike to affect australian travellers

데이터 정리

In [8]:

```python
# Use CountVectorizer to find three letter tokens, remove stop_words,
# remove tokens that don't appear in at least 20 documents,
# remove tokens that appear in more than 20% of the documents
vect = CountVectorizer(min_df=20, max_df=0.2, stop_words='english',
                       token_pattern='(?u)\\b\\w\\w\\w+\\b')

# Fit and transform
X = vect.fit_transform(documents.headline_text)
```

```
# Convert sparse matrix to gensim corpus.
corpus = gensim.matutils.Sparse2Corpus(X, documents_columns=False)
```

LDA model 생성

In [12]:

```
# Mapping from word IDs to words (To be used in LdaModel's id2word parameter)
id_map = dict((v, k) for k, v in vect.vocabulary_.items())

# Use the gensim.models.ldamodel.LdaModel constructor to estimate
# LDA model parameters on the corpus, and save to the variable `ldamodel`
ldamodel = gensim.models.LdaMulticore(corpus=corpus, id2word=id_map, passes=2, num_topics=5, workers=2)
```

토픽별 단어 구성 확인

In [13]:

```
for idx, topic in ldamodel.print_topics(-1):
    print("Topic: {} \nWords: {}".format(idx, topic))
    print("\n")
```

```
Topic: 0
Words: 0.431*"war" + 0.405*"nsw" + 0.125*"iraq" + 0.006*"man" + 0.006*"police" + 0.005*"rain" + 0.005*"new" + 0.005*"govt" + 0.00
5*"council" + 0.005*"court"


Topic: 1
Words: 0.417*"police" + 0.294*"iraq" + 0.245*"court" + 0.011*"rain" + 0.011*"govt" + 0.005*"man" + 0.004*"nsw" + 0.004*"new" + 0.
004*"council" + 0.004*"war"


Topic: 2
Words: 0.643*"rain" + 0.156*"govt" + 0.121*"man" + 0.032*"nsw" + 0.016*"iraq" + 0.007*"police" + 0.007*"new" + 0.006*"court" + 0.
006*"war" + 0.006*"council"


Topic: 3
Words: 0.528*"man" + 0.215*"govt" + 0.140*"court" + 0.078*"new" + 0.016*"police" + 0.009*"nsw" + 0.004*"iraq" + 0.004*"council" +
0.004*"rain" + 0.003*"war"


Topic: 4
Words: 0.469*"council" + 0.344*"new" + 0.064*"police" + 0.060*"iraq" + 0.021*"nsw" + 0.021*"rain" + 0.006*"man" + 0.005*"govt" +
0.005*"court" + 0.005*"war"
```

토픽 분포 확인

In [16]:
```python
def topic_distribution(string_input):
    string_input = [string_input]
    # Fit and transform
    X = vect.transform(string_input)

    # Convert sparse matrix to gensim corpus.
    corpus = gensim.matutils.Sparse2Corpus(X, documents_columns=False)

    output = list(ldamodel[corpus])[0]
    return output

# 토픽의 비율: 0번 토픽 - 0.2, 1번 토픽 - 0.2, ...
topic_distribution(documents['headline_text'].iloc[0])
```

Out[16]:
```
[(0, 0.2), (1, 0.2), (2, 0.2), (3, 0.2), (4, 0.2)]
```