딥러닝을 활용한 자연어 분석

문장분석

김용범

무영인터내쇼날

Jupyter Notebook 소스 살펴보기

http://nbviewer.jupyter.org/github/YongBeomKim/nltk_tutorial/blob/master/02.sentence.ipynb

자연어 분석과정

1. **형태론(Morphology) :** 단어와 형태소를 연구
2. **통사론(syntax) :** 문법적 구조분석(Parsing)
3. **의미론(Senmantics) :** 단어 의미차이 **ex) 뉘앙스, 톤, 의도(긍/부정)**

Phrase
토큰 활용

불용어 처리

단어의 활용

Stop Words

1. **연관성이 낮은 단어**들을 제외하고 분석
2. **내용**과 **목적**에 따라서, **불용어 처리여부** 및 해당 **목적에 맞는 불용어 말뭉치 DB**등을 판단해야 한다

Stop Words

In [10]:
```
texts = 'I like such a Wonderful Snow Ice Cream'
texts = texts.lower()
texts
```

Out[10]:
```
'i like such a wonderful snow ice cream'
```

In [11]:
```
from nltk import word_tokenize
tokens = word_tokenize(texts)
tokens
```

Out[11]:
```
['i', 'like', 'such', 'a', 'wonderful', 'snow', 'ice', 'cream']
```

Stop Words

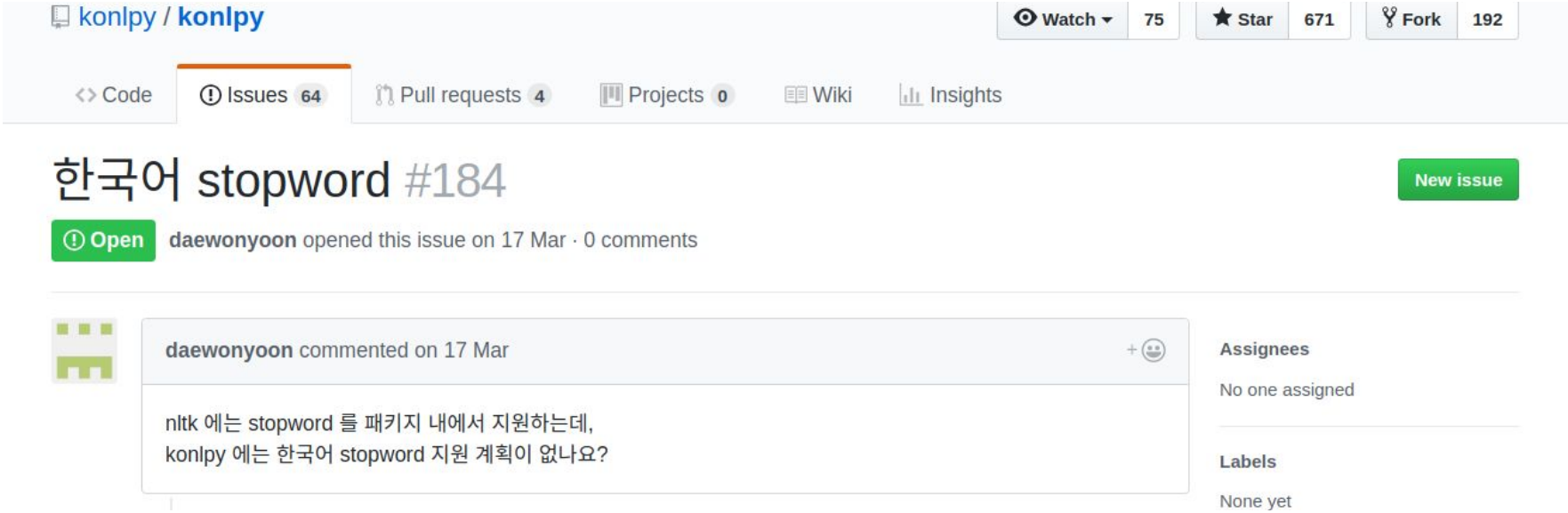
In [12]:
```
from nltk.corpus import stopwords
stopwords.words('english')[::18]
```

Out[12]:
```
['i', 'her', 'those', 'an', 'into', 'further', 'such', 'now', 'mightn']
```

In [13]:
```
tokens = [word   for word in tokens
                 if word not in stopwords.words('english')]
print(tokens)
```

```
['like', 'wonderful', 'snow', 'ice', 'cream']
```

한글의 경우


konlpy / konlpy
Watch 75
Star 671
Fork 192
Code
Issues 64
Pull requests 4
Projects 0
Wiki
Insights
한국어 stopword #184
New issue
Open
daewonyoon opened this issue on 17 Mar · 0 comments
daewonyoon commented on 17 Mar
nltk 에는 stopword 를 패키지 내에서 지원하는데,
konlpy 에는 한국어 stopword 지원 계획이 없나요?
Assignees
No one assigned
Labels
None yet

한글의 경우

In [14]:
```
f = open('./data/한국어불용어100.txt', 'r')
s = f.read(); f.close()

stop_words = [ txt.split('\t')[:2]  for txt in s.split('\n') ]
stopword   = {}
for txt in stop_words:
    try:    stopword[txt[0]] = txt[1]
    except: pass
stopword
```

Out[14]:
```
{'이': 'NP',
 '있': 'VA',
 '하': 'VV',
 '것': 'NNB',
 '들': 'VV',
 '그': 'MM',
 '되': 'VV',
 '수': 'NNB',
 '보': 'VX',
 '않': 'VX',
 '없': 'VA',
```

Phrase
통계이론

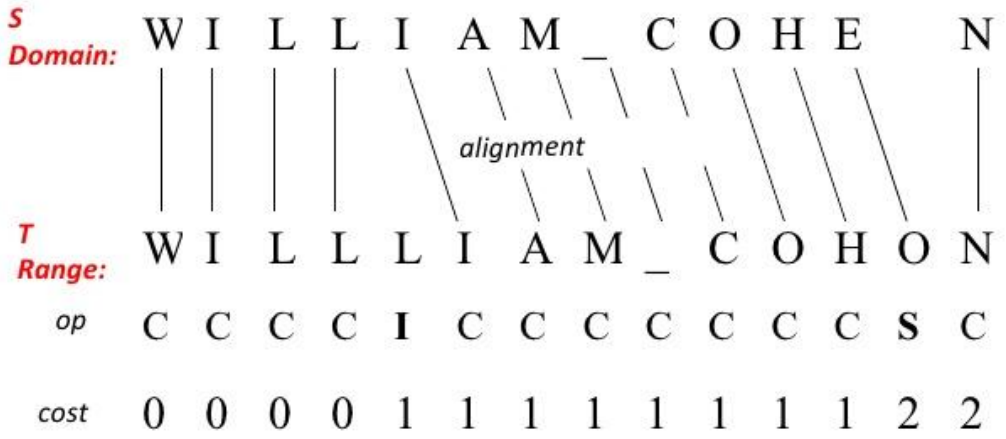
통계 이론

[토큰 List] 활용

레벤슈타인 편집거리

Levenshtein distance - example

- distance("William Cohen", "Willliam Cohon")



레벤슈타인 편집거리

1. **비교 대상간**의 **Token 의 갯수**가 **일치**
2. **두 단어/ 문장**이 **같은 내용**이 되려면 몇 번의 수정을 필요로 하는지 계산
3. **최소 작업순번**의 측정 값으로 **문장의 유사도**를 판단

레벤슈타인 편집거리

In [15]:
```
text1 = "자연 언어에 대한 연구는 오래전부터 이어져 오고 있음에도 2018년까지도 사람처럼 이해하지는 못한다.
text2 = "자연 언어에 대한 연구는 오래전부터 이어져 들어서도 아직 컴퓨터가 사람처럼 이해하지는 못한다.".sp
text3 = "자연 아직 컴퓨터가 언어에 들어서도 못한다 이어져 사람처럼 이해하지는 대한 연구는 오래전부터.".sp
len(text1), len(text2), len(text3)
```

Out[15]:
```
(12, 12, 12)
```

In [16]:
```
from nltk.metrics import edit_distance
edit_distance('파이썬 알고리즘', '파파미 알탕')
```

Out[16]:
```
5
```

In [17]:
```
print('생략된 단어가 다를 때 : {} \n어휘 순서를 바꿨을 때 : {}'.format(
    edit_distance(text1, text2),
    edit_distance(text2, text3)))
```

```
생략된 단어가 다를 때 : 3
어휘 순서를 바꿨을 때 : 10
```

문장간의 Accuracy - 동일한 token 의 [list] 객체를 필요

In [17]:
```
# 02 accuracy 정확도 측정
from nltk.metrics import accuracy
accuracy('파이썬', '파이프')
```

Out[17]: 0.6666666666666666

In [18]:
```
print('생략된 단어가 다를 때 {:.4} \n어휘 순서를 바꿨을 때 {:.4}'.format(
    accuracy(text1, text2),
    accuracy(text2, text3)))
```

생략된 단어가 다를 때 0.75
어휘 순서를 바꿨을 때 0.08333

통계 이론

Set(토큰) 활용

Token { set } 객체를 활용 - 혼동 Matrix

1. **precision =** Correct / (Correct + Incorrect + Missing)
2. **recall =** Correct / (Correct + Incorrect + Spurious<가짜>)
3. **f_measure =** (2 X Precision X Recall) / (Precision + Recall)

		Predicted Data		Total
		Predicted Condition POSITIVE	Predicted Condition NEGATIVE	
Acutal Data	Condition TRUE	TP True Positive	FN False Negative	P
	Condition FALSE	FP False Positive	TN True Negative	N
Total		P`	N`	P+N

혼동 Matrix - precision (정확도)

In [20]:
```
text1 = set(text1)
text2 = set(text2)
text3 = set(text3)
len(text1), len(text2), len(text3)
```

Out[20]:
```
(12, 12, 12)
```

In [21]:
```
from nltk.metrics import precision
precision({'파이썬'}, {'파르썬'})
```

Out[21]:
```
0.0
```

In [22]:
```
print('생략된 단어가 다를 때 {:.4} \n어휘 순서를 바꿨을 때 {:.4}'.format(
    precision(set(text1), set(text2)),
    precision(set(text2), set(text3))))
```

```
생략된 단어가 다를 때 0.75
어휘 순서를 바꿨을 때 0.8333
```

혼동 **Matrix** - recall/ f-measure

In [23]:
```
# recall 어휘의 재현율

from nltk.metrics import recall
print('생략된 단어가 다를 때 {:.4} \n어휘 순서를 바꿨을 때 {:.4}'.format(
    recall(text1, text2),
    recall(text2, text3)))
```

```
생략된 단어가 다를 때 0.75
어휘 순서를 바꿨을 때 0.8333
```

In [24]:
```
# f-measure 측정

from nltk.metrics import f_measure
print('생략된 단어가 다를 때 {:.4} \n어휘 순서를 바꿨을 때 {:.4}'.format(
    f_measure(text1, text2),
    f_measure(text2, text3)))
```

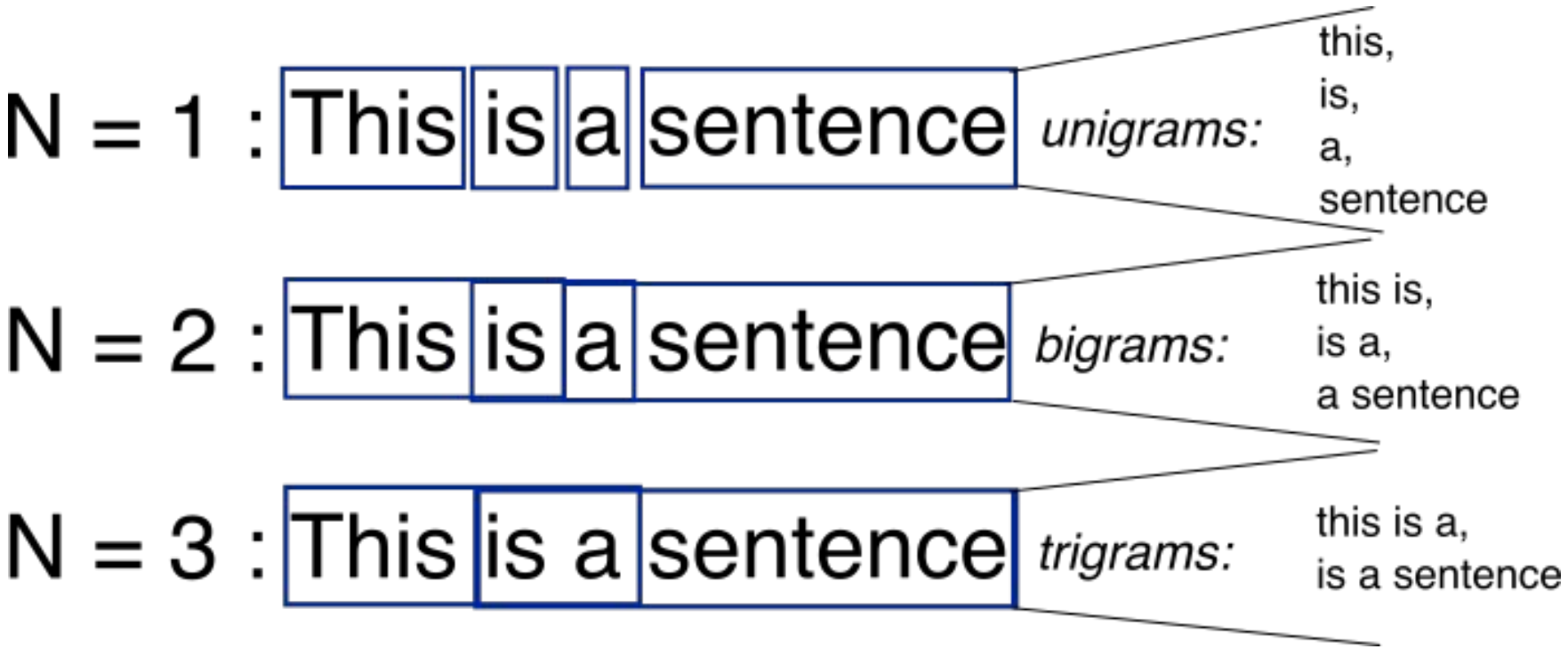
```
생략된 단어가 다를 때 0.75
어휘 순서를 바꿨을 때 0.8333
```

Phrase N-Gram

N-Gram

Token 집합

N Gram


N = 1 : This is a sentence
unigrams:
this,
is,
a,
sentence
N = 2 : This is a sentence
bigrams:
this is,
is a,
a sentence
N = 3 : This is a sentence
trigrams:
this is a,
is a sentence

N Gram

1. 통계적 이론, 편집 알고리즘은 **token 갯수 일치**를 필요
2. 기준보다 갯수가 적으면 **Zero Padding** 활용
3. 문장 **token 최대 기준 갯수**가 변경되면 기존의 모델을 활용하기 어려운 단점이
4. **독립적 관리** 가능한, **분류기준**이 필요

N Gram

In [25]:
```
texts_sample = texts_Berlin[:20]
texts_sample
```

Out[25]:
```
['존경하는',
 '여러분',
 '고국에',
 '여러분',
 '하울젠',
 '쾨르버재단',
 '이사님과',
 '모드로',
 '총리님을',
 '비롯한',
 '여러분',
```

N Gram

In [26]:
```
from nltk.util import ngrams

texts_sample = [txt for txt in ngrams(texts_sample, 3)]
texts_sample[:5]
```

Out[26]:
```
[('존경하는', '여러분', '고국에'),
 ('여러분', '고국에', '여러분'),
 ('고국에', '여러분', '하울젠'),
 ('여러분', '하울젠', '쾨르버재단'),
 ('하울젠', '쾨르버재단', '이사님과')]
```

PMI

N-gram 상관분석

Point wise Mutual Information

점 위치 상호관계를 활용한 정보

1. **단어간의 거리**를 비교측정하여 **객체의 상관성**을 분석한다
2. 연어 (**근접어:collocation**) 관계 활용하여 분석가능한 객체를 생성한다 (**Bi-gram, Tri-gram**)
3. PMI 는 **단어간의 상관관계 확률론**을 근거로, **단어간의 독립을 가정할 때 발생확률** 과 **문서에서 측정된 동시발생확률** 을 비교하여 상관성을 분석한다

PMI - 한글만 추출

In [27]:
```
from nltk.tokenize import RegexpTokenizer
re_capt = RegexpTokenizer('[가-힣]\w+')
raw_texts = re_capt.tokenize(texts_Berlin_raw)
raw_texts[:10]
```

Out[27]:
```
['존경하는', '독일', '국민', '여러분', '고국에', '계신', '국민', '여러분', '하울젠', '쾨르버재단']
```

In [28]:
```
texts = ''
for txt in raw_texts:
    texts += txt + " "
texts[:200]
```

Out[28]:
```
'존경하는 독일 국민 여러분 고국에 계신 국민 여러분 하울젠 쾨르버재단 이사님과 모드로 동독 총리님을 비롯한 내외 귀빈 여러분 먼저 냉전과 분단을 넘어 통일을 이루고 힘으로 유럽통합과 국제평화를 선도하고 있는 독일과 독일 국민에게 무한한 경의를 표합니다 오늘 자리를 마련해 주신 독일 정부와 쾨르버 재단에도 감사드립니다 아울러 얼마 별세하신 헬무트 총리의 가족'
```

PMI - 한글에 태그를 추가

In [29]:
```
%%time
# 베를린 선언문에 Tag 속성 추가하기
from konlpy.tag import Twitter
twitter = Twitter()
tagged_words = twitter.pos(texts)
```

```
CPU times: user 1.78 s, sys: 51.4 ms, total: 1.83 s
Wall time: 565 ms
```

PMI - 연어관계 객체를 생성한 뒤, 객체 상위 PMI Bi-gram 출력

In [30]:
```
from nltk import collocations

finder = collocations.BigramCollocationFinder.from_words(tagged_words)
finder
```

Out[30]:
```
<nltk.collocations.BigramCollocationFinder at 0x7fc643f2ddd8>
```

In [31]:
```
# top 10 n-grams with highest PMI
measures = collocations.BigramAssocMeasures()
finder.nbest(measures.pmi, 10)
```

Out[31]:
```
[(('가능하며', 'Adjective'), ('불가', 'Noun')),
 (('가스', 'Noun'), ('관', 'Noun')),
 (('가운데', 'Noun'), ('현재', 'Noun')),
 (('감사', 'Noun'), ('드립니', 'Verb')),
 (('갖춰', 'Verb'), ('지', 'PreEomi')),
 (('같은', 'Adjective'), ('공감', 'Noun')),
 (('거나', 'Eomi'), ('깨져', 'Verb')),
 (('건너지', 'Verb'), ('않기', 'Verb')),
 (('걷어', 'Verb'), ('차는', 'Verb')),
 (('검증', 'Noun'), ('가능하며', 'Adjective'))]
```

PMI - 연어관계 객체를 생성한 뒤, 객체 상위 PMI Tri-gram 출력

In [32]:
```
finder = collocations.TrigramCollocationFinder.from_words(tagged_words)
finder
```

Out[32]:
```
<nltk.collocations.TrigramCollocationFinder at 0x7fc643f2db70>
```

In [33]:
```
measures = collocations.TrigramAssocMeasures()
finder.nbest(measures.pmi, 10)
```

Out[33]:
```
[(('가능하며', 'Adjective'), ('불가', 'Noun'), ('역적', 'Noun')),
 (('가스', 'Noun'), ('관', 'Noun'), ('연결', 'Noun')),
 (('가운데', 'Noun'), ('현재', 'Noun'), ('생존', 'Noun')),
 (('같은', 'Adjective'), ('공감', 'Noun'), ('대', 'Suffix')),
 (('거나', 'Eomi'), ('깨져', 'Verb'), ('서도', 'Noun')),
 (('검증', 'Noun'), ('가능하며', 'Adjective'), ('불가', 'Noun')),
 (('견', 'Noun'), ('지하', 'Noun'), ('면서', 'Noun')),
 (('과도', 'Josa'), ('같은', 'Adjective'), ('공감', 'Noun')),
 (('들어서는', 'Verb'), ('대전', 'Noun'), ('환', 'Noun')),
 (('록', 'Eomi'), ('앞장서서', 'Verb'), ('돕겠', 'Verb'))]
```

Tf-idf

상대빈도분석

Term Frequency-Inverse Document Frequency

1. 문서의 내용을 쉽게 벡터로 표현하는 고전적 방식
2. **Term Frequency** : 특정 용어의 발생빈도
3. (문서 Token 출현빈도) / (문서 전체 Token 갯수)
4. **Inverse Document Frequency** (문서 빈도의 역)
5. 문서를 이해하는데 Token의 중요도
6. **일반 문서대비 출현빈도**

Term Frequency-Inverse Document Frequency

1. **TF**는 해당 문서만 있으면 바로 연산이 가능하지만
2. **IDF**는 모집단의 Corpus 별 통계값 (일반문서의 Token 출현빈도)이 있어야 연산가능

Term Frequency-Inverse Document Frequency

	name character	token_stem character	count_per_doc integer	count_of_docs integer	tfidf numeric
111	Akebono Brake Corporation	act_euro	1	1	0.0809
112	Akebono Brake Corporation	act_euro_flyer	1	1	0.0809
113	Akebono Brake Corporation	aftermarket	1	2	0.0687
114	Akebono Brake Corporation	aftermarket_brake	1	1	0.0809
115	Akebono Brake Corporation	aftermarket_brake_usa	1	1	0.0809
116	Akebono Brake Corporation	akebono	3	1	0.2426
117	Akebono Brake Corporation	akebono_brake	2	1	0.1617
118	Akebono Brake Corporation	akebono_brake_trust	1	1	0.0809
119	Akebono Brake Corporation	akebono_proud	1	1	0.0809
120	Akebono Brake Corporation	akebono_proud_manufactur	1	1	0.0809

TF - IDF (트럼프 취임연설문 불러오기 / 전처리)

In [43]:
```
f = open('./data/trump.txt', 'r')
texts_org = f.read()
f.close()

from nltk import word_tokenize
texts = word_tokenize(texts_org)
texts[:5]
```

Out[43]:
```
['Chief', 'Justice', 'Roberts', ',', 'President']
```

In [44]:
```
from nltk.corpus import stopwords
stopword_eng = stopwords.words('english')

import string
punct = string.punctuation
punct = [punct[i] for i in range(len(punct))]
punct = punct + stopword_eng + ['\n']
len(punct)
```

Out[44]:
```
186
```

TF - IDF (Tf-idf 학습객체 생성하기)

In [45]:
```
texts = [txt.lower()    for txt in texts   if txt.lower() not in punct]
document = ''
for txt in texts:
    document += txt + ' '
```

In [46]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf_vec   = TfidfVectorizer()
index_value = {i[1]:i[0] for i in tfidf_vec.vocabulary_.items()}
transformed = tfidf_vec.fit_transform(raw_documents = [document])
transformed
```

Out[46]:
```
<1x456 sparse matrix of type '<class 'numpy.float64'>'
        with 456 stored elements in Compressed Sparse Row format>
```

TF - IDF (Numpy 결과를 Pandas Series 객체로 변환/ 조회하기)

In [47]:
```
fully_indexed = []

import numpy as np
transformed = np.array(transformed.todense())

for row in transformed:
    fully_indexed.append({index_value[column]:value for (column,value) in enumerate(row)})
```

In [48]:
```
import pandas as pd
tfidf = pd.Series(fully_indexed[0]).sort_values(ascending=False)
tfidf[:15]
```

Out[48]:
```
america     0.423619
american    0.232990
people      0.211809
country     0.190628
nation      0.190628
one         0.169447
every       0.148266
never       0.127086
world       0.127086
back        0.127086
```

TF - IDF (Pandas Series 객체 활용)

```
In [49]: tfidf['great']

Out[49]: 0.12708556268223822
```