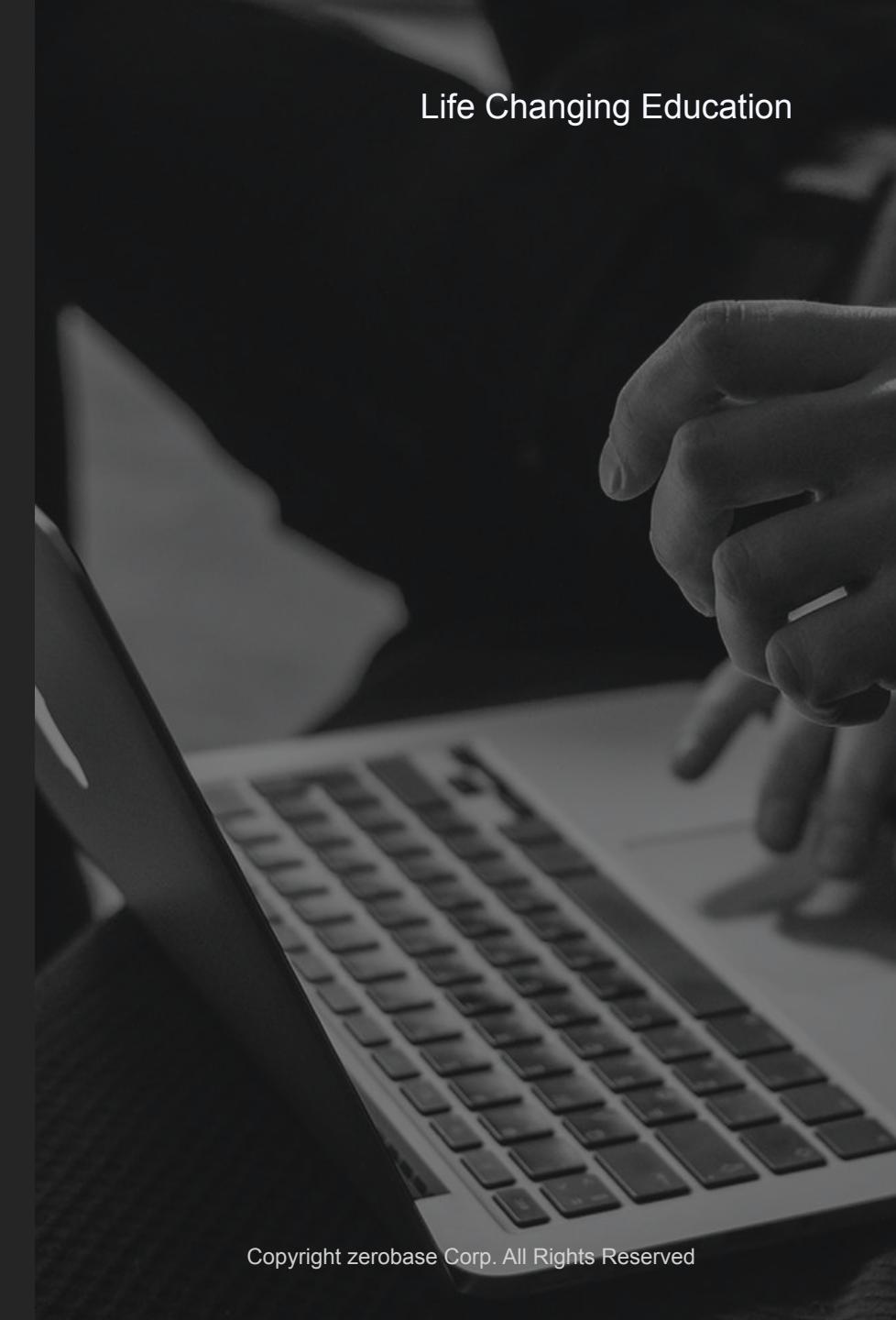
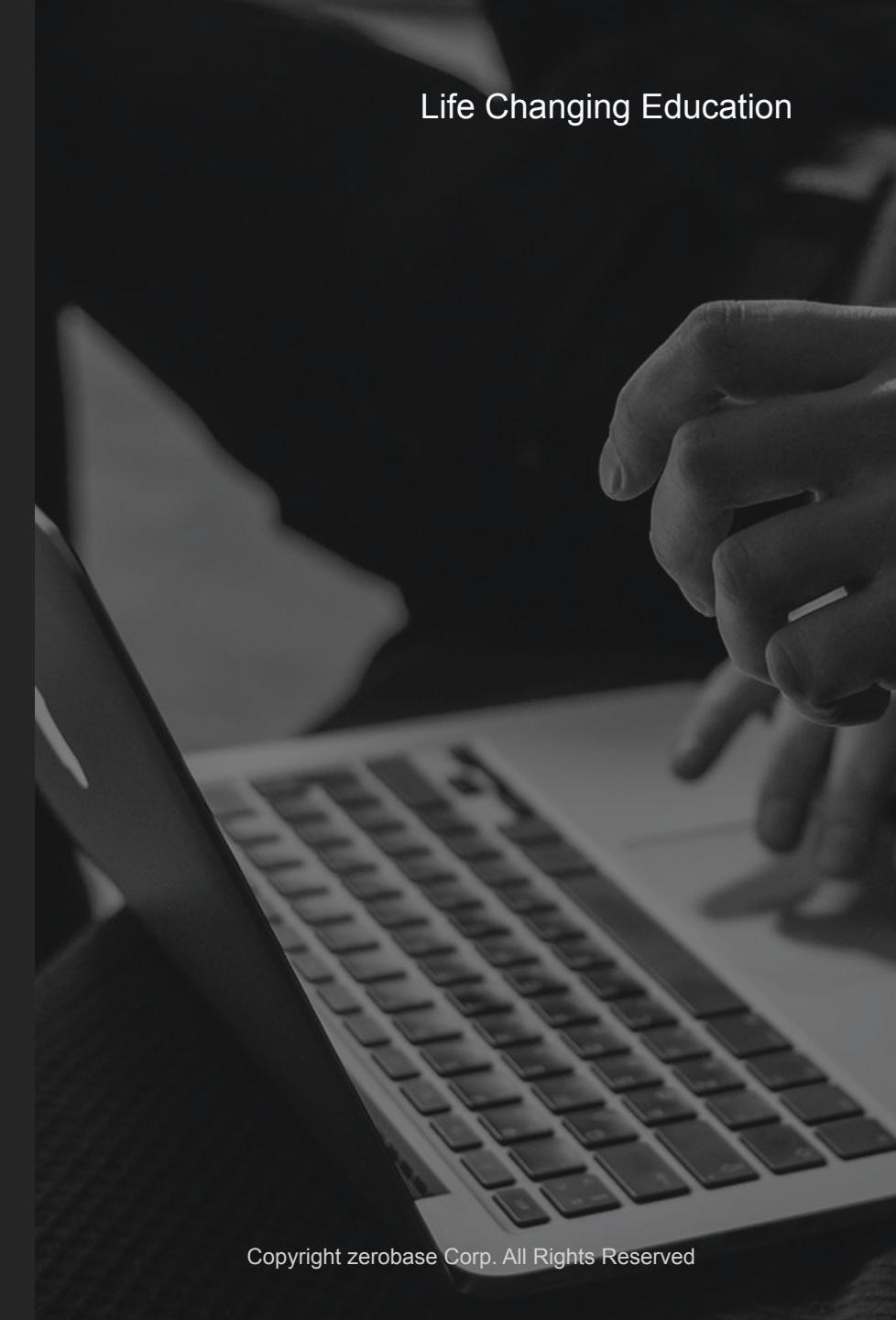


Chapter 06. Natural Language Processing



형태소 분석



```
In [3]: from konlpy.tag import Kkma  
kkma = Kkma()
```

```
In [4]: kkma.sentences("한국어 분석을 시작합니다 재미있어요~~")
```

```
Out[4]: ['한국어 분석을 시작합니다', '재미있어요~~']
```

```
In [5]: kkma.nouns("한국어 분석을 시작합니다 재미있어요~~")
```

```
Out[5]: ['한국어', '분석']
```

```
In [6]: kkma.pos("한국어 분석을 시작합니다 재미있어요~~")
```

```
Out[6]: [('한국어', 'NNG'),  
         ('분석', 'NNG'),  
         ('을', 'JKO'),  
         ('시작하', 'VV'),  
         ('ㅂ니다', 'EFN'),  
         ('재미있', 'VA'),  
         ('어요', 'EFN'),  
         ('~~', 'SW')]
```

```
In [7]: from konlpy.tag import Hannanum  
  
hannanum = Hannanum()
```

```
In [8]: hannanum.nouns("한국어 분석을 시작합니다 재미있어요~~")  
  
Out[8]: ['한국어', '분석', '시작']
```

```
In [9]: hannanum.morphs("한국어 분석을 시작합니다 재미있어요~~")  
  
Out[9]: ['한국어', '분석', '을', '시작', '하', 'ㅂ니다', '재미있', '어요', '~~']
```

```
In [10]: hannanum.pos("한국어 분석을 시작합니다 재미있어요~~")  
  
Out[10]: [('한국어', 'N'),  
          ('분석', 'N'),  
          ('을', 'J'),  
          ('시작', 'N'),  
          ('하', 'X'),  
          ('ㅂ니다', 'E'),  
          ('재미있', 'P'),  
          ('어요', 'E'),  
          ('~~', 'S')]
```

```
In [11]: from konlpy.tag import Twitter  
  
t = Twitter()
```

```
In [12]: t.nouns("한국어 분석을 시작합니다 재미있어요~~")
```

```
Out[12]: ['한국어', '분석', '시작']
```

```
In [13]: t.morphs("한국어 분석을 시작합니다 재미있어요~~")
```

```
Out[13]: ['한국어', '분석', '을', '시작', '합니다', '재미있어요', '~~']
```

```
In [14]: t.pos("한국어 분석을 시작합니다 재미있어요~~")
```

```
Out[14]: [('한국어', 'Noun'),  
          ('분석', 'Noun'),  
          ('을', 'Josa'),  
          ('시작', 'Noun'),  
          ('합니다', 'Verb'),  
          ('재미있어요', 'Adjective'),  
          ('~~', 'Punctuation')]
```

제26회 한글 및 한국어 정보처리 학술대회 논문집 (2014년)

KoNLPy: 쉽고 간결한 한국어 정보처리 파이썬 패키지

박은정⁰, 조성준

서울대학교 산업공학부

ejpark04@snu.ac.kr, zoon@snu.ac.kr

KoNLPy: Korean natural language processing in Python

Eunjeong L. Park⁰, Sungzoon Cho

Seoul National University, Industrial Engineering Department

요약

파이썬은 간결한 아름다움을 추구하는 동시에 강력한 스트링 연산이 가능한 언어다. KoNLPy는 그러한 특장점을 살려, 파이썬으로 한국어 정보처리를 할 수 있게 하는 패키지이다. 꼬꼬마, 한나눔, MeCab-ko 등 국내외에서 개발된 여러 형태소 분석기를 포함하고, 자연어처리에 필요한 각종 사전, 말뭉치, 도구 및 다양한 튜토리얼을 포함하여 누구나 손쉽게 한국어 분석을 할 수 있도록 만들었다.

주제어: 말뭉치 언어학, 말뭉치 활용 도구, 파이썬

출처: <http://hclt.kr/symp/?intpg=2&lnb=conference>

[표 1] 여러 오픈소스 한국어 형태소 분석기와 개발 언어.

이름	연도	언어	라이센스
KTS [14]	1995	C/C++	GPL v2
한나눔 [13]	1999	Java	GPL v3
MACH [11]	2002	C/C++	custom
Arirang	2009	Java	Apache v2
꼬꼬마 [7]	2010	Java	GPL v2
KoNLP [5]	2011	R	GPL v3
MeCab-ko [6]	2013	C/C++	GPL v2, LGPL, BSD
KOMORAN	2013	Java	custom

KoNLPy는, 이와 같은 파이썬의 이점 위에 다음의 설계 철학을 바탕으로 개발되었다.

쉽고 간단한 사용법. 직관적인 함수명을 사용한다. 매뉴얼을 따로 읽는데 많은 시간을 보내지 않아도 설치를 받자마자 바로 실행해서 결과를 볼 수 있게 한다.

확장가능성. 형태소 분석기 뿐 아니라 다양한 자연어 처리 기능과 말뭉치를 포괄하는 것을 목표로 하며, 따라서 말뭉치, 메소드 등이 추가되는 것을 감안하여 개발을 진행한다. NLTK가 다양한 스테머(stemmer)를 지원하는 것과 마찬가지로, 여러 형태소 분석기 중에서 목적과 취향에 맞는 것을 쉽게 선택할 수 있도록 한다.

친절하고 상세한 문서. 패키지에서 중요도가 다소 저 평가될 수 있는 부분이 문서화인데, 사실 상세한 문서와 풍부한 예제는 초심자에게 가장 큰 도움이 되는 부분이기도 하다. 특히 NLTK, Gensim[10] 등 다른 파이썬 패키지들과 함께 사용할 때, 어떻게 응용하여 사용할 수 있는지 보여주는 것도 중요하다.

개방과 공유. 패키지는 사용 환경이 계속 변할 수 있기 때문에 지속가능성(sustainability)을 유지할 수 있는지가 중요하다. 또한, 실질적인 사용자의 니즈(needs)를 파악하기 위해서는 누구나 개발에 참여를 할 수 있게 하는 것도 중요하다. 따라서 본 패키지의 소스는 온라인에 공개하여 참여를 독려하고 있다.³⁾

개요

NLTK 덕에 파이썬으로 자연어처리를 하는 것이 편리해졌다.
단, 한국어만 분석하려하지 않는다면.

파이썬으로 한국어를 분석할 수는 없을까?
국문, 영문, 중문 등 다양한 문자가 섞여 있는 문서는 어떻게 분석할 수 있을까?

이 발표에서는 자연어처리의 기초적인 개념을 다룬 후, NLTK 등의 자연어처리 라이브러리와 한국어 분석을 위해 개발중인 KoNLPy를 소개한다.
또, 파이썬으로 한국어를 분석할 때 유용한 몇 가지 트릭을 공유한다.

- KoNLPy docs: <http://konlpy.readthedocs.org>
- Slides URL: <http://www.lucypark.kr/slides/2014-pyconkr>
- Slides code: <https://gist.github.com/e9t/546faa368424e04e25c7>

Lucy Park의 자료에서 발췌



박은정

(a.k.a. lucypark,
echojuliett, e9t)

개발하는 데이터 분석가.

- 서울대학교 데이터마이닝 센터 박사과정
- "대한민국 정치의 모든 것" 만드는 팀포풀 멤버
- Just another yak shaver...

11:49 <@sanxiyn> 또다시 yak shaving의 신비한 세계
11:51 <@sanxiyn> yak shaving이 뭔지 다 아시죠?

Lucy Park의 자료에서 발췌

PyCon(파이콘)은 세계 각국의 파이썬 프로그래밍 언어 커뮤니티에서 주관하는 비영리 컨퍼런스입니다.

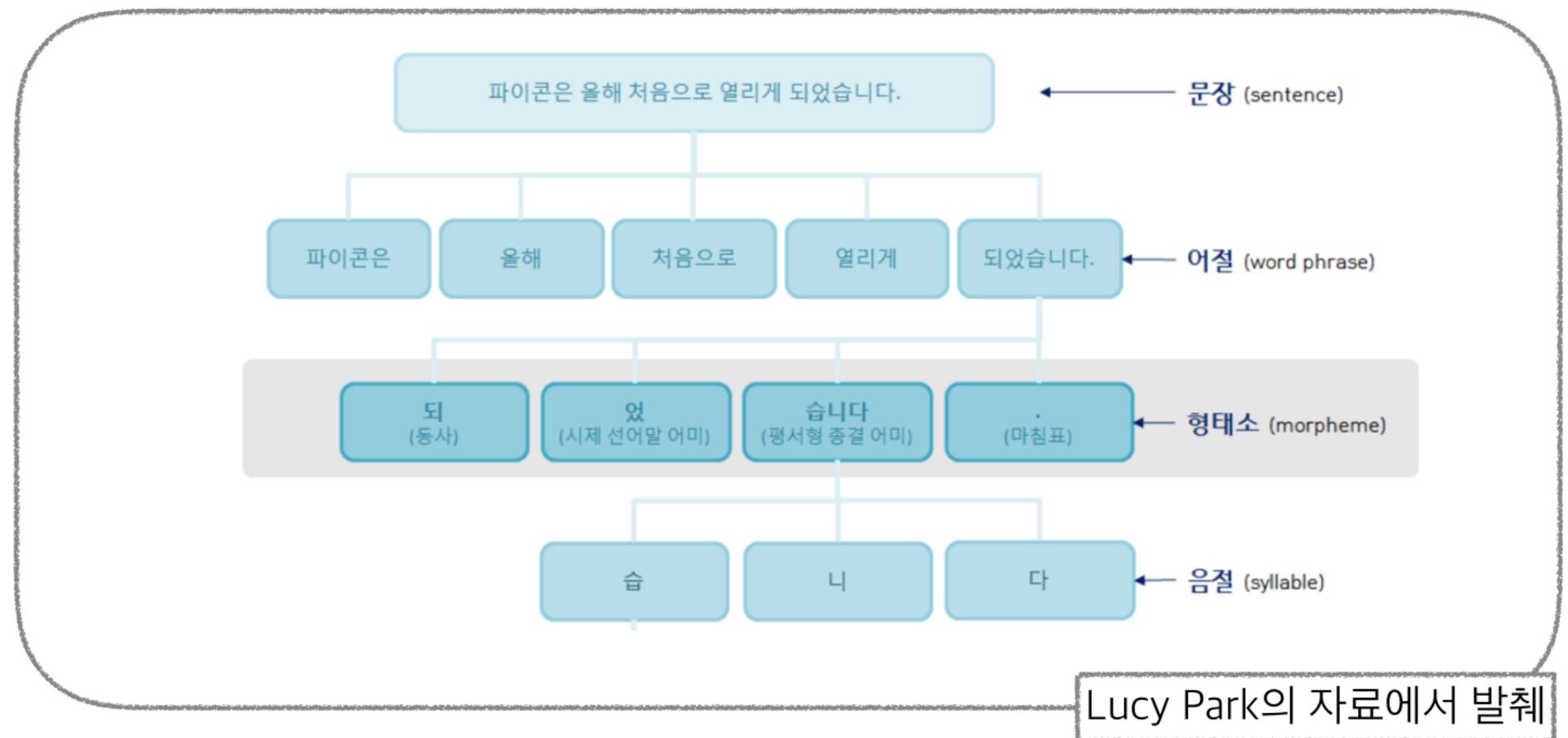
파이썬 마을을 시작으로 한 한국 파이썬 커뮤니티는 벌써 그 역사가 15년이나 되었지만, 한국 파이썬 사용자들을 위한 파이콘은 올해 처음으로 열리게 되었습니다. 본 컨퍼런스를 준비/운영하는 파이콘 한국팀은 건강한 국내 파이썬 생태계에

문서 (document)

문단 (paragraph)

문장 (sentence)

Lucy Park의 자료에서 발췌



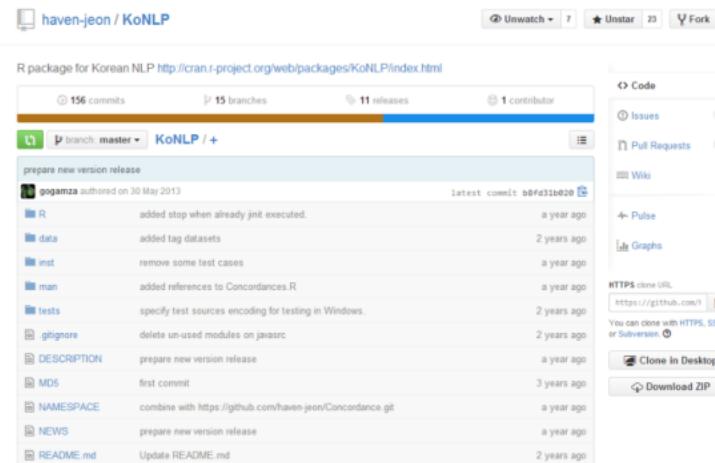
형태소

언어의 최소 의미 단위.

Lucy Park의 자료에서 발췌

KoNLP, for R

<https://github.com/haven-jeon/KoNLP>



- 한나눔 형태소 분석기 R interface
- 세종계획 한국어 코퍼스, 사전 등을 마련한 10년 계획 정부사업의 확장적 사용
- 그 외 NLP를 편리하게 하는 각종 함수 구현
- 많은 down-to-earth 예제를 담은 documentation
- "Python으로도 이런게 있으면 좋겠다!" 이름에도 내포돼있듯 KoNLP의 가장 큰 ins

Lucy Park의 자료에서 발췌

NLTK, for Python

<http://nltk.org>

NLTK 3.0 documentation

[NEXT](#) | [MODULES](#) | [INDEX](#)

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](#) such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning, and an active [discussion forum](#).

Thanks to a hands-on guide introducing programming fundamentals alongside topics in computational linguistics, 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](#) 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. The book is being updated for Python 3 and NLTK 3. (The original Python 2 version is still available at http://nltk.org/book_2ed.)

TABLE OF CONTENTS

NLTK News
Installing NLTK
Installing NLTK Data
Contribute to NLTK
FAQ
Wiki
API
HOWTO

SEARCH

Enter search terms or a module, class or function name.

- Porter, snowball, Lancaster 등 다양한 stemming 알고리즘 포함
- 그 외 chunking, NER, classification 알고리즘 포함
- 50개가 넘는 (주로 영어지만 다양한 언어의) 코퍼스 포함
- 역시 풍부한 문서
- (Natural) language free, platform free, and free
- "한국어만 지원되면 정말 좋겠다!"

Lucy Park의 자료에서 발췌

KoNLPy, for Python

<http://konlpy.rtfd.org>



Star 17
KoNLPy is a Python package for Korean natural language processing.

KoNLPy: Korean NLP in Python

[Build](#) [Pending](#) [Docs](#) [Issue](#)

KoNLPy (pronounced "ko en el PI") is a Python package for natural language processing (NLP) of the Korean language. For installation directions, see [here](#).

```
>>> from konlpy.tag import Kkma
>>> from konlpy.utils import pprint
>>> kkma = Kkma()
>>> pprint(kkma.sentences(u'네, 인녕하세요. 반갑습니다.'))
[네, 안녕하세요...,
 반갑습니다.]
>>> pprint(kkma.nouns(u'질문이나 견의사항은 것들 이슈 트래커에 남겨주세요.'))
[질문,
 견의,
 견의사항,
 사항,
 것들,
 이슈,
 트래커]
>>> pprint(kkma.pos(u'오류보고는 실천환경, 어려운 세지와 함께 설명을 최대한 살피해!^^'))
```

- "Standing on the shoulders of giants"
 - 2014년 7월, 한나눔 형태소 분석기만 담아 첫 릴리즈
 - 2014년 8월, 꼬꼬마, MeCab-ko 형태소 분석기도 포함하여 v0.3.0 릴리즈
 - 국회 의안 등 재사용/재배포가 가능한 공문서 위주로 toyin data 추가
 - 그 외 각종 튜토리얼, `konlpy.utils pprint` 등 편리한 함수 추가
- GitHub을 통해 누구나 논의와 개발에 참여할 수 있습니다!

Lucy Park의 자료에서 발췌

가볍게 한 번 터치해보자 ~~~

```
In [3]: from konlpy.tag import Kkma
kkma = Kkma()
```

- 꼬꼬마 엔진을 불러오자 ~~~

```
In [4]: kkma.sentences("한국어 분석을 시작합니다 재미있어요~~")
```

```
Out[4]: ['한국어 분석을 시작합니다', '재미있어요~~']
```

- 문장 분석을 시켜보자…
- 마침표가 없어도 문장으로 구분할 줄 안다…

```
In [5]: kkma.nouns("한국어 분석을 시작합니다 재미있어요~~")
```

```
Out[5]: ['한국어', '분석']
```

- 명사 분석을 수행해 보자…

```
In [6]: kkma.pos("한국어 분석을 시작합니다 재미있어요~~")
```

```
Out[6]: [('한국어', 'NNG'),  
          ('분석', 'NNG'),  
          ('을', 'JKO'),  
          ('시작하', 'VV'),  
          ('습니다', 'EFN'),  
          ('재미있', 'VA'),  
          ('어요', 'EFN'),  
          ('~~', 'SW')]
```

- 형태소 분석을 시켜보자 ~~~

```
In [7]: from konlpy.tag import Hannanum  
  
hannanum = Hannanum()
```

- 이번에는 한나눔 엔진을 사용해 보자…

```
In [8]: hannanum.nouns("한국어 분석을 시작합니다 재미있어요~~")
```

```
Out[8]: ['한국어', '분석', '시작']
```

```
In [9]: hannanum.morphs("한국어 분석을 시작합니다 재미있어요~~")
```

```
Out[9]: ['한국어', '분석', '을', '시작', '하', 'ㅂ니다', '재미있', '어요', '~~']
```

```
In [10]: hannanum.pos("한국어 분석을 시작합니다 재미있어요~~")
```

```
Out[10]: [('한국어', 'N'),  
          ('분석', 'N'),  
          ('을', 'J'),  
          ('시작', 'N'),  
          ('하', 'X'),  
          ('ㅂ니다', 'E'),  
          ('재미있', 'P'),  
          ('어요', 'E'),  
          ('~~', 'S')]
```

```
In [11]: from konlpy.tag import Twitter  
  
t = Twitter()
```

- 이번에는 트위터 엔진 ~~~

```
In [12]: t.nouns("한국어 분석을 시작합니다 재미있어요~~")
```

```
Out[12]: ['한국어', '분석', '시작']
```

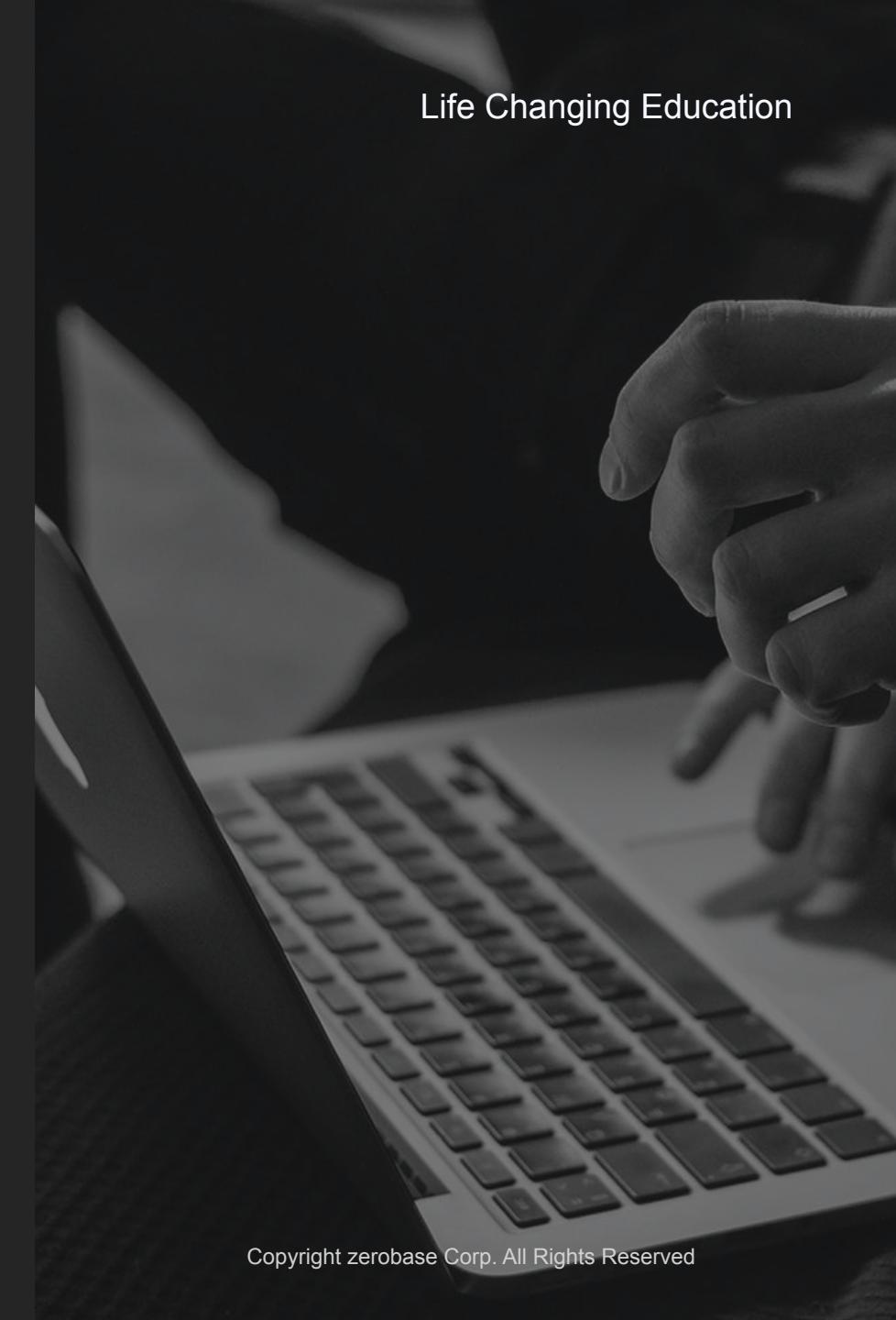
```
In [13]: t.morphs("한국어 분석을 시작합니다 재미있어요~~")
```

```
Out[13]: ['한국어', '분석', '을', '시작', '합니다', '재미있어요', '~~']
```

```
In [14]: t.pos("한국어 분석을 시작합니다 재미있어요~~")
```

```
Out[14]: [('한국어', 'Noun'),  
          ('분석', 'Noun'),  
          ('을', 'Josa'),  
          ('시작', 'Noun'),  
          ('합니다', 'Verb'),  
          ('재미있어요', 'Adjective'),  
          ('~~', 'Punctuation')]
```

워드클라우드



아주 재미있는

Word Cloud 한 번 하고 가죠^^

```
In [15]: from wordcloud import WordCloud, STOPWORDS  
  
import numpy as np  
from PIL import Image
```

- 워드클라우드 해보기 ~~~

```
In [16]: text = open("../data/06_alice.txt").read()
alice_mask = np.array(Image.open("../data/06_alice_mask.png"))

stopwords = set(STOPWORDS)
stopwords.add("said")
```

- 그림파일 하나 읽어놓고…
- 이상한 나라의 앨리스 소설도 읽어두고…
- 본문에서 많이 등장하는 said 단어는 stopword 처리하도록 하고…

```
In [17]: import matplotlib.pyplot as plt
import platform

path = "c:/Windows/Fonts/malgun.ttf"
from matplotlib import font_manager, rc

if platform.system() == "Darwin":
    rc("font", family="AppleGothic")
elif platform.system() == "Windows":
    font_name = font_manager.FontProperties(fname=path).get_name()
    rc("font", family=font_name)
else:
    print("Unknown system... sorry~~~~")

get_ipython().run_line_magic("matplotlib", "inline")
```

```
In [18]: plt.figure(figsize=(8, 8))
plt.imshow(alice_mask, cmap=plt.cm.gray, interpolation="bilinear")
plt.axis("off")
plt.show()
```

- 앤리스 그림 ~~~



```
In [19]: wc = WordCloud(  
    background_color="white", max_words=2000, mask=alice_mask, stopwords=stopwords  
)  
wc = wc.generate(text)  
wc.words_
```



```
Out[19]: {'Alice': 1.0,  
          'little': 0.29508196721311475,  
          'one': 0.27595628415300544,  
          'know': 0.2459016393442623,  
          'went': 0.226775956284153,  
          'thing': 0.2185792349726776,  
          'time': 0.2103825136612022,  
          'Queen': 0.20765027322404372,  
          'see': 0.1830601092896175,  
          'King': 0.17486338797814208,  
          'well': 0.1721311475409836,}
```

- WordCloud 모듈은 자체적으로 단어를 추출해서
- 빈도수를 조사하고 정규화하는 기능을 가지고 있다.

```
In [20]: plt.figure(figsize=(12, 12))
plt.imshow(wc, interpolation="bilinear")
plt.axis("off")
plt.show()
```

- Magic WordCloud ~~~



```
In [21]: text = open("../data/06_a_new_hope.txt").read()  
  
text = text.replace("HAN", "Han")  
text = text.replace("LUKE'S", "Luke")  
  
mask = np.array(Image.open("../data/06_stormtrooper_mask.png"))
```

- 이번에는 Star Wars…

```
In [22]: stopwords = set(STOPWORDS)
stopwords.add("int")
stopwords.add("ext")
```

- 이번에는 stopwords 에 좀 더 신경쓰고

```
In [23]: wc = WordCloud(  
    max_words=1000, mask=mask, stopwords=stopwords, margin=10, random_state=1  
).generate(text)  
  
default_colors = wc.to_array()
```

- Word Cloud 설정을 지정하고 ~~~

```
In [24]: import random

def grey_color_func(
    word, font_size, position, orientation, random_state=None, **kwargs
):
    return "hsl(0, 0%%, %d%%)" % random.randint(60, 100)
```

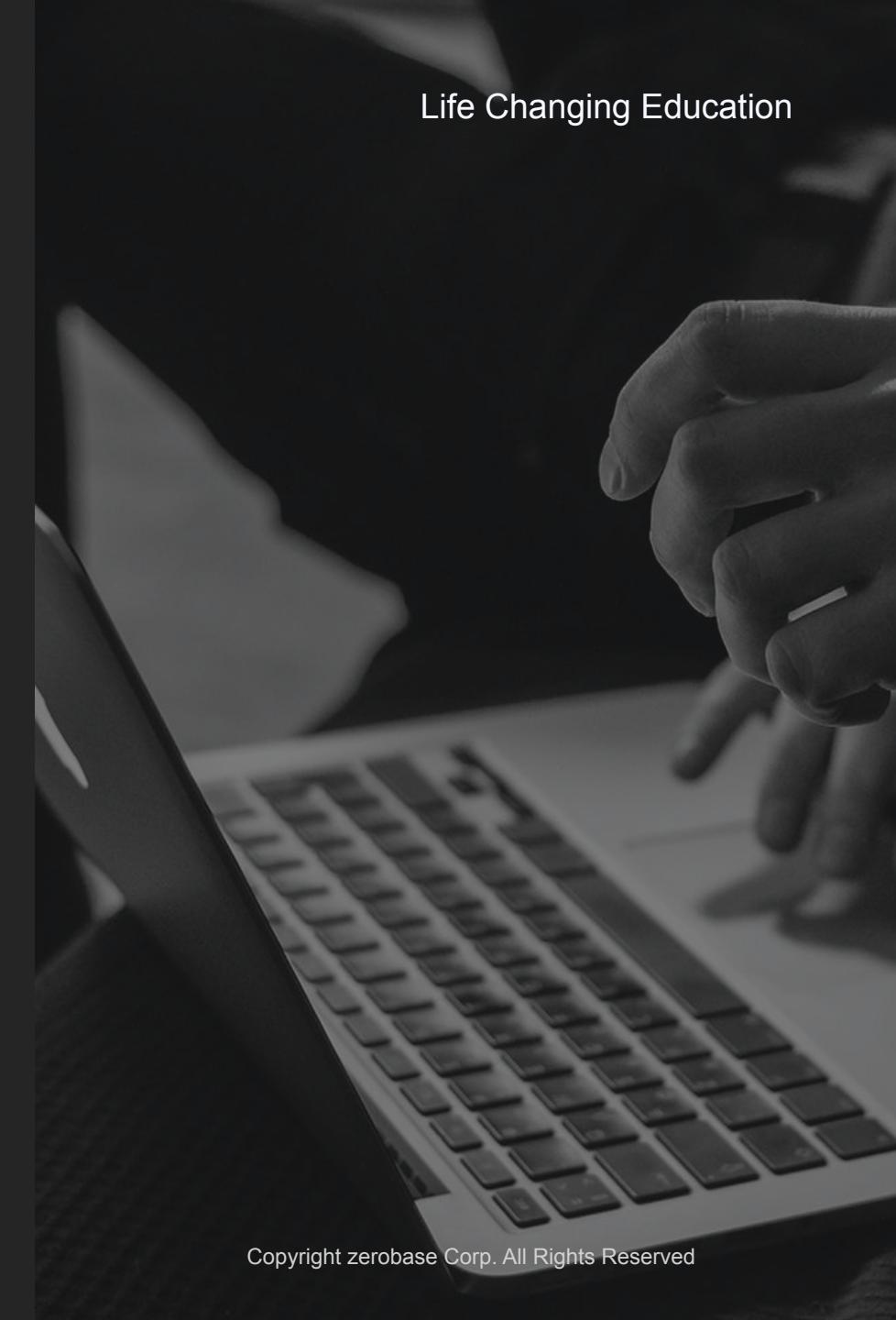
- 특별히 그레이톤으로 그리기 위한 색상함수를 정의하고…

```
In [25]: plt.figure(figsize=(12, 12))
plt.imshow(
    wc.recolor(color_func=grey_color_func, random_state=3), interpolation="bilinear"
)
plt.axis("off")
plt.show()
```

- 그려보자 ~~~



육아휴직법안분석



실제 문서 가지고 한 번 해보죠 ~~

육아 휴직 관련 법안

육아휴직관련 법안 대한민국 국회 제 1809890호 의안

```
In [26]: import nltk
```

```
In [27]: from konlpy.corpus import kobill  
  
files_ko = kobill.fileids()  
doc_ko = kobill.open("1809890.txt").read()
```

- KoNLPy는 대한 민국 법령을 가지고 있다

In [28]: doc_ko

Out[28]: '지방공무원법 일부개정법률안\n\n(정의화의원 대표발의)\n\n의 안\n번 호\n9890\n\n발의화. 이명수. 김을동 \n\n이사철. 여상규. 안규백\n\n황영철. 박영아. 김정훈\n\n김학송 의원(10인)\n\n모의 따뜻한 사랑과 보살핌이 필요\n\n한 나이이나, 현재 공무원이 자녀를 양육하기 위하여 육아휴직
어 초등학교 저학년인 \n\n자녀를 돌보기 위해서는 해당 부모님은 일자리를 그만 두어야 하고 \n\n임.\n\n따라서 육아휴직이 가능한 자녀의 연령을 만 8세 이하로 개정하려\n\n는 것임(안 제63조\n\n지방공무원법 일부개정법률안\n\n지방공무원법 일부를 다음과 같이 개정한다.\n\n제63조제2항제\n\n8세 이하(취학 중인 경우에는 초등학교 2학년 이하를 말한다)의 자녀를"\n\n로 한다.\n\n부-\n\n\x0c신 ·구조문대비표\n\n현 행\n\n개 정 안\n\n제63조(휴직) ① (생 략)
다음 각 호의 어\n\n ② ----- \n\n느 하나에 해당하는 사유로 휴\n\n임용권자는 휴직\n\n----- \n\n을 명할 수 있다. 다만, 제4호\n\n대통령령으로 정\n\n----- \n\n하는 특별한 사정이 없으면 휴\n\n한다.\n\n----- .\n\n 1. ~ 3. (생 략)\n\n 1. ~ 3. (현행과 같음)\n\n'

```
In [29]: from konlpy.tag import Twitter  
  
t = Twitter()  
tokens_ko = t.nouns(doc_ko)  
tokens_ko
```

```
Out[29]: ['지방공무원법',  
          '일부',  
          '개정',  
          '법률',  
          '안',  
          '정의화',  
          '의원',  
          '대표',  
          '발의',  
          '의',  
          '안',  
          '번',
```

- Twitter 엔진으로 명사 분석 ~~~
- Twitter => Okt

```
In [30]: ko = nltk.Text(tokens_ko, name="대한민국 국회 의안 제 1809890호")
```

```
In [31]: print(len(ko.tokens)) # returns number of tokens (document length)
print(len(set(ko.tokens))) # returns number of unique tokens
ko.vocab() # returns frequency distribution
```

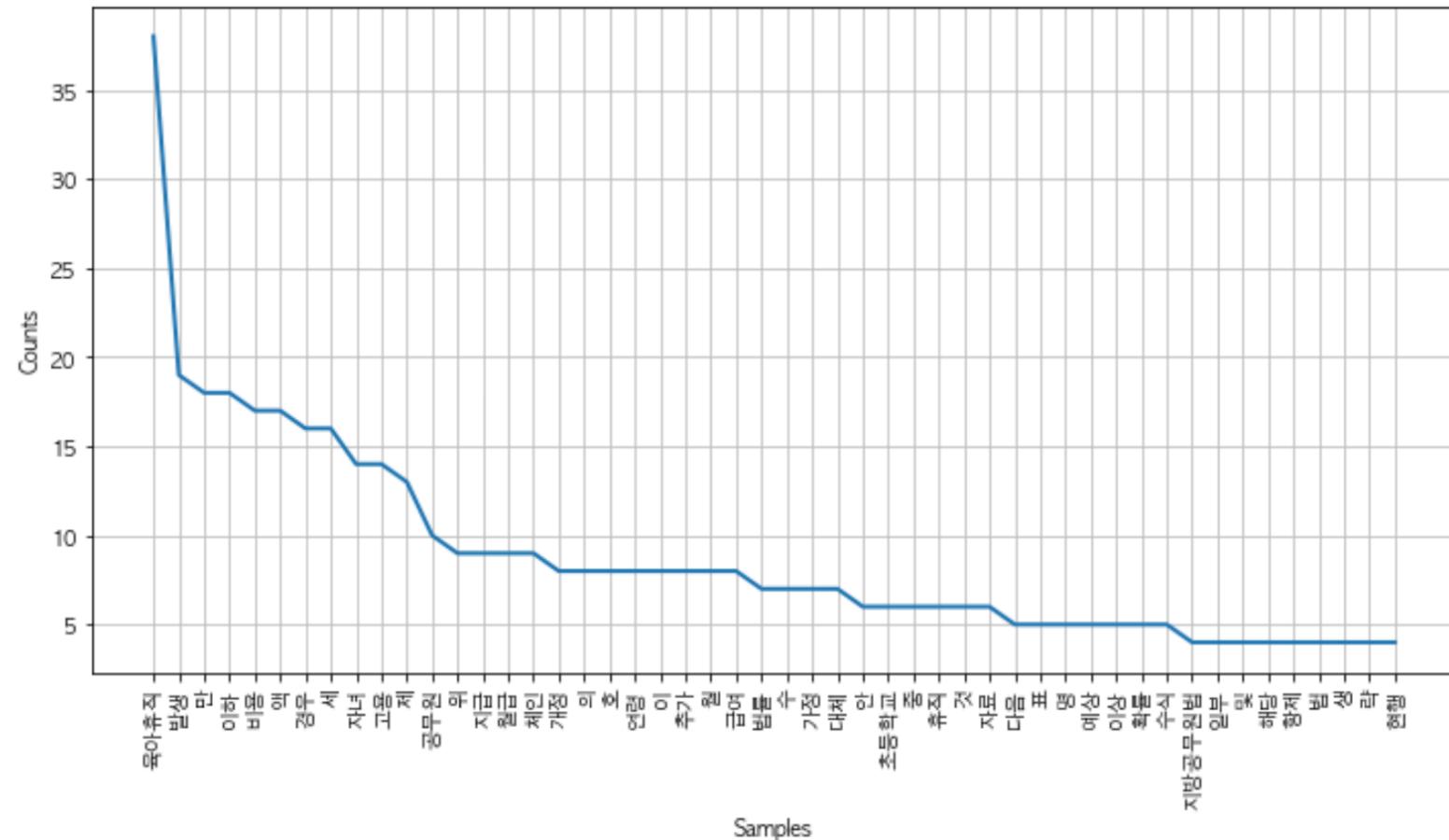
```
735
```

```
250
```

```
Out[31]: FreqDist({'육아휴직': 38, '발생': 19, '만': 18, '이하': 18, '비용': 17, '액': 17, '경우': 16, '세': 16, '자녀': 14, '고용': 14, ...})
```

- nltk 를 사용해서 토큰(빈도수 포함) 분석 ~~~

```
In [32]: plt.figure(figsize=(12, 6))
ko.plot(50)
plt.show()
```



```
In [33]: stop_words = [
```

```
    ".",
    "(",
    ")",
    ",",
    "'",
    "。",
    "-",
    "X",
    ") .",
    "x",
    "의",
    "자",
    "에",
    "안",
    "번",
    "호",
    "을",
    "이",
    "다",
    "만",
    "로",
    "가",
    "를",
```

- 한글 stopword는 상황에 따라 복잡해서, 일단 손으로 그냥 잡음…

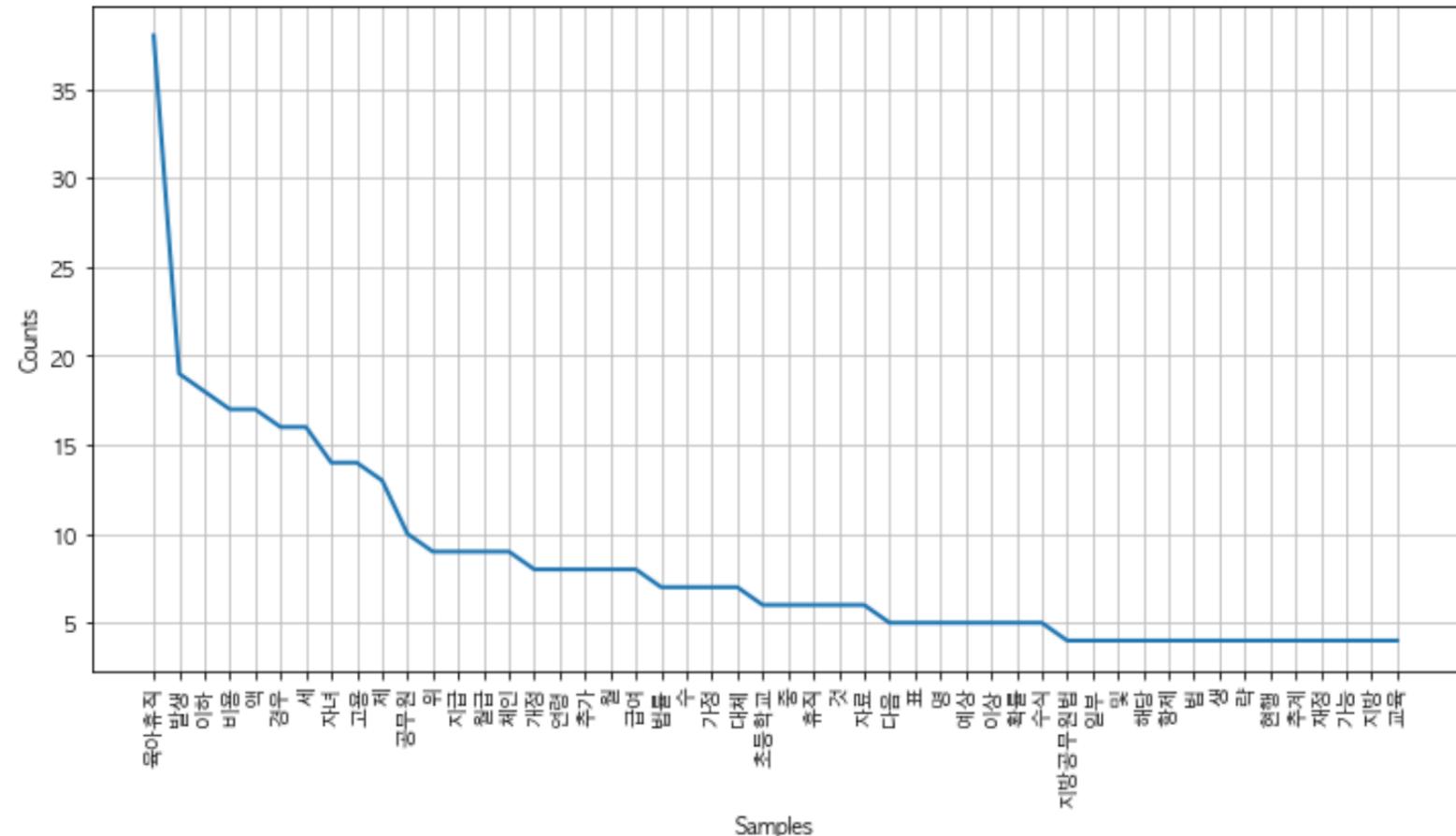
```
]
```

```
ko = [each_word for each_word in ko if each_word not in stop_words]  
ko
```

```
'직',  
'임용',  
'휴직',  
'명',  
'수',  
'다만',  
'제',  
'경우',  
'대통령령',  
'정',  
'사정',  
'직',  
'명',  
'생',  
'략',  
'현행',  
'세',  
'이하',
```

```
In [34]: ko = nltk.Text(ko, name="대한민국 국회 의안 제 1809890호")

plt.figure(figsize=(12, 6))
ko.plot(50) # Plot sorted frequency of top 50 tokens
plt.show()
```

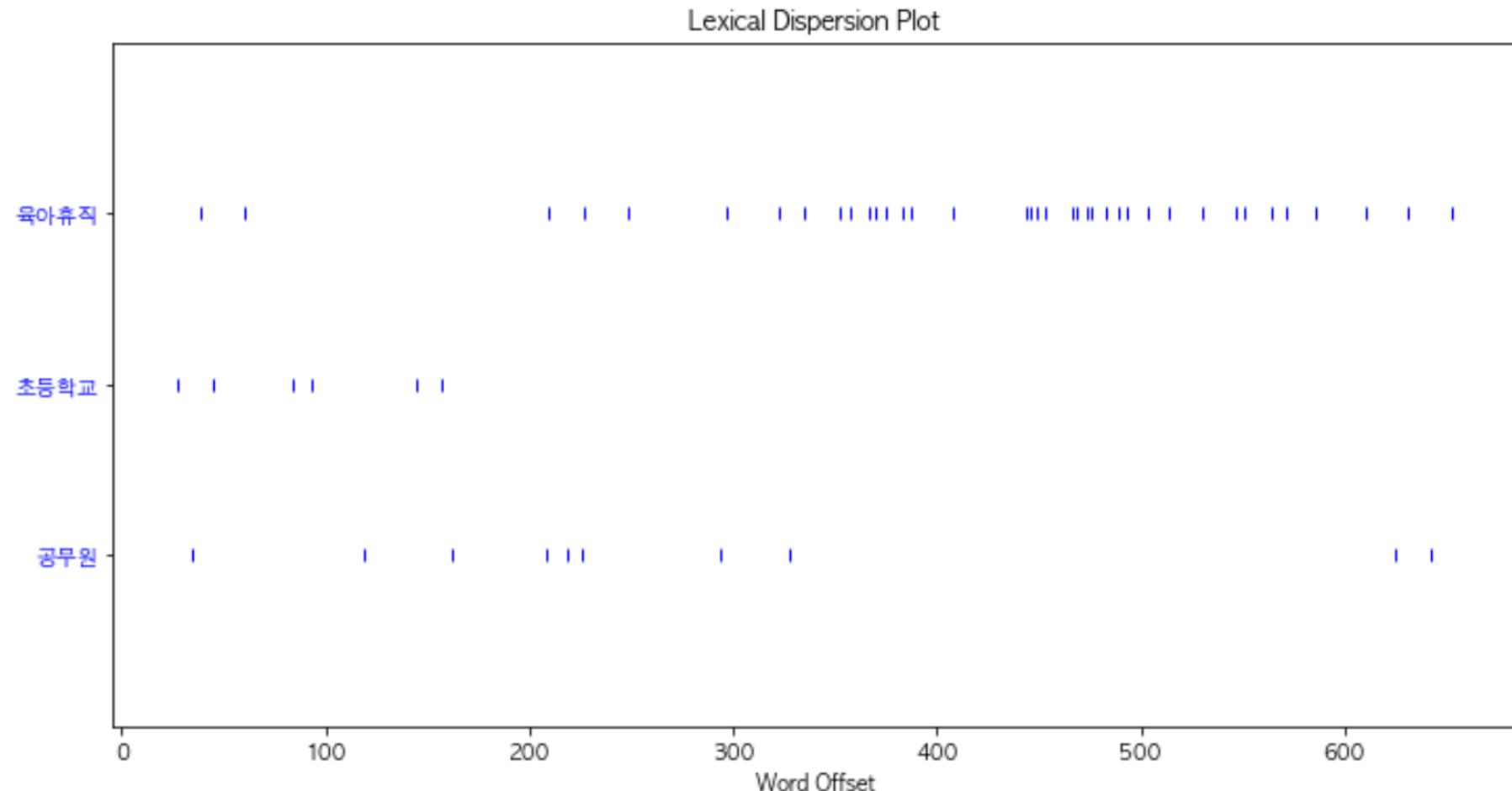


```
In [35]: ko.count("초등학교")
```

```
Out[35]: 6
```

- 특정 단어의 빈도수 조사

```
In [36]: plt.figure(figsize=(12, 6))
ko.dispersion_plot(["육아휴직", "초등학교", "공무원"])
```



```
In [37]: ko.concordance("초등학교")
```

Displaying 6 of 6 matches:

안규백 황영철 박영아 김정훈 김학송 의원 인 제안 이유 및 내용 초등학교 저학년 경우 부모 사랑 필요 나이 현재 공무원 자녀 양육 위 육아
나이 현재 공무원 자녀 양육 위 육아휴직 수 자녀 나이 세 이하 초등학교 저학년 자녀 위 해당 부모님 일자리 곧 출산 의욕 저하 문제 수
일부 개정 법률 지방공무원법 일부 다음 개정 제 항제 중 세 이하 초등학교 취학 전 자녀 세 이하 취학 중인 경우 초등학교 학년 이하 말 자
항제 중 세 이하 초등학교 취학 전 자녀 세 이하 취학 중인 경우 초등학교 학년 이하 말 자녀 부 칙 법 공포 날 시행 신 구조 문대비 표
수 다만 제 경우 대통령령 정 사정 직 명 생 략 현행 세 이하 초등학교 취 세 이하 취학 중인 경우 학 전 자녀 양육 위 초등학교 학년
이하 초등학교 취 세 이하 취학 중인 경우 학 전 자녀 양육 위 초등학교 학년 이하 여 여자 공무원 말 자녀 임신 출산 때 생 략 생 략

영어에서 연어(連語, collocation)란 무엇인가?

연어(連語, collocation)란 "함께 위치하는 단어들"(co + location)이란 뜻으로, 어휘의 조합 또는 짝을 이루는 말들을 일컫습니다. 예를 들면 'friend'와 주로 같이 쓰이는 형용사는 'best', 'good', 그리고 'loyal'입니다. 예문은 다음과 같습니다.

"She talks to her **best** friend every day."

(best)

출처: <http://ko.talkenglish.com/how-to-use/collocations.aspx>

In [38]: ko.collocations()

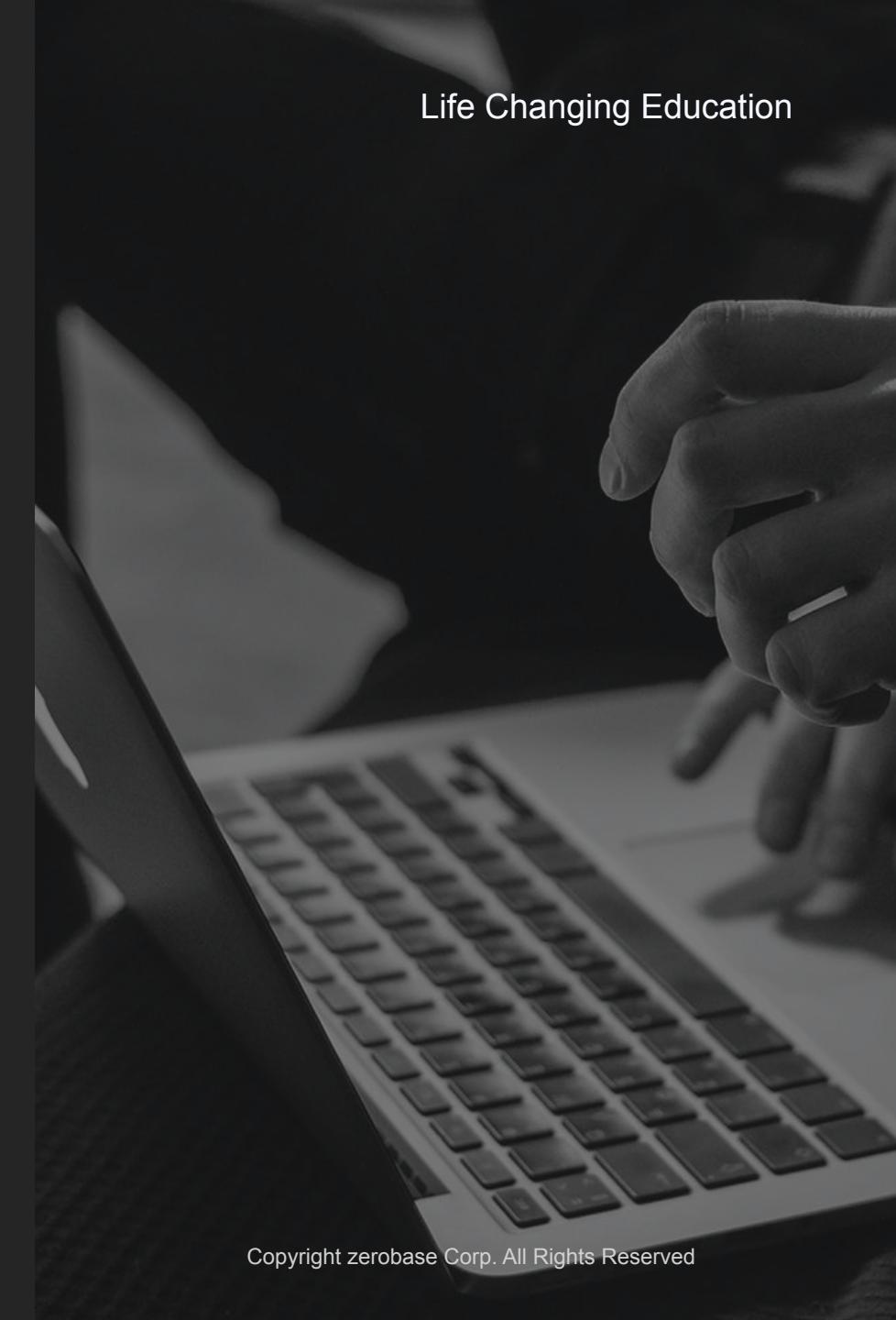
초등학교 저학년; 근로자 육아휴직; 육아휴직 대상자; 공무원 육아휴직

```
[n 104]: data = ko.vocab().most_common(150)

# for win : font_path='c:/Windows/Fonts/malgun.ttf'
wordcloud = WordCloud(
    font_path="/Library/Fonts/Arial Unicode.ttf",
    relative_scaling=0.2,
    background_color="white",
).generate_from_frequencies(dict(data))
plt.figure(figsize=(12, 8))
plt.imshow(wordcloud)
plt.axis("off")
plt.show()
```



나이브베이즈 분류



Naïve Bayes Classifier

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

나이브 베이즈 분류

위키백과, 우리 모두의 백과사전.

기계 학습분야에서, '나이브 베이즈 분류(Naïve Bayes Classification)'는 특성들 사이의 독립을 가정하는 베이즈 정리를 적용한 확률 분류기의 일종으로 1950년대 이후 광범위하게 연구되고 있다.

통계 및 컴퓨터 과학 문헌에서, 나이브 베이즈는 단순 베이즈, 독립 베이즈를 포함한 다양한 이름으로 알려져 있으며, 1960년대 초에 텍스트 검색 커뮤니티에 다른 이름으로 소개되기도 하였다.^[1]

나이브 베이즈 분류는 텍스트 분류에 사용됨으로써 문서를 여러 범주 (예: 스팸, 스포츠, 정치) 중 하나로 판단하는 문제에 대한 대중적인 방법으로 남아있다. 또한, 자동 의료 진단 분야에서의 응용사례^[2]를 보면, 적절한 전처리를 하면 더 진보된 방법들 (예: 서포트 벡터 머신 (Support Vector Machine))과도 충분한 경쟁력을 보임을 알 수 있다.

출처: https://ko.wikipedia.org/wiki/나이브_베이즈_분류

Naive Bayes Classifier

```
In [40]: from nltk.tokenize import word_tokenize
        import nltk
```

- 영어로 먼저 경험해 보자… 영어는 nltk만 있으면 된다~~~

```
In [41]: train = [  
    ("i like you", "pos"),  
    ("i hate you", "neg"),  
    ("you like me", "neg"),  
    ("i like her", "pos"),  
]
```

- Naïve Bayes 분류기는 지도학습이라서 정답을 알려주어야 한다…

```
In [42]: all_words = set(  
    word.lower() for sentence in train for word in word_tokenize(sentence[0]))  
all_words  
  
Out[42]: {'hate', 'her', 'i', 'like', 'me', 'you'}
```

- 전체 말뭉치를 만든다...

```
In [43]: t = [{word: (word in word_tokenize(x[0])) for word in all_words}, x[1]) for x in train]  
t
```

```
Out[43]: [({'her': False,  
            'like': True,  
            'hate': False,  
            'you': True,  
            'i': True,  
            'me': False},  
           'pos'),  
          ({'her': False,  
            'like': False,  
            'hate': True,  
            'you': True,  
            'i': True,  
            'me': False},  
           'neg'),  
          ({'her': False,  
            'like': True,  
            'hate': False,  
            'you': True,  
            'i': False,  
            'me': True},  
           'neg'),
```

- 말 둉치 대비해서 단어가 있고 없음을 표기한다.

```
In [44]: classifier = nltk.NaiveBayesClassifier.train(t)
classifier.show_most_informative_features()
```

Most Informative Features

hate = False	pos : neg	=	1.7 : 1.0
her = False	neg : pos	=	1.7 : 1.0
i = True	pos : neg	=	1.7 : 1.0
like = True	pos : neg	=	1.7 : 1.0
me = False	pos : neg	=	1.7 : 1.0
you = True	neg : pos	=	1.7 : 1.0

- Naïve Bayes 분류가 훈련을 시작했다…
- like가 있을 때, Positive할 확률이 1.7:1.00이다… 뭐 이런식으로~~~
- 이렇게 각 단어별로 독립적으로 확률을 계산하기 때문에 naïve하다고 한다~~

```
In [45]: test_sentence = "i like MeRui"
test_sent_features = {
    word.lower(): (word in word_tokenize(test_sentence.lower())) for word in all_words
}
test_sent_features
```



```
Out[45]: {'her': False,
          'like': True,
          'hate': False,
          'you': False,
          'i': True,
          'me': False}
```

- 이제 학습 결과를 가지고 테스트 해보자… I like MeRui~~~

```
In [46]: classifier.classify(test_sent_features)
```

```
Out[46]: 'pos'
```

- 결과는 positive

뭐~ 이제… 한글로 해보자~~~

```
In [47]: from konlpy.tag import Twitter
```

```
In [48]: pos_tagger = Twitter()
```

```
In [49]: train = [
    ("메리가 좋아", "pos"),
    ("고양이도 좋아", "pos"),
    ("난 수업이 지루해", "neg"),
    ("메리는 이쁜 고양이야", "pos"),
    ("난 마치고 메리랑 놀거야", "pos"),
]
```

- 정답을 알고 있는 문장으로 훈련용 데이터를 주자~~~

```
In [50]: all_words = set(  
    word.lower() for sentence in train for word in word_tokenize(sentence[0]))  
)  
all_words
```

```
Out[50]: {'고양이도',  
          '고양이야',  
          '난',  
          '놀거야',  
          '마치고',  
          '메리가',  
          '메리는',  
          '메리랑',  
          '수업이',  
          '이쁜',  
          '좋아',  
          '지루해'}
```

- 전체 말뭉치를 만들자…
- 앗.. 이런… 메리가, 메리는, 메리랑을 모두 다른 단어로 인식한다.ㅠㅠ.

```
In [51]: t = [( {word: (word in word_tokenize(x[0])) for word in all_words}, x[1]) for x in train]  
t
```

```
Out[51]: [ ({'고양이도': False,  
'메리가': True,  
'이쁜': False,  
'좋아': True,  
'고양이야': False,  
'마치고': False,  
'지루해': False,  
'난': False,  
'메리랑': False,  
'메리는': False,  
'놀거야': False,  
'수업이': False},  
'pos'),
```

```
In [52]: classifier = nltk.NaiveBayesClassifier.train(t)
classifier.show_most_informative_features()
```

Most Informative Features

난 = True	neg : pos =	2.5 : 1.0
좋아 = False	neg : pos =	1.5 : 1.0
고양이도 = False	neg : pos =	1.1 : 1.0
고양이야 = False	neg : pos =	1.1 : 1.0
놀거야 = False	neg : pos =	1.1 : 1.0
마치고 = False	neg : pos =	1.1 : 1.0
메리가 = False	neg : pos =	1.1 : 1.0
메리는 = False	neg : pos =	1.1 : 1.0
메리랑 = False	neg : pos =	1.1 : 1.0
이쁜 = False	neg : pos =	1.1 : 1.0

- 벌써부터 내가 원하는 대로 가지 않을 것 같은 느낌이 든다…

```
In [53]: test_sentence = "난 수업이 마치면 메리랑 놀거야"
```

```
In [54]: test_sent_features = {
    word.lower(): (word in word_tokenize(test_sentence.lower())) for word in all_words
}
test_sent_features
```

```
Out[54]: {'고양이도': False,
          '메리가': False,
          '이쁜': False,
          '좋아': False,
          '고양이야': False,
          '마치고': False,
          '지루해': False,
          '난': True,
          '메리랑': True,
          '메리는': False,
          '놀거야': True,
          '수업이': True}
```

```
In [55]: classifier.classify(test_sent_features)
```

```
Out[55]: 'neg'
```

- negative ???
- 왜 ???

이래서… 한글은 형태소 분석이 필수…
그래서 여기서는
Lucy Park님의 추천대로 진행하자…

```
In [56]: def tokenize(doc):
    return ["/".join(t) for t in pos_tagger.pos(doc, norm=True, stem=True)]
```

- 형태소 분석을 한 후 품사를 단어 뒤에 붙여 넣도록 하자

```
In [57]: train_docs = [(tokenize(row[0]), row[1]) for row in train]
train_docs
```

```
Out[57]: [[['메리/Noun', '가/Josa', '좋다/Adjective'], 'pos'),
           [['고양이/Noun', '도/Josa', '좋다/Adjective'], 'pos'),
           [['난/Noun', '수업/Noun', '이/Josa', '지루하다/Adjective'], 'neg'),
           [['메리/Noun', '는/Josa', '이쁘다/Adjective', '고양이/Noun', '야/Josa'], 'pos'),
           [['난/Noun', '마치/Noun', '고/Josa', '메리/Noun', '랑/Josa', '놀다/Verb'], 'pos'))]
```

- 이렇게 된다…

```
In [58]: tokens = [t for d in train_docs for t in d[0]]  
tokens
```

```
Out[58]: ['메리/Noun',  
          '가/Josa',  
          '좋다/Adjective',  
          '고양이/Noun',  
          '도/Josa',  
          '좋다/Adjective',  
          '난/Noun',  
          '수업/Noun',  
          '이/Josa',  
          '지루하다/Adjective',  
          '메리/Noun',  
          '는/Josa',  
          '이쁘다/Adjective',  
          '고양이/Noun',  
          '야/Josa',  
          '난/Noun',  
          '마치/Noun',  
          '고/Josa',  
          '메리/Noun',  
          '랑/Josa',  
          '놀다/Verb']
```

- 풀어서 말뭉치를 만들자…

```
In [59]: def term_exists(doc):
    return {word: (word in set(doc)) for word in tokens}
```

```
In [60]: train_xy = [(term_exists(d), c) for d, c in train_docs]
train_xy
```

```
Out[60]: [({'메리/Noun': True,
            '가/Josa': True,
            '좋다/Adjective': True,
            '고양이/Noun': False,
            '도/Josa': False,
            '난/Noun': False,
            '수업/Noun': False,
            '이/Josa': False,
            '지루하다/Adjective': False,
            '는/Josa': False,
            '이쁘다/Adjective': False,
            '야/Josa': False},
```

```
In [61]: classifier = nltk.NaiveBayesClassifier.train(train_xy)

In [62]: test_sentence = [("난 수업이 마치면 메리랑 놀거야")]

In [63]: test_docs = pos_tagger.pos(test_sentence[0])
test_docs

Out[63]: [('난', 'Noun'),
           ('수업', 'Noun'),
           ('이', 'Josa'),
           ('마치', 'Noun'),
           ('면', 'Josa'),
           ('메리', 'Noun'),
           ('랑', 'Josa'),
           ('놀거야', 'Verb')]
```

```
In [64]: classifier.show_most_informative_features()
```

Most Informative Features

난/Noun = True	neg : pos	=	2.5 : 1.0
메리/Noun = False	neg : pos	=	2.5 : 1.0
고양이/Noun = False	neg : pos	=	1.5 : 1.0
좋다/Adjective = False	neg : pos	=	1.5 : 1.0
가/Josa = False	neg : pos	=	1.1 : 1.0
고/Josa = False	neg : pos	=	1.1 : 1.0
놀다/Verb = False	neg : pos	=	1.1 : 1.0
는/Josa = False	neg : pos	=	1.1 : 1.0
도/Josa = False	neg : pos	=	1.1 : 1.0
랑/Josa = False	neg : pos	=	1.1 : 1.0

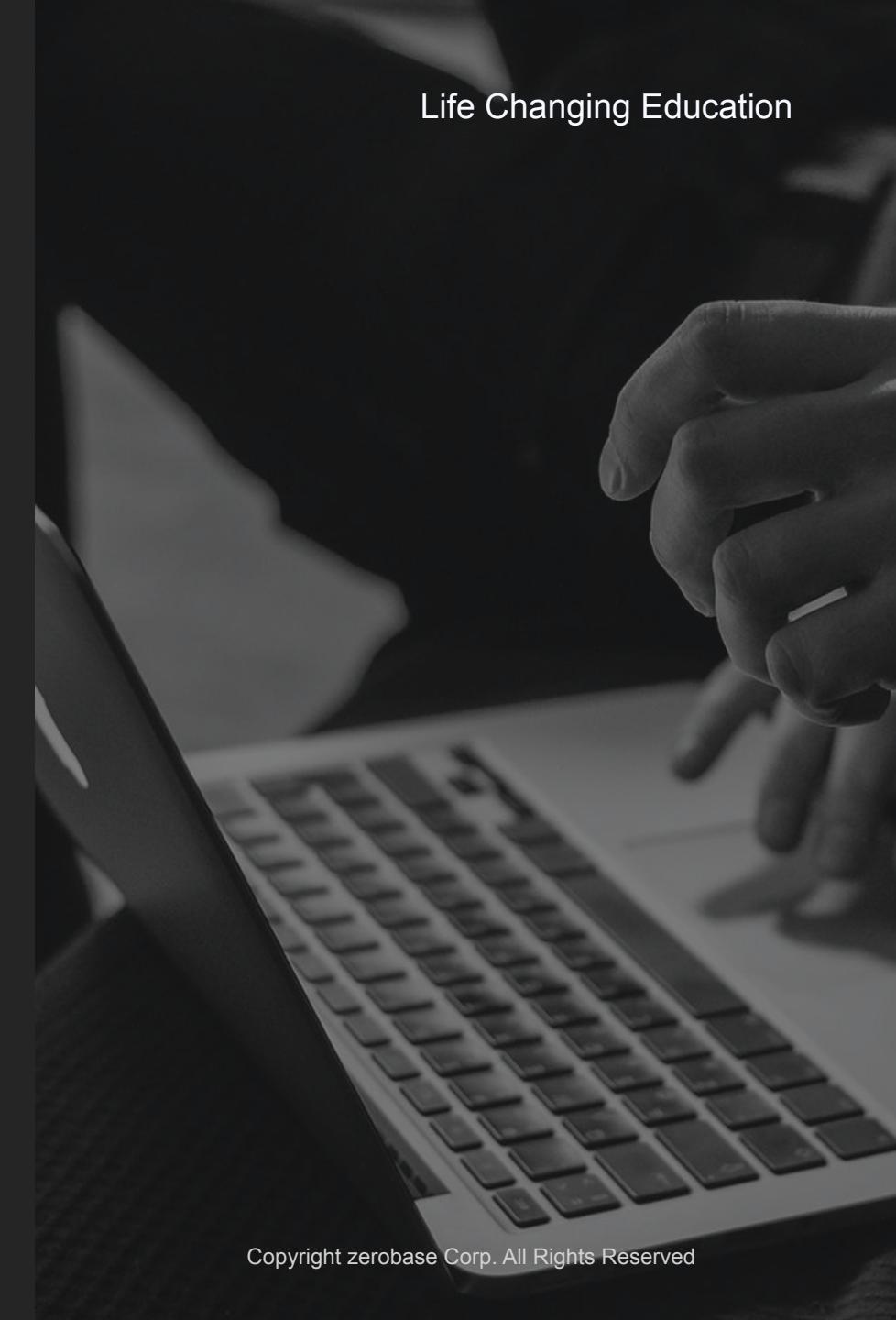
```
In [65]: test_sent_features = {word: (word in tokens) for word in test_docs}  
test_sent_features
```

```
Out[65]: {('난', 'Noun'): False,  
          ('수업', 'Noun'): False,  
          ('이', 'Josa'): False,  
          ('마치', 'Noun'): False,  
          ('면', 'Josa'): False,  
          ('메리', 'Noun'): False,  
          ('랑', 'Josa'): False,  
          ('놀거야', 'Verb'): False}
```

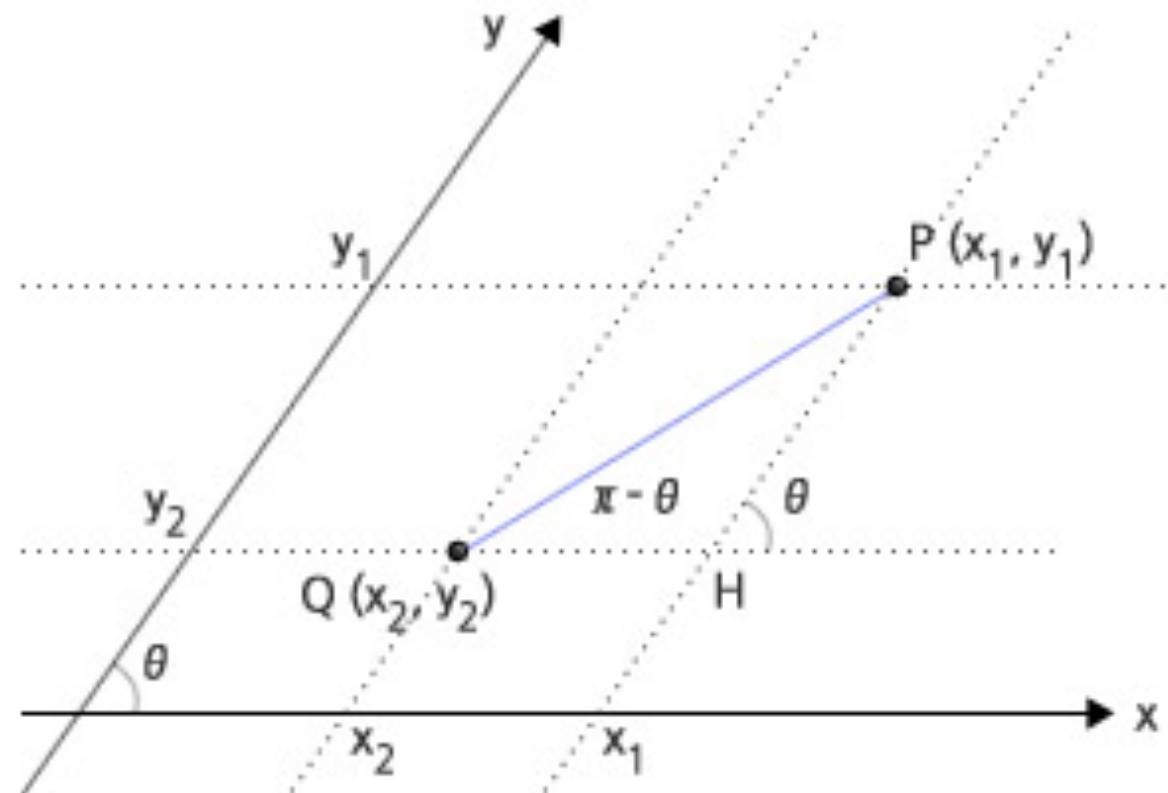
```
In [66]: classifier.classify(test_sent_features)
```

```
Out[66]: 'pos'
```

문장의 유사도



만약 두 점사이의 거리를 구하는 것이라면 쉽죠~~~



만약 문장을 점처럼

일종의 벡터로 표현할 수 있다면

두 문장 사이의 거리를 구해서…

유사한 문장…

좀 더 유사한 문장을 찾을 수 있을까요?

CountVectorizer

zero-base /

```
[36] 1 from sklearn.feature_extraction.text import CountVectorizer
```

✓ 0.2s

```
[37] 1 vectorizer = CountVectorizer(min_df=1)
```

✓ 0.2s

- **sklearn**이 제공하는 문장을 벡터로 변환하는 함수 **CountVectorizer**

```
[36] 1 from sklearn.feature_extraction.text import CountVectorizer
      ✓ 0.2s

[37] 1 vectorizer = CountVectorizer(min_df=1)
      ✓ 0.2s

[38] 1 contents = ['상처받은 아이들은 너무 일찍 커버려',
      2           |         |         |
      3           |         |         |
      4           |         |         |
      '내가 상처받은 거 아는 사람 불편해',
      '잘 사는 사람들은 좋은 사람 되기 쉬워',
      '아무 일도 아니야 괜찮아']
```

- 이번에 사용할 훈련용 문장
- 거리를 구하는 것이므로 지도할 내용이 없다

```
1 from konlpy.tag import Okt  
2  
3 t = Okt()  
[6] ✓ 0.9s
```

```
1 contents_tokens = [t.morphs(row) for row in contents]  
2 contents_tokens
```

```
[40] ✓ 0.8s  
... [[['상처', '받은', '아이', '들', '은', '너무', '일찍', '커버', '려'],  
  ['내', '가', '상처', '받은', '거', '아는', '사람', '불편해'],  
  ['잘', '사는', '사람', '들', '은', '좋은', '사람', '되기', '쉬워'],  
  ['아무', '일도', '아니야', '괜찮아']]
```

- 형태소 분석 엔진은 Okt
- 한글은 형태소 분석이 필수 ~

```
1 contents_for_vectorize = []
2
3 for content in contents_tokens:
4     sentence = ''
5     for word in content:
6         sentence = sentence + ' ' + word
7
8     contents_for_vectorize.append(sentence)
9
10 contents_for_vectorize
```

[41]

✓ 0.3s

```
... ['상처 받은 아이 들 은 너무 일찍 커버 려',
  '내 가 상처 받은 거 아는 사람 불편해',
  '잘 사는 사람 들 은 좋은 사람 되기 쉬워',
  '아무 일도 아니야 괜찮아']
```

- 형태소 분석된 결과를 다시 하나의 문장식으로 합친다(논란의 여지가 있지만)

```
[42] 1 X = vectorizer.fit_transform(contents_for_vectorize)
      2 X
      ✓ 0.2s
...
<4x17 sparse matrix of type '<class 'numpy.int64'>'  

      with 20 stored elements in Compressed Sparse Row format>
```

```
[43] 1 num_samples, num_features = X.shape
      2 num_samples, num_features
      ✓ 0.4s
...
(4, 17)
```

- 벡터 라이즈 수행
- 네 개의 문장에 전체 말뭉치의 단어가 17개였다

```
▶ ▾ 1 vectorizer.get_feature_names()
[44] ✓ 0.4s
...
['괜찮아',
 '너무',
 '되기',
 '받은',
 '불편해',
 '사는',
 '사람',
 '상처',
```

- 확인

```
▶ ▾ 1 X.toarray().transpose()  
[45] ✓ 0.3s  
... array([[0, 0, 0, 1],  
           [1, 0, 0, 0],  
           [0, 0, 1, 0],  
           [1, 1, 0, 0],  
           [0, 1, 0, 0],  
           [0, 0, 1, 0],  
           [0, 1, 2, 0],  
           [1, 1, 0, 0],  
           [0, 0, 1, 0],  
           [0, 1, 0, 0],  
           [0, 0, 0, 1],  
           [0, 0, 0, 1],  
           [1, 0, 0, 0],  
           [0, 0, 0, 1],  
           [1, 0, 0, 0],  
           [0, 0, 1, 0],  
           [1, 0, 0, 0]])  
['괜찮아',  
 '너무',  
 '되기',  
 '받은',  
 '불편해',  
 '사는',  
 '사람',  
 '상처',  
 '쉬워',  
 '아는',  
 '아니야',  
 '아무',  
 '아이',  
 '일도',  
 '일찍',  
 '좋은',  
 '커버']  
1 contents = ['상처받은 아이들은 너무 일찍 커버려',  
2             '내가 상처받은 거 아는 사람 불편해',  
3             '잘 사는 사람들은 좋은 사람 되기 쉬워',  
4             '아무 일도 아니야 괜찮아']
```

```
1 new_post = ['상처받기 싫어 괜찮아']
2 new_post_tokens = [t.morphs(row) for row in new_post]
3
4 new_post_for_vectorize = []
5
6 for content in new_post_tokens:
7     sentence = ''
8     for word in content:
9         sentence = sentence + ' ' + word
10
11     new_post_for_vectorize.append(sentence)
12
13 new_post_for_vectorize
```

[46]

✓ 0.1s

...

['상처 받기 싫어 괜찮아']

- 테스트용 문장

```
[47] 1 new_post_vec = vectorizer.transform(new_post_for_vectorize)
     2 new_post_vec.toarray()
    ✓ 0.3s
...
... array([[1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]])
```

- 벡터로 표현
- 새로운 테스트용 문장을 만들고 벡터를 만들었으니…
- 거리를 구할 수 있다

```
1 import scipy as sp
```

[48] ✓ 0.6s

```
1 def dist_raw(v1, v2):
2     delta = v1 - v2
3     return sp.linalg.norm(delta.toarray())
```

[49] ✓ 0.1s

- 단순히 기하학적인 거리를 사용해보자

```
1 dist = [dist_raw(each, new_post_vec) for each in X]
2 dist
[50] ✓ 0.7s
... [2.449489742783178, 2.23606797749979, 3.1622776601683795, 2.0]
```

```
1 print('Best post is ', dist.index(min(dist)), ', dist = ', min(dist))
2 print('Test post is --> ', new_post)
3 print('Best dist post is --> ', contents[dist.index(min(dist))])
[51] ✓ 0.8s
... Best post is 3 , dist = 2.0
Test post is --> ['상처받기 싫어 괜찮아']
Best dist post is --> 아무 일도 아니야 괜찮아
```

```
1 for i in range(0, len(contents)):  
2     print(X.getrow(i).toarray())  
3  
4 print('-' * 40)  
5 print(new_post_vec.toarray())
```

[52] ✓ 0.6s

```
... [[0 1 0 1 0 0 0 1 0 0 0 0 1 0 1 0 1 0 1]]  
[[0 0 0 1 1 0 1 1 0 1 0 0 0 0 0 0 0 0]]  
[[0 0 1 0 0 1 2 0 1 0 0 0 0 0 0 1 0]]  
[[1 0 0 0 0 0 0 0 0 1 1 0 1 0 0 0]]  
-----  
[[1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0]]
```

- 결국 관건은 벡터로 잘 만드는 것과,
- 만들어진 벡터 사이의 거리를 잘 계산하는 것

그럼…

단순히 단어를 카운트하는 것

최금 복잡하게 형태소를 카운트하는 것…

말고… 뭐가 있을까???

tf-idf

文 A 21개 언어 ▾

위키백과, 우리 모두의 백과사전.

TF-IDF(Term Frequency – Inverse Document Frequency)는 [정보 검색](#)과 [텍스트 마이닝](#)에서 이용하는 가중치로, 여러 문서로 이루어진 문서군이 있을 때 어떤 단어가 특정 문서 내에서 얼마나 중요한 것인지를 나타내는 통계적 수치이다. 문서의 [핵심어](#)를 추출하거나, 검색 엔진에서 검색 결과의 순위를 결정하거나, 문서들 사이의 비슷한 정도를 구하는 등의 용도로 사용할 수 있다.

TF(단어 빈도, term frequency)는 특정한 단어가 문서 내에 얼마나 자주 등장하는지를 나타내는 값으로, 이 값이 높을수록 문서에서 중요하다고 생각할 수 있다. 하지만 단어 자체가 문서군 내에서 자주 사용 되는 경우, 이것은 그 단어가 흔하게 등장한다는 것을 의미한다. 이것을 DF(문서 빈도, document frequency)라고 하며, 이 값의 역수를 IDF(역문서 빈도, inverse document frequency)라고 한다. TF-IDF는 TF와 IDF를 곱한 값이다.

IDF 값은 문서군의 성격에 따라 결정된다. 예를 들어 '[원자](#)'라는 낱말은 일반적인 문서들 사이에서는 잘 나오지 않기 때문에 IDF 값이 높아지고 문서의 핵심어가 될 수 있지만, 원자에 대한 문서를 모아놓은 문서군의 경우 이 낱말은 상투어가 되어 각 문서들을 세분화하여 구분할 수 있는 다른 낱말들이 높은 가중치를 얻게 된다.

수학적 설명 [편집]



TF-IDF는 단어 빈도와 역문서 빈도의 곱이다. 두 값을 산출하는 방식에는 여러 가지가 있다. 단어 빈도 $tf(t, d)$ 의 경우, 이 값을 산출하는 가장 간단한 방법은 단순히 문서 내에 나타나는 해당 단어의 총 빈도수를 사용하는 것이다. 문서 d 내에서 단어 t 의 총 빈도를 $f(t, d)$ 라 할 경우, 가장 단순한 tf 산출 방식은 $tf(t, d) = f(t, d)$ 로 표현된다. 그 밖에 TF값을 산출하는 방식에는 다음과 같은 것들이 있다. ^{[1]:118}

- [불린 빈도](#): $tf(t, d) = t$ 가 d 에 한 번이라도 나타나면 1, 아니면 0;
- [로그 스케일 빈도](#): $tf(t, d) = \log(f(t, d) + 1)$;
- 증가 빈도: 최빈 단어를 분모로 target 단어의 TF를 나눈 값으로, 일반적으로는 문서의 길이가 상대적으로 길 경우, 단어 빈도값을 조절하기 위해 사용한다.

$$tf(t, d) = 0.5 + \frac{0.5 \times f(t, d)}{\max\{f(w, d) : w \in d\}}$$

역문서 빈도는 한 단어가 문서 집합 전체에서 얼마나 공통적으로 나타나는지를 나타내는 값이다. 전체 문서의 수를 해당 단어를 포함한 문서의 수로 나눈 뒤 [로그](#)를 취하여 얻을 수 있다.

$$idf(t, D) = \log \frac{|D|}{|\{d \in D : t \in d\}|}$$

출처: <https://ko.wikipedia.org/wiki/Tf-idf>

한 문서에서 많이 등장한 단어에 가중치를
(Term Freq.)

또한 한편으로...

전체 문서에서 많이 나타나는 단어는 중요하지 않게...

(Inverse Document Freq.)

그래서 나타나는 개념 TF-IDF

```
1 from sklearn.feature_extraction.text import TfidfVectorizer
```

[53] ✓ 0.3s

```
1 vectorizer = TfidfVectorizer(min_df=1, decode_error='ignore')
```

[54] ✓ 0.6s

누군가의 말~~~~

C/C++에는 천재가 많고~~~

Python에는 고마운 사람이 많다~~~^^

```
1 X = vectorizer.fit_transform(contents_for_vectorize)
2 num_samples, num_features = X.shape
3 num_samples, num_features
[55] ✓ 0.3s
... (4, 17)
```



1 X.toarray().transpose()

[56]

✓ 0.4s

```
... array([[0.          , 0.          , 0.          , 0.5        ],
       [0.43671931, 0.          , 0.          , 0.          ],
       [0.          , 0.          , 0.39264414, 0.          ],
       [0.34431452, 0.40104275, 0.          , 0.          ],
       [0.          , 0.50867187, 0.          , 0.          ],
       [0.          , 0.          , 0.39264414, 0.          ],
       [0.          , 0.40104275, 0.6191303 , 0.          ],
       [0.34431452, 0.40104275, 0.          , 0.          ],
       [0.          , 0.          , 0.39264414, 0.          ],
       [0.          , 0.50867187, 0.          , 0.          ],
       [0.          , 0.          , 0.          , 0.5        ],
       [0.          , 0.          , 0.          , 0.5        ],
       [0.43671931, 0.          , 0.          , 0.          ],
       [0.          , 0.          , 0.          , 0.5        ],
       [0.43671931, 0.          , 0.          , 0.          ],
       [0.          , 0.          , 0.39264414, 0.          ],
       [0.43671931, 0.          , 0.          , 0.        ]])
```

```
1 new_post_vec = vectorizer.transform(new_post_for_vectorize)
2 new_post_vec.toarray()
[57] ✓ 0.6s
...
array([[0.78528828, 0.          , 0.          , 0.          , 0.          ,
       0.          , 0.          , 0.6191303 , 0.          , 0.          ,
       0.          , 0.          , 0.          , 0.          , 0.          ,
       0.          , 0.        ]])
```

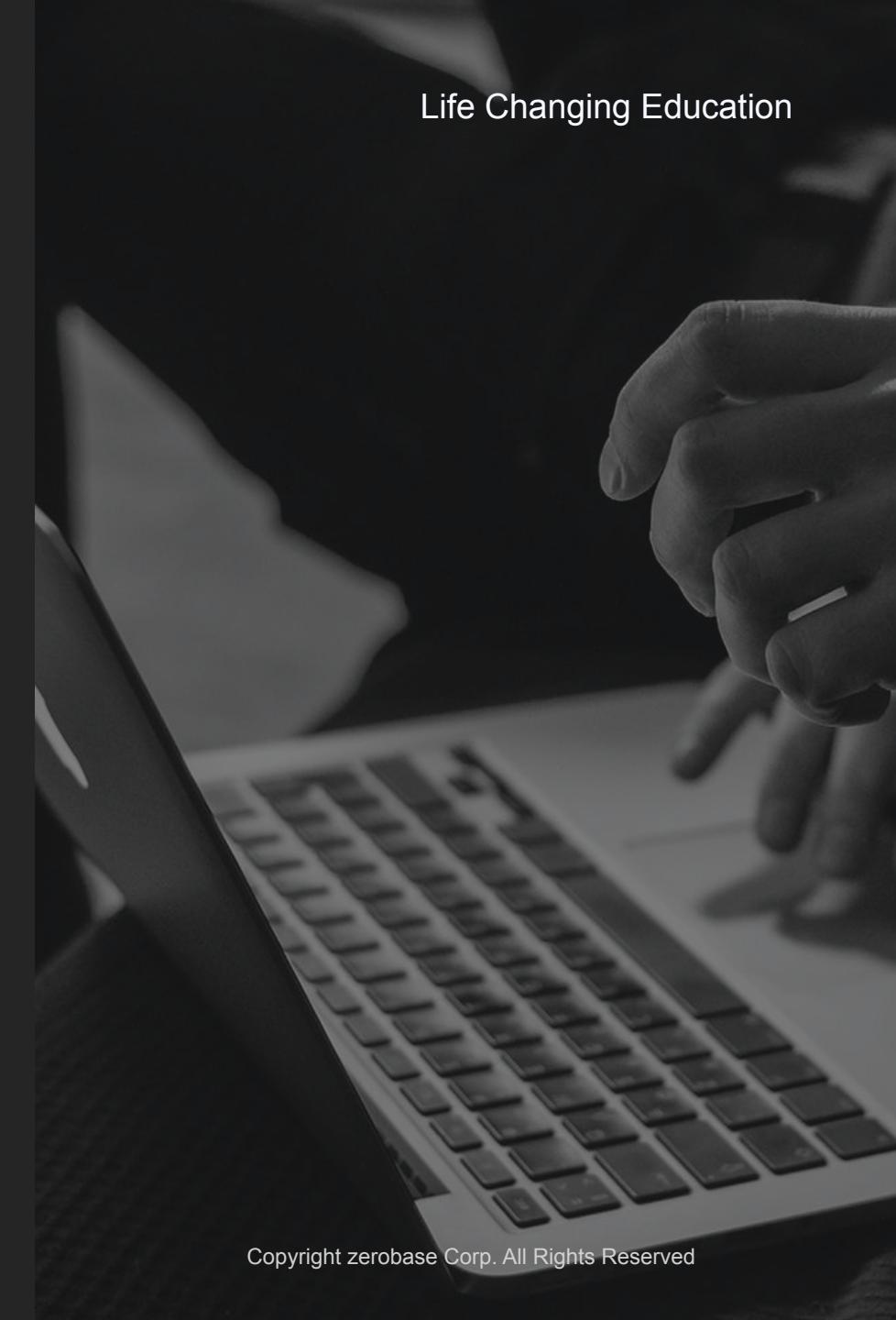
```
1 def dist_norm(v1, v2):
2     v1_normalized = v1 / sp.linalg.norm(v1.toarray())
3     v2_normalized = v2 / sp.linalg.norm(v2.toarray())
4
5     delta = v1_normalized - v2_normalized
6
7     return sp.linalg.norm(delta.toarray())
```

[58] ✓ 0.4s

```
1 dist = [dist_norm(each, new_post_vec) for each in X]
2
3 print('Best post is ', dist.index(min(dist)), ', dist = ', min(dist))
4 print('Test post is --> ', new_post)
5 print('Best dist post is --> ', contents[dist.index(min(dist))])
[59] ✓ 0.7s
```

```
... Best post is 3 , dist = 1.1021396119773588
      Test post is --> ['상처받기 싫어 괜찮아']
      Best dist post is --> 아무 일도 아니야 괜찮아
```

네이버 지식인 검색 결과에서 유사한 문장 찾기



```
1 import urllib.request  
2 import json  
3 import datetime
```

[60]

✓ 0.3s

```
1 def gen_search_url(api_node, search_text, start_num, disp_num):  
2     base = 'https://openapi.naver.com/v1/search'  
3     node = '/' + api_node + '.json'  
4     param_query = '?query=' + urllib.parse.quote(search_text)  
5     param_start = '&start=' + str(start_num)  
6     param_disp = '&display=' + str(disp_num)  
7  
8     return base + node + param_query + param_start + param_disp
```

[61] ✓ 0.3s

```
1 def get_result_onpage(url):  
2     request = urllib.request.Request(url)  
3     request.add_header('X-Naver-Client-Id', client_id)  
4     request.add_header('X-Naver-Client-Secret', client_secret)  
5  
6     response = urllib.request.urlopen(request)  
7  
8     print('[%s] Url Request Success' % datetime.datetime.now())  
9  
10    return json.loads(response.read().decode('utf-8'))
```

[62] ✓ 0.1s

```
1 client_id = 'XML88gKKc5xdQeaU9HF5'  
2 client_secret = 'kWoWWX0Tadaao'  
3  
4 url = gen_search_url('kin', '파이썬', 10, 10)  
5 one_result = get_result_onpage(url)  
6 one_result  
[4] ✓ 0.2s
```

```
1 one_result['items'][0]['description']
```

[11] ✓ 0.2s

... '.... 생겨 파이썬을 독학해보려 합니다. 파이썬은 빅데이터에 유리하다고 들었습니다. 독학을 할 때, 파이썬

```
1 def delete_tag(input_str):
2     input_str = input_str.replace('<b>', '')
3     input_str = input_str.replace('</b>', '')
4
5     return input_str
```

[12] ✓ 0.3s

```
1 def get_description(pages):
2     contents = []
3
4     for sentences in pages['items']:
5         contents.append(delete_tag(sentences['description']))
6
7     return contents
```

[14] ✓ 0.2s

```
1 contents = get_description(one_result)
2 contents
```

```
[15] ✓ 0.6s
```

```
... ['.... 생겨 파이썬을 독학해보려 합니다. 파이썬은 빅데이터에 유리하다고 들었습니다. 독학을 할 때, 파이썬으로.... 그리고, 파이썬을
```

```
1 from sklearn.feature_extraction.text import CountVectorizer
2 from konlpy.tag import Okt
3
4 t = Okt()
5 vectorizer = CountVectorizer(min_df=1)
```

[16] ✓ 0.3s

```
1 contents_tokens = [t.morphs(row) for row in contents]
2 contents_tokens
```

[17] ✓ 4.4s

... [[...,
'생겨',
...], ...]

```
1 contents_for_vectorize = []
2
3 for content in contents_tokens:
4     sentence = ''
5
6     for word in content:
7         sentence = sentence + ' ' + word
8
9     contents_for_vectorize.append(sentence)
10
11 contents_for_vectorize
[18]    ✓ 0.3s
```

... ['... 생겨 파이썬 을 독학 해보려 합니다 . 파이썬 은 빅데이터 에 유리하다고 들었습니다 . 독학 을 할 때 , 파이썬 으로 ...'
 ' [퀴즈 1 , 퀴즈 2 , 퀴즈 3] 이렇게 해서 코드 를 짠 다음 에 퀴즈 1 를 고르면 그 퀴즈 가 나오는 그런 코드 는 파이썬 에

```
1 X = vectorizer.fit_transform(contents_for_vectorize)
2 X
[19] ✓ 0.1s
...
<10x180 sparse matrix of type '<class 'numpy.int64'>'  

     with 211 stored elements in Compressed Sparse Row format>
```

```
1 num_samples, num_features = X.shape
2 num_samples, num_features
[20] ✓ 0.4s
...
(10, 180)
```

vectorize 결과

zero-base /

```
1  vectorizer.get_feature_names()  
[21]  ✓  0.4s  
...  ['16',  
     'apple',  
     'banana',  
     'cherry',  
     'cnt',
```

```
1 new_post = ['파이썬을 배우는데 좋은 방법이 어떤 것인지 추천해주세요']
2 new_post_tokens = [t.morphs(row) for row in new_post]
3
4 new_post_for_vectorize = []
5
6 for content in new_post_tokens:
7     sentence = ''
8     for word in content:
9         sentence = sentence + ' ' + word
10
11     new_post_for_vectorize.append(sentence)
12
13 new_post_for_vectorize
```

[24]

✓ 0.1s

...

['파이썬을 배우는데 좋은 방법이 어떤 것인지 추천해주세요']

vectorize

zero-base /

```
1 import scipy as sp
2
3 def dist_raw(v1, v2):
4     delta = v1 - v2
5     return sp.linalg.norm(delta.toarray())
```

[26] ✓ 0.3s

```
1 dist = [dist_raw(each, new_post_vec) for each in X]
2 dist
```

[27] ✓ 0.5s

```
... [6.928203230275509,
    7.14142842854285,
    5.916079783099616,
    7.0,
    8.246211251235321,
    5.656854249492381,
```

```
1 print('Best post is ', dist.index(min(dist)), ', dist = ', min(dist))
2 print('Test post is --> ', new_post)
3 print('Best dist post is --> ', contents[dist.index(min(dist))])
[28] ✓ 0.2s
...
Best post is 5 , dist = 5.656854249492381
Test post is --> ['파이썬을 배우는데 좋은 방법이 어떤 것인지 추천해주세요']
Best dist post is --> 영상보고 파이썬 다운받고 파이참다운받았는데 ㅠㅠㅠ 뭐가 문제일까요 아무것도 몰라서 자세히 설명...
```

Normalize - 거리를 구하는 또 다른 방법

zero-base /

```
1 def dist_norm(v1, v2):
2     v1_normalized = v1 / sp.linalg.norm(v1.toarray())
3     v2_normalized = v2 / sp.linalg.norm(v2.toarray())
4
5     delta = v1_normalized - v2_normalized
6
7     return sp.linalg.norm(delta.toarray())
```

[29] ✓ 0.4s

```
1 dist = [dist_norm(each, new_post_vec) for each in X]
2
3 print('Best post is ', dist.index(min(dist)), ', dist = ', min(dist))
4 print('Test post is --> ', new_post)
5 print('Best dist post is --> ', contents[dist.index(min(dist))])
```

[30] ✓ 0.5s

... Best post is 0 , dist = 0.8496944740049874

Test post is --> ['파이썬을 배우는데 좋은 방법이 어떤 것인지 추천해주세요']

Best dist post is --> ... 생겨 파이썬을 독학해보려 합니다. 파이썬은 빅데이터에 유리하다고 들었습니다. 독학을 할 때, 파

```
1 def tfidf(t, d, D):
2     tf = float(d.count(t)) / sum(d.count(w) for w in set(d))
3     idf = sp.log(float(len(D)) / (len([doc for doc in D if t in doc])))
4     return tf * idf
```

[31] ✓ 0.4s

```
1 from sklearn.feature_extraction.text import TfidfVectorizer
2 vectorizer = TfidfVectorizer(min_df=1, decode_error='ignore')
```

[32] ✓ 0.5s

```
1 X = vectorizer.fit_transform(contents_for_vectorize)
2 num_samples, num_features = X.shape
3 num_samples, num_features
```

[33] ✓ 0.6s

... (10, 180)

```
▶ 
  1 new_post_vec = vectorizer.transform(new_post_for_vectorize)
  2 new_post_vec.toarray()
```

[34] ✓ 0.4s

```
...
... array([[0.        , 0.        , 0.        , 0.        , 0.        ,
          0.        , 0.        , 0.        , 0.        , 0.        ,
          0.        , 0.        , 0.        , 0.        , 0.        ,
          0.        , 0.        , 0.        , 0.        , 0.        ,
          0.        , 0.        , 0.        , 0.        , 0.        ]])
```

```
1 dist = [dist_norm(each, new_post_vec) for each in X]
2
3 print('Best post is ', dist.index(min(dist)), ', dist = ', min(dist))
4 print('Test post is --> ', new_post)
5 print('Best dist post is --> ', contents[dist.index(min(dist))])
[35] ✓ 0.6s
...
... Best post is 0 , dist = 1.1578296711088993
Test post is --> ['파이썬을 배우는데 좋은 방법이 어떤 것인지 추천해주세요']
Best dist post is --> ... 생겨 파이썬을 독학해보려 합니다. 파이썬은 빅데이터에 유리하다고 들었습니다. 독학을 할 때, 파
```