Text Preprocessing (1-5)

김경환 박은빈 신승배

토큰화

Word Tokenization

→ 단순 구두점이나 특수문자 정제(cleaning) ❌

→ 언어별 토큰화 기준

Time is an illusion. Lunchtime double so!

"Time", "is", "an", "illustion", "Lunchtime", "double", "so"

선택의 순간

```
from nltk.tokenize import WordPunctTokenizer
WordPunctTokenizer().tokenize("Don't be fooled by the dark soun
```

```
['Don',
 "'",
 't',
 'be',
 'fooled',
```

```
from nltk.tokenize import word_tokenize
word_tokenize("Don't be fooled by the dark sounding na
```

```
['Do',
 "n't",
 'be',
 'fooled',
```

```
from tensorflow.keras.preprocessing.text import text_to_wor
text_to_word_sequence("Don't be fooled by the dark soundinc
```

```
["don't",
 'be',
 'fooled',
 'by',
```

단순 제외

구두점 or 특수 문자

- 마침표(.)
- Ph.D
- AT&T
- $45.55
- 01/02/06

줄임말 & 띄어쓰기

아포스트로피 (‘)

New York, rock 'n' roll

Penn Treebank Tokenization

```
from nltk.tokenize import TreebankWordTokenizer

tokenizer=TreebankWordTokenizer()
text="Starting a home-based restaurant may be an ideal. \
it doesn't have a food chain or restaurant of their own."
```

Penn Treebank [cont'd]

```
['Starting',
 'a',
 'home-based',
 'restaurant',
 'may',
 'be',
 'an',
 'ideal.',
 'it',
 'does',
 "n't",
 'have',
 'a',
 'food',
 'chain',
 'or',
 'restaurant',
 'of',
 'their',
 'own',
 '.']
```

Sentence Tokenization

문장 분류(Sentence Segmentation)

토큰의 단위가 문장(sentence)인 경우

Sentence Tokenization [cont'd]

➡️ IP 192.168.56.31 서버에 들어가서 로그 파일 저장해서 ukairia777@gmail.com로 결과 좀 보내줘. 그러고나서 점심 먹으러 가자.

➡️ Since I'm actively looking for Ph.D. students, I get the same question a dozen times every year.

Sentence Tokenization [cont'd]

```
[ ] from nltk.tokenize import sent_tokenize
    text="His barber kept his word. But keeping such a huge secret to himself was driving
    pprint(sent_tokenize(text))
```

```
['His barber kept his word.',
 'But keeping such a huge secret to himself was driving him crazy.',
 'Finally, the barber went up a mountain and almost to the edge of a cliff.',
 'He dug a hole in the midst of some reeds.',
 'He looked about, to make sure no one was near.']
```

```
[ ] from nltk.tokenize import sent_tokenize
    text="I am actively looking for Ph.D. students. and you are a Ph.D student."
    pprint(sent_tokenize(text))
```

```
['I am actively looking for Ph.D. students.', 'and you are a Ph.D student.']
```

```
import kss

text='딥 러닝 자연어 처리가 재미있기는 합니다. 그런데 문제는 영어보다 한국어로 할 때 너무 어려워요. 농담아니에요. 이제 해보면 알걸요?'
pprint(kss.split_sentences(text))
```

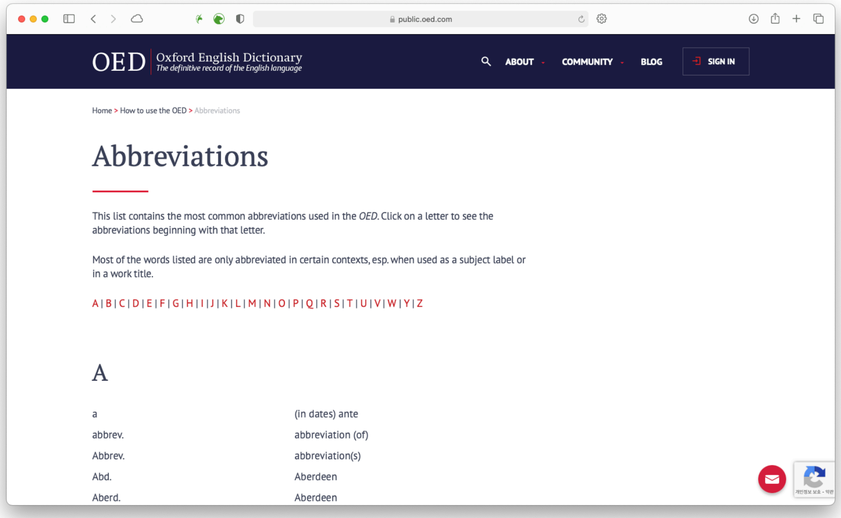
```
['딥 러닝 자연어 처리가 재미있기는 합니다.',
 '그런데 문제는 영어보다 한국어로 할 때 너무 어려워요.',
 '농담아니에요.',
 '이제 해보면 알걸요?']
```

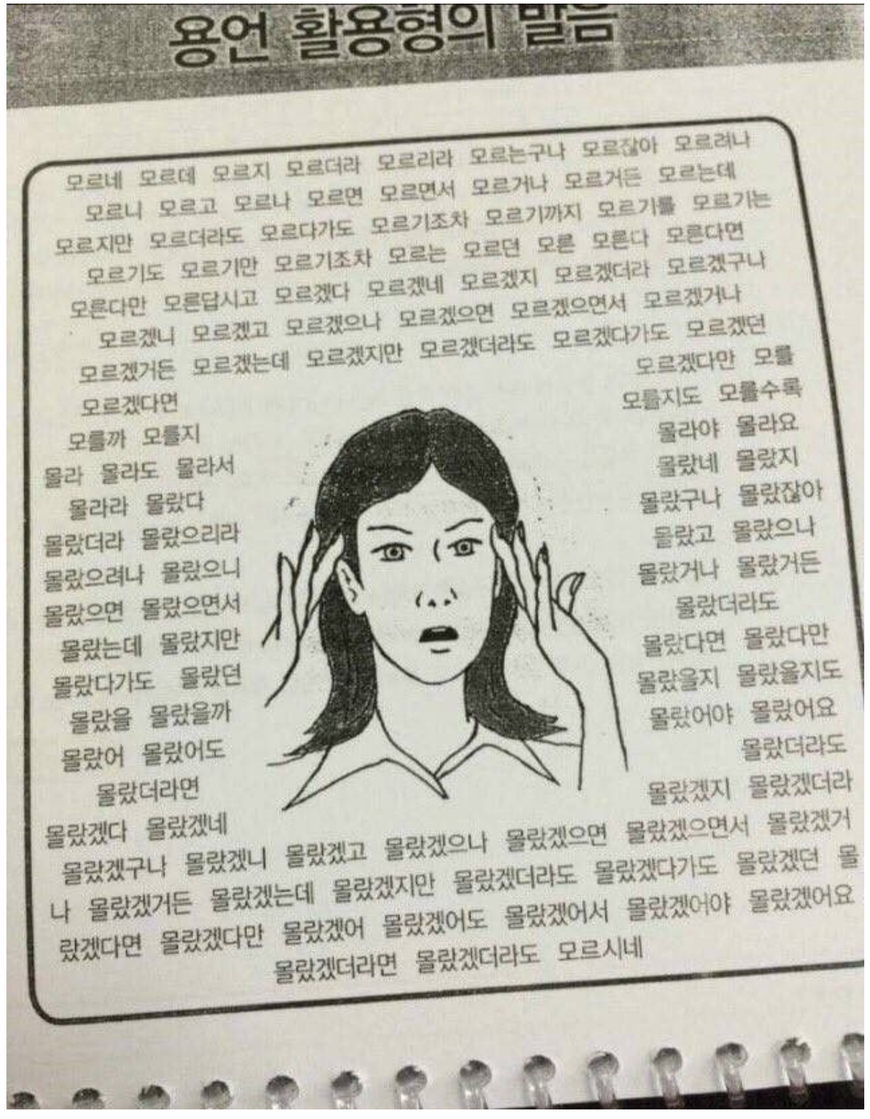
Binary Classifier

1. 마침표(.)가 단어의 일부분일 경우
   → 마침표가 약어(abbreivation)로 쓰이는 경우

2. 마침표(.)가 정말로 문장의 구분자(boundary)일 경우

Abbreviation dictionary ➡️





여기 자리있어요 : 여기 자리 주인있어요
여기 자리있어요 : 빈자리 있어요

학원 끊었어 : 학원을 다니기 시작했다
학원 끊었어 : 학원을 그만 뒀다

커텐 좀 쳐줘 : 커텐 좀 열어줘
커텐 좀 쳐줘 : 커텐 좀 닫아줘

연패 : 연속해서 이기다
연패 : 연속해서 지다
연패 : 연속해서 우승하다

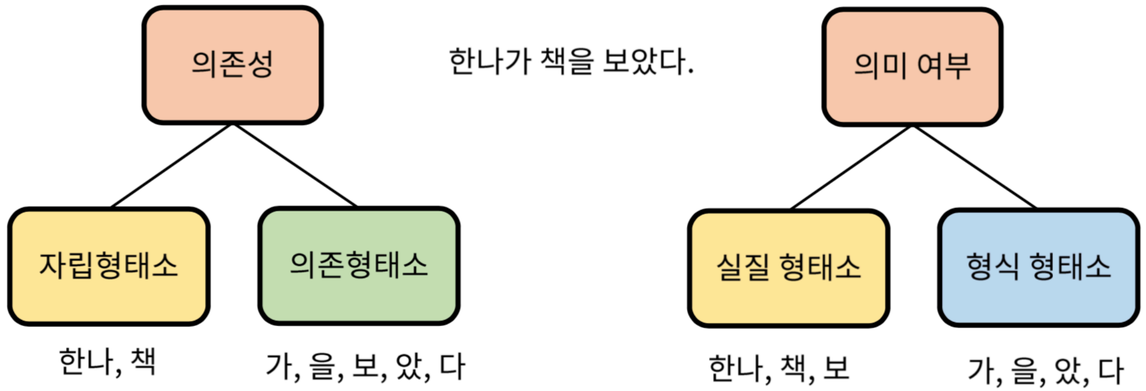
우리 못본지 오래됐네 = 우리 본지 오래됐네
잘하면 망할듯 : 잘했는데 왜 망함?

교착어

https://zomzom.tistory.com/1074

형태소 Morpheme

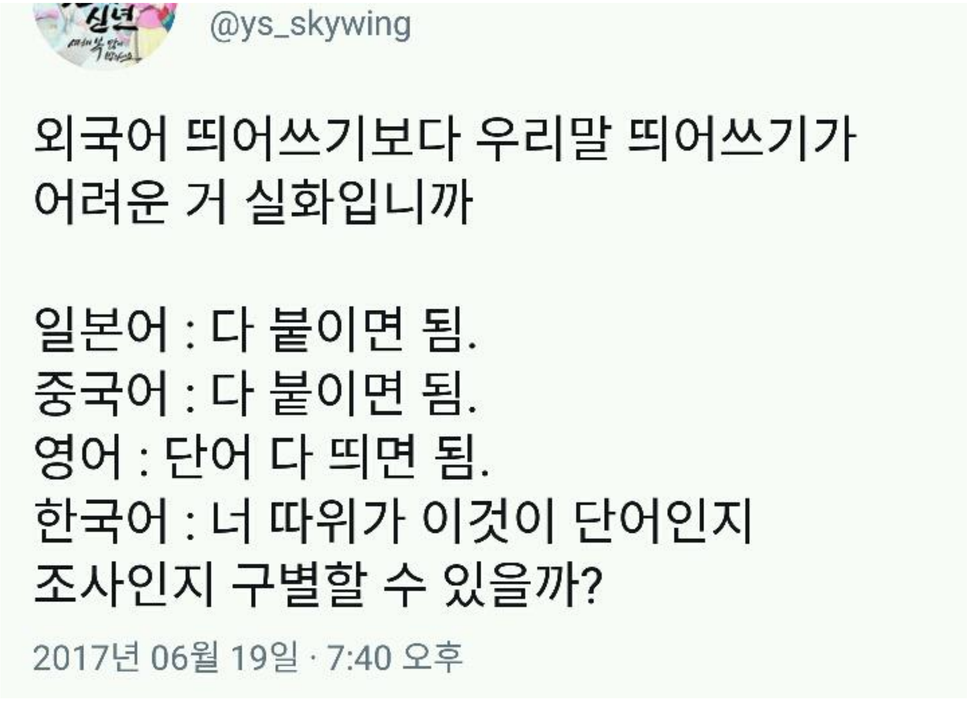
형태소: 언어학적으로 말을 분석할 때, 의미가 있는 가장 작은 말의 단위

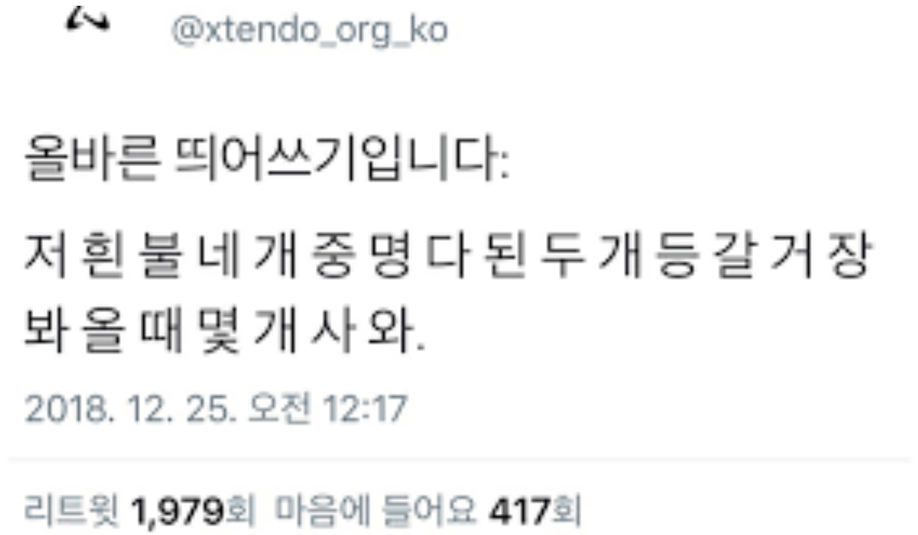


다양한 방법으로 형태소의 종류를 나누어 볼 수 있다.

https://heung-bae-lee.github.io/2020/01/19/NLP_02/

띄어쓰기


@ys_skywing
외국어 띄어쓰기보다 우리말 띄어쓰기가
어려운 거 실화입니까
일본어 : 다 붙이면 됨.
중국어 : 다 붙이면 됨.
영어 : 단어 다 띄면 됨.
한국어 : 너 따위가 이것이 단어인지
조사인지 구별할 수 있을까?
2017년 06월 19일 · 7:40 오후


@xtendo_org_ko
올바른 띄어쓰기입니다:
저 흰 불 네 개 중 명 다 된 두 개 등 갈 거 장
봐 올 때 몇 개 사 와.
2018. 12. 25. 오전 12:17
리트윗 1,979회 마음에 들어요 417회

띄어

chosun.com

朝鮮日報

로그인

문화·라이프 >

前 국립국어원장의 고백 "띄어쓰기, 나도 자신 없다"

[이상규 교수, 국어 규정에 反旗]

-現 띄어쓰기 규정에서는…
海 ①동해(○) 카리브해(×)
고기 ②쇠고기(○) 토끼고기(×)
돈 ③큰돈(○) 작은돈(×)

"복잡한 띄어쓰기·사이시옷… 우리말 환경 더 위축시켜"

유석재 기자

입력 2013.05.22 03:02

"'불어(佛語)'는 붙여 쓰는 것이 맞고, '프랑스 어'는 띄어 쓰는 것이 맞게 돼 있습니다. 똑같은 대상을 가리키는 말이 한 단어가 됐다가 두 단어가 되기도 합니다. 한국말은 어렵다는 인식을 가져옵니다."

전직 국립국어원장이 "복잡하기 짝이 없는 현행 띄어쓰기 규정이 국어를 망친다"고 주장하고 나섰다. 2006~2009년 노무현·이명박 두 정부에 걸쳐 국립국어원장을 지낸 이상규(60·**사진**) 경북대 국문과 교수다. 이 교수는 오는 25일 한글학회 주최로 서울 종로구 한글회관에서 열리는 '616돌

많이 본 뉴스

1 "시원하게 함 해봐라" 조폭 한마디에 주차장은 피로 물들었다

띄어쓰기 [cont'd]

제가이렇게띄어쓰기를전혀하지않고글을썼다고하더라도글을이해할수있습니다.

띄어쓰기 [cont'd]

Tobeornottobethatisthequestion

품사 태깅 Part-of-Speech

```
from nltk.tag import pos_tag

x = word_tokenize(text)

pprint(pos_tag(x))
```

```
[('I', 'PRP'),
 ('am', 'VBP'),
 ('actively', 'RB'),
 ('looking', 'VBG'),
 ('for', 'IN'),
 ('Ph.D.', 'NNP'),
 ('students', 'NNS'),
 ('.', '.'),
 ('and', 'CC'),
 ('you', 'PRP'),
 ('are', 'VBP'),
 ('a', 'DT'),
 ('Ph.D.', 'NNP'),
 ('student', 'NN'),
 ('.', '.')]
```

```
pprint(okt.pos("열심히 코딩한 당신, 연휴에는 여행을 가봐요"))
```

```
[('열심히', 'Adverb'),
 ('코딩', 'Noun'),
 ('한', 'Josa'),
 ('당신', 'Noun'),
 (',', 'Punctuation'),
 ('연휴', 'Noun'),
 ('에는', 'Josa'),
 ('여행', 'Noun'),
 ('을', 'Josa'),
 ('가봐요', 'Verb')]
```

정제 및 정규화

정제 및 정규화

이음동의어의 정규화

➡️ USA / US

대, 소문자 통합

➡️ US / us

➡️ General Motors

불필요한 단어 제거

등장 빈도수 낮은 단어

너무 적게 등장해서
도움이 되지 않는 단어

길이가 짧은 단어

영어 단어 평균 길이 6~7
한국어 단어 평균 길이 2~3

배울 학(學) / 학교 교(校) → s, c, h, o, o, l
용(龍) → d, r, a, g, o, n

길이가 짧은 단어 [cont'd]

```
import re
text = "I was wondering if anyone out there could enlighten me on this car."
shortword = re.compile(r'\W*\b\w{1,2}\b')
print(shortword.sub('', text))
```

```
 was wondering anyone out there could enlighten this car.
```

어간 및 표제어 추출과 불용어

표제어 추출 (Lemmatization)

표제어 : 기본 사전형 단어

표제어 추출: 단어들이 다른 형태를 가지더라도, 그 뿌리 단어를 찾아가서 단어의 개수를 줄일 수 있는지 판단

문맥을 고려하며, 수행했을 때의 결과는 해당 단어의 품사 정보(pos 태그)를 보존

ex) am, are, is 뿌리 단어는 be

가장 섬세한 방법는 단어의 형태학적 파싱(형태소 분석)을 먼저 진행하는 것

1)어간(stem) : 단어의 의미를 담고 있는 핵심 부분

2)접사 (affix) : 단어에 추가적인 의미를 주는 부분

* 분리되지않는 독립적인 형태소도 존재

표제어 추출 [cont'd]

NLTK 에서는 표제어 추출을 위한 도구인 WordNetLemmatizer를 지원

```
from nltk.stem import WordNetLemmatizer

n = WordNetLemmatizer()
words = ['policy', 'doing', 'organization', 'have', 'going', 'love', 'lives',
'fly', 'dies', 'watched', 'has', 'starting']

print([n.lemmatize(w) for w in words])
```

```
['policy', 'doing', 'organization', 'have', 'going', 'love', 'life', 'fly', 'dy', 'watched', 'ha', 'starting']
```

표제어 추출 [cont'd]

본래 단어의 품사 정보를 입력하면 정확한 표제어를 추출할 수 있다

```
n.lemmatize('dies', 'v')
```

```
'die'
```

```
n.lemmatize('watched', 'v')
```

```
'watch'
```

```
n.lemmatize('has', 'v')
```

```
'have'
```

어간 추출 (Stemming)

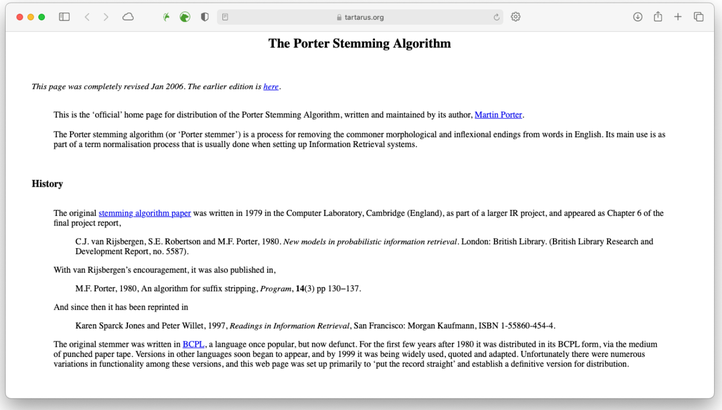
: 정해진 규칙만 보고 단어의 어미를 자르는 어림짐작의 작업

섬세한 작업이 아니기에 결과가 사전 부재 단어 가능성 존재

어간 추출을 수행한 결과는 품사 정보가 보존되지(POS 태그를 고려하지) 않는다

※ Porter 알고리즘의 상세 규칙 ➡️

https://tartarus.org/martin/PorterStemmer/



어간 추출 [cont'd]

포터 알고리즘의 어간 추출 규칙

- ~ALIZE → AL ,
- ~ANCE → 제거,
- ~ICAL → IC

```
words = ['formalize', 'allowance', 'electricical']
print([s.stem(w) for w in words])
```

➡️

```
['formal', 'allow', 'electric']
```

어간 추출 [cont'd]

```
from nltk.stem import PorterStemmer
from nltk.tokenize import word_tokenize

s = PorterStemmer()
text = "This was not the map we found in Billy Bones's chest, but an accurate
copy, complete in all things--names and heights and soundings--with the single
exception of the red crosses and the written notes."

words = word_tokenize(text)
print(words)
```

```
['This', 'was', 'not', 'the', 'map', 'we', 'found', 'in', 'Billy', 'Bones', "'s", 'chest', ',', 'but', 'an', 'accurate', 'copy', ',',
'complete', 'in', 'all', 'things', '--', 'names', 'and', 'heights', 'and', 'soundings', '--', 'with', 'the', 'single', 'exception', 'of',
'the', 'red', 'crosses', 'and', 'the', 'written', 'notes', '.']
```

```
print([s.stem(w) for w in words])
```

```
['thi', 'wa', 'not', 'the', 'map', 'we', 'found', 'in', 'billi', 'bone', "'s", 'chest', ',', 'but', 'an', 'accur', 'copi', ',',
'complet', 'in', 'all', 'thing', '--', 'name', 'and', 'height', 'and', 'sound', '--', 'with', 'the', 'singl', 'except', 'of', 'the',
'red', 'cross', 'and', 'the', 'written', 'note', '.']
```

어간 추출 [cont'd] - Algorithm Comparison

포터 알고리즘(Porter Algorithm) & 랭커스터 알고리즘(Lancaster Algorithm)

동일한 단어의 나열에도 알고리즘에 따른 다른 결과 도출

```
from nltk.stem import PorterStemmer

s = PorterStemmer()
words = ['policy', 'doing', 'organization', 'have', 'going', 'love', 'lives', 'fly', 'dies', 'watched',
       'has', 'starting']

print([s.stem(w) for w in words])
```

```
from nltk.stem import LancasterStemmer

l = LancasterStemmer()
words = ['policy', 'doing', 'organization', 'have', 'going', 'love', 'lives', 'fly', 'dies', 'watched',
    'has', 'starting']

print([l.stem(w) for w in words])
```

어간 추출 [cont'd] - Algorithm Comparison

포터 알고리즘(Porter Algorithm) & 랭커스터 알고리즘(Lancaster Algorithm)

동일한 단어의 나열에도 알고리즘에 따른 다른 결과 도출

➡️ 알고리즘에 따른 사용자의 판단 필요

```
from nltk.stem import PorterStemmer
```

```
['polici', 'do', 'organ', 'have', 'go', 'love', 'live', 'fli', 'die', 'watch', 'ha', 'start']
```

```
from nltk.stem import LancasterStemmer
```

```
['policy', 'doing', 'org', 'hav', 'going', 'lov', 'liv', 'fly', 'die', 'watch', 'has', 'start']
```

표제어 및 어간 추출 특징 비교

표제어 추출(Lemmatization)

am → am

the going → the go

having → hav

어간 추출 (Stemming)

am → be

the going → the going

having → have

한국어에서의 어간 추출

5언 9품사 구조

언	품사
체언	명사, 대명사, 수사
수식언	관형사, 부사
관계언	조사
독립언	감탄사
용언	동사, 형용사

활용 : 용언의 어간(stem)이 어미(ending)를 가지는 일

- 어간(stem): 용언(동사, 형용사)을 활용할 때, 원칙적으로 모양이 변하지 않는 부분
- 어미(ending): 용언의 어간 뒤에 붙어서 활용하면서 변하는 부분

한국어 어간 추출 [cont'd]

규칙 활용 : 어간이 어미를 취할 때, 어간의 모습이 일정

→ 어간과 어미가 합쳐질 때, 어간의 형태 불변

```
잡/어간 + 다/어미
```

불규칙 활용 : 어간이 어미를 취할 때 어간의 모습이 바뀌거나, 취하는 어미가 특수한 어미인 경우

→ 어간의 모습이 바뀐경우

```
듣-, 돕-, 곱-, 잇-, 오르-, 노랗-' 등이 '듣/들-, 돕/도우-, 곱/고우-, 잇/이-, 올/올-, 노랗/노라-
```

→ 취하는 어미가 특수한 어미일 경우

```
오르+ 아/어→올라, 하+아/어→하여, 이르+아/어→이르러, 푸르+아/어→푸르러
```

➡️ 단순한 분리만으로 어간 추출이 되지 않고 좀 더 복잡한 규칙을 필요로 한다

한국어 표제어/어간추출 예시

KoNLPy : 대부분의 한글 형태소 분석기를 지원

```
! git clone https://github.com/SOMJANG/Mecab-ko-for-Google-Colab.git
! pip3 install konlpy
! bash install_mecab-ko_on_colab190912.sh
```

```
from konlpy.tag import Hannanum, Kkma, Okt, Komoran, Mecab, Twitter

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

Khaiii : 딥러닝 기반 형태소 분석기

한국어 표제어/어간추출 예시

Khaiii : 딥러닝 기반 형태소 분석기

```
!git clone https://github.com/kakao/khaiii.git
!pip install cmake
!mkdir build
!cd build && cmake /content/khaiii
!cd /content/build/ && make all
!cd /content/build/ && make resource
!cd /content/build && make install
!cd /content/build && make package_python
!pip install /content/build/package_python
```

한국어 표제어/어간추출 예시 [cont'd]

아버지가방에들어가신다

```
hannanum
[('아버지가방에들어가', 'N'), ('이', 'J'), ('시ㄴ다', 'E')]
kkma
[('아버지', 'NNG'), ('가방', 'NNG'), ('에', 'JKM'), ('들어가', 'VV'), ('시', 'EPH'), ('ㄴ다', 'EFN')]
okt
[('아버지', 'Noun'), ('가방', 'Noun'), ('에', 'Josa'), ('들어가신다', 'Verb')]
komoran
[('아버지', 'NNG'), ('가방', 'NNP'), ('에', 'JKB'), ('들어가', 'VV'), ('시', 'EP'), ('ㄴ다', 'EC')]
Mecab
[('아버지', 'NNG'), ('가', 'JKS'), ('방', 'NNG'), ('에', 'JKB'), ('들어가', 'VV'), ('신다','EP+EC')]
khaiii
아버지/NNG + 가/JKS + 방/NNG + 에/JKB + 들어가/VV + 시/EP + ㄴ다/EC
```

한국어 표제어/어간추출 예시 [cont'd]

삼성전자 10만 원 가자

```
hannanum
[('삼성전자', 'N'), ('10', 'N'), ('만', 'J'), ('원', 'N'), ('가', 'P'), ('자', 'E')]
kkma
[('삼성전자', 'NNG'), ('10', 'NR'), ('만', 'NR'), ('원', 'NNM'), ('가자', 'NNG')]
okt
[('삼성', 'Noun'), ('전자', 'Noun'), ('10만', 'Number'), ('원', 'Noun'), ('가자', 'Verb')]
komoran
[('삼성전자', 'NNP'), ('10', 'SN'), ('만', 'NR'), ('원', 'NNB'), ('가자', 'NNP')]
Mecab
[('삼성전자', 'NNP'), ('10', 'SN'), ('만', 'NR'), ('원', 'NNBC'), ('가', 'VV'), ('자', 'EC')]
khaiii
삼성전자/NNP + 10/SN + 만/NR + 원/NNB + 가/NNG + 자/EC
```

불용어 (Stopword)

불용어 :자주 등장은 하지만, 큰 의미가 없는 단어

NLTK를 통해서 불용어 제거하기

→ 100여개 이상의 영어 단어들을 불용어로 패키지 내에서 미리 정의하고 있음

```
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

example = "Family is not an important thing. It's everything."
stop_words = set(stopwords.words('english'))
word_tokens = word_tokenize(example)
result = []

for w in word_tokens:
    if w not in stop_words:
        result.append(w)

print(word_tokens)
print(result)
```

```
['Family', 'is', 'not', 'an', 'important', 'thing', '.', 'It', "'s", 'everything', '.'] ['Family', 'important', 'thing', '.', 'It', "'s", 'everything', '.']
```

한국어에서 불용어 제거하기

토큰화 후에 조사, 접속사 등을 제거하는 방법이 있으나
사용자가 직접 불용어 사전을 만들어 제거하는 경우가 많다

```
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

example = "고기를 아무렇게나 구우려고 하면 안 돼. 고기라고 다 같은 게 아니거든. 예컨대 삼겹살을 구울 때는 중요한 게 있지."
stop_words = "아무거나 아무렇게나 어찌하든지 같다 비슷하다 예컨대 이럴정도로 하면 아니거든"

stop_words = stop_words.split(' ‘)
word_tokens = word_tokenize(example)

result = [word for word in word_tokens if not word in stop_words]

print(word_tokens)
print(result)
```

```
['고기를', '아무렇게나', '구우려고', '하면', '안', '돼', '.', '고기라고', '다', '같은', '게', '아니거든', '.', '예컨대',
'삼겹살을', '구울', '때는', '중요한', '게', '있지', '.']

[‘고기를', '구우려고', '안', '돼', '.', '고기라고', '다', '같은', '게', '.', '삼겹살을', '구울', '때는', '중요한', '게', '있지', '.']
```

한국어 불용어 리스트 ➡️ https://www.ranks.nl/stopwords/korean

정규표현식

특수문자

.	한 개의 임의의 문자를 나타냅니다. (줄바꿈 문자인 ₩n는 제외)
?	앞의 문자가 존재할 수도 있고, 존재하지 않을 수도 있습니다. (문자가 0개 또는 1개)
*	앞의 문자가 무한개로 존재할 수도 있고, 존재하지 않을 수도 있습니다. (문자가 0개 이상)
+	앞의 문자가 최소 한 개 이상 존재합니다. (문자가 1개 이상)
^	뒤의 문자로 문자열이 시작됩니다.
$	앞의 문자로 문자열이 끝납니다.
{숫자}	숫자만큼 반복합니다.
{숫자1, 숫자2}	숫자1 이상 숫자2 이하만큼 반복합니다. ?, *, +를 이것으로 대체할 수 있습니다.
{숫자,}	숫자 이상만큼 반복합니다.
[ ]	대괄호 안의 문자들 중 한 개의 문자와 매치합니다.
[^문자]	해당 문자를 제외한 문자를 매치합니다.
|	A|B와 같이 쓰이며 A 또는 B의 의미를 가집니다.

[amk]라고 한다면 a 또는 m 또는 k 중 하나라도 존재하면 매치를 의미합니다.
[a-z]와 같이 범위를 지정할 수도 있습니다.
[a-zA-Z]는 알파벳 전체를 의미하는 범위이며, 문자열에 알파벳이 존재하면 매치를 의미합니다.

문자 규칙

₩	역 슬래쉬 문자 자체를 의미합니다
₩d	모든 숫자를 의미합니다. [0-9]와 의미가 동일합니다.
₩D	숫자를 제외한 모든 문자를 의미합니다. [^0-9]와 의미가 동일합니다.
₩s	공백을 의미합니다. [₩t₩n₩r₩f₩v]와 의미가 동일합니다.
₩S	공백을 제외한 문자를 의미합니다. [^ ₩t₩n₩r₩f₩v]와 의미가 동일합니다.
₩w	문자 또는 숫자를 의미합니다. [a-zA-Z0-9]와 의미가 동일합니다.
₩W	문자 또는 숫자가 아닌 문자를 의미합니다. [^a-zA-Z0-9]와 의미가 동일합니다.

모듈 함수

re.compile( )	정규표현식을 컴파일하는 함수입니다. 다시 말해, 파이썬에게 전해주는 역할을 합니다. 찾고자 하는 패턴이 빈번한 경우에는 미리 컴파일해놓고 사용하면 속도와 편의성면에서 유리합니다.
re.search( )	문자열 전체에 대해서 정규표현식과 매치되는지를 검색합니다.
re.match( )	문자열의 처음이 정규표현식과 매치되는지를 검색합니다.
re.split( )	정규 표현식을 기준으로 문자열을 분리하여 리스트로 리턴합니다.
re.findall( )	문자열에서 정규 표현식과 매치되는 모든 경우의 문자열을 찾아서 리스트로 리턴합니다. 만약, 매치되는 문자열이 없다면 빈 리스트가 리턴됩니다.
re.finditer( )	문자열에서 정규 표현식과 매치되는 모든 경우의 문자열에 대한 이터레이터 객체를 리턴합니다.
re.sub( )	문자열에서 정규 표현식과 일치하는 부분에 대해서 다른 문자열로 대체합니다.

전처리 예제

단어 분리

'₩s+'는 공백, 뒤에 붙는 +는 최소 1개 이상의 패턴을 찾아낸다는 의미

split 는 주어진 정규표현식을 기준으로 분리하는 메소드

```
import re
text = """100 John PROF 101 James STUD 102 Mac STUD"""
re.split('\s+', text)
```

```
['100', 'John', 'PROF', '101', 'James', 'STUD', '102', 'Mac', 'STUD']
```

숫자 검출

₩d+ 는 숫자1개이상을 의미

findall 은 해당 정규표현식에 일치하는 값을 모두 찾아내는 메소드

```
re.findall('\d+',text)
```

➡️

```
['100', '101', '102]
```

영어 4글자 단어 추출

[A-Z]{4} 는 영어 대문자 4개를 의미하는 정규표현식

findall 은 해당 정규표현식에 일치하는 값을 모두 찾아내는 메소드

```
re.findall('[A-Z]{4}',text)
```

➡️

```
['PROF', 'STUD', 'STUD']
```

이름 추출

이름(고유명사)은 첫 문자가 대문자 이고, 그후에 소문자가 여러 번 등장

```
text = """100 John PROF 101 James STUD 102 Mac STUD"""
re.findall('[A-Z][a-z]+',text)
```

```
['John', 'James', 'Mac']
```

정규표현식을 이용한 토큰화

토큰화 - 정규 표현식에 매칭되는 문자열 기준

- NLTK에서는 정규 표현식을 사용해서 단어 토큰화를 수행하는 RegexpTokenizer를 지원
- RegexpTokenizer()에서 괄호 안에 원하는 정규 표현식을 넣어서 토큰화를 수행

```
import nltk from nltk.tokenize
import RegexpTokenizer

tokenizer=RegexpTokenizer("[\w]+")

tokenizer.tokenize("Don't be fooled by the dark sounding name, Mr. Jone's
Orphanage is as cheery as cheery goes for a pastry shop")
```

문자 또는 숫자 1개 이상

```
['Don', 't', 'be', 'fooled', 'by', 'the', 'dark', 'sounding', 'name', 'Mr', 'Jone', 's', 'Orphanage', 'is', 'as', 'cheery', 'as', 'cheery', 'goes', 'for', 'a', 'pastry', 'shop']
```

문장에서 구두점이 제외하고 단어들 만을 가지고 토큰화를 수행
따라서, Don't 가 Don 과 t 로 분리

토큰화 [cont'd]

- 정규 표현식에 매칭되는 문자열 분리 기준

```
import nltk from nltk.tokenize
import RegexpTokenizer

tokenizer=RegexpTokenizer("[\s]+", gaps=True)   공백 1개 이상을 의미

tokenizer.tokenize("Don't be fooled by the dark sounding name, Mr. Jone's
Orphanage is as cheery as cheery goes for a pastry shop")
```

```
["Don't", 'be', 'fooled', 'by', 'the', 'dark', 'sounding', 'name,', 'Mr.', "Jone's", 'Orphanage', 'is', 'as', 'cheery', 'as', 'cheery', 'goes', 'for', 'a', 'pastry', 'shop']
```

토큰화 [cont'd]

- 한국어, 한글

```
import nltk from nltk.tokenize
import RegexpTokenizer tokenizer=RegexpTokenizer("[가-힣]+")

tokenizer.tokenize("""I'm on the next level 저 너머의 문을 열어 Next level 널 결국
엔 내가 부셔 Next level Kosmo에 닿을 때까지 Next level 제껴라 제껴라 제껴라""")
```

```
['저', '너머의', '문을', '열어', '널', '결국엔', '내가', '부셔', '에', '닿을', '때까지', '제껴라', '제껴라', '제껴라']
```

추가 사례

특수문자 제거

regex은 한글 영어, 구두점 기타 괄호 등을 포함하여

의미가 있어보이는 문자열이 아닌 정규표현식 (^ 주의)

```
import re
regex = "[^가-힣|A-Za-z0-9| \\.| \\! | \\?| \\'| \\\"| \\(| \\)| \\[| \\]| \\{| \\}| \\+ | \\~ ]+"
text = "이모지 😊 특수문자  # ☆ 제외하기"
re.sub(regex, text)
```

‘이모지 특수문자 제외하기’

불필요한 영문자 제거

"([A-za-z0-0]){10,}" 10자 이상의 영어,숫자로 이루어진 문자열을 의미

대부분의 경우 이런경우 의미가 있는 문자열이 아님

```
import re
regex = "([A-za-z0-0]){10,}"
text = "배송 빨라서 좋아요 ajklsdhajksdhajkasdassdhjk"
re.sub(regex, text)
```

```
'배송 빨라서 좋아요'
```

반복 문자열 처리

"(.{1,}?)₩₩1{1,}"

- (.{1,}?) : 1개 이상 문자열 혹은 0개
- ₩₩1{1,} : 첫번째 매칭된 그룹[(.{1,}?) 를 의미]이 1번이상 반복

➡️ 같은 문자열(예제에서는 좋아요)가 1번이상 반복되는 경우

```
import re
text = "좋아요좋아요좋아요좋아요좋아요"

while re.search("(.{1,}?)\\1{1,}",  text):
  text = re.sub("(.{1,}?)(\\1{1,})", "\\1", text)
  print(str)
```

좋아요좋아요좋아요좋아요좋아요 ➔ 좋아요

```
좋아요
```

반복 문자열 처리 [cont'd]

"(.{2,})₩₩s{1,}₩₩1"

- (.{2,}) : 2자 이상의 모든 문자열
- ₩₩s{1,} : 1개 이상의 공백 문자열
- ₩₩1 : 첫번째 매칭된 그룹 (여기에서는 (.{2,}) 를 의미)

➡️ 공백으로 분리된 문자열이 반복되는 경우

```
import re
text = "좋아요 좋아요 좋아요 좋아요 좋아요"

while re.search("(.{2,})\\s{1,}\\1",  text):
  text = re.sub("(.{2,})\\s{1,}\\1", "\\1", text)
  print(str)
```

1번째 매칭 : 좋아요 좋아요 좋아요 좋아요 좋아요 ➔ 좋아요 좋아요

2번째 매칭 : 좋아요 좋아요 좋아요 ➔ 좋아요 좋아요

3번째 매칭 : 좋아요 좋아요 ➔ 좋아요

```
좋아요 좋아요 좋아요
좋아요 좋아요
좋아요
```