Lecture 4: 텍스트 전처리(Text Preprocessing)

텍스트 전처리(Text Preprocessing)

> 텍스트 전처리는 풀고자 하는 문제의 용도에 맞게 텍스트를 사전에 처리하는 작업
> 텍스트 전처리를 제대로 하지 않으면 자연어 처리 기법들이 제대로 동작하지 않음

자연어 전처리 과정

- 목적
  - 문서/문장을 의미가 있는 최소 단위로 나누고 분석
- Tokenize
  - 대상이 되는 문서/문장을 최소 단위로 나눔
- Cleaning and Normalization
  - 최소 단위를 표준화
- POS-tagging
  - 최소 의미단위로 나누어진 대상에 대해 품사를 부착
- Chunking
  - POS-tagging의 결과를 명사구, 형용사구, 분사구 등과 같은 말모듬으로 다시 합치는 과정

용어 정의

- 문서 (Document)
  - 한 덩어리의 텍스트로서 짧은 문장에서부터 긴 문서까지를 모두 포함하는 의미
  - 문서의 집합을 말뭉치 (corpus)라고 함
  - 문서의 레벨에 따라 말뭉치의 레벨이 바뀔 수 있음(문장, 문단, 페이지, 댓글 등)
- 어휘사전 (Lexicon)
  - 어휘(lexical)의 집합 또는 어휘에 대한 정의 혹은 설명을 가진 사전
  - 특징 별 어휘사전이 나뉘어서 존재하기도 함(인물사전, 영어사전, 건물사전 등)

용어 정의

- 불용어 (Stop-word)
  - 텍스트 분석에 있어서, 또는 분석결과에 출현하더라도 아무런 의미가 없는 단어의 집합
  - 정보전달 보다는 주로 기능적인 역할을 하는 단어에 해당함
    - 한국어 예 : 그거, 여기, 이제, 은, 는, 이, ...
    - 영어 예 : a, an, the, of, the, ...
  - 빈출어 (Common-word)
    - 너무 많이 출현하여 분석 결과에서 의미 또는 중요도가 떨어지는 단어의 집합
    - 예 : 기사, 기자, 제목, 사진, 네이버, 검색, 보다, 연기, 평점, 공감, 비공감
- 형태소 (Morpheme) ▸ 뜻을 가진 가장 작은 말의 단위
  - 동사, 명사, 조사, 문장부호 등 보통 품사 (Part of Speech, POS) 단위를 의미함

용어 정의

- 단어 주머니 (Bag of Words, BoW)
  - 문서에 함께 사용된 단어의 집합
  - 중복된 단어는 하나로 취급하며, 순서의 의미를 고려하지 않음
    - 예 : “아버지가 방에 들어가신다.” → [“아버지”, “방”, “들어가다”]
- 토큰화 (Tokenization)
  - 토큰 (token)
    - 유용한 의미적 단위로 함께 모여지는 일련의 문자열
    - 구분 기호 사이의 글자 시퀀스
  - 문헌 단위의 문자열이 주어졌을 때 토큰들로 문자열을 분리하는 작업
  - 구두점 등 불필요한 글자들을 제외하기도 함
  - 영어는 언어학적 특성상 단어에 조사가 붙지 않아 한글보다 토큰화가 쉬움

토큰화(Tokenization)

- Tokenize
  - Document를 Sentence의 집합으로 분리
  - Sentence를 Word의 집합으로 분리
  - 의미 없는 문자 등을 걸러 냄
- 영어 vs. 한글
  - 영어는 공백(space) 기준으로 비교적 쉽게 tokenize 가능
  - 한글은 구조상 형태소(morpheme) 분석이 필요
  - 복합명사, 조사, 어미 등을 분리해내는 작업이 필요
  - 영어에 비해 어렵고 정확도 낮음

단어 토큰화(Word Tokenization)

- 토큰의 기준을 단어(word)로 하는 경우
- 단어(word)는 단어 단위 외에도 단어구, 의미를 갖는 문자열로도 포함

> ex) 구두점(punctuation)과 같은 문자는 제외시키는 간단한 단어 토큰화 (구두점: 마침표(.), 컴마(,), 물음표(?), 세미콜론(;), 느낌표(!) 등의 기호)

```
입력: Time is an illusion. Lunchtime double so!
출력 : "Time", "is", "an", "illustion", "Lunchtime", "double", "so"
```

- 구두점을 지운 뒤에 띄어쓰기(whitespace)를 기준으로 구분
- 띄어쓰기 단위로 자르면 사실상 단어 토큰이 구분되는 영어와 달리, 한국어는 띄어쓰기만으로는 단어 토큰을 구분하기 어려움

WordPunctTokenizer

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

> 아포스트로피가 들어간 상황에서 Don't와 Jone's는 어떻게 토큰화할 수 있을까?

```
from nltk.tokenize import word_tokenize
from nltk.tokenize import WordPunctTokenizer

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

```
print(word_tokenize(text))
```

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

```
print(WordPunctTokenizer().tokenize(text))
```

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

토큰화 고려사항

- 구두점이나 특수 문자를 단순 제외해서는 안됨
  - 단어 자체에 구두점을 갖고 있는 경우: `m.p.h`, `Ph.D`, `AT&T`
  - 가격: `$45.55`
  - 날짜: `01/02/06`
  - 수치 표현: `123,456,789`와 같이 세 자리 단위로 컴마
- 줄임말과 단어 내에 띄어쓰기가 있는 경우
  - `we're`는 we are의 줄임말 (re를 접어(clitic)이라고 함)
  - `New York`, `rock 'n' roll`

표준 토큰화 예제

Penn Treebank Tokenization의 규칙

- 규칙 1. 하이푼으로 구성된 단어는 하나로 유지한다.
- 규칙 2. doesn't와 같이 아포스트로피로 '접어'가 함께하는 단어는 분리해준다.

> "Starting a home-based restaurant may be an ideal. it doesn't have a food chain or restaurant of their own."

```
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."
print(tokenizer.tokenize(text))
```

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

문장 토큰화(Sentence Tokenization)

코퍼스 내에서 문장 단위로 구분하는 작업으로 때로는 문장 분류(sentence segmentation) 또는 문장경계인식(sentence boundary detection)이라고도 함
어떻게 주어진 코퍼스로부터 문장 단위로 분류할 수 있을까?

- ?나 마침표(.)나 ! 기준으로 문장 구분
- !나 ?는 문장의 구분을 위한 꽤 명확한 구분자(boundary)
- 마침표는 문장의 끝이 아니더라도 등장 가능

```
IP 192.168.56.31 서버에 들어가서 로그 파일 저장해서 aaa@gmail.com로 결과 좀 보내줘. 그 후 점심 먹으러 가자.
```

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

Sentence Segmentation: NLTK

```
text = "For strains harboring the pYV plasmid and Yop-encoding plasmids, bacteria were grown with aeration at 26 °C overnight in broth supplemented with 2.5 mm CaCl2 and 100 µg/ml ampicillin and then subcultured and grown at 26 °C until A600 of 0.2. At this point, the cultures were shifted to 37 °C and aerated for 1 h. A multiplicity of infection of 50:1 was used for YPIII(p-) incubations, and a multiplicity of infection of 25:1 was used for other derivatives. For the pYopE-expressing plasmid, 0.1 mm isopropyl-β-d-thiogalactopyranoside was supplemented during infection to induce YopE expression."
```

```
from ekorpkit.preprocessors.segmenter import NLTKSegmenter
seg = NLTKSegmenter()
print(seg.segment(text))
```

```
['For strains harboring the pYV plasmid and Yop-encoding plasmids, bacteria were grown with aeration at 26 °C overnight in broth supplemented with 2.5 mm CaCl2 and 100 µg/ml ampicillin and then subcultured and grown at 26 °C until A600 of 0.2.',
'At this point, the cultures were shifted to 37 °C and aerated for 1 h. A multiplicity of infection of 50:1 was used for YPIII(p-) incubations, and a multiplicity of infection of 25:1 was used for other derivatives.',
'For the pYopE-expressing plasmid, 0.1 mm isopropyl-β-d-thiogalactopyranoside was supplemented during infection to induce YopE expression.']
```

Sentence Segmentation: PySBD

```
from ekorpkit.preprocessors.segmenter import PySBDSegmenter
seg = PySBDSegmenter()
print(seg.segment(text))
```

```
['For strains harboring the pYV plasmid and Yop-encoding plasmids, bacteria were grown with
aeration at 26 °C overnight in broth supplemented with 2.5 mm CaCl2 and 100 μg/ml ampicillin and
then subcultured and grown at 26 °C until A600 of 0.2. ',
'At this point, the cultures were shifted to 37 °C and aerated for 1 h. ',
'A multiplicity of infection of 50:1 was used for YPIII(p-) incubations, and a multiplicity of
infection of 25:1 was used for other derivatives. ',
'For the pYopE-expressing plasmid, 0.1 mm isopropyl-β-d-thiogalactopyranoside was supplemented
during infection to induce YopE expression.']
```

Sentence Segmentation: KSS

```
text = "일본기상청과 태평양지진해일경보센터는 3월 11일 오후 2시 49분경에 일본 동해안을 비롯하여 대만, 알래스카, 하와이, 괌, 캘리포니아, 칠레 등 태평양 연안 50여 국가에 지진해일 주의보와 경보를 발령하였다. 다행히도 우리나라는 지진발생위치로부터 1,000km 이상 떨어진데다 일본 열도가 가로막아 지진해일이 도달하지 않았다. 지진해일은 일본 소마항에 7.3m, 카마이시항에 4.1m, 미야코항에 4m 등 일본 동해안 전역에서 관측되었다. 지진해일이 원해로 전파되면서 대만(19시 40분)에서 소규모 지진해일과 하와이 섬에서 1.4m(23시 9분)의 지진해일이 관측되었다. 다음날인 3월 12일 새벽 1시 57분경에는 진앙지로부터 약 7,500km 떨어진 캘리포니아 크레센트시티에서 2.2m의 지진해일이 관측되었다."
```

```
from ekorpkit.preprocessors.segmenter import KSSSegmenter
seg = KSSSegmenter(backend='None')
print(seg.segment(text))
```

```
['일본기상청과 태평양지진해일경보센터는 3월 11일 오후 2시 49분경에 일본 동해안을 비롯하여 대만, 알래스카, 하와이, 괌, 캘리포니아, 칠레 등 태평양 연안 50여 국가에 지진해일 주의보와 경보를 발령하였다.',
'다행히도 우리나라는 지진발생위치로부터 1,000km 이상 떨어진데다 일본 열도가 가로막아 지진해일이 도달하지 않았다.',
'지진해일은 일본 소마항에 7.3m, 카마이시항에 4.1m, 미야코항에 4m 등 일본 동해안 전역에서 관측되었다.',
'지진해일이 원해로 전파되면서 대만(19시 40분)에서 소규모 지진해일과 하와이 섬에서 1.4m(23시 9분)의 지진해일이 관측되었다.',
'다음날인 3월 12일 새벽 1시 57분경에는 진앙지로부터 약 7,500km 떨어진 캘리포니아 크레센트시티에서 2.2m의 지진해일이 관측되었다.']
```

한국어 토큰화의 어려움

- 영어는 New York과 같은 합성어나 he's 와 같이 줄임말에 대한 예외처리만 한다면, 띄어쓰기(whitespace)를 기준으로 하는 띄어쓰기 토큰화를 수행해도 단어 토큰화가 잘 작동
- 한국어는 영어와는 달리 띄어쓰기만으로는 토큰화를 하기에 부족
- 한국어의 띄어쓰기 단위는 `어절`
- 어절 토큰화는 한국어 NLP에서 지양됨
- 근본적인 이유는 한국어가 영어와는 다른 형태를 가지는 언어인 교착어라는 점에서 기인
- 교착어란 조사, 어미 등을 붙여서 말을 만드는 언어를 지칭

교착어의 특성

> 대부분의 한국어 NLP에서 기본 단위는 어절이 아닌 형태소

> 형태소(morpheme)란 뜻을 가진 가장 작은 말의 단위

- 자립 형태소: 접사, 어미, 조사와 상관없이 자립하여 사용할 수 있는 형태소로 그 자체로 단어
  - 체언(명사, 대명사, 수사), 수식언(관형사, 부사), 감탄사 등
- 의존 형태소: 다른 형태소와 결합하여 사용되는 형태소
  - 접사, 어미, 조사, 어간

> ex) 문장 : 에디가 책을 읽었다

- 띄어쓰기 단위 토큰화: [`에디가` , `책을` , `읽었다`]
- 형태소 단위로 분해
  - 자립 형태소 : `에디` , `책`
  - 의존 형태소 : `–가` , `–을` , `읽–` , `–었` , `–다`

한국어 띄어쓰기

한국어는 영어권 언어와 비교하여 띄어쓰기가 어렵고 잘 지켜지지 않는 경향 존재

- 한국어의 경우 띄어쓰기가 지켜지지 않아도 글을 쉽게 이해할 수 있는 언어
- 띄어쓰기가 없던 한국어에 띄어쓰기가 보편화된 것도 근대(1933년, 한글맞춤법통일안)의 일

띄어쓰기를 전혀 하지 않은 한국어와 영어 두 가지 경우를 봅시다.

> `이렇게띄어쓰기를전혀하지않고글을썼다고하더라도글을이해할수있다.`

> `Tobeornottobethatisthequestion.`

한국어는 수많은 코퍼스에서 띄어쓰기가 무시되는 경우가 많아 자연어 처리가 어려움

정제(Cleaning) 및 정규화(Normalization)

토큰화(tokenization): 코퍼스에서 용도에 맞게 토큰을 분류하는 작업
토큰화 전후, 텍스트 데이터를 용도에 맞게 정제(cleaning) 및 정규화(normalization) 필요

- 정제(cleaning) : 갖고 있는 코퍼스로부터 노이즈 데이터 제거
- 정규화(normalization) : 표현 방법이 다른 단어들을 동일 단어로 통합

> 정제 작업은 토큰화 작업에 방해가 되는 부분들을 배제시키고 토큰화 작업을 수행하기 위해서 토큰화 작업 전 주로 수행하지만, 토큰화 작업 이후에도 여전히 남아있는 노이즈들을 제거하기위해 지속적으로 이루어지기도 한다.

규칙에 기반한 표기가 다른 단어들의 통합

필요에 따라 직접 코딩을 통해 정의할 수 있는 정규화 규칙의 예로서 같은 의미를 갖고 있음에도, 표기가 다른 단어들을 하나의 단어로 정규화하는 방법을 사용할 수 있다.

> USA와 US는 같은 의미를 가지므로 하나의 단어로 정규화해볼 수 있다.
> uh-huh와 uhhuh는 형태는 다르지만 여전히 같은 의미를 갖고 있다.
> 이러한 정규화를 거치게 되면, US를 찾아도 USA도 함께 찾을 수 있다.

대, 소문자 통합

영어권 언어에서 대, 소문자를 통합하는 것은 단어의 개수를 줄일 수 있는 정규화 방법

- 영어권 언어에서 대문자는 문장의 맨 앞 등과 같은 특정 상황에서만 쓰이고, 대부분의 글은 소문자로 작성되기 때문
- 대문자를 소문자로 변환

대문자와 소문자가 구분되어야 하는 경우

- 미국을 뜻하는 단어 US와 우리를 뜻하는 us
- 회사 이름(General Motors)나, 사람 이름(Bush): 대문자로 유지

예외 사항을 크게 고려하지 않고, 모든 코퍼스를 소문자로 바꾸는 것이 종종 더 실용적인 해결책

불필요한 단어의 제거

노이즈 데이터(noise data)

- 자연어가 아니면서 아무 의미도 갖지 않는 글자들(특수 문자 등)
- 분석하고자 하는 목적에 맞지 않는 불필요 단어들

불필요 단어들을 제거하는 방법

- 불용어 제거
- 등장 빈도가 적은 단어, 길이가 짧은 단어 제거
- 등장 빈도가 적은 단어
    - 텍스트 데이터에서 너무 적게 등장해서 자연어 처리에 도움이 되지 않는 단어 존재

길이가 짧은 단어

- 영어: 길이가 짧은 단어들은 대부분 불용어에 해당
- 한국어: 길이가 짧은 단어라고 삭제하는 방법이 유효하지 않을 수 있음
- 영어 단어의 평균 길이는 6~7 정도, 한국어 단어의 평균 길이는 2~3 정도
- 한국어 단어는 한자어가 많고, 한 글자만으로도 의미를 가진 경우가 많음
- 영어는 길이가 2~3 이하인 단어를 제거하는 것의 효과 큼
  - 길이가 1인 단어 제거: 주로 의미를 갖지 못하는 단어인 관사 'a'와 주어로 쓰이는 'I' 제거
  - 길이가 2인 단어 제거: it, at, to, on, in, by 등과 같은 대부분 불용어 제거
  - 길이가 3인 단어 제거: fox, dog, car 등 길이가 3인 명사들이 제거되기 시작

```
# 길이가 1~2인 단어들을 정규 표현식을 이용하여 삭제
shortword = re.compile(r'\W*\b\w{1,2}\b')
print(shortword.sub('', text))
```

Text Normalization (정규화)

- 동일한 의미의 단어가 다른 형태를 갖는 것을 보완
  - 다른 형태의 단어들을 통일시켜 표준단어로 변환, 복잡성을 줄이기 위함 (BoW)
- Stemming (어간 추출)
  - 단수 – 복수, 현재형 – 미래형 등 단어의 다양한 변형을 하나로 통일
  - 의미가 아닌 규칙에 의한 변환
  - 영어의 경우, Porter stemmer, Lancaster stemmer 등이 유명
- Lemmatization (표제어 추출)
  - 사전을 이용하여 단어의 원형 - 기본형을 추출
  - 품사(part-of-speech)를 고려
  - 영어의 경우, WordNet을 이용한 WordNet lemmatizer가 많이 쓰임

표제어 추출(Lemmatization)

- 표제어(Lemma)는 '기본 사전형 단어' 정도의 의미
- 표제어 추출은 단어들로부터 표제어를 찾아가는 과정
- 표제어 추출은 단어들이 다른 형태를 가지더라도, 그 뿌리 단어를 찾아가서 단어의 개수를 줄일 수 있는지 판단
- ex) am, are, is의 표제어는 be

형태학적 파싱

> 표제어 추출을 위해 단어의 형태학적 파싱 필요

- 형태학(morphology)이란 형태소로부터 단어들을 만들어가는 학문
- 형태소의 종류로 어간(stem)과 접사(affix)가 존재

1. 어간(stem)
   : 단어의 의미를 담고 있는 단어의 핵심 부분.
2. 접사(affix)
   : 단어에 추가적인 의미를 주는 부분.

> 형태학적 파싱은 이 두 가지 구성 요소를 분리하는 작업
> ex) cats라는 단어에 대한 형태학적 파싱 결과: cat(어간)와 -s(접사)를 분리

WordNetLemmatizer

```
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
words = ['policy', 'doing', 'organization', 'have', 'going', 'love', 'lives', 'fly',
'dies', 'watched', 'has', 'starting']
```

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

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

'die'

```
lemmatizer.lemmatize('starting', 'v')
```

'start'

어간 추출(Stemming)

- 어간 추출은 형태학적 분석을 단순화한 버전
- 정해진 규칙만 보고 단어의 어미를 자르는 어림짐작의 작업
- 어간 추출 후에 나오는 결과 단어는 사전에 존재하지 않는 단어일 수도 있다
- 사전이 아닌 알고리즘(규칙)에 의해 변환하기 때문

Porter Stemmer

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

stemmer = PorterStemmer()
text = "Gold is often seen as an alternative currency in times of global economic uncertainty
and a refuge from financial risk."
```

```
tokenized_text = word_tokenize(text)
print([stemmer.stem(word) for word in tokenized_text])
```

['gold', 'is', 'often', 'seen', 'as', 'an', 'altern', 'currenc', 'in', 'time', 'of', 'global', 'econom', 'uncertainti', 'and', 'a', 'refug', 'from', 'financi', 'risk', '.']

> alternative → altern
>
> currency → currenc
>
> economic → econom
>
> uncertainty → uncertainti
>
> financial → financi

Porter Stemmer - Rules

> ALIZE → AL
> ANCE → 제거
> ICAL → IC

```
words = ['formalize', 'allowance', 'electricical']

print([stemmer.stem(word) for word in words])
```

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

LancasterStemmer

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

porter_stemmer = PorterStemmer()
lancaster_stemmer = LancasterStemmer()
text = "Gold is often seen as an alternative currency in times of global economic uncertainty
and a refuge from financial risk."
tokenized_text = word_tokenize(text)
```

```
print([porter_stemmer.stem(w) for w in tokenized_text])
print([lancaster_stemmer.stem(w) for w in tokenized_text])
```

['gold', 'is', 'often', 'seen', 'as', 'an', 'altern', 'currenc', 'in', 'time', 'of', 'global', 'econom', 'uncertainti', 'and', 'a', 'refug', 'from', 'financi', 'risk', '.']

['gold', 'is', 'oft', 'seen', 'as', 'an', 'altern', 'cur', 'in', 'tim', 'of', 'glob', 'econom', 'uncertainty', 'and', 'a', 'refug', 'from', 'fin', 'risk', '.']

규칙에 기반한 알고리즘의 오류

ex) 포터 알고리즘에서 organization을 어간 추출했을 때의 결과

> organization → organ

- organ에 대해서 어간 추출을 한다고 하더라도 결과는 역시 organ
- 의미가 동일한 경우에만 같은 단어를 얻기를 원하는 정규화의 목적 위배

	Stemming	Lemmatization
am	am	be
the going	the go	the going
having	hav	have

한국어에서의 어간 추출

한국어의 5언 9품사

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

용언에 해당되는 '동사'와 '형용사'는 어간(stem)과 어미(ending)의 결합으로 구성

활용(conjugation)

> 활용이란 용언의 어간(stem)이 어미(ending)를 가지는 것 지칭

- 어간(stem) :
  - 용언(동사, 형용사)을 활용할 때, 원칙적으로 모양이 변하지 않는 부분.
  - 활용에서 어미에 선행하는 부분. 때론 어간의 모양도 바뀔 수 있음(예: 긋다, 긋고, 그어서, 그어라).
- 어미(ending):
  - 용언의 어간 뒤에 붙어서 활용하면서 변하는 부분이며, 여러 문법적 기능을 수행

> 어간이 어미를 취할 때, 어간의 모습이 일정하다면 `규칙 활용`, 어간이나 어미의 모습이 변하는 `불규칙 활용`

활용(conjugation)

- 규칙 활용
  - 규칙 활용은 어간이 어미를 취할 때, 어간의 모습이 일정

    `잡/어간 + 다/어미`
  - 어간이 어미가 붙기전의 모습과 어미가 붙은 후의 모습이 같으므로, 규칙 기반으로 어미를 단순히 분리해주면 어간 추출이 됩니다.
- 불규칙 활용
  - 불규칙 활용은 어간이 어미를 취할 때 어간의 모습이 바뀌거나 취하는 어미가 특수한 어미일 경우

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

    `오르+ 아/어` → `올라`, `하+아/어` → `하여`, `이르+아/어` → `이르러`
  - 어미가 붙는 과정에서 어간의 모습이 바뀌었으므로 단순한 분리만으로 어간 추출이 되지 않고 좀 더 복잡한 규칙 필요

불용어(Stopwords)

불용어(stopwords): 자주 등장하지만 분석에 있어서는 큰 도움이 되지 않는 단어

ex) I, my, me, over, 조사, 접미사 같은 단어들은 문장에서는 자주 등장하지만 실제 의미 분석을 하는 데는 거의 기여하는 바가 크지 않다

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

stop_words_list = stopwords.words('english')
print(len(stop_words_list))
print(stop_words_list[:10])
```

179

['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're"]

NLTK를 통해서 불용어 제거

```
word_tokens = word_tokenize(text)

result = []
for word in word_tokens:
    if word not in stop_words_list:
        result.append(word)

print(word_tokens)
print(result)
```

['Gold', 'is', 'often', 'seen', 'as', 'an', 'alternative', 'currency', 'in', 'times', 'of', 'global', 'economic', 'uncertainty', 'and', 'a', 'refuge', 'from', 'financial', 'risk', '.']
['Gold', 'often', 'seen', 'alternative', 'currency', 'times', 'global', 'economic', 'uncertainty', 'refuge', 'financial', 'risk', '.']