Text Preprocessing

Jehyuk Lee

Department of AI, Big Data & Management

Kookmin University

Contents

- Text Preprocessing
- Text Preprocessing for English
- Text Preprocessing for Korean
- Additional Preprocessing steps for ML task

1. Text Preprocessing Overview

자연어(Natural Language)

- **자연어**
  - 우리의 말은 '문자'로 구성
  - 말의 의미는 '단어'로 구성
  - 단어는 의미의 최소 단위 → 이를 컴퓨터에게 이해시켜야 함
    - 단어를 벡터형식으로 표현하면서 의미도 내포해야 함
  - How?

Dynamic Web Page

- **텍스트는 비구조적(unstructured)이고 지저분함(noisy)**
  - 이를 분석에 사용하려면 전처리가 필요함
  - 특히, ML모델에 사용하기 위해서는 추가 전처리가 필요

Text Preprocessing

- **텍스트 전처리 과정**
  - 다음과 같은 전처리 과정들을 거쳐서 정제된 텍스트 데이터를 생성한다.

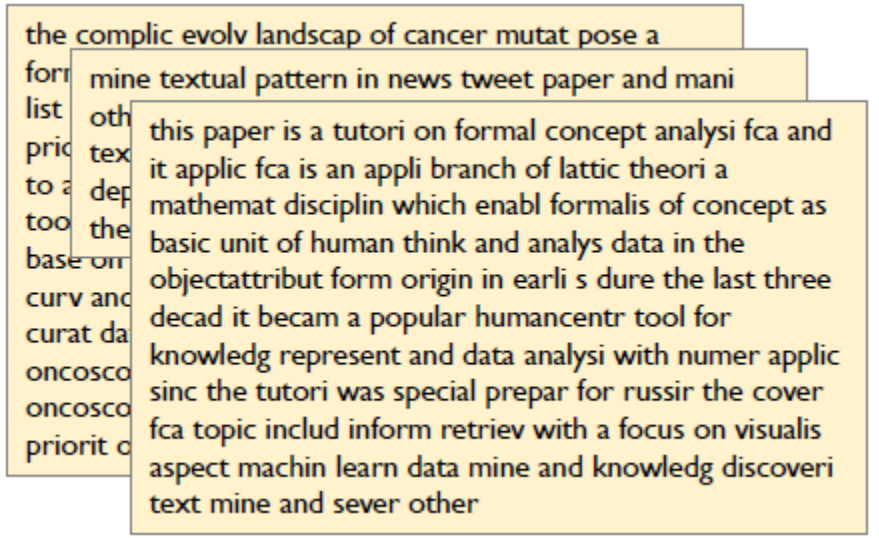
Tokenization	• 주어진 말뭉치에서 토큰(token)이라는 단위로 나누는 작업
POS Tagging	• Token에 앞뒤 문맥 상 적합한 품사를 태깅하는 작업
Normalization & Cleaning	• **Normalization:** 표현 방법이 다른 단어들을 통합하여 같은 언어로 만드는 작업 • **Cleaning:** 말뭉치에서 노이즈를 제거하는 작업
Stopword Removal	• 무의미한 단어 토큰을 제거하는 작업
Lemmatization & Stemming	• **Lemmatization:** 기본 사전형 단어인 표제어를 추출하는 작업 • **Stemming:** 규칙 기반으로 단어를 원형으로 복원하는 작업

Tokenization (토큰화)

- **주어진 말뭉치(corpus)에서 토큰이라고 불리는 단위로 나누는 행위**
  - 토큰은 의미 있는 단위로 정의함
    - 보통 문장, 단어 단위로 토큰화를 수행함
    - 한글은 조금 더 복잡한 토큰화 작업이 필요함

Tokenization (토큰화)

- **말뭉치(Corpus)**
  - 대량의 텍스트 데이터
    - 맹목적으로 수집된 텍스트 데이터 X
    - 자연어 처리 관련 목적으로 수집된 텍스트 데이터
  - 이 데이터에는 자연어에 대한 사람의 지식이 담겨있다고 가정
    - 문장을 쓰는 방법/단어 선택 방법/단어 의미...
  - 말뭉치에서, 자동적/효율적으로 핵심을 추출하려는 목적

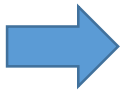


(그림 출처: https://github.com/pilsung-kang/Text-Analytics/blob/master/04%20Text%20Representation%20I%20-%20Classic%20Methods/04_Text%20Representation%20I_Classic%20Methods.pdf)

Tokenization (토큰화)

- **문장 토큰화 (Sentence Tokenization)**
  - 문서 혹은 말뭉치(corpus)를 문장 단위로 나누는 행위
  - 문장을 구분하는 기호(., !, ? 등)를 사용하면 될 것 같지만?
    - 그렇게 간단한 문제가 아님
    - 잘못된 예시) Ph.D → Ph/D, 255.255.255.0 → 255/255/255/0
  - 별도의 Package를 사용

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.

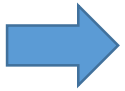


['His barber kept his word.'/
'But keeping such a huge secret to himself was driving him crazy.'/
'Finally, the barber went up a mountain and almost to the edge of a cliff.'/
'He dug a hole in the midst of some reeds.'/
'He looked about, to make sure no one was near.']

Tokenization (토큰화)

- **단어 토큰화 (Word Tokenization)**
  - 문서 혹은 말뭉치(corpus)를 단어 단위로 나누는 행위
  - 특수문자(., !,?, “, ‘ 등)를 제거하고 띄어쓰기를 기준으로 나누면 될 것 같지만?
    - 그렇게 간단한 문제가 아님
    - 잘못된 예시) Don’t → Don/t, Mr.Jone’s → Mr/Jone/s
  - 별도의 Package를 사용. 특히, 한국어는 고유한 특성으로 복잡한 과정이 필요

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.



['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', '.']

Tokenization (토큰화)

- **토큰화시 고려해야 할 사항**
  - 구두점, 특수문자를 단순히 제거해서는 안됨
    - 잘못된 예시) $45.45 → 45/55, AT&T → AT/T
  - 줄임말 처리 방법
    - 예시) We're → We are, I'm → I am으로 인식해야 함
  - 단어 내 띄어쓰기가 있는 경우
    - 예시) New York, Los Angeles → 하나의 단어로 인식해야 함

Tokenization (토큰화)

- **POS(Part-of-speech) Tagging**
  - 토큰에 앞뒤 문맥상 적합한 품사를 태깅하는 작업
  - 같은 단어라도 다른 품사로 사용될 수 있음
    - 예시) fly: (v)날다/(n)파리
    - 예시) 못: (n)'못'을 박다 / (adv)'못'먹다
  - 토큰의 의미를 제대로 해석하려면, 해당 토큰이 어떤 품사로 사용되었는지 알아야 함

Cleaning (정제)

- **갖고있는 말뭉치(corpus)에서 노이즈 데이터를 제거하는 행위**
  - 다음과 같은 행위들이 포함됨
    - 등장 빈도가 적은 데이터 제거
    - (ENG)길이가 짧은 단어 제거
      - I, by, at, on 등
    - 불용어(Stopword) 제거
    - 오탈자 교정/제거
      - 참고: https://medium.com/@yashj302/spell-check-and-correction-nlp-python-f6a000e3709d
    - 띄어쓰기 교정
  - Cleaning은 Tokenization 이후 뿐만 아니라 이전에도 진행함

Normalization (정규화)

- **같은 의미/다른 표현의 단어들을 하나로 통합, 같은 단어로 만드는 행위**
  - 다음과 같은 행위들이 포함됨
    - 대소문자 통합
    - Hmm, Hmmm, Hmmmm → Hmm
    - 10, 159, 237 → num(숫자가 중요하지 않은 경우)

Lemmatization (표제어 추출)

- **대상 단어의 품사에 맞는 단어의 표제어를 추출하는 행위**
  - 표제어(lemma): 기본 사전형 단어
  - 형변환 사전을 기반으로 표제어를 추출함
    - 예시) is, are, am → be, cooking(v) → cook, cooking(n) → cooking(n)
  - 말뭉치에 있는 단어의 수를 줄이는 기법으로 정규화 기법의 일종

Stemming (어간 추출)

- **규칙 기반으로 단어의 변형된 형태를 제거/치환하여 어간을 추출하는 작업**
  - 어간을 추출하지만, 형태학적 분석보다는 정해진 규칙으로 어미를 잘라내는 작업과 유사
  - Stemming 결과는 존재하지 않는 단어일 수 있음

- **형태학적 parsing**
  - 형태소(morpheme): 의미가 있는 가장 작은 말의 단위
    - 어간(stem): 단어의 의미를 담고 있는 핵심 부분
    - 접사(affix): 단어에 추가적인 의미를 주는 부분
  - 형태학(morphology): 형태소로부터 단어를 형성하는 학문
    - 형태학적 parsing: 어간/접사를 분리하는 과정

Stemming vs Lemmatization

Word	Stemming	Lemmatization
Love	Lov	Love
Loves	Lov	Love
Loved	Lov	Love
Loving	Lov	Love
Innovation	Innovat	Innovation
Innovations	Innovat	Innovation
Innovate	Innovat	Innovate
Innovates	Innovat	Innovate
Innovative	Innovat	Innovative

(Source: https://github.com/pilsung-kang/Text-Analytics/blob/master/02%20Text%20Preprocessing/02_Text%20Preprocessing_part2.pdf)

Stemming vs Lemmatization

Stemming

Commonly used in Information Retrieval:

- Can be achieved with rule-based algorithms, usually based on suffix-stripping
- Standard algorithm for English: the *Porter* stemmer
- Advantages: simple & fast
- Disadvantages:
  - Rules are language-dependent
  - Can create words that do not exist in the language, e.g., *computers* → *comput*
  - Often reduces different words to the same stem, e.g., *army, arm* → *arm*
    stocks, stockings → *stock*
- Stemming for German: German stemmer in the full-text search engine *Lucene*, *Snowball* stemmer with German rule file

Lemmatization

Lemmatization is the process of deriving the base form, or *lemma*, of a word from one of its inflected forms. This requires a morphological analysis, which in turn typically requires a *lexicon*.

- Advantages:
  - identifies the *lemma* (root form), which is an actual word
  - less errors than in stemming
- Disadvantages:
  - more complex than stemming, slower
  - requires additional language-dependent resources
- While stemming is good enough for Information Retrieval, Text Mining often requires lemmatization
  - Semantics is more important (we need to distinguish an *army* and an *arm*!)
  - Errors in low-level components can multiply when running downstream

(Source: https://github.com/pilsung-kang/Text-Analytics/blob/master/02%20Text%20Preprocessing/02_Text%20Preprocessing_part2.pdf)

Stopword Removal (불용어 제거)

- 불용어(Stopword)는 무의미한 단어 토큰을 의미
  - 예시) I, my, me, ~를, ~을 등

- 이러한 stopword를 제거하는 행위를 stopword removal이라 함

2. Text Preprocessing for English

Tokenization

- **주어진 말뭉치(corpus)에서 토큰이라고 불리는 단위로 나누는 행위**
  - 문장 토큰화(Sentence Tokenization)
    - NLTK의 sent_tokenization을 사용

```
my_str = """
Please access the server with IP address 192.168.56.31, save the log file,
and send it to aaa@gmail.com. After that, let's go eat lunch.
Think about what you want to eat.
"""
```

```
sent_tokens = nltk.sent_tokenize(my_str)
```

```
['Please access the server with IP address 192.168.56.31, save the log file, and send it to aaa@gmail.com.',
 "After that, let's go eat lunch.",
 'Think about what you want to eat.']
```

Tokenization

- **주어진 말뭉치(corpus)에서 토큰이라고 불리는 단위로 나누는 행위**
  - 단어 토큰화(Word Tokenization)
    - 대부분의 경우 띄어쓰기로 단어 토큰이 구분 됨
      - 그러나, 예외 case들도 분명히 존재함
    - NLTK의 word_tokenize, WordPuctTokenizer, RegexTokenizer을 사용

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

```
result = nltk.word_tokenize(my_str)
print(result)
```

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

```
result = nltk.wordpunct_tokenize(my_str)
print(result)
```

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

Cleaning

- **갖고있는 말뭉치(corpus)에서 노이즈 데이터를 제거하는 행위**
  - 다음과 같은 행위들이 포함됨
    - 등장 빈도가 적은 데이터 제거
    - (ENG)길이가 짧은 단어 제거
      - I, by, at, on 등
    - 불용어(Stopword) 제거
    - 오탈자 교정/제거
      - 참고: https://medium.com/@yashj302/spell-check-and-correction-nlp-python-f6a000e3709d
    - 띄어쓰기 교정
  - Cleaning은 Tokenization 이후 뿐만 아니라 이전에도 진행함

Normalization

- **같은 의미/다른 표현의 단어들을 하나로 통합, 같은 단어로 만드는 행위**
  - 다음과 같은 행위들이 포함됨
    - 대소문자 통합
    - Hmm, Hmmm, Hmmmm → Hmm
    - 10, 159, 237 → num(숫자가 중요하지 않은 경우)

Lemmatization

- **대상 단어의 품사에 맞는 단어의 표제어를 추출하는 행위**
  - NLTK의 WordNetLemmatizer를 사용

```
my_str = """I am actively looking for Ph.D. students.
And you are a Ph.D. student.
Would you like to join my laboratory?"""
```

```
lemmatizer = WordNetLemmatizer()
print("표제어 추출 전: ")
print(results)
print("표제어 추출 후: ")
print([lemmatizer.lemmatize(word) for word in results])
```

```
표제어 추출 전:
['i', 'am', 'actively', 'looking', 'for', 'ph.d.', 'students', 'and', 'you', 'are', 'a', 'ph.d.', 'student', 'would', 'you', 'like', 'to',
'join', 'my', 'laboratory']
표제어 추출 후:
['i', 'am', 'actively', 'looking', 'for', 'ph.d.', 'student', 'and', 'you', 'are', 'a', 'ph.d.', 'student', 'would', 'you', 'like', 'to',
'join', 'my', 'laboratory']
```

Lemmatization

- **대상 단어의 품사에 맞는 단어의 표제어를 추출하는 행위**
  - 품사에 좀 더 정확한 정보를 입력해야 Lemmatization 성능이 좋음
    - NLTK의 Pos tag를 사용하면 좋으나, WordNetLemmatizer와 사용하는 품사의 종류가 다름
    - 추가로 변환하는 작업이 필요함

```
my_str = """I am actively looking for Ph.D. students.
And you are a Ph.D. student.
Would you like to join my laboratory?"""
```

```
print("표제어 추출 후: ")
print(lemmas)
```

```
표제어 추출 전:
['i', 'am', 'actively', 'looking', 'for', 'ph.d.', 'students', 'and', 'you', 'are', 'a', 'ph.d.', 'student', 'would', 'you', 'like', 'to',
'join', 'my', 'laboratory']
표제어 추출 후:
['i', 'be', 'actively', 'look', 'for', 'ph.d.', 'student', 'and', 'you', 'be', 'a', 'ph.d.', 'student', 'would', 'you', 'like', 'to', 'joi
n', 'my', 'laboratory']
```

Stopword removal

- **NLTK에서는 미리 정한 stopword list가 존재**
  - nltk.corpus.stopwords

```
stop = stopwords.words('english')
```

```
print(stop)
```

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "you'll", "you'd", 'your', 'yours', 'yourself',
'yourselves', 'he', 'him', 'his', 'himself', 'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 'the
ir', 'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'those', 'am', 'is', 'are', 'was', 'wer
e', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'beca
use', 'as', 'until', 'while', 'of', 'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'afte
r', 'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further', 'then', 'once', 'here',
'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more', 'most', 'other', 'some', 'such', 'no', 'nor', 'not',
'only', 'own', 'same', 'so', 'than', 'too', 'very', 's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'l
l', 'm', 'o', 're', 've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn', "hadn't", 'has
n', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn', "mustn't", 'needn', "needn't", 'shan', "shan't", 's
houldn', "shouldn't", 'wasn', "wasn't", 'weren', "weren't", 'won', "won't", 'wouldn', "wouldn't"]
```

- **이 외에도 사용자가 직접 stopword list를 추가하여 사용할 수 있음**

```
stop.extend(['hey', 'amazon'])
```

3. Text Preprocessing for Korean

Tokenization

- **주어진 말뭉치(corpus)에서 토큰이라고 불리는 단위로 나누는 행위**
  - 문장 토큰화(Sentence Tokenization)
    - KSS의 split_sentence를 사용

동작 예시

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

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

Tokenization

- **주어진 말뭉치(corpus)에서 토큰이라고 불리는 단위로 나누는 행위**
  - 단어 토큰화(Word Tokenization)
    - 영어: 대부분의 경우 띄어쓰기로 단어 토큰이 구분 됨
      - 그러나, 예외 case들도 분명히 존재함
    - 한국어: 띄어쓰기로 단어 토큰이 구분되지 않음
      - 한국어에서 띄어쓰기로 구분된 단위를 '어절'이라고 함
      - '어절'단위로 토큰화를 하면 안된다! Why?

Tokenization

- **한국어는 교착어이다**
  - **교착어:** 어간에 조사, 어미 등의 문법 형태소가 결합하여 문법적인 기능이 부여되는 언어
    - 예시: 조사(~는, ~를, ~이, ~가)

예시문

'**사과가** 건강에 좋다고 하더라구. **사과에** 비타민이 많다잖아. 그래서 **사과를** 사러 근처 슈퍼에 갔더니 **사과** 가격이 글쎄 3개에 5천원이라고 해서 사지 않았어.'

이 예시문을 띄어쓰기 단위로 토큰화를 할 경우에는
'사과가', '사과에', '사과를', '사과'가 전부 다른 단어로 간주된다.

Tokenization

- **한국어는 교착어이다**
  - 따라서, 한국어 토큰화에서는 반드시 형태소 분석이 들어가야 한다.
  - **형태소:** 뜻을 가진 가장 작은 말의 단위
    - **자립 형태소**
      - 접사, 어미, 조사와 상관없이 자립하여 사용할 수 있는 형태소
      - 체언(명사, 대명사, 수사), 수식언(관형사, 부사), 감탄사 등
    - **의존 형태소**
      - 다른 형태소와 결합하여 사용되는 형태소
      - 접사, 어미, 조사, 어간

Tokenization

- **한국어는 교착어이다**
  - 따라서, 한국어 토크나이저는 일반적으로 형태소 분석기가 사용된다.
  - 다양한 한국어 토크나이저가 존재함
    - 지도 학습 (Supervised learning) 기반
      - 언어학 전문가들이 태깅한 형태소 분석 말뭉치로부터 학습된 지도 학습 기반 모델
      - KoNLPy의 5개 형태소 분석기(mecab, 꼬꼬마, 한나눔, Okt, komoran)
      - 카카오의 Khaiii 형태소 분석기
    - 비지도 학습 (Unsupervised learning) 기반
      - 데이터의 패턴을 모델 스스로 학습하게 함으로써 형태소를 분석하는 방법
        - 데이터에 자주 등장하는 단어들을 형태소로 인식
      - Soynlp, sentencepiece 등

Tokenization

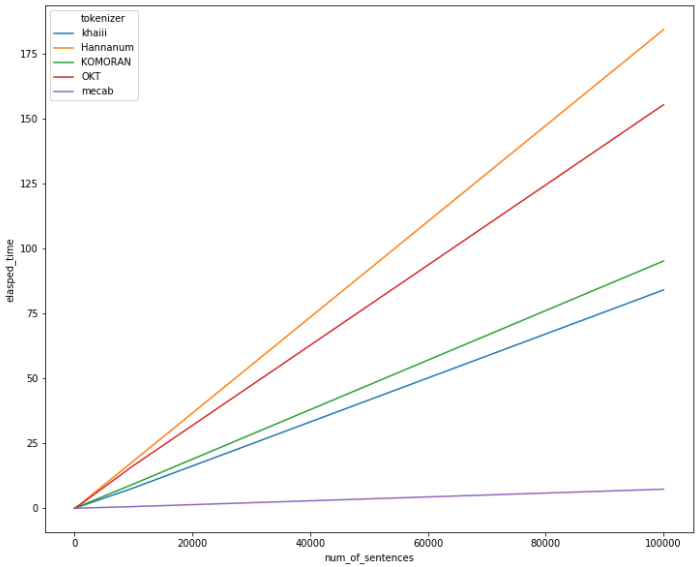
- **Task에 맞는 형태소 분석기를 사용**
  - **Mecab:** 연산속도도 빠르고, 분석 품질도 상위권 하지만 윈도우에서는 설치가 매우 어려움
  - **KOMORAN:** 자소분리나 오탈자에 대해 분석 품질 보장
  - **Hannanum, Khaiii:** 일부 케이스에 대한 분석 품질이 조금 떨어짐
  - **꼬꼬마:** 분석 시간이 조금 길다는 단점

→ 분석 결과는 https://iostream.tistory.com/144를 참고

> Warning:
>
> - KoNLPy's `Mecab()` class is not supported on Windows machines.

Mecab이 필요하다면 다음 페이지 참고
→ https://uwgdqo.tistory.com/363

10만 문장 분석시 소요 시간



(Source: https://iostream.tistory.com/144)

POS Tagging

- 한국어 형태소 분석기가 Tokenizer 역할을 함
  - KonIpy의 다양한 형태소 분석기를 활용하여 POS tagging을 할 수 있음

(Source: https://iostream.tistory.com/144)

Cleaning

- **갖고있는 말뭉치(corpus)에서 노이즈 데이터를 제거하는 행위**
  - 다음과 같은 행위들이 포함됨
    - 등장 빈도가 적은 데이터 제거
    - 불용어(Stopword) 제거
    - 오탈자 교정/제거
      - 참고: https://medium.com/@yashj302/spell-check-and-correction-nlp-python-f6a000e3709d
    - 띄어쓰기 교정
  - Cleaning은 Tokenization 이후 뿐만 아니라 이전에도 진행함

Cleaning

- **오탈자 교정/제거**
  - 네이버/카카오의 API를 활용한 오탈자 교정
  - py-hanspell
  - 그러나, 성능은 부산대학교 맞춤법 검사기가 가장 좋다고 알려져 있음
    - 권혁철 교수 & 부산대 인공지능연구실 & 나라인포테크
    - 참고: https://medium.com/@yashj302/spell-check-and-correction-nlp-python-f6a000e3709d

Cleaning

- 오탈자 교정/제거
  - py-hanspell을 이용한 오탈자 교정

```
from hanspell import spell_checker

sent = "맞춤법 틀리면 외 않되?"
spelled_sent = spell_checker.check(sent)

hanspell_sent = spelled_sent.checked
print(hanspell_sent)
```

```
맞춤법 틀리면 왜 안돼?
```

```
sent = "오지호는극중두얼굴의사나이성준역을맡았다.성준은국내유일의태백권전승자를가리는결전의날을앞두고20년간동고동락한사형인진수(정의욱분)를찾으러속세로내려온인물이다."
spelled_sent = spell_checker.check(sent)

hanspell_sent = spelled_sent.checked
print(hanspell_sent)
print(kospacing_sent) # 앞서 사용한 kospacing 패키지에서 얻은 결과
```

```
오지호는 극 중 두 얼굴의 사나이 성준 역을 맡았다. 성준은 국내 유일의 태백권 전승자를 가리는 결전의 날을 앞두고 20년간 동고동락한 사형인 진수(정인욱 분)를 찾으러 속세로 내려온 인물이다.
오지호는 극중 두 얼굴의 사나이 성준 역을 맡았다. 성준은 국내 유일의 태백권 전승자를 가리는 결전의 날을 앞두고 20년간 동고동락한 사형인 진수(정의욱 분)를 찾으러 속세로 내려온 인물이다.
```

Cleaning

- **띄어쓰기 교정**
  - 한국어는 띄어쓰기가 어려울 뿐만 아니라 잘 지켜지지 않음
    - 띄어쓰기가 되어 있지 않아도 알아볼 수 있음
    - 정제되지 않은 데이터에서는 띄어쓰기가 되어 있지 않은 경우가 대부분

> 한국어에서는띄어쓰기가중요하지않아서이렇게해도알아볼수있다.

  - 영어는 띄어쓰기가 비교적 잘 지켜짐
    - 띄어쓰기가 되어있지 않으면 알아보기 어려움

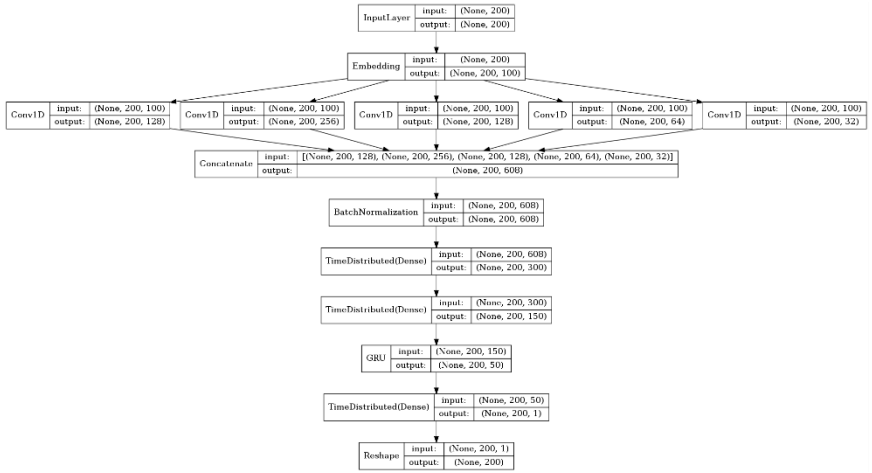
> tobeornottobethatisthequestion

Cleaning

- **띄어쓰기 교정**
  - KoSpacing
    - 딥러닝 기반의 띄어쓰기 교정 package

"김형호영화시장분석가는'1987'의네이버영화정보네티즌10점평에서언급된단어들을지난해12월27일부터올해1월10일까지통계프로그램R과KoNLP패키지로텍스트마이닝하여분석했다

"김형호 영화시장 분석가는 '1987'의 네이버 영화 정보 네티즌 10점 평에서 언급된 단어들을 지난해 12월 27일부터 올해 1월 10일까지 통계 프로그램 R과 KoNLP 패키지로 텍스트마이닝하여 분석했다."



(Source: https://github.com/haven-jeon/KoSpacing)

Normalization

- **같은 의미/다른 표현의 단어들을 하나로 통합, 같은 단어로 만드는 행위**
  - Hmm, Hmmm, Hmmmm → Hmm
    - soynlp의 normalizer를 사용

```
from soynlp.normalizer import *
```

```
print(emoticon_normalize('아ㅋㅋㅋㅋ이영화존잼쓰ㅠㅠㅠㅠㅠ', num_repeats=2))
print(emoticon_normalize('아ㅋㅋㅋㅋㅋㅋㅋㅋㅋ이영화존잼쓰ㅠㅠㅠㅠ', num_repeats=2))
print(emoticon_normalize('아ㅋㅋㅋㅋㅋㅋㅋㅋㅋㅋㅋ이영화존잼쓰ㅠㅠㅠㅠㅠㅠ', num_repeats=2))
print(emoticon_normalize('아ㅋㅋㅋㅋㅋㅋㅋㅋㅋㅋㅋㅋㅋㅋㅋㅋ이영화존잼쓰ㅠㅠㅠㅠㅠㅠㅠㅠ', num_repeats=2))
```

```
아ㅋㅋ영화존잼쓰ㅠㅠ
아ㅋㅋ영화존잼쓰ㅠㅠ
아ㅋㅋ영화존잼쓰ㅠㅠ
아ㅋㅋ영화존잼쓰ㅠㅠ
```

```
print(repeat_normalize('와하하하하하하하하하핫', num_repeats=2))
print(repeat_normalize('와하하하하하하핫', num_repeats=2))
print(repeat_normalize('와하하하하핫', num_repeats=2))
```

```
와하하핫
와하하핫
와하하핫
```

Lemmatization & Stemming

- **대상 단어의 품사에 맞는 단어의 표제어/어간을 추출하는 행위**
  - 표제어(lemma): 기본 사전형 단어
  - 형변환 사전을 기반으로 표제어를 추출함
    - 예시) is, are, am → be, cooking(v) → cook, cooking(n) → cooking(n)
  - 말뭉치에 있는 단어의 수를 줄이는 기법으로 정규화 기법의 일종
  - 형태소 분석기에서 기본적인 표제어 추출과 어간 추출 기능은 제공한다.
    - 그러나, 성능이 별로 좋지는 못함. **Why?**

Lemmatization & Stemming

- **한국어는 다음과 같이 5언 9품사의 구조를 갖고 있음**
  - 어간 추출이 어려운 이유는 가변어의 다양한 활용 때문
    - 용언: 부사가 그 앞에 올 수 있고, 선어말어미가 그 뒤에 올 수 있고, 그 뒤에 어말어미
    - 어미: 용언 뒤에 와야 함

형태	기능	품사	태그
불변어	체언	명사	NNG
		대명사	NNP
		수사	NR
	독립언	감탄사	XG
	관계언	조사	조사
	수식언	관형사	관형사
		부사	부사
가변어	용언	동사	VV
		형용사	VA

Lemmatization & Stemming

- **용언의 활용**
  - 활용(conjugation)
    - 용언의 어간(stem)이 어미(ending)을 가지는 일
  - 가변어는 어간에 결합되는 어미가 달라지면서 표현형이 바뀜
    - **원형(Canonical form, lemma)**
      - 어간에 어미 '-다'가 결합된 형태
      - 예시: 하 + 다 / 가 + 다
    - **표현형(Surficial form)**
      - 그 외의 어미가 결합된 형태
      - 예시: 하 + ㄴ다 → 한다

형태	기능	품사	태그
불변어	체언	명사	NNG
		대명사	NNP
		수사	NR
	독립언	감탄사	XG
	관계언	조사	조사
	수식언	관형사	관형사
		부사	부사
가변어	용언	동사	VV
		형용사	VA

(Source: https://github.com/lovit/textmining-tutorial/blob/master/topics/topic2_word_tokenizing/lemmatizer.pdf)

Lemmatization & Stemming

- **용언의 두가지 활용 (1): 규칙 활용**
  - 자음/모음열 기준에서 변화가 없는 활용
  - (1) 두 단어가 그대로 결합되는 경우
    - 잡다: 잡/어간 + 다/어미
  - (2) 어간 마지막 글자의 종성이 없는 경우, 받침 추가
    - 말한다: 말하/어간 + ㄴ다/어미

(Source: https://github.com/lovit/textmining-tutorial/blob/master/topics/topic2_word_tokenizing/lemmatizer.pdf)

Lemmatization & Stemming

- **용언의 두가지 활용 (2): 불규칙 활용**
  - 자음/모음열 기준에서 변화가 생기는 활용
  - (1) 어간 마지막 종성이 'ㄷ'이고 어미 첫글자가 'ㅇ': 'ㄷ'→'ㄹ'
    - 깨달아: 깨닫/어간 + 아/어미
    - (질문을) 물었다: 묻/어간 + 었다/어미
    - (물건을) 묻었다: 묻/어간 + 었다/어미
  - (2) 어간 마지막 종성이 'ㅂ'이고 어미 첫글자가 'ㅇ': 'ㅂ' → 'ㅗ/ㅜ'
    - 더러워: 더럽/어간 + 워/어미
    - 고와: 곱/어간 + 아/어미
    - 아름다워: 아름답/어간 + 아/어미
  - 이 외에도 수많은 불규칙 활용 규칙이 존재
    - https://namu.wiki/w/%ED%95%9C%EA%B5%AD%EC%96%B4/%EB%B6%88%EA%B7%9C%EC%B9%99%20%ED%99%9C%EC%9A%A9

(Source: https://github.com/lovit/textmining-tutorial/blob/master/topics/topic2_word_tokenizing/lemmatizer.pdf)

Lemmatization & Stemming

- **활용과 표제어 추출**
  - **활용(Conjugation):** 용언의 원형을 표현형으로 변환하는 과정
  - **표제어 추출(lemmatization):** (용언에 한하여) 용언의 표현형을 원형으로 변환하는 과정

- 한국어는 다양한 활용으로 인해 lemmatization이 어렵다.
  - 따라서, 라이브러리에서 제공하는 결과를 확인해봐야 한다.
  - 참고: soynlp의 lemmatization 함수
    - https://lovit.github.io/nlp/2018/06/07/lemmatizer/

(Source: https://github.com/lovit/textmining-tutorial/blob/master/topics/topic2_word_tokenizing/lemmatizer.pdf)

4. Additional Preprocessing steps for ML task

Additional Text Preprocessing steps

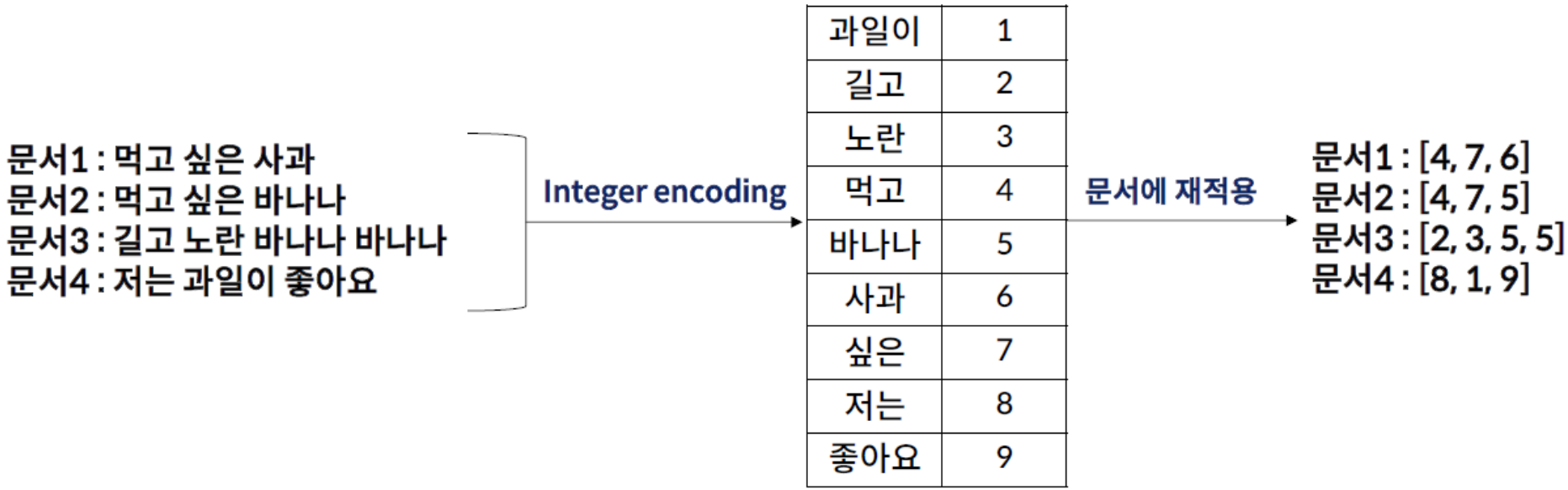
- 일반적으로, text 전처리는 이전 section에 나왔던 step들을 의미

- 그러나, 모델링을 하기 위해서는 추가로 몇가지 작업들이 더 필요
  - Task에 따라 필요할 수도, 불필요할 수도 있음

Integer Encoding

- **컴퓨터는 텍스트보다 숫자를 잘 처리할 수 있음**
  - 텍스트를 숫자로 바꾸자!

- **각 단어를 고유한 정수에 mapping하는 작업**
  - 중복을 허용하지 않음 → 이러한 모든 단어들의 집합을 단어집합(vocabulary)



Padding

- **가상의 단어를 추가하여 길이를 맞춰주는 역할**
  - 모든 문장의 길이는 서로 다름 → Integer Encoding 결과도 서로 다름 → 맞춰주자!
  - 기계가 이를 병렬 연산할 수 있어 연산속도가 빨라짐
  - 특히, 인공신경망(Artificial Neural Network) 계열 모델에서 많이 필요함

과일이	1
길고	2
노란	3
먹고	4
바나나	5
사과	6
싶은	7
저는	8
좋아요	9

문서에 재적용 →

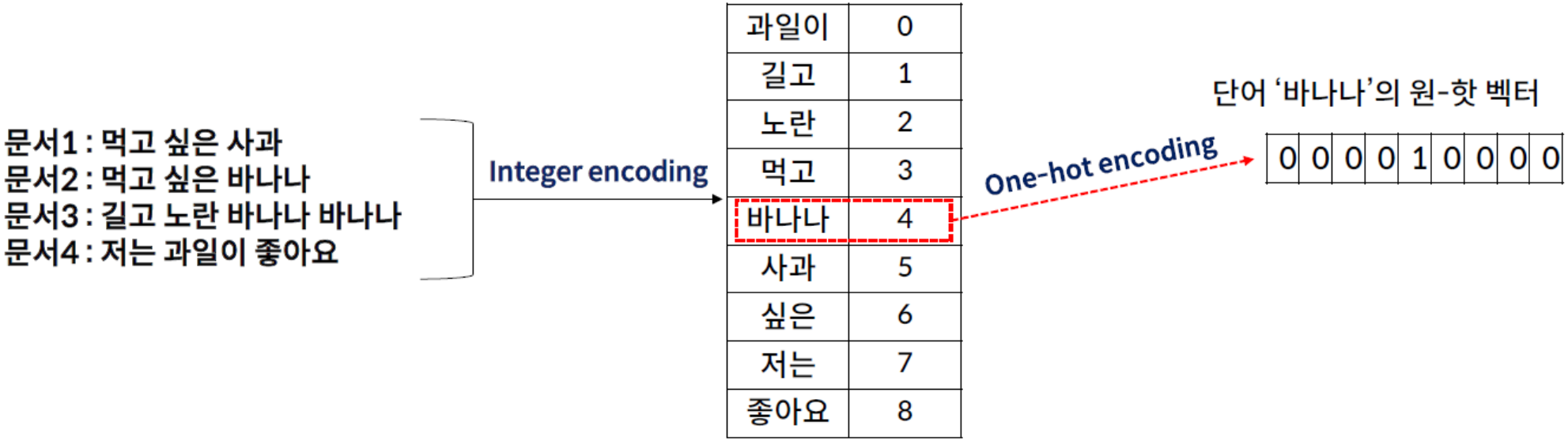
문서1 : [4, 7, 6]
문서2 : [4, 7, 5]
문서3 : [2, 3, 5, 5]
문서4 : [8, 1, 9]

0번 단어 'PAD'추가 →

문서1 : [4, 7, 6, 0]
문서2 : [4, 7, 5, 0]
문서3 : [2, 3, 5, 5]
문서4 : [8, 1, 9, 0]

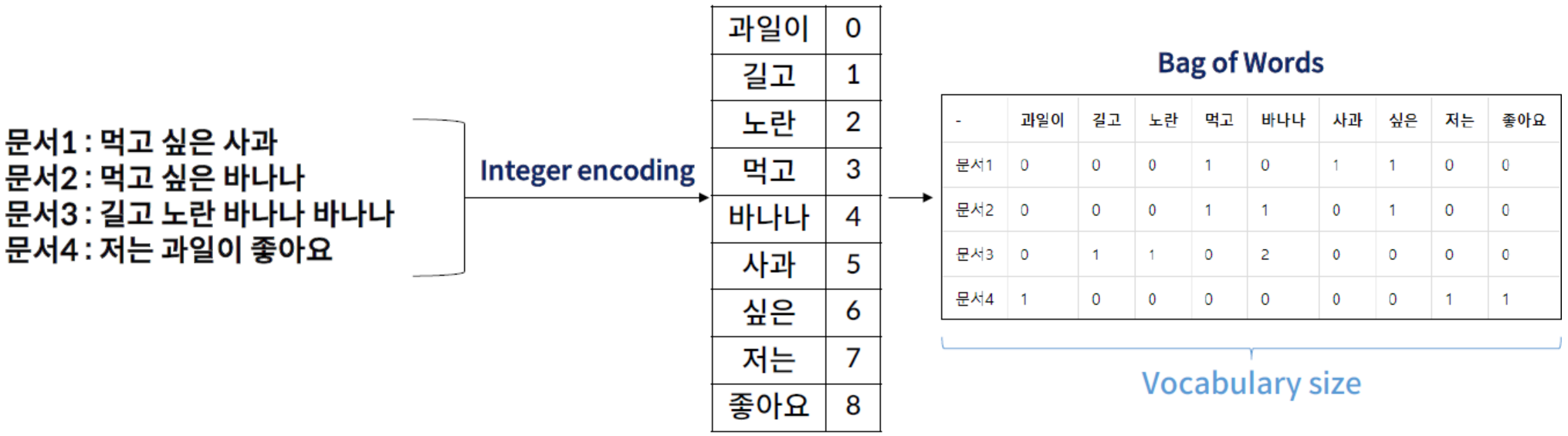
Vectorization: One-hot Encoding

- **단어(혹은 단어 index)를 벡터로 변환하는 작업**
  - 각 단어에 고유한 정수 index를 부여하고, 해당 index는 1, 나머지는 0인 벡터 생성
  - 전체 단어 집합의 크기를 벡터의 차원으로 갖는다.
  - 선형회귀(linear regression)에서 범주형 변수를 dummy variable로 변환하는 것과 유사
  - **단점:** 차원의 크기, 단어간의 관계성 파악 불가 → Word Embedding으로 이를 해결
    - 추후에 알아보자!



Vectorization: Document Term Matrix, DTM

- **문서를 vector로 변환하는 작업**
  - 문서는 단어의 집합 → 문서 vector는 단어 vector의 집합
  - 문서에서 각 단어의 등장횟수를 값으로 갖는 matrix
    - Document vectorization section에서 좀 더 알아보자!



-	과일이	길고	노란	먹고	바나나	사과	싶은	저는	좋아요
문서1	0	0	0	1	0	1	1	0	0
문서2	0	0	0	1	1	0	1	0	0
문서3	0	1	1	0	2	0	0	0	0
문서4	1	0	0	0	0	0	0	1	1

End of the documents