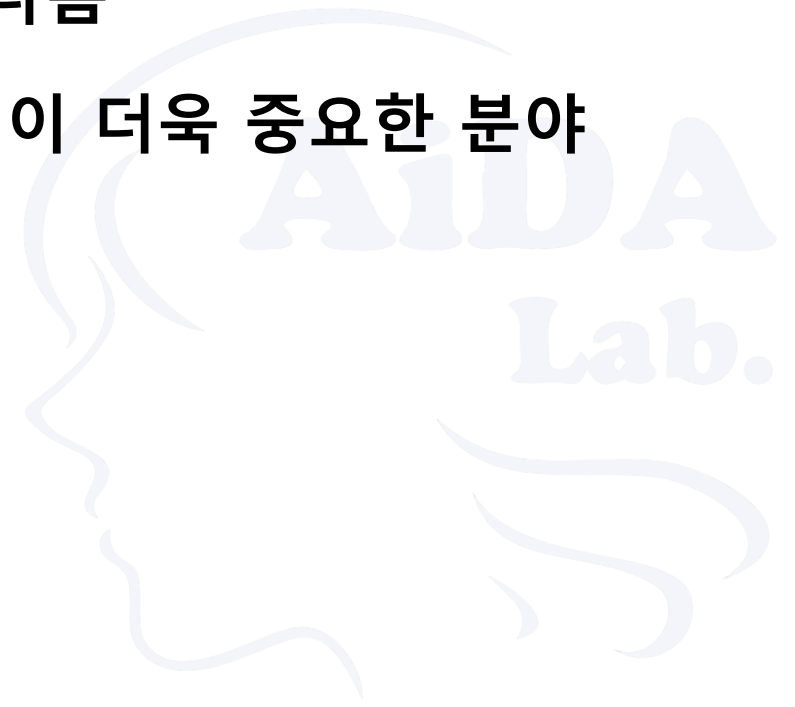


02. 전처리: 토큰화

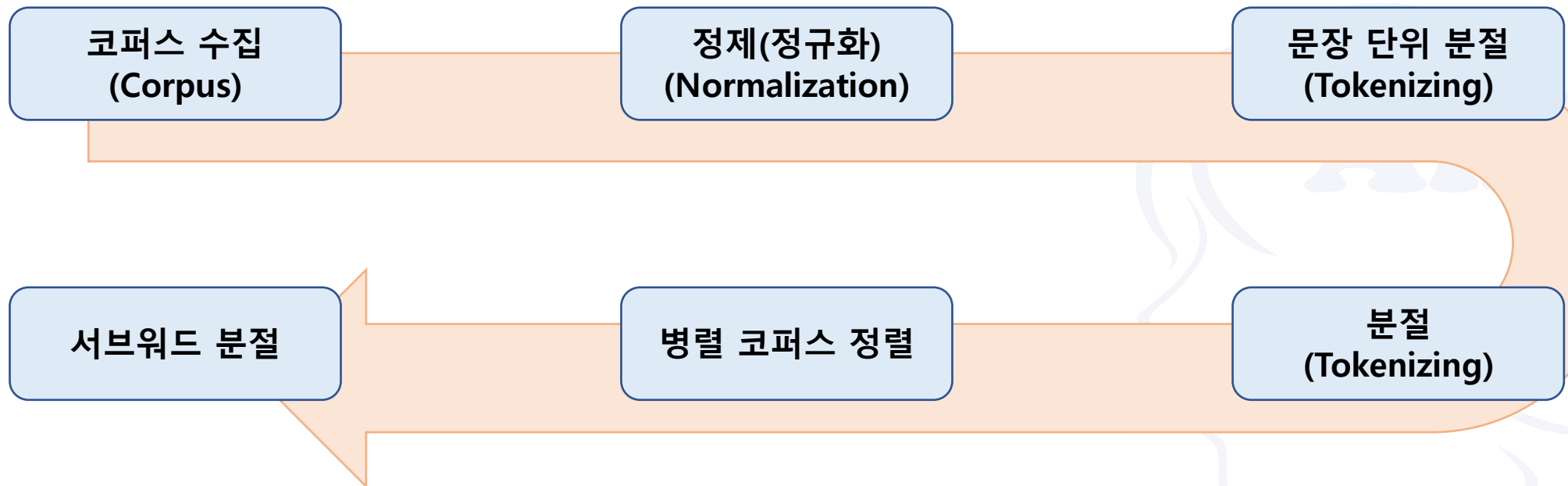


- 자연어 처리에서 가장 중요한 것은 “전처리 과정”이다
 - 외부로 드러나지 않는 과정이다 보니 상대적으로 주목도가 낮음
 - 다루고자 하는 언어, 문제에 따라 전처리 과정이 다름
 - 기술의 내용이 기반이 되어야 하지만 경험의 누적이 더욱 중요한 분야



- 일반적인 전처리 과정

- 언어, 문제에 따라 다른 전처리 과정을 거치지만 일반적으로 다음과 같음



- 여러 단어로 이루어진 문장을 가리킴
- 자연어 처리를 위하여 머신러닝/딥러닝을 수행하려면 훈련데이터가 필요하며, 다수의 문장으로 구성된 코퍼스가 훈련데이터로 사용됨
- 코퍼스가 많고 오류가 없을 수록 자연어 처리 모델은 더욱 정교해지고 정확도가 높아짐

- 코퍼스의 분류

- 구성 언어에 따른 분류

- 단일 언어 코퍼스(Monolingual Corpus)
 - 이중 언어 코퍼스(Bilingual Corpus)
 - 다중 언어 코퍼스(Multilingual Corpus)

- 구성 방법에 따른 분류

- 병렬 코퍼스(Parallel Corpus) : 각 언어가 서로 쌍으로 구성되는 코퍼스

- 예:

영문	한글
I love to go to school.	나는 학교에 가는 것을 좋아한다.
I am a doctor.	나는 의사이다.

- 코퍼스 수집

- 공개 데이터 사용하기

- 감성 분석 등의 텍스트 분류 데이터, 기계번역을 위한 언어 쌍 데이터 등 각종 대회 또는 논문을 위한 데이터가 주류
 - 데이터의 분량과 내용의 분야가 한정적

- 유료 데이터 구매하기

- 비용 문제



- 웹 크롤링

- 다양한 웹 사이트에서 크롤링 가능 → 다양한 분야의 데이터 획득 가능
- 특정 분야에 대한 자연어 처리가 아니라면 최대한 많은, 다양한 분야의 데이터가 필수
- 무분별한 웹 크롤링은 법적 문제 유발 가능, 웹 서버에 불필요한 트래픽 가중
→ 적절한 사이트에서, 올바른 방법으로, 상업적인 목적이 아닌 경우로..
와 같이 제한된 크롤링 권장

- 단일 언어 코퍼스 수집
 - 가장 손쉽게 구할 수 있는 코퍼스
 - 올바른 도메인의 코퍼스 수집 필요
 - 사용 가능한 형태로 가공하는 과정 필요
 - 위키피디아 등의 사이트에서 덤프 데이터를 제공하기도 함
 - 적법한 인터넷 사이트 등에서 다운로드 가능
 - 머신러닝 경진대회 플랫폼 사이트 “캐글” 등에서 쉽게 다운로드 가능

- 도메인 별 단일 언어 코퍼스 수집 사례

문체	도메인	수집처	정제 난이도
대화체	일반	채팅 로그	높음
대화체	일반	블로그	높음
문어체	시사	뉴스 기사	낮음
문어체	과학, 교양, 역사 등	위키피디아	중간
문어체	과학, 교양, 역사, 서브 컬처 등	나무위키	중간
대화체	일반 (각 분야별 게시판 존재)	클리앙	중간
문어체	일반, 시사 등	PGR21	중간
대화체	일반	드라마, 영화 자막	낮음

- 다중 언어 코퍼스 수집

- 다중 언어 코퍼스는 기계번역을 목적으로 하는 경우가 많음
- 단일 언어 코퍼스에 비하여 수집하기 어려움
- 자막 데이터 등은 저작권이 있는 경우가 많으므로 저작권 정보 확인 필수
- 자막 등은 번역 품질 문제가 큰 영향을 미치므로 주의 요함
- 자막 등은 대화형 언어이며, 언어 별 특징에 따른 문제도 고려 요함
 - 예: 영어 자막에서 대명사로서 지칭할 때, "아빠" → "you", "친구 명" → "you"

- 텍스트를 사용하기 위한 필수 과정
- 원하는 업무, 문제, 응용 분야에 따라 필요한 정제의 수준, 깊이 상이
 - 예시
 - 음성 인식을 위한 언어 모델: 괄호, 기호, 특수문자 등 포함 금지
 - 개인정보, 민감한 정보: 제거 또는 변조해서 모델링

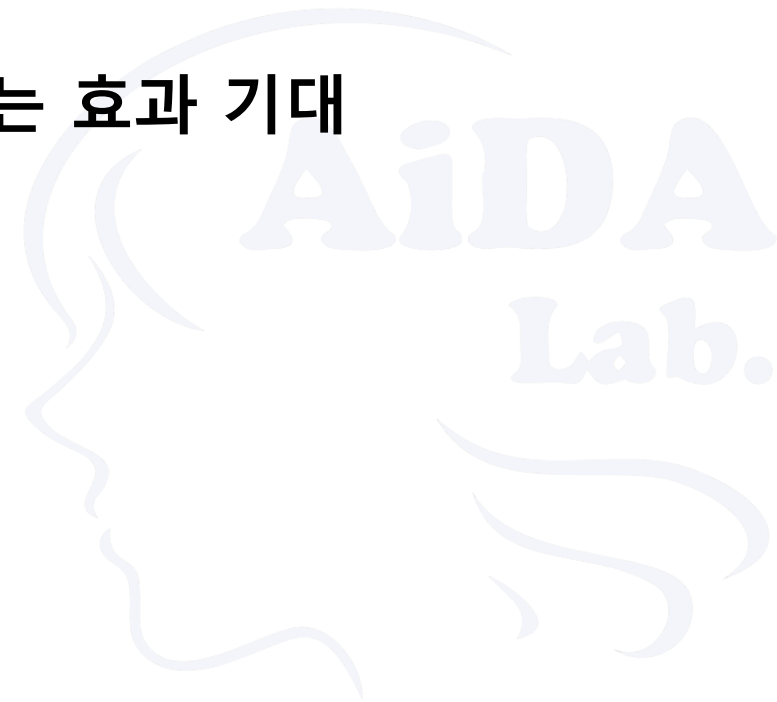


- 전각 문자 제거

- 중국어, 일본어 문서는 대부분 전각 문자로 표기됨
- 한국어 문서의 일부는 전각 문자로 표기된 기호, 숫자 등을 사용함
- 데이터 처리는 반각 문자를 기준으로 하므로 전각 문자를 반각 문자로 변형하는 작업이 요구됨

- 대소문자 통일

- 일부 영어 코퍼스에서는 약자 등에서 대소문자 표현이 통일되지 않음
 - 예: New York City 약자 → NYC, nyc, N.Y.C., N.Y.C 등
- 데이터를 하나의 형태로 통일하여 희소성을 줄이는 효과 기대
 - 희소 표현에 의한 메모리 사용량 감소 가능
- 딥러닝 모델이 확산되면서 중요도가 떨어짐



- 정규 표현식을 사용한 정제

- 다량의 코퍼스는 특수문자, 기호 등의 노이즈가 많음(특히 크롤링의 경우)
- 웹 사이트의 성격에 따라 일정한 패턴을 가지는 경우도 많음
- 정규 표현식의 사용 방법의 예
 - **[]** 사용: [2345cde] → "2 or 3 or 4 or 5 or c or d or e"의 의미
 - **-** 사용: [2-5c-e] → "2 or 3 or 4 or 5 or c or d or e"의 의미
 - **[^]** 사용: [^2-5c-e] " → 2~5, c~e를 제외한 한 글자"의 의미 (Not의 의미)
 - **()** 사용: (x)(yz) → 그룹 1:x, 그룹 2: yz (그룹을 구성할 수 있음)

• 지정 문자 사용

지정 문자	설명	예제
^	이 패턴으로 시작해야 함	^abc : abc로 시작해야 함 (abcde, abc123 등)
\$	이 패턴으로 종료되어야 함	xyz\$: xyz로 종료되어야 함 (123xyz, strxyz 등)
[문자들]	문자들 중의 하나여야 함 가능한 문자들의 집합을 정의함	[Pp]ython : "Python" 또는 "python"
[^문자들]	[문자들]의 반대 피해야 할 문자들의 집합을 정의함	[^aeiou] : 소문자 모음(母音)이 아닌 문자들
	두 패턴 중 하나여야 함 (OR 기능)	a b : a 또는 b 여야 함
?	앞 패턴이 없거나 하나여야 함 (Optional 패턴을 정의할 때 사용)	\ d? : 숫자가 하나 있거나 없어야 함
+	앞 패턴이 하나 이상이어야 함	\ d+ : 숫자가 하나 이상이어야 함
*	앞 패턴이 0개 이상이어야 함	\ d* : 숫자가 없거나 하나 이상이어야 함

정제 (Normalization, 정규화)

• 지정 문자 사용

지정 문자	설명	예제
패턴{n}	앞 패턴이 n번 반복해서 나타나는 경우	\ d{3} : 숫자가 3개 있어야 함
패턴{n,m}	앞 패턴이 최소 n번, 최대 m번 반복해서 나타나는 경우(n 또는 m은 생략가능)	\ d{3,5} : 숫자가 3개, 4개, 또는 5개 있어야 함
\ d	숫자 ([0-9]와 같음)	\ d \ d \ d : 0~9 범위의 숫자가 3개 (123, 000 등)
\ D	숫자를 제외한 모든 문자([^0-9]와 같음)	
\ s	공백 문자(white space) ([\ t \ n \ r \ f]와 같음)	\ s \ s : 공백문자 2개 (\ r \ n, \ t \ t 등)
\ S	공백 문자를 제외한 모든 문자	
\ w	alphanumeric(알파벳+숫자) + '_' 포함 ([A-Za-z0-9_]와 같음)	\ w \ w \ w : 문자가 3개 (xyz, ABC 등)
\ W	non-alphanumeric 문자 및 '_' 제외 ([^A-Za-z0-9_]와 같음)	
.	뉴 라인(\ n)을 제외한 모든 문자	.{3} : 문자가 3개 (F15, 0x0 등)

- 그룹 사용

- 정규 표현식에서 괄호는 그룹을 의미

- 예

- 전화번호 패턴

- $\backslash d\{3\}-\backslash d\{4\}-\backslash d\{4\}$: 010-1234-5678 → 일반적인 사용

- $(\backslash d\{3\})-(\backslash d\{4\}-\backslash d\{4\})$: 010-1234-5678

- 지역번호 3자리는 group(1), 나머지 번호는 group(2)로 묶어서 파라미터로 가지고 올 수 있음

- 전체 번호를 가져올 때는 group(0)으로 사용

- 정규 표현식 적용 예제

- 자연어 문제를 풀고자 한다.
- 문서의 마지막 줄에 전화번호 등의 개인 정보가 포함된 문서를 데이터 셋으로 사용하려고 한다.
- 이때, 해당 개인 정보를 제외하고 사용해야 한다.

Hello Seokhwan, I would like to introduce regular expression in this section.

~~~

Thank you!

Sincerely,

Seokhwan: +82-10-1234-5678

## • 정제 작업

- 마지막 줄에 전화번호가 있다 → 삭제 → 마지막 줄에 전화번호가 없는 경우는?? → **X**
- 다양한 상황을 고려하여야 한다

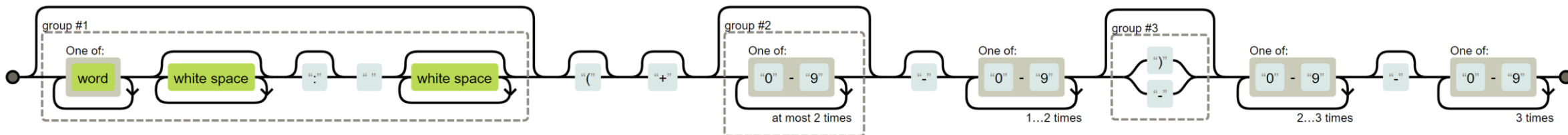
- 이름이 전화번호 앞에 나올 수도 있다.
- 이름 뒤에 콜론(:) 이 나올 수도 있다.
- 콜론 앞/뒤로는 탭 또는 공백이 다수 존재할 수도 있다.
- 전화번호는 국가번호를 포함할 수도 있다.
- 국가번호는 최대 3자리이다.
- 국가번호의 앞에는 '+'가 붙을 수도 있다.
- 전화번호 사이에 '-'가 들어갈 수도 있다.
- 전화번호는 빈칸 없이 표현된다.

- 전화번호 맨 앞과 지역번호(또는 010)의 다음에는 괄호가 들어갈 수도 있다.
- 괄호는 한쪽만 나올 수도 있다.
- 지역번호 자리의 맨 처음에 나오는 0은 빠질 수도 있다.(즉 2자리가 될 수도 있다.)
- 지역번호 다음의 번호 그룹은 3자리 또는 4자리의 숫자이다.
- 마지막은 항상 4자리 숫자이다.
- 기타 등등...

# 정제 (Normalization, 정규화)

## • 정제 작업 결과

- $([\backslash w]^+ \backslash s^*:? \backslash s^*)? \backslash (? \backslash +?([0-9]\{1,3\})? \backslash -?[0-9]\{2,3\}(\backslash | \backslash -)?[0-9]\{3,4\} \backslash -?[0-9]\{4\})$



<https://regexper.com/> 에서 디스플레이하기

**REGEXPER**  
You thought you only had two problems...

Changelog  
Documentation  
Source on GitLab

`([\w]+\s*:\s*(?!\+?([0-9]{1,3})?-\?[0-9]{2,3}(\||\-)?[0-9]{3,4}\-[0-9]{4})`

Display [Download SVG](#) // [Download PNG](#) // [Permalink](#)

Created by Jeff Avallone // Generated images licensed:

- 자연어 처리에 사용되는 데이터

- 입력단위: 기본적으로 문장 단위

- 한 줄에 한 문장 요구
- 여러 문장이 한 줄에 있거나, 한 문장이 여러 줄에 걸쳐 있는 경우  
→ 문장 단위 분절이 요구됨



- 문장 단위 분절의 기준

- 마침표(.)를 기준으로 잡으면 → U.S와 같은 약자, 3.14와 같은 소수점 등의 문제 발생
- 적절한 알고리즘 또는 모델이 필요함  
→ 직접 구현하기 어려우므로 잘 알려진 자연어 처리 툴킷, 패키지 등의 활용 권장

- 단어 단위 분절: 토큰나이징(형태소 분석)
  - 풀고자 하는 문제에 따라 형태소 분석 또는 단순 분절을 통한 정규화 수행
  - 가장 기본적인 기준은 띄어쓰기
    - 비슷한 문화권인 한국, 중국, 일본의 경우
      - 한국: 근대 이후 띄어쓰기 도입. 아직 제대로 사용하지 못하는 사람 많음
      - 중국: 띄어쓰기 사용하지 않음. 문자단위로 처리해도 그다지 차이가 없음
      - 일본: 띄어쓰기 사용하지 않음. 적절한 언어모델 구성을 위해 띄어쓰기가 요구되고 있음
    - 영어의 경우
      - 띄어쓰기가 필수. 대부분의 경우 규칙이 잘 지켜짐

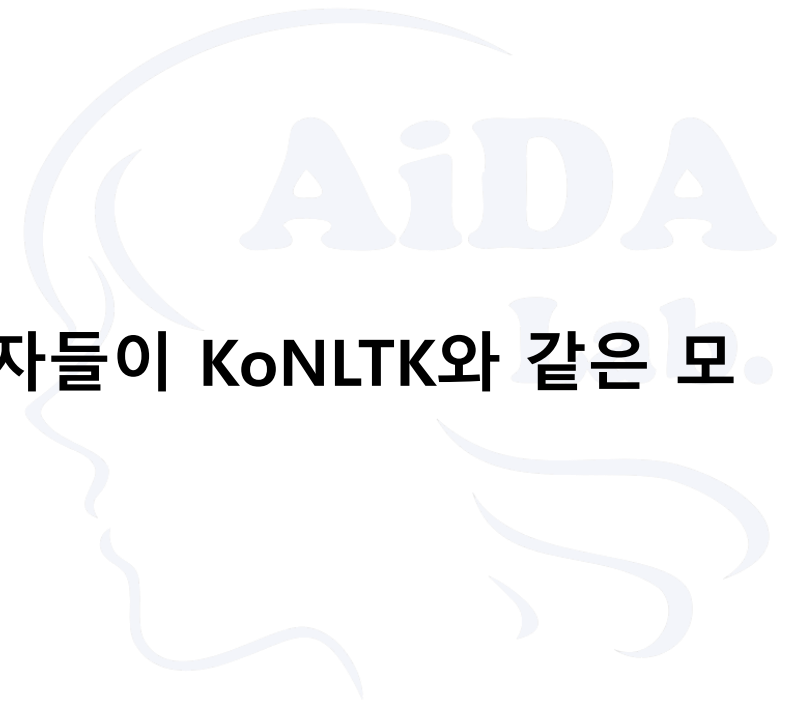
- **NLTK (Natural Language Toolkit)**

- 자연어 처리 기능을 제공하는 파이썬 라이브러리
- 텍스트로부터 단어 개수, 출현 빈도, 어휘 다양도 같은 통계적인 정보를 쉽게 얻을 수 있음
- 손쉽게 이용할 수 있도록 모듈 형식으로 기능을 제공함
- 전처리 모듈: 분류 토큰화(tokenization), 스템밍(stemming) 등
- 분석 모듈: 구문분석(parsing), 클러스터링, 감정 분석(sentiment analysis), 태깅(tagging) 등
- 추론 모듈: 시맨틱 추론(semantic reasoning) 등
- 각 모듈은 최소 2개 이상의 알고리즘을 제공하여 사용자가 원하는 알고리즘을 선택할 수 있도록 지원

(예) 분류 알고리즘의 경우 SVMs, 나이브 베이즈, 로지스틱 회귀와 결정 트리를 제공 중.  
이들 중 하나를 선택해 분류 알고리즘을 적용할 수 있음



- 문서화가 잘 되어 있음
  - 포럼이 활성화 되어 있어 개발 지원을 받을 수 있음
  - 많은 레퍼런스를 포함함
  - 속도가 다소 느림
- 
- 원래 한국어를 지원하지 않았기때문에 국내 연구자들이 KoNLTK와 같은 모듈을 개발하여 사용하였음
  - 최근 한국어도 지원 언어에 포함됨.



## • 각 언어별 주요 자연어 처리 지원 패키지

| 언어  | 패키지명            | 개발언어                | 특징                                                                                  |
|-----|-----------------|---------------------|-------------------------------------------------------------------------------------|
| 한국어 | Mecab           | C++                 | 일본어 Mecab을 Wrapping함. 속도가 가장 빠르지만 설치가 까다로움<br>정상적으로 설치를 성공한 경우, KoNLPy에서 호출하여 사용 가능 |
| 한국어 | KoNLPy          | Python Wrapping(복합) | Pip를 통해 설치 가능. 사용이 쉬움. 일부 모듈의 경우 속도가 느림                                             |
| 일본어 | Mecab           | C++                 | 속도 빠름                                                                               |
| 중국어 | Stanford Parser | Java                | 미국 스탠포드에서 개발                                                                        |
| 중국어 | PKU Parser      | Java                | 북경대에서 개발. 스탠포드 파서와 성능 차이가 거의 없음                                                     |
| 중국어 | Jieba           | Python              | 가장 최근에 개발됨. 파이썬으로 개발되어 시스템 구성에 용이함                                                  |

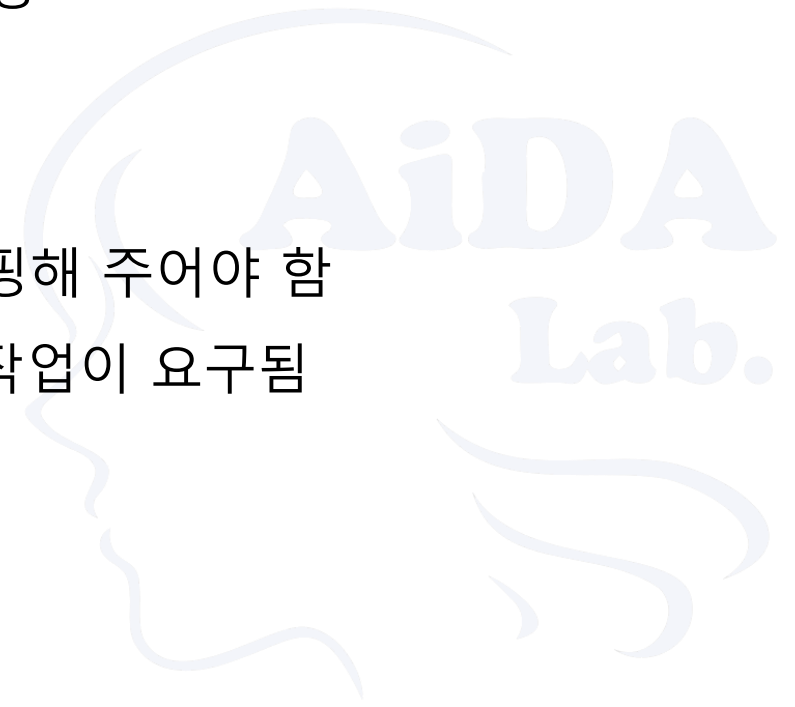
- 병렬 코퍼스

- 동일한 내용을 가진 복수 언어 말뭉치(코퍼스)
  - 예: 한국어 원문과 영어 번역문을 문장 단위로 대응시켜 DB를 구축한 것
- 데이터 자체가 복수 언어를 대상으로 만들기 때문에 언어의 대조 연구나 기계번역, 대역 사전구축, 언어 교육(작문 교육, 회화 교육) 등에 유용하게 사용됨

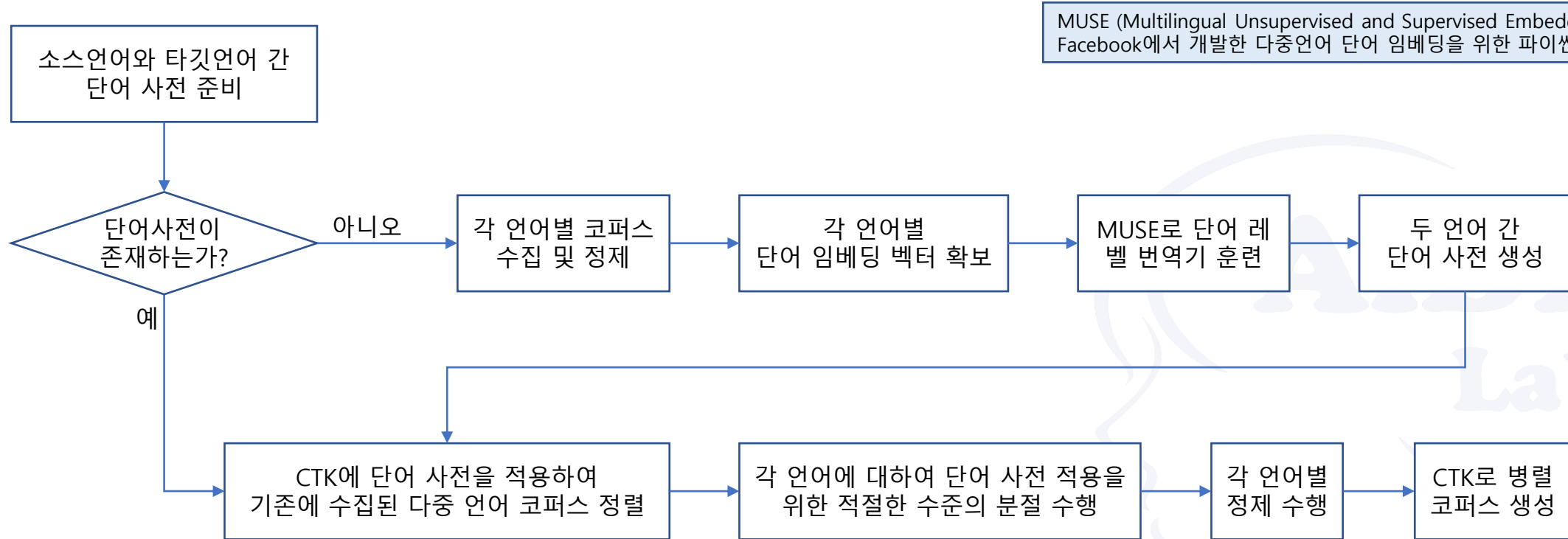
- 병렬 코퍼스 정렬

- 대부분의 병렬 코퍼스들은 “여러 문장” 단위로 정렬

- 예: 영자 신문에서 크롤링한 영문기사와 한글기사의 매핑
      - 문서 간의 매핑
      - 문장 사이의 정렬은 이루어지지 않음
      - 각 문장을 정렬하고 동일한 내용의 문장끼리 매핑해 주어야 함
      - 불필요한 내용 정리, 문장 간 정렬 재정비 등의 작업이 요구됨



## • 병렬 코퍼스 제작 프로세스



CTK (Champollion Toolkit) : 가능한 한 많은 언어 쌍에 대하여 즉시 사용할 수 있는 병렬 텍스트 문장 정렬도구를 제공하는 오픈소스 툴 키트

- 사전 생성

- 단어 사전 구축에는 많은 비용이 요구됨

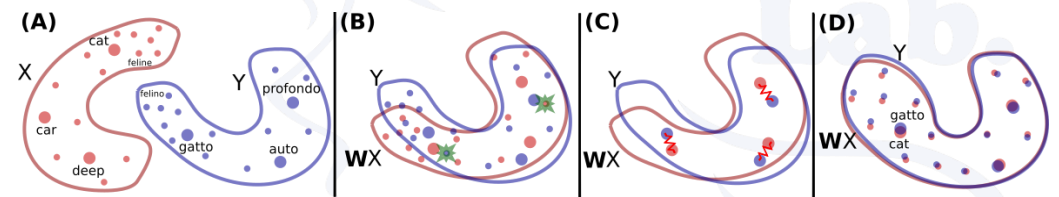
- 기 구축된 사전 활용 또는 MUSE를 이용한 단어 사전 자동 생성(구축) 활용

- MUSE: 병렬 코퍼스가 없는 상황에서 사전 구축을 위한 방법, 코드 제공

- 각 단일 언어 코퍼스를 통해 구축한 언어별 단어 임베딩 벡터에 대해 다른 언어의 임베딩 벡터와 매핑을 통하여 단어간 번역 수행

- 비지도 학습을 적용한 모델

- 구축된 사전은 CTK의 입력으로 사용됨



[https://github.com/facebookresearch/MUSE/blob/main/outline\\_all.png](https://github.com/facebookresearch/MUSE/blob/main/outline_all.png)

- MUSE를 통한 비지도 학습의 결과인 단어사전의 예

stories <> 이야기

stories <> 소설

contact <> 연락

contact <> 연락처

contact <> 접촉

green <> 녹색

green <> 초록색

green <> 빨간색

dark <> 어두운

dark <> 어둠

dark <> 짙

song <> 노래

song <> 곡

song <> 음악

salt <> 소금

- 비지도 학습이지만 꽤 정확한 단어간 번역을 확인할 수 있음
- <>을 구분 문자로 사용

AiDA  
Lab.

- CTK를 활용한 정렬

- CTK (Champollion Toolkit)

- 구축된 단어사전을 이용하여 여러 라인으로 구성된 언어별 문서에 대해 문장 정렬 수행

- 정렬 결과 예시

omitted <=> 1  
omitted <=> 2  
omitted <=> 3  
1 <=> 4  
2 <=> 5  
3 <=> 6  
4,5 <=> 7  
6 <=> 8  
7 <=> 9  
8 <=> 10  
9 <=> omitted

일대일, 일대다, 다대일 매핑 가능  
버려지는 경우도 가능

타깃언어의 1, 2, 3번째 문장: 짝을 찾지 못해서 버려짐

소스언어의 1, 2, 3번째 문장: 타깃 언어의 4, 5, 6번째 문장과 매핑

소스언어의 4, 5번째 문장: 타깃 언어의 7번째 문장과 매핑

소스언어의 6, 7, 8번째 문장: 타깃 언어의 8, 9, 10번째 문장과 매핑

소스언어의 9번째 문장: 짝을 찾지 못해서 버려짐



## • 서브워드 분절 기법

- “단어는 의미를 가진 더 작은 서브워드들의 조합으로 이루어진다”는 가정 하에 적용되는 알고리즘
- BPE(Byte Pair Encoding) 알고리즘을 기반으로 함
- 현재 시점에서 필수 전처리 방법으로 꼽힘

영어와 한국어의 서브워드 분절 사례

| 언어  | 단어          | 조합                                           |
|-----|-------------|----------------------------------------------|
| 영어  | concentrate | con(=together) + centr(=center) + ate(=make) |
| 한국어 | 집중(集中)      | 集(모을 집) + 中(가운데 중)                           |

- 서브워드 분절 효과

- 적절하게 분절하면 어휘 수 감소 및 희소성의 효과적 감소 가능
- Unknown 토큰에 대한 효율적인 대처 가능
  - 자연어 처리에서는 문장을 입력 받을 때 단어들의 시퀀스로 받아들임
  - Unknown이 발생하면 언어모델의 확률이 크게 하락하고 적절한 문장 생성이 어려워짐

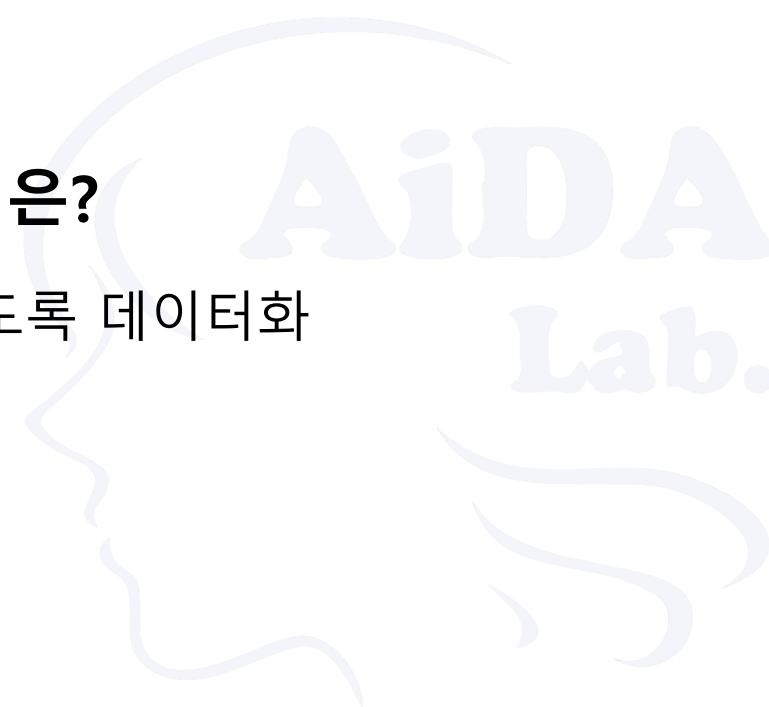
# Tokenizing



# Tokenizing(형태소 분석) 개요



- 자연어: 우리가 일상 생활에서 사용하는 언어
- 기본적으로 컴퓨터는 자연어를 이해하지 못한다
- 컴퓨터에게 자연어를 이해하게 하려면?
  - 여러 방법이 연구되어 왔으나 가장 일반적인 방법은?
    - 토큰나이징과 임베딩 기반으로 컴퓨터가 이해할 수 있도록 데이터화
    - 텍스트 유사도를 이용하여 문맥 분류



- Tokenizing이란?

- 주어진 문장에서 토큰 단위로 정보를 나누는 작업
- 문장 형태의 데이터를 처리하기 위해 제일 처음 수행해야 하는 기본적인 작업
- 주로 텍스트 전처리 과정에서 사용됨
- 토큰: 일정한 의미가 있는 가장 작은 정보 단위

- Tokenizing 과정

- 어떤 문장을 일정한 의미가 있는 가장 작은 단어로 나눈다
- 나뉜 단어들에 이용해 의미를 분석한다 (이때 가장 기본이 되는 단어가 토큰이다)

- 토큰화(Tokenizing)의 기본적인 방식
  - 단어 단위 토큰화 : 단어(어절) 단위로 토큰화
  - 문자 단위 토큰화 : 문자 단위로 토큰화
  - 서브 워드 단위 토큰화 : 서브 워드 단위로 토큰화



- 단어 단위 토큰화

- 가장 쉬운 방법은 공백으로 분리(별도의 토크나이저가 없어도 무방)

- 단점: 어휘 집합의 크기가 매우 커질 수 있음
  - "갓었어", "갓었는데요"는 서로 다른 토큰이 됨
  - 표현이 살짝만 바뀌어도 관련된 모든 경우의 수가 어휘 집합에 포함되어야 함

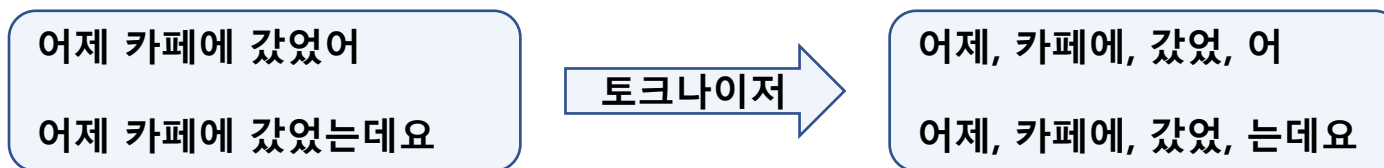
어제 카페에 갓었어  
어제 카페에 갓었는데요

토크나이저

어제, 카페에, 갓었어  
어제, 카페에, 갓었는데요

# 토큰화(Tokenizing)의 방식

- 사전 학습된 토큰라이저를 사용한다면
  - 어휘 집합의 비대화를 다소 완화 가능 (완전히 해결하기는 어려움)



- 하나의 언어로 모델을 구축할 때, 어휘 집합의 크기는 10만개를 가뿐히 넘어감
- 어휘 집합의 크기가 커질 수록 모델의 학습은 어려워짐



## • 문자 단위 토큰화

### • 한글의 경우 표현 가능한 글자는 11,172개

- 알파벳, 숫자, 기호를 모두 고려해도 어휘 집합의 크기는 15,000개 정도

### • 해당 언어의 모든 문자를 어휘 집합에 포함 → 미등록 토큰 문제 없음

### • 단점

- 각 토큰은 의미 있는 단위가 될 수 없음

어제 카페에 갔었어  
어제 카페에 갔었는데요

토큰나이저

어,제,카,페,에,갔,었,어  
어,제,카,페,에,갔,었,는,데,요

- 어미에 따른 변화, 조사의 사용 등 한글의 특징이 모두 사라짐
- 분석 결과인 토큰 시퀀스의 길이가 단어 단위 토큰화의 결과보다 상대적으로 길어짐
- 언어 모델에 입력할 토큰 시퀀스가 길면 모델의 학습이 어려워지고 결과적으로 성능하락

- 서브 워드 단위 토큰화

- 단어 단위 토큰화와 문자 단위 토큰화의 중간 형태

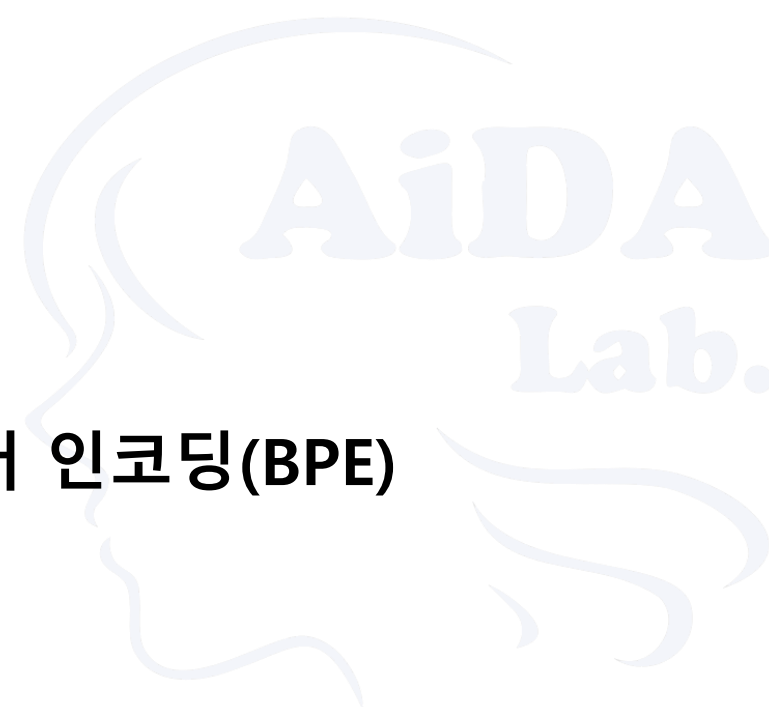
- 두 토큰화 방식의 장점만 적용

- 어휘 집합의 크기가 지나치게 커지지 않음

- 미등록 토큰 문제 회피 가능

- 분석 결과의 토큰 시퀀스가 너무 길어지지 않음

- 대표적인 서브워드 단위 토큰화 기법: 바이트 페어 인코딩(BPE)



- BPE (Byte Pair Encoding)

- 1994년 제안된 정보 압축 알고리즘
- 데이터에서 가장 많이 등장한 문자열을 병합하여 데이터를 압축하는 기법
- 데이터에 등장한 글자를 초기 사전으로 구성하여 연속된 두 글자를 한 글자로 병합하는 방식 적용
- 최근에는 자연어 처리 모델에 널리 쓰이는 토큰화 기법
  - 대표적인 활용 모델: GPT 모델

- BPE 알고리즘의 적용 예

- 초기 사전: (a, b, c, d) → 4개, 문자열: aaabdaaabac → 11자

- aaabdaaabac → aa를 Z로 병합 → ZabdZabac
- ZabdZabac → ab를 Y로 병합 → ZYdZYac
- ZYdZYac → ZY를 X로 병합 → XdXac

- BPE 수행 이후

- 사전: (a, b, c, d, Z, Y, X) → 7개
- 결과 문자열: XdXac → 5자

BPE기반 토큰화 기법은 사전 크기의 증가를 억제하면서도 정보를 효율적으로 압축할 수 있는 알고리즘이다.

BPE 어휘 집합은 고빈도 바이그램 쌍을 병합하는 방식으로 구축한다.

- BPE 알고리즘의 특징

- 분석 대상 언어에 대한 지식이 필요하지 않음
- 말뭉치(코퍼스)에서 자주 나타나는 문자열(서브 워드)을 토큰으로 분석
- 자연어 처리에서 BPE가 처음 적용된 분야는 기계번역 분야

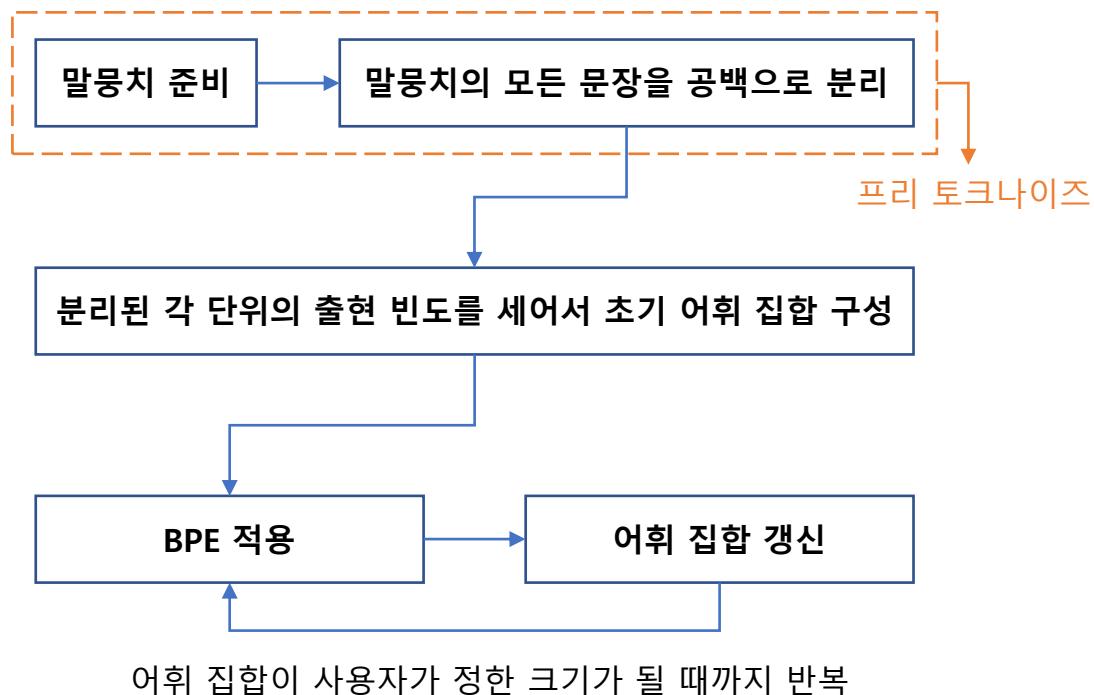
- BPE 활용 토큰화 절차

1. 어휘 집합 구축: 자주 등장하는 문자열 병합 후 어휘집합 추가 반복
2. 토큰화: 문장의 각 어절에서 어휘 집합에 있는 서브 워드를 어절에서 분리

# BPE (Byte Pair Encoding)

## • BPE 어휘 집합 구축하기

초기 어휘 집합  
b, g, h, n, p, s, u



BPE 어휘 집합 구축 결과  
b, g, h, n, p, s, u, ug, un, hug

프리 토크나이징 결과

| 토큰   | 빈도 |
|------|----|
| hug  | 10 |
| pug  | 5  |
| pun  | 12 |
| bun  | 4  |
| hugs | 5  |

초기 어휘 집합으로  
다시 작성한 빈도표

| 토큰         | 빈도 |
|------------|----|
| h, u, g    | 10 |
| p, u, g    | 5  |
| p, u, n    | 12 |
| b, u, n    | 4  |
| h, u, g, s | 5  |

바이그램 쌍으로 나열

| 토큰   | 빈도 |
|------|----|
| h, u | 10 |
| u, g | 10 |
| p, u | 5  |
| u, g | 5  |
| p, u | 12 |
| u, n | 12 |
| b, u | 4  |
| u, n | 4  |
| h, u | 5  |
| u, g | 5  |
| g, s | 5  |

u, g 병합

| 토큰       | 빈도 |
|----------|----|
| h, ug    | 10 |
| p, ug    | 5  |
| p, u, n  | 12 |
| b, u, n  | 4  |
| h, ug, s | 5  |

같은 바이그램 쌍 합치기

| 토큰   | 빈도 |
|------|----|
| b, u | 4  |
| g, s | 5  |
| h, u | 15 |
| p, u | 17 |
| u, g | 20 |
| u, n | 16 |

바이그램 쌍 빈도로 나열

| 토큰    | 빈도 |
|-------|----|
| b, u  | 4  |
| h, ug | 15 |
| p, u  | 12 |
| p, ug | 5  |
| u, n  | 16 |
| ug, s | 5  |

u, n 병합

| 토큰       | 빈도 |
|----------|----|
| h, ug    | 10 |
| p, ug    | 5  |
| p, un    | 12 |
| b, un    | 4  |
| h, ug, s | 5  |

바이그램 쌍 빈도로 나열

| 토큰    | 빈도 |
|-------|----|
| b, un | 4  |
| h, ug | 15 |
| p, ug | 5  |
| p, un | 12 |
| ug, s | 5  |

- BPE 토큰화

- 어휘 집합과 병합 우선순위를 기준으로 토큰화 수행

- 병합 우선순위

| 토큰   | 빈도 |
|------|----|
| b, u | 4  |
| g, s | 5  |
| h, u | 15 |
| p, u | 17 |
| u, g | 20 |
| u, n | 16 |

| 토큰    | 빈도 |
|-------|----|
| b, u  | 4  |
| h, ug | 15 |
| p, u  | 12 |
| p, ug | 5  |
| u, n  | 16 |
| ug, s | 5  |

| 토큰    | 빈도 |
|-------|----|
| b, un | 4  |
| h, ug | 15 |
| p, ug | 5  |
| p, un | 12 |
| ug, s | 5  |



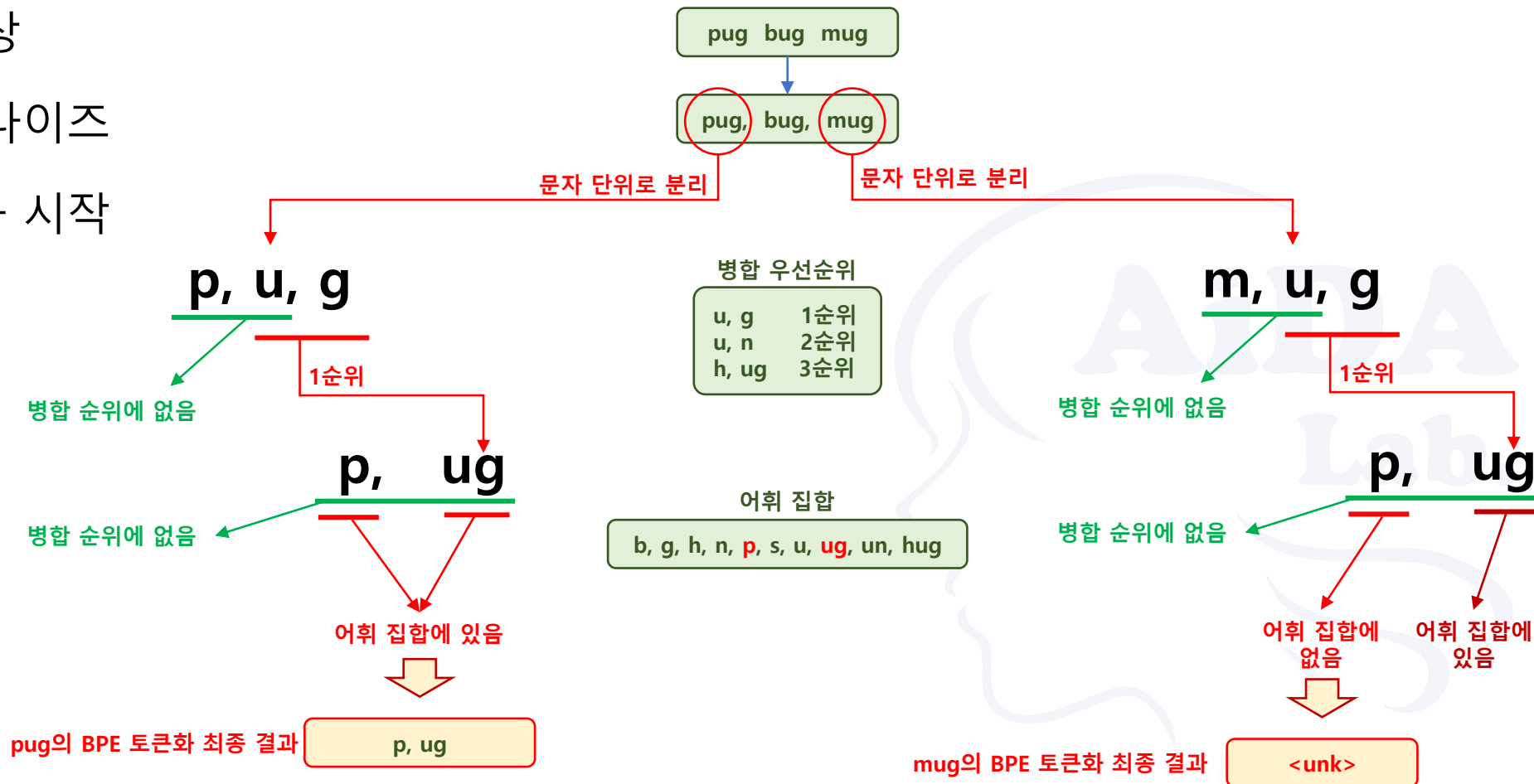
병합 우선순위

u, g 1순위  
u, n 2순위  
h, ug 3순위

# BPE (Byte Pair Encoding)

## • BPE 토큰화 예시

- 토큰화 대상
- 프리 토크나이즈
- BPE 토큰화 시작



마찬가지로 bug의 BPE 토큰화 최종 결과는 b, ug가 됨

<unk> : Unknown Token



# BPE (Byte Pair Encoding)

실습



- 한국어 Tokenizing을 구현하려면?
  - 한국어 문법에 대한 깊은 이해가 필수!!!
  - 비전공자는 어떻게 해야 하나?
    - 한국어 Tokenizing을 지원하는 파이썬 라이브러리를 사용한다.
    - 대표적인 한국어 자연어 처리 지원 라이브러리: KoNLPy (코엔엘파이)

- “토큰 단위를 어떻게 정의하느냐”가 자연어 처리 성능에 큰 영향
- 한국어 분석에서 사용하는 토큰의 단위는 “형태소”
  - 형태소란?
    - 언어학에서 사용되는 용어
    - 일정한 의미가 있는 가장 작은 말의 단위 (의미상 더 이상 쪼개지지 않는 단어)
  - 형태소를 토큰 단위로 사용한다면?
    - 단어와 품사 정보를 같이 활용할 수 있다 → 효과적 처리가 가능함

- 문장을 어떻게 형태소 단위로 분리(Tokenizing)할 수 있는가?
  - 영어의 경우는 Tokenizing이 쉽다
    - 단어의 변화가 크지 않다
    - 띄어쓰기로 단어를 구분한다
    - 따라서, 공백을 기준으로 토큰나이징을 수행해도 문제가 없다
  - 한글의 경우는 Tokenizing이 어렵다
    - 한국어는 명사와 조사를 띄어 쓰지 않는다 → 공백을 기준으로 토큰나이징 수행 불가능
    - 용언에 따라 여러 가지 어미가 붙는다 → 일정한 기준을 찾기 어렵다
    - 따라서 복잡한 특성을 고려하여 문장에서 형태소를 분석할 수 있는 "형태소 분석기" 가 필수
    - 다양한 문법적 특징을 반영하고 언어적 속성의 구조를 파악할 수 있어야 함

- 한국어 Tokenizing의 어려움의 예

- 아버지가 방에 들어가신다
  - 아버지 가방에 들어가신다
- } 컴퓨터는 이런 것을 구분하지 못함

- 형태소 분석기를 사용해서 분석한 결과

- 아버지가 방에 들어가신다.  
→ [ ('아버지', '명사'), ('가', '조사'), ('방', '명사'), ('에', '조사'), ('들어가신다', '동사(서술어)'),  
('.', '마침표(기능적인 기호')) ]

## • 한국어의 9품사

| 품사  | 설명                                                      |
|-----|---------------------------------------------------------|
| 명사  | 주로 물건이나 사람, 동식물을 가리킬 때 쓰는 품사 [강아지, 철수, 챗봇, ...]         |
| 대명사 | 사람이나 사물의 이름을 대신해서 쓰는 품사 [너, 우리, 무엇, 그것, ...]            |
| 수사  | 숫자나 순서를 나타내는 품사 [하나, 둘, 1, 2, 첫째, 둘째, ...]              |
| 동사  | 동작이나 작용을 나타내는 품사 [먹었다, 보았다, 간다, ...]                    |
| 형용사 | 사물의 성질이나 상태를 나타내는 품사 [아름답다, 맵다, 희다, ...]                |
| 관형사 | 체언(명사, 대명사, 수사) 앞에서 체언을 수식하는 품사 [이, 그, 저, 새, 헌, 옛, ...] |
| 부사  | 동사, 형용사, 동사구, 문장 전체를 수식하는 역할을 맡은 품사 [정말, 벌써, 매우, ...]   |
| 조사  | 명사, 부사 따위에 붙어 문법 관계를 맺어주는 품사 [~이, ~가, ~에서, ...]         |
| 감탄사 | 감탄이나 놀람, 느낌, 응답 등을 나타내는 품사 [까, 와, 아하, 헉, ...]           |

- **형태소 분석기**

- 복잡한 한국어 문법때문에 형태소 분석기의 개발은 매우 어렵다
- KoNLPy 등의 라이브러리 사용은 필수
  - KoNLPy 내부에서 다양한 형태소 분석기를 통합하여 라이브러리 형태로 제공
  - 다소 튜닝이 필요하지만 기본적인 성능이 뛰어난 편 → 실무에서도 많이 사용 중
- KoNLPy가 제공하는 대표적인 형태소 분석기
  - Kkma (꼬꼬마, 서울대에서 개발, GPL2 라이선스), Komoran (코모란, Shineware에서 자바로 개발, Apache 2.0 라이선스), Okt (트위터에서 개발, Apache 2.0 라이선스)
  - 통합제공하기 때문에 세 종류 모두 사용법이 거의 동일함

## • KoNLPy의 형태소 분석기 비교

| 형태소 분석기 | 장점                                                                                            | 단점                                                                                                      |
|---------|-----------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------|
| Kkma    | <ul style="list-style-type: none"><li>• 분석 품질이 좋음</li><li>• 지원하는 품사 태그가 가장 많음</li></ul>       | <ul style="list-style-type: none"><li>• 분석 속도가 느림</li><li>• 사용자 사전으로 추가한 복합 명사에 대하여 불완전하게 동작함</li></ul> |
| Komorán | <ul style="list-style-type: none"><li>• 자소가 분리된 문장이나 오타자에 강함</li><li>• 사용자 사전 관리 용이</li></ul> | <ul style="list-style-type: none"><li>• 적당한 분석 품질과 분석 속도</li></ul>                                      |
| Okt     | <ul style="list-style-type: none"><li>• 매우 빠른 분석 속도</li><li>• 정규화 기능 지원</li></ul>             | <ul style="list-style-type: none"><li>• 사용자 사전 관리가 어려움</li><li>• 용언 분석에 일관성이 부족함</li></ul>              |



## • 사용자 사전 구축

- 챗봇의 데이터 입력단은 인터넷 구어체와 관련이 많음  
→ 일반적으로 딱딱한 구어체, 문어체 등을 사용하지 않음
- 새롭게 생겨나는 단어나 문장은 형태소 분석기가 인식하지 못하는 경우 많음
  - 기존의 많은 문장을 이용하여 형태소 분석기 모델이 개발되었으므로 새로운 형태의 단어, 문장은 학습 데이터에 포함되어 있지 않음 → 인식률 저하의 원인
- 문제의 해결을 위해 대부분의 형태소 분석기는 사용자 사전을 추가할 수 있도록 구성됨