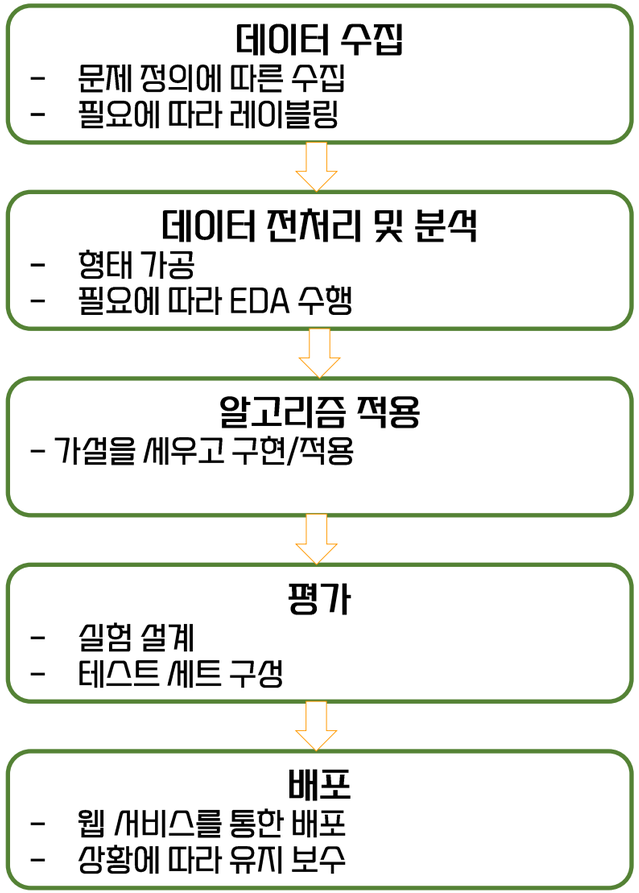
# Text Preprocessing

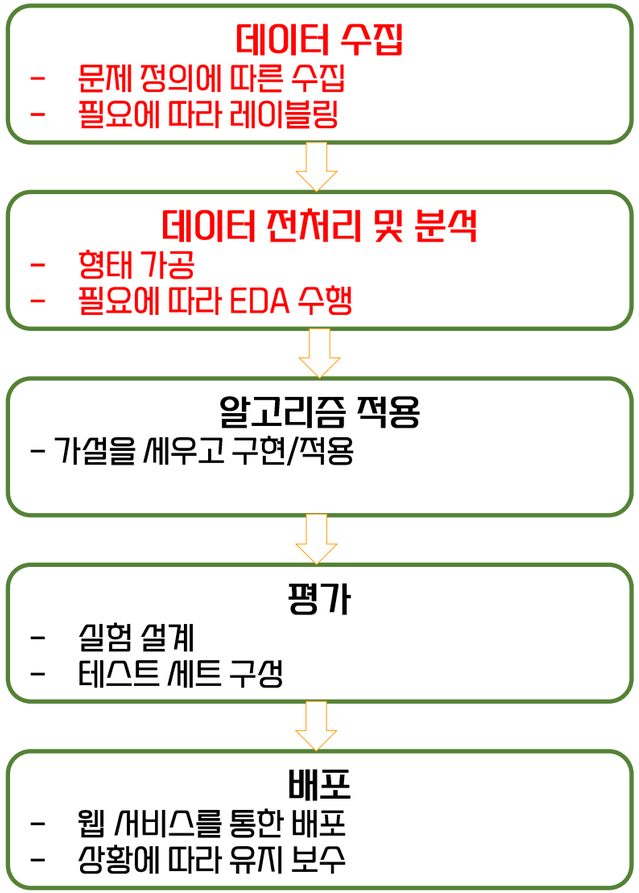
# 전처리의 늪

- 가장 재미 없고 반복적인 끝이 없는 작업입니다..
- 하지만 우리가 모델을 쓰는 것 만큼, 아니 더 중요할 수도 있는 작업이 전처리 작업입니다.

# NLP 프로젝트 Workflow



# NLP 프로젝트 Workflow



# 전처리 Workflow

**데이터(코퍼스) 수집**
- 구입, 외주, 크롤링 등

**정제(Cleaning)**
- Task에 따른 노이즈 제거
- 인코딩 변환

**레이블링(Labeling)**
- 문장 또는 단어마다 레이블링을 수행

**토큰화(Tokenization)**
- 형태소 분석기를 활용하여 분절 수행

**Subword Segmentation(선택)**
- 단어보다 더 작은 의미 단위 추가 분절 수행

**Batchify**
- 사전 생성 및 word2idx 매핑
- 효율화를 위한 전/후 처리

# 말뭉치(Corpus) 정의하기

- 말뭉치(Corpus)란 자연어 처리를 위한 문장들로 구성된 데이터 세트입니다.  여러 개의 말뭉치가 있는 경우 Corpora 라고 합니다.

- 포함된 언어의 개수에 따라서 다음과 같이 정의됩니다.
  - Monolingual Corpus ( 하나의 언어로 이루어진 Corpus )
  - Bi-Lingual Corpus ( 두 언어로 이루어진 Corpus )
  - Multilingual Corpus ( 여러 언어로 이루어진 Corpus )

- 대응되는 문장 쌍이 레이블링 되어있는 형태를 병렬 코퍼스(Parallel Corpus)라고 합니다.

| English | Korean |
|---|---|
| I am a teacher | 나는 선생님 입니다. |
| I love cat | 나는 고양이를 좋아합니다. |

| 질문 | 응답 |
|---|---|
| 나 내일 그냥 놀래 | 응 내일 놀고 모레 두고 봐 |
| 내일 뭐해? | 바빠요. |

# 서비스 Pipeline

**정제(Cleaning)**
- 학습 데이터와 같은 방식의 정제 수행

**토큰화(Tokenization)**
- 학습 데이터와 같은 방식의 분절 수행

**Subword Segmentation**
- 학습 데이터로부터 얻은 "모델"을 활용하여 똑같은 분절 수행

일반적인 모델이 아닌
subword segmentation을 수행하는 모델

**Batchify**
- 학습 데이터로부터 얻은 사전에 따른 word2idx 매핑

**Prediction**
- 모델에 넣고 추론. 필요에 따라 Search 수행(자연어 생성)

**Detokenization(선택)**
- 사람이 읽을 수 있는 형태로 변환(idx2word)
- 분절 복원

# 데이터(코퍼스) 수집

# 데이터 구입 및 외주의 한계

- 구입
  - 정제 및 레이블링이 완료된 양질의 데이터를 얻을 수 있습니다.
  - 양이 매우 제한적 입니다.
  - 구입처 : 대학교, 한국전자통신연구원(ETRI), 플리토 등

- 외주
  - 수집, 정제 및 레이블링을 외주 줄 수 있습니다.
  - 가장 높은 비용을 필요로 하며, 양이 매우 제한적일 수 있습니다.
  - 품질 관리를 위한 인력이 추가로 필요할 수 있습니다.
    - 외주 업체 관리, 공수, 아르바이트생 관리 등…

# 무료 공개 데이터

- 공개 사이트
  - AI-HUB(https://www.aihub.or.kr/)
  - WMT Competetion
  - Kaggle
  - …

- 마찬가지로 양이 매우 제한적입니다.
- 한국어 코퍼스는 흔치 않습니다.

# 크롤링

- 무한한 양의 코퍼스를 수집할 수 있습니다.
  - 원하는 도메인 별로 수집이 가능합니다.

- 하지만 품질이 천차만별이며, 정제 과정에 많은 노력이 필요할 수 있습니다.
  - 특수문자, 이모티콘, 노이즈, 띄어쓰기 등등…

# 크롤링

- 데이터에 대한 저작권이 화두로 떠오르고 있지만 아직까지는 어느정도 회색지대 입니다.
  - https://www.joongang.co.kr/article/25064802#home 참고
- 따라서 적법한 절차에 따른 크롤링은 필수입니다. 사이트의 robots.txt에 크롤링을 허용하는 url, 크롤링을 허용하지 않는 url을 게시해 놨기 때문에, 확인 후 크롤링을 해야 합니다.
- 저작권이 존재하는 코퍼스로부터 학습한 모델과 그 생성물의 저작권은 누가 갖을까요? 여기에 대한 정확한 답은 아직 없습니다. 그래서 회색지대라고 표현합니다.

# 데이터 수집처

- 아래 표는 상용화 서비스가 아닌 연구용, 학습용으로 참고하세요 ^^;;

| 수집처 | 도메인 | 문체 | 수집 난이도 | 양방향 | 정제 난이도 | 비고 |
|---|---|---|---|---|---|---|
| 블로그 | 일반 | 대화체 | 낮음 | X | 최상 | |
| 지식iN | 다양함 | 대화체 | 낮음 | X | 중간 | |
| 뉴스기사 | 시사 | 문어체 | 낮음 | O | 낮음 | 문법 준수 |
| 위키피디아 | 다양함 | 문어체 | 덤프 제공 | O | 낮음 | |
| 나무위키 | 다양함 | 문어체 | 낮음 | X | 낮음 | |
| 커뮤니티 | 다양함 | 대화체 | 중간 | X | 높음 | |
| TED | 다양함 | 대화체 | 낮음 | O | 낮음 | |
| 자막 | 일반 | 대화체 | 낮음 | O | 높음 | |

# 데이터(코퍼스) 정제

# 데이터 정제(Data Cleaning)의 두 단계

- 기계적인 노이즈 제거(무조건 제거해야 하는 노이즈)
  - 전각문자 변환.
  - 비즈니스(Task)에 따른 (전형적인) 노이즈 제거
    - 괄호 내 텍스트 제거. ex) 샹스치콤(상하이 스파이스 치킨버거 콤보라는 뜻 ㅎ) 주세요
    - 불필요한 이모티콘, 특수문자 제거. ex) 나는 오늘 기분이 나쁘다 🖤
- Interactive 노이즈 제거
  - 코퍼스의 특성에 따른 노이즈 제거
  - 작업자가 상황을 확인하며 작업 수행

# 데이터 정제(Data Cleaning)에서 주의할 점

- 비즈니스(Task)에 따른 특성
  - 풀고자 하는 문제의 특성에 따라 전처리 전략이 다릅니다.
  - 신중한 접근이 필요합니다. 예를 들면 이모티콘은 진짜 필요 없는 정보인가요?
    - 네 그렇게 하세요 ^^ vs 네 그렇게 하세요~~ vs 네 그렇게 하세요 ——
    - 배송이 진짜 빠르네요 ㅋ vs 배송이 진짜 빠르네요 ㅋㅋ vs 배송이 진짜 빠르네요 ㅋㅋㅋ

- 언어, 도메인, 코퍼스에 따른 특성
  - 각 언어, 도메인, 코퍼스 별 특성이 다르므로 다른 형태의 전처리 전략이 필요합니다.

# 전각문자 제거

- 유니코드 이전의 한글, 한자, 일본어는 전각문자로 취급 되었습니다.

- 한자 등과 함께 표기된 반각 문자로 표기 가능한 전각문자의 경우 반각문자로 치환합니다.
  - 중국어, 일본어 문서의 경우 많은 경우 전각 치환이 필요합니다.
  - 오래된 한국어 문서의 경우에도 종종 전각 치환이 필요합니다.

- 전각 문자의 예시

| 전각 | 반각 |
| --- | --- |
| ！ " ＄％ ｀ （ ） ＊ ＋ － ． ， ／ | !"$%`()*+-.,/ |
| ０１２３４５６７８９ | 0123456789 |
| ａ ｂ ｃ ｄ ｅ ｆ ｇ ｈ ｉ ｊ ｋ ｌ ｍ ｎ ｏ ｐ ｑ ｒ ｓ ｔ ｕ ｖ ｗ ｘ ｙ ｚ | abcdefghijklmnopqrstuvwxyz |
| Ａ Ｂ Ｃ Ｄ Ｅ Ｆ Ｇ Ｈ Ｉ Ｊ Ｋ Ｌ Ｍ Ｎ Ｏ Ｐ Ｑ Ｒ Ｓ Ｔ Ｕ Ｖ Ｗ Ｘ Ｙ Ｚ | ABCDEFGHIJKLMNOPQRSTUVWXYZ |

# 대소문자 통일

- 코퍼스에 따라 대소문자 표기법이 다릅니다.

- 하나의 단어를 다양하게 표현하면 희소성이 높아집니다. 하지만 딥러닝의 시대에 오면서 필요성 하락 및 생략할 수 있는 정제 방법입니다.

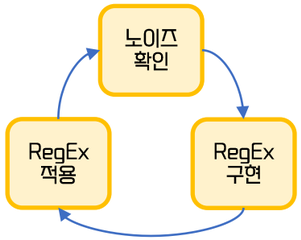
| 순번 | Republic Of Korea Army |
|---|---|
| 1 | ROKA |
| 2 | R.O.K.A |
| 3 | roka |
| 4 | Roka |
| ... | |

# 정규식(Regular Expression)을 활용한 정제

- 정규식을 활용하면 복잡한 규칙의 노이즈도 제거 / 치환이 가능합니다.
- 코딩 없이 단순히 텍스트 데이터(VSCode, Sublime Text)에서도 정규식을 활용해 정제할 수 있습니다.

# Interactive 노이즈 제거 과정

- 규칙에 의해 노이즈를 제거하기 때문에 노이즈 전부를 제거하는 것은 어렵습니다. 모든 데이터를 다 살펴 볼 수 없기 때문입니다. 따라서 반복적인 규칙 생성 및 적용 과정이 필요합니다.

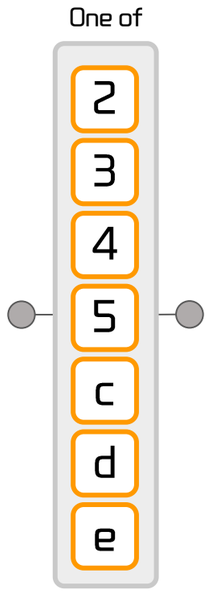



- 끝이 없는 과정
  - 노력과 품질 사이의 trade-off가 일어납니다.
  - Sweet Spot을 찾는 것이 좋습니다. 모델을 만드는 것이 목적이 되어야 하는데, 노이즈 전처리 한다고 시간을 모두 뺏길 수 없기 때문입니다.

# 정규식(Regular Expression)

# [*expr*] : One of

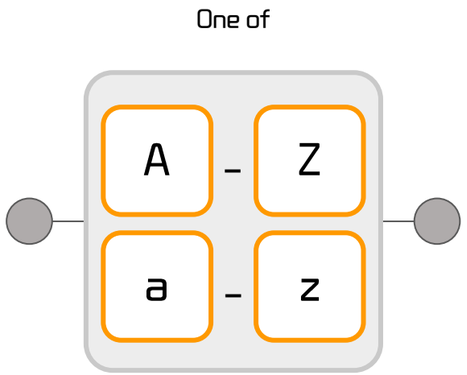
- [ ] 내의 여러 개의 문자(character) 중 한 개를 선택합니다
  - 2, 3, 4, 5, c, d, e 중 한 개의 문자
    - [2345cde]
    - (2|3|4|5|c|d|e)



# [$expr - expr$] : One of from to

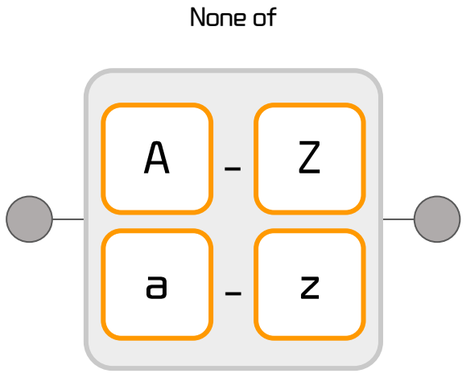
- [-] 에서 첫 글자와 마지막 글자 사이의 값을 한 개 선택합니다.
  - 대문자A부터 대문자 Z
    - [A-Z]
  - 대문자 A부터 대문자 Z, 소문자 a부터 소문자 z
    - [A-Za-z]
  - 한글 전체
    - [ㄱ-ㅎㅏ-ㅣ가-힣]

- 띄어쓰기도 인식하기 때문에 공백도 주의합시다!



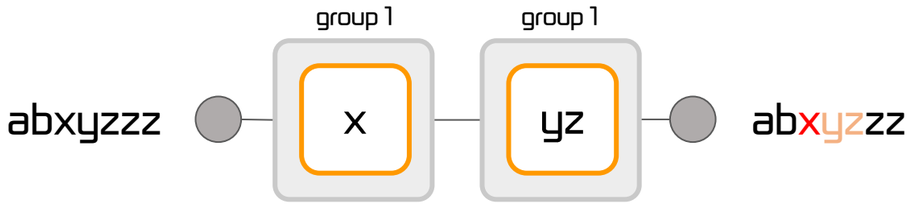
# [^*expr*] : None of

- [^expr] 에서 expr을 제외한 모든 문자를 선택합니다
  - 영어가 아닌 문자
    - [^A-Za-z]
  - 숫자와 영어가 아닌 문자
    - [^0-9a-zA-Z]



# (*expr*) : Group

- 문자열을 정규화 시키기 위해 괄호를 기준으로 끊어줍니다.
  - x를 1번 그룹으로, yz를 2번 그룹으로 지정하기
    - (x)(yz)



# 정규식 예시

- 양 끝에 소문자 알파벳으로 둘러쌓인 “bc” 제거하기
  - abcd : a, d로 둘러 쌓여있기 때문에 제거 가능
  - 0bc1 : 0, 1 숫자로 둘러 쌓여있기 때문에 제거 불가

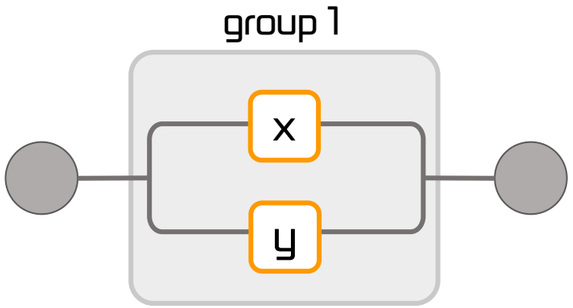
# 정규식 예시

- 양 끝에 소문자 알파벳으로 둘러쌓인 “bc” 제거하기
  - abcd : a, d로 둘러 쌓여있기 때문에 제거 가능
  - 0bc1 : 0, 1 숫자로 둘러 쌓여있기 때문에 제거 불가

```
([a-z])bc([a-z])
```

# | : Or

- |(or) 기호를 이용해 여러 개 중 하나를 구현할 수 있습니다.
  - x 또는 y가 나타나면 그룹 1번에 지정
    - (x|y)

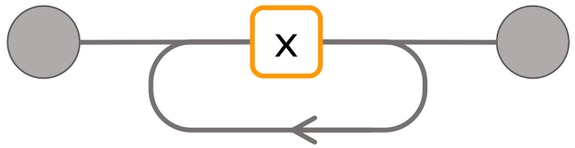


# ? : exists

- ? 기호를 이용해 0번 또는 1번 나타난 문자를 검사할 수 있습니다.
  - x가 0번 또는 1번 나타남
    - x?

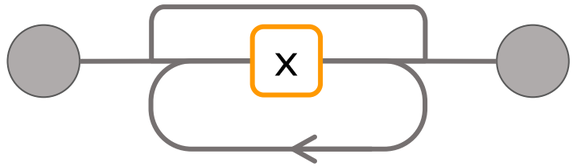
# + : least once

- + 기호를 1번 이상 나타난 문자를 검사할 수 있습니다.
  - x가 1번 이상 나타남
    - x+



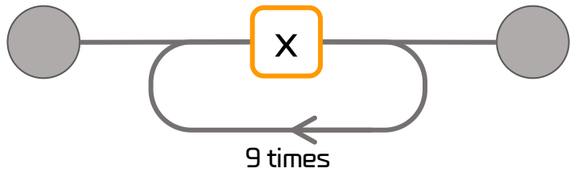
# * : greedy

- * 기호를 이용해 0번 이상 나타난 문자를 검사할 수 있습니다.
- 강력한 표현으로 유의해서 사용해야 합니다.
  - x가 0번 이상 나타남
    - x*



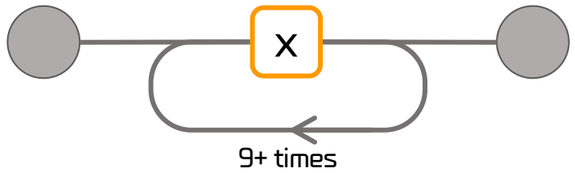
# $\{n\}, \{n,\}, \{n,m\}$ : n times, least n times, n to m times

- n회 반복을 표현하기 위해 사용합니다.
  - x가 9회 반복
    - x{n}



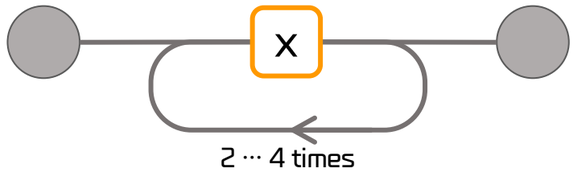
# $\{n\}, \{n,\}, \{n,m\}$ : n times, least n times, n to m times

- n회 반복을 표현하기 위해 사용합니다.
  - x가 9회 이상 반복
    - x{9,}



# $\{n\}, \{n,\}, \{n,m\}$ : n times, least n times, n to m times

- n회 반복을 표현하기 위해 사용합니다.
  - x가 2회 이상 5회 미만 반복
    - x{2, 5}



# . : Any Character

- 그냥 . 만 찍으면 모든 문자를 의미합니다.
- 매우 강력한 표현으로, 유의해서 사용해야 합니다.

# ^ $ : Start of Line, End of Line

- 문장의 시작과 끝을 표시합니다.
- x로 문장이 시작해서 x로 끝나는 경우
  - ^x$

# 코퍼스 레이블링

# 레이블

- 텍스트 분류(Text Classification)
  - 감성 분석 등
  - 입력 데이터 : 문장
  - 출력 : Class
- 토큰 분류(Token Classification)
  - 형태소 분석 등
  - 입력 데이터 : 문장
  - 출력 : 토큰 별 태그. 각 토큰이 어떤 태그인지 Sequence 형식으로 등장할 수 있다 .
- Sequence-to-Sequence
  - 입력 데이터 : 문장
  - 출력 데이터 : 문장

# 레이블 예시

- x : 문장, y : 클래스
  - TSV(Tab Separated Value) 형태의 하나의 파일로 보통 구성이 됩니다. CSV로도 구성이 됩니다.
  - 각 row가 문장과 대응되는 레이블로 구성되며, 문장 컬럼과 레이블 컬럼으로 구성됩니다.

- x : 문장, y : 문장
  - TSV 형태의 하나의 파일로 구성됩니다.
  - 각 row가 대응되는 문장 쌍으로 구성되며, 각 문장별로 컬럼이 구성됩니다.
  - 두 개 이상의 파일로 구성되는 경우도 있으며 같은 순서의 row가 대응되는 문장 쌍이 됩니다.

# Human Labeling

- 사람이 직접 레이블링 하는 것에 거부감을 느낄 필요는 없습니다. 물론 데이터 수집 시에 레이블링 된 문장을 얻을 수 있다면 훨씬 괜찮겠지만, 매번 원하는 대로 데이터를 수집하기엔 매우 힘듭니다.

- 엑셀, 판다스 등 효율적으로 레이블링할 수 있는 도구들을 많이 사용하는 것이 좋습니다.

# Tokenization

# Sentence Segmentation

- 일반적으로 훈련에 사용하는 입력 데이터는 다음과 같은 형태를 띕니다.

  “I’m hurting baby”

  “I’m broken down”

  “I need your loving, loving I need it now”

- 하지만 우리가 수집한 코퍼스는 다음과 같은 형태를 띌 수도 있습니다.

  “I’m hurting baby. I’m broken down”

  “I need”

  “your loving, loving”

  “I need it now”

# Sentence Segmentation

- 일반적으로 훈련에 사용하는 입력 데이터는 다음과 같은 형태를 띕니다.

  “I’m hurting baby”

  “I’m broken down”

  “I need your loving, loving I need it now”

- 하지만 우리가 수집한 코퍼스는 다음과 같은 형태를 띌 수도 있습니다.

  “I’m hurting baby. I’m broken down” 한 라인에 여러 문장이 들어있는 경우

  “I need”

  “your loving, loving”

  “I need it now” 한 문장이 여러 라인에 걸쳐 들어있는 경우

문장 나누기(Sentence Segmentation)을 할 때 주의해야 할 점을 알아봅니다.

# Sentence Segmentation

- 일반적으로 훈련에 사용하는 입력 데이터는 다음과 같은 형태를 띕니다.

  “I’m hurting baby”

  “I’m broken down”

  “I need your loving, loving I need it now”

- 하지만 우리가 수집한 코퍼스는 다음과 같은 형태를 띌 수도 있습니다.

  “I’m hurting baby. I’m broken down” 한 라인에 여러 문장이 들어있는 경우

  “I need”

  “your loving, loving”

  “I need it now” 한 문장이 여러 라인에 걸쳐 들어있는 경우

- Sentence Segmentation을 통해 원하는 형태로 변환시켜 줍니다.
  - 하지만 무조건 마침표를 이용해서 문장 나누기(Sentence Segmentation)을 하는 것은 위험합니다.
  - 일반적으로는 영어는 NLTK, spaCy, 한국어는 KSS(Korean Sentence Spliter)를 사용합니다.

# Word Tokenization

- Tokenization의 목적은 두 개 이상의 다른 토큰들의 결합으로 이루어진 단어를 쪼개어, 단어 집합의 숫자를 줄이고, 희소성을 낮추기 위함 입니다. 단순히 띄어쓰기가 아닌, 여러 규칙을 적용해 문장 내 단어를 쪼갭니다.

- 특히 한국어 같은 경우 교착어라는 특징 때문에 어근에 접사가 붙어 다양한 단어가 파생됩니다. 따라서 희소성에 있어서 다른 언어에 비해 매우 불리합니다.

# 토큰화와 띄어쓰기

- 영어 : 띄어쓰기가 이미 잘 되어 있습니다. NLTK를 사용하여 Comma 등을 후처리 하면 됩니다.
- 중국어 : 기본적인 띄어쓰기가 존재하지 않습니다. character 단위로 사용해도 무방합니다.
- 일본어 : 기본적인 띄어쓰기가 없습니다.
- 한국어 : 한국어는 띄어쓰기가 도입된지 얼마 되지 않기 때문에, 사용자 마다 띄어쓰기를 다르게 사용하는 경우가 많습니다. 따라서 띄어쓰기 교정기나 문법 교정기를 통해 띄어쓰기를 보정한 후 토큰화 하는 것이 좋습니다.

# 형태소 분석 및 품사 태깅

- 형태소 분석이란 형태소를 비롯하여 어근, 접두사/접미사, 품사(POS, part-of-speech) 등 다양한 언어적 속성의 구조를 파악하는 것을 의미합니다.

- 품사 태깅이란 형태소의 뜻과 문맥을 고려하여 그것에 마크업을 하는 일입니다.

# 토큰화 방법에 따른 특징

# 토큰 평균 길이에 따른 성격과 특징

- 형태소 분석기에 따라 다른 분절이 만들어 질 수 있고, 어떤 형태소 분석기를 쓰는지에 따라서 잘게 쪼개어 지거나, 더 길게 쪼개어 집니다.

- 토큰의 길이가 짧을 수록 다음과 같은 특징을 지닙니다.
  - Vocabulary 크기가 감소합니다. 이는 희소성 문제가 감소합니다. 이에 따라 OOV가 줄어듭니다.
  - 시퀀스의 길이가 길어지기 때문에 모델의 부담이 증가합니다.
  - 극단적으로 문자(character)단위가 될 수 있습니다.

- 반대로 토큰의 길이가 길 수록 다음과 같은 특징을 지닙니다.
  - Vocabulary 크기가 증가하고, 이는 희소성 문제가 증대됩니다. 이에 따라 OOV도 증가됩니다.
  - Sequence의 길이가 짧아져 모델의 부담이 감소됩니다.

# 정보량에 따른 이상적인 형태

- 빈도가 높을 경우 하나의 토큰으로 나타낼 수 있습니다.
  - “나는 밥을 먹는다” 라는 문장에 100만개의 문장에서 50만개의 문장에서 존재한다면?
    - “나는 밥을 먹는다” 라는 문장을 하나의 토큰으로 처리!
- 빈도가 낮을 경우 더 잘게 쪼개어 각각 빈도가 높은 토큰으로 구성이 가능합니다.
  - “까르보불닭치킨”, “까르보나라”, “불닭”, “치킨” → “까르보”, “나라”, “불닭”, “치킨”
- 압축 알고리즘을 떠올려 볼 수 있겠습니다.

# 서브워드(subword) 토큰화

# 단어보다 더 작은 의미 단위

- 많은 언어들에서 단어는 더 작은 의미 단위들이 모여 구성됩니다.

| 언어 | 단어 | 조합 |
|---|---|---|
| 영어 | Preprocess | pre+process |
| 한국어 | 전처리 | 前(앞 전) + 處理(곳 처, 다스릴 리) |

- preprocess의 subword는 [pre, process], 전처리의 subword는 [전, 처리] 등으로 분절할 수 있습니다. 이렇게 작은 의미 단위로 분절할 수 있다면 작은 집합으로 많은 단어들을 표현할 수 있습니다.

- 하지만 이를 위해서는 언어별 subword 사전이 존재해야 합니다.

# Byte Pair Encoding (BPE) 알고리즘

- 압축 알고리즘을 활용하여 subword segmentation을 적용합니다.
- 학습 코퍼스(Training Corpus)를 활용하여 BPE 모델을 학습한 이후에 학습과 테스트 코퍼스에 적용시킵니다.
- 장점
  - 희소성을 통계에 기반하여 효과적으로 낮출 수 있습니다.
  - 언어별 특성에 대한 정보 없이, 더 작은 의미 단위로 분절 할 수 있습니다.
  - OOV를 없앨 수 있습니다. 단, 학습시에 사용되는 문자로만 구성된 경우 입니다.
    - 예를 들어 한국어로 되어 있는 코퍼스를 이용해서 학습 했는데, 갑자기 아랍 코퍼스가 들어오면 어쩔 수 없이 OOV가 등장하게 됩니다.
- 단점
  - 학습 데이터 별로 BPE 모델도 따로 생성되어야 합니다.

# BPE 학습과 적용

- BPE 학습
  1. 단어 사전 생성( 빈도 포함 )
  2. 문자(Character) 단위로 분절 후, pair 별 빈도 카운트
  3. 최다 빈도 pair를 골라 merge(병합) 수행
  4. pair 별 빈도 카운트 업데이트
  5. 3번 과정 반복

- BPE 적용
  1. 각 단어를 문자(Character) 단위로 분절
  2. 단어 내에서 “학습 과정에서 병합(merge)에 활용된 pair의 순서대로” 병합 수행

# BPE 학습 예시

vocab {'l o w 〈/w〉':5,'l o w e r 〈/w〉':2,'n e w e s t 〈/w〉':6,'w i d e s t 〈/w〉':3}
pairs (l,o):7,(o,w):7,(w,〈/w〉):5,(w,e):8,(e,r):2,(r,〈/w〉):2,(n,e):6,(e,w):6,(e,s):9,(s,t):9,(t,〈/w〉):9,(w,i):3,(i,d):3,(d,e):3
**best pair ('e','s')9**

vocab {'l o w 〈/w〉':5,'l o w e r 〈/w〉':2,'n e w es t 〈/w〉':6,'w i d es t 〈/w〉':3}
pairs (l,o):7,(o,w):7,(w,〈/w〉):5,(w,e):2,(e,r):2,(r,〈/w〉):2,(n,e):6,(e,w):6,(w,es):6,(es,t):9,(t,〈/w〉):9,(w,i):3,(i,d):3,(d,es):3
**best pair ('es','t')9**

vocab {'l o w 〈/w〉':5,'l o w e r 〈/w〉':2,'n e w est 〈/w〉':6,'w i d est 〈/w〉':3}
pairs (l,o):7,(o,w):7,(w,〈/w〉):5,(w,e):2,(e,r):2,(r,〈/w〉):2,(n,e):6,(e,w):6,(w,est):6,(est,〈/w〉):9,(w,i):3,(i,d):3,(d,est):3
**best pair ('est','〈/w〉')9**

vocab {'l o w 〈/w〉':5,'l o w e r 〈/w〉':2,'n e w est〈/w〉':6,'w i d est〈/w〉':3}
pairs (l,o):7,(o,w):7,(w,〈/w〉):5,(w,e):2,(e,r):2,(r,〈/w〉):2,(n,e):6,(e,w):6,(w,est〈/w〉):6,(w,i):3,(i,d):3,(d,est〈/w〉):3
**best pair ('l','o')7**

vocab {'lo w 〈/w〉':5,'lo w e r 〈/w〉':2,'n e w est〈/w〉':6,'w i d est〈/w〉':3}
pairs (lo,w):7,(w,〈/w〉):5,(w,e):2,(e,r):2,(r,〈/w〉):2,(n,e):6,(e,w):6,(w,est〈/w〉):6,(w,i):3,(i,d):3,(d,est〈/w〉):3
**best pair ('lo','w')7**

# BPE 학습 예시

vocab {'low 〈/w〉':5,'low e r 〈/w〉':2,'n e w est〈/w〉':6,'w i d est〈/w〉':3}
pairs (low,〈/w〉):5,(low,e):2,(e,r):2,(r,〈/w〉):2,(n,e):6,(e,w):6,(w,est〈/w〉):6,(w,i):3,(i,d):3,(d,est〈/w〉):3
**best pair ('n','e')6**

vocab {'low 〈/w〉':5,'low e r 〈/w〉':2,'ne w est〈/w〉':6,'w i d est〈/w〉':3}
pairs (low,〈/w〉):5,(low,e):2,(e,r):2,(r,〈/w〉):2,(ne,w):6,(w,est〈/w〉):6,(w,i):3,(i,d):3,(d,est〈/w〉):3
**best pair ('ne','w')6**

vocab {'low 〈/w〉':5,'low e r 〈/w〉':2,'new est〈/w〉':6,'w i d est〈/w〉':3}
pairs (low,〈/w〉):5,(low,e):2,(e,r):2,(r,〈/w〉):2,(new,est〈/w〉):6,(w,i):3,(i,d):3,(d,est〈/w〉):3
**best pair ('new','est〈/w〉')6**

vocab {'low 〈/w〉':5,'low e r 〈/w〉':2,'newest〈/w〉':6,'w i d est〈/w〉':3}
pairs (low,〈/w〉):5,(low,e):2,(e,r):2,(r,〈/w〉):2,(w,i):3,(i,d):3,(d,est〈/w〉):3
**best pair ('low','〈/w〉')5**

vocab {'low〈/w〉':5,'low e r 〈/w〉':2,'newest〈/w〉':6,'w i d est〈/w〉':3}
pairs (low,e):2,(e,r):2,(r,〈/w〉):2,(w,i):3,(i,d):3,(d,est〈/w〉):3
**best pair ('w','i')3**

vocab {'low〈/w〉':5,'low e r 〈/w〉':2,'newest〈/w〉':6,'wi d est〈/w〉':3}

# Segmentation 예시

- latest news
  1) l a t e s t 〈/w〉 n e w s 〈/w〉
  2) l a t es t〈/w〉 n e w s 〈/w〉
  3) l a t est 〈/w〉 n e w s 〈/w〉
  4) l a t est〈/w〉 n e w s 〈/w〉
  5) l a t est〈/w〉 ne w s 〈/w〉
  6) l a t est〈/w〉 new s 〈/w〉

- merged pair
  1) ('e','s')
  2) ('es','t')
  3) ('est','〈/w〉')
  4) ('l','o')
  5) ('lo','w')
  6) ('n','e')
  7) ('ne','w')
  8) ('new','est〈/w〉')
  9) ('low','〈/w〉')
  10) ('w','i')

# Segmentation 예시

- latest news
  1) l a t e s t 〈/w〉 n e w s 〈/w〉
  2) l a t es t〈/w〉 n e w s 〈/w〉
  3) l a t est 〈/w〉 n e w s 〈/w〉
  4) l a t est〈/w〉 n e w s 〈/w〉
  5) l a t est〈/w〉 ne w s 〈/w〉
  6) l a t est〈/w〉 new s 〈/w〉

Segmentation 결과 : {l, a, t, est〈/w〉, new, s, 〈/w〉}

- merged pair
  1) ('e','s')
  2) ('es','t')
  3) ('est','〈/w〉')
  4) ('l','o')
  5) ('lo','w')
  6) ('n','e')
  7) ('ne','w')
  8) ('new','est〈/w〉')
  9) ('low','〈/w〉')
  10) ('w','i')

# Subword Segmentation Modules

- Subword-nmt
- WordPiece
- SentencePiece