

4장 전처리

2022-09-29

목차

- 4.1 전처리
- 4.2 코퍼스 수집
- 4.3 정제
- 4.4 문장 단위 분절
- 4.5 분절
- 4.6 병렬 코퍼스 정렬
- 4.7 서브 워드 분절
- 4.8 분절 복원
- 4.9 토치 텍스트

2022-09-29

4.1 전처리

4.1.1 코퍼스란?

코퍼스란?

- '말뭉치' 라고도 불리는 코퍼스corpus는 보통 여러 단어들로 이루어진 문장이다.
- 자연어처리 분야의 머신러닝에 필요한 훈련 데이터 역할
- 코퍼스의 종류는 **단일 언어 monolingual 코퍼스** (한 가지 언어), **이중 언어 bilingual 코퍼스**(두 개의 언어), **다중 언어 multilingual 코퍼스** (두개 이상의 더 많은 수의 언어), 언어 간에 쌍으로 구성되는 코퍼스를 **병렬 parallel 코퍼스**가 있다.

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

▶ 병렬 코퍼스 사례

2022-09-29



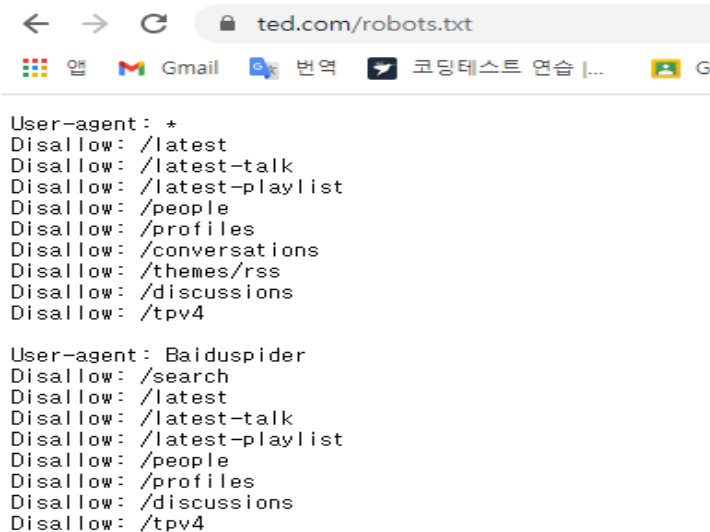
4.1.2 전처리 과정 개요



▶ 전처리 과정 개요도

4.2 코퍼스 수집

- 공개된 데이터 (감성 분석, 텍스트 분류 문제 또는, 기계 번역 문제 등을 위한 데이터), 데이터 구매 등 데이터를 구하는 방법은 많지만 한정적이다. 그래서 이 책에서는 크롤링(crawling)을 이용해서 데이터를 수집하였다.
- 해당 웹사이트의 크롤링(crawling) 허용 여부는 사이트의 robots.txt를 통해서 확인 가능하다.

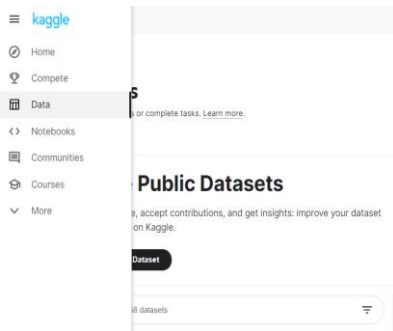


```
User-agent: *
Disallow: /latest
Disallow: /latest-talk
Disallow: /latest-playlist
Disallow: /people
Disallow: /profiles
Disallow: /conversations
Disallow: /themes/rss
Disallow: /discussions
Disallow: /tpv4

User-agent: Baiduspider
Disallow: /search
Disallow: /latest
Disallow: /latest-talk
Disallow: /latest-playlist
Disallow: /people
Disallow: /profiles
Disallow: /discussions
Disallow: /tpv4
```

- 크롤링(crawling)할 때는 selenium이나 beautiful-soup을 이용해서 웹 브라우저 드라이브를 직접 컨트롤하거나 HTML 코드를 쉽고 간단하게 파싱 할 수 있다.
- 왼쪽 이미지는 robots.txt를 통해서 ted 웹 사이트의 크롤링 (crawling)은 모든 User-agent에서 일부 경우에 한해 disallow인 것을 확인할 수 있다.

4.2.1 단일 코퍼스 수집



머신 러닝 경진대회 플랫폼
사이트 캐글(Kaggle)

Index of /kowiki/

20200820/	02-Nov-2020 01:31	-
20201001/	21-Nov-2020 01:43	-
20201030/	02-Dec-2020 01:30	-
20201101/	23-Dec-2020 01:28	-
20201120/	02-Jan-2021 01:32	-
20201201/	04-Dec-2020 06:43	-
20201220/	22-Dec-2020 03:15	-
20210101/	03-Jan-2021 15:04	-
latest/	03-Jan-2021 15:03	-

위키피디아나 각종 위키의
덤프 데이터

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

▶ 도메인별 단일 언어 코퍼스 수집 사례

필요에 따라 올바른 도메인의 코퍼스를 수집하고, 크롤링할 때는 해당 사이트의 robots.txt 등을 확인하고 적절한 절차를 통해 수행하길 권장하며, 사용할 수 있는 형태로 가공하는 과정이 필요하다.

4.2.1 단일 코퍼스 수집

한글 가짜 뉴스 (단일 코퍼스의 예시)

```
In [13]: import pandas as pd

ko_data = pd.read_csv('./data/ko_news_train.csv')

ko = ko_data['content']

for idx, line in enumerate(ko):
    if idx <= 5:
        print(idx, line)
```

```
0 [이데일리 MARKETPOINT] 15:32 현재 코스닥 기관 678억 순매도
1 "실적기반" 저가에 매집해야 할 8월 급등유망주 TOP 5 전격공개
2 하이신타론, 선취수수료 없는 월 0.4% 최저금리 상품 출시
3 종합 경제정보 미디어 이데일리 - 무단전재 & 재배포 금지
4 전국적인 소비 붐 조성에 기여할 예정
5 [이데일리 권오석 기자] 중소벤처기업부(이하 중기부)는 대한민국 동행세일에 7개 TV홈쇼핑사가 홍보와 판매에 동참한다고 26일 밝혔다
```

4.2.2 다중 언어 코퍼스 수집

다중 언어 코퍼스 수집

자동 기계번역을 위한 병렬 코퍼스를 구하기란 단일 언어 코퍼스에 비해 상당히 어려우며, 저작권에도 주의해야한다.

자막을 병렬 코퍼스로 사용할 때는 번역 품질 저하의 문제 뿐만 아니라, 'he'나 'she'와 같은 대명사가 사람 이름 등의 고유명사로 표현될 때도 많다. 영어에는 존댓말이 없기 때문에 아빠를 'you'라고 번역하는 등의 문제점을 감수해야한다.

문체	도메인	수집처	정제 난이도	정렬 난이도
문어체	시사, 과학 등	OPUS ¹	낮음	중간
대화체	시사, 교양, 과학 등	TED	낮음	중간
문어체	시사	중앙일보영자신문 ²	낮음	중간
문어체	시사	동아일보영자신문 ³	낮음	중간
문어체	일반	Korean Parallel Data ⁴	낮음	낮음
대화체	일반	드라마, 영화 자막	낮음	중간

▶ 도메인별 다중 언어 코퍼스 수집 사례

1. <http://opus.nlpl.eu/>
2. <http://koreajoongangdaily.joins.com/news/list/List.aspx?gCat=060201>
3. <http://english.donga.com/>
4. <https://sites.google.com/site/koreanparalleldata/>

4.3 정제

정제normalization는 텍스트를 사용하기에 앞서 필수적인 과정이다.

예) 음성 인식은 기호나 특수 문자들을 포함해서는 안되며, 전화번호나 이메일 주소, 신용카드 번호와 같은 개인정보나 민감한 정보들은 제거하거나 변조해서 모델링해야 할 수도 있다.

이처럼 필요한 형태를 얻어내려면 효과적인 정제 방법을 사용해야 한다.

4.3.1. 전각 문자 제거

```
!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKL  
MNOPQRSTUVWXYZ [\]^_`abcdefghijklmnopqrstuvwxyz  
xyz{|}~
```

- 중국어, 일본어 문서, 일부의 한국어 문서의 숫자, 영자, 기호가 전각 문자일 때가 있다.

Ex) 같게 생긴 숫자여도 반각 숫자 (1,2,3)은 숫자로 인식하는 반면, 전각(1,2,3)은 일반 문자로 인식하는 경우가 있어서 반각 문자로 변환해줘야 한다.

4.3 정제

4.3.2 대소문자 통일

하나의 의미를 지니는 여러 단어를 하나의 형태로 통일해 희소성을 줄이는 효과를 기대할 수 있다. 하지만, Word Embedding 이 생기면서 비슷한 값을 벡터로 나타낼 수 있어서 문제를 해결할 필요성이 줄어들었다.

```
en_data = pd.read_csv("data/en_news_train.csv")

import re
from nltk.stem.porter import PorterStemmer

content = en_data['title']

ps = PorterStemmer()
for idx, line in enumerate(content):
    co = []
    if idx <= 10:
        result = re.sub('[^a-zA-Z]', '', line) # 영어를 제외한 모든 문자를 제거하기 / 4,3,3 정규표현식 re.sub
        result = result.lower() # 모든 단어들을 소문자로 변경해주기
        co.append(result)
    print(co)

['house dem aide we didn t even see comeys letter until jason chaffetz tweeted it']
['flynn hillary clinton big woman on campus breitbart']
['why the truth might get you fired']
['civilians killed in single us airstrike have been identified']
['iranian woman jailed for fictional unpublished story about woman stoned to death for adultery']
['jackie mason hollywood would love trump if he bombed north korea over lack of trans bathrooms exclusive video breitbart']
['life life of luxury elton johns favorite shark pictures to stare at during long transcontinental flights']
['beno t hamon wins french socialist party s presidential nomination the new york times']
['excerpts from a draft script for donald trump s q ampa with a black church s pastor the new york times']
['a back channel plan for ukraine and russia courtesy of trump associates the new york times']
['obamas organizing for action partners with soros linked indivisible to disrupt trump s agenda']
```

- ✓ `re.sub[^a-zA-Z]`: 영어를 제외한 모든 문자를 삭제한다는 의미
- ✓ NLTK의 String 메서드인 `lower()`를 사용함.

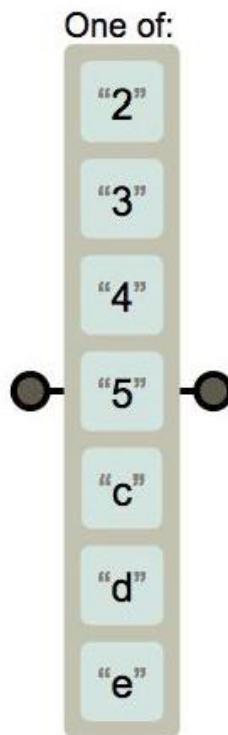
4.3.3 정규 표현식을 사용한 정제

크롤링을 통해 얻어낸 다량의 코퍼스는 보통 특수 문자, 기호 등에 의해 노이즈가 섞일 때가 많다. 이를 효율적으로 감지하고 제거해야 한다.

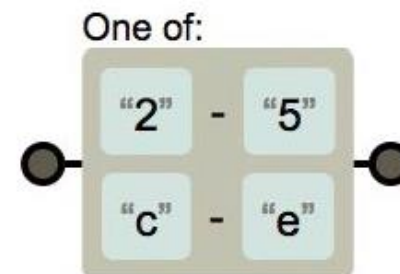


4.3.3 정규 표현식을 사용한 정제

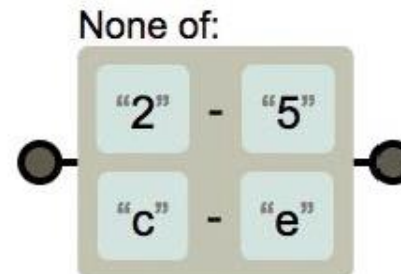
- [] 사용
 - '2 or 3 or 4 or 5 or c or d or e'
 - [2345cde] of (2|3|4|5|c|d|e)



- 사용
 - : 연속된 숫자 또는 알파벳 표현
 - : [2-5c-e]



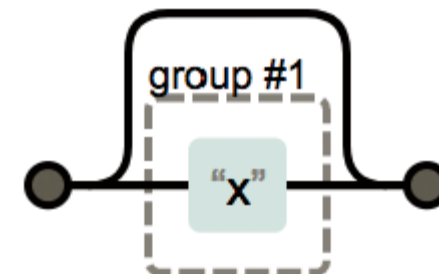
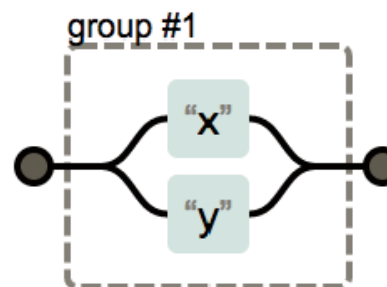
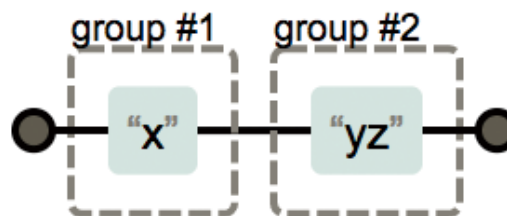
- [^] 사용
 - : Not 기호
 - : [^2-5c-e]
 - : 2부터 5까지, 그리고 c부터 e까지를 제외한 한 글자





4.3.3 정규 표현식을 사용한 정제

- () 사용
 - 그룹을 만들 수 있음
 - (x)(yz)
- | 사용
 - '또는'에 해당 하는 or를 의미
 - (x|y)
- ?, *, + 사용
 - ? : 앞의 수식하는 부분이 나타나지 않거나 한번만 나타날 때 사용
 - x?

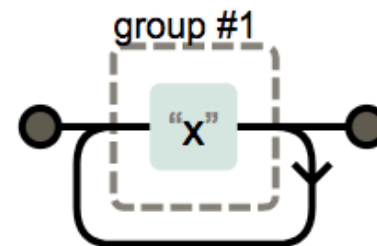




4.3.3 정규 표현식을 사용한 정제

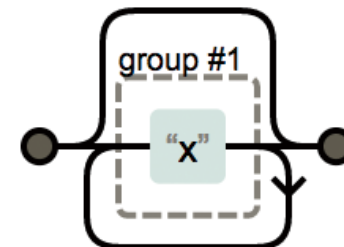
- $+$: 앞의 수식하는 부분이 한 번 이상 나타날 경우

- x^+



- $*$: 앞의 수식하는 부분이 나타나지 않거나 여러 번 나타날 경우

- x^*

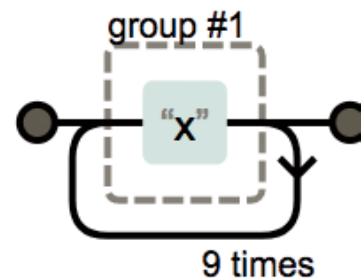


- $\{n\}$, $\{n,\}$, $\{n,m\}$ 사용

- 정확하게 반복 횟수의 범위를 지정

- $x\{n\}$

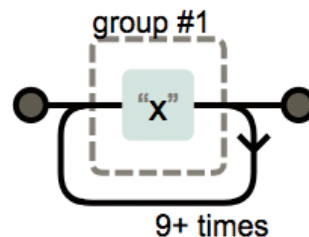
- $n = 9$ 일 때 9번 반복



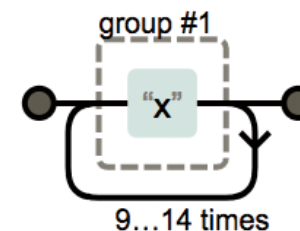


4.3.3 정규 표현식을 사용한 정제

- $x\{n,\}$
 - $n = 9$ 일 때 9번 이상 반복



- $x\{n, m\}$
 - $n = 9, m = 14$ 일 때, 9번에서 14번 사이를 반복



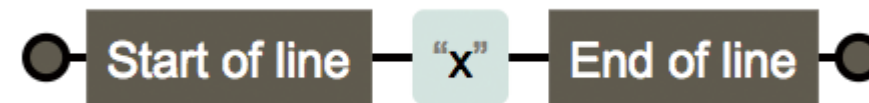
- $.$ 사용
 - 어떤 글자도 $.$ 에 포함됨
 - $.$





4.3.3 정규 표현식을 사용한 정제

- ^와 \$ 사용
 - '[', ']' 에 포함되지 않는 경우
 - ^는 라인의 시작, \$는 라인의 종료를 의미
 - ^x\$



- 지정 문자 사용

지정 문자	설명
\s	공백 문자(white space)
\S	공백 문자를 제외한 모든 문자
\w	alphanumeric(알파벳 + 숫자) + '_'([A-Za-z0-9_]와 같음)
\W	non-alphanumeric 문자 및 '_' 제외([A-Za-z0-9_]와 같음)
\d	숫자([0-9]와 같음)
\D	숫자를 제외한 모든 문자([0-9]와 같음)

▶ 지정 문자 개요



4.3.3 정규 표현식을 사용한 정제

- 예제
 - 문서의 마지막 줄에 나오는 전화번호와 같은 개인정보를 제외

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

```
~~~
```

```
Thank you!
```

```
Sincerely,
```

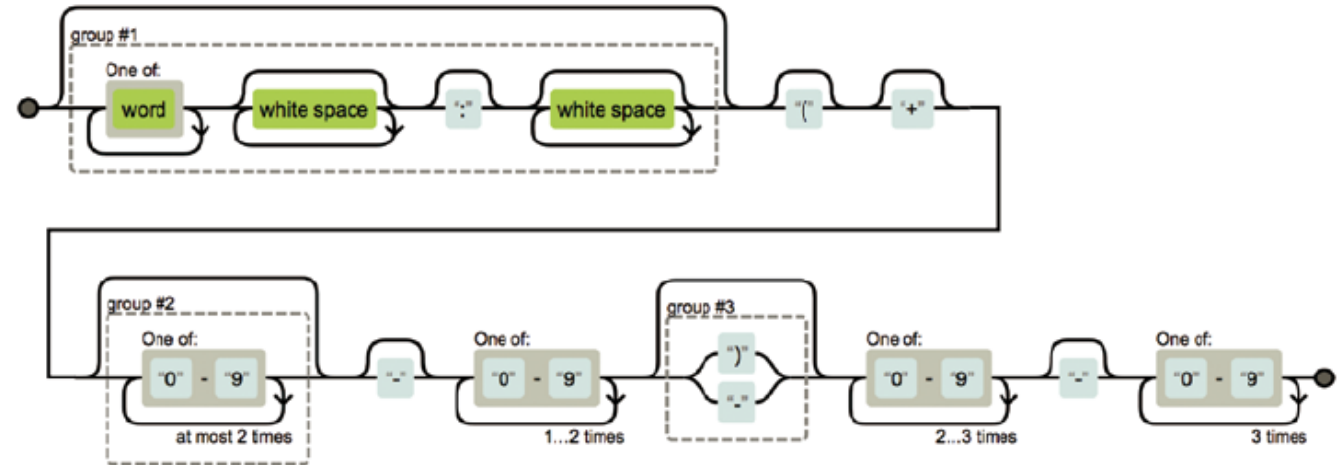
```
Ki: +82-10-1234-5678
```


4.3.3 정규 표현식을 사용한 정제



- 이름이 전화번호 앞에 나올 수도 있다.
- 이름 뒤에는 콜론(:)이 나올 수도 있다.
- 콜론 앞/뒤로는 (탭을 포함한) 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 다음 번호 그룹은 3에서 4자리 숫자이다.
- 마지막은 항상 4자리 숫자이다.

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





4.3.3 정규 표현식을 사용한 정제

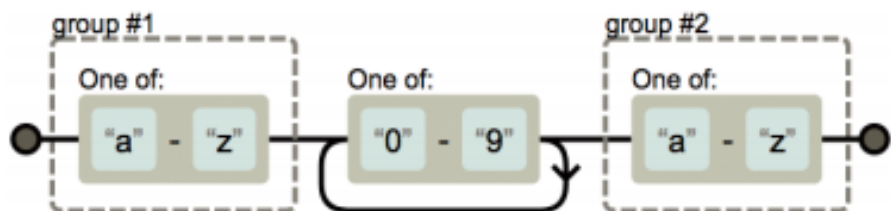
- 파이썬에서 정규 표현식 사용
 - 원하는 패턴을 포함한 문자열 탐색
 - 문자열 내에서 원하는 부분을 치환

```
>>> import re
>>> regex = r"([\w]+\s*:?*\s*)?(?!\+?([0-9]{1,3})?\-?[0-9]{2,3}(\)|!|-)?[0-9]{3,4}\-?[0-9]{4})"
>>> x = "Ki: +82-10-9420-4104"
>>> re.sub(regex, "REMOVED", x)
'REMOVED'
>>> x = "CONTENT jiu 02)9420-4104"
>>> re.sub(regex, "REMOVED", x)
'CONTENT REMOVED'
```

4.3.3 정규 표현식을 사용한 정제

치환자 사용

Q: 알파벳(소문자) 사이에 있는 숫자를 제거 하시오.



```
x = '''abcdefg  
12345  
ab12  
a1bc2d  
12ab  
a1b  
1a2  
a1  
1a  
hijklmnop'''  
  
regex = r'([a-z])[0-9]+([a-z])'  
to = r'\1\2'  
  
y = '\n'.join([re.sub(regex, to, x_i) for x_i in x.split('\n')])  
y
```

```
'abcdefg\n12345\nab12\nabcd\n12ab\nab\n1a2\nna1\n1a\nhijklmnop'
```

4.4 문장 단위 분절

- 보통은 입력 단위가 문장 단위인 경우가 다수
- 한 라인에 한 문장만 있어야 함
- 여러 문장이 한 라인에 있거나, 한 문장이 여러 라인에 걸쳐 있는 경우에는 문장 단위 분절이 필요
- 다만, 단순히 마침표만을 기준으로 문장 단위 분절을 수행하면 U.S. 와 같은 영어 약자나 3.141592와 같은 소수점 등 여러 가지 문제에 마주칠 수 있음
- 따라서 널리 사용되는NLTK의 분절 모듈을 사용하는 것이 좋다.

자연어처리는 인공지능의 한 줄기입니다. 시퀀스 투 시퀀스의 등장 이후로 딥러닝을 활용한 자연어처리는 새로운 전기를 맞이하게 되었습니다. 문장을 받아 단순히 수치로 나타내던 시절을 넘어, 원하대로 문장을 만들어낼 수 있게 된 것입니다.



4.4.1 문장 단위 분절 예제

```
In [1]: import sys, fileinput, re
        from nltk.tokenize import sent_tokenize
```

```
In [4]: path = './data/input/ko_n.txt'
```

```
In [5]: if __name__ == "__main__":
        for line in fileinput.input(path, openhook=fileinput.hook_encoded("utf-8")):
            if line.strip() != "":
                line = re.sub(r'([a-z])\s+([A-Z])', r'\1. \2', line.strip())

                sentences = sent_tokenize(line.strip())

                for s in sentences:
                    if s != "":
                        sys.stdout.write(s + "\n")
```

자연어처리는 인공지능의 한 줄기입니다.
시퀀스 투 시퀀스의 등장 이후로 딥러닝을 활용한 자연어처리는 새로운 전기를 맞이하게 되었습니다.
문장을 받아 단순히 수치로 나타내던 시절을 넘어, 원하대로 문장을 만들어낼 수 있게 된 것입니다.



4.4.2 문장 합치기 및 분절 예제

자연어처리는 인공지능의 한 줄기입니다. 시퀀스 투 시퀀스의 등장 이후로 \n
딥러닝을 활용한 자연어처리는 새로운 전기를 맞이하게 되었습니다. 문장을 \n
받아 단순히 수치로 나타내던 시절을 넘어, 원하대로 문장을 만들어낼 수 \n
있게 된 것입니다.

```
In [4]: if __name__ == "__main__":  
        buf = []  
        for line in fileinput.input(path, openhook=fileinput.hook_encoded("utf-8")):  
            if line.strip() != "":  
                buf += [line.strip()]  
                sentences = sent_tokenize(" ".join(buf))  
  
                if len(sentences) > 1:  
                    buf = sentences[-1:]  
  
                sys.stdout.write('###'.join(sentences[:-1]) + '###')  
        sys.stdout.write(" ".join(buf) + '###')
```

자연어처리는 인공지능의 한 줄기입니다.
시퀀스 투 시퀀스의 등장 이후로 딥러닝을 활용한 자연어처리는 새로운 전기를 맞이하게 되었습니다.
문장을 받아 단순히 수치로 나타내던 시절을 넘어, 원하대로 문장을 만들어낼 수 있게 된 것입니다.

4.5 분절

형태소 분석, 단순한 규칙을 통한 분절을 통해 정규화 수행하며, 보통 띄어쓰기를 사용한다.

일본어, 중국어는 띄어쓰기가 없다. 한국어는 띄어쓰기의 표준화 과정이 충분하지 않아서 띄어쓰기 표준화 하는 과정이 다시 필요하다. 영어는 띄어쓰기 규칙을 잘 따르고 있기 때문에 이에 따라서 처리해주면 용이함.

4.5.1 한국어 분절

-Mecab

한국어 분절에 가장 많이 사용되고 있음. Mecab이 정상적으로 설치된 경우, KoNLPy에서 불러 사용할 수 있음.

- KoNLPy

한국어 형태소 분석기들을 모아 놓은 랩핑 라이브러리를 제공한다. 속도 면에서는 Mecab에 비해서 불리하다.

하지만 설치와 사용이 쉽고, 다양한 라이브러리들을 모아 놓았다는 장점으로 널리 사용되고 있다.

4.5 분절

4.5.1 한국어 분절 (Mecab 예시)

```
!pip install eunjeon
```

Requirement already satisfied: eunjeon in c:\users\hlp\anaconda3\envs\pytorch\lib\site-packages (0.4.0)

- 한국어 분절 적용 후

```
from eunjeon import Mecab

tokenizer = Mecab()
# tokens = []
for text in ko:
    txt = re.sub('[^가-힣a-z]', '', text)
    print(txt), print()
    token = tokenizer.pos(txt) # Pos tagging
    mo = tokenizer.morphs(txt) # 형태소 단위로 tokenized
    mo = ' '.join(mo)
    no = tokenizer.nouns(txt) # 명사 추출
    no = ' '.join(no)
    print(token), print(), print(mo), print(), print(no)
    break
```

이데일리현재코스닥기관억순매도

[('이', 'MM'), ('데일리', 'NNP'), ('현재', 'NNG'), ('코스닥', 'NNG'), ('기관', 'NNG'), ('억', 'NR'), ('순매도', 'NNG')]

이 데일리 현재 코스닥 기관 억 순매도

데일리 현재 코스닥 기관 억 순매도

4.5 분절

4.5.2 영어 분절

- Moses

영어의 경우 띄어쓰기가 잘 통일되어 있는 편이므로, 띄어쓰기 자체에 대한 큰 정규화 이슈는 없다.

'쉼표', '마침표', '인용부호'등을 띄어 주어야한다. Moses에서 제공하는 분절 기능을 통해 이전 처리를 수행한다.

```
!pip install nltk==3.2.5
```

```
Requirement already satisfied: nltk==3.2.5 in c:\users\nlp\anaconda3\envs\pytorch\lib\site-packages (3.2.5)
```

```
Requirement already satisfied: six in c:\users\nlp\anaconda3\envs\pytorch\lib\site-packages (from nltk==3.2.5) (1.15.0)
```

- 영어 분절 적용 전

```
e = open('./data/input.en.txt', mode='rt')
```

```
e.read(200)
```

```
"Natural language processing is one of biggest stream in artificial intelligence, and it becomes very popular after seq2seq's invention. "
```

4.5 분절

4.5.2 영어 분절

- Moses (영어 분절 적용 후 예시)

- 영어 분절 적용 후

```
import sys, fileinput
from nltk.tokenize.moses import MosesTokenizer

t = MosesTokenizer()

path = './data/input.en.txt'

if __name__ == "__main__":
    for line in fileinput.input(path):
        if line.strip() != "":
            tokens = t.tokenize(line.strip(), escape=False)

            sys.stdout.write(' '.join(tokens) + "\n")
        else:
            sys.stdout.write("\n")
```

Natural language processing is one of biggest stream in artificial intelligence , and it becomes very popular after seq2seq 's invention .

4.5 분절

4.5.3 중국어 분절

- 스탠포드 파서

스탠포드 대학교에서 제작한 중국어 파서 프로그램이다. 자바에서 제작됨.

- <https://nlp.stanford.edu/software/lex-parser.shtml>

- JIEBA

비교적 늦게 개발 및 업데이트된 Jieba 라이브러리는, 성능상으로는 다른 형태소 분석기들과 큰 차이가 없지만,

Python으로 구현 되어 사용하기에 용이하고, 실제 사용화를 위한 배치 시스템을 구현할 때, 매우 쉽다.

특히 딥 러닝을 사용한 머신 러닝 시스템을 대부분 파이썬 앞에서 동작하기 때문에 일관성 있는 시스템을 구성하기 매우 좋다.

- <https://github.com/fxsjy/jieba>

4.6 병렬 코퍼스 정렬

4.6.1 병렬 코퍼스 제작 프로세스 개요

1. 소스 언어와 타겟 언어 사이의 **단어 사전을 준비**합니다.
2. 만약 준비된 단어 사전이 없다면, 다음 작업을 수행합니다. 만약 이미 단어 사전을 갖고 있다면 7번으로 건너 뛩니다.
3. 각 언어에 대해서 **코퍼스를 수집 및 정제** 합니다.
4. 각 언어에 대해서 **단어 임베딩 벡터**를 구합니다. 단어 임베딩 벡터에 대해서는 추후 다루겠습니다.
5. **MUSE를 통해 단어 레벨 번역기를 훈련**합니다.
6. 훈련된 단어 레벨 번역기를 통해 **두 언어 사이의 단어 사전을 생성**합니다.
7. 만들어진 단어 사전을 넣어 **Champlion**을 통해 기존에 수집된 다중 언어 코퍼스를 정렬합니다.
8. 각 언어에 대해서 단어 사전을 적용하기 위해 **알맞은 수준의 분절**을 수행합니다.
9. 각 언어에 대해서 **정제**를 수행합니다.
10. **Champlion**을 사용하여 병렬 코퍼스를 생성합니다.

4.6 병렬 코퍼스 정렬

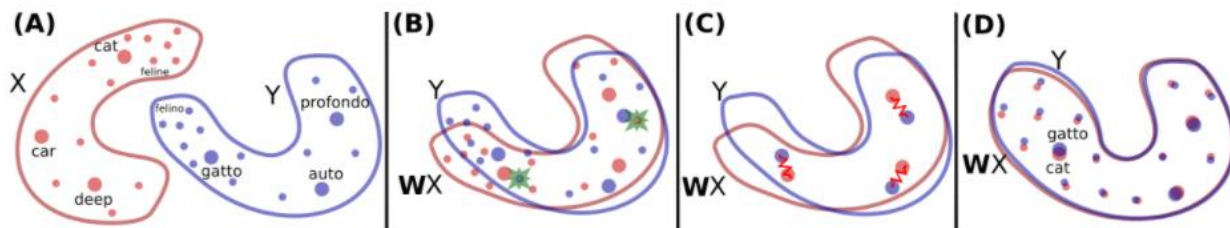
4.6.2 사전 생성

페이스북의 MUSE는 병렬 코퍼스가 없는 상황에서 사전을 구축하는 방법과 코드를 제공한다.

각 단일 언어 코퍼스를 통해 구축한 언어별 단어 임베딩 벡터에 대해 다른 언어의 임베딩 벡터와 맵핑 시켜 단어 간 번역을 수행할 수 있다. 이는 각 언어별 코퍼스가 많을수록, 임베딩 벡터가 많을수록 더욱 정확하게 수행된다.

MUSE는 병렬 코퍼스가 없는 상황에서도 수행할 수 있기 때문에 비지도 학습이라고 한다.

MUSE: Multilingual Unsupervised and Supervised Embeddings



- <https://github.com/facebookresearch/MUSE>

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

CTK의 입력을 사용하여
소스 언어와 타킷 언어를 맵핑한 결과

4.6 병렬 코퍼스 정렬

4.6.3 CTK을 활용한 정렬

CTKChampollion Toolkit은 이중 언어 코퍼스의 문장 정렬을 수행하는 오픈소스이다.

기존에 구축된 단어 사전을 이용하거나, 앞에서 와 같이 자동으로 구축한 단어 사전을 참고하여 CTK은 문장 정렬을 수행한다. 여러 라인으로 구성된 언어별 문서에 대해 문장 정렬한 결과의 예제이다.

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

타깃 언어의 1, 2, 3번째 문장은 짝을 찾지 못하고 버려졌고, 소스 언어의 1, 2, 3번째 문장은 각각 타깃 언어의 4, 5, 6번째 문장과 매핑된 것을 알 수 있습니다. 또한 소스 언어의 4, 5번째 두 문장은 타깃 언어의 7번 문장에 동시에 매핑된 것을 알 수 있습니다. 이와 같이 어떤 문장들은 버려지기도 하고, 일대일one-to-one 매핑이 이루어지기도 하며, 일대다one-to-many, 다대일many-to-one 매핑이 이루어지기도 한다.

4.6 병렬 코퍼스 정렬

4.6.3 CTK을 활용한 정렬

CTKChampollion Toolkit의 python 오픈 소스이다.

```
import sys, argparse, os

BIN = CTK_ROOT + "/bin/champollion"
CMD = "%s -c %f -d %s %s %s"
OMIT = "omitted"
DIR_PATH = './tmp/'
INTERMEDIATE_FN = DIR_PATH + "tmp.txt"
```

```
def read_alignment(fn):
    aligns = []

    f = open(fn, 'r')

    for line in f:
        if line.strip() != "":
            srcs, tgts = line.strip().split('<=> ')

            if srcs == OMIT:
                srcs = []
            else:
                srcs = list(map(int, srcs.split(',')))

            if tgts == OMIT:
                tgts = []
            else:
                tgts = list(map(int, tgts.split(',')))

            aligns += [(srcs, tgts)]

    f.close()

    return aligns
```

```
def get_aligned_corpus(src_fn, tgt_fn, aligns):
    f_src = open(src_fn, 'r')
    f_tgt = open(tgt_fn, 'r')

    for align in aligns:
        srcs, tgts = align

        src_buf, tgt_buf = [], []

        for src in srcs:
            src_buf += [f_src.readline().strip()]
        for tgt in tgts:
            tgt_buf += [f_tgt.readline().strip()]

        if len(src_buf) > 0 and len(tgt_buf) > 0:
            sys.stdout.write("%s\t%s\n" % (" ".join(src_buf), " ".join(tgt_buf)))

    f_tgt.close()
    f_src.close()

def parse_argument():
    p = argparse.ArgumentParser()

    p.add_argument('--src', required = True)
    p.add_argument('--tgt', required = True)
    p.add_argument('--src_ref', default = None)
    p.add_argument('--tgt_ref', default = None)
    p.add_argument('--dict', required = True)
    p.add_argument('--ratio', type = float, default = 1.1966)

    config = p.parse_args()

    return config
```

```
if __name__ == "__main__":
    if not os.path.exists(DIR_PATH):
        os.mkdir(DIR_PATH)

    config = parse_argument()

    if config.src_ref is None:
        config.src_ref = config.src
    if config.tgt_ref is None:
        config.tgt_ref = config.tgt

    cmd = CMD % (BIN, config.ratio, config.dict, config.src_ref, config.tgt_ref, INTERMEDIATE_FN)
    os.system(cmd)

    aligns = read_alignment(INTERMEDIATE_FN)
    get_aligned_corpus(config.src, config.tgt, aligns)
```

<https://github.com/LowResourceLanguages/champollion>

4.7 서브워드 분절

- **BPE(Byte Pair encoding) 알고리즘**을 통한 서브 워드 단위 분절은 현재 필수 전처리 방법이다.

'단어는 의미를 가진 더 작은 서브 워드들의 조합으로 이루어진다'는 가정하에 적용되는 알고리즘이다.

영어, 한국어 역시 라틴어와 한자를 기반으로 형성된 언어이기 때문에 많은 서브 워드들이 존재한다.

- Ex1) concentrate = con(=together) + centr(=center) + ate(=make)

- Ex2) 집중(輯中) = 輯 (모을 집) + 中 (가운데 중)

따라서 적절한 서브 워드들을 발견하여 해당 단위로 쪼개어주면 어휘 수를 줄일 수 있고, 희소성을 효과적으로 줄일 수 있다.

서브 워드 단위 분절로 얻는 가장 대표적인 효과는 unknown 토큰에 대한 효율적인 대처이다.

자연어 생성 부분에서 unknown 토큰이 나타나면 적절한 문장의 임베딩 생성이 어려워지고 이에 따라서 이전 단어를 기반으로 다음 단어를 예측하기 때문에 더더욱 어렵다.

4.7 서브워드 분절

4.7.1 예제

다음과 같이 BPE를 적용하면 원래의 띄어쓰기 공백 외에도 BPE의 적용으로 인한 공백이 추가된다.

이때 특수 문자 _ 를 사용하여 기존의 띄어쓰기 또는 단어의 시작을 나타냅니다.

한글 Mecab에 의해 분절된 원문

- 한글 Mecab에 의해 tokenization 된 원문

```
from eunjeon import Mecab
tokenizer = Mecab()

with open('data/train_tokenizer.txt', 'r', encoding='utf-8') as f:
    test_tokenizer = f.read().split('\n')
    for t in test_tokenizer:
        token_text = tokenizer.morphs(t)
        token_text = ' '.join(token_text)
        print(token_text)
```

```
글 ㅋ
GDNTOPCLASSINTHECLUB
뭐 야 이 평점 들 은 . ... 나쁜 진 알 지만 10 점 짜리 는 더더욱 아니 잮아
지루 하 지 는 알 는데 완전 막장 임 . ... 돈 주 고 보 기 에 는 . ...
```

4.7 서버워드 분절

4.7.1 예제

한글 BPE 적용 (Google의 SentencePiece 모듈 사용)

- 2.3. 학습한 BPE 모델을 load 하여 subword tokenize 수행

```
## check
import sentencepiece as spm
sp = spm.SentencePieceProcessor()
sp.Load('{}model'.format(sp_model_path))

with open('data/train_tokenizer.txt', 'r', encoding='utf-8') as f:
    test_tokenizer = f.read().split('\n')
    for idx, t in enumerate(test_tokenizer):
        if idx <= 3:
            tokens = sp.encode_as_pieces(t)
            ids = sp.encode_as_ids(t)

            print(tokens), print(ids), print()

            tokens = sp.decode_pieces(tokens)
            ids = sp.decode_ids(ids)

            print(ids), print(tokens)

['_꺇', ' ', '꺇']
[1082, 305]

한
=
=

['_G', 'D', 'N', 'T', 'O', 'P', 'CL', 'A', 'SS', 'IN', 'THE', 'CL', 'U', 'B']
[11752, 1099, 7397, 2447, 4562, 6878, 30939, 2904, 20559, 14575, 8282, 30939, 10765, 3313]

GONTOPCLASSINTHECLUB
GONTOPCLASSINTHECLUB
['_꺇', ' ', '이', '평점', '들은', '...', '나쁜진', '않지만', '_10', '점', '_짜리', '는', '_더더욱', '_아니잖아']
[791, 142, 174, 526, 166, 11861, 2474, 171, 151, 4079, 127, 14653, 3551]

꺇야 이 평점들은... 나쁜진 않지만 10점 짜리는 더더욱 아니잖아
꺇야 이 평점들은... 나쁜진 않지만 10점 짜리는 더더욱 아니잖아
['_지루하지', '는', '_알은데', '_완전', '_막장', '_일', '...', '_돈주고', '_보기에는', '...']
[1608, 127, 4364, 198, 789, 246, 123, 1279, 3622, 166]

지루하지는 않은데 완전 막장임... 돈주고 보기에는...
지루하지는 않은데 완전 막장임... 돈주고 보기에는...
```

4.7 서브워드 분절

4.7.1 예제

영어 NLTK에 의해 tokenization이 된 원문

```
en_data = pd.read_csv("data/en_news_train.csv")

import re
import nltk

content = en_data['title']

total = []

for idx, line in enumerate(content):
    co = []
    if idx <= 5:
        result = nltk.word_tokenize(line)
        result = ' '.join(result)
        co.append(result)
        print(co)
    total.append(co)
```

```
['House Dem Aide : We Didn ' t Even See Comey ' s Letter Until Jason Chaffetz Tweeted It']
['FLYNN : Hillary Clinton , Big Woman on Campus - Breitbart']
['Why the Truth Might Get You Fired']
['15 Civilians Killed In Single US Airstrike Have Been Identified']
['Iranian woman jailed for fictional unpublished story about woman stoned to death for adultery']
['Jackie Mason : Hollywood Would Love Trump if He Bombed North Korea over Lack of Trans Bathrooms ( Exclusive Video ) - Breitbart']
```

4.7 서브워드 분절

4.7.1 예제

영어 BPE 적용 (google sentencepiece 모듈 적용)

```
import sentencepiece as spm

sp = spm.SentencePieceProcessor(model_file='test/test_model.model')

for i in range(len(total)):
    if i <= 5:
        en_ids = sp.encode(total[i], out_type=int)
        en_tokens = sp.encode(total[i], out_type=str)
        print(en_ids)

        for en in en_tokens:
            en_tokens_ = ' '.join(en)
            print(en_tokens_)
```

[[4, 165, 196, 16, 304, 16, 26, 104, 140, 16, 4, 224, 416, 304, 140, 24, 4, 0, 4, 15, 892, 168, 16, 16, 408, 501, 34, 4, 0, 51, 4, 828, 19
0, 4, 352, 24, 15, 128, 4, 336, 185, 112, 408, 414, 57, 57, 16, 15, 997, 243, 63, 16, 16, 228, 225]]
_Hous_e_Dem_Aide_: _We_Didn_ ' _t_Even_See_Come_y_ ' _s_ Let ter _ Un til _Jas on _C ha f f e t z _T w
e e t e d _I t
[[280, 277, 595, 199, 199, 4, 224, 4, 165, 128, 180, 334, 408, 31, 55, 15, 112, 4, 5, 296, 25, 37, 4, 321, 21, 438, 59, 408, 579, 193, 4,
35, 296, 56, 114, 66, 48, 15]]
_F L Y N N _ : _H i l l a r y _C l i n t o n _ , _B i g _ W o m a n _o n _C a m p u s _ - _B r e i t b a r t
[[4, 701, 7, 243, 30, 53, 95, 157, 25, 37, 60, 15, 519, 16, 15, 361, 280, 158, 12]]
_W h y _t h e _T r u t h _M i g h t _G e t _Y o u _F i r e d
[[4, 357, 556, 408, 25, 87, 128, 25, 84, 8, 4, 401, 128, 332, 379, 168, 18, 82, 4, 352, 262, 104, 158, 85, 83, 206, 4, 165, 19, 125, 296,
16, 98, 9, 17, 109, 270, 25, 12]]
_1 5 _C i v i l i a n s _K i l l e d _I n _S i n g l e _U S _A i r s t r i k e _H a v e _B e e n _I d e n t i f i e d
[[9, 30, 84, 25, 84, 490, 4, 999, 19, 128, 12, 42, 74, 25, 409, 73, 210, 29, 53, 66, 31, 289, 12, 559, 106, 490, 821, 17, 10, 86, 19, 95,
42, 11, 17, 229, 190, 34]]
_I r a n i a n _w o m a n _j a i l e d _f o r _f i c t i o n a l _u n p u b l i s h e d _s t o r y _a b o u t _w o m a n _s t o n e d _t o _d e a t h _f o r _a d u l t e r y
[[4, 336, 19, 28, 46, 25, 16, 157, 185, 112, 4, 224, 4, 165, 21, 31, 41, 63, 21, 21, 17, 4, 321, 21, 229, 17, 4, 277, 21, 125, 243, 30, 15
6, 29, 94, 151, 296, 21, 26, 66, 12, 283, 30, 95, 4, 401, 54, 16, 19, 255, 4, 277, 19, 28, 46, 14, 243, 30, 84, 8, 296, 19, 95, 131, 21, 2
6, 8, 420, 4, 120, 297, 28, 31, 193, 217, 4, 0, 140, 16, 21, 4, 394, 4, 35, 296, 56, 114, 66, 48, 15]]
_J a c k i e _M a s o n _ : _H o l l y w o o d _W o u l d _L o v e _T r u m p _i f _H e _B o m b e d _N o r t h _K o r e a _o v e r _L a c k _
o f _T r a n s _B a t h r o o m s _ (_E x c l u s i v e _V i d e o _) _ - _B r e i t b a r t

4.7 서브워드 분절

4.7.2 오픈 소스

- 서브 워드 단위 분절을 수행하기 위한 오픈소스들은 다음과 같다.
 - 구글의 SentencePiece 모듈이 속도가 빠르지만, 논문 원저자의 파이썬 코드는 수정이 쉬워 편의에 따라 사용하면 된다.
- Sennrich(원저자)의 깃허브: <https://github.com/rsennrich/subword-nmt>
 - 본서의 저자가 수정한 버전: <https://github.com/kh-kim/subword-nmt>
 - Google의 SentencePiece 모듈: <https://github.com/google/sentencepiece>

4.8 분절 복원

- 4.8.1 분절 후처리
- 복원을 수월하게 하기 위해, 분절 이후에는 특수 문자를 분절 과정에서 새롭게 생긴 공백 다음에 삽입한다.
- 다음은 기존의 공백과 전처리 단계에서 생성된 공백을 서로 구분하기 위한 특수 문자 '_' 을 삽입하는 파이썬 스크립트 예제이다.

Post Tokenization

```
import sys
STR = '_'

if __name__ == "__main__":
    ref_fn = sys.argv[1]
    f = open(ref_fn)

    for ref in f:
        ref_tokens = ref.strip().split(' ')
        input_line = sys.stdin.readline().strip()

        if input_line != '':
            tokens = input_line.split(' ')

            idx = 0
            buf = []

            # We assume that stdin has more tokens than reference input.
            for ref_token in ref_tokens:
                tmp_buf = []

                while idx < len(tokens):
                    if tokens[idx].strip() == '':
                        idx += 1
                        continue

                    tmp_buf += [tokens[idx]]
                    idx += 1

                    if ''.join(tmp_buf) == ref_token:
                        break

                if len(tmp_buf) > 0:
                    buf += [STR + tmp_buf[0].strip() + tmp_buf[1:]]

            sys.stdout.write(' '.join(buf) + '\n')
        else:
            sys.stdout.write('\n')

    f.close()
```

```
$ cat [before filename] | python tokenizer.py | python post_tokenize.py [before filename]
```

4.8 분절 복원

4.8.2 분절 복원 예제

- 앞서 설명한 분절 복원을 sed 명령어를 통해 수행할 경우의 예제

Detokenization

```
# bash에서 sed 명령어를 통해 수행 할 경우에 대한 예제
```

```
sed "s/ //g" | sed "s/___/ /g" | sed "s/_//g" | sed "s/^#/g"
```

```
# python script 예제 처럼 처리
```

```
import sys

if __name__ == "__main__":
    for line in sys.stdin:
        if line.strip() != "":
            line = line.strip().replace(' ', '').replace('___', ' ').replace('_', '').strip()

            sys.stdout.write(line + '\n')
```

4.8 토치 텍스트 (torch Text)

데이터 입력을 준비하는 부분이다. 토치 텍스트(torch Text)는 자연어 처리 문제 또는 텍스트에 관한 머신 러닝이나 딥 러닝을 수행하는 데이터를 읽고 전 처리하는 코드를 모아둔 라이브러리이다. 토치 텍스트를 활용한 기본적인 데이터 로딩 방법을 실습 할 예정이다.

4.9.1 설치 방법

4.9.1 설치 방법

```
: !pip install torchtext
```

```
Requirement already satisfied: torchtext in c:\users\nlp\anaconda3\envs\pytorch\lib\site-packages (0.6.0)
Requirement already satisfied: torch in c:\users\nlp\anaconda3\envs\pytorch\lib\site-packages (from torchtext) (1.6.0)
Requirement already satisfied: sentencepiece in c:\users\nlp\anaconda3\envs\pytorch\lib\site-packages (from torchtext) (0.1.94)
Requirement already satisfied: tqdm in c:\users\nlp\anaconda3\envs\pytorch\lib\site-packages (from torchtext) (4.50.2)
Requirement already satisfied: numpy in c:\users\nlp\anaconda3\envs\pytorch\lib\site-packages (from torchtext) (1.19.1)
Requirement already satisfied: six in c:\users\nlp\anaconda3\envs\pytorch\lib\site-packages (from torchtext) (1.15.0)
Requirement already satisfied: requests in c:\users\nlp\anaconda3\envs\pytorch\lib\site-packages (from torchtext) (2.24.0)
Requirement already satisfied: future in c:\users\nlp\anaconda3\envs\pytorch\lib\site-packages (from torch->torchtext) (0.18.2)
Requirement already satisfied: idna<3,>=2.5 in c:\users\nlp\anaconda3\envs\pytorch\lib\site-packages (from requests->torchtext) (2.10)
Requirement already satisfied: urllib3!=1.25.0,!<1.25.1,<1.26,>=1.21.1 in c:\users\nlp\anaconda3\envs\pytorch\lib\site-packages (from requests->torchtext) (1.25.11)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\nlp\anaconda3\envs\pytorch\lib\site-packages (from requests->torchtext) (2019.11.28)
Requirement already satisfied: chardet<4,>=3.0.2 in c:\users\nlp\anaconda3\envs\pytorch\lib\site-packages (from requests->torchtext) (3.0.4)
```


4.8 토치 텍스트 (torch Text)

4.9.2 예제 코드

- 학습 데이터는 크게 세 가지 형태로 분류한다. 이 세 가지 종류의 데이터 형태를 다루는 방법을 실습할 예정이다.

x 데이터	y 데이터	활용분야
코퍼스	클래스	텍스트 분류, 감성 분석
코퍼스	-	언어 모델
코퍼스	코퍼스	기계번역, 요약, 질의응답

▶ 세 가지 형태로 분류되는 학습 데이터

- 텍스트 파일 내의 필드를 먼저 정의한다.
- (텍스트 파일 내에서 탭을 사용하여 필드(또는 열column)를 구분하는 방식을 자연어 처리 분야의 입력에서 가장 많이 사용한다.)
- 정의된 각 필드를 Dataset 클래스를 통해 읽어 들인다.
- 읽어들이는 코퍼스는 미리 주어진 미니배치 크기에 따라서 나뉘 수 있도록 이터레이터에 들어간다.
- 미니배치를 구성하는 과정에서 미니배치 내에 문장의 길이가 다를 경우에는 필요에 따라 문장의 앞 또는 뒤에 패딩padding(PAD)을 삽입한다.
- (이 패딩은 추후 소개할 BOS, EOS와 함께 하나의 단어 또는 토큰과 같은 취급을 받는다. 이후에 훈련 코퍼스에 대해 어휘 사전을 만들어 각 단어(토큰)를 숫자로 맵핑 하는 작업을 수행한다.)

4.8 토치 텍스트 (torch Text)

| 코퍼스와 레이블 읽기 |

첫 번째 예제는 한 줄에서 클래스와 텍스트가 탭 ('\t')으로 구분된 데이터의 입력을 받는 내용 입니다.

이런 예제는 주로 텍스트 분류를 위해 사용

```
class TextClassificationDataLoader():
    def __init__(self, train_fn, valid_fn, tokenizer,
                 batch_size = 64,
                 device = 'cpu',
                 max_vocab = 9999999,
                 fix_length = None,
                 use_eos = False,
                 shuffle = True):
        super(TextClassificationDataLoader, self).__init__()

        self.label = data.Field(sequential = False, use_vocab = False)
        self.text = data.Field(tokenize = tokenizer,
                               use_vocab = True,
                               batch_first = True,
                               include_lengths = True,
                               fix_length = fix_length,
                               eos_token = '<EOS>' if use_eos else None
                              )

        train, valid = data.TabularDataset.splits(path = '',
                                                  train = train_fn,
                                                  validation = valid_fn,
                                                  format = 'tsv',
                                                  fields = [('label', self.label), ('text', self.text)]
                                                  )

        self.train_iter, self.valid_iter = data.BucketIterator.splits((train, valid),
                                                                      batch_size = batch_size,
                                                                      device = 'cuda:%d' % device if device >= 0 else 'cpu',
                                                                      shuffle = shuffle
                                                                      )

        self.label.build_vocab(train)
        self.text.build_vocab(train, max_size = max_vocab)
```

4.8 토치 텍스트 (torch Text)

| 코퍼스 읽기 |

- 한 라인에 텍스트로만 채워져 있을 때를 위한 코드 이다. 주로 언어모델(language model)을 훈련 시키는 상황 사용 된다.
- LanguageModelDataset을 통해서 미리 정의된 필드를 텍스트 파일에서 읽어 들인다. 이때 각 문장의 길이에 따라 정렬을 통해 비슷한 길이의 문장끼리 미니배치를 만들어준다. 이 작업을 통해서 매우 상이한 길이의 문장들이 하나의 미니배치에 묶여 훈련 시간에서 손해보는 것을 방지한다.

```
from torchtext import data, datasets

PAD = 1
BOS = 2
EOS = 3

class DataLoader():

    def __init__(self, train_fn, valid_fn,
                 batch_size = 64,
                 device = 'cpu',
                 max_vocab = 99999999,
                 max_length = 255,
                 fix_length = None,
                 use_bos = True,
                 use_eos = True,
                 shuffle = True):

        super(DataLoader, self).__init__()

        self.text = data.Field(sequential = True,
                               use_vocab = True,
                               batch_first = True,
                               include_lengths = True,
                               fix_length = fix_length,
                               init_token = '<BOS>' if use_bos else None,
                               eos_token = '<EOS>' if use_eos else None)

        train = LanguageModelDataset(path = train_fn,
                                     fields = [('text', self.text)],
                                     max_length = max_length)

        valid = LanguageModelDataset(path = valid_fn,
                                     fields = [('text', self.text)],
                                     max_length = max_length)
```

```
self.train_iter = data.BucketIterator(train,
                                     batch_size = batch_size,
                                     device = 'cuda:%d' % device if device >= 0 else 'cpu',
                                     shuffle = shuffle,
                                     sort_key=lambda x: -len(x.text),
                                     sort_within_batch = True)

self.valid_iter = data.BucketIterator(valid,
                                     batch_size = batch_size,
                                     device = 'cuda:%d' % device if device >= 0 else 'cpu',
                                     shuffle = False,
                                     sort_key=lambda x: -len(x.text),
                                     sort_within_batch = True)

self.text.build_vocab(train, max_size = max_vocab)

# LanguageModelDataset을 통해서 미리 정의된 필드를 텍스트 파일에서 읽어들인다.
# 이때 각 문장의 길이에 따라 정렬을 통해 비슷한 길이의 문장끼리 미니배치를 만들어준다.
class LanguageModelDataset(data.Dataset):

    def __init__(self, path, fields, max_length=None, **kwargs):
        if not isinstance(fields[0], (tuple, list)):
            fields = [('text', fields[0])]

        examples = []

        path = 'data/'

        with open(path) as f:
            for line in f:
                line = line.strip()
                if max_length and max_length < len(line.split()):
                    continue
                if line != '':
                    examples.append(data.Example.fromlist(
                        [line], fields))

        super(LanguageModelDataset, self).__init__(examples, fields, **kwargs)
```

```
if __name__ == '__main__':
    import sys
    loader = DataLoader(sys.argv[1], sys.argv[2])

    for batch_index, batch in enumerate(loader.train_iter):
        print(batch.text)

        if batch_index >= 1:
            break
```

4.8 토치 텍스트 (torch Text)

| 병렬 코퍼스 읽기 |

- 텍스트로만 채워진 두개의 파일을 동시에 입력 데이터로 읽어 들이는 예제 입니다. 이때, 두 파일의 corpus는 병렬(parallel) 데이터로 취급 되어 같은 라인 수로 채워져 있어야 한다. 주로 기계번역(machine translation) 이나 요약(summarization) 등에 사용 할 수 있습니다.
- **Tabular Dataset** 클래스를 이용해서 하나의 파일에서 두 개의 열column에 각 언어의 문장을 표현한다.
- **LanguageModelDataset** 과 마찬가지로 길이에 따라서 미니배치를 구성한다.

```
import os
from torchtext import data, datasets

PAD = 1
BOS = 2
EOS = 3

class DataLoader():

    def __init__(self, train_fn = None,
                 valid_fn = None,
                 exts = None,
                 batch_size = 64,
                 device = 'cpu',
                 max_vocab = 99999999,
                 max_length = 255,
                 fix_length = None,
                 use_bos = True,
                 use_eos = True,
                 shuffle = True
                 ):

        super(DataLoader, self).__init__()

        self.src = data.Field(sequential = True,
                              use_vocab = True,
                              batch_first = True,
                              include_lengths = True,
                              fix_length = fix_length,
                              init_token = None,
                              eos_token = None
                              )

        self.tgt = data.Field(sequential = True,
                              use_vocab = True,
                              batch_first = True,
                              include_lengths = True,
                              fix_length = fix_length,
                              init_token = '<BOS>' if use_bos else None,
                              eos_token = '<EOS>' if use_eos else None
                              )
```

4.8 토치 텍스트 (torch Text)

| 병렬 코퍼스 읽기 |

```
if train_fn is not None and valid_fn is not None and exts is not None:
    train = TranslationDataset(path = train_fn, exts = exts,
                              fields = [('src', self.src), ('tgt', self.tgt)],
                              max_length = max_length)
    valid = TranslationDataset(path = valid_fn, exts = exts,
                              fields = [('src', self.src), ('tgt', self.tgt)],
                              max_length = max_length)

    self.train_iter = data.BucketIterator(train,
                                          batch_size = batch_size,
                                          device = 'cuda:%d' % device if device >= 0 else 'cpu',
                                          shuffle = shuffle,
                                          sort_key=lambda x: len(x.tgt) + (max_length + len(x.src)),
                                          sort_within_batch = True)
    self.valid_iter = data.BucketIterator(valid,
                                          batch_size = batch_size,
                                          device = 'cuda:%d' % device if device >= 0 else 'cpu',
                                          shuffle = False,
                                          sort_key=lambda x: len(x.tgt) + (max_length + len(x.src)),
                                          sort_within_batch = True)

    self.src.build_vocab(train, max_size = max_vocab)
    self.tgt.build_vocab(train, max_size = max_vocab)

def load_vocab(self, src_vocab, tgt_vocab):
    self.src.vocab = src_vocab
    self.tgt.vocab = tgt_vocab
```

```
class TranslationDataset(data.Dataset):
    """Defines a dataset for machine translation."""

    @staticmethod
    def sort_key(ex):
        return data.interleave_keys(len(ex.src), len(ex.trg))

    def __init__(self, path, exts, fields, max_length=None, **kwargs):
        """Create a TranslationDataset given paths and fields.
        Arguments:
            path: Common prefix of paths to the data files for both languages.
            exts: A tuple containing the extension to path for each language.
            fields: A tuple containing the fields that will be used for data
                   in each language.
            Remaining keyword arguments: Passed to the constructor of
            data.Dataset.
        """
        if not isinstance(fields[0], (tuple, list)):
            fields = [('src', fields[0]), ('trg', fields[1])]

        if not path.endswith('.'):
            path += '.'

        src_path, trg_path = tuple(os.path.expanduser(path + x) for x in exts)

        examples = []
        with open(src_path) as src_file, open(trg_path) as trg_file:
            for src_line, trg_line in zip(src_file, trg_file):
                src_line, trg_line = src_line.strip(), trg_line.strip()
                if max_length and max_length < max(len(src_line.split()), len(trg_line.split())):
                    continue
                if src_line != '' and trg_line != '':
                    examples.append(data.Example.fromlist(
                        [src_line, trg_line], fields))

        super(TranslationDataset, self).__init__(examples, fields, **kwargs)
```