Preprocessing

Noam Chomsky – Image from Wikipedia

Preprocessing

자연어처리에 있어서 전처리는 매우 중요합니다. 사실 어떻게 보면 image를 다루는 computer vision이나 눈에 보이지 않는 signal을 다루는 audio 또는 speech 분야에 비하면 데이터의 성격이 다루기 쉬워 보일 수 있습니다. 하지만, 오히려 discrete한 symbol로 이루어져 있기 때문에 continuous한 값들로 이루어진 다른 데이터의 형태에 비해 훨씬 더 정제의 중요성이 강조됩니다. 예를 들어 이미지는 1000x1000 image의 픽셀 하나의 색깔이 살짝 바뀌어도 그 이미지의 의미는 변화가 없다고 할 수 있지만, 문장에서의 단어 하나의 변화는 문장의 뉘앙스를 바꿀 수도 있고, 아예 문장의 의미 자체를 완전히 다른 것으로 만들어 버릴 수도 있습니다. 따라서, NLP에서의 전처리는 매우 중요한 의미를 가지고 있습니다.

Corpus란?

Corpus(코퍼스)는 말뭉치라고 불리우기도 합니다. (복수 표현은 corpora) 보통 문장은 여러 단어들로 이루어져 있고, 이러한 문장들의 집합을 corpus라고 합니다. NLP분야의 machine learning을 수행하기 위해서는 훈련 데이터가 필요한데 보통 이런 다수의 문장들로 구성된 corpus가 필요합니다.

한가지 언어로 구성된 corpus는 monolingual corpus라고 부르며, 두개의 언어로 구성된 corpus는 bilingual corpus라고 부릅니다. 더 많은 숫자의 언어로 구성되면 multilingual corpus가 됩니다. 이때, 예를 들어 아래와 같이 언어간에 쌍으로 구성된 corpus를 parallel corpus라고 부릅니다.

English	Korean
I love to go to school.	나는 학교에 가는 것을 좋아한다.
I am a doctor.	나는 의사 입니다.

이러한 corpus가 많을 수록, 오류가 없을 수록 우리는 NLP machine learning을 더욱 정교하게 할 수 있고, model의 정확도를 높힐 수 있습니다. 우리은 이 chapter에서 corpus를 수집하고 정제하여 효율성을 높힐 수 있도록하는 preprocessing에 대해서 다루어 보도록 하겠습니다.

Preprocess Overview

NLP에서의 전처리는 목적에 따라 약간씩 다르지만 대체로 아래와 같이 비슷한 과정을 지니고 있습니다. 이번 챕터에서는 이러한 과정에 대해서 다루고 넘어갈 것입니다.

1. 말뭉치(corpus) 수집
2. 말뭉치(corpus) 정제(normalization)
    1. Cleaning
    2. Sentence Tokenization
    3. Tokenization
    4. Subword segmentation

Collecting Corpus

Corpus를 구하는 방법은 여러가지가 있습니다. Open된 데이터를 사용할 수도 있고, 구매를 할 수도 있습니다. Open된 데이터는 주로 각종 Task(e.g. Sentiment Analysis and Machine Translation)들의 대회 또는 논문을 위한 데이터 입니다. 여기서는 crawling을 통한 수집을 주로 다루도록 하겠습니다. 다양한 웹사이트에서 crawling을 수행 할 수 있는만큼, 다양한 domain의 corpus를 모을 수 있습니다. 만약 특정 domain만을 위한 NLP task가 아니라면, 특정 domain에 편향(biased)되지 않도록 최대한 다양한 domain에서 corpus를 수집하는 것이 중요합니다.

하지만 무작정 웹사이트로부터 corpus를 crawling하는 것은 법적인 문제가 될 수 있습니다. 저작권 뿐만 아니라, 불필요한 traffic을 웹서버에 가중시킴으로써, 문제가 생길 수 있습니다. 따라서 올바른 방법으로 적절한 웹사이트에서 상업적인 목적이 아닌 경우에 제한된 crawling을 할 것을 권장합니다. 해당 웹사이트의 Crawling에 대한 허용 여부는 그 사이트의 robots.txt를 보면 확인 할 수 있습니다. 예를 들어 TED의 robot.txt는 다음과 같이 확인 할 수 있습니다.

```
$ wget https://www.ted.com/robots.txt
$ cat robots.txt
User-agent: *
Disallow: /latest
Disallow: /latest-talk
Disallow: /latest-playlist
Disallow: /people
```

```
Disallow: /profiles
Disallow: /conversations

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

모든 User-agent에 대해서 일부 경우에 대해서 disallow 인 것을 확인 할 수 있습니다. robots.txt에 대한 좀 더 자세한 내용은 http://www.robotstxt.org/ 에서 확인 할 수 있습니다.

Crawling 할 때에는 selenium이라는 package를 사용하여 web-browser driver를 직접 control하여 crawling을 수행하곤 합니다. 이때, phantomJS나 chrome과 같은 headless browser를 사용하면, 실제 사용자가 화면에서 보던 것과 같은 page를 다운로드 받을 수 있습니다. (예를 들어 wget과 같이 그냥 다운로드 할 경우, 브라우저와 다른 모양의 페이지가 다운로드 되는 경우도 많습니다.) 이후, beautiful-soup이라는 package를 사용하여 HTML 코드를 쉽고 간단하게 parsing할 수 있습니다.

Monolingual Corpora

사실 가장 손 쉽게 구할 수 있는 종류의 corpus 입니다. 경우에 따라서 Wikipedia나 각종 Wiki에서는 dump 데이터를 제공하기도 합니다. 따라서 해당 데이터를 다운로드 및 수집하는 것은 손쉽게 대량의 corpus를 얻을 수 있는 방법 중에 하나 입니다. 아래는 domain에 따른 대표적인 corpus의 수집 방식 입니다. 또한 Kaggle에서도 많은 종류의 dataset이 대량으로 upload되어 있으니 필요에 따라 다운로드 받아 사용하면 매우 유용합니다. Crawling을 수행 할 때에는 해당 사이트의 robots.txt 등을 확인하여 적법한 절차를 통해 crawling을 수행하길 권장 합니다.

문체	domain	수집처	정제 난이도
대화체	일반	채팅 로그	높음
대화체	일반	블로그	높음
문어체	시사	뉴스 기사	낮음

문체	domain	수집처	정제 난이도
문어체	과학, 교양, 역사 등	Wikipedia	중간
문어체	과학, 교양, 역사, 서브컬쳐 등	나무위키	중간
대화체	일반(각 분야별 게시판 존재)	클리앙	중간
문어체	일반, 시사 등	PGR21	중간
대화체	일반	드라마, 영화 자막	낮음

자막은 저작권이 있는 경우가 많기 때문에 저작권 관련 정보를 잘 확인 하는 것도 중요합니다.

Multilingual Corpora

Machine Translation을 위한 parallel corpus를 구하는 것은 monolingual corpus에 비해서 상당히 어렵습니다. Crawling을 수행 할 때에는 해당 사이트의 robots.txt 등을 확인하여 적법한 절차를 통해 crawling을 수행하길 권장 합니다. 자막은 저작권이 있는 경우가 많기 때문에 저작권 관련 정보를 잘 확인 하는 것도 중요합니다.

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

자막을 parallel corpus로 사용할 경우 몇가지 문제점이 있습니다. 번역 품질의 저하로 인한 문제는 둘째치고, 'he'나 'she'같은 대명사가 사람 이름과 같은 고유명사로 표현되어 있는 경우도 많습니다. 따라서, 이러한 문제점들을 두루 고려해야 할 필요성이 있습니다.

Speech with Transcript

Speech 데이터와 그와 함께 annotated된 transcript 데이터를 구하는 것은 정말 어렵습니다. 주로 연구용으로 공개 된 데이터들을 이용하거나 자막을 활용하여 영상에서 추출 하는 방법도 생각 해 볼 수 있습니다. # Cleaning

정제(normalization)는 텍스트를 사용하기에 앞서 필수적인 과정입니다. 원하는 Task에 따라, 또는 application에 따라서 필요한 정제의 수준 또는 깊이가 다를 수 있습니다. 예를 들어 음성인식을 위한 언어모델의 경우에는 사람의 음성을 그대로 받아적어야 하기 때문에, 괄호 또는 별표와 같은 기호나 특수문자들은 포함되어서는 안됩니다. 또한, 전화번호나 이메일 주소, 신용카드 번호와 같은 개인정보나 민감한 정보들은 제거되거나 변조된 채로 모델링 되야 할 수도 있습니다. 각 case에 따라서 필요한 형태를 얻어내기 위해서는 효과적인 정제 방법을 사용해야 합니다.

전각문자 제거

대부분의 중국어와 일본어 문서, 그리고 일부 한국어 문서들은 숫자, 영자, 기호가 전각문자로 되어 있는 경우가 있습니다. 이러한 경우에 일반적으로 사용되는 반각문자로 변환해 주는 작업이 필요합니다. 대표적으로 반각/전각문자로 혼용되는 문자들은 아래와 같습니다. 아래의 문자들을 각 문자에 해당하는 반각문자로 바꾸어주는 작업이 필요합니다.

" ’

대소문자 통일

일부 영어 corpus에서는 또는 corpus마다 약자 등에서의 대소문자 표현이 통일되지 않은 경우가 있습니다. 예를 들어 New York City의 줄임말(약자)인 NYC의 경우에 아래와 같은 다양한 표현이 가능합니다.

번호	New York City
1	NYC
2	nyc
3	N.Y.C.
4	N.Y.C

따라서 이러한 다양한 표현을 일원화 시켜주는 것은 한개의 의미를 지니는 여러 단어의 형태를 하나로 통일시켜 줌으로써, Sparsity를 감소시키는 효과를 거둘 수 있습니다. 하지만, deep learning에 접어들어 word embedding을 통한 효과적인 표현이 가능해지면서, 다양한 표현을 비슷한 값의 vector로 나타낼 수 있게 되어, 대소문자 통일과 같은 작은 문제(전체 corpus에서 차지하는 비율이 적은 문제)에 대한 해결 필요성이 줄어들었습니다.

Regular Expression

또한, crawling을 통해 얻어낸 다량의 corpus는 보통 특수문자, 기호 등에 의해서 noise가 섞여 있는 경우가 많습니다. 또한, 웹사이트의 성격에 따라 일정한 패턴을 띄고 있는 경우도 많습니다. 이러한 noise들을 효율적으로 감지하고 없애기 위해서는 regular expression (정규식)의 사용은 필수적 입니다. 따라서, 이번 section은 regular expression(regex)에 대해서 살펴 봅니다.

[]의 사용

```
[2345cde]
(2|3|4|5|c|d|e)
```

-의 사용

```
[2-5c-e]
```

[^]의 사용

```
[^2-5c-e]
```

()의 사용

```
(x)(yz)
```

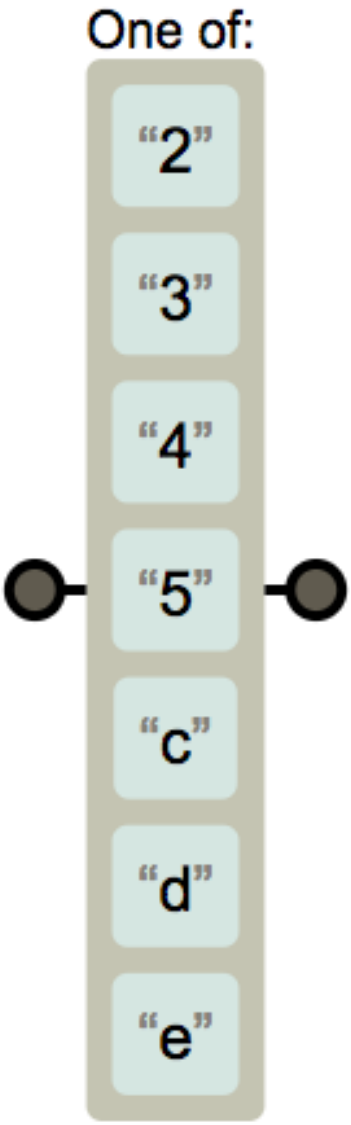


Figure 1:

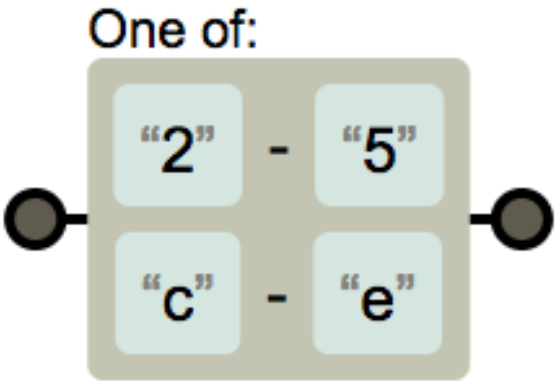


Figure 2:

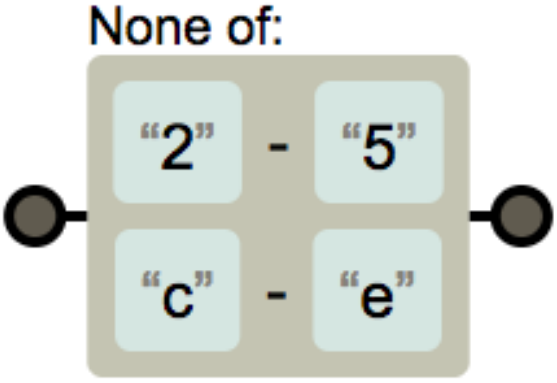


Figure 3:

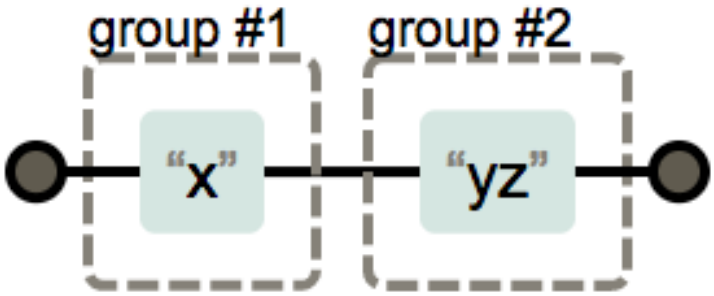


Figure 4:

|의 사용

(x|y)

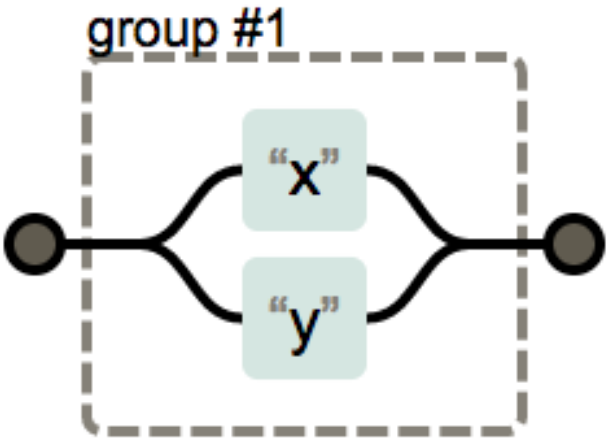


Figure 5:

?, *, +의 사용

?는 앞의 수식하는 부분이 나타나지 않거나 한번만 나타날 경우 사용 합니다.

x?

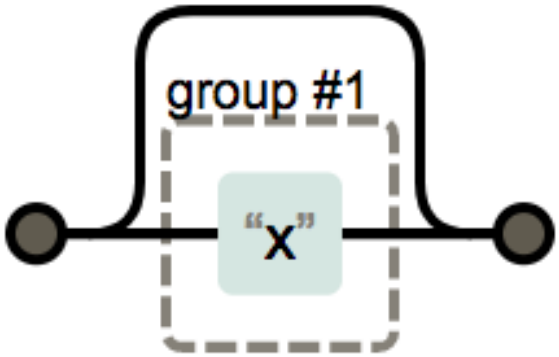


Figure 6:

+는 앞의 수식하는 부분이 한 번 이상 나타날 경우 사용 합니다.

```
x+
```

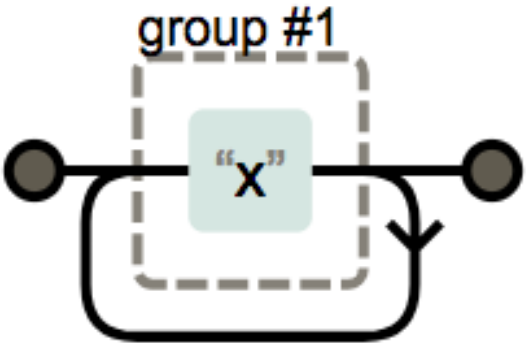


Figure 7:

*는 앞의 수식하는 부분이 나타나지 않거나 여러번 나타날 경우 사용 합니다.

```
x*
```

{n}, {n,}, {n,m}의 사용

```
x{n}
```

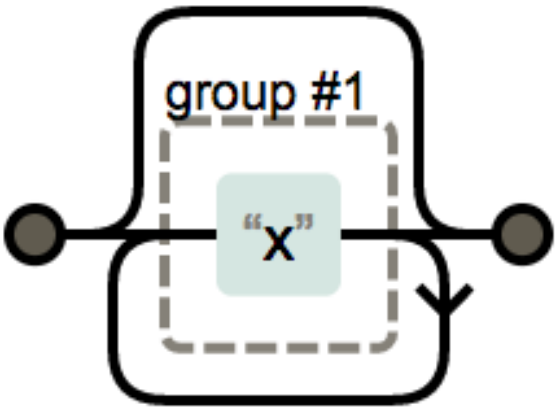


Figure 8:

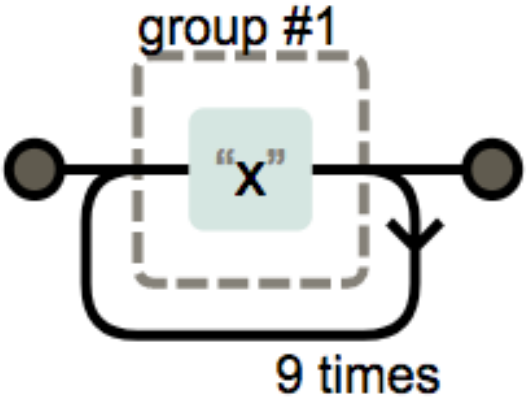


Figure 9:

`x{n,}`

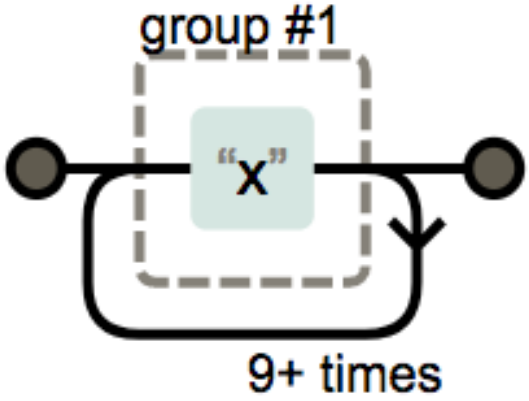


Figure 10:

`x{n,m}`

. 의 사용

.

^와 $의 사용

`^x$`

지정문자의 사용

Meta Characters	Description
₩s	공백문자(white space)
₩S	공백문자를 제외한 모든 문자
₩w	alphanumeric(알파벳 + 숫자) + '_' ([A-Za-z0-9_]와 같음)
₩W	non-alphanumeric 문자 '_'도 제외 ([^A-Za-z0-9_]와 같음)
₩d	숫자 ([0-9]와 같음)

Meta Characters	Description
₩D	숫자를 제외한 모든 문자 ([^0-9]와 같음)

Example

실제 예를 들어 보겠습니다. NLP 문제를 풀고 있는 중에, 문서의 마지막 줄에 종종 아래와 같은 개인의 전화번호 정보가 포함되어 있는 문서를 dataset으로 사용하려 할 때, 해당 정보를 제외하고 사용하고 싶다고 가정 해 보겠습니다.

> Hello Ki,
> I would like to introduce regular expression in this section.
> ~~
> Thank you!
> Sincerely,
> Ki: +82-10-1234-5678

무턱대고 마지막 줄을 지우기에는 마지막 줄에 전화번호 정보가 없는 경우도 많기 때문에 선택적으로 지워야 할 것 같습니다. 따라서 데이터를 쭈욱 훑어가며 살펴보니, 마지막 줄은 아래와 같은 규칙을 따르는 것 같습니다.

- 이름이 전화번호 앞에 나올 수도 있다.
- 이름 뒤에는 콜론(:)이 나올 수도 있다.
- 콜론 앞/뒤로는 공백(tab 포함)이 다수가 존재할 수도 있다.
- 전화번호는 국가번호를 포함할 수도 있다.
- 국가번호는 최대 3자리이다.
- 국가번호의 앞에는 '+'가 붙을 수도 있다.
- 전화번호 사이에 '-'가 들어갈 수도 있다.
- 전화번호는 빈칸이 없이 표현 된다.
- 전화번호의 맨 앞과 지역번호(또는 010)의 다음에는 괄호가 들어갈 수도 있다.
- 괄호는 한쪽만 나올 수도 있다.
- 지역번호 자리의 맨 처음 나오는 0은 빠질 수도 있다. 즉, 2자리가 될 수도 있다.
- 지역번호 다음 번호 그룹은 3에서 4자리 숫자이다.
- 마지막은 항상 4자리 숫자이다.

위의 규칙을 따르는 regular expression을 표현하면 아래와 같습니다.

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

위의 수식을 그림으로 표현하면 아래와 같습니다.

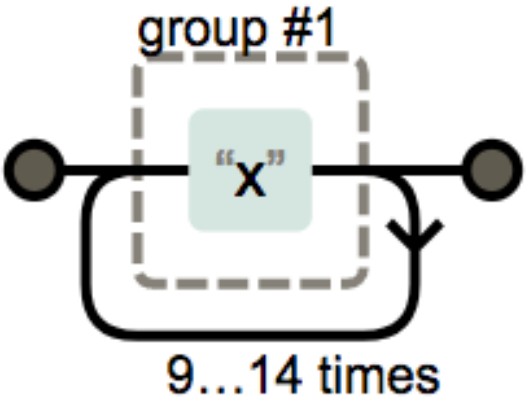


Figure 11:

Figure 12:

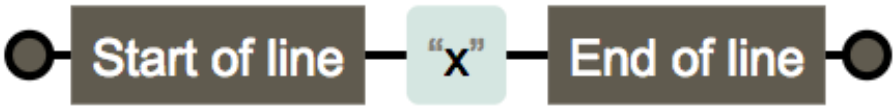
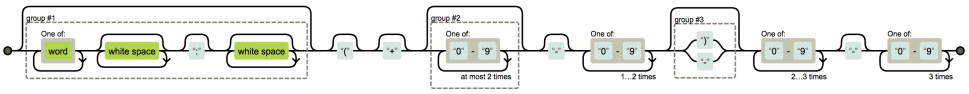


Figure 13:



[Image from regexper.com]

Python에서의 Regular Expression

```
>>> import re
>>> regex = r"([\w]+\s*:?\s*)?\(?\+?([0-9]{1,3})?\-?[0-9]{2,3}(\)|\-)?[0-9]{3,4}\-
>>> 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'
```

re.sub

r“”

₩1, ₩2, … 치환자의 사용

이제까지 다룬 정규식 표현만으로도 많은 부분을 cover할 수 있지만, 아직 2% 부족함이 남아 있습니다. 예를 들어 아래와 같은 case를 다루어 보겠습니다.

알파벳(소문자) 사이에 있는 숫자를 제거하라.
abcdefg
12345
ab12
a1bc2d
12ab
a1b
1a2
a1
1a
hijklmnop

만약 그냥 [0-9]+으로 숫자를 찾아서 없애면 두번째 줄의 숫자만 있는 경우와 숫자가 가장자리에 있는 경우도 사라지게 됩니다. 그럼 어떻게 해야 할까요? 이때 유용한 방법이 치환자를 사용하는 것 입니다.

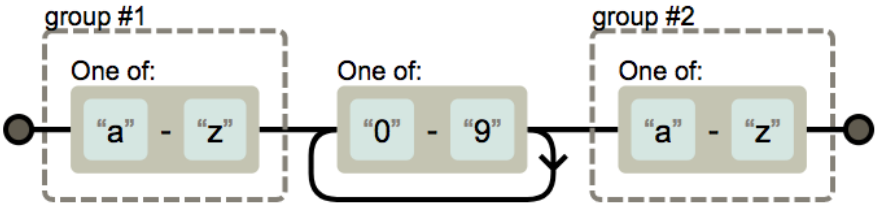


Figure 14:

괄호로 묶인 부분은 group으로 지정이 되고, 바뀔 문자열 내에서 역슬래시(₩)와 함께 숫자로 가리킬 수 있습니다. 예를 들어 아래와 같이 구현 할 수 있습니다.

```
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')])
```

위의 방법은 굳이 python과 같은 programming 언어가 아니더라도 sublime text와 같이 지원되는 text editor들이 있으니 editor 상에서의 정제를 할 때에도 유용하게 사용할 수 있습니다.

Sentence Tokenization

우리가 다루고자 하는 task들은 입력 단위가 문장단위인 경우가 많습니다. 즉, 한 line에 한 문장만 있어야 합니다. 여러 문장이 한 line에 있거나, 한 문장이 여러 line에 걸쳐 있는 경우에 대해서 sentence tokenization이 필요합니다. 단순이 마침표만을 기준으로 sentence tokenization을 수행하게 되면, U.S. 와 같은 영어 약자나 3.141592와 같은 소숫점 등 여러가지 문제점에 마주칠 수 있습니다. 따라서 직접 tokenization해주기 보다는 NLTK에서 제공하는 tokenizer를 이용하기를 권장합니다. – 물론, 이 경우에도 완벽하게 처리하지는 못하며, 일부 추가적인 전/후처리가 필요할 수도 있습니다.

Tokenization

한 line에 여러 문장이 들어 있는 경우에 대한 Python script 예제.

before:

```
  TED      1,000     TED       .                                --      .            .
```

```
import sys, fileinput, re
from nltk.tokenize import sent_tokenize

if __name__ == "__main__":
    for line in fileinput.input():
        if line.strip() != "":
            line = re.sub(r'([a-z])\.([A-Z])', r'\1. \2', line.strip())

            sentences = sent_tokenize(line.strip())

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

after:

```
  TED      1,000     TED       .
                        --      .
          .
                  .
```

```
      1,000    ,          1,000            .
       1,000             ?
```

Combine and Tokenization

여러 line에 한 문장이 들어 있는 경우에 대한 Python script 예제.

before:

```
  TED     1,000    TED     .

            --
    .          .
                .
      1,000    ,
         1,000           .

1,000             ?
1,000  TED
```

```
import sys, fileinput
from nltk.tokenize import sent_tokenize

if __name__ == "__main__":
    buf = []

    for line in fileinput.input():
        if line.strip() != "":
            buf += [line.strip()]
            sentences = sent_tokenize(" ".join(buf))

            if len(sentences) > 1:
                buf = sentences[1:]

                sys.stdout.write(sentences[0] + '\n')

    sys.stdout.write(" ".join(buf) + "\n")
```

after:

```
  TED      1,000     TED       .
                          --       .
          .
                    .
      1,000     ,           1,000              .
       1,000                ?
1,000  TED
```

Part of Speech Tagging, Tokenization (Segmentaion)

우리가 하고자 하는 task에 따라서 Part-of-speech (POS) tagging 또는 단순한 segmentation을 통해 normalization을 수행합니다. 주로 띄어쓰기에 대해서 살펴보도록 하겠습니다. 아래에 정리한 프로그램들이 기본적으로 띄어쓰기만 제공하는 프로그램이 아니더라도, tag정보 등을 제외하게 되면 띄어쓰기(segmentation)를 수행하는 것과 동일합니다.

띄어쓰기(tokenization or segmentation)에 대해서 살펴 보겠습니다. 한국어, 영어와 달리 일본어, 중국어의 경우에는 띄어쓰기가 없는 경우가 없습니다. 또한, 한국어의 경우에도 띄어쓰기가 도입된 것은 근대에 이르러서이기 때문에, 띄어쓰기의 표준화가 부족하여 띄어쓰기가 중구난방인 경우가 많습니다. 특히나, 한국어의 경우에는 띄어쓰기가 문장의 의미 해석에 큰 영향을 끼치지 않기 때문에 더욱이 이런 현상은 가중화 됩니다. 따라서, 한국어의 경우에는 띄어쓰기가 이미 되어 있지만, 제각각인 경우가 많기 때문에 normalization을 해주는 의미로써 다시한번 적용 됩니다. 또한, 교착어로써 접사를 어근에서 분리해주는 역할도 하여, sparseness를 해소하는 역할도 합니다. 이러한 한국어의 띄어쓰기 특성은 앞서 Why Korean NLP is Hell에서 다루었습니다.

일본어, 중국어의 경우에는 띄어쓰기가 없기 때문에, 애초에 모든 문장들이 같은 띄어쓰기 형태(띄어쓰기가 없는 형태)를 띄고 있지만, 적절한 language model 구성을 위하여 띄어쓰기가 필요합니다. 다만, 중국어의 경우에는 character 단위로 문장을 실제 처리하더라도 크게 문제가 없기도 합니다.

영어의 경우에는 기본적으로 띄어쓰기가 있고, 대부분의 경우 매우 잘 표준을 따르고 있습니다. 다만 language model을 구성함에 있어서 좀 더 용이하게 해 주기 위한 일부 처리들을 해주면 더 좋습니다. NLTK를 사용하여 이러한 전처리를 수행합니다.

각 언어별 주요 프로그램들을 정리하면 아래와 같습니다.

언어	프로젝트	설명
한국어	Mecab-ko	C++로 일본어 Mecab을 wrap-ping하였으며, 속도가 가장 빠름. 설치가 종종 까다로움
한국어	KoNLPy	Python을 위한 wrapping(복합) 설치 가능하며 사용이 쉬우나, 일부 tag-ger의 경우에는 속도가 느림
일본어	Mecab-ko	C++로 속도가 빠르다
중국어	Stanford Parser	Java로 미국 스탠포드에서 개발
중국어	PKU Parser	Java로 북경대에서 개발. Stan-ford Parser와 성능 차이가 거의 없음

언어	프로젝트	개발형태
중국어	Jieba	Python 기반으로 최근에 개발됨. Python으로 제작되어 시스템 구성에 용이

사실 일반적인 또는 전형적인 쉬운 문장(표준어를 사용하며 문장 구조가 명확한 문장)의 경우에는 대부분의 프로그램들의 성능이 비슷합니다. 하지만 각 프로그램 성능의 차이를 만드는 것은 바로 신조어나 보지 못한 고유명사를 처리하는 능력입니다. 따라서 어떤 한 프로그램을 선택하고자 할때에, 그런 부분에 초점을 맞추어 성능을 평가하고 선택해야 할 필요성이 있습니다.

Korean

Mecab

한국어 tokenization에 가장 많이 사용되는 프로그램은 Mecab입니다. 원래 Mecab은 일본어 tokenizer 오픈소스로 개발되었지만, 이를 한국어 tokenizing에 성공적으로 적용시켜 널리 사용되고 있습니다. 아래와 같이 설치 가능합니다. - 참고사이트: http://konlpy-ko.readthedocs.io/ko/v0.4.3/install/#ubuntu

```
$ sudo apt-get install curl
$ bash <(curl -s https://raw.githubusercontent.com/konlpy/konlpy/master/scripts/me
```

위의 instruction을 따라 정상적으로 설치 된 Mecab의 경우에는, KoNLPy에서 불러 사용할 수 있습니다. - 참고사이트: http://konlpy-ko.readthedocs.io/ko/v0.4.3/api/konlpy.tag/#mecab-class

또는 아래와 같이 bash상에서 mecab명령어를 통해서 직접 실행 가능하며 standard input/output을 사용하여 tokenization을 수행할 수 있습니다.

```
$ echo "   ,    !" | mecab
   NNG,*,T, ,*,*,*,*
```

```
    XSV,*,F, ,*,*,*,*
   EP+EF,*,F,  ,Inflect,EP,EF, /EP/*+ /EF/*
,   SC,*,*,*,*,*,*,*
   VA,*,T,  ,*,*,*,*
  EF,*,F,   ,*,*,*,*
!   SF,*,*,*,*,*,*,*
EOS
```

만약, 단순히 띄어쓰기만 적용하고 싶을 경우에는 -O wakati 옵션을 사용하여 실행시키면 됩니다. 마찬가지로 standard input/output을 통해 수행 할 수 있습니다.

```
$ echo "   ,    !" | mecab -O wakati
      ,      !
```

위와 같이 기본적으로 어근과 접사를 분리하는 역할을 수행하며, 또는 잘못된 띄어쓰기에 대해서도 올바르게 교정을 해 주는 역할 또한 수행 합니다. 어근과 접사를 분리함으로써 어휘의 숫자를 효과적으로 줄여 sparsity를 해소할 수 있습니다.

KoNLPy

KoNLPy는 여러 한국어 tokenizer 또는 tagger들을 모아놓은 wrapper를 제공합니다. 이름에서 알 수 있듯이 Python wrapper를 제공하므로 시스템 연동 및 구성이 용이할 수 있습니다. 하지만 내부 라이브러리들은 각기 다른 언어(Java, C++ 등)로 이루어져 있어 호환성에 문제가 생기기도 합니다. 또한, 일부 library들은 상대적으로 제작/업데이트가 오래되었고, java로 구현되어 있어 C++로 구현되어 있는 Mecab에 비해 속도면에서 불리함을 갖고 있습니다. 따라서 대용량의 corpus를 처리할 때에 불리하기도 합니다. 하지만 설치 및 사용이 쉽고 다양한 library들을 모아놓았다는 장점 때문에 사랑받고 널리 이용되고 있습니다.

참고사이트: http://konlpy-ko.readthedocs.io/

English

Moses

영어의 경우에는 위에서 언급했듯이, 기본적인 띄어쓰기가 잘 통일되어 있는 편이기 때문에, 띄어쓰기 자체에 대해서는 큰 normalization 이슈가 없습니다. 다만 comma, period, quotation 등을 띄어주어야 합니다. Moses에서 제공하는 tokenizer를

통해서 이런 처리를 수행하게 됩니다. NLTK 예전 버전(3.2.5)에서는 Moses를 랩핑(wrapping)하여 tokenizer를 제공하였습니다. 3.2.5버전의 NLTK를 설치하기 위한 pip 명령어는 다음과 같습니다.

```
$ pip install nltk==3.2.5
```

아래는 NLTK를 사용한 Python script 예제 입니다.

before: 〉North Korea's state mouthpiece, the Rodong Sinmun, is also keeping mum on Kim's summit with Trump while denouncing ever-tougher U.S. sanctions on the rogue state.

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

t = MosesTokenizer()

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

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

after: 〉North Korea 's state mouthpiece , the Rodong Sinmun , is also keeping mum on Kim 's summit with Trump while denouncing ever-tougher U.S. sanctions on the rogue state .

Chinese

Stanford Parser

스탠포드 대학교에서 제작한 중국어 parser입니다. Java로 제작되었습니다.

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

JIEBA

비교적 가장 늦게 개발/업데이트가 이루어진 Jieba library는, 사실 성능상에서 다른 parser들과 큰 차이가 없지만, python으로 구현되어 있어 우리가 실제 상용화를 위한 배치 시스템을 구현 할 때에 매우 쉽고 용이할 수 있습니다. 특히, 딥러닝을 사용한 머신러닝 시스템은 대부분 Python위에서 동작하기 때문에 일관성 있는 시스템을 구성하기 매우 좋습니다.

참고사이트: https://github.com/fxsjy/jieba # Align Parallel Corpus

대부분의 parallel corpora는 여러 문장 단위로 align이 되어 있는 경우가 많습니다. 이러한 경우에는 한 문장씩에 대해서 align을 해주어야 합니다. 또한, 이러한 과정에서 일부 parallel 하지 않은 문장들을 걸러내야 하고, 문장 간 align이 잘 맞지 않는 경우 align을 재정비 해 주거나 아예 걸러내야 합니다. 이러한 과정에 대해서 살펴 봅니다.

Process Overview for Parallel Corpus Alignment

Alignment를 수행하기 위한 전체 과정을 요약하면 아래와 같습니다.

1. Building a dictionary between source language to target language.
    1. Collect and normalize (clean + tokenize) corpus for each language.
    2. Get word embedding vector for each language.
    3. Get word-level-translator using MUSE.
2. Align collected semi-parallel corpus using Champollion.
    1. Sentence-tokenize for each language.
    2. Normalize (clean + tokenize) corpus for each language.
    3. Align parallel corpus using Champollion.

Building Dictionary

만약 기존에 단어 (번역) 사전을 구축해 놓았다면 그것을 이용하면 되지만, 단어 사전을 구축하는 것 또한 비용이 들기 때문에, 일반적으로는 쉽지 않습니다. 따라서, 단어 사전을 구축하는 것 또한 자동으로 할 수 있습니다.

Facebook의 MUSE는 parallel corpora가 없는 상황에서 사전을 구축하는 방법을 제시하고, 코드를 제공합니다. 각 Monolingual corpus를 통해 구축한 언어 별 word embedding vector에 대해서 다른 언어의 embedding vector와 mapping하도록 함으로써, 단어 간 번역을 할 수 있습니다. 이는 각 언어 별 corpus가 많을 수록

embedding vector가 많을수록 더욱 정확하게 수행 됩니다. MUSE는 parallel corpora가 없는 상황에서도 수행 할 수 있기 때문에 unsupervised learning이라고 할 수 있습니다.

아래는 MUSE를 통해 구한 영한 사전의 일부로써, 꽤 정확한 단어 간의 번역을 볼 수 있습니다. 이렇게 구성한 사전은 champollion의 입력으로 사용되어, champollion은 이 사전을 바탕으로 parallel corpus의 sentence alignment를 수행합니다. 〈〉을 delimiter로 사용하여 한 라인에 source 언어의 단어와 target 언어의 단어를 표현 합니다.

```
stories <>
stories <>
contact <>
contact <>
contact <>
green <>
green <>
green <>
dark <>
dark <>
dark <>
song <>
song <>
song <>
salt <>
```

Align via Champollion

Chapollion Toolkit(CTK)는 parallel corpus의 sentence alignement를 수행하는 open-source입니다. Perl을 사용하여 구현되었으며, 이집트 상형문자를 처음으로 해독해낸 역사학자 Champollion의 이름을 따서 명명되었습니다.

(Jean-Fran□ois Champollion, Image from Wikipedia)

기-구축된 단어 (번역) 사전을 이용하거나, 위와 같이 자동으로 구축한 단어 사전을 참고하여 Champollion은 sentence alignment를 수행합니다. 여러 line으로 구성된 각 언어 별 하나의 document에 대해서 sentence alignment를 수행한 결과는 아래와 같습니다.

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

위의 결과를 해석해 보면, target 언어의 1, 2, 3번째 문장은 짝을 찾지 못하고 버려졌고, source 언어의 1, 2, 3번째 문장은 각각 target 언어의 4, 5, 6번째 문장과 mapping 된 것을 알 수 있습니다. 또한, source 언어의 4, 5번째 두 문장은 target 언어의 7번 문장에 동시에 mapping 된 것을 볼 수 있습니다.

이와 같이 어떤 문장들은 버려지기도 하고, 일대일 mapping이 이루어지기도 하며,

일대다, 다대일 mapping이 이루어지기도 합니다.

아래는 champollion을 쉽게 사용하기 위한 Python script 예제입니다. CTK_ROOT에 Chapollion Toolkit의 위치를 지정하여 사용할 수 있습니다.

```
import sys, argparse, os

BIN = CTK_ROOT + "/bin/champollion"
CMD = "%s -c %f -d %s %s %s %s"
OMIT = "omitted"
INTERMEDIATE_FN = "./tmp/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 = 1.2750)

    config = p.parse_args()

    return config

if __name__ == "__main__":
    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, 
    os.system(cmd)

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

특기할 점은 ratio parameter의 역할입니다. 이 parameter는 실제 champollion의 -c 옵션으로 mapping되어 사용되는데, champollion 상에서의 설명은 다음과 같습니다.

```
$ ./champollion
usage: ./champollion [-hdscn] <X token file> <Y token file> <alignment file>

      -h       : this (help) message
      -d dictf : use dictf as the translation dictionary
      -s xstop : use words in file xstop as X stop words
      -c n     : number of Y chars for each X char
      -n       : disallow 1-3, 3-1, 1-4, 4-1 alignments
               (faster, lower performance)
```

즉, source language의 charactor 당 target language의 charactor 비율을 의미합니다. 이를 기반하여 champollion은 문장 내 모든 단어에 대해서 번역 단어를 모르더라도 sentence alignment를 수행할 수 있게 됩니다. # Segmentation using Subword (Byte Pare Encoding, BPE)

Byte Pair Encoding (BPE) 알고리즘을 통한 subword segmentation은 [Sennrich at el.2015]에서 처음 제안되었고, 현재는 주류 전처리 방법으로 자리잡아 사용되고 있습니다.

Subword segmenation은 기본적으로 단어는 여러 sub-word들의 조합으로 이루어진다는 가정 하에 적용 되는 알고리즘입니다. 실제로 영어나 한국어의 경우에도 latin어와 한자를 기반으로 형성 된 언어이기 때문에, 많은 단어들이 sub-word들로 구성되어 있습니다. 따라서 적절한 subword detection을 통해서 subword 단위로 쪼개어 주면 어휘수를 줄일 수 있고 sparsity를 효과적으로 감소시킬 수 있습니다.

언어	단어	조합
영어	Concentrate	con(=together) + centr(=center) + ate(=make)

언어	단어	조합
한국어	집중(□□)	□(모을 집) + □(가운데 중)

Sparsity 감소 이외에도, 가장 대표적인 subword segmentation으로 얻는 효과는 unknown(UNK) token에 대한 효율적인 대처 입니다. Natrual Language Generation (NLG)를 포함한 대부분의 딥러닝 NLP 알고리즘들은 문장을 입력으로 받을 때 단순히 word sequence로써 받아들이게 됩니다. 따라서 UNK가 나타나게 되면 이후의 language model의 확률은 매우 망가져버리게 됩니다. 따라서 적절한 문장의 encoding또는 generation이 어렵습니다. – 특히 문장 generation의 경우에는 이전 단어를 기반으로 다음 단어를 예측하기 때문에 더더욱 어렵습니다.

하지만 subword tokenization을 통해서 신조어나 typo(오타) 같은 UNK에 대해서 subword나 character 단위로 쪼개줌으로써 known token들의 조합으로 만들어버릴 수 있습니다. 이로써, UNK 자체를 없앰으로서 효율적으로 UNK에 대처할 수 있고, NLP 결과물의 품질을 향상시킬 수 있습니다.

Example

아래와 같이 BPE를 적용하면 원래의 띄어쓰기 공백 이외에도 BPE의 적용으로 인한 공백이 추가됩니다. 따라서 원래의 띄어쓰기와 BPE로 인한 띄어쓰기를 구분해야 할 필요성이 있습니다. 그래서 특수문자(기존의 _가 아닌 □)를 사용하여 기존의 띄어쓰기(또는 단어의 시작)를 나타냅니다. 이를 통해 후에 설명할 detokenization에서 문장을 BPE이전의 형태로 원상복구 할 수 있습니다.

한글 Mecab에 의해 tokenization 된 원문 **현재 TED 웹 사이트 에 는 1 , 000 개 가 넘 는 TED 강연 들 이 있 습니다 . 여기 계신 여러분 의 대다수 는 정말 대단 한 일 이라고 생각 하 시 겠 죠 – 전 다릅니다 . 전 그렇게 생각 하 지 않 아요 . 저 는 여기 한 가지 문제점 이 있 다고 생각 합니다 . 왜냐하면 강연 이 1 , 000 개 라는 것 은 , 공유 할 만 한 아이디어 들 이 1 , 000 개 이상 이 라는 뜻 이 되 기 때문 이 죠 . 도대체 무슨 수로 1 , 000 개 나 되 는 아이디어 를 널리 알릴 건가요 ? 1 , 000 개 의 TED 영상 전부 를 보 면서 그 모든 아이디어 들 을 머리 속 에 집 어 넣 으려고 해도 , 250 시간 이상 의 시간 이 필요 할 겁니다 . 250 시간 이상 의 시간 이 필요 할 겁니다 . 간단 한 계산 을 해 봤 는데요 . 정말 그렇게 하 는 경우 1 인 당 경제 적 손실 은 15 , 000 달러 정도 가 됩니다 .**

한글 BPE 적용 **□현재 □TED □웹 □사이트 □에 □는 □1 □, □000 □개 □가 □넘 □는 □TED □강 연 □들 □이 □있 □습니다 □. □여기 □계 신 □여러분**

□의 □대 다 수 □는 □정말 □대단 □한 □일 □이 □라고 □생각 □하 □시 □겠 □죠 □ - □전 □다 릅니다 □. □전 □그렇게 □생각 □하 □지 □않 □아요 □. □저 □는 □여기 □한 □가지 □문 제 점 □이 □있 □다고 □생각 □합니다 □. □왜냐하면 □강 연 □이 □1 □, □000 □개 □라는 □것 □은 □, □공유 □할 □만 □한 □아이디어 □들 □이 □1 □, □000 □개 □이상 □이 □라는 □뜻 □이 □되 □기 □때문 □이 □죠 □. □도 대체 □무슨 □수 로 □1 □, □000 □개 □나 □되 □는 □아이디어 □를 □널 리 □알 릴 □건 가요 □? □1 □, □000 □개 □의 □TED □영상 □전부 □를 □보 □면서 □그 □모든 □아이디어 □들 □을 □머리 □속 □에 □집 □어 □넣 □으 려고 □해도 □, □2 50 □시간 □이상 □의 □시간 □이 □필요 □할 □겁니다 □. □2 50 □시간 □이상 □의 □시간 □이 □필요 □할 □겁니다 □. □간단 □한 □계산 □을 □해 □봤 □는데요 □. □정말 □그렇게 □하 □는 □경우 □1 □인 □당 □경제 □적 □손 실 □은 □15 □, □000 □달러 □정도 □가 □됩니다 □.

영어 NLTK에 의해 tokenization이 된 원문 There 's currently over a thousand TED Talks on the TED website . And I guess many of you here think that this is quite fantastic , except for me , I don 't agree with this . I think we have a situation here . Because if you think about it , 1,000 TED Talks , that 's over 1,000 ideas worth spreading . How on earth are you going to spread a thousand ideas ? Even if you just try to get all of those ideas into your head by watching all those thousand TED videos , it would actually currently take you over 250 hours to do so . And I did a little calculation of this . The damage to the economy for each one who does this is around $ 15,000 . So having seen this danger to the economy , I thought , we need to find a solution to this problem . Here 's my approach to it all .

영어 BPE 적용 □There □'s □currently □over □a □thous and □TED □T al ks □on □the □TED □we b site □. □And □I □guess □many □of □you □here □think □that □this □is □quite □f ant as tic □, □ex cept □for □me □, □I □don □'t □agree □with □this □. □I □think □we □have □a □situation □here □. □Because □if □you □think □about □it □, □1 ,000 □TED □T al ks □, □that □'s □over □1 ,000 □ideas □worth □sp reading □. □How □on □earth □are □you □going □to □spread □a □thous and □ideas □? □Even □if □you □just □try □to □get □all □of □those □ideas □into □your □head □by □watching □all □those □thous and □TED □vide os □, □it □would □actually □currently □take □you □over □2 50 □hours □to □do □so □. □And □I □did □a □little □cal cu lation □of □this □. □The □damage □to □the □economy □for □each □one □who □does □this □is □around □$ □15 ,000 □. □So □having □seen □this □dang er □to □the □economy □, □I □thought □, □we □need □to □find □a □solution □to □this □problem □.

□Here □'s □my □approach □to □it □all □.

원문	subword segmentation
대다수	대 + 다 + 수
문제점	문 + 제 + 점
건가요	건 + 가요
website	we + b + site
except	ex + cept
250	2 + 50
15,000	15 + ,000

위와 같이 subword segmentation에 의해서 추가적으로 쪼개진 단어들은 적절한 subword의 형태로 나누어진 것을 볼 수 있습니다. 특히, 숫자 같은 경우에는 자주 쓰이는 숫자 50이나 ,000의 경우 성공적으로 하나의 subword로 지정된 것을 볼 수 있습니다.
Detokenization

아래는 preprocessing에서 tokenization을 수행 하고, 다시 복원(detokenization)하는 과정을 설명 한 것입니다. 하나씩 따라가보도록 하겠습니다.

아래와 같은 영어 원문이 주어지면,

```
There's currently over a thousand TED Talks on the TED website.
```

각 언어 별 tokenizer(영어의 경우 NLTK)를 통해 tokenization을 수행하고, 기존의 띄어쓰기와 tokenization에 의해 수행된 공백과의 구분을 위해 □을 원래의 공백 위치에 삽입합니다.

```
There 's currently over a thousand TED Talks on the TED website .
```

여기에 subword segmentation을 수행하며 이전 step까지의 공백과 subword segmentation으로 인한 공백을 구분하기 위한 □를 삽입합니다.

```
There 's currently over a thous and TED T al ks on the TED we b site
```

이렇게 preprocessing 과정이 종료되었습니다. 이런 형태의 tokenized문장을 NLP 모델에 훈련시키면 같은 형태로 tokenized된 문장을 생성 해 낼 것 입니다. 그럼 이런 문장을 복원(detokenization)하여 사람이 읽기 좋은 형태로 만들어 주어야 합니다.

먼저 whitespace를 제거합니다.

```
There 's currently over a thousand TED Talks on the TED website .
```

그리고 □가 2개가 동시에 있는 문자열 □□을 white space로 치환합니다. 그럼 한 개 짜리 □만 남습니다.

```
There 's currently over a thousand TED Talks on the TED website .
```

마지막 남은 □를 제거합니다. 그럼 문장 복원이 완성 됩니다.

```
There's currently over a thousand TED Talks on the TED website.
```

Post Tokenization

아래는 기존의 공백과 전처리 한 단계로 인해 생성된 공백을 구분하기 위한 □을 삽입하는 python script 예제 입니다.

```
import sys

STR = ' '

if __name__ == "__main__":
    ref_fn = sys.argv[1]

    f = open(ref_fn, 'r')

    for ref in f:
        ref_tokens = ref.strip().split(' ')
        tokens = sys.stdin.readline().strip().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):
                tmp_buf += [tokens[idx]]
                idx += 1

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

```
                    break

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

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

    f.close()
```

위 script의 사용 방법은 아래와 같습니다. 주로 다른 tokenizer 수행 후에 바로 붙여 사용하여 좋습니다.

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

Detokenization

아래는 앞서 설명한 detokenization을 bash에서 sed 명령어를 통해 수행 할 경우에 대한 예제 입니다.

```
sed "s/ //g" | sed "s/ / /g" | sed "s/ //g" | sed "s/^\s//g"
```

또는 아래의 python script 예제 처럼 처리 할 수도 있습니다.

```
import sys

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

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

TorchText

사실 딥러닝 코드를 작성하다 보면, 신경망 모델 자체를 코딩하는 시간보다 그 모델을 훈련하도록 하는 코드를 짜는 시간이 더 오래걸리기 마련입니다. 데이터 입력을 준비하는 부분도 이에 해당 합니다. TorchText는 NLP 또는 텍스트와 관련된

기계학습 또는 딥러닝을 수행하기 위한 데이터를 읽고 전처리 하는 코드를 모아놓은 라이브러리 입니다.

이번 섹션에서는 TorchText를 활용한 기본적인 데이터 로딩 방법에 대한 실습을 해 보도록 하겠습니다. 좀 더 자세한 내용은 TorchText 문서를 참조 해 주세요.

How to install

TorchText는 pip을 통해서 쉽게 설치 가능 합니다. 아래와 같이 명령어를 수행하여 설치 합니다.

```
$ pip install torchtext
```

Example

사실 NLP분야에서 주로 사용되는 학습 데이터의 형태는 크게 3가지로 분류할 수 있습니다. 주로 우리의 신경망이 하고자 하는 바는 입력 x를 받아 알맞은 출력 y를 반환해 주는 함수의 형태이므로, x, y의 형태에 따라서 분류 해 볼 수 있습니다.

x 데이터	y 데이터	어플리케이션
corpus	클래스(class)	텍스트 분류(text classification), 감성분석(sentiment analysis)
corpus	–	언어모델(language modeling)
corpus	corpus	기계번역(machine translation), 요약(summarization), QnA

따라서 우리는 이 3가지 종류의 데이터 형태를 다루는 방법에 대해서 실습합니다. 사실 TorchText는 훨씬 더 복잡하고 정교한 함수들을 제공합니다. 하지만 글쓴이가 느끼기에는 좀 과도하고 직관적이지 않은 부분들이 많아, 제공되는 함수들의 사용을 최소화 하여 복잡하지 않고 간단한 방식으로 구현해보고자 합니다.

Reading Corpus with Labeling

첫번째 예제는 한 줄 안에서 클래스(class)와 텍스트(text)가 tab(탭, '₩t')으로 구분되어 있는 데이터의 입력을 받기 위한 예제 입니다. 주로 이런 예제는 텍스트 분류(text classification)을 위해 사용 됩니다.

```
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
                                                  )

        self.train_iter, self.valid_iter = data.BucketIterator.splits((train, vali
                                                                       batch_size
                                                                       device =
                                                                       shuffle =
                                                                       )


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

Reading Corpus

이 예제는 한 라인에 텍스트로만 채워져 있을 때를 위한 코드 입니다. 주로 언어모델(language model)을 훈련 시키는 상황에서 사용 될 수 있습니다.

```
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 dev
                                                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 dev
                                                shuffle = False,
                                                sort_key=lambda x: -len(x.text),
                                                sort_within_batch = True
                                                )

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

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 = []
        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
```

Reading Parallel(Bi-lingual) Corpus

아래의 코드는 텍스트로만 채워진 두개의 파일을 동시에 입력 데이터로 읽어들이는 예제 입니다. 이때, 두 파일의 corpus는 병렬(parallel) 데이터로 취급되어 같은 라인 수로 채워져 있어야 합니다. 주로 기계번역(machine translation)이나 요약(summarization) 등에 사용 할 수 있습니다.

```
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
                              )

        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', s
                                       max_length = max_length
                                       )
            valid = TranslationDataset(path = valid_fn, exts = exts,
                                       fields = [('src', self.src), ('tgt', s
                                       max_length = max_length
                                       )

            self.train_iter = data.BucketIterator(train,
                                                  batch_size = batch_size,
                                                  device = 'cuda:%d' % device if
                                                  shuffle = shuffle,
```

```
                                                    sort_key=lambda x: len(x.tgt)
                                                    sort_within_batch = True
                                                    )
                self.valid_iter = data.BucketIterator(valid,
                                                    batch_size = batch_size,
                                                    device = 'cuda:%d' % device if
                                                    shuffle = False,
                                                    sort_key=lambda x: len(x.tgt)
                                                    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
            continue
        if src_line != '' and trg_line != '':
            examples.append(data.Example.fromlist(
                [src_line, trg_line], fields))

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