NATURAL LANGUAGE PROCESSING

LECTURE 7: Preprocessing

INDEX

Tokenization Pipeline

- Normalization

  : a set of operations you apply to a raw string to make it less random or "cleaner".

- Pre-Tokenization

  : Pre-tokenization is the act of splitting a text into smaller objects that give an upper bound to what your tokens will be at the end of training. A good way to think of this is that the pre-tokenizer will split your text into "words" and then, your final tokens will be parts of those words.

- The Model

  : Once the input texts are normalized and pre-tokenized, the Tokenizer applies the model on the pre-tokens. This is the part of the pipeline that needs training on your corpus (or that has been trained if you are using a pretrained tokenizer). BPE, Unigram, .. etc

- Post-Processing

https://huggingface.co/docs/tokenizers/python/latest/pipeline.html#

Normalization

- Cleaning : remove noise data from the corpus
- Normalization : normalize representations of words (e.g., cased / uncased)

- Stemming, Lemmatization
- Cased / Uncased
- Removing unnecessary words
  - Rare words
  - Words with a very short length
- Regular Expression

Regular Expression

- Regular expression can be used if the noise data has a certain pattern
  - Regular Expression : a sequence of characters that specifies a *search pattern*
  - In python, we can use re package or NLTK.

```
import re
text="Regular expression : A regular expression, regex or regexp[1] (sometimes called a rational expression)[2][3] i
s, in theoretical computer science and formal language theory, a sequence of characters that define a search patter
n."
re.sub('[^a-zA-Z]',' ',text)
```

```
'Regular expression   A regular expression  regex or regexp     sometimes called a rational expression        is  in
theoretical computer science and formal language theory  a sequence of characters that define a search pattern '
```

Regular Expression

특수 문자	설명
.	한 개의 임의의 문자를 나타냅니다. (줄바꿈 문자인 \n는 제외)
?	앞의 문자가 존재할 수도 있고, 존재하지 않을 수도 있습니다. (문자가 0개 또는 1개)
*	앞의 문자가 무한개로 존재할 수도 있고, 존재하지 않을 수도 있습니다. (문자가 0개 이상)
+	앞의 문자가 최소 한 개 이상 존재합니다. (문자가 1개 이상)
^	뒤의 문자로 문자열이 시작됩니다.
$	앞의 문자로 문자열이 끝납니다.
{숫자}	숫자만큼 반복합니다.
{숫자1, 숫자2}	숫자1 이상 숫자2 이하만큼 반복합니다. ?, *, +를 이것으로 대체할 수 있습니다.
{숫자,}	숫자 이상만큼 반복합니다.
[ ]	대괄호 안의 문자들 중 한 개의 문자와 매치합니다. [amk]라고 한다면 a 또는 m 또는 k 중 하나라도 존재하면 매치를 의미합니다. [a-z]와 같이 범위를 지정할 수도 있습니다. [a-zA-Z]는 알파벳 전체를 의미하는 범위이며, 문자열에 알파벳이 존재하면 매치를 의미합니다.
[^문자]	해당 문자를 제외한 문자를 매치합니다.
|	A|B와 같이 쓰이며 A 또는 B의 의미를 가집니다.

Regular Expression

문자 규칙	설명
\	역 슬래쉬 문자 자체를 의미합니다
\d	모든 숫자를 의미합니다. [0-9]와 의미가 동일합니다.
\D	숫자를 제외한 모든 문자를 의미합니다. [^0-9]와 의미가 동일합니다.
\s	공백을 의미합니다. [\t\n\r\f\v]와 의미가 동일합니다.
\S	공백을 제외한 문자를 의미합니다. [^ \t\n\r\f\v]와 의미가 동일합니다.
\w	문자 또는 숫자를 의미합니다. [a-zA-Z0-9]와 의미가 동일합니다.
\W	문자 또는 숫자가 아닌 문자를 의미합니다. [^a-zA-Z0-9]와 의미가 동일합니다.

Regular Expression

- Functions in re module

모듈 함수	설명
re.compile()	정규표현식을 컴파일하는 함수입니다. 다시 말해, 파이썬에게 전해주는 역할을 합니다. 찾고자 하는 패턴이 빈번한 경우에는 미리 컴파일해놓고 사용하면 속도와 편의성면에서 유리합니다.
re.search()	문자열 전체에 대해서 정규표현식과 매치되는지를 검색합니다.
re.match()	문자열의 처음이 정규표현식과 매치되는지를 검색합니다.
re.split()	정규 표현식을 기준으로 문자열을 분리하여 리스트로 리턴합니다.
re.findall()	문자열에서 정규 표현식과 매치되는 모든 경우의 문자열을 찾아서 리스트로 리턴합니다. 만약, 매치되는 문자열이 없다면 빈 리스트가 리턴됩니다.
re.finditer()	문자열에서 정규 표현식과 매치되는 모든 경우의 문자열에 대한 이터레이터 객체를 리턴합니다.
re.sub()	문자열에서 정규 표현식과 일치하는 부분에 대해서 다른 문자열로 대체합니다.

- RegexpTokenizer in NLTK

```
import nltk
from nltk.tokenize import RegexpTokenizer
tokenizer=RegexpTokenizer("[\w]+")
print(tokenizer.tokenize("Don't be fooled by the dark sounding name, Mr. Jone's Orphanage is as cheery as cheery goes for a pastry shop"))
```

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

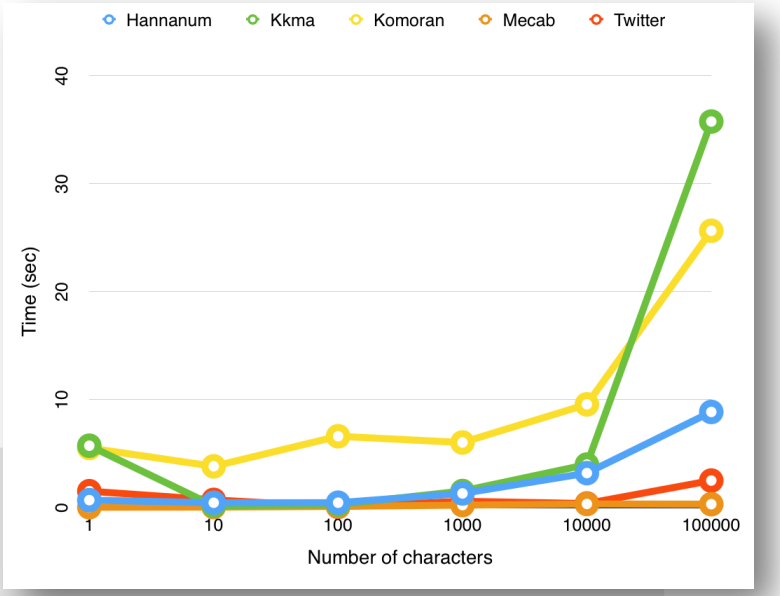
Pre-tokenization

- Basic approach: (1) remove special characters (, . ! ?), (2) split by whitespace
- What if special characters are semantically important?
  - e. g., apostrophe - aren't, don't, ..

- Korean text data requires much more considerations
  - Rather than splitting by whitespace, the morpheme(형태소) has to be carefully considered.
  - Moreover, spacing in Korean text data should be corrected in most cases.
  - Korean text preprocessing tools:
    - Supervised learning based: KoNLPy, Khaiii
    - Unsupervised learning based: soynlp

Pre-tokenization

- KoNLPy(https://konlpy.org/en/latest/)
  - Korean NLP in python - open source morpheme analyzers (tokenizer)
  - Hannanum, Kkma, Komoran, Mecab, Okt Classes



- tokenizer.morphs(text) : Parse phrase to morphemes
- tokenizer.nouns(text) : Noun extractors
- tokenizer.pos(text) : POS tagger

text = '환영합니다! 자연어 처리 수업은 재미있게 듣고 계신가요?'

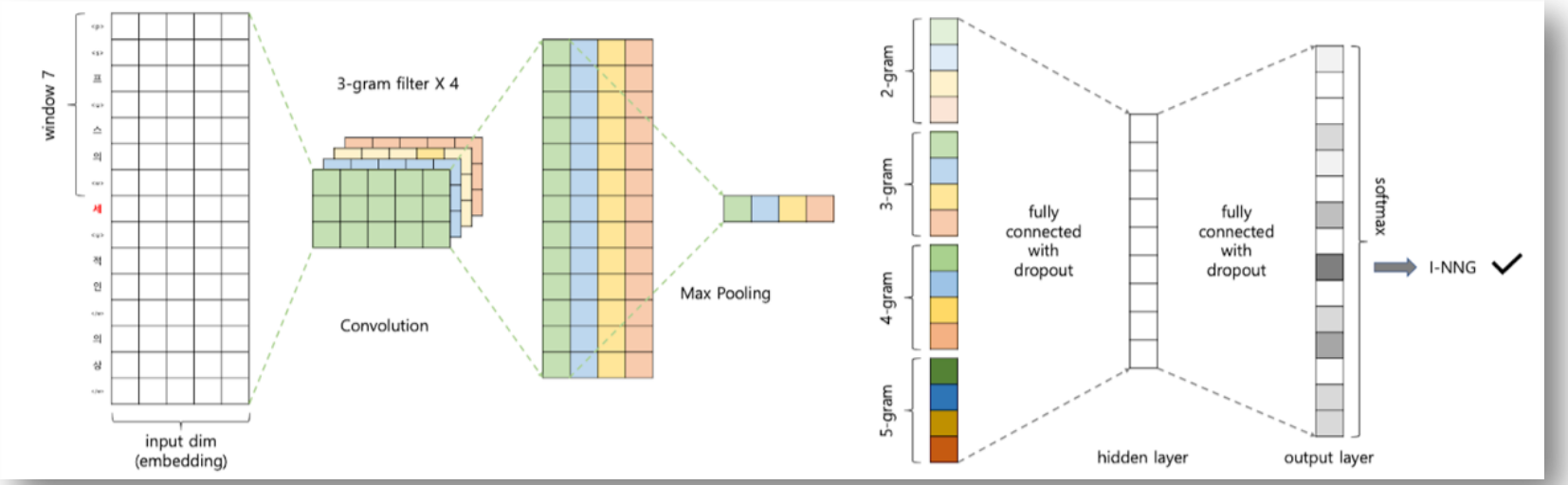
['환영', '하', 'ㅂ니다', '!', '자연어', '처리', '수업', '은', '재미있', '게', '듣', '고', '계시', 'ㄴ가요', '?']

['환영', '자연어', '처리', '수업']

[('환영', 'NNG'), ('하', 'XSV'), ('ㅂ니다', 'EFN'), ('!', 'SF'), ('자연어', 'NNG'), ('처리', 'NNG'), ('수업', 'NNG'), ('은', 'JX'), ('재미있', 'VA'), ('게', 'ECD'), ('듣', 'VV'), ('고', 'ECE'), ('계시', 'VXA'), ('ㄴ가요', 'EFQ'), ('?', 'SF')]

Pre-tokenization

- Khaiii(https://tech.kakao.com/2018/12/13/khaiii/)
  - Kakao Hangul Analyzer III
  - Trained data: 세종코퍼스
  - Model: CNN



Pre-tokenization

- PyKoSpacing

  (https://github.com/haven-jeon/PyKoSpacing)

  - 띄어쓰기가 되어 있지 않은 문장을 띄어쓰기가 되어 있는 문장으로 바꿔줌.

```
sent = '구름 자연어 처리 전문가 양성 과정 1기에 오신 여러분을 환영합니다!'
```

```
new_sent = sent.replace(" ", '') # 띄어쓰기가 없는 문장 임의로 만들기
print(new_sent)
```

```
구름자연어처리전문가양성과정1기에오신여러분을환영합니다!
```

```
from pykospacing import Spacing
spacing = Spacing()
kospacing_sent = spacing(new_sent)

print('띄어쓰기가 없는 문장 :\n', new_sent)
print('정답 문장:\n', sent)
print('띄어쓰기 교정 후:\n', kospacing_sent)
```

```
띄어쓰기가 없는 문장 :
 구름자연어처리전문가양성과정1기에오신여러분을환영합니다!
정답 문장:
 구름 자연어 처리 전문가 양성 과정 1기에 오신 여러분을 환영합니다!
띄어쓰기 교정 후:
 구름자연어 처리 전문가 양성과정 1기에 오신 여러분을 환영합니다!
```

Pre-tokenization

- Py-Hanspell

  (https://github.com/ssut/py-hanspell)

  - 네이버 한글 맞춤법 검사기를 바탕으로 만들어진 패키지

```
from hanspell import spell_checker

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

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

```
맞춤법 틀리면 왜 안돼? 쓰고 싶은 대로 쓰면 되지
```

Pre-tokenization

- soynlp (https://github.com/lovit/soynlp)
  - Tokenizer is trained from given data pattern by unsupervised learning.
  - Word score using data statistics: if the score is large, soynlp consider the word as a morpheme.
    - Cohesion probability – 주어진 문자열이 함께 등장할 경우, 응집 확률이 높음.
    - Branching entropy – 단어 앞뒤로 다양한 다른 단어가 등장할 경우, 브랜칭 엔트로피가 높음.