NLP Libraries

2023 전산언어학 겨울학교 1일차 3교시

이규민
<his.nigel at gmail dot com>

Key Concepts

- `.py` 스크립트
- 멀티파일 프로그래밍
- 라이브러리
- `pip`
- `import`, `from`, `as`

Key Concepts [Cont.]

- `NLTK`: tokenization, lemmatizion, stemming, frequency, concordance
- `KoNLPy`: 한국어의 처리
- `spaCy`
- 컴퓨터는 언어를 구분해서 처리할까?

`.py` 스크립트

- Python으로 작성한 파일
- 독립적으로 특정 기능을 실행하거나,
- 다른 스크립트 파일 혹은 노트북 파일 등에서 불러와 사용할 수 있도록 함수 등을 정리함
- 노트북 파일(`.ipynb`)과 달리, 불러올 때 마다 처음부터 끝까지 차례대로 모두 실행됨

```
    else:
        parsed = parse_highlights(path_load(hwpml_paths, 'hml'))

    counter = 0

    for doc in tqdm(master['document'], desc="Iterating through the json..."):
        if doc['id'].lower() in parsed:
            doc['metadata']['highlighted_cells'] = parsed[doc['id'].lower()]
            counter += 1

    print(f"Changed {counter} files.")

    output_name = ''.join([os.path.split(original_path)[0],
        '/', 'updated_', os.path.split(original_path)[1]
        ])

    with open(output_name, 'w') as f:
        json.dump(master, f, ensure_ascii=False, indent=4)

    print(f"Updated file saved as {output_name}")

    return

def main():
    paths = "./hwpml_edited"
    original_path = './corpus_table_22edit.json'
```

멀티파일 프로그래밍

- 프로그램을 구성하는 파일 일부를 `.py` 스크립트로 별도로 정리한 뒤,
- 스크립트나 노트북 파일 등에서 스크립트등에 정리된 요소를 불러와 기능을 수행하는 것
- Why?
  - 프로그램이 너무 길어져서 더 이상 다루지 못하는 상황을 방지
  - 프로그램의 기능을 세분화하여 단계적으로 개발
  - 여러 프로그램에 사용될 수 있는 기능을 스크립트에 담아 이를 다른 프로젝트에 재활용

멀티파일 프로그래밍 [Cont.]

- `import`: 특정 스크립트의 내용 전체를 불러오기
- `from xxx import yyy`: `xxx` 스크립트에서 `yyy`만 불러오기
- `as`: `import`로 불러온 내용을 부르는 별명을 설정


4:14 PM Mon Mar 14
File Edit Selection View Go Run Terminal Help
run_nli.py — [ll2021]TransBERT — code-server
EXPLORER
[LL2021]TRANSBERT
bin
build
dist
examples
acl.sh
extract_features.py
gpu.sh
run_classifier.py
run_copa.py
run_imdb.py
run_lm_finetuning.py
run_mnli.py
run_nli.py
run_rocstories_1.5.py
run_rocstories.py
run_squad.py
run_swag.py
run_twitter.py
notebooks
pytorch_pretrained_bert
pytorch_pretrained_bert.egg-info
ROCStories
samples
tests
.gitattributes
README.md
requirements.txt
setup.py
examples > run_nli.py > tqdm
self.input_mask = input_mask
self.segment_ids = segment_ids
self.label_id = label_id
class DataProcessor(object):
"""Base class for data converters for sequence classification data sets."""
def get_train_examples(self, data_dir):
"""Gets a collection of `InputExample`s for the train set."""
raise NotImplementedError()
def get_dev_examples(self, data_dir):
"""Gets a collection of `InputExample`s for the dev set."""
raise NotImplementedError()
def get_labels(self):
"""Gets the list of labels for this data set."""
raise NotImplementedError()
@classmethod
def _read_tsv(cls, input_file, quotechar=None):
"""Reads a tab separated value file."""
with open(input_file, "r", encoding='utf-8') as f:
reader = csv.reader(f, delimiter="\t", quotechar=quotechar)
lines = []
for line in reader:
lines.append(line)
return lines
class MrpcProcessor(DataProcessor):
"""Processor for the MRPC data set (GLUE version)."""
def get_train_examples(self, data_dir):
"""See base class."""
logger.info("LOOKING AT {}".format(os.path.join(data_dir, "train.tsv")))
return self._create_examples(
self._read_tsv(os.path.join(data_dir, "train.tsv")), "train")
def get_dev_examples(self, data_dir):
"""See base class."""
return self._create_examples(
self._read_tsv(os.path.join(data_dir, "dev.tsv")), "dev")
def get_labels(self):
"""See base class."""
return ["0", "1"]
OUTLINE
TIMELINE
121.159.63.164:8080
master
Python 3.9.7 64-bit ('base': conda)
Ln 27, Col 30
Spaces: 4
UTF-8
LF
Python

라이브러리

- *library*, a.k.a. *package*, *module*, ...
- 다른 프로그램에서 불러와 사용될 목적으로 정리된 스크립트의 묶음
- 반복되는 작업이나 복잡한 작업 등을 구체적인 코드를 보지 않은 채로 사용할 수 있음
- e.g., `random` 라이브러리
  - 임의의 수를 생성하는 것은 여러 수리통계적 과정과 컴퓨터 작동 원리를 참고해야 하는 작업
  - Python의 기본 라이브러리인 `ramdom`을 통해 임의의 수를 간단히 생성할 수 있음

PyPI

- 라이브러리는:
  - Python에서 기본으로 제공하는 '시스템 라이브러리'와,
  - 시스템 라이브러리가 아니지만 사용자의 컴퓨터에 저장되어 언제든 불러올 수 있는 라이브러리로 구성됨
- PyPI(Python Package Index)
  - Python 사용자들이 자발적으로 업로드하는 공개 라이브러리를 관리하는 제단 공식 서비스
  - 작성 일자 기준 7,504,291개의 라이브러리 파일이 공유되고 있는 중

`pip`

- `pip` (**P**reffered **I**nstallation **P**rogram)
  - PyPI를 비롯한 라이브러리 저장소에서 라이브러리 파일을 다운받아,
  - 사용자의 환경에 맞게 설치하고
  - 이에 대한 업데이트 및 삭제 등 라이브러리 관리를 위한 프로그램


4:14 PM Mon Mar 14
File Edit Selection View Go Run Terminal Help
run_nli.py — [ll2021]TransBERT — code-server
EXPLORER
[LL2021]TRANSBERT
bin
build
dist
examples
acl.sh
extract_features.py
gpu.sh
run_classifier.py
run_copa.py
run_imdb.py
run_lm_finetuning.py
run_mnli.py
run_nli.py
run_rocstories_1.5.py
run_rocstories.py
run_squad.py
run_swag.py
run_twitter.py
notebooks
pytorch_pretrained_bert
pytorch_pretrained_bert.egg-info
ROCStories
samples
tests
.gitattributes
README.md
requirements.txt
setup.py
examples > run_nli.py > tqdm
self.input_mask = input_mask
self.segment_ids = segment_ids
self.label_id = label_id
class DataProcessor(object):
"""Base class for data converters for sequence classification data sets."""
def get_train_examples(self, data_dir):
"""Gets a collection of `InputExample`s for the train set."""
raise NotImplementedError()
def get_dev_examples(self, data_dir):
"""Gets a collection of `InputExample`s for the dev set."""
raise NotImplementedError()
def get_labels(self):
"""Gets the list of labels for this data set."""
raise NotImplementedError()
@classmethod
def _read_tsv(cls, input_file, quotechar=None):
"""Reads a tab separated value file."""
with open(input_file, "r", encoding='utf-8') as f:
reader = csv.reader(f, delimiter="\t", quotechar=quotechar)
lines = []
for line in reader:
lines.append(line)
return lines
class MrpcProcessor(DataProcessor):
"""Processor for the MRPC data set (GLUE version)."""
def get_train_examples(self, data_dir):
"""See base class."""
logger.info("LOOKING AT {}".format(os.path.join(data_dir, "train.tsv")))
return self._create_examples(
self._read_tsv(os.path.join(data_dir, "train.tsv")), "train")
def get_dev_examples(self, data_dir):
"""See base class."""
return self._create_examples(
self._read_tsv(os.path.join(data_dir, "dev.tsv")), "dev")
def get_labels(self):
"""See base class."""
return ["0", "1"]
OUTLINE
TIMELINE
121.159.63.164:8080
master
Python 3.9.7 64-bit ('base': conda)
Ln 27, Col 30
Spaces: 4
UTF-8
LF
Python

자연어처리의 기본 기술들

NLTK

- **N**atural **L**anguage **T**ool**k**it
  - Bird, Klein, & Loper(2009)
  - Python을 통해 자연어처리를 하는 방법을 정리한 라이브러리
  - 기본 Python 기능을 이용한 자연어처리 이외에도, 기계학습 모델 등을 활용한 처리 및 훈련 지원
  - 전통적인 자연어의 전산 처리 방법을 알아보고,
  - 공개된 코드를 통해 이러한 처리를 진행하는 방법을 공부하는데 좋은 역할을 함
  - 기능에 따라 지원하는 언어가 다르며, 기본적으로 영어 처리를 기본으로 상정하고 개발됨

자연어처리의 기본 기술들

tokenization

- **토큰**
  - 자연어 처리 분석의 대상이 되는 단위
  - 영어의 경우, `it's`에서 `'s`와 같은 clitic을 떨어뜨린 '단어'를 주로 토큰으로 삼음
  - 한국어의 경우 띄어쓰기가 단어 단어와 정확히 일치하지 않음
    - 가령, `그래도 해는 해라서 동쪽에서 뜬다`라는 문장에서 `해`라는 단어는 각각 `-는`과 `-라서`와 결함하여 띄어쓰기 단위를 형성함
    - 띄어쓰기 단위는 기계적으로 구분하기 용이하므로 텍스트의 크기 등을 논할 때 이러한 단위의 개수를 주로 사용함(**어절**)
    - 의존형태소의 분리는 기계학습을 통한 처리로 분리할 수 있지만, 항상 정확한 분리는 어려움

자연어처리의 기본 기술들

tokenization

- **토큰화**tokenization
  - 문장을 토큰으로 분리하는 기법
  - 영어의 경우, 정규식을 기반으로 한 규칙을 통한 분리 기법인 Moses 방법
  - Punkt는 기계학습을 통해 텍스트 특성에 맞는 약어를 떨어뜨리지 않도록 학습하게 한 방법
  - `NLTK`에서는 Moses와 더불어, 일반적인 영어 텍스트에 훈련된 Punkt 모델 또한 제공
- `nltk.tokenize.word_tokenize`
- cf) `nltk.tokenize.sent_tokenize`

자연어처리의 기본 기술들

lemmatization & stemming

- 목적 : 활용된 단어를 원형으로 복구
  - e.g., `love`, `loves`, `loved` 등을 모두 `love`로 복구하여 이러한 활용형 모두가 동일하게 집계되도록 할 경우 사용
- **Lemmatization**은 단어의 원형 정보가 저장된 전자사전을 사용하여 복구
  - `go → went`와 같은 변형에도 적용 가능
  - 사전에 없는 단어는 규칙적인 affixation을 거쳤더라도 복구할 수 없음
- **Stemming**은 해당 언어의 형태론적 규칙을 적용하여 복구
  - 영어의 경우, affixation을 주로 되돌리는 방식으로 진행됨
  - `go → went`와 같은 변형은 다루지 못함
  - `studied`와 같이 철자상의 변형이 진행된 경우, 원형이 `study`인지 `studi`인지 정보가 없기 때문에 `-ed`를 제거한 `studi` 형태로만 변형
  - 사전에 등재되지 않은 신조어 및 오타 등에도 작동
- `nltk.stem.WordNetLemmatizer` 및 `nltk.stem.PorterStemmer`

자연어처리의 기본 기술들

frequency

- 토큰의 **빈도**는 특정 토큰의 출현 회수를 지칭함
- 토큰의 **절대 빈도**absolute frequency는 빈도를 그대로 측정한 결과
  - `len()`, `.count()`, `nltk.FreqDist`
  - 동일한 문서 안에서는 절대 빈도를 통한 논의가 가능하지만, 문서의 전체 크기가 다른 경우 절대 빈도를 통한 논의는 할 수 없음
- 토큰의 **상대 빈도**relative frequency는 문서 크기에 대한 절대 빈도의 비율
  - 주로 문서 크기를 백만 토큰이라 상정하고 계산하는 Per-million Frequency가 자주 보고됨

자연어처리의 기본 기술들

concordance

- **용례**는 문서에 등장하는 특정 토큰을 맥락과 함께 제시한 것
  - 용례에 제시된 사례 하나하나를 **K**ey**w**ord **i**n **C**ontext(**KWiC**)이라고 지칭하기도 함
  - 주로 토큰의 등장 사례를 기반으로 특정 토큰의 사용 양상을 정성적으로 분석하거나,
  - 토큰이 출연한 맥락에 특정한 분석을 진행하기 위해 사용
- `nltk.Text.concordance()` 및 `nltk.Text.concordance_list()`


4:14 PM Mon Mar 14
File Edit Selection View Go Run Terminal Help
run_nli.py — [ll2021]TransBERT — code-server
EXPLORER
[LL2021]TRANSBERT
bin
build
dist
examples
acl.sh
extract_features.py
gpu.sh
run_classifier.py
run_copa.py
run_imdb.py
run_lm_finetuning.py
run_mnli.py
run_nli.py
run_rocstories_1.5.py
run_rocstories.py
run_squad.py
run_swag.py
run_twitter.py
notebooks
pytorch_pretrained_bert
pytorch_pretrained_bert.egg-info
ROCStories
samples
tests
.gitattributes
README.md
requirements.txt
setup.py
examples > run_nli.py > tqdm
self.input_mask = input_mask
self.segment_ids = segment_ids
self.label_id = label_id
class DataProcessor(object):
"""Base class for data converters for sequence classification data sets."""
def get_train_examples(self, data_dir):
"""Gets a collection of `InputExample`s for the train set."""
raise NotImplementedError()
def get_dev_examples(self, data_dir):
"""Gets a collection of `InputExample`s for the dev set."""
raise NotImplementedError()
def get_labels(self):
"""Gets the list of labels for this data set."""
raise NotImplementedError()
@classmethod
def _read_tsv(cls, input_file, quotechar=None):
"""Reads a tab separated value file."""
with open(input_file, "r", encoding='utf-8') as f:
reader = csv.reader(f, delimiter="\t", quotechar=quotechar)
lines = []
for line in reader:
lines.append(line)
return lines
class MrpcProcessor(DataProcessor):
"""Processor for the MRPC data set (GLUE version)."""
def get_train_examples(self, data_dir):
"""See base class."""
logger.info("LOOKING AT {}".format(os.path.join(data_dir, "train.tsv")))
return self._create_examples(
self._read_tsv(os.path.join(data_dir, "train.tsv")), "train")
def get_dev_examples(self, data_dir):
"""See base class."""
return self._create_examples(
self._read_tsv(os.path.join(data_dir, "dev.tsv")), "dev")
def get_labels(self):
"""See base class."""
return ["0", "1"]
OUTLINE
TIMELINE
121.159.63.164:8080
master
Python 3.9.7 64-bit ('base': conda)
Ln 27, Col 30
Spaces: 4
UTF-8
LF
Python

한국어의 처리

- **형태소 분석**Part-of-Speech Tagging은 텍스트를 형태소 단위로 토큰화한 뒤, 각 토큰의 품사를 기계학습을 기반으로 추측하는 과정
  - cf) `nltk.pos_tag()`
- 한국어의 경우,
  - 띄어쓰기를 기준으로 토큰화할 경우 의존형태소만 다른 어절이 완전히 다른 객체로 처리됨
  - Moses 토큰화 방식과 같이 규칙으로 처리하는 경우, 동일한 형태임에도 분절 양상이 다른 경우가 있어 어려움
    - e.g., `여기서 밥 먹고 '가자'` 와 `'가자' 지구에서 온 소식입니다.`
  - 이에, 기계학습을 통해 확률통계적으로 형태소 분석을 진행하는 형태소 분석기를 사용하는 것이 일반적

KoNLPy

- KoNLPy
  - 세종형태분석말뭉치 등 형태소 분석이 진행된 데이터들이 공개됨에 따라 다양한 형태소 분석기가 개발되었으나, 사용 환경이나 방식 등이 다른 경우가 많아 사용이 어려움
  - KoNLPy는 이러한 형태소 분석기 중 공개된 것들을 정리하여, 통일된 Python 문법으로 사용할 수 있게 정리한 라이브러리
  - 현재 한나눔, 꼬꼬마, 코모란, Mecab-ko, Okt의 다섯개 분석기를 지원함
    - Mecab-ko의 경우는 별도 설치가 필요하며, 윈도에서 실행할 수 없음
  - 분석기에 따라 처리 방법과 학습 데이터가 다르므로, 용도에 따라 적절한 분석기를 선택하여 사용하여야 함
    - 가령, Mecab-ko의 경우 처리 속도가 빠르다는 장점이 있으나 프로그램 관리가 어려우며, Okt는 트위터 분석을 목적으로 개발되어 해시태그나 '@'으로 구분되는 사용자 이름을 포함한 웹상 텍스트 분석이 용이한 등 차이가 있음
    - 분석기별로 사용하는 형태소 태그는 이곳에 정리됨

이외의 형태소 분석기

- `Kiwi`
  - 기계학습을 활용하여 모호성 해소 등의 작업에 비교적 높은 성능을 내며,
  - Mecab-ko에 비견할 수 있는 속도를 내는 형태소 분석기
  - 다른 분석기에 비해 지금도 활발히 개발 중이며, 오느날의 컴퓨팅 맥락에 잘 맞음
  - 윈도 환경에서 작동하는 그래픽 프로그램 또한 제공함
  - Python에서의 사용은 공식 Python 패키지를 통해 가능
- `khaiii`
  - 다음카카오에서 딥러닝을 기반으로 개발한 형태소분석기
  - 사용자 예외 추가를 통한 오류 수정이 비교적 용이함
  - 오타와 띄어쓰기 오류와 같은 비정규적 입력에서도 비교적 잘 작동함
  - 딥러닝을 사용하였다고 다른 방식에 비해 성능이 월등히 높지는 않음


4:14 PM Mon Mar 14
File Edit Selection View Go Run Terminal Help
run_nli.py — [ll2021]TransBERT — code-server
EXPLORER
[LL2021]TRANSBERT
bin
build
dist
examples
acl.sh
extract_features.py
gpu.sh
run_classifier.py
run_copa.py
run_imdb.py
run_lm_finetuning.py
run_mnli.py
run_nli.py
run_rocstories_1.5.py
run_rocstories.py
run_squad.py
run_swag.py
run_twitter.py
notebooks
pytorch_pretrained_bert
pytorch_pretrained_bert.egg-info
ROCStories
samples
tests
.gitattributes
README.md
requirements.txt
setup.py
examples > run_nli.py > tqdm
self.input_mask = input_mask
self.segment_ids = segment_ids
self.label_id = label_id
class DataProcessor(object):
"""Base class for data converters for sequence classification data sets."""
def get_train_examples(self, data_dir):
"""Gets a collection of `InputExample`s for the train set."""
raise NotImplementedError()
def get_dev_examples(self, data_dir):
"""Gets a collection of `InputExample`s for the dev set."""
raise NotImplementedError()
def get_labels(self):
"""Gets the list of labels for this data set."""
raise NotImplementedError()
@classmethod
def _read_tsv(cls, input_file, quotechar=None):
"""Reads a tab separated value file."""
with open(input_file, "r", encoding='utf-8') as f:
reader = csv.reader(f, delimiter="\t", quotechar=quotechar)
lines = []
for line in reader:
lines.append(line)
return lines
class MrpcProcessor(DataProcessor):
"""Processor for the MRPC data set (GLUE version)."""
def get_train_examples(self, data_dir):
"""See base class."""
logger.info("LOOKING AT {}".format(os.path.join(data_dir, "train.tsv")))
return self._create_examples(
self._read_tsv(os.path.join(data_dir, "train.tsv")), "train")
def get_dev_examples(self, data_dir):
"""See base class."""
return self._create_examples(
self._read_tsv(os.path.join(data_dir, "dev.tsv")), "dev")
def get_labels(self):
"""See base class."""
return ["0", "1"]
OUTLINE
TIMELINE
121.159.63.164:8080
master
Python 3.9.7 64-bit ('base': conda)
Ln 27, Col 30
Spaces: 4
UTF-8
LF
Python

spaCy

- 실질적으로 자연어처리 기술을 통해 서비스를 제공하는 것을 돕기 위한 라이브러리
- 분석하고자 하는 언어와 '효율' 혹은 '성능' 중 하나를 선택하면, 딥러닝을 활용한 모델을 포함하여 가장 최적화된 방식으로 별도로 제외하지 않은 모든 분석을 진행
- 분석이 진행된 결과에 대해 필요한 정보를 추출하여 사용

컴퓨터는 언어를 구분해서 처리할까?

- 컴퓨터에게 스트링의 형식으로 입력되는 문자는 언어에 구속되지 않은 임의의 바이트 기호 체계
- 가령, `us`와 `우리`는 사람에게는 당연스럽게도 특정 언어로 해석되지만,
- Python을 기준으로 컴퓨터에게는 그저 `0x0075 0x0073`이라는 바이트의 나열과 `0xC6B0 0xB9AC`라는 바이트의 나열에 불과함
- 따라서, 분석하는 언어가 무엇이든 스트링의 나열을 입력으로 받아 처리하는 방법론은 스트링의 나열이기만 하면 처리할 수 있음
- e.g.,
  - 한국어 텍스트의 concordance가 필요한 경우,
  - `KoNLPy`를 통해 형태소 분석을 진행한 뒤,
  - 이를 목적에 맞게 토큰의 리스트로 정리
    - 형태가 같지만 품사가 다른 형태소를 구분하고자 한다면 리스트의 항목 하나가 형태와 품사를 모두 보여주는 스트링으로 처리
    - 구분할 필요가 없으면 형태만 처리
  - 리스트를 `NLTK`를 통해 concordance로 처리


4:14 PM Mon Mar 14
File Edit Selection View Go Run Terminal Help
run_nli.py — [ll2021]TransBERT — code-server
EXPLORER
[LL2021]TRANSBERT
bin
build
dist
examples
acl.sh
extract_features.py
gpu.sh
run_classifier.py
run_copa.py
run_imdb.py
run_lm_finetuning.py
run_mnli.py
run_nli.py
run_rocstories_1.5.py
run_rocstories.py
run_squad.py
run_swag.py
run_twitter.py
notebooks
pytorch_pretrained_bert
pytorch_pretrained_bert.egg-info
ROCStories
samples
tests
.gitattributes
README.md
requirements.txt
setup.py
examples > run_nli.py > tqdm
self.input_mask = input_mask
self.segment_ids = segment_ids
self.label_id = label_id
class DataProcessor(object):
"""Base class for data converters for sequence classification data sets."""
def get_train_examples(self, data_dir):
"""Gets a collection of `InputExample`s for the train set."""
raise NotImplementedError()
def get_dev_examples(self, data_dir):
"""Gets a collection of `InputExample`s for the dev set."""
raise NotImplementedError()
def get_labels(self):
"""Gets the list of labels for this data set."""
raise NotImplementedError()
@classmethod
def _read_tsv(cls, input_file, quotechar=None):
"""Reads a tab separated value file."""
with open(input_file, "r", encoding='utf-8') as f:
reader = csv.reader(f, delimiter="\t", quotechar=quotechar)
lines = []
for line in reader:
lines.append(line)
return lines
class MrpcProcessor(DataProcessor):
"""Processor for the MRPC data set (GLUE version)."""
def get_train_examples(self, data_dir):
"""See base class."""
logger.info("LOOKING AT {}".format(os.path.join(data_dir, "train.tsv")))
return self._create_examples(
self._read_tsv(os.path.join(data_dir, "train.tsv")), "train")
def get_dev_examples(self, data_dir):
"""See base class."""
return self._create_examples(
self._read_tsv(os.path.join(data_dir, "dev.tsv")), "dev")
def get_labels(self):
"""See base class."""
return ["0", "1"]
OUTLINE
TIMELINE
121.159.63.164:8080
master
Python 3.9.7 64-bit ('base': conda)
Ln 27, Col 30
Spaces: 4
UTF-8
LF
Python

Key Concepts

1. 효율적인 프로그래밍을 위해 함수 등의 기능을 별도의 스크립트로 작성한 뒤, 이를 필요에 따라 `import` 등으로 불러와 사용한다.
2. 이러한 별도의 스크립트를 정리한 라이브러리를 사용하여 복잡한 작업을 수월히 진행할 수 있다.
3. 라이브러리는 `pip`로 설치 및 관리한다
4. `NLTK`는 Python을 통한 자연어처리 기법을 정리한 라이브러리로, tokenization, lemmatizaiton, stemming, frequency table, concordance 등을 지원한다.
5. `KoNLPy`는 한국어 형태소분석 도구들을 정리한 라이브러리이며, 이 외에도 Kiwi, khaiii등의 형태소 분석기를 Python에서 사용할 수 있다.
6. `spaCy`는 다양한 언어에 대해 자연어 처리 기술을 한번에 진행하게 해 주는 라이브러리다.
7. 컴퓨터는 언어는 구분하지는 않기 때문에, 여러 함수가 요구하는 형식만 맞추어 준다면 실제 언어와 상관없이 사용할 수 있다.

앞으로 다를 라이브러리

- `gensim`은 텍스트 데이터를 다루는 기계학습 모델을 정리한 라이브러리
- `scikit-learn`(sklearn)은 기계학습에서 자주 사용되는 연산들을 정리한 라이브러리
- `PyTorch`와 `Tensorflow`는 GPU 가속을 활용하여 텐서 연산을 진행하여, 딥러닝을 수행할 수 있도록 도와주는 라이브러리
- HuggingFace의 `transformers`와 `datasets`는 `PyTorch`를 기반으로 각각 훈련된 딥러닝 모델과 훈련을 위한 데이터를 체계적으로 다룰 수 있도록 도와주는 라이브러리