NPL
WITH PYTORCH

PART I

TORCHTEXT

TORCHTEXT

◆ PyTorch 텍스트 라이브러리

- 텍스트에 대한 추상화 기능 제공하는 **자연어 처리 라이브러리**
- **데이터 처리 유틸리티와 인기 있는 자연어 데이터 세트로 구성**
- **2024년 4월 v0.18.0 마지막 안정적인 릴리스 버전 ➔ 개발 중단**

- 제공 기능
  - 파일 로드(File Loading) : 다양한 포맷 코퍼스 로드
  - 토큰화(Tokenization) : 문장을 단어 단위로 분리
  - 단어 집합(Vocab) : 단어 집합 생성
  - 정수 인코딩(Integer encoding) : 전체 코퍼스 단어들 고유한 정수 맵핑
  - 단어 벡터(Word Vector) : 단어 집합 단어들에 고유한 임베딩 벡터 생성
  - 배치화(Batching) : 훈련 샘플들 배치 생성 및 패딩 작업(Padding) 진행

TORCHTEXT

◆ 버전 및 설치

- https://pypi.org/project/torchtext/

PyTorch version	torchtext version	Supported Python version
nightly build	main	>=3.8, <=3.11
2.2.0	0.17.0	>=3.8, <=3.11
2.1.0	0.16.0	>=3.8, <=3.11
2.0.0	0.15.0	>=3.8, <=3.11
1.13.0	0.14.0	>=3.7, <=3.10
1.12.0	0.13.0	>=3.7, <=3.10
1.11.0	0.12.0	>=3.6, <=3.9
1.10.0	0.11.0	>=3.6, <=3.9
1.9.1	0.10.1	>=3.6, <=3.9
1.9	0.10	>=3.6, <=3.9

TORCHTEXT

◆ 버전 및 설치

- **torchtext v0.11.0**

[가상환경]

```
conda create -n TEXT_011_110_38 python=3.8
conda env list
```

[파이토치]

```
conda install pytorch==1.10.0 torchvision==0.11.0 torchaudio==0.10.0 cpuonly -c pytorch
```

[토치텍스트]

```
conda install -c pytorch torchtext==0.11.0
```

TORCHTEXT

◆ 버전 및 설치

- **torchtext v0.15.0**

[가상환경]

```
conda create -n TEXT_015_200_38 python=3.8
conda env list
```

[파이토치]

```
conda install pytorch==2.0.0 torchvision==0.15.0 torchaudio==2.0.0 cpuonly -c pytorch
```

[토치텍스트]

```
conda install -c pytorch torchtext==0.15.0
```

TORCHTEXT

◆ 버전 및 설치

- **torchtext v0.18.0** ← 최종 마지막 버전

[가상환경]

conda create -n TEXT_018_230_38 python=3.8

[파이토치]

conda install pytorch torchvision torchaudio cpuonly -c pytorch

[토치텍스트- 의존성패키지]

conda install -c conda-forge portalocker>=2.0.0

[토치텍스트]

conda install -c pytorch torchtext torchdata

TORCHTEXT

◆ 서브 패키지

- torchtext.nn
- torchtext.data.functional
- torchtext.data.metrics
- torchtext.data.utils
- torchtext.datasets
- torchtext.vocab
- torchtext.utils
- torchtext.transforms
- torchtext.functional
- torchtext.models

torchtext.data.functional

- generate_sp_model
- load_sp_model
- sentencepiece_numericalizer
- sentencepiece_tokenizer
- custom_replace
- simple_space_split
- numericalize_tokens_from_iterator
- filter_wikipedia_xml
- to_map_style_dataset

torchtext.vocab

- Vocab
- vocab
- build_vocab_from_iterator
- Vectors
- GloVe
- FastText
- CharNGram

TORCHTEXT

◆ 서브 패키지

Docs > torchtext.datasets

Datasets

- Text Classification
  - AG_NEWS
  - AmazonReviewFull
  - AmazonReviewPolarity
  - CoLA
  - DBpedia
  - IMDb
  - MNLI
  - MRPC
  - QNLI
  - QQP
  - RTE
  - SogouNews
  - SST2
  - STSB
  - WNLI
  - YahooAnswers
  - YelpReviewFull
  - YelpReviewPolarity

Docs > torchtext.datasets

- Language Modeling
  - PennTreebank
  - WikiText-2
  - WikiText103
- Machine Translation
  - IWSLT2016
  - IWSLT2017
  - Multi30k
- Sequence Tagging
  - CoNLL2000Chunking
  - UDPOS
- Question Answer
  - SQuAD 1.0
  - SQuAD 2.0
- Unsupervised Learning
  - CC100
  - EnWik9

TORCHTEXT

◆ iterator 반복자

- **순차적으로 다음 데이터를 리턴할 수 있는 객체**
- **next() 함수 내장**해서 순환하는 다음 값 반환

- 생성 ➔ iter( ) 함수

```
a = [1, 2, 3]
iter_a = iter( a )
print( type(iter_a) )
```

TORCHTEXT

◆ iterator 반복자

```
class MyIter:
    def __init__(self, data):
        self.data = data
        self.posiion=0

    def __iter__(self): return self

    def __next__(self):
        if self.posiion >= len(self.data): raise StopIteration
        result = self.data[self.posiion]
        self.posiion += 1
        return result
```

- _ _next_ _ 메서드 존재
  iterator 객체

TORCHTEXT

◆ iterator 반복자

```
class ReverseIter:
    def __init__(self, data):
        self.data = data
        self.posiion= len(self.data) - 1

    def __iter__(self): return self

    def __next__(self):
        if self.posiion >= len(self.data): raise StopIteration
        result = self.data[self.posiion]
        self.posiion -= 1
        return result
```

TORCHTEXT

◆ iterator 반복자

```
class MyCounter:
    def __init__(self, data):
        self.data = data

    def __iter__(self):
        return MyIter(self.data)
```

- _ _iter_ _ 메서드 존재하는 iterable 객체
- Iterator 객체 반환함!

TORCHTEXT

◆ Generator 반복자

- **이터레이터를 생성해 주는 함수**
- 함수 내부에 **yield** 사용되며, yield로 호출한 곳에 값 전달
- 호출 시에 값을 메모리에 올림 ➔ **지연 평가(Lazy Evaluation) 방식/메모리 효율적**
- 내부적으로 _ _iter_ _(), _ _next_ _() 존재

- **키워드 yield**
  - **잠시 함수 실행 멈춤 ➔ 호출한 곳에 값 전달**
  - 현재 실행 항태 계속 유지/ 다시 함수 호출 시 현재 실행 상태 기반 코드 실행
  - return처럼 **값 반환 후 종료 되지 않음!**

TORCHTEXT

◆ Generator 반복자

```
def generator_func():
    for i in [11,22,33]:
        yield i
```

```
gen=generator_func()
print(f'get => {gen}')
```

get => <generator object generator_func at 0x0000021FB53F3F20>

```
def generator_func():
    a = [11,22,33]
    yield from a
```

```
for value in gen:
    print( value )
```

1 2 3

TORCHTEXT

◆ Generator 반복자

- Generator Expression 또는 Generator Comprehension

  * 형식 : (express for in)

```
square_gen = ( num ** 2 for num in range(5) )

print( type(square_gen) )

next(square_gen), next(square_gen), next(square_gen)
```

TORCHTEXT

◆ Generator 반복자

```
def generator_func():
    for i in [11,22,33]:
        yield i
```

```
gen=generator_func()
print(f'get => {gen}')
```

get => <generator object generator_func at 0x0000021FB53F3F20>

```
def generator_func():
    a = [11,22,33]
    yield from a
```

```
for value in gen:
    print( value )
```

1 2 3

TORCHTEXT

◆ torchtext.data / torchtext.legacy.data 필드 정의

- 피쳐별 전처리 진행 방법 지정

```
# 필드 정의
TEXT = data.Field(sequential=True,
                  use_vocab=True,
                  tokenize=str.split,
                  lower=True,
                  batch_first=True,
                  fix_length=20)

LABEL = data.Field(sequential=False,
                   use_vocab=False,
                   batch_first=False,
                   is_target=True)
```

sequential : 시퀀스 데이터 여부 (True 기본값)
use_vocab : 단어 집합 만들 것인지 여부 (True 기본값)
tokenize : 토큰화 함수 지정 (string.split 기본값)
lower : 영어 데이터 전부 소문자화 (False 기본값)
batch_first : 미니 배치 차원 맨 앞 (False 기본값)
is_target : 레이블 데이터 여부 (False 기본값)
fix_length : 패딩 최대 허용 길이

TORCHTEXT

◆ DataSet 생성 - TabularDataset

- 데이터 로딩하여 필드에서 정의했던 토큰화 방법으로 토큰화 수행

```
train_data, test_data = TabularDataset.splits(
        path='.', train='train_data.csv', test='test_data.csv', format='csv',
        fields=[('text', TEXT), ('label', LABEL)], skip_header=True)
```

- path : 파일이 위치한 경로.
- format : 데이터 포맷
- fields : 정의한 필드 지정 (데이터의 컬럼, 지정할 필드이름)
- skip_header : 데이터의 첫번째 줄 무시.

TORCHTEXT

◆ 단어 집합(Vocabulary) 생성

▪ Tokenizer 생성

torchtext.data.utils.**get_tokenizer(** tokenizer , language='en' **)**

➔ **반환 : tokenizer 인스턴스**

- tokenizer — tokenizer 함수 이름
  - None : split( )
  - basic_English : basic_english_normalize( )
  - tokenizer library : 라이브러리 관련 함수 반환
- language — 토큰화 언어 (기: en)

TORCHTEXT

◆ 단어 집합(Vocabulary) 생성

- **Voca 객체 : 토큰을 인덱스와 매핑, 토큰 ⇔ 정수인덱스 변환**

CLASS torchtext.vocab.**Vocab(vocab)**

• get_stoi()	: token을 정수인덱스로 반환
• get_itos()	: 정수인덱스를 token으로 반환
• __getitem__()	: token에 매핑되는 정수인덱스값 반환
• lookup_token()	: 정수인덱스에 매핑되는 token 반환
• forward()	: encode 진행(문장 → 토큰화 → id값 변경) , nn.Module의 forward()
lookup_indices()	: encode 진행(문장 → 토큰화 → id값 변경)
• lookup_tokens()	: decode 진행(id → 적합한 토큰)

TORCHTEXT

◆ 단어 집합(Vocabulary) 생성

▪ **Voca 객체 생성 : 토큰을 인덱스와 매핑**

torchtext.vocab.**vocab(** ordered_dict, min_freq=1, specials=None, special_first=True **)**

➔ **반환 : Vocab 인스턴스**

- Ordered_dict – 토큰을 해당 발생 빈도에 매핑하는 순서가 지정된 사전
- min_freq – 어휘에 토큰을 포함하는 데 필요한 최소 빈도
- Specials - 추가할 특수 기호. 공급된 토큰의 순서는 유지됨
- Special_first – 기호를 처음에 삽입할지 아니면 끝에 삽입할지 결정

TORCHTEXT

◆ 단어 집합(Vocabulary) 생성

- **Voca 객체 생성 : iterator이용하여 생성**

torchtext.vocab.**build_vocab_from_iterator(** iterator, min_freq = 1, specials = None, special_first = True, max_tokens = None **)** ➔ **반환 Vocab**

- iterator – Vocab 빌드 반복자, 토큰 목록또는 반복자 생성
- min_freq – Vocab에 토큰을 포함하기 위한 최소 빈도수
- specials – 추가할 특수 기호
- special_first – 추가 특수 기호 처음 또는 끝 삽입여부 결정
- max_tokens – max_tokens - len(specials)값으로 최대 토큰 수

TORCHTEXT

◆ 단어 집합(Vocabulary) 생성

- 각 단어에 고유한 정수를 맵핑해주는 정수 인코딩(Integer enoding) 작업 필요

```
TEXT.build_vocab(train_data, min_freq=10, max_size=10000)
```

- min_freq : 단어 집합에 추가 시 단어의 최소 등장 빈도 조건 추가.
- max_size : 단어 집합의 최대 크기 지정

**** 토치텍스트가 임의로 특별 토큰인 <unk>와 <pad> 추가**
<unk>번호는 0번, <pad>번호는 1번 부여

TORCHTEXT

◆ 데이터로더 생성

- 데이터셋에서 미니 배치만큼 데이터를 로드하게 만들어주는 역할
- 토치텍스트에서는 Iterator를 사용하여 데이터로더 생성

```
from torchtext.data import Iterator

batch_size = 5

train_loader = Iterator( dataset=train_data, batch_size = batch_size )
test_loader = Iterator( dataset=test_data, batch_size = batch_size )
```