

# Tokenizer 구현 보고서- 24기 이동렬

## 목차

[모듈 Introduction](#)

[parent-부모클래스](#)

[preprocessor - 전처리기](#)

[tokenizer - BPE](#)

[tokenizer - Word](#)

[Debugging Log](#)

[협업방식 & 느낀 점](#)

이 pdf는 아래 링크에서도 보실 수 있습니다.

<https://iodized-helmet-6aa.notion.site/Tokenizer-24-78c8242799654fb2af14cff1ed0f148a?pvs=4>

## 모듈 Introduction



main.py

YBIGTA/

——parent.py - BPE, WordTokenizer가 겹치는 기능을 구현했습니다.

——preprocessor.py - lower case 등 확장성을 고려해 간단한 전처리 모듈을 분리했습니다.

——tokenizers.py - BPE, Word Tokenizer로 구성되어있습니다.

## parent.py

```
from typing import List, Optional, Union
from YBIGTA.preprocessor import PreProcessor

class BaseTokenizer:
    def __init__(self, corpus: Optional[Union[List[str], str]] = None):
        self.preprocessor = PreProcessor(corpus)
        self.vocab = {}
        self.corpus = self.preprocessor.corpus

    def add_corpus(self, corpus: Union[List[str], str]) -> None:
        self.preprocessor = PreProcessor(corpus)
        new_corpus = self.preprocessor.corpus
        self.corpus.extend(new_corpus)

    def train(self) -> None:
        raise NotImplementedError("need child class.")

    def tokenize(self) -> Union[List[List[int]], List[int]]:
        raise NotImplementedError("need child class.")
```

```
def __call__(self, text: Union[List[str], str], padding: bool = False, max_length: Optional[int] = None):
    return self.tokenize(text, padding, max_length)
```

## 공통 기능을 담았습니다.

### 1. init

- corpus를 받아올 때 전처리해 받아올 수 있게 처리했습니다.
- Typing을 이용해 어떤 것이 인자로 들어오는지 표시했습니다.
- corpus가 들어오지 않고 그냥 객체가 생성되더라도 문제가 없도록 처리했습니다.

### 2. add\_corpus

- 기존 corpus에 extend를 할 수 있도록 처리했습니다. 역시 받아올 때 전처리를 통하도록 처리했습니다.
- Typing을 이용해 어떤 것이 인자로 들어오는지 표시했습니다.

### 3. train, tokenize

- 상속받아 사용하도록 처리했습니다.

### 4. call

- 호출하기만 해도 객체로 사용할 수 있도록 처리했습니다.

## preprocessor.py

```
from typing import List, Optional, Union

class PreProcessor:
    def __init__(self, corpus: Optional[Union[List[str], str]] = None):
        self.corpus = self.preprocess(corpus) if corpus else []

    def preprocess(self, text: Optional[Union[List[str], str]]) -> Optional[Union[List[str], str]]:
        if isinstance(text, str):
            return text.lower()
        elif isinstance(text, list):
            return [sentence.lower() for sentence in text]
        else:
            return text
```

## 전처리를 구현했습니다.

lower case로 바꿔줘 소문자와 대문자를 중복으로 계산하지 않도록 구현했습니다.

### 1. init

- preprocess를 자동으로 호출해 corpus 속성에 할당.

```
self.preprocessor = PreProcessor(corpus)
self.corpus = self.preprocessor.corpus
```

- 이런식으로 BPE, Word Tokenizer에 전달했습니다!

### 2. preprocess

- str, List[str] 입력의 양식에 따라 분기처리를 해두었습니다! 넣는 입력에 맞게 전처리를 실행합니다.
- 소문자와 대문자를 중복으로 계산하지 않도록 구현했습니다.

## tokenizers.py - BPETokenizer

## import

```
import re, collections
from typing import Optional, List, Union
from collections import Counter, defaultdict
from YBIGTA.parent import BaseTokenizer
from YBIGTA.preprocessor import PreProcessor
```

- YBIGTA 폴더에서 부모 클래스 parent.py와 preprocessor.py를 임포트해왔습니다.
- module 임포트 순서는 python 내장 모듈, 로컬 모듈 순으로 정리했습니다.

## init

```
class BPETokenizer(BaseTokenizer):
    def __init__(self, corpus: Optional[Union[List[str], str]] = None):
        super().__init__(corpus)
        self.preprocessor = PreProcessor(corpus)
        self.corpus = self.preprocessor.corpus
        self.split = {}
        self.merge = {}
        self.token_to_id = {}
        self.id_to_token = {}
        self.current_id = 0
```

- super().\_\_init\_\_(corpus)로 부모 클래스를 상속합니다.
- 만들어둔 preprocessor로 전처리합니다.
- BPE 알고리즘에서 사용할 Empty Set 을 만들어두었습니다.
- tokenize할 때 사용할 self.token\_to\_id set을 만들어두었습니다.

## train

```
def train(self, n_iter: int = None) -> None:
    # BPE 알고리즘에 따라 훈련 데이터에서 철자들을 합치고, 빈도수를 기반으로 토큰을 생성하는 로직
    # 빈도수 계산
    self.token_counts = Counter(self.corpus)

    alphabet = []
    for word in self.token_counts.keys():
        for letter in word:
            if letter not in alphabet:
                alphabet.append(letter)
    alphabet.sort()

    # add the special token </w> at the beginning of the vocabulary
    vocab = ["</w>"] + alphabet.copy()
    for token in vocab:
        # token_id를 current_id로 초기화합니다.
        self.token_to_id[token] = self.current_id
        self.id_to_token[self.current_id] = token
        self.current_id += 1

    # split each word into individual characters before training
    self.splits = {word: [c for c in word] for word in self.corpus}
    while n_iter>0:
        # pair 별 빈도수를 계산합니다.
```

```

pair_freqs = self.compute_pair_freqs()

# best pair를 초기화합니다.
best_pair = ""
max_freq = None
for pair, freq in pair_freqs.items():
    if max_freq is None or max_freq < freq:
        best_pair = pair
        max_freq = freq

# best pair를 merge합니다.
self.splits = self.merge_pair(*best_pair)

# merge한 best pair를 set에 추가합니다.
self.merge[best_pair] = best_pair[0] + best_pair[1]

# 어휘 모음에 추가합니다.
vocab.append(best_pair[0] + best_pair[1])

# tokenid를 매기는 과정입니다.
token = best_pair[0] + best_pair[1]
self.token_to_id[token] = self.current_id
self.id_to_token[self.current_id] = token
self.current_id += 1

# n_iteration이 끝나면 종료되도록 했습니다.
n_iter -= 1
return self.merge

```

## 논문의 알고리즘은 정규식에 의존했습니다.

→ 정규 표현식을 사용하며 높은 비용을 사용할 수 있으므로, 각 항목을 dictionary 처리함으로써 효율을 높이려고 하였습니다.

→ 그외 알고리즘 효율을 높이려는 시도는 실패했습니다...

- 설명은 주석으로 달아두었습니다.

## compute\_pair\_freqs

```

def compute_pair_freqs(self):
    """pair의 빈도 계산"""

    # pair 빈도 초기화
    pair_freqs = defaultdict(int)
    for word, freq in self.token_counts.items():
        split = self.splits[word]
        if len(split) == 1:
            continue
        for i in range(len(split) - 1):

            #pair를 합쳐 key로 사용
            pair = (split[i], split[i + 1])
            pair_freqs[pair] += freq
    return pair_freqs

```

- train에서 사용한 메서드이자, pair의 빈도를 구하는 메서드입니다.

## merge\_pair

```
def merge_pair(self, a, b):
    """Merge the given pair."""

    for word in self.token_counts.keys():
        split = self.splits[word]
        if len(split) == 1:
            continue
        i = 0
        while i < len(split) - 1:
            if split[i] == a and split[i + 1] == b:
                split = split[:i] + [a + b] + split[i + 2 :]
            else:
                i += 1
        self.splits[word] = split
        breakpoint()
    return self.splits
```

- 찾은 pair는 split에 업데이트 해 best pair로 또 선정되는 것을 막았습니다.
  - h, u, g → h, ug가 될 수 있도록 했습니다.

## tokenize

```
def tokenize(self, text: Optional[Union[List[str], str]],
             padding: bool = False, max_length: Optional[int] = None):
    # 분기처리
    if isinstance(text, str):
        tokens = text.split()
    elif isinstance(text, list):
        tokens = [t.split() for t in text]
    else:
        raise ValueError("Input must be either a string or a list of strings.")

    # BPE 토크나이저로 텍스트를 토큰화
    tokens_bpe = []

    for sentence_tokens in tokens:
        sentence_tokens_bpe = []

        for word in sentence_tokens:
            # train 해둔 BPE 알고리즘에 따라 토큰화
            tokenized_word = [self.token_to_id.get(char, self.current_id) for char in word]
            sentence_tokens_bpe.extend(tokenized_word)
            # 토큰화가 끝난 것들은 tokens_bpe로 append
        tokens_bpe.append(sentence_tokens_bpe)

    # 필요에 따라 패딩 수행, 각 text가 max_length를 벗어나면 truncate 되도록 설정

    if padding:
        max_len = max_length or max(len(tokens_bpe) for tokens_bpe in tokens_bpe)
        tokens_bpe = [sentence + [0] * (max_len - len(sentence)) if len(sentence) < max_len
```

- train이 끝난 모델에 corpus를 대입해 tokenize하는 과정.
- max\_length를 넘어가지 않도록 설정
- padding 인자 설정

# tokernizers.py - WordTokenizer

## init

```
class WordTokenizer(BaseTokenizer):
    def __init__(self, corpus: Optional[Union[List[str], str]] = None):
        super().__init__(corpus)
        self.preprocessor = PreProcessor(corpus)
        self.corpus = self.preprocessor.corpus
```

- 마찬가지로 parent에서 상속
- super **init**으로 상속
- 전처리기 사용

## train

```
def train(self, n_iter) -> None:
    if isinstance(self.corpus, str):
        # Tokenize based on whitespace
        words = self.corpus.split()
        # Remove duplicate words while preserving the order
        unique_words = list(dict.fromkeys(words))
        # Assign token IDs
        self.token_to_id = {word: i for i, word in enumerate(unique_words)}
        self.id_to_token = {i: word for i, word in enumerate(unique_words)}
        self.current_id = 0
    elif isinstance(self.corpus, list):
        # Tokenize each sentence based on whitespace, words에서부터 안들어가있는데...?
        words = [word for sentence in self.corpus for word in sentence.split()]
        # Remove duplicate words while preserving the order
        unique_words = list(dict.fromkeys(words))
        # Assign token IDs
        self.token_to_id = {word: i for i, word in enumerate(unique_words)}
        self.id_to_token = {i: word for i, word in enumerate(unique_words)}
        self.current_id = 0

    else:
        raise ValueError("Input must be either a string or a list of strings.")
    return self.token_to_id
```

- 분기처리를 해주었습니다.
- n\_iter 오류가 없도록 인자를 받도록 설정해주었습니다. 받아놓고 사용하지 않았습니다.
- split처리를 통해 띄어쓰기 분리를 하였습니다.
- unique\_words에 대해 tokenid를 달아주었습니다.

## tokenize

```
def tokenize(self, text: Optional[Union[List[str], str]], padding: bool = False, max_length: Opt
    self.preprocessor = PreProcessor(text)
    text = self.preprocessor.corpus
    if not self.token_to_id:
        raise ValueError("Tokenizer has not been trained. Call train method first.")

    if isinstance(text, str):
        words = text.split()
    elif isinstance(text, list):
```

```

        words = [word for sentence in text for word in sentence.split()]
    else:
        raise ValueError("Input must be either a string or a list of strings.")

    # Convert words to token ids
    tokens_word = [self.token_to_id.get(word, self.current_id) for word in words]
    # Padding if necessary
    if padding:
        max_len = max_length or len(tokens_word)
        tokens_word = tokens_word[:max_len] + [0] * (max_len - len(tokens_word))

    return tokens_word

```

- 설정해둔 token - token id 에 대해 가져가도록 설정해두었습니다.
- max\_length를 넘지 않도록 설정해두었습니다.

## Debugging Log

- 기억에 남는 이슈들을 정리했습니다.

### Debugging 방식

```
breakpoint()
```

- 2회차에서 배운 breakpoint()를 적극 활용했습니다. 아래 이슈들 말고 자잘한 디버깅에 아주 의미있게 사용했습니다.

### Issue #1 Merge? BPETokenizer?

- 논문 알고리즘을 보면 띄어쓰기 해둔 친구들을 합쳐 pair가 다시 쪼개져 나오지 않도록 처리하는데, 우선 개선은 나중에 하자, 라고 생각하고 논문 알고리즘을 그대로 갖다 썼다가 생겼던 문제였습니다.
- 한 데이터로 pair를 찾고, 합치는 걸 같이 저장하는 오류를 자꾸 범해, symbols[i], symbols[i+1]을 한 짝으로 보고, merge 메소드에서 정규식을 이용하는 걸 대체했습니다.
- BPETokenizer를 더 제대로 이해했다면 빨리 끝났겠구나, 라는 생각을 했습니다. 지능 자체를 디버깅할 순 없고, 앞으로 알고리즘부터 확실히! 대충 통치고 빨리 과제하려 하지 말고, 제대로 봐야겠다고 생각했습니다.

### Issue #2 padding이 아닌데 자꾸 나오는 0은 왜?

- padding은 분명히 앞뒤로 들어가는 친구일텐데, 왜 0이 사이사이 튀어나오는지 몰라 중간에 zero mapping을 추가해 0으로 mapping 되는 친구들을 모아 print하도록 했습니다. 양이 너무 많아 대조할 수가 없어 breakpoint보다 print로 처리했습니다.

- 가설 1 - 띄어쓰기가 들어갔나?
  - 아니었습니다. corpus를 단어로 전환하는 과정에서 띄어쓰기가 다 날아갔습니다.
- 가설 2 - 반복되는 a the 등의 정관사?
  - 아니었습니다. 0, 0이면 the the a a 인데 그럴리가 없지요.
- zero mapping print
  - 해보니 온갖 단어가 다 튀어나왔고
- 가설 3 - main.py의 indexing?
  - 혹시 리스트에서 indexing하다가 하나가 걸려 빠져서 train 안된 corpus가 들어와서 그러나?
  - 크기를 키워도, iter를 키워도, 여전히 안됐고

breakpoint()를 이용해 모든 변수를 다 확인해본 결과 부모 클래스인 BaseTokenizer의 add\_corpus에서 문제가 있었던 것을 발견했습니다.

### 알 수 있었던 점!

- 다른 모듈에서 버그가 발생할 수도 있다.
- 같은 변수를 업데이트해 다시 할당할 땐 순서를 주의

## 협업방식 & 느낀점

- 각자 혼자 진행하다가 막히는 부분이 있으면 질문하기로 했습니다!
- 유사공대 산업공학과 소속으로 간단한 파이썬 코딩만 해보고, 심지어 나만 알아보는 코드를 써왔는데, 이렇게 다른 파일을 모듈로 임포트하고 나름 긴(?) 코드도 써보고, `typing`을 이용해 변수를 명확하게 보여주는 경험을 해 너무 좋았습니다. 절대 기한 안에 못 해낼거라고 생각했는데 막상 해내니까 기분이 너무 짜릿하고 좋습니다! 야호! 배워가는 기분이 참 좋습니다.
- 스타일 가이드 읽기가 좋았습니다. 아무도 예쁜 코드를 넘겨준 적도 없고, 넘겨줘본 적도 없었는데, 조직 분위기가 그런 코드를 써야 한다고 지향하는 것이 좋았고, 공식문서를 읽는 법도 몰랐는데 어떤 순서로 읽으면 되는지 알려주셔서 더 좋았던 것 같습니다. 한학기 수업 듣는 것보다 1주짜리 과제가 더 알찼습니다. 감사합니다!






# Assignment#1

## Style Guide for Python Code

PEP 8 – Style Guide for Python Code | [peps.python.org](https://peps.python.org)

Python Enhancement Proposals (PEPs)

 <https://peps.python.org/pep-0008/#introduction>

### #1 쓰는 것 이상으로 읽을 것

- consistency를 항상 고려하자. 읽는 이의 readability

### #2 Code Lay-out

- 인덴트
  - 함수 변수와 함수 내용이 구별되게
  - Tab말고 space 4번 이용할 것
  - 예쁜 if문은?

```
if (this_is_one_thing
    and that_is_another_thing):
    do_something()
```

- txt 백슬래시로 나누기
- 연산자 앞에서 나누기
- 클래스, 메서드 사이에 한 줄~두 줄 나누기
  - 그동안 너무 붙이거나 너무 마구잡이로 떼놓았던게 아닌가 싶다...
- Encoding
  - UTF-8이 기본, 어쩔 수 없이 테스트 목적으로 사람 이름 등을 위해 non ascii code 쓸 때는 눈에 안띄는 걸로
- 모듈 임포트는 항상 한줄씩, 제일 위에, standard/related thrid party/local 순으로

crash가 날 땐

```
import myclass
import foo.bar.yourclass
```

이렇게 쓰고, myclass.Myclass, foo.bar.yourclass.YourClass로 explicitly하게 쓰기

- Module Level Dunder Names

```
from __future__ import barry_as_FLUFL
```

```
__all__ = ['a', 'b', 'c']
__version__ = '0.1'
__author__ = 'Cardinal Biggles'
```

```
import os
import sys
```

이런 식으로 중요한 모듈 임포트 전에 `dunders` 넣기

## White Space

- , 전에만. `x, y` or `y, x` / `x, y`는 no
- function annotation의 경우

```
# Correct:
def munge(input: AnyStr): ...
def munge() -> PosInt: ...# 애도 한칸 떼주기
# Wrong:
def munge(input:AnyStr): ...
def munge()->PosInt: ...
```

- 함수 매개변수로 다룰 땐 = 앞뒤로 떼지 말 것, but 기본 값을 다룰 땐 뺄 것
- 한줄에 if 문 늘어뜨리지 말 것

## Trailing Comma

```
single_element_tuple = (42,)
```

이렇게 single tuple 마지막엔 꼭 , 넣을 것

## Naming Conventions

- 모듈은 소문자로 쓸 것
- class는 각 단어 첫 글자를 대문자로 써야 함
- type 변수는 CapWords로. `T`, `AnyStr`, `Num`,
- 충돌시 `class_`가 `clss`보다 나음
- Constant는 ALL CAPITAL LETTERS
- 공개속성은 underscore하지 말 것

## Programming Recommendations

- 아래와 같이 비교 메소드 사용하기

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def __eq__(self, other):
        return self.age == other.age

    def __ne__(self, other):
        return self.age != other.age

    def __lt__(self, other):
        return self.age < other.age

    def __le__(self, other):
```

```

        return self.age <= other.age

    def __gt__(self, other):
        return self.age > other.age

    def __ge__(self, other):
        return self.age >= other.age

# 사용 예시
person1 = Person("Alice", 25)
person2 = Person("Bob", 30)

print(person1 == person2) # False
print(person1 != person2) # True
print(person1 < person2)  # True
print(person1 <= person2) # True
print(person1 > person2)  # False
print(person1 >= person2) # False

```

- 변수에 람다 할당할 바엔 함수로 만들 것
- except: '(bare except)'는 모든 예외를 잡아내서 오히려 시스템 예외도 잡아냄. 디버깅 어려울 수도.
- except Exception을 사용해서 시스템예외는 제외
- try를 할 땐 최소한의 코드 블럭만 사용.

```

# Correct:
try:
    value = collection[key]
except KeyError:
    return key_not_found(key)
else:
    return handle_value(value)

# Wrong:
try:
    # Too broad!
    return handle_value(collection[key])
except KeyError:
    # Will also catch KeyError raised by handle_value()
    return key_not_found(key)

```

- try, with, finally로 신속하게 파일 정리

```

# 예시 1: with 문 사용
with open('example.txt', 'r') as file:
    # 파일을 읽는 작업 수행
    content = file.read()
# 여기서 파일은 자동으로 닫힘

# 예시 2: try/finally 사용
file = open('example.txt', 'r')
try:
    # 파일을 읽는 작업 수행
    content = file.read()
finally:
    file.close()
# 파일이 예외 발생 여부에 관계없이 닫힘

```

- if가 return하면, else도 None을 리턴하게 해야지. robust하게 작동하게 하는 것이 목표인 듯
- 와 이건 진짜 좋다

```
# Correct:
if foo.startswith('bar'):
# Wrong:
if foo[:3] == 'bar':
```

- isinstance(object, class)로 instance 여부 확인 가능. type 쓰지 말 것
- 빈 list는 false이니까 바로 쓰기

```
if not seq
if greeting == True: # 이것도 틀리다.
if greeting is True: # Worse
```

## 읽고 나서

- 목적은 간결하고, 읽기 쉽고, 에러나지 않는 코드

### 간결한 코드

- 리스트 자체가 False를 갖거나, 변수가 굳이 == True인 걸 쓰지 않고 이용하는 방법 등으로 코드를 최소화하면서

### 읽기 쉬운 코드

- Type변수명을 CapitalWord로 구성하거나, class라는 변수가 충돌할 때 cls가 아닌 class\_로 쓰도록 하는 것은 읽기 쉬움을 위한 것이고,

### 에러나지 않는 코드

- try, except, with 등을 이용해 에러나지 않게 관리하며 코드를 짤 수 있다.

## 소감

그동안은 항상 나 혼자 읽기 좋은 거, 빨리 해서 과제 pass 할 수 있는 코드, 이런 것들만 써왔던 것 같은데, Typing을 배우고, 긴 코드를 쓰기 시작하면서 이걸 더 예쁘게 쓰고 싶다는 생각을 했던 것 같다.

그런 타이밍에 좋은 과제를 받아서 읽어서 흡수할 수 있는 좋은 기회였다. 다만, 한번 읽는다고 체화될 것 같진 않아서 여러번 더 읽어야겠다는 생각!