Q1. PEP 8 – Style Guide for Python Code

PEP 8은 파이썬의 코딩 스타일 가이드입니다. 위 가이드를 따라서 다른 개발자 그룹과의 협업이나, 시간이 지난 후의 나의 코드를 읽는 등의 상황에서 일관된, 높은 가독성의 코드를 짜도록 유도해줍니다. 그러나 PEP 8의 모든 가이드를 강제적으로 항상 따르는 것이 아니라 프로젝트의 상황에 맞게 이를 일부 수정, 혹은 개선해서 따르는 것이 가장 현명합니다.

PEP8의 목차는 다음과 같습니다. Code Lay-out, String Quotes, Whitespace in Expressions and Statements, When to Use Trailing Commas, Comments, Naming Conventions, Programming Recommendations. 이중 Code Lay-out, Whitespace in Expressions and Statements, Comments, Naming Conventions, Programming Recommendations 파트의 일부분을 간략히 정리하겠습니다.

우선 Code Lay-out에서는 들여쓰기, 띄어쓰기를 가이드합니다. 들여쓰기는 공백 4칸, 한 줄은 최대 79자까지로 제한합니다. 함수와 클래스 정의는 2줄씩 띄어 쓰며, 클래스의 메소드 정의는 1줄씩 띄어 씁니다.

Whitespace in Expressions and Statements에서는 불필요한 띄어쓰기(e.g. argument, =, default value 사이의 공백이나 조건문의 수식 사이의 불필요한 공백)은 제거하도록 가이드합니다. [ ]와 ( ), 쉼표, 쌍점(:)과 쌍반점(;) 앞의 공백 또한 불필요한 띄어쓰기입니다. Comments에서는 코드와 일치하는 필요한 주석만을 영어로 달 것을 권장합니다.

Naming Conventions에서는 다음 변수 명 규칙을 소개합니다.

_single_leading_underscore: 이런 형태의 변수는 '내부적으로 사용되는 변수'입니다.

single_trailing_underscore_: 이 형태는 파이썬의 기본 키워드와 변수명이 충돌하는 것을 방지하기 위해 사용됩니다.

__double_leading_underscore: 이 형태는 클래스의 속성명을 변경하는데 사용됩니다. 이를 통해 클래스 속성의 이름 충돌을 방지할 수 있습니다.

double_leading_and_trailing_underscore: 이 형태는 파이썬의 '매직 메소드'를 정의할 때 사용됩니다. (e.g. __init__)

Programming Recommendations의 예시는 다음과 같습니다. 1) None을 비교할 때는 is나 is not을 사용합니다. 2) 단어의 앞이나 뒤를 비교할 때는 startswith()와 endwith()을 사용합니다. 3) 객체의 타입을 비교할 때는 isinstance()를 사용합니다. 4) boolean을 조건문으로 사용할 때는 not사용이면 충분합니다. 5) improt는 파일의 맨 위에, 모듈과 패키지 간에 한 줄씩 띄워서 작성합니다. 6) string모듈보다는string메소드를 사용합니다.

Q2. Tokenizer code report

1. Class report

A. TextPreprocessor class: input type 고려하여 packing, 기호와 숫자 제거 preprocessing

B. Tokenizer class: bpe_tokenizer와 word_tokenizer의 부모 class. 공통 method 생성.

C. BPETokenizer class: bpe_algorithm의 논리에 따라 method 정의. 적절한 모듈화 진행하기.

D. WordTokenizer class: space 제거 후 bpe class와 같은 논리로 token저장.

2. BPETokenizer class의 methods 구조

- 변수는 word_freqs = {word:freq}, vocab = {pair:freq}, alphabet = [token](뒤일수록 늦게 생성된 token), splits = {word:tokenized word(list)}. word_freqs는 단어의 출현 빈도를 저장하여 vocab의 수정에 사용됨. vocab은 모든 pair와 freq를 보관하고 token이 merge될 때마다 수정됨. alphabet은 최초에 letters만을 포함하고 그 뒤로 새로운 token이 생성될 때 alphanet에 저장. alphabet에 늦게 저장될수록 corpus에서의 출현 빈도수가 적다는 뜻 -> 큰 의미를 갖는다는 뜻. (내포하는 의미는 l,o,w<low) 즉, tokenize()를 구성할 때 의미가 최대한 덜 휘발되도록 alphabet의 뒤부터 확인. splits는 corpus에 나타난 단어를 어떻게 tokenize하는지 저장. merge된 toeken들을 모두 반영. tokenize()할 때 splits에 input의 단어가 있는지를 우선적으로 확인함으로써 시간 복잡도 감소.

A. compute_word_freqs(self) -> None:

최초로 word_freqs, alphabet, splits을 생성하는 함수.

자세히는, for loop을 돌면서 word_freqs dictionary에 {word:freq} 형태로 저장하고, 동시에 각 word 안에 있는 새로운 letter도 alphabet에 저장한다. splits에는 {word:letter단위로 분리된 list}

B. get_stats(self) -> None:

최초로 vocab을 계산하는 함수.

자세히는, compute_word_freqs()에서 저장한 word_freqs를 이용해 for loop을 돌면서 pair의 frequency를 self.vocab dictionary에 {pair:freq} 형태로 저장한다.

C. merge_vocab(self, pair: List[tuple[str, str]]) -> None:

새로운 token(best pair)을 포함하는 word만 splits를 수정하고, merge로 인해 수정되야 하는 pair를 고려하여 해당하는 pair만을 수정. 시간복잡도 감소.

D. train(self, n_iter: int) -> None:

A, B로 최초 변수들 세팅. For loop마다 best pair 찾고 C 수행, best pair(new token) alphabet에 저

장.

E. find_token(self, raw_word: str) -> tuple[str, str]:

tokenize()를 위한 함수. alphabet을 뒤에서부터 보며 input이 갖고 있는 token을 찾아 tokenize후 찾은 token, input에서 token을 뺀 string output.

F. token_id(self, token: str) -> int:

tokenize()를 위한 함수. Token을 넣으면 token의 id output

G. tokenize(self, text: Union[List[str], str], padding: bool = False, max_length: Optional[int] = None) -> List[List[int]]:

Text를 tokenize하는 함수.

첫 째로 splits 확인, 없으면 find_token()을 이용하여 tokenize 수행. 이들을 token_id()를 이용해 tokenized_text에 int들로 저장. 이후 padding과 max_length operation수행.

3. 제시된 조건들을 충족시킨 방법

A. 상속활용: BPETokenizer와 WordTokenizer에서 공통으로 활용하는 전처리 관련 함수, add_corpus, tokenize, __call__ 등을 부모 class인 Tokenizer로 정의.

B. 입력 타입에 따른 분기 처리: TextPreprocessor, tokenize()

C. 의도하지 않은 입력에 대한 에러처리: 맨 처음 들어가는 TextPreprocessor를 통해 list나 str type이 아니면 ValueError 메시지를 출력.

D. 알고리즘 변경: 논문 알고리즘에서는 train의 loop를 돌 때 모든 pair에 대해서 freq를 계산했지만, 여기서는 그럴 필요 없이 merge할 때 새로 생긴 pair에 대해서만 vocab수정. splits에 저장된 단어는 빠르게 tokenize가능.

4. 협업 방식

A. 박수연: TextPreprocessor, BPETokenizer class 코드 작성

B. 이승준: BPETokenizer class의 merge_vocab, train, tokenize 함수 수정, WordTokenizer class 코드 작성