1주차 레포트

YBIGTA 24기 박정양

과제 1 소감문

이러한 문서가 나온 이유 자체가 결국 복잡한 코드를 짤 때는 많은 사람들이 함께 코드를 작성하게 되기 때문이라고 생각하였습니다. 결국에 코딩을 배우는 이유가 복잡한 코드를 작성하기 위함이고, 그 코드를 작성하기 위해 여러 사람이 협업을 진행하게 됩니다. 이 때 모든 사람이 코드를 작성하는 방법이 다르고, 코드의 가독성이 좋지 않다면 협업의 전체적인 효율성이 떨어지게 될 것입니다. 따라서 처음 배울 때부터 코드를 가독성 좋게 구현하는 습관을 만들어야 나중에 복잡한 코드를 작성할 때도 가독성이 좋은 코드를 작성할 수 있음을 깨달았습니다.
또한 가독성이 좋은 코드의 장점으로는 유지보수를 진행할 때 효율성이 좋다는 것입니다. 프로젝트를 진행한 뒤 여러가지 이유로 코드를 수정해야 하는 상황이 발생할 수 있는데, 이 때 가이드라인을 따르지 않아서 가독성이 나쁜 코드의 경우 가독성이 좋은 코드보다 어떤 부분에서 문제가 생긴것인지 확인하는 것이 어렵습니다. 따라서 유지보수를 할 때 더 많은 비용이 들게 됩니다.
다만, 협업으로 진행될 때는 해당 프로젝트의 코딩 스타일을 먼저 따라가라는 말도 있었고, 가이드라인을 따르기 어려울 때 다른 방법을 이용하라는 말도 있었습니다. 그만큼 완전한 정답은 없지만, 코딩을 하다 보면 결국 다른 사람들과 함께 코딩을 하게 되고, 이 때 충돌을 막기 위해 이러한 가이드라인이 존재한다는 것을 깨달았습니다.

과제 2 보고서

저희 Tokenizer, BPETokenizer, WordTokenizer 총 3개의 class를 구현했고, BPETokenizer와 WordTokenizer는 Tokenizer를 상속하게 하였습니다. BPETokenizer를 김채현님이, Tokenizer와 WordTokenizer를 박정양이 개발하였습니다.
main.py에서 parsing이 잘못되어 WordTokenizer가 실행되지 않는 문제가 있었습니다. 이는 파이썬이 글자가 존재하면 True로 받아들이기 때문에 –use_bpe False로 실행해도 True가 저장되는 문제였습니다. 그래서 –use_bpe의 값의 default 값을 False로 설정하고, 코드 실행 시 WordTokenizer를 쓰고 싶으면 –use_bpe를 이용하지 않음으로써 해결했습니다. BPETokenizer를 이용하고 싶으면 기존과 똑같이 –use_bpe True를 argument로 넣어주면 이용이 가능합니다.
Tokenizer class는 init, add_corpus, build_vocab, call method를 가지고 있습니다. 이 4개의 method는 BPETokenizer와 WordTokenizer 모두 공통된 내용으로 가지고 있는 method이기 때문에 부모 클래스인 Tokenizer에 구현하였습니다. Tokenizer class의 각 method의 구조와 작동원리는 다음과 같습니다.
1. init function - corpus가 입력되면 저장합니다. 이후 vocab을 초기화하고 corpus가 입

력되었다면 build_vocab 함수를 호출합니다.
2. add_corpus function - corpus를 추가로 받아서 build_vocab 함수를 호출합니다.
3. build_vocab function - corpus를 띄어쓰기를 기준으로 나누어서 vocab에 저장합니다.
4. call function - 별도의 메소드 호출 없이 그냥 객체 자체를 call해도 tokenize처럼 사용할 수 있게 하기 위해 필요한 함수입니다.

BPETokenizer class는 Tokenizer class를 상속받지만, init 함수에 bpe_codes를 초기화하는 것이 필요하기 때문에 init function을 새로 만들어주었습니다. 그 외에 get_stats, merge_tokens, train, tokenize method가 존재합니다. 각 method의 구조는 다음과 같습니다.
1. get_stats - 글자가 연속해서 나오는 횟수를 알아냅니다. vocab에 단어별로 corpus에 등장한 횟수인 frequency가 저장되어 있을 때, vocab의 모든 key 값을 글자별로 하나씩 보면서 다음 단어와의 조합에 해당하는 pairs vocabulary에 frequency값을 더해줍니다.
2. merge_tokens - 2개의 토큰을 병합하는 함수입니다. 2개의 string을 입력받아서 병합을 진행한 뒤 vocabulary에 추가합니다.
3. train - 입력받은 iteration 횟수만큼 빈도가 가장 높은 pair를 뽑아서 merge를 진행합니다.
4. tokenize - 입력받은 text를 vocab에서 찾아서 token으로 변경합니다.

WordTokenizer class에는 train, tokenize method가 존재합니다.
1. train function - [UNK] token이 필요해서 vocab에 강제로 넣어줬습니다. 그 이후 token과 index를 matching 하여 dictionary를 생성하는 코드입니다.
2. tokenize function - 새로 들어오는 text를 현재 존재하는 vocab을 이용해서 tokenize하는 코드입니다. text에 존재하는 문장들을 단어 단위로 쪼갭니다. 쪼개진 단어가 vocabulary에 있으면 token index로 나타내고, 없으면 [UNK]의 index로 나타냅니다.