boostcamp ai tech

NAVER CONNECT 부스트캠프 AI Tech

NLP KLUE 프로젝트

Relation Extraction
문장 내 개체간 관계 추출

Member
김주원_T5056, 강민재_T5005, 김태민_T5067, 신혁준_T5119 윤상원_T5131

1. 프로젝트 개요

프로젝트 주제	**문장 내 개체간 관계 추출(KLUE RE):** 문장의 단어(Entity)에 대한 속성과 관계를 예측하는 NLP Task
프로젝트 구현내용	1. Hugging Face 의 Pretrained 모델과 KLUE RE 데이터셋을 활용해 주어진 subject, object entity 간의 30 개 중 하나의 relation 예측하는 AI 모델 구축 2. 리더보드 평가지표인 Micro F1-Score 와 AUPRC 높은 점수에 도달할 수 있도록 데이터 전처리(Entity Representation), 데이터 증강, 모델링 및 하이퍼 파라미터 튜닝을 진행
개발 환경	**GPU:** Tesla V100 서버 4 개 (RAM32G) /Tesla V4 (RAM52G) /GeForce RTX 4090ti 로컬 (RAM 24GB) **개발 Tool:** PyCharm, Jupyter notebook, VS Code [서버 SSH 연결], Colab Pro +, wandb
협업 환경	**Github Repository:** Baseline 코드 공유 및 버전 관리 **Notion:** KLUE 프로젝트 페이지를 통한 역할분담, 대회 협업 관련 Ground Rule 설정, 아이디어 브레인 스토밍, 대회관련 회의 내용 기록 **SLACK, Zoom:** 실시간 대면/비대면 회의

2. 프로젝트 팀 구성 및 역할

- 모더레이터 외에도 Github 관리자를 두어 베이스라인 코드의 버전 관리를 원활하게 하고, 같은 분야라도 다른 작업을 진행할 수 있도록 분업을 하여 협업을 진행하였다.

이름	역할
강민재	**EDA**(길이, 레이블, 토큰, entity 편향 확인)**, Error Analysis, 데이터 증강**(단순 복제, class inverse 관계 교체 증강)**, 데이터 전처리**(subject, object entity 스페셜 토큰 처리)
김태민	**모델 실험**(Attention layer 추가 실험, Linear/LSTM layer 추가 실험, Loss, Optimizer 실험)**, 데이터 전처리**(Entity Representation – ENTITY, Typed Entity)
김주원	**모델 튜닝, 프로젝트 매니저**(노션관리, 회의 진행)**, EDA, 모델 앙상블**(Hard Voting, Soft Voting, Weighted Voting)**, Error Analysis**(Error Analysis 라이브러리 개발)
윤상원	**Github 베이스라인 코드 관리**(코드 리팩토링, 버그 픽스, 코드 버전 컨트롤)**, 모델 실험**(TAPT 적용)**, 데이터 전처리**(Entity Representation – ENTITY, Typed Entity)**, EDA**(UNK 관련 EDA)**, 모델 튜닝**
신혁준	**EDA**(데이터 heuristic 체크, Label 별 관계 편향 조사)**, 데이터 증강** (동일 entity start_idx, end_idx 교체, Easy Data Augmentation – SR 기반 증강, 클래스 Down Sampling)

3. 프로젝트 수행 절차 및 방법

팀 협업을 위해 개선점 파악을 위해 지난 NLP 기초 프로젝트 관련한 회고를 진행하였다. 회고를 바탕으로 프로젝트의 팀 목표인 "함께 성장"과 "실험 기록하기"를 설정했다. 그리고 목표를 이루기 위한 Ground Rule 을 설정하여 프로젝트가 원활하게 돌아갈 수 있도록 팀 규칙을 정했다. 또한, 날짜 단위로 간략한 목표를 설정하여 협업을 원활하게 진행할 수 있도록 계획을 하여 프로젝트를 진행했다.

3.1 협업 관련 Ground Rule

-a. 실험 & 노션 관련 Ground Rule: 본인 실험을 시작할 때, Project 대시보드에 본인의 작업을 할당한 뒤 시작한다. 작업은 '하나의 아이디어' 단위로 생성하고 Task, 진행상태를 표시한다. 작업이 마무리 되면 실험 결과가 성능의 향상에 상관없이 '실험 대시보드'에 기록하고 상태를 완료 표시로 바꿔 마무리한다.작업 단위로 관리하되 그 실험을 어떤 가설에서 진행하게 되었는지, 성공했다면 성공했다고 생각하는 이유, 실패했다면 실패한 이유에 대해 간략하게 정리한다.

-b. Commit 관련 Ground Rule:

- **1. 전체 main branch Pull Request 관련 Rule** : main branch 에 대한 pull request 는 Baseline Code 를 업데이트 할 때 마다 진행한다. commit message 에는 점수, 데이터, 버전 내용이 들어가도록 작성하고 push 한다.

- **2. 개인 Branch Commit 관련 Rule** : git commit & push 는 코드의 유의미한 변화가 있을 때 마다 진행한다. Commit message 에는 코드 수정 내역(추가/변경/삭제)이 들어가도록 작성하고 push 한다.

-c. Submission 관련 Ground Rule: 각 사람별로 submission 횟수는 2 회씩 할당한다. 추가로 submission 을 하고 싶으면 SLACK 단체 톡방에서 다른 캠퍼에게 물어봐 여유횟수를 파악한 뒤 추가 submission 을 진행한다. Submission 을 할 때에 다른 팀원이 어떤 실험의 submission 인지 파악할 수 있도록 사용한 모델, 데이터, 기법, 하이퍼파라미터 등이 들어갈 수 있도록 Description 을 기입한다.

-d. 회의 관련 Ground Rule: 전원이 진행하지 않는 Small 회의는 다양한 방식(Zep, Google Meet, Zoom)으로 진행하고 회의 내용을 기록한다.

3.2 프로젝트 진행 Time line

- 세부적인 대회 진행 절차는 아래와 같다.

내용	05/02	05/03	05/04	05/08	05/09	05/10	05/11	05/12	05/15	05/16	05/17	05/18~05/22
강의 듣기												
모든 강의 완강 & 대회 사전 조사												
강의 & 플랫폼에서 활용할 수 있는 지식 토론												
베이스 라인 코드 개발												
베이스라인 코드 이해 & 협의												
베이스라인 코드 완성(Pytorch lightning)												
모델 실험(Layer추가,Optimizer, loss 변경 등)												
데이터 전처리(Entity Representation)												
코드 버전 컨트롤												
EDA												
사전 데이터 분석												
Output에 대한 Error Analysis & 방향 설정												
데이터 증강												
보고서 작성												

4. 프로젝트 수행 결과

4.1. 순위

순위	분류	micro_f1	auprc	제출 횟수
7	Public Score (대회 진행)	75.5115	80.5521	53
8	Private Score(최종)	73.9434	82.7705	53

4.2 베이스라인 코드 리팩토링

지난번 대회에서 나왔던 피드백인 다음 대회에서는 모델을 직접 커스터마이징을 해보자는 의견에 따라, 기존 huggingface trainer 로 짜여 있던 베이스라인 코드에서 상대적으로 모델을 커스터마이징하기 용이한 pytorch lightning 으로 리팩토링을 진행했다.

4.2.1 기존 베이스라인에서의 개선점

-a. 문제점 해결: 기존의 baseline 에서 subject 와 object 를 파싱하는 부분에서 딕셔너리 형태의 값을 문자열로 읽어와서 ‘,’와 ‘:’를 구분자로 하여 분리하고 있기 때문에 ‘,’ 또는 ‘:’가 객체 안에 들어가 있는 경우, 객체의 값을 잘못 파싱한다는 문제점이 있다는 것을 파악하고, 해당 부분의 버그를 수정했다.

- b. 기록 방식 보안: f1 score 와 loss, learning rate, confusion matrix 와 같은 다양한 지표들을 로그로 남겨 학습 진행 상황을 실시간으로 확인할 수 있게 하였고, wandb 와 연동하여 이를 시각적으로 확인하고, 기록을 남길 수 있게 했다.

- c. 수정이 쉬운 코드: 베이스라인과 달리 config parser 와 yaml 파일을 이용하여 팀원들이 수정하기 쉽고, hard coding 시 발생할 수 있는 부분들을 최소화할 수 있도록 베이스라인 코드를 배포하여 팀원이 실험을 원활하게 진행할 수 있도록 하였다.

4.2.2 베이스라인 코드 작성시 유의점

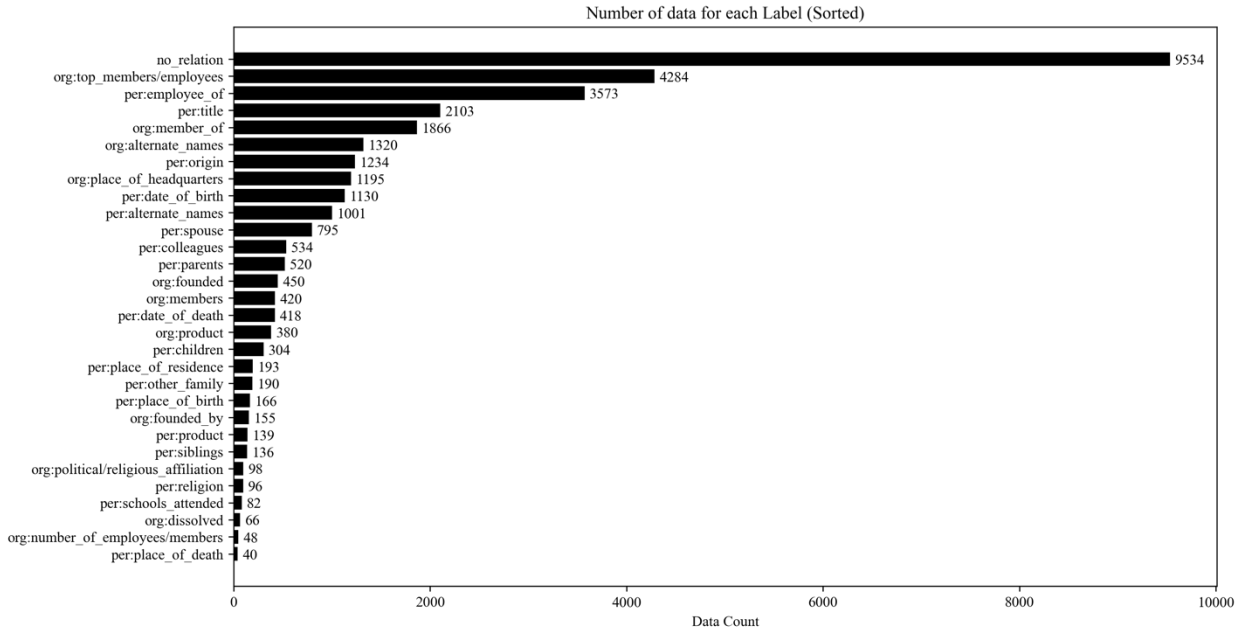
코드에 대한 깊은 이해 없이, 의도치 않은 방식으로 코드가 동작하는 상황이 없게끔 pytorch lightning 의 공식 문서[1]와 소스 코드를 확인하며 구현했다. 또한, 팀원 모두가 사용하는 베이스라인 코드이다 보니, 팀원들이 코드를 확장하기 쉽도록, 토크나이저에 special token 을 추가할 수 있도록 구성하고, 모델의 embedding layer 의 크기를 토크나이저의 크기와 맞추는 등 예외 상황들을 고려하여 코드를 구현했다.

4.2. EDA(Exploratory Data Analysis)

주어진 데이터의 문장의 길이 관련, 클래스 분포, 데이터 Entity 와 관련된 데이터 분석을 진행했고, 이를 통해 문제 해결 전략을 수립하였다. 분석을 바탕으로 가설을 세웠다.

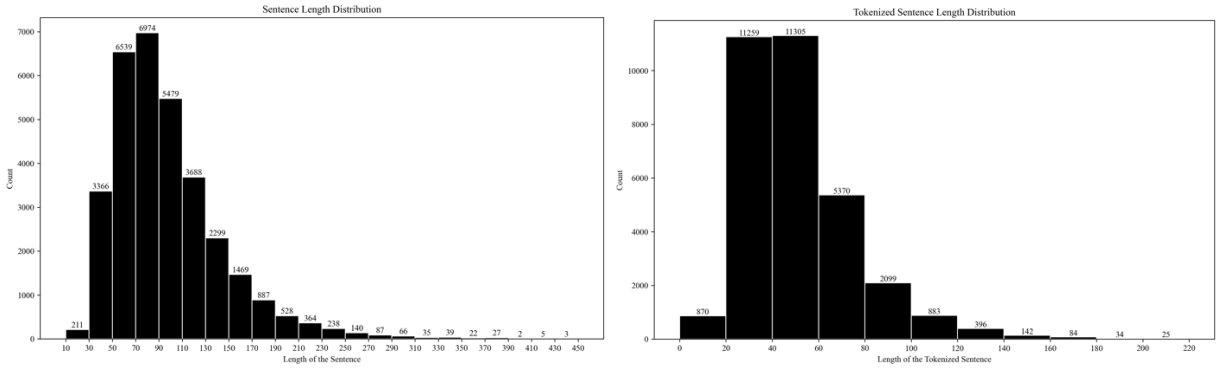
4.2.1 라벨 분포 관련 EDA

train data 의 정답(label) 클래스가 총 30 개이며, 각 클래스별 데이터 개수를 확인하였다. no-relation 을 비롯한 3 개 클래스가 유독 많은 데이터를 차지함을 확인하였다. 이번 task 의 평가 지표인 micro-f1 점수를 구할 때 제외되는 no-relation 을 차치하더라도, 아래의 그래프에서 확인 할 수 있듯이 가장 적은 데이터를 갖는 per:place_of_death 와 가장 많은 데이터를 갖는 org:top_members/employees 의 데이터 개수 차이가 100 배 이상임을 확인하였다. 따라서 train data 에서 class imbalance 를 발견하였고, validation data 가 주어지지 않은 상황에서 적절한 데이터 split 과 증강이 필요하다고 판단하였다.

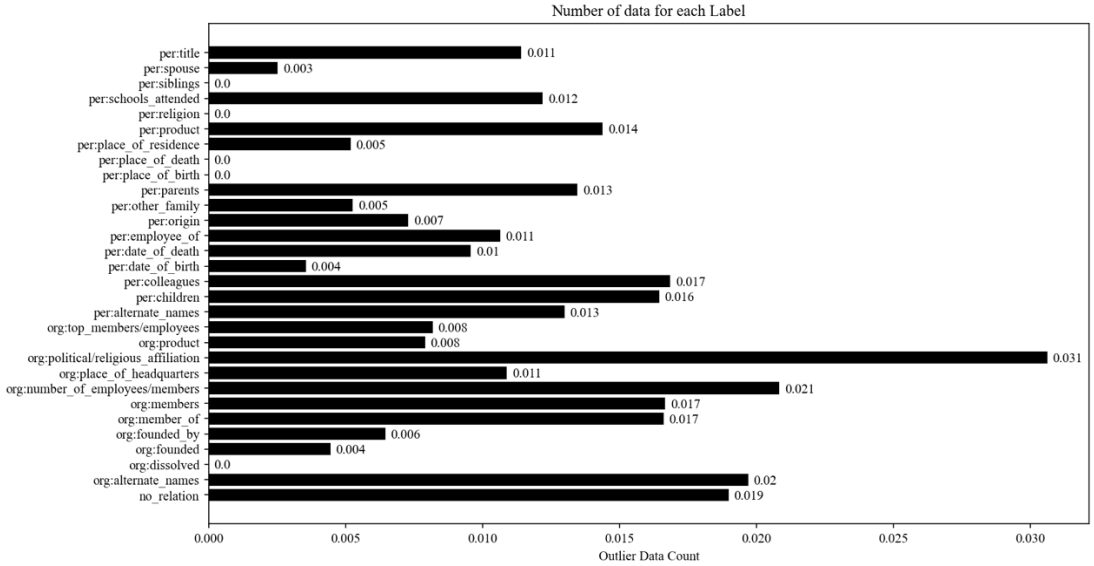


4.2.2 문장 길이/Tokenizing 관련 EDA

Train data 의 원본 문장의 길이 분포를 확인하였고, 원본 문장에 대해서 sentence length 가 250 을 초과하는 데이터를 outlier 로 설정하였다. 분석 결과 sentence length 에 대한 outlier 가 존재함을 발견하였다. 추가로 베이스라인 모델인 klue/bert-base 의 토크나이저로 토큰화한 문장에 대한 길이 분포도 원본 문장과 유사함을 확인하였다.길이 편향에 대한 가설을 설정하기 위해서는 전체 데이터에 존재하는 문장 길이에 대한 outlier 의 개수보다는, 각 클래스별로 outlier 의 비율을 확인하는 게 의미가 있다고 판단하였다.



아래 그래프에서 확인할 수 있듯이 org:political/religious_affiliation 를 제외하고는 길이가 긴 문장의 비율이 특히 높은 클래스는 없었다. sentence length 에 대한 outlier 비율이 높은 'org:political/religious_affiliation' 클래스에 대해서는 길이에 대한 편향이 발생할 수 있다고 판단하였다.

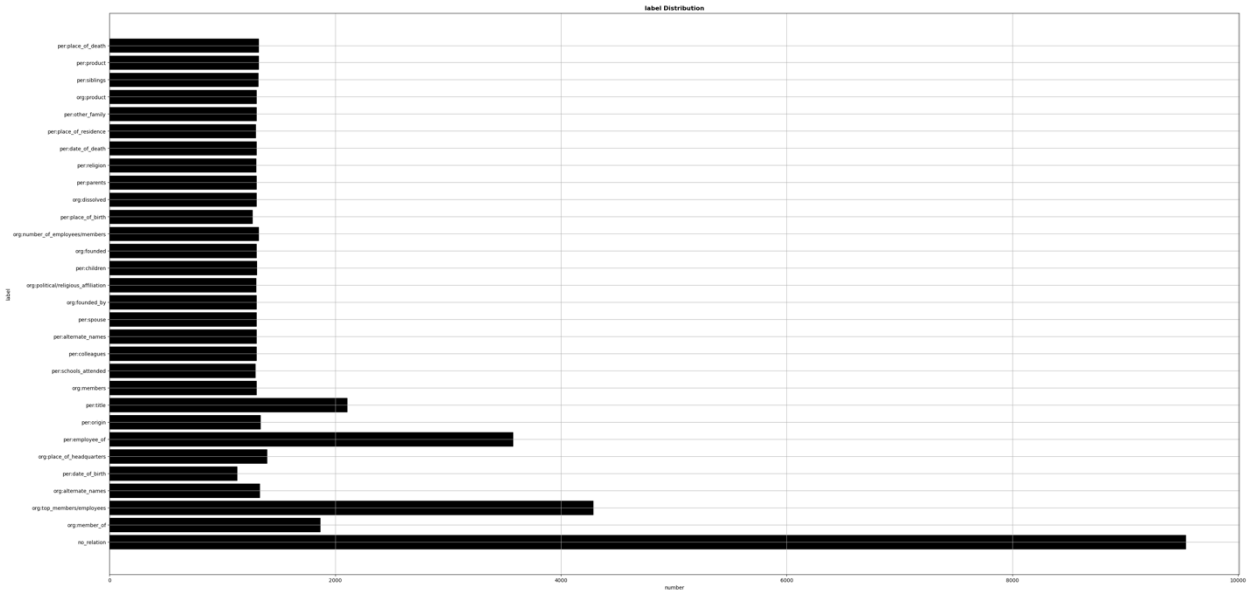


4.3 데이터 증강 : MLM(Masked Language Model) BERT 모델을 이용한 데이터 증강

- a. 구현 방법: 아래와 같이 학습을 할 때는 아래와 같이 train 과 test 데이터의 기존 문장에 랜덤하게 MASK 토큰를 씌워 klue/roberta-large 모델을 이용하여 MLM 학습을 2 epoch 만큼 진행한다. 이후에 데이터 증강을 진행할 때는 train data 에 대해서만 증강하고, 랜덤하게 Masking 을 단어에 씌우는게 아니라 단어와 단어 사이에 배치한다. 그리고 각 모델마다 확률 적으로 높은 n 개의 mask 를 내놓고, 따라서 새롭게 증강된 모델은 기존 train 데이터에서 관계가 바뀌지 않은 채 원본 문장에서 하나의 토큰이 추가된 문장 데이터를 생성한다.

분류	문장
원본 데이터	결국, 대한민국 정부의 반대에도 불구하고 주한미군은 약 500 명의 군사고문단만 남기고 마지막 남아 있던 부대가 1949 년 6 월 29 일 철수하였다.
학습데이터의 Masking 예시	결국, 대한민국 정부의 **[MASK]** 불구하고 주한미군은 약 500 명의 군사고문단만 남기고 마지막 남아 있던 부대가 1949 년 6 월 29 일 철수하였다.
증강이전 데이터의 Masking 예시	결국, 대한민국 정부의 반대에도 불구하고 주한미군은 **[MASK]** 약 500 명의 군사고문단만 남기고 마지막 남아 있던 부대가 1949 년 6 월 29 일 철수하였다.
증강 이후 데이터	결국, 대한민국 정부의 반대에도 불구하고 주한미군은 **당시** 약 500 명의 군사고문단만 남기고 마지막 남아 있던 부대가 1949 년 6 월 29 일 철수하였다.

문장을 생성한 뒤 분포가 이미 풍부히 많고 error analysis 결과 f1-score 가 높은 여섯 개의 class (no_relation, per:title, org:place_of_headquarters, per:employee_of, per:date_of_birth, org:member_of)를 제외한 나머지 클래스 데이터를 랜덤하게 sampling 하여 2 만개의 데이터를 증강하여 아래와 같은 분포를 띄는 증강 데이터를 만들었다.



- b. 구현 이유(가설) :. 해당 기법으로는 증강시 문장 안에 있는 띄어쓰기 수만큼 마스킹을 진행할 수 있고, 각 Masking 마다 확률적으로 높은 토큰의 단어를 여러 개를 선택하여 조합할 수 있어서 많은 양의 데이터를 증강하여 분포를 맞출 수 있다고 생각했다. 따라서 앞서 EDA 를 통해 살펴보았던 데이터의 class 불균형 문제를 해소하여 성능 향상을 가져올 것이라고 판단했다. 또한 BERT 는 positional Embedding 을 통해 구조를 학습하게 되기 때문에 이러한 방식으로 증강을 할 경우 문장에 구조적인 변화를 가지고 올 수 있기 때문에 학습을 할 경우 더 어렵게 학습을 하게 되어 성능이 향상이 될 것이라고 가정했다.
- c. 결과: klue/bert-base 모델에서 public score 가 64.5 점에서 66 점으로 1.5 점 상승했다.

4.4. 데이터 전처리 :: Entity Representation

-a. 가설: *An Improved Baseline for Sentence-level Relation Extraction* 논문[2]을 참고하면 아래와 같이 새로운 Entity Representation 를 적용할 경우 두 개체 간의 관계를 더 정확하게 추론하는 RE Task 에서 성능 향상을 이뤄낸 것을 확인 할 수 있었다. 따라서 논문[2]에 제시된 Entity Marker, Punct Entity marker, Typed Entity Marker 등을 적용한다면 성능의 향상이 이뤄진다고 생각했다.
-b. 구현 내용 & 결과 : 아래와 같은 example 이 출력될 수 있도록 Entity Marker, Typed Entity, SUB/OBJ marker, Punct Entity Marker 를 구현하고 klue/roberta-large, klue/bert-base 에 적용하였다. 모든 entity 에서 0.8~3.4 점 정도의 성능 향상이 이루어졌다.

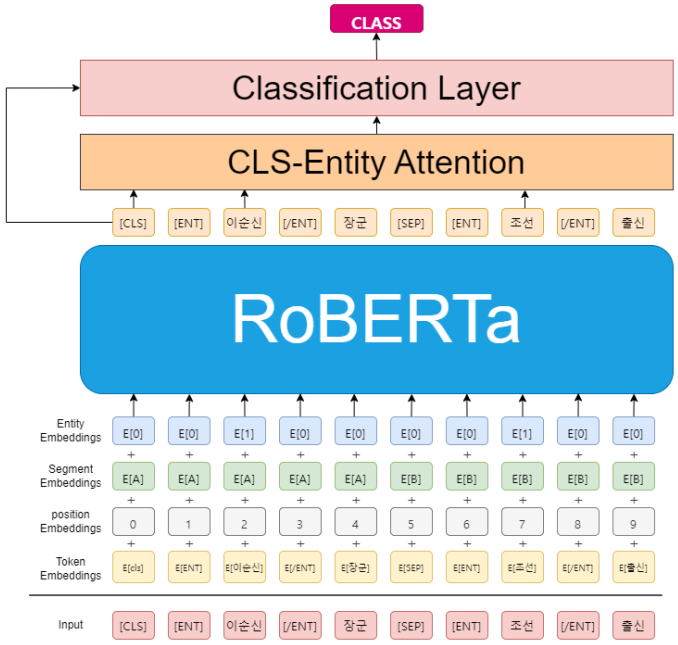
Method	Input Example	bert(base)	RoBerta(large)
Baseline	〈Something〉는 '조지 해리슨'이 쓰고 '비틀즈'가 1969 년 앨범 《Abbey Road》에 담은 노래다.	64.5	67.1
Entity marker	〈Something〉는 [Entity]조지 해리슨[/Entity]이 쓰고 [Entity]비틀즈[/Entity]가 1969 년 앨범 《Abbey Road》에 담은 노래다.	66.5(+2.0)	-
Typed entity marker	〈Something〉는 [PER]조지 해리슨[/PER]이 쓰고 [ORG]비틀즈[/ORG]가 1969 년 앨범 《Abbey Road》에 담은 노래다.	66.7(+2.2)	-
SUB,OBJ marker	〈Something〉는 [OBJ]조지 해리슨[/OBJ]이 쓰고 [SUB]비틀즈[/SUB]가 1969 년 앨범 《Abbey Road》에 담은 노래다.	65.3(0.8)	-
punct(한글)	〈Something〉는 @*사람*조지 해리슨@이 쓰고 @*단체*비틀즈@가 1969 년 앨범 《Abbey Road》에 담은 노래다.	-	70.8(+3.4)

* '-' 로 체크 된 부분은 시간 관계상 실험이 진행되지 못한 부분임

* punct 에서는 논문[2]에서는 영어로 표기 되지만, 한국어 데이터임으로 한국어 데이터에 맞게 수정하여 구현하였음.

4.5. 모델 실험

4.5.1 CLS-Entity Attention

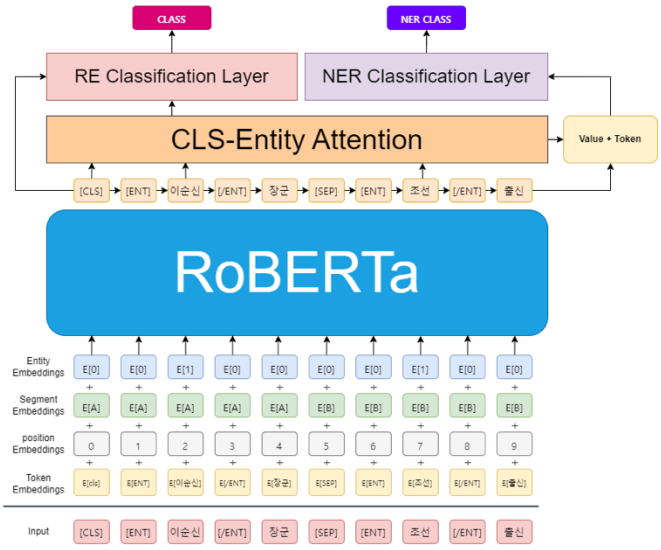


-a. 가설 : CLS Token 은 문맥의 모든 정보를 담아내고 있으며 각 토큰의 객체 정보만을 남겨서 Attention 을 진행해주어 이를 최종 입력에 더해줌으로써 조금 더 문맥에서 집중해야 할 객체 정보를 표기할 경우 모델이 객체를 보다 더 잘 인식할 것이다.

-b. 구현 방법 : 객체 토큰만 1 이고 나머지가 0 인 index_ids 라는 텐서를 만들고 torch.zeros 를 통해 Attention 입력과 동일한 사이즈의 텐서를 만들어준다. 이를 index ids 와 결합하여 객체 토큰의 정보만 1 이고 나머진 0 인 텐서를 만든다. 추후 객체 토큰 텐서(0,1 만있는 output size 와 동일한 tensor)와 기존 모든 토큰(roberta output token)을 곱셈하여 객체 토큰 텐서의 정보만 유지해주고 이를 CLS token 과 Attention 을 진행해 최종 입력에 더해준다.

-c. 결과 : 기존 Typed Entity 적용된 klue/roberta-large 모델이 70 점이었으나, 해당 모델에 CLS-Attention 을 적용한 기법은 71 점으로 1 점 정도의 성능향상을 이뤄냈다.

4.5.2 Multi Tasking Learning

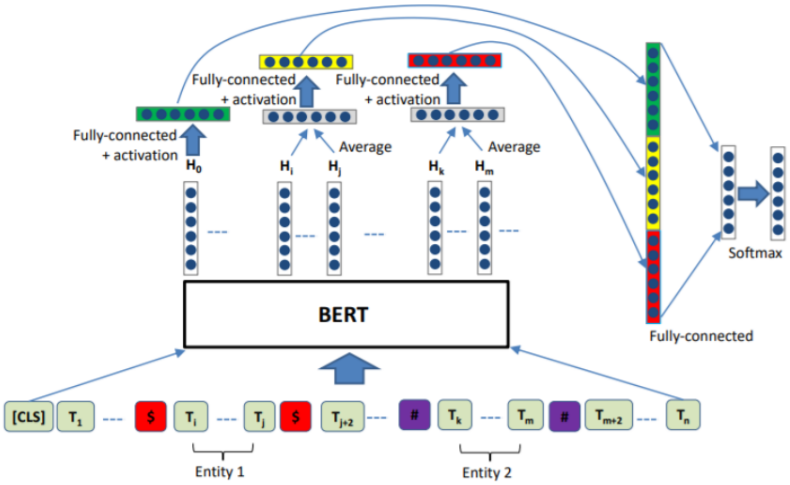


-a. 가설: 기존 RE task 말고 NER task 도 함께 수행하여 BIO 태깅 정보까지 학습에 활용한다면. 각 토큰의 태깅 명까지 유추함으로 모델이 조금 더 객체에 집중할 것이라고 가설을 세웠다.

-b. 구현방법: 각 토큰 SUB, OBJ 토큰들을 기준으로 type 을 BIO 태깅으로 한 라벨을 생성해낸다. 이를 모델 학습 코드에서 활용하여 최종적인 기존 RE task 의 loss 와 NER task 의 loss 를 서로 더하여 역전파를 시켜준다.

-c. 결과: klue/roberta-large 모델에 CLS-Token Attention 과 함께 적용하여 72.7 점의 f1-score 가 나왔고, 기존 CLS-Token Attention 모델의 71 점보다 1.7 점 상승하였다.

4.5.3 R-BERT



- a. 가설: *Enriching Pre-trained Language Model with Entity Information for Relation Classification* 논문[3]에서 나온 R-BERT 의 경우 모든 토큰의 정보를 취합하여 새로운 Linear layer 를 정의한다. 논문[3]의 내용과 마찬가지로 한글 task 에서도 성능 향상이 있을 것이라고 생각하였다.
- b.구현방법: CLS Token 그리고 각 객체 토큰들은 Average 하여 Fully Connected layer 와 tanh 활성화 함수를 통과시킨다. 추후 마지막 차원으로 모든 토큰을 concat 시켜 이를 Fully Connected layer 에 넣어준다. klue/bert-base 는 256 차원의 FC layer 를 이용했고, klue/roberta-large 는 512 차원의 FC layer 를 이용했다.
- c. 결과 :klue/bert-base 모델과 klue/roberta-large 에 적용하였고 적용 전후로 각각 아래와 같이 성능이 향상하였다.

	klue/bert-base	klue/roberta-large
Baseline 모델	64.5	67.1
FC Layer 적용 이후	66.0(+1.5)	67.4(+0.3)

4.5.3. TAPT
- a. 가설: *Don't Stop Pretraining: Adapt Language Models to Domains and Tasks* 논문[4]에서 Task 에 특화된 적은 양의 데이터를 한번 더 사전 학습시킴으로써 모델의 성능이 올랐다는 내용을 참고하여 문장 내 개체 간 관계 추출 task 에 특화된 데이터를 사전 학습시키면 해당 유형의 문장들에 대한 이해도가 높아져 모델의 성능이 좋아질 것이다 라는 가설을 세웠다.
-b.구현 방법: 이번 대회에서 주어진 train 데이터와 test 데이터를 데이터 셋으로 활용하여 사전 학습된 모델에 추가로 Masked Language Modeling 을 실시했다. 1 epoch 학습을 진행하였고, learning rate 는 모델을 제안한 논문[4]을 조사하여 사전 학습에서 사용된 learning rate 의 1/10 값을 사용했다.
- c. 결과 : klue/bert-base 모델의 경우, public f1 score 기준 TAPT 를 적용한 이후 64.5 점에서 65.4 점으로 올라 0.9 점가량 상승하였고, klue/roberta-large 모델의 경우, public f1 score 기준 TAPT 를 적용한 이후 67.1 점에서 68.7 점으로 올라 1.6 점 가량 상승했다.

4.5.4 Optimizer, Loss 수정
- a. AdaBelief Optimizer: *Adapting Stepsizes by the Belief in Observed Gradients* 논문[5]에서 AdamW 대신 Adabelief 를 사용하여 BLEU score 를 0.2 점 정도 높인 것을 확인하였다. 따라서 이 optimizer 를 활용한다면 성능 향상을 이뤄낼 수 있을 것이라고 생각하여 adabelief-pytorch 의 라이브러리를 활용하여 optimzer 를 Adablief 로 교체했고, 기존 klue/bert-base 베이스라인에서 public 이 64.5 이었던 점수를 Adabelief 는 66.9 로 약 2.4 점 가량 성능을 향상시켰다.

-b. Focal Loss: 기존의 Cross Entropy 함수는 클래스 불균형에 대해 약한 경우가 있다. *Focal Loss for Dense Object Detection* 논문[6]에서 해당 경우에 Loss 함수를 Focal Loss 함수로 클래스 불균형에 강하다는 실험 결과를 확인하였다. 따라서, 데이터셋이

클래스 불균형이 심해 효과가 있을 것이라고 생각하였고, 이를 적용하였다. 그 결과 성능 향상은 미비하지만 R-BERT 모델에서 66.4 에서 66.5 로 성능이 0.1 점 상승함을 확인하였다.

4.6 모델 앙상블

앙상블은 다음과 같은 총 3 가지 테크닉을 이용해서 진행했다.
- a. Hard Voting 앙상블: 여러 개의 모델 중 가장 많은 모델이 예측한 class 를 최종적으로 선택하는 방식으로 앙상블을 진행했다. 동점 클래스가 발생하여 가장 많이 예측한 클래스가 2 개 이상일 경우 최종 후보군의 클래스 들 중 확률이 더 높은 값을 선택하여 앙상블을 진행했다.
- b. Soft Voting 앙상블: 모델들의 예측 확률을 클래스 별로 모두 더해 평균을 구해 가장 높은 평균을 가진 클래스를 선택하는 방식으로 앙상블을 진행했다.
- c. F1-Score Weighted 앙상블: 각 모델들의 f1-score 을 softmax 값을 취해 각 모델의 확률에 곱하여 가중치를 부여한다. 이후 해당 모델들의 클래스 별로 합을 구하고, 평균을 취해 가장 높은 확률을 가진 class 를 추가하여 모델의 점수를 구했다.

위의 모델 기법 및 증강데이터를 선택하여 적용한 모델들을 세가지 앙상블을 적용해 여러 번 실험을 진행했다. 최종적으로 klue/roberta-large 모델에서 TAPT 를 적용한 모델 들 중 Punct Entity Marker 가 적용된 모델, 일반 Entity Marker 가 적용된 모델, label smoothing 과 증강 데이터가 적용된 모델 총 세개의 모델을 soft voting 기법으로 만든 앙상블을 진행했고, private micro-f1 score 가 73.9434 이고, AUPRC 가 82.7705 점의 점수를 얻을 수 있었다.

5. 자체 평가 의견

5.1. 잘한 점

- 베이스라인 코드를 pytorch lightning 으로 리팩토링하여 모델을 직접 커스터마이징해 볼 수 있었다.
- 노션에 모든 실험을 기록하며 대회를 진행했다.
- 파트장을 통한 깃허브 버전 관리가 우수했다.
- 실험 기록에 대한 체계를 구체화하였고, 팀원들이 모두 대회 시작 이전에 정한 룰에 따라서 각자의 역할을 수행하였다

5.2 실험에서 실패한 증강 실험

-a. 단순 복제 증강: EDA 과정에서 길이에 대한 편향의 발생 가능성을 확인했음에도 불구하고, 심각한 label imbalane 로 다른 증강 기법의 한계로 인해 단순 복제 증강을 수행하였다. 따라서 일부 클래스는 같은 문장이 20 번 이상 반복되었고, 이에 따라 길이에 대한 편향이 발생하여 성능이 저하되었다고 생각한다.
-b. SUB/OBJ Entity swap 을 통한 증강: org:members 와 org:members 와 같이 역관계에 있는 entity 의 데이터를 Entity swap 을 통해 데이터를 증강하고 관계가 달라진 부분을 반영해 대응하는 클래스로 label 을 변경한느 방식으로 증강하였다.또한, subject 와 object 에 대한 학습이 원활하게 이루어지도록 데이터 피딩 방식을 변경하였다. 이 과정에서 스페셜 토큰을 총 24 개 추가하였는데, 너무 많은 스페셜 토큰을 추가한 데 비해 일부 클래스에서 학습 데이터가 충분하지 않았기 때문에 오히려 모델 학습에 방해가 되었다고 생각한다. 추가로 데이터 증강 섹션에 첨부한 figure 에서 확인할 수 있듯이, 일부 클래스를 대상으로만 증강이 적용되어, 여전히 label imbalance 문제가 남아있는 상태였기 때문에 증강 기법의 완성도 측면에서도 문제가 있었다고 생각한다.
-c. Downsampling: Label 분포 비율이 높은 'no_relation', 'org:top_members/employees', 'per:employee_of'를 1500 까지 줄여 분포를 맞췄다. Downsapling 이후 Public Micro F1 Score 가 62.18 에서 58.43 으로 하락했다.

5.3 프로젝트 협업 프로세스에서 아쉬웠던 점

- log 를 wandb 뿐만 아니라, 파일로도 확인할 수 있도록 python 의 기본 라이브러리인 logging 을 사용해서 실험 결과를 확인할 수 있도록 해야될 것 같다.
- 그리고 베이스라인을 수정하고 파이토치 라이트닝을 써보면서 라이트닝만의 장단점을 알 수 있었다.
- wandb 그리고 노션을 통한 기록이 중요하다고 생각된다.

5.4 프로젝트를 통해 배운점 또는 시사점

- 직접 loss function, optimizer, lr scheduler 등을 수정하면서 이러한 요소들이 모델 학습에 미치는 영향을 더 직관적으로 느낄 수 있었다.

- Data 증강할 때는 구체적으로 가설을 나누어서 세우고 증강해야겠다. 단순히 Data 분포를 맞추겠다는 목표보다 error anslysis 를 기반으로 조금 더 세밀하게 증강해야겠다.
- 문제를 해결할 때, 주어진 문제를 단계별로 쪼개어 적절한 문제정의를 단계별로 해석하는 것이 중요하다는 점을 깨달았다. 또한 필요에 따라서는 ML/DL 기법 뿐만이 아닌 사람이 직접 데이터를 살펴보고 rule base 알고리즘을 적절히 적용하는 것도 모델 성능을 향상할 수 있는 방법임을 배웠다.

6. 참고 문헌(References)

[1] Pytorch Lightning 공식 Document, https://lightning.ai/docs/pytorch/latest/
[2] Wenxuan Zhou, Muhao Chen, An Improved Baseline for Sentence-level Relation Extraction, ACL (pp161–168), Feb 2021
[3] Shanchan Wu, Yifan He, Enriching Pre-trained Language Model with Entity Information for Relation Classification, CIKM '19: Proceedings of the 28th ACM International Conference on Information and Knowledge Management, Pages 2361–2364, May 2019
[4]Suchin Gururangan, Ana Marasović, Swabha Swayamdipta, Kyle Lo, Iz Beltagy, Doug Downey, Noah A. Smith. Don't Stop Pretraining: Adapt Language Models to Domains and Tasks. ACL, April 2020.
[5] Juntang Zhuang, Tommy Tang, Yifan Ding, Sekhar Tatikonda, Nicha Dvornek, Xenophon Papademetris, James S. Duncan, AdaBelief Optimizer: Adapting Stepsizes by the Belief in Observed Gradients ,NeurIPS2020,Oct 2020
[6] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, Piotr Dollár, Focal Loss for Dense Object Detection, ICCV, Aug 2017

boostcamp ai tech

NAVER CONNECT 부스트캠프 AI Tech

NLP KLUE 프로젝트

개인 회고

Member
김주원_T5056, 강민재_T5005, 김태민_T5067, 신혁준_T5119 윤상원_T5131

강민재_T5005

대회에서 주어진 데이터를 분석하고, 분석한 내용을 바탕으로 증강할 때, 지난 대회에서 사용한 기법들을 그대로 적용하기 어렵다고 느꼈다. 문장 유사도 분류 과제에서는 두 입력 문장에 대해서, 일관된 기준을 적용하여 전처리를 수행하여도, 문맥이 달라져도 유사도 레이블에는 큰 영향을 미치지 않는다. 하지만, 관계 추출 과제에서는 단일 문장 내에 두 개체와 주변 단어로부터 문맥을 이해해야 하므로, 데이터 전처리를 수행할 때 문맥이 조금이라도 훼손되지 않도록 더 정교한 기법이 수반돼야 한다고 생각했다. 따라서, 가능한 원본 문장의 형태는 최대한 보존하며 데이터 증강을 수행하려 하였다. 그래서 시도한 방법이, 문장 단순 복제와 클래스의 의미 관계를 활용하여 레이블만 변경하는 방식의 증강 기법이다. 그런데 모델 학습 단계에서 성능 향상을 끌어내지 못하였고, 실험 전에 세운 가설이 딥러닝 모델 관점에서는 유효하지 않음을 확인하였다.

이번 대회에서 데이터 분석 역량에 대한 한계를 느끼고 원인과 개선 방안을 파악해보았다. 태스크나 데이터 특성에 독립적인 일관적으로 효과적인 기법을 찾고자 했지만 실패했다. 대회에서 주어지는 데이터는 이미 한 단계 정제가 되어있는 상태이기 때문에 전처리나 증강을 통해 유의미한 효과를 보기가 어렵다고 생각했다. 문장의 품질이 좋지 못하다고 해도 그 품질이 일관적이라면 정제되지 않은 상태라고 생각하는 게 적절하지 않다고 느껴졌다.

또한 딥러닝 모델을 구현하여 주어진 문제를 해결하는 게 주요 목표이긴 하지만, 룰을 기반으로 한 알고리즘을 적절히 사용하여 모델의 문제 해결력을 향상하는 것도 필요하다고 느꼈다. 딥러닝 기법에 얽매여 다양한 관점에서 생각하지 못한 게 아쉽게 느껴졌다. 실제로 EDA 과정에서도 이를 적용할 수 있는 정보를 발견했음에도 시도해보지 못하였다.

per:parents

per:children

per:siblings

per:spouse

per:other_family

두 개체의 관계를 유추할 수 있는 문장은 클래스에 따라 자주 사용되는 어휘나 문장 구조가 존재할 것이라고 예상하였다. 이에 따라 클래스별로 데이터를 형태소, 명사, 토큰 기준으로 자주 등장하는 어휘를 확인하였다. 그림에서 확인할 수 있듯이, 클래스별로 연관된 어휘가 자주 등장한다. 예를 들어, 가족 관계와 관련된 클래스(per:siblings, per:other_family 등)에는 실제로 형, 동생, 아들과 같이 클래스와 연관된 어휘의 등장 빈도가 높다. 이 내용을 활용하여 딥러닝 모델의 예측 결과에 적절한 후처리를 했다면 더 좋은 성능을 내는 모델을 만들 수 있었을 것이다.

이를 바탕으로 다음 프로젝트에서는 정교한 EDA 를 바탕으로 주어진 데이터의 특징과 분포를 더욱 깊게 이해하고 명확한 문제 정의의 필요성을 느꼈다. 추가로. 데이터 분석의 목적을 잊지 않고, 새롭게 찾아낸 정보를 주의 깊게 관찰하며 인사이트를 발굴하는 역량을 키우기 위한 훈련이 이루어져야겠다는 교훈을 느꼈다.

👨🏻‍💻 김주원_T5056

팀내에서 나의 역할

팀 내에서 이번에 나는 팀장으로 모더레이터의 역할을 맡았다. 대회 초반에는 모더레이터로서 NLP 기초 프로젝트에 대한 wrap-up 을 진행하면서 팀원들로부터 지난 대회에 대한 팀적인 개선점에 대해서 파악하고, 이를 바탕으로 초기 세팅인 팀의 목표 설정, Ground Rule 을 설정하여 발생했던 문제점을 보완하기 위해 노력했다. 이외에도 대회 Github Readme, 노션, AI Stages 의 팀 등 다양한 팀 적인 초기 설정을 담당하고 진행했다. 이후에는 팀장으로 대부분의 회의를 진행했으며 팀원들의 작업 진행 여부를 파악하고 작업을 할당하는 등 모더레이터이자 전체적인 프로젝트 매니저 역할을 맡았다. 또 개인으로써는 데이터에 대한 EDA 를 맡고, Error Analysis 관련 모듈을 구현하여 팀원에게 활용할 수 있도록 배포했으며 soft voting/ hard voting/f1 weighted score 등 세 가지 종류의 앙상블 모델 코드를 구현 했으며, 최종적으로는 MLM 을 이용하여 데이터 증강을 하여 모델의 성능향상에 기여했다.

AI 개발자로 성장을 하기위해 시도했던 모험들(어떤 것을 도전했는가? 어떤 것들을 익혔는가?)

이번에 AI 개발자로서 시도했던 내용은 Error Analysis 에 깊이 파고들었다는 것, MLM 을 적용해서 데이터 증강을 진행했던 부분, 여러 가지 앙상블 기법을 모듈화하고 진행했던 부분, EDA 를 진행했던 부분이다. 특히, Error Analysis 를 진행하기 위해 Andrew Ng 의 Deep Learning Specialization 강의를 추가로 듣고 사전 조사를 진행한 부분을 바탕으로 dev 데이터에 대한 confusion matrix, f1-score 등을 파악하고 증강 방법을 계획했다. 또한 다양한 앙상블 방식을 찾아보고, 코드를 구글링하는 것이 아니라 이론적인 내용을 충분히 숙지하고, 직접 코드로 구현했다. 이렇게 ML /DL 기법에 대한 다양한 시도를 진행했고, 이를 통해 한 뼘 더 성장했다고 느낀다.

함께 일하기 위한 동료가 되기 위해 기울였던 노력들

이전 대회에서는 내 의견이 때로는 너무 강해 상대방의 말을 올바르게 이해를 못 하는 경우가 몇 번 있었다. 하지만, 이번에는 팀원들이 말하고 나서 요약하여 다시 한번 질문함으로써 상대방의 의도와 다르게 내가 이해했는지 체크하였다. 이 부분을 통해서 상대방과의 대화에서 입장차이를 빠르게 좁힐 수 있었고, 더 잘 소통할 수 있었다.

이전과 다른 노력들

앞서 언급한 Error Analysis 코드, 증강 관련 코드, 앙상블 관련 코드를 혼자서만 사용할 수 있도록 한 게 아니라 코드를 모두 모듈화하여 베이스라인 코드에서 누구나 코드를 활용할 수 있도록 구현하였다. 또한, 리팩토링을 진행하여 config parser 와 yaml 파일로 증강과 앙상블에 대한 관련 설정을 할 수 있도록 하여 코드의 유지보수성을 높였다. 즉 짧은 시간이었지만, 일반화된 좋은 코드를 작성하기 위해 노력했고, 그 결과 팀원 누구나 해당 코드를 활용하여 실험을 진행할 수 있었다.

부딪힌 한계, 개선해야할 부분

우선 팀장으로서 부딪힌 한계는 의견이 대립하는 구간에서 팀장으로서 더 나은 절충점을 제안하지 못하는 부분이 있다. 따라서 당황하여 팀원의 입장을 그대로 수용하거나, 아니면 내 의지대로 밀고 나가는 부분이 있는 것 같다. 즉, 최선의 절충점을 찾지 못했던 것이었는데 이런 일들을 줄여 나갈 수 있도록 당황하지 않고 여유를 가지고 말해야겠다는 생각이 들었다. 또한, 팀원들이 비록 노션에 기록하는 부분이나 함께 성장하는 부분 등 목표로 했던 부분은 일차적으로 달성했지만, 다 다른 생각을 하고 프로젝트가 진행된 것 같다는 생각이 들었다. 모두 하나의 방향성을 진행하는 것은 어렵지만, 적극적으로 실험 방향성에 관해서 이야기하는 시간을 만듦으로써 해당 문제를 해결해 나가는 팀적인 시간을 확보해야 할 것 같다.

두 번째로는 시간이 허락하지 못해 데이터 증강에 대한 실험을 많이 하지 못했던 부분이다. 중간에 K-Digital HACKATHON 에 참가하게 되어 두 번째 주에 관련된 내용에 대한 많은 회의를 진행하고 보고서를 팀원들과 작성하며, 2 주차에는 적극적으로 대회에 참여할 수 있는 spare time 을 만들 수 없었다. 따라서 시간 부족으로 error analysis 를 바탕으로 한 여러 가지 데이터 증강을

해보지 못한 부분이 아쉽다. 앞으로 효율적으로 시간 관리를 하기 위해 불필요한 프로세스(많은 paper work)를 없애고, 개발자 역량을 키울 수 있도록 더 적극적으로 대회에 참가할 수 있는 상황을 만들어야 할 것 같다.

👨🏻‍💻 김태민_T5067 캠퍼

1. 나는 내 학습 목표를 달성하기 위해 무엇을 어떻게 했는가?

이번 대회는 저번 대회와는 달리 각 파트를 집중적으로 나누어 진행하였다. 저번 대회는 모델 실험의 전체적인 진행 총괄을 맡았지만 그 대신 이번 대회에서 모델링의 기법(Low level)에 대해 집중적으로 탐색하고 파고들었다. 이번 대회에 내가 맡은 파트인 모델 성능 향상을 달성하기 위해 각종 논문과 지금까지 해온 모델링 기법으로 나만의 모델을 만들며 NLP task 에 대한 모델링에 대해 집중적으로 파고들 수 있었다.

2. 마주한 한계는 무엇이며, 아쉬웠던 점은 무엇인가?

개인적으로 모델로는 점수 향상이 있었지만 데이터에서의 유의미한 큰 점수 향상은 존재하지 않았다고 볼 수 있다. 모든 대회의 기본은 모델보다 데이터가 중요한 것은 알고 있었다. 또한 모델로는 성능을 올리는데 어느 정도 한계점이 분명하며 한국어는 더욱 그렇다. 데이터가 아쉬운 부분과 또 아쉬운 점으로는 모델링은 성공적이었지만 모든 모델을 학습에 사용할 때 통합된 환경을 갖추지 못한 것과 모델은 만들었어도 쉽게 불러다 쓰지 못하고 하드코딩으로 진행한 방식이 아쉬운 부분으로 남았다.

3. 한계/교훈을 바탕으로 다음 프로젝트에서 시도해보고 싶은 점은 무엇인가?

부스트 캠프의 슬로건인 함께 성장하자는 마인드와 같이 이번에는 모델링을 담당하였지만 다음 대회에는 데이터를 담당할 예정이다. 기존 다른 외부 대회에서 모든 부분을 혼자서 처리하였는데 점수의 직접적인 데이터 파트만 시도해본다면 데이터 분석의 실력이 향상 될 것이라고 생각된다.

4. 나는 어떤 방식으로 모델을 개선했는가?

기존의 논문에 나와 있는 모델들을 가장 먼저 구현하였다. RBERT 와 같은 모델들을 기반으로 백본 모델을 교체하거나 임베딩 레이어의 추가 등등 가장 확실하게 성능 향상이 입증된 부분부터 차례대로 구현하였다. 추후 커스터마이징하게 나만의 모델을 구성하였는데 처음 허깅페이스의 모델 레이어를 수정하여도 기존에 입증된 모델들을 먼저 구현하여 보다 쉽게 나만의 모델을 만들 수 있었다. 구현된 모든 모델은 위에 나와 있다.

5. 내가 해본 시도중 어떠한 실패를 경험 했는가? 실패의 과정에서 어떠한 교훈을 얻었는가?

모델링과 전처리의 시도 중에서 점수의 실패보단 순서의 실패가 가장 눈에 보였다. 논문에서 가장 높은 점수에 해당하는 전처리 방법이 있음에도 불구하고 모든 방법을 구현하고 2 번째로 높은 방법을 기준으로 추후 전처리 코드와 모델들을 연결한 점이 실수였다. 각 모델별로 수행해야 할 전처리 방법이 다른데 최종적으로 제일 높은 점수의 모델은 1 번째로 높은 전처리 방법에 기본 Base 모델을 강제적으로 사용할 수밖에 없었다. 2 번째 높은 방법과 기존 모델들을 연결해 만들었던 점이 1 번째 방법에 가장 높은 성능의 모델을 연결할 수 없었단 부분이 실패였다. 2 번쨰로 높은 방법을 기준으로 한 이유는 val score 를 보고 결정하였는데 각 seed 별로 다를 수 있는 점을 간과하였다. 추후 방식부터는 방법이 의미가 있는지 없는지에 대한 검증과 score 는 보다 엄격하게 여러 번을 수행하여 평균을 내야 한다고 경험하였다. 또한 다양한 모델들을 항상 언제 어디서든 통합적으로 관리해야 할 필요성을 많이 느꼈다.

6. 협업 과정에서 잘된점/아쉬웠던 점은 어떤것이 있는가?

협업은 저번 대회와 비교하여 많은 부분에서 원활하게 진행하였다. 우선으로 github 으로 코드를 관리하였으며 이를 통해 저번에는 손수 코드를 고쳐야 했지만 직접적인 github 코드 관리자를 두어 merge 등 보다 편하게 코드 관리를 실시할 수 있었으며 약 일주일에 한 번씩 코드 버전 컨트롤을 하여 리더보드 등수에 집중하기보다는 전체적인 코드의 통합성과 관리를 진행하였다는 점이 단순한 등수 높이는 것보다 많은 인사이트를 얻어갈 수 있었다. 또한 이번 협업에서는 각 파트를 명백하게 나누어 수행하였는데 비록 잘하는 사람이 모든 것을 관리하는 게 가장 높은 점수를 얻을 수 있겠지만 아직은 교육이다 보니 각 파트를 맡아서 각자가 스스로 생각하여 서로 많은 것을 배운다는 점에서 함께 성장한다는 슬로건에서 의미가 있었다고 생각된다. 아쉬운 점으로는 각 파트를 명백하게 나누었을 경우 한 사람은 데이터 한 사람은 모델만 배우는 단점이 있다. 하지만 다음 대회부터는 순환되게 진행하여 문제가 없을 것이라고 생각하지만 대신 리더보드의 등수는 많이 낮을 것이라고 생각된다.

👨🏻‍💻 신혁준_T5119

- 내가 시도한 기술적인 도전

EDA

저번 대회보다 자세하게 30 개의 Label 을 기준으로 데이터를 직접 탐색했다. 데이터 탐색을 통해 Label 이 가진 특성과 규칙을 찾고자 했다. 또한 찾아낸 규칙을 바탕으로 극단적인 Label Distribution 을 균등하게 조정하고자 했다.

konlpy 의 Okt 로 형태소 분석하여 토큰을 확인해 Label 의 단서를 확인했다. 'no_relation' Label 을 제외한 29 개의 Label 은 문장에서 Entity 사이의 관계를 알 수 있는 단어가 있었다. Word Cloud 라이브러리를 이용하여 탐색한 결과 개체명과 조사, 어미 등 빈도수가 높은 단어로 인해 Label 의 단서가 될 단어를 탐색하기 어려웠다. Dataset 을 Label 별로 나누고 외국어, 조사, 숫자, 개체명을 제외하여 토큰의 수를 확인했다.

특정 단어로 인해 편향이 발생할 수 있음을 확인했다. 이번 대회에서 새롭게 배운 점이다. Label 을 토큰 단위로 확인하니 label 에서 특정 단어가 의도하지 않게 Label 의 예측에 영향을 주고 있음을 관찰했다. 직접 문장을 보고 관찰했을 때와 토큰으로 나누어 확인했을 때 다른 장점이 있었다.

문장을 직접 관찰했을 때는 Label 에 영향을 줄 근거 단어를 중심으로 관찰했다. 자식과 부모의 관계의 'per:children'의 경우 '아들', '아버지' , '친모' 등의 토큰이 Label 의 토큰 사전에서 빈도가 높았다. 자식과 부모의 관계를 알 수 있는 단어로 문장의 Label 을 예측할 때 근거가 된다. **학습에서 Label 을 결정할 때 긍정적인 영향을 주는 단어로 판단할 수 있다.**

Label 별 토큰 사전으로 관찰했을 때는 의도하지 않게 특정 토큰이 Label 의 예측에 영향을 주고 있음을 관찰했다. 'per:date_of_birth'는 사람과 출생지의 관계이다. 문장에서 직접적으로 관계를 알 수 있는 토큰은 '태어났다.'(35 회), '출신'(18 회), 등이 있다. 그러나 'per:date_of_birth'의 토큰 사전에서 가장 높은 빈도는 '축구'(54 회), '선수'(49 회)이다. 축구 선수의 출생지 문장이 많았고 축구는 Label 을 'per:date_of_birth'로 판단하는 근거가 되지 않는다. 'per:date_of_birth'는 135 Dataset 으로 전체 32470 Dataset 중 0.4%에 해당한다. 따라서 Label 을 추론하는데 편향이 발생할 수 있음을 확인했다. 직접 Label 을 관찰했을 때와 비교하여 토큰 사전으로 관찰했을 때는 학습에 부정적인 영향을 줄 토큰을 확인할 수 있었다.

- 마주한 한계

Data Augmentation

데이터 증강 기법으로 동의어 교체, 동일 Entity start_idx, end_idx 교체 기법을 적용했다. 하지만 Raw Data 와 비교했을 때 Micro F1 Score 가 떨어졌다. 동의어 교체 기법으로 이전 대회에서 모델의 성능을 향상했다. 이번 대회에서 같은 기법을 Task 에 맞게 수정했지만 실패한 원인을 분석했다.

단순히 Label 분포를 균등하게 맞추려는 증강이라 실패했다고 생각한다. 초반 EDA 를 통해서 Label 분포가 특정 Label 에서 극단적으로 많거나, 적기 때문에 증강이 필요하다고 판단했다. 저번 대회에서 Label 분포를 적절히 맞추어 높은 성능의 모델을 만들었다. 이번 대회도 Label 분포를 균등하게 맞추려고 증강했다. 하지만 Dataset 이 너무 적은 Label 은 분포를 균등하게 맞출 정도로 증강이 어려웠고 단순히 양적인 Dataset 증강으로 이어졌다. 저번 대회에서 Data 증강의 효과가 Dataset 수가 증강해서인지, Label 분포가 균등해서인지 ablation study 가 되지 않았다고 판단했다. 이전 대회에서 증강 성공 원인을 정확히 파악하지 않고, 증강 기법만을 적용하여 증강이 실패했다고 생각한다.

증강할 Label 을 Confusion Matrix 를 통해서 구체적으로 파악해야 한다. Label 의 Dataset 이 적다고 무조건 에러가 높은 것은 아니다. Dev Data 의 Confusion Matrix 를 확인했을 때 Dataset 이 적어도 정답률이 높은 Label 이 있었다. 기계 입장에서 Label 을 구분하기 쉬운 문제라고 생각할 수 있다. 반면 모델과 데이터를 바꾸었을 때 Confusion Matrix 에서 계속 높은 오답률의 Label 을 확인했다. 구체적으로 Confusion Matrix 를 확인한 예로 'per:place_of_residence'와 'per:origin'을 비교할 수 있다. 'per:place_of_residence'는 주체의 거주지, 'per:origin' 주체의 소속 관계이다. 'per:origin'은 1234 Dataset, 'per:place_of_residence'는 193 Dataset 으로 분포에 차이가 있다. Confusion Matrix 에서 'per:place_of_residence'는 Dataset 이 더 많은 'per:origin'으로 틀리게 예측하고 있었다. 하지만 확인된 'per:place_of_residence'의 문제를 적절히

해결하지 않고 'per:place_of_residence' 와 'per:origin'를 모두 증강했다. 결론적으로 'per:place_of_residence' 대비 'per:origin'의 Dataset 이 많았기 때문에 'per:origin'에서 더 많은 증강 데이터가 만들어졌고 'per:origin'으로 예측하는 'per:place_of_residence'가 늘어나 Micro F1 Score 가 하락했다. 리더보드 1 등 팀의 발표를 보니 같은 문제점을 확인했고 'per:place_of_residence'만 증강하여 'per:origin'으로 예측하는 오답이 줄었다고 발표했다. 데이터를 증강할 때 각 Label 의 Dataset 이 학습 후 모델에서 어떤 영향을 미칠지 예측하고 Confusion Matrix 로 파악해야겠다.

- 학습과정에서의 교훈

이번 대회에서 NLP 에서 Data 의 목적이 무엇인지 생각했다. Data 를 수집하고 전처리, 증강하는 이유는 단순히 어법에 맞게 문장을 고치고 Unknown 토큰을 줄이기 위함, Label 의 분포를 균등하게 맞춤이 아닌 듯하다. Data 는 Input 으로 들어가 모델에서 학습을 거쳐 Output 으로 나온다. **전처리와 증강은 Input Data 가 계획한 Output 으로 나오도록 설계하는 초기 과정임을 깨달았다.** 단순히 Dataset 의 수와 형태에 집중하지 않고, Output 이 어떤 형태일지 고려하며 전처리와 증강해야겠다.

👨🏻‍💻 윤상원 T5131

Code Management

베이스라인 코드 리팩토링

대회에서 제공된 베이스라인 코드를 전체적으로 수정하는 작업을 진행했다.

먼저 HuggingFace Trainer 로 모델을 학습시키는 코드를 Pytorch Lightning 을 사용하여 모델을 학습시키도록 수정했다. 지난 대회에서 모델을 직접 커스터마이징하지 못했던 가장 큰 이유가 HuggingFace Trainer 를 사용했기 때문이었고, 따라서 이번 대회에서는 모델링 작업을 진행하기 위해서 Pytorch Lightning 을 적용했다.

기존 베이스라인 코드에 존재하던 버그들도 수정했다. 버그 중 하나를 예로 들자면 subject 와 object 를 파싱하는 부분에서 딕셔너리 형태의 값을 문자열로 읽어와 ‘,’와 ‘:’를 구분자로 하여 객체를 추출하는 코드가 있었다. 이 코드의 경우, 객체 안에 ‘,’ 또는 ‘:’가 포함되어 있으면, 의도치 않은 방식으로 문자열이 분리되어서 객체를 잘못된 값으로 읽어오는 경우가 존재했다. 따라서 이 부분을 eval 함수로 딕셔너리 형태의 문자열을 딕셔너리로 치환하여 key 를 통해 객체의 값을 올바르게 가져왔다.

```
# 기존 베이스라인 코드
for i,j in zip(dataset['subject_entity'], dataset['object_entity']):
    i = i[1:-1].split(',')[0].split(':')[1]
    j = j[1:-1].split(',')[0].split(':')[1]

    subject_entity.append(i)
    object_entity.append(j)
```

```
# 수정된 베이스라인 코드
for i,j in zip(dataset['subject_entity'], dataset['object_entity']):
    sub_dict, obj_dict = eval(i), eval(j)
    subject = sub_dict["word"]
    object = obj_dict["word"]

    subject_entity.append(subject)
    object_entity.append(object)
```

지난 대회에서 실험 환경의 config 를 hard coding 으로 수정할 때의 번거로움과 human error 를 경험했기 때문에 이번 대회에서는 argparse 와 omegaconf 모듈을 활용해 하나의 yaml 파일로 전체 실험의 config 를 제어할 수 있도록 구현했다. 또한, yaml 파일을 공유하는 것만으로도 간단하게 실험을 재현할 수 있고, 실험의 config 를 기록으로 남길 수 있다는 장점도 얻을 수 있었다.

```
def config_parser():
    """argparse로 yaml 파일의 경로를 받아온 뒤, omegaconf로 해당 yaml 파일에 정의된 config 값을 읽어와서 반환하는 함수"""
    parser = argparse.ArgumentParser()
    parser.add_argument('--config', type = str, default = 'config/default.yaml')
    args = parser.parse_args()

    config = omegaconf.OmegaConf.load(args.config)
    return config
```

베이스라인 코드는 뼈대가 되는 코드로서, 다른 팀원들이 이 코드에서 확장하여 구현하기 쉽도록 베이스라인 코드를 구현했다. 그 예로, 토크나이저에 special token 을 추가할 수 있도록 구성하고, 그에 따라 모델의 embedding layer 의 입력 차원의 크기를 토크나이저의 크기에 맞게 resize 하도록 구현하여 special token 이 추가되더라도 모델이 에러 없이 동작할 수 있게 했다.

main 코드 관리

팀원 개개인이 실험한 내용들을 쌓아 올리기 위해서는 main 코드에 팀원들이 구현한 내용들이 잘 반영되어야 한다는 것을 지난 대회를 통해 깨달았기 때문에 이번 대회에서는 main 코드 관리를 신경 써서 진행했다. 또한, 팀원이 구현한 내용을 Pull Request 하면 해당 내용을 검수하고, 다른 팀원들이 손쉽게 사용할 수 있도록 코드의 컨벤션을 따르도록 수정하거나, 함수에 주석을 달고, 비슷한 기능을 하는 함수와 클래스끼리 모듈화하는 등의 리팩토링을 진행했다.

Modeling :TAPT

Don't Stop Pretraining: Adapt Language Models to Domains and Tasks 논문[1]에서 제안한 TAPT(Task Adaptive Pretraining)은 기존의 대량의 말뭉치로 사전 학습하여 일반화 성능을 끌어올린 사전 학습 모델에 task 에 특화된 데이터만을 가지고 추가로 MLM(Masked Language Modeling)을 진행하는 것이라고 논문에서 정의하고 있다. 모델에 TAPT 를 적용하면 해당 task 에 등장하는 문장들에 대한 이해도가 높아진다는 것이 논문에서 주장하는 내용이다. 따라서 우리 모델에도 train 과 test

데이터 셋을 활용해 TAPT 를 적용하면 성능이 향상될 것이라고 가설을 세우고 실험을 진행했다. KLUE 데이터 셋으로 사전 학습된 BERT 와 RoBERTa 모델에 앞서 언급한 대로 대회에서 제공한 train 과 test 데이터 셋을 바탕으로 Mask Language Modeling 을 진행했다. 이후, ablation study 를 위해 다른 조건들을 동일하게 고정한 뒤, TAPT 를 적용한 모델과 적용하지 않은 모델의 성능을 비교해 보았다.

아래 표는 BERT 모델, RoBERTa 모델에서 TAPT 적용 전후의 public score 를 비교한 결과이다. micro f1 score 기준 각각 1.3 점, 1.6 점 가량 상승한 것을 확인할 수 있다.

	TAPT 적용 전		TAPT 적용 후	
평가 metric	Public f1-score	Public AUPRC	Public f1-score	Public AUPRC
klue/bert-base	64.0741	66.4684	65.3753(+1.3)	67.3594
klue/roberta-large	67.1473	71.3321	68.7156(+1.6)	67.1473

두 모델 중 성능이 더 잘 나온 TAPT 를 적용한 RoBERTa 모델의 경우 다른 팀원들이 사용하기 쉽도록 HuggingFace Hub 에 모델을 업로드했다.

지난 대회에 비해 개선된 점

모델 측면
이번 대회에서는 Pytorch Lightning 을 사용하면서, HuggingFace Trainer 를 사용했던 이전 대회에 비해 쉽게 모델을 직접 수정하고, 다양한 learning rate scheduler, loss function, optimizer 를 적용할 수 있었다. 이러한 경험을 통해 모델의 구조와 동작 방식에 대해 더 깊게 이해할 수 있었고, learning rate scheduler, loss function, optimizer 가 모델에 미치는 영향을 더 직관적으로 살펴볼 수 있었다.

코드 관리 측면
main 코드 관리를 진행함으로써 팀원들 각자 구현한 코드를 공유하며 효율적으로 실험 진행될 수 있었다.

실험 진행 측면
지난 대회에서는 한 번의 실험에서 여러 기법을 동시에 적용하여 ablation study 가 제대로 이뤄지지 못했는데, 이번 대회에서는 ablation study 가 이뤄질 수 있도록 한 번의 실험에서는 하나의 조건만 조작하여 해당 조건이 모델의 성능에 미치는 영향을 파악할 수 있었다.

다음 대회에서 보완할 점

실험 진행 측면
이번 대회에서 코드 관리 파트를 담당하게 되어 대회 초반에는 데이터에 대한 분석이나 모델 관련된 실험을 진행하지 못했다. 그로 인해, 대회 막바지에 급하게 실험을 시도하다 보니 데이터와 평가 지표, 모델에 대한 분석을 깊게 하지 않고, 관계 추출 task 에 적용할 수 있는 기법들을 마구잡이로 시도했던 것 같다. 다음 대회부터는 실험을 한두 개 덜 하더라도 평가 지표, 주어진 데이터와 현재 사용하는 모델들의 분석을 통해 현시점의 문제를 정의하고, 관련된 선행 연구를 조사하여 실험에 적용하는 방식으로 진행할 것이다.

[1] *Suchin Gururangan, Ana Marasović, Swabha Swayamdipta, Kyle Lo, Iz Beltagy, Doug Downey, Noah A. Smith. Don't Stop Pretraining: Adapt Language Models to Domains and Tasks. ACL, April 2020.*