Wrap UP 리포트

NLP 2조 퍼스트펭귄

1. 프로젝트 개요

- 프로젝트 주제
  - 관계 추출(Relation Extraction) task로, 문장 속에서 단어(Entity)에 대한 속성과 관계를 예측하는 문제입니다.
- 프로젝트 개요(프로젝트 구현 내용, 컨셉, 교육 내용과의 관련성 등)
  - 문장, 단어에 대한 정보를 통해 ,문장 속에서 단어 사이의 관계를 추론하는 모델을 학습시킵니다.
  - 입력으로 들어온 *sentence*, *subject_entity, object_entity* 정보에 대해 두 *entity*의 relation을 30개 label 중 하나로 예측합니다.
- 활용 장비 및 재료(개발 환경, 협업 tool 등)
  - 대회에서 제공한 v100 GPU 서버에서 프로젝트의 모든 실험을 진행했습니다.
  - 팀 notion을 활용해 프로젝트 계획과 진행상황을 공유했고, 팀 wandb 연결을 통해 실험 결과를 공유했습니다.
- 프로젝트 구조 및 사용 데이터셋의 구조도(연관도)
  - 프로젝트 구조

```
|-- code
|   |-- __pycache__
|   |   `-- load_data.cpython-38.pyc
|   |-- best_model
|   |-- dict_label_to_num.pkl
|   |-- dict_num_to_label.pkl
|   |-- inference.py
|   |-- load_data.py
|   |-- logs
|   |-- prediction
|   |   `-- sample_submission.csv
|   |-- requirements.txt
|   |-- results
|   `-- train.py
|-- dataset
    |-- test
    |   `-- test_data.csv
```

```
    `-- train
        `-- train.csv
```

        - klue RE dataset을 이용해 Entity간의 관계추출(RE) F1 score 향상
    - 데이터 셋의 구조

```
├─id : 각 data의 고유 id
├─sentence : 각 entity가 포함된 전체 문장
├─subject_entity : {단어, 시작 인덱스, 끝 인덱스, 타입} 정보
├─object_entity : {단어, 시작 인덱스, 끝 인덱스, 타입} 정보
├─label : 두 entity의 관계 분류 (sub:obj)
└─source : 데이터의 출처
```

        - subject_entity와 object_entity의 관계를 기준으로 30종류의 label을 가지고 있는 문장들
- 기대 효과
    - 모델의 구조적 정보에 대한 예측 수행 능력 향상을 이룰 수 있습니다. 이 과정에서 수행하는 관계 추출은 지식 그래프 구축을 위한 핵심 구성 요소로, 구조화된 검색, 감정 분석, 질문 답변하기, 요약과 같은 다양한 자연어처리 응용 프로그램에서 요구됩니다.

2. 프로젝트 팀 구성 및 역할

- 고우진
  - EDA, 오답분석, RECENT 논문 구현, Data Augmentation, Loss function, Tokenizer
- 현승엽
  - 오답 분석, RECENT 논문 구현, Data Augmentation, Ensemble
- 이종윤
  - EDA, 오답 분석, Entity의 type이나 위치를 알려주는 Entity token 추가
- 김상윤
  - EDA, Entity special token 추가, entity embedding layer 추가 구현

3. 프로젝트 수행 절차, 방법 및 결과

EDA

- 우선 데이터의 분포나 결측치를 확인하여 데이터에 문제가 없는지 확인 했습니다.
- **[UNK] token**의 발생 분포와 원인을 파악해 보았습니다.
- **오답 데이터의 label 분포**를 확인해 보았습니다.
- 오답 데이터 중 label 마다 **no_relation으로 예측한 분포**를 확인해 보았습니다.
    - no_relation의 수가 많기 때문에 전반적으로 no_relation 이라고 예측할 모형이 학습될 경향을 보인다고 판단했습니다.

Tokenizer

> 💡 bert tokenizer, roberta tokenizer 등으로 주어진 train data를 tokenize 하는 경우 [UNK] 토큰이 많이 발생하였습니다.. 성능이 안나오는 이유가 [UNK] 토큰의 빈번한 출현으로 인한 정보손실일 수도 있다고 생각하여 **[UNK] 토큰의 출현을 줄일 수 있는 다양한 방안**을 아래와 같이 시도해 보았습니다.

1. subject와 object가 [UNK]로 토큰화되는 경우, **통으로** 해당 subject or object를 사전에 하나의 **단어(token)로 추가**해 주었습니다.
   :(리더보드,
2. word piece tokenizer의 경우에는 한 글자 때문에 [UNK]로 토큰화되는 경우가 빈번합니다. 따라서 이러한 경우를 방지하기 위해 주어진 sentence들을 모두 **음절단위로 토큰화**하고(ex> 안녕하세요 → 안, ##녕, ##하, ##세, ##요) **해당 음절들을 기존에 사전에 모두 추가**해주었습니다. 이때 기존에 사전에 없는 단어들만 추가되기 때문에 중복의 문제는 걱정할 필요 없습니다.
3. 2의 경우 새롭게 추가된 단어들 중 한자의 수가 많았습니다. 새롭게 한자를 사전에 추가해 주면 대응하는 embedding vector는 랜덤한 초기값을 갖게 되어 정보를 크게 얻을 수 없습니다. 만약 한자를 한글로 번역해 집어 넣는 다면 [UNK]로 토큰화 되는 경우의 수를 줄일수 있고, 더욱 많은 정보를 얻을 수 있다고 생각해 한자를 한글로 번역해 주었습니다. **“hanja” 모듈을 사용하여 한자를 한글로 번역**해 주었습니다
    - [UNK] token이 전체 data의 10%에서 발생했고, 이 중 1/3은 한자에서 발생했습니다.
      → 교체 후 entity 간 구분 문제를 해결하기 위해 entity special token추가도 함께 수행했습니다.
4. 3과 동일하게 한자를 한글로 번역해 주었습니다. 여기에 더하여, 만약 2의 방법대로 진행시 일본어, 그리스어, 아랍어, 의미없는 형상문자 등이 모두 개별 토큰으로 인식되게 됩니다. 이러한 토큰에 대응하는 embedding vector는 역시 랜덤한 초기값을 갖기 때문

에 크게 얻을 수 있는 정보가 없다고 생각하였습니다. 오히려 큰 의미가 없는 위와같은 토큰들은 차라리 [UNK]으로 일괄처리하면 좋을 수 있다는 생각을 하였습니다. 따라서 2의 경우처럼 주어진 sentence들을 모두 음절단위로 토큰화하고 모든 음절을 기존 사전에 추가하는 것이 아니라, **음절들 중 한글에 해당하는 것만 사전에 추가**해 주었습니다.

Data

- **Augmentation**

1. Bert가 syntactic한 부분을 잘 학습한다는 점에 착안하여, 동일한 pair-type entity를 가지는 문장들간의 **subject와 object를 바꿈**으로써 데이터 증강을 시도해봤습니다.
2. 데이터가 불균형하다는 점에 착안하여 데이터가 적은 label의 경우 **단순히 복사**하여 그 수를 비슷하게 맞추는 **oversampling**을 시도해봤습니다.
    a. 데이터 수가 가장 많고, 적은 레이블의 경우 각각 약 9500, 40개였는데 각 label 데이터가 최소 2000개가 되도록 복사하여 데이터를 증가시켰습니다.
3. 데이터가 불균형하다는 점에 착안하여 데이터가 많은 label의 경우 그 수를 줄여 나머지 레이블과 비슷하게 맞춰주는 **undersampling**을 시도해봤습니다.
    a. 가장 많은 no_relation 레이블에서 절반을 랜덤하게 선택하여 그 수를 반으로 줄인 데이터로 모형을 학습했다.

- **Entity Token**

: entity의 위치와 type 정보를 명시하는 special_token을 추가

1. 일반 token으로 추가
    a. Entity 앞뒤로 type_token(ex.[PER],[/PER],[LOC],[/LOC])을 추가해준 경우
    b. Entity가 앞에 붙은 Sentence 내부에만 Entity 앞뒤로 type_token을 추가해준 경우
    c. Entity를 앞에 붙이지 않고 Sentence 내부에만 Entity 앞뒤로 type_token을 추가해준 경우
2. subject, object entity만 구분하도록 [sub], [/sub], [obj], [/obj] 네 개의 special token 추가
3. sub, obj와 더불어 각 entity의 type을 구분하도록 [sub_PER], [/sub_PER], [obj_ORG], … 등 16개의 special token 추가

    → klue/roberta-large에 적용 시, [sub/obj _ EntityType] 형태의 16개의 special 토큰을 추가 했을 때 성능 향상이 가장 좋게 나타났습니다.

: 리더보드 기준, 단일 모델에서 63점 → 69.27점 도달. (한자 번역 tokenizer 및 hinge loss 적용)

- input dataset에 entity token을 0과 1로 구분지어 명시한 **entity_ids data를 추가**했습니다. (entity embedding 추가 시 활용)

Model

- **Loss function**
  - focal-loss
    - 데이터가 불균형하다는 점에 착안하여 focal loss를 사용해 보았습니다.
  - weighted cross entropy
    - 데이터가 불균형하다는 점에 착안하여 weighted cross entropy loss를 사용해 보았습니다.
  - **hinge loss**
    - cross entropy loss의 경우 올바르게 맞춘 경우에도 loss가 발생하는데에 반해, hinge loss(svm)의 경우에는 정답을 맞추지 못하는 경우에만 loss가 발생했습니다. 이 점에 착안하여 hinge loss를 사용해 보았습니다.
- **RE task 관련 논문(RECENT) 구현**
  - 기존 task는 한번에 30개의 label에 대한 분류를 진행했습니다. RECENT에서는 좀 더 효과적인 분류를 진행하기 위해 (1) **no_relation vs relation 으로 1차적으로 이진분류**를 진행한 후, (2) relation으로 예측된 경우 **pair-type entity를 기준으로 데이터를 분류**시키고, 각각의 경우 **해당하는 label set에 대해 분류를 진행**합니다. 데이터를 pair-type entity 별로 나눴을 때, 각 그룹이 갖는 label 수는 8~19개로 기존의 30개에 비해 감소하기 때문에 분류 난이도가 낮아진다는 장점이 있다고 판단했습니다.

    → 예측 성능은 val_f1 : 50.17 로 매우 안좋게 나왔다. 그 이유는 no_relation과 나머지 label을 학습시키는 각각의 분류기의 성능이 좋지 않은 점이 가장 큰 것이고, 각 pair-type 그룹 별 데이터 분포를 고려하지 못하고 학습시킨 것도 원인이라고 생각했습니다.
- **Entity embedding layer 추가**

  → 예측 성능 오히려 하락했습니다(val_f1 기준, 82.489 → 71.116). 구현의 완성도가 떨어졌던 것으로 예상됩니다.
- **Hyper parameter tuning**

- early-stopping
  - 평가 기준이 되는 f1을 evaluation metric 기준으로 설정하여, evaluation마다 상위 5개의 check_point를 저장해나갔고 학습 후 가장 성능이 좋은 모델을 저장했습니다.
- Mixed Precision(fp16) 적용으로 더 큰 batch_size를 적용할 수 있었습니다.
- wandb_sweep을 활용하여 다양한 learning_rate를 적용해 보았습니다.
- klue/roberta-large에서 배치사이즈가 클수록 좋은 성능을 보였습니다.
  (2→4→8→16→32→64)

- **Ensemble**
  - soft-voting 방식을 이용하여 성능이 높은 여러 모델들을 앙상블 방식으로 개선하였습니다.

→ 최종 리더보드 기준 F1 score : 72.3460점 획득했습니다.

4. 프로젝트 수행 결과

성능 분석

Aa Model	# 설명	Loss Function	Batch Size	eval_f1	# f1 score(리더보드)
klue/roberta-base		cross-entropy	16	81.867	
klue/roberta-base		cross-entropy	32	82.233	
klue/roberta-base	lr = 1e-5, 일반 토큰 추가 X	cross-entropy	64		58.9329
klue/roberta-base	lr = 1e-4, 일반 토큰 추가 X	cross-entropy	64		64.4331
klue/roberta-base	lr = 5e-5, 일반 토큰 추가 X	cross-entropy	64	84.057	65.9225

Aa Model	# 설명	Loss Function	Batch Size	eval_f1	# f1 score(리더보드)
klue/roberta-base	lr = 5e-5, 일반 토큰 추가 (a)	cross-entropy	64		65.5374
klue/roberta-base	lr = 5e-5, 일반 토큰 추가 (b)	cross-entropy	64		64.3095
klue/roberta-base	lr = 5e-5, 일반 토큰 추가 (c)	cross-entropy	64		57.379
klue/roberta-base	lr = 5e-5, 일반 토큰 추가 X	cross-entropy	96	83.635	64.8023
klue/roberta-base		focal loss	64	83.745	
klue/roberta-base		weighted cross-entropy	64	82.794	
klue/roberta-large		cross-entropy	32	83.972	
klue/roberta-base	RECENT	cross-entropy	64		50.172
klue/roberta-large	Tokenizer3 & EntityToken3	hinge_loss	48		69.237
제목 없음					
klue/bert-base	Tokenizer2	cross-entropy	32	82.604	
klue/bert-base	Tokenizer2	cross-entropy	64	83.14	
klue/bert-base	Tokenizer2	hinge_loss	16	83.505	
klue/bert-base	Tokenizer2	hinge_loss	32	83.676	
klue/bert-base	Tokenizer2	hinge_loss	64	83.762	
klue/bert-base	Tokenizer3	cross-entropy	64		62.247
klue/bert-base	Tokenizer4	cross-entropy	64		66.339
klue/bert-base	Tokenizer4	cross-entropy	128		65.45
klue/bert-base	Tokenizer4	cross-entropy	160		65.332

Aa Model	# 설명	Loss Function	Batch Size	eval_f1	# f1 score(리더보드)
klue/bert-base	Entity embedding layer	hinge_loss	16	71.116	
klue/bert-base	Tokenizer3 & EntityToken3	hinge_loss	16	83.78	
제목 없음					
beomi/kcbert-large	lr = 5e-5, 일반 토큰 추가 X	cross-entropy	32		52.9752
beomi/KcELECTRA-base-v2022	lr = 5e-5, 일반 토큰 추가 X	cross-entropy	64	80.004	56.7836
l-yohai/klue-roberta-base-finetuned-ner	lr = 5e-5, 일반 토큰 추가 X	cross-entropy	64	82.661	65.1861
bert-base-multilingual-cased		cross-entropy	32	81.963	
bert-base-multilingual-cased		cross-entropy	64	81.807	
monologg/koelectra-base-v3-discriminator		cross-entropy	64	82.097	
kykim/funnel-kor-base		cross-entropy	32	82.649	
kykim/funnel-kor-base		cross-entropy	64	43.752	

5. 자체 평가 의견

- 잘했던 부분
  - 팀원 모두가 여러가지 시도를 많이 해본 점
  - 코드 이해, 구현의 부분에서 많이 성장했다.
- 아쉬웠던 부분
  - 코드 공유/관리/리뷰의 과정이 제대로 이루어지지 않은 점

→ 관련 내용 미리 공부하여 다음 대회 때 개선하기

- 초반에 EDA를 충실하게 하지 못한 점

개인 회고

고우진

나는 내 학습목표를 달성하기 위해 무엇을 어떻게 했는가?

- 우리 팀과 나의 학습목표는 무엇이었나?
  - 우리 팀의 목표는 다양한 방법론(EDA, 전처리, 모델링 등)을 이용하여 모델의 성능 향상을 이끌어 내는 것이었고, 그 중에서도 나의 역할은 적절한 모델링 및 전처리를 통해 팀의 목표를 달성하는 것이었다.
- 개인 학습 측면
  - Huggingface의 transformers 라이브러리가 제공하는 다양한 모듈의 소스코드를 파헤쳐가며 그 사용법과 구체적인 내부 구조를 파악하였다.
  - RE task에 적합한 다양한 모델 및 관련된 논문들을 찾고 공부하였다.
  - 다양한 모델링을 진행하였다.
- 공동 학습 측면
  - 적합한 모델과 방법론을 찾고 공부한 후에 조원들과 공유하였다.
  - 조원들이 이해가 되지 않는 부분에 대해 자세히 설명해 주었다.

나는 어떤 방식으로 모델을 개선했는가?

- 사용한 지식과 기술

1. **Data augmentation**
   a. Bert가 syntatic한 부분을 잘 학습한다는 점에 착안하여, 동일한 pair-type entity를 가지는 문장들간의 subject와 object를 바꿈으로써 데이터 증강을 시

도하였다

b. 데이터가 불균형하다는 점에 착안하여 데이터가 적은 label의 경우 단순히 복사하여 그 수를 비슷하게 맞추는 oversampling을 시도하였다

c. 데이터가 불균형하다는 점에 착안하여 데이터가 많은 label의 경우 그 수를 줄여 나머지 레이블과 비슷하게 맞춰주는 undersampling을 시도하였다

2. **RE task 관련 논문(RECENT)참고**

a. 현재의 task는 한번에 30개의 label에 대한 분류를 진행하고 있다. RECENT에서는 좀 더 효과적인 분류를 진행하기 위해 (1) no_relation vs relation 으로 1차적으로 이진분류를 진행한 후, (2) relation으로 예측된 경우 pair-type entity를 기준으로 데이터를 분류시키고, 각각의 경우 해당하는 label set에 대해 분류를 진행한다.

3. **다양한 loss function 사용**

a. 데이터가 불균형하다는 점에 착안하여 weighted cross entropy loss를 사용해 보았다.

b. 데이터가 불균형하다는 점에 착안하여 focal loss를 사용해 보았다.

c. cross entropy loss의 경우 올바르게 맞춘 경우에도 loss가 발생하는데에 반해, hinge loss(svm)의 경우에는 정답을 맞추지 못하는 경우에만 loss가 발생한다. 이 점에 착안하여 hinge loss를 사용해 보았다.

4. **Tokenizer**

bert tokenizer, roberta tokenizer 등으로 주어진 train data를 tokenize 하는 경우 [UNK] 토큰이 많이 발생하였다. 성능이 안나오는 이유가 [UNK] 토큰의 빈번한 출현으로 인한 정보손실일 수도 있다고 생각하여 [UNK] 토큰의 출현을 줄일 수 있는 다양한 방안을 시도해 보았다

a. subject와 object가 [UNK]로 토큰화되는 경우, 통으로 해당 subject or object를 사전에 하나의 단어로 추가해 주었다.

b. word piece tokenizer의 경우에는 한 글자 때문에 [UNK]로 토큰화되는 경우가 빈번하다. 따라서 이러한 경우를 방지하기 위해 주어진 sentence들을 모두 음절단위로 토큰화하고(ex> 안녕하세요 → 안, ##녕, ##하, ##세, ##요) 해당 음절들을 기존에 사전에 모두 추가해주었다. 이때 기존에 사전에 없는 단어들만 추가되기 때문에 중복의 문제는 걱정할 필요 없다.(bert, roberta의 경우 ~~~~~~개가 추가 되었다)

c. b의 경우 새롭게 추가된 단어들 중 한자의 수가 많았다. 새롭게 한자를 사전에 추가해 주면 대응하는 embedding vector는 랜덤한 초기값을 갖게 되어 정보를 크게 얻을 수 없다. 만약 한자를 한글로 번역해 집어 넣는 다면 [UNK]로 토큰화

되는 수를 줄일수 있고, 더욱 많은 정보를 얻을 수 있다고 생각해 한자를 한글로 번역해 주었다. “hanja” 모듈을 사용하여 한자를 한글로 번역해 주었다

d. c와 동일하게 한자를 한글로 번역해 주었다. 여기에 더하여, 만약 b의 방법대로 진행시 일본어, 그리스어, 아랍어, 의미없는 형상문자 등이 모두 개별 토큰으로 인식되게 된다. 이러한 토큰에 대응하는 embedding vector는 역시 랜덤한 초기값을 갖기 때문에 크게 얻을 수 있는 정보가 없다고 생각하였다. 오히려 큰 의미가 없는 위와같은 토큰들은 차라리 [UNK]으로 일괄처리하면 좋을 수 있다는 생각을 하였다. 따라서 b의 경우처럼 주어진 sentence들을 모두 음절단위로 토큰화하고 모든 음절을 기존 사전에 추가하는 것이 아니라, 음절들 중 한글에 해당하는 것만 사전에 추가해 주었다.

5. **Entity marker 추가 및 input 형태 변경**
   기존의 input의 형태는 subject + [sep] + object + [sep] +sentence 이다

   a. <sub:sub_type>,</sub:sub_type>,<obj:obj_type>,</obj:obj_type>에 해당하는 24개의 special token을 사전에 추가하였고 input의 형태를 다음과 같이 바꿔 주었다.
   → sentence일부 + <sub:sub_type> + subject + </sub:sub_type> + sentence일부 + <obj:obj_type> + object + </obj:obj_type> + sentence일부

6. **Ensemble**

   a. soft-voting 방식을 이용하여 성능이 높은 여러 모델들을 앙상블 방식으로 개선하였다.

전과 비교해서, 내가 새롭게 시도한 변화는 무엇이고, 어떤 효과가 있었는가?

- 위에서 언급했듯이 다양한 방법론을 시도해보았지만, 앙상블을 제외하고는 뚜렷한 성능 향상으로 이어지지는 못하였다.

마주한 한계는 무엇이며, 아쉬웠던 점은 무엇인가?

- 팀원들과의 체계적인 소통과 코드 공유 부족으로, 진행 과정에서 비효율적인 시간들이 많이 발생하였다
- 코드관리가 효과적으로 수행되지 못하였다
- hyperparameter tuning을 중구난방식으로 하여 효과적이지 못한 것 같다
- 다양한 실험을 진행하였지만 모든 내용을 구체적으로 기록하지 못해 실험이 효과적으로 진행되지 못하였다.

한계/교훈을 바탕으로 다음 프로젝트에서 스스로 새롭게 시도해볼 것은 무엇일까?

- 효과적인 hyperparameter tuning을 진행해 보겠다.
- 다양한 실험내용 및 결과를 구체적으로 기록해야겠다.

현승엽

나는 내 학습목표를 달성하기 위해 무엇을 어떻게 했는가?

- 이전 대회에서 아쉬웠던 점이 생각만 하고, 공부만 하고 뭔가 시도해 보지 못한 것이었다. 그래서 이번에는 단 한 가지라도 무언가 구현해 보고 시도해 보자는 것이 개인적인 작은 목표였는데 이번에는 생각했던 방법들을 실제 사용해볼 수 있었다.
    - 논문을 통해 RECENT라는 방법을 구현해서 적용

나는 어떤 방식으로 모델을 개선했는가?

- (모델 개선은 실패, 시도한 내용 작성)
- Data Augmentation

  같은 label에 해당하는 dataset들의 (subject entity, object entity)쌍을 교체해주는 방법으로 data augmentation을 적용시켜봤다. 구조적으로 문장안에 두 단어의 관계를 정의해주는 특정 단어, 문구가 있을 것이라고 생각해서 이 방법을 시도해봤다.

  → 성능에 큰 변화 없었음. 문장에서의 sementic한 패턴을 잡아내는 것엔 위 방법이 오히려 방해가 됐고, 각 entity이외의 다른 단어들은 동일한게 너무 많기 때문에 성공하지 못한 방법이라고 생각한다.

- Baseline모델에서 confusion matrix를 통해 오답의 분포를 확인하니, 모형이 no_relation으로 예측했는데 틀린 경우가 많아서 이를 먼저 분류해주기 위해, no_relation vs else로 분류한 후 나머지 29개를 분류하면 더 좋은 성능을 보일 것이라 예상했다.데이터 수가 많기 때문에 모델이 no_relation으로 분류하는 경향이 많아졌기 때문이다.

  → 성능이 오히려 줄어들었다. binary classifier의 학습이 부족했고 29개 레이블에 대한 분류기도 잘 되지 않았기 때문에

- 데이터 불균형이 문제라고 생각하여 over sampling / under sampling을 적용하여 baseline 모형을 학습했지만 성능 향상이 있지 않았다.
    - under sampling : ‘no_relation’의 수를 반으로 잘라내고 학습. no_relation으로 예측했다가 틀리는 경우는 줄었지만, no_relation을 맞추는 능력도 같이 줄어들었다. 특정 레이블에선 조금 더 잘 맞추는 경향을 보였지만 어떤 레이블의 경우엔 모형이 예측하는 분포가 너무 골고루 퍼지게 되는 경우도 발생했다.
    - over sampling : 모든 label 수를 2000이상이 되도록 만들었다. 단순 복제하는 방법을 사용. 다른 over sampling 방식을 사용해보지 않은 것이 아쉽다.
- RECENT 방법 구현

  subject/object entity type별로 나눈 뒤, 각 type별로 레이블을 분류시키는 방법. 총 12개의 타입으로 데이터를 나누게되면, 기존에 30개 레이블을 예측했어야하는 것과 다르게 타입별로 9~15개의 타입을 분류하면 되므로 좀 더 쉬운 task로 만드는 방법.

  → 성능이 좋지 않았다. 타입별 데이터 수가 많지도 않고, 특히 no_relation이 많은 타입의 경우 분류가 어려웠던 것 같다.

전과 비교해서, 내가 새롭게 시도한 변화는 무엇이고, 어떤 효과가 있었는가?

- 대회 마지막에 앙상블코드를 구현하여 다양한 조합을 찾아 팀 리더보드에서 가장 높은 점수를 받을 수 있었습니다.

마주한 한계는 무엇이며, 아쉬웠던 점은 무엇인가?

- 가지고 있는 train 데이터를 나눠서 train/dev 데이터셋으로 나눠서 사용했는데, 이 시점에서 왜 k-fold cross validation을 적용할 생각을 하지 못했는지 모르겠다. 계속 다른 걸 해봐야겠다는 생각에 CV 이외에도 기본적인 걸 놓친거 같아 많이 아쉬웠다.
    - 좀 더 다양한 baseline model 비교
    - learning rate scheduler 사용/비교
    - 다양한 EDA 분석
- 다음엔 매일매일 내가 한 일들 or 팀의 프로젝트 진행 상황 등을 따로 페이지를 만들어 기록해놓으면 좋을 것 같다.
- 생각이 너무 baseline 방법에 갇혀있던 것 같다. 마지막에 김성현 마스터님이 말씀해주신 생성모델 사용에 대해선 정말 한번도 생각해본 적이 없었다. 이외에도 많은 방법들이 있을텐데 이에 대한 생각을 해보지 않았다는 것이 아쉬우면서 부족했던 점이라고 생각한다.

한계/교훈을 바탕으로 다음 프로젝트에서 스스로 새롭게 시도해볼 것은 무엇일까?

- wandb, sweep 뿐만 아니라 streamlit, SFTP 등의 툴을 이용하여 프로젝트 진행이 원활히 될 수 있는 환경을 구축하는 일을 도전해보고 싶다.
- 내가 구현한 코드를 모든 팀원들이 동일하게 사용할 수 있도록 코드를 작성하고 그런 코드 파일들을 용도에 맞게 정리하고 관리하는 역할을 맡아서 해보고 싶다.

  비슷한 맥락에서 같은 것을 구현하더라도 지금보다 좀 더 간결하게 코드를 작성할 수 있도록 노력할 것이다.
- 생각이 너무 baseline 방법에 갇혀있던 것 같다. 아직 다른 방법이나 모형들을 잘 몰라서 그런 것일 수 있지만 틀에서 벗어난 새로운 방법을 생각해보고 공부해서 시도해볼 것이다.

이종윤

나는 내 학습목표를 달성하기 위해 무엇을 어떻게 했는가?

- 우리 팀과 나의 학습목표는 무엇이었나?
  - 관계 추출에 대한 논문들을 참고해 좋은 성능을 나타내었던 방법들을 klue 데이터에도 적용시켜보며 좋은 방법들을 찾아본다.
- 개인 학습 측면
  - 강의에서 들었던 것들을 실제로 대회에서 적용시켜 보며 확인하려 했다.
- 공동 학습 측면
  - 매일 아침 각자가 해본 결과를 공유하며 서로 피드백한다.

나는 어떤 방식으로 모델을 개선했는가?

- 사용한 지식과 기술
  - EDA를 통해 데이터의 분포나 특징들을 확인해본다.
  - Entity의 위치나 type을 알려주는 token을 추가해본다.
  - 오답을 분석해서 어떤 부분에서 틀렸는지 확인해본다.

- 내가 한 행동의 결과로 어떤 지점을 달성하고, 어떠한 깨달음을 얻었는가?
    - 적용해본 방법들이 원래의 기존 방식을 취했을때보다 좋은 성능을 내지 못했다.

전과 비교해서, 내가 새롭게 시도한 변화는 무엇이고, 어떤 효과가 있었는가?

- 저번 대회에서는 어떤 부분들이 틀려서 오답으로 나왔는지 분석해보지 않았기 때문에 이번에는 어떤 부분들이 왜 틀렸는지 확인하려 했었지만 틀린 부분들의 공통점을 찾기가 쉽지 않았다.

마주한 한계는 무엇이며, 아쉬웠던 점은 무엇인가?

- 다양한 방법들을 적용해보아도 성능이 좋아지지 않아 사실 동기 부여가 잘 되지 않았다. 지금 와서 생각해보면 roberta-base로만 실험을 해봤었는데 roberta-large로 실험을 했다면 더 좋은 결과가 있었을지 궁금하다. val loss가 낮아졌다 높아지는 구간이 생기면 무조건 over-fitting이 생겨서 학습을 중단했었는데 그렇게 시간을 좀 낭비하고 제대로 결과를 판단하지 못했던 것 같다.

김상윤

나는 내 학습목표를 달성하기 위해 무엇을 어떻게 했는가?

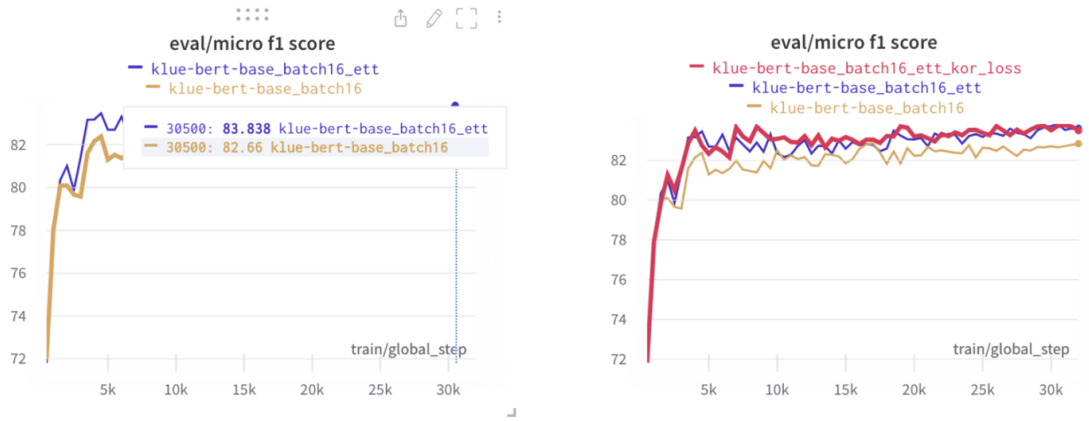
- 우리 팀과 나의 학습목표는 무엇이었나?
    - 저희 팀은 최대한 **다양한 시도**들을 나눠서 진행함으로써 RE task에서 어떤 방법론을 적용했을 때 좋은 성능을 이끌어낼 수 있나 파악했고, 그 과정에서 직접 구현하여 결과를 확인해 보는 것을 목표로 했습니다.
    - 저는 맡은 방법론의 **도입 배경과, 그 구현 방법** 그리고 어떤 효과를 얻을 수 있었는지 얻은 **결과에 대한 이해**를 목표로 했습니다. 기본적으로 점수 상승을 위한 시도들을 했지만, 그 과정에서 앞서 언급한 측면을 위해 충분한 시간을 투자했습니다.
- 개인 학습 측면
    - 데이터와 모델에 새로운 방법을 도입하기 전에 그 근거와 배경을 이해하고 수행하기 위해, 충분한 시간을 투자했습니다. **기존 데이터 구성과 형태를 파악**한 후 이에 기반하여 데이터에 변화를 주었습니다. 또한 모델 구조를 바꾸는 과정에서는 기반이 되는 **transformers 라이브러리의 구현 코드 흐름을 이해**하고 필요한 구현을 추가

및 수정하기 위해 많은 시간을 투자했습니다. 이를 통해 결과에 대한 설명력을 더했습니다.

- 공동 학습 측면
  - 최종적으로 좋은 점수를 내는 것을 목표로 했지만, 단순히 size가 큰 모델과 앙상블 활용을 방법론으로 생각하지 않고, 기본 모델에 다양한 시도를 적용하고 실험 해보았습니다.

나는 어떤 방식으로 모델을 개선했는가?

- 사용한 지식과 기술
  - 전체 data 중 10%에 해당하는 **[UNK]토큰을 줄이기 위한 시도**를 했습니다. 이 중 1/3이 한자에 대한 tokenize가 안 되어 발생했기에 한자를 한글로 바꾸는 hanja 라이브러리를 sentence에 적용했습니다.
  - 한자를 한글로 모두 바꾸니, 문장 내 두 entity가 구분이 안 되는 문제가 발생했습니다. 또한 문장 자체에서 entity가 명시되지 않아 모델에서 인지하기 어려웠기에 **entity를 표시하는 special token을 추가**했습니다.
    - subject / object 정보와, entity의 type 정보를 모두 구분하는 16개의 special token을 추가했습니다.
    - subject / object 정보만 구분하는 4개의 special token을 추가했습니다.
  - 모델의 loss function을 **hinge loss로 변경**하여 적용해보았습니다.
  - early stopping의 기준을 eval_loss에서 **eval_micro_f1 score로 변**경했습니다.
  - BERT(RoBerta) 모델의 embedding layer에서 entity에 대당하는 토큰에 가중을 줄 수 있도록 **Entity Embedding Layer를 추가**하는 모델 구조 변경을 시도했습니다.
- 내가 한 행동의 결과로 어떤 지점을 달성하고, 어떠한 깨달음을 얻었는가?
  - bert-base 사전학습 모델을 기반으로 fine-tuning data에 중국어→한글 변환 적용하거나, 모델 loss function 종류 변화를 줬을 때 성능에 향상이 없거나 더 떨어질 때도 있던 반면,
  - data에 **entity_special token 추가로 성능 향상**을 확인했습니다. (bert-base, eval_f1 : 82.6 → 83.838) 앞선 두 방법론을 추가 적용했을 때도 이와 유사한 성능이 확인 됐습니다.



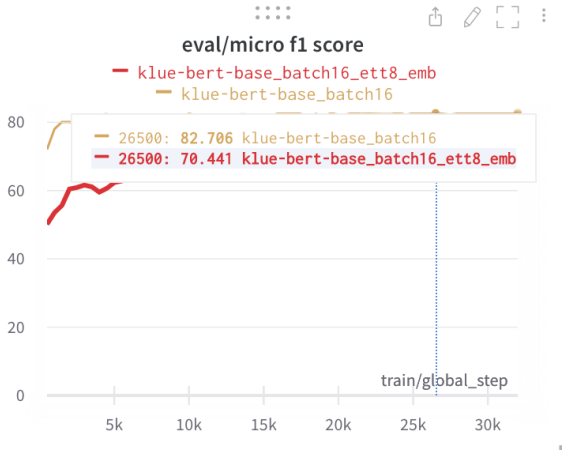
- **roberta-large모델에 세 방법론을 함께 적용** 했을 때는 약간의 성능 향상을 이끌어 낼 수 있었습니다.
  : 리더보드 기준, f1_score 63점대 → **69.27**점 도달.
- 어떤 방법론에 대해 단순히 한 번 시도 후 판단할 것이 아니라, 매번 **다양한 조건에 따라서 결과를 고려**해 봐야 하는 것을 알게되었습니다.

전과 비교해서, 내가 새롭게 시도한 변화는 무엇이고, 어떤 효과가 있었는가?

- 데이터 전처리를 통해 정제하고 변화를 주어 성능 상승을 처음 이끌어 보았고, **좋은 데이터로 학습하는 것에 대한 중요성**을 크게 느끼게 된 경험이었습니다.

마주한 한계는 무엇이며, 아쉬웠던 점은 무엇인가?

- 모델의 상단 output 부분에 layer를 추가하는 것이 아닌 중간 구조에 변화를 주려는 시도에서, **구현에 어려움**을 겪었습니다. 결국 좋은 성능 상승을 이끌어 내지 못 했고 구현에 문제가 있던 것인지 확신을 갖지 못 한채 마무리 되어 아쉬웠습니다.



- 팀원들과의 **체계적인** 소통과 **코드 공유 부족**으로, 진행 과정에서 비효율적인 시간들이 많이 발생했던 것 같습니다.

한계/교훈을 바탕으로 다음 프로젝트에서 스스로 새롭게 시도해볼 것은 무엇일까?

- **데이터 정제**에 더 많은 시간을 투자하면 더 좋은 결과를 얻을 수 있을 것이라 생각합니다.
- 모델 구현 코드의 흐름을 이해하는데 많은 투자를 했기에 다음번에는 새로운 구조의 모델을 도입한다면 더 **완성도 있게 구현** 할 수 있을 것 같습니다.
- 팀원들과 **github을 잘 활용하여 코드 공유**와, **코드 리뷰**를 진행해 볼 것입니다.