데이터셋 정보

- **train.csv [**파일**]** 학습 데이터
  - title : 글의 제목
  - full_text : 전체 글(Text)
  - generated : 해당 글이 AI로 생성되었는 지의 유무 (0 : 사람이 작성, 1 : 생성 AI가 작성)
- **test.csv [**파일**]** 평가 데이터
  - ID : 평가 샘플 고유 식별자
  - title : 글의 제목
  - paragraph_index : 글을 구성하는 문단의 번호(순서)
  - paragraph_text : 문단(Text)
- **sample_submission.csv [**파일**]** - 제출 양식
  - ID : 평가 샘플 고유 식별자
  - generated : 해당 글이 AI로 생성되었을 확률 (0~1)

0. 방향성 고민

- GPT, Gemini 활용하면서 LLM만의 특징이 될만한 스타일이 있었는지?
  - 표준어 일수 있지만 일반적으로는 잘 사용되지 않은 문법이 있었던 것 같다.
  - 개념 설명 등을 답변할 때 문장 사이에 ‘,’를 사람보다 많이 썼던 것 같다.
- 룰 기반으로 train set을 구분할 수 있을지?
  - 문장 별로 사용하는 어미 분석
  - 특수기호, 특정 단어 사용 여부 및 빈도 측정
- 모델 학습 단계에서 토큰 수를 줄이거나 질을 높이기 위한 전처리 단계 검토

1. 데이터셋 검토

- 문장 사이를 구분하는 ‘,’ 사용 샘플 비교
- train set 의 샘플들은 일관적으로 인물, 사건, 장소, 이론 등 다양한 주제를 설명하고 있다.
- 샘플 몇 개를 같은 주제에 대해 설명하는 것을 색출
- 축구 선수에 대해 설명하는 샘플, 사람 vs AI
  - 문장 사이에 불필요해 보이는 ‘,’ 를 사람이 더 자주 사용

<사람 작성>

<AI 작성>

- 특정 지역에 대해 설명하는 샘플
  - 일반적인 단어의 나열을 제외하고 문장을 나누는데 사용된 ‘,’ 를 확인
  - AI가 더 많은 문장 수 대비 ‘,’ 사용 빈도가 적음.

<사람 작성>

<AI 작성>

- 이 샘플들을 비교하면서 단어 또는 절을 나열하는 방식도 차이가 있는지 확인해볼 필요가 있다고 생각됨

  (나열도 사람이 더 자주 쓸지도?)

2. 문장 어미 비교

- 샘플 확인 중 AI 작성 글에서는 'ㅂ니다' 와 '있다, 였다 등' 두 어미를 혼용하는 경우 많아, 전체 비율을 확인해봄.

- 문장 단위로 '. ', '? ','! ' 기준으로 분리하고 'ㅂ니다' 로 종료되는 문장 수 count

- 그외 경우로 종료되는 문장 수를 count하고 그 값을 따로 추출해서 저장.

  - ex) ['한다', '있다', '는다', '있다', '없다', '없죠']
- train set 전체에서 어미를 혼용하여 사용한 비율.

[train data] -ai : 7995 중 5442, 비중 0.6806754221 -human : 89177 중 4094, 비중 0.04590869843

	전체 샘플	혼용 케이스	비율
AI	7995	5442	68%
사람	89177	4094	4.5%

3. 모델 학습 전**,** 전처리 전략

- 특수 문자들 중에서 (), [], {}, <> 와 같은 쌍으로 존재하는 텍스트에 주목.
- () 안에는 바로 앞 단어의 한자, 연도, 부연 설명들이 있어 토큰화 진행시 제거가 가능할 것이라 예상.
- 그래도 필요할지 몰라, 별도 컬럼으로 저장하고 진행.
- 그외 특수 기호는 책 제목, 인용구 등 여러 방면으로 사용
- 특수 기호 쌍 목록

paired_symbols = [('(', ')'), # () ('[', ']'), # [] ('{', '}'), # {} ('〈', '〉'), # 〈〉 ('《', '》'), # 《》 ('「', '」'), # 「」 ('『', '』'), # 『』 ('‘', '’'), # ‘’ ('“', '”'), # “” ("'", "'"), # " ("", "") # ""
사

- 쌍으로 이루어진 특수기호 사용 빈도
    - 샘플 당 pair_count 평균
        - 0(사람): 11.7
        - 1(AI): 7.7
    - (문장 당 빈수수) 대비 평균
        - 0(사람): 0.370425 #3문장 마다 특수 기호를 사용한다고 볼 수 있다.
        - 1(AI): 0.218490 #5문장 마다
- 특수 기호 자체를 안 쓰는 샘플도 꽤 많은데, 평균이 너무 높다?
    - 극단치 확인을 위해 분석해보니

        - 문장수 1437, 특수쌍 2245..
    - 문장 수 기준 상위 2~3% 제외하고 모델 학습해볼 예정

4. 진행상황

- 각 문장을 토큰화하여 품사로 태깅한 후 패턴에 대한 모델 학습 시도
    - ex) 수난곡(受難曲)은 배우의 연기 없이 → [Noun], [Josa], [Noun], [Josa], [Noun]
- ()를 포함해 안에 텍스트 제거 후 진행 → 특수기호 쌍 자체를 Noun 또는 특수하게 처리해서 진행 시도

Base Model

Dacon 해당 과제 submission.csv 제출시 public score

평가기준 ROC-AUC

- “klue/bert-base” : 76점
- “skt/kobert-base-v1” : 68점
- **“monologg/koelectra-base-v3-discriminator” : 84**점

아이디어 컨셉

학습 데이터는 전체 글(Full Text)에 대해, 일부 문단이나 문장만 ***AI***가 작성된 경우에도 글 전체에 'AI 작성' 라벨(1)이 부여되며, 문단 단위 라벨은 제공되지 않습니다.

Testdata 형식은 문단단위

그러면 기존에 1로 표시된 문서도 문단단위로 분할했을때 사람이 작성한 부분과 AI가 작성한 부분이 혼재되어 있을수 있음

- 학습 데이터에 노이즈가 섞여있다.

Attention-MIL의 개념

- 상황: Train 라벨은 문서 단위(해당 문서에 AI 문단이 하나라도 있으면 1), 하지만 Test는 문단 단위 확률을 제출해야 함.

- 전략: 문서를 **bag**, 각 문단을 **instance**로 보고,

    1. 문단 인코더(KoELECTRA)로 각 문단 임베딩 h_i와 문단 로짓 s_i를 만든다.

    2. **Attention** 풀링으로 문단들의 중요도 $α_i$를 학습해서 문서 표현 $z=\sum_i \alpha_i h_i$를 만든 다음 문서 로짓 S를 예측한다.

    3. 학습은 문서 라벨에 대해 BCE 손실.

        추론에서 문단 확률 $σ(s_i)$과 문단 중요도 $α_i$를 그대로 사용한다.

- $α_i$: "이 문서에서 **i**번째 문단이 얼마나 중요한가" (시각화 가능)

- 장점: 문맥**/**가중 평균으로 문서 신호를 안정적으로 모으고, 해석성**(α)** 확보.

주요 수식

$$ e_i=u^⊤tanh(Wh_i),
α_i=\frac{exp(e_i)}{∑_jexp(e_j)}, z=∑_iα_ih_i, S=w^⊤z $$

Attention-MIL 진행 과정

1. 각 문단을 인코더로 지나가게 하면 문단 특징 벡터(임베딩)이 나옴
    - KoELECTRA 사용
2. 각 문단에 중요도(α)를 추가함
    - α가 크면 문서의 라벨을 결정하는데 크게 작용했다는 뜻
3. 최종 문서 라벨은 "문단 특징 벡터 x α"를 가중 평균 해서 만듬
    - 중요한 문단은 크게, 덜 중요한 문단은 작게 섞임
4. 문서 라벨과 비교해서 틀렸다면 α와 문단 특징 뽑는 방식을 조금씩 고침(학습)
5. 학습이 끝나고 얻는 결과
    - 문단 점수(그 문단이 AI일 확률)
    - 문단 중요도 α(그 문단이 이글에서 얼마나 핵심인지)

전체 진행과정

1. Preprocessing
    - 문서단위 data ⇒ 문단단위 data 변환
        - 학습 / 추론 단위를 문단으로 통일
    - HTML 태그 제거 / 한자 제거
        - 토크나이저가 불필요한 토큰을 만들지 않도록 정리
        - 길이 감소, 잡음 저감
2. Encoder : KoELECTRA + LoRA
    - 문단 텍스트를 고정 길이의 의미 벡터로 변환
    - 학습은 LoRA만 업데이트하여 빠르고 안정적인 미세튜닝
    - input - 토크나이즈 된 텐서 input_ids $[B, L]$, attention_mask $[B, L]$
        - B : 배치 크기
        - L : 최대 토큰 길이(max_len) = 192
    - output
        - 토큰 히든 : $H_{all} [B, L, d]$
        - CLS 임베딩 : $h_{cls} = H_{all} [:, 0, :] → [B, d]$
    - LoRA target_modules
        - $Query, Value$
3. MIL Pooler
    - 한 문서 안의 여러 문단 임베딩을 모아 가중합
    - 각 문단의 “문서라벨에 대한 중요도”를 학습적으로 산출
    - input - 같은 문서에 속한 문단 임베딩
        - $H = [h_1, h_2, … h_n] h_i∈R^d$
    - output
        - 문단 중요도 : $α ∈ R^n, α_i ≥ 0, Σα_i = 1$
        - 문서 표현 : $z = Σ_i α_i · h_i ∈ R^d$
4. Bag Head
    - 문서 표현 z를 최종 이진 로짓 S로 사상 (문서가 AI일 확률)
    - input - $z ∈ R^d$
    - output - $S ∈ R$
5. Training Loss
    - 문서 라벨(약지도)만 주어졌을 때도 문단 중요도와 인코더를 함께 학습
    - input
        - 문서 로짓 S
        - 문서 라벨 $y_{doc} ∈ {0,1}$
    - output - 스칼라 손실 L
6. Backpropagation
    - L을 기준으로 LoRA 어댑터 + 헤드 + MIL 파라미터(W,u)를 업데이트
    - input
        - 손실 L, 파라미터 $θ$
    - output
        - 업데이트된 $θ$
        - 원본 encoder 가중치는 고정, LoRA만 갱신
7. **Temperature Scaling**
    - logit을 하나의 스칼라 온도 T로 나누어 확률을 다시 계산하는 사후 보정기법

    - $p_{doc} = σ(S / T)$

현재 직면한 문제

1. 학습 시간
    - Epoch당 8시간 소요
2. 세션 유지
    - Colab pro+ 이용으로 최대 24시간 유지가 된다고 했지만 어떤 이유에선지 자꾸만 세션이 끊어짐
3. 위 아이디어대로 진행을 했지만 성능이 잘 안나옴
    - 초반 제출한 문서단위 LoRA submission 기준 84점
    - 지금건 55점
    - 왜 성능이 내려갔을까?
    - 학습 신호가 희석되거나나 설계가 데이터 분포와 안맞을 수도?
4. 확률이 0.5 근처에 몰리는 현상

성능 향상에 관한 아이디어

1. **TS(temperature scaling)**를 문단**(instance)**에도 적용하면 확률이 평평해져서 0.5 근처로 몰리기 쉬움
2. 시퀀스 킬이 (MAX_LEN) 조정
3. 문단 샘플링
4. 배치 크기 조정
5. loss 조정
6. LoRA 용량 조정
    - r, lora_alpha

기존 모델의 문제 가정

- 모델 구조가 '1'인 문단을 충분히 끌어올리지 못했나?
- gpu RAM을 비효율적으로 사용하나?
- 문단 수가 많으면 인코더 호출이 늘어질수도?
- TS를 좀 더 예리하게 다듬을 필요가 있나?

개선점

1. 배치크기 늘리기
    - BATCH_BAGS 2 → 4
2. 문단 서브샘플링 사용

    - TRAIN_PARAS_PER_DOC 8 → 16
    - 문서당 K개의 랜덤문단 사용
    - 검증단계는 전체 문단 사용
3. Hard-Positive Subsamling
    - 매 epoch마다 의심문단을 캐시해서 훈련 배치에 항상 포함
    - label이 1인 문서에서 핵심문단이 빠지지 않음 → recall 개선
4. MIL Pooler에 LSE 추가
    - 문서 안 여러 문단을 하나의 점수로 합칠때 “가장 의심스러운 문단을 더 세게” 반영
5. Instance 보조 손실
    - 문단 로짓이 bac BCE로만 간접 학습하는 것을 보완
    - 문단 확률이 0 / 1방향으로 벌어지게함
6. 클래스 가중치 강화
    - 기존 같은 비율로 맞추던 가중치를 label 1에 1.3배 곱함
    - label 1 탐지율을 높임 → recall 개선
7. Temperature Scaling 문서 로짓에만 적용
    - Instance(s_i)에 T 를 적용하면 확률이 평평해져 0.5 근처로 수렴하던 문제 방지

결과

submission score

- public : 86.3
- pravate : 86.9