

MRC

Open-Domain Question Answering

Member

김민서, 김수진, 양가연, 이예서, 홍성민, 홍성재

목차

0. 프로젝트 개요

1. Data

- 1.1. 데이터 EDA
- 1.2. 데이터 증강
 - chatGPT API
 - negative passage
- 1.3. 외부 데이터
 - ETRI
 - KorQuAD
- 1.4 데이터 전처리
 - Context 자체 길이 축소
 - 위키피디아 전처리

2. Reader

- 2.1. 모델 선정
- 2.2 개선
 - batch_size 조절
 - Model customizing
 - 형태소 분석기 앙상블을 통한 후처리 작업
 - Distillation
 - Retrospective Reader(Retro Reader)

3. Retrieval

- 3.1. Sparse
 - TFIDF
 - BM25
 - Elastic(BM25)
 - Retrieval 성능 비교
- 3.2. Dense
- 3.3. colBERT
- 3.4. Reranker

4. Ensemble

- 4.1. hard ensemble
- 4.2. soft ensemble
- 4.3. weighted ensemble
- 4.4. merged ensemble

5. 개인회고

0. 프로젝트 개요

프로젝트 개요

Question Answering(QA)는 다양한 종류의 질문에 대답하는 인공지능을 만드는 연구분야이다. 다양한 QA 시스템 중, Open-Domain Question Answering(ODQA)는 주어지는 지문이 따로 존재하지 않고 사전에 구축되어있는 Knowledge resource에서 질문에 대답할 수 있는 문서를 찾는 과정이 추가된다. 본 프로젝트에서 구현해야하는 모델은 질문에 관련된 문서를 찾아주는 “Retriever”단계와 관련된 문서를 읽고 적절한 답변을 찾거나 만들어주는 “Reader”단계, two-stage로 구성되어있다.

프로젝트 팀 구성 및 역할

팀원	역할
김민서	데이터분석, LLM기반 데이터 증강, 모델실험(증강 데이터 비교), 앙상블
김수진	외부데이터 리서치 및 학습, 데이터 전처리, 모델 리서치, 모델 개선, 모델 실험, Retrieval 구현 및 실험(ColBERT), 앙상블
양가연	Retrieval(BM25) 구현 및 실험, 모델 리서치, 모델 개선(Custom Layer, Distillation), 모델 실험, 앙상블(hard ensemble, soft ensemble, weighted ensemble)
이예서	PM(마일스톤 및 이슈 관리), 인프라 담당(개발환경 구성 스크립트화), 베이스라인 코드 템플릿화, 데이터 전처리(cleansing), 데이터 증강(negative passage), Retrieval 구현 및 실험(Elasticsearch, Reranking), 앙상블(merged ensemble)
홍성민	모델 리서치, 모델 실험, Retrieval 구현 및 실험(Dense), 앙상블
홍성재	EDA, 데이터 전처리, Reader성능 개선 관련 조사(Retrospective Reader), 앙상블

1. Data

1.1. Data EDA

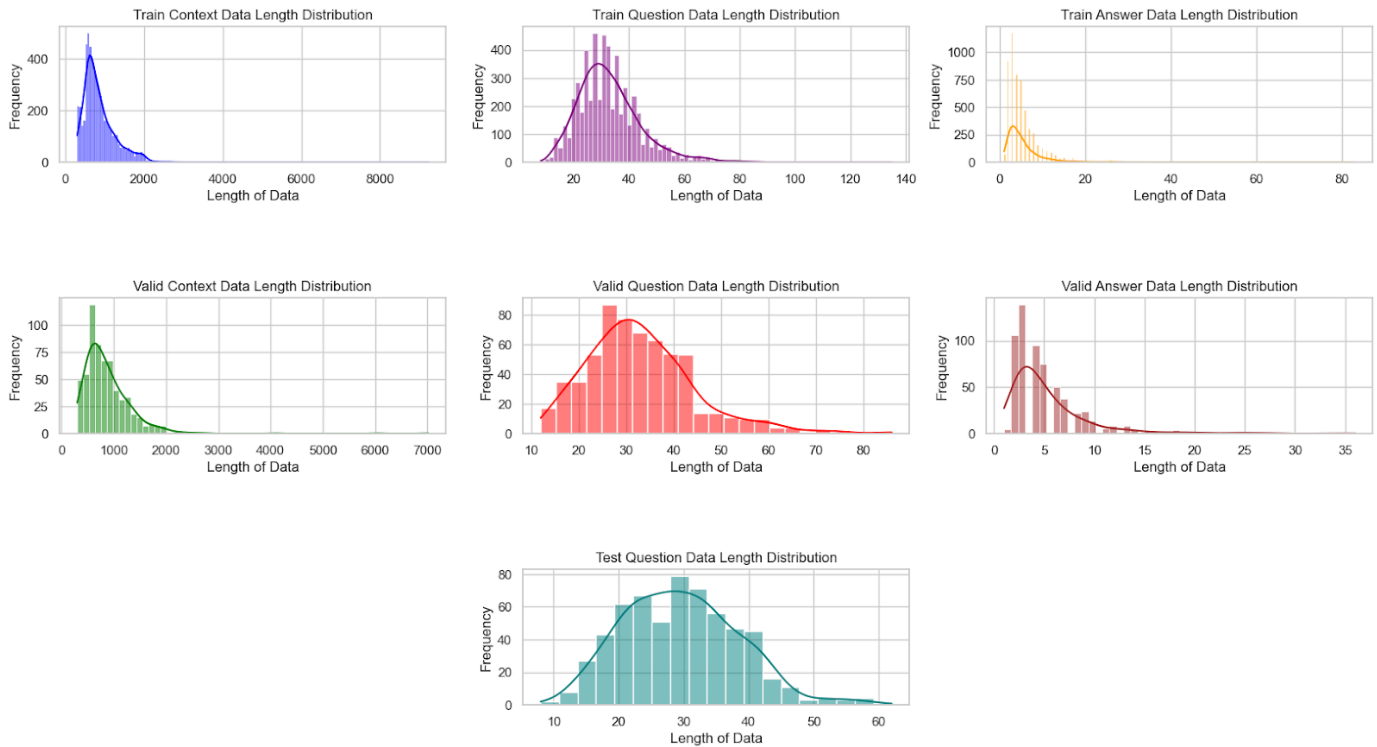
1.1.1. 필요성

데이터 분석의 첫 단계로, 주어진 데이터의 특성을 이해하고 데이터 품질을 점검하기 위해 탐색적 데이터 분석(EDA, Exploratory Data Analysis)을 진행했다. 이는 모델 학습 전에 데이터의 분포와 특성을 파악하여, 데이터의 품질 문제나 이상치를 사전에 확인하고 적절한 대응을 위한 목적이다.

특히, 이번 ODQA(오픈 도메인 질문 응답) 과제에서는 retriever와 reader 두 단계의 성능을 극대화하기 위해 데이터의 구조와 패턴을 명확히 이해하는 것이 필요했다. 따라서 각 데이터셋의 질문, 지문(context), 그리고 답변(answer)의 길이를 분석하고 데이터셋의 전체적인 구성과 분포를 확인하는 작업을 수행했다.

1.1.2. 데이터 길이 분석

우선, 학습(train) 및 검증(validation) 데이터셋의 각 요소인 질문(question), 지문(context), 답변(answer)의 길이 분포를 분석했다. 각 데이터셋에서 이들 요소의 평균 길이, 최소 길이, 최대 길이를 계산하고, 전체 데이터에서 이상치(outlier)로 판단될 수 있는 극단적인 길이의 항목들을 파악했다. 이러한 분석은 질문과 지문, 답변의 길이가 모델 성능에 미치는 영향을 사전에 이해하고, 적절한 전처리 방법을 고려하기 위해 진행되었다.



(Train, Dev, Test 데이터셋에서 Context, Question, Answer의 길이 분포 시각화(하단 : Test 데이터셋은 Question 길이만 존재하여 시각화))

분석 결과, 테스트(test) 데이터셋의 길이 분포가 학습 데이터셋의 분포와 큰 차이가 나지 않는 것을 확인했다. 또한, 약간의 최댓값 차이는 직접 확인하여 잘못된 값이 있는지 검토했으며, 별다른 문제는 발견되지 않았다.

1.1.3. 데이터 결측치 확인

다음으로 데이터셋에 결측치가 있는지 확인했다. 질문, 지문, 답변 중 어느 하나라도 비어있는 경우 모델 학습에 영향을 줄 수 있기 때문에, 모든 데이터가 온전히 제공되고 있는지 확인하는 과정을 거쳤다. 이번 분석에서 학습 데이터셋과 검증 데이터셋 모두 결측치가 없는 것으로 확인되었다.

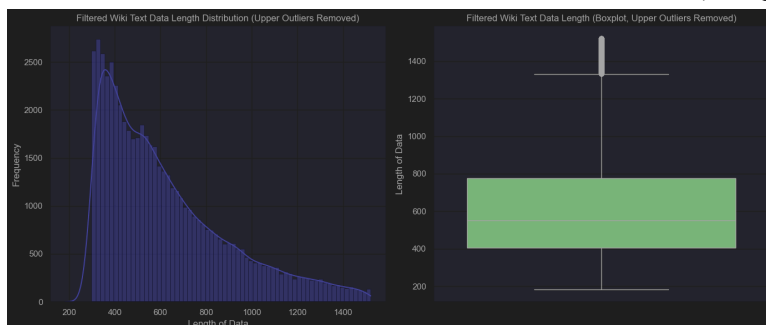
1.1.4 Train 데이터셋의 context와 Wikipedia 데이터 비교

ODQA 과제의 특성상, retriever 단계에서는 주어진 질문에 적절한 지문을 찾아야 한다. 이를 위해 학습 데이터셋의 지문(context)이 retriever에서 사용할 wikipedia.json 파일에 포함되어 있는지 확인하는 과정을 거쳤다. 학습 데이터의 지문이 wikipedia.json 파일에 포함되어 있을 경우, retriever가 학습한 내용과 실제 검색 단계에서 사용하는 데이터 간의 일관성이 높아져 모델의 성능 향상에 도움이 된다고 생각했다.

그 결과, 학습 데이터의 지문(context)과 제목(title)이 wikipedia 데이터의 지문과 제목과 일치하지 않는 경우가 일부 발견되었다. 자세히 확인해본 결과, 지문은 완전히 일치하지만 제목이 일치하지 않는 경우가 있었으며, 학습 시에는 제목을 참고하지 않기 때문에 이러한 불일치는 학습에 영향을 주지 않을 것으로 판단하여 별도의 전처리는 진행하지 않았다.

1.1.5 Wikipedia 데이터 분포 분석

추가적으로, wikipedia.json 파일에 저장된 모든 지문의 길이 분포를 분석하고, 이 데이터가 학습 데이터셋으로 발췌되어 사용된 지문과 길이 면에서 얼마나 유사한지 비교했다. 이 비교를 통해 학습 데이터셋의 지문이 얼마나 균형 잡힌 분포를 가지고 있는지, 그리고 wikipedia 데이터와의 차이점이 있는지 파악할 수 있었다. 실제로 길이 분포를 그래프로 시각화해본 결과, 문서 하나당 46099자를 포함하는 상당한 아웃라이어가 관측되었다. 이러한 아웃라이어를 처리하기 위해 IQR(Interquartile Range)을 활용하여 상단 아웃라이어를 제거하였다.



상단의 극단적인 길이 값을 가진 아웃라이어를 제거한 후, 데이터 분포를 시각화 하였다. 추가적으로, 아웃라이어에 해당하는 데이터들을 직접 눈으로 확인한 결과, retriever과정에서 문제가 있을정도의 데이터 자체에 문제가 있거나 품질 이슈는 발견되지 않았다.

1.2. Data 증강

1.2.1. 필요성

모델의 일반화 성능을 높이기 위해 학습 데이터의 다양성을 늘리기 위해 데이터증강을 진행했다. 그 결과 Reader의 성능을 향상시킬 수 있었다. 그러기 위해 학습 데이터의 질문을 추가로 생성하거나 동의어로 변형하여 데이터 증강을 진행했다.

1.2.1. chatGPT API를 사용한 증강

(1) Train Question 질문 추가 생성

첫 번째 방법으로, 기존 학습 데이터의 지문(context)을 활용하여 새로운 질문을 생성하는 작업을 진행했다. 이를 위해 GPT API를 사용하여 지문을 입력하고, 해당 지문에 대한 질문과 답변을 생성하도록 했다. 그러나 초기 시도에서는 모델이 답변을 지문에서 추출하지 않고 자주 새롭게 생성하는 문제가 발생했다. 이 문제를 해결하기 위해 방법론을 재구성하였고, 새로운 접근법은 우선 지문에서 중요한 키워드를 추출하여 이를 정답으로 삼고, 해당 정답에 대한 질문을 생성하는 방식이었다. 이러한 방식을 통해 답변이 지문 내에 포함될 수 있었으며, 보다 일관성 있게 질문을 생성할 수 있었다. RoBERTa-large 모델을 사용하였으며 데이터셋을 제외하고 모든 환경 세팅을 동일하다. 제출 후 Public score를 기준으로 성능을 평가하였다.

데이터 셋	PUBLIC EM / F1 Score (%)	Private EM / F1 Score (%)
대회 제공 base	62.5000 / 74.9200	61.3900 / 73.7900
증강 데이터	55.8300 / 68.5900	56.3900 / 69.5000

(2) Train Question 동의어 증강

두 번째 방법으로는 기존 질문에 동의어를 활용하여 질문을 증강하는 작업을 진행했다. 이 과정에서도 GPT API를 사용했으며, 지문(context)과 답변(answer)은 그대로 유지한 채 질문(question)을 동의어를 사용하거나 문맥을 변형한 형태로 생성하도록 프롬프팅을 진행했다. 이 방식으로 각 원본 질문마다 동의어 및 문맥 변형 질문을 하나씩 생성하여 총 3,952개의 질문을 두 배로 증강하였다. 이와 같은 데이터 증강 과정을 통해 학습 데이터의 다양성을 크게 향상시킬 수 있었으며, 이를 통해 모델이 다양한 질문 형태에 대해 보다 일반화된 성능을 보일 수 있도록 했다. RoBERTa-large 모델을 사용하였으며 데이터셋을 제외하고 모든 환경 세팅을 동일하다. inference 후 Test Score와 제출 후 Public score를 기준으로 성능을 평가하였다.

데이터 셋	TEST EM / F1 Score (%)	PUBLIC EM / F1 Score (%)	Private EM / F1 Score (%)
korQuAD(1epoch) + 대회 제공 base	65.4167 / 72.5055	68.3300 / 77.4100	66.1100 / 79.1800
korQuAD(1epoch) + 증강 데이터	58.3333 / 64.4567	63.3300 / 73.0600	63.6100 / 76.6700

1.2.2. negative passage

실제 task에서는 context가 여러 개의 passage가 연결된 긴 문서로 주어지는데 반해, 학습에서 사용하는 context는 original context 문서 하나만 학습하기 때문에 Test task에 비해 train에서의 난이도가 훨씬 쉽다. 때문에 Reader가 여러 Context에서 정확한 Context를 찾는 능력은 현재 학습으로는 얻을 수 없다는 점이 낮은 성능의 원인이라 생각했다. context에 negative passage를 추가하는 방식으로 학습을 어렵게 한다면 점수 향상을 기대할 수 있을 것이라 생각했다.

train dataset에서 각 question을 쿼리하여 elasticsearch top k 100으로 100개의 passage를 가져와 rerank하여 5개로 추렸다. 가져온 passage에 original passage가 포함되어 있다면 그대로 사용하고, 포함되어 있지 않다면 맨 끝의(가장 유사도가 낮은) passage를 제거하고 original passage를 추가하였다. 실제 task에서 retrieval가 판단하는 유사도 순위를 그대로 학습에서도 반영하도록 하였다. 이렇게 negative passage가 더해진 데이터 셋을 만들어 학습한 결과는 다음과 같다. 프로젝트 막바지에 진행된 실험이기 때문에 현재까지 가장 성능이 잘 나오던 방법론을 따라 KorQuAD를 학습한 모델에 전이 학습하는 방식으로 실험하였다.

데이터 셋	model	EM / F1 Score (%)
대회 제공 base	RoBERTa-large (fine-tuned by KorQuAD)	69.5800 / 78.1500
negative passage 증강	RoBERTa-large (fine-tuned by KorQuAD)	72.0800 / 81.8000
negative passage 증강 + 대회 제공 base	RoBERTa-large (fine-tuned by KorQuAD)	70.8300 / 80.8600

결과는 예상했던 대로 negative passage를 함께 학습한 모델의 성능이 더 높게 나왔다. 여기서 아쉬웠던 점은 negative passage로 증강한 데이터 셋으로 학습한 뒤 대회 제공 base로 한번 더 전이 학습을 진행한 모델이 성능이 낮게 나와 실험을 멈췄다는 점이다. 전이 학습 순서를 바꿔보거나 원본 데이터 셋과 증강한 데이터 셋을 섞어서 랜덤하게 제공하는 등 더 많은 아이디어로 충분한 실험을 진행하지 못하고 시간에 쫓겨 마무리했다. 이 실험은 더 깊게 진행했다면 성능 개선의 여지와 다른 아이디어가 많이 나왔을 것 같아서 아쉽다.

1.3. 외부 데이터

Pretraining 관련 논문¹에서는 pretrained된 domain과 pretraining할 도메인이 유사하지 않을수록 더 큰 잠재적 성능 향상을 기대할 수 있음을 보여준다(DAPT). 또 작업 데이터가 넓은 도메인의 좁게 정의된 하위 집합일 때, 작업 데이터셋 자체나 해당 작업과 관련된 데이터로 pretraining이 유용할 수 있다고 한다(TAPT). KLUE 관련 논문²에서는 KLUE/RoBERTa-large 모델 Pretraining 데이터는 MODU 데이터셋을 포함한 다양한 데이터가 사용되었으며 WIKIPEDIA 데이터셋은 사용하지 않을 것을 확인할 수 있었다. 두 논문 내용을 바탕으로 pretrained된 도메인이 아니며 작업 데이터셋과 관련된 WIKIPEDIA 내 문서로 생성된 외부 데이터셋을 활용했다. 본 실험에서는 외부 데이터셋에 대한 overfitting을 방지하기 위해, 외부 데이터셋으로 먼저 1 epoch 학습을 진행한 후, train 데이터셋으로 3 epoch 학습을 수행하였다.

1.3.1. ETRI

ETRI의 상/중/하 난이도 별 패러프레이즈된 위키피디아 단문질문 QA Datasets³(각 300개)을 사용하였다. context의 길이가 대회 제공 train 데이터셋에 비해서 짧으며 하나의 context에 대해서 다양한 질문으로 구성되어있다. 상/중/하로 구분된 난이도에 따른 질문으로 어려운 질문과 쉬운 질문을 모두 학습할 수 있을 것이라고 생각했다. 데이터를 분석하는 과정에서 해당 데이터셋의 문제점을 발견하였다. context, question, answer로 구성된 데이터셋에서 answer는 context에 포함되지만 question에 대한 answer는 context에서 찾을 수 없는 경우를 발견하였다. (아래의 표 참고) 이는 WIKIPEDIA의 문서 중 정답의 근거만 가져왔다는 것을 알 수 있었다. 따라서 이러한 경우 context 내용을 추가해주었다. 난이도 상에 대한 context 추가 후 성능 향상이 있을 때 나머지 난이도에 대해 context를 추가하기로 하였다.

context	question	answer
1949년 6월 26일, 12시 36분, 서울의 자택인 경교장에서 육군포병 소위 안두희에게 총격당하였다. 곧 병원으로 옮겨졌으나 74세의 나이로 사망하였다.	안두희가 김구를 살해한 날은?	1949년 6월 26일

이 데이터를 사용해서 3가지의 실험을 진행했다. RoBERTa-large 모델을 사용하였으며 데이터를 제외하고 모든 환경 세팅을 동일하다. inference 후 public score(test dataset)을 기준으로 성능을 평가하였다. (1) 난이도 상에 해당하는 데이터셋(300개) (2) 난이도 상중하를 모두 포함하는 데이터셋(900개) (3) context 내용을 추가한 난이도 상에 해당하는 데이터셋(300개)

데이터 셋	Public EM / F1 Score (%)
대회 제공 base	62.5000 / 74.9200
(1) 난이도 상에 해당하는 데이터셋(300개)	62.9200 / 73.8600
(2) 난이도 상중하를 모두 포함하는 데이터셋(900개)	52.0800 / 65.4400
(3) context 내용을 추가한 난이도 상에 해당하는 데이터셋(300개)	59.5800 / 71.3000

실험 결과 난이도 상에 해당하는 데이터셋을 추가한 경우에만 EM Score가 0.42%(한 문제) 향상되었다. 이 경우에도 F1 Score는 떨어졌다.

1.3.2. KorQuAD

KLUE 관련 논문에서 “KorQuAD 2.0은 KorQuAD 1.0 및 우리 데이터셋(KLUE)과는 내용 구성이 상당히 다르다. 특히, HTML 태그와 표 등의 요소가 포함되어 있다.”는 내용을 찾을 수 있다. 대회 제공 데이터는 KLUE와 유사하다고 판단하여 KorQuAD 1.0을 사용했다.

이 데이터를 사용하여 세가지 실험을 진행했다. RoBERTa-large 모델을 사용하였으며 데이터를 제외하고 모든 환경 세팅을 동일하다. train 후 Eval score(validation dataset)를 기준으로 성능을 평가하였다.

train 후, 성능을 비교해보았다. (1) KorQuAD train dataset 중 context 길이가 3000이상인 데이터 삭제 (60399개) (2) (1)에 validation dataset 추가(66173개) (3) (1)에서 validation 데이터의 개수 만큼 삭제한 후 validation dataset 추가 (60399개)

데이터 셋	Eval EM / F1 Score (%)
(1) KorQuAD (train) dataset (60399개)	73.3333 / 81.2839
(2) KorQuAD (train + validation) dataset (66173개)	70.8333 / 79.9101
(3) KorQuAD (train + validation) dataset (60399개)	70.0 / 79.5045

¹ Gururangan, Suchin, et al. "Don't Stop Pretraining: Adapt Language Models to Domains and Tasks." arXiv, 2020, arxiv.org/abs/2004.10964.

² Park, Jangwon, et al. "KLUE: Korean Language Understanding Evaluation." arXiv, 2021, arxiv.org/abs/2105.09680.

³ <https://aiopen.etri.re.kr/corpusModel>

(1)의 train 후의 성능이 가장 높아 해당 데이터로 inference를 진행 후 public score(test dataset)로 성능을 비교해보았다. uomnf97/klue-roberta-finetuned-v2와 RoBERTa-large 모델을 사용하였다.

데이터 셋	model	Public EM / F1 Score (%)
대회 제공 base	RoBERTa-large	65.4200 / 73.7900
(1) KorQuAD (train) dataset	RoBERTa-large	62.5000 / 74.9200
(1) KorQuAD (train) dataset	uomnf97/klue-roberta-finetuned-v2	61.6700 / 72.0200

RoBERTa-large를 사용해서 (1)으로 train 후 base 데이터셋으로 학습하였을 때, base 데이터셋만으로 학습했을 때보다 성능이 향상되었다.

1.4. 데이터 전처리

1.4.1 (dataset) Context 자체 길이 축소

Answer를 찾는데 Context가 너무 길다면, 불필요한 참조 범위가 넓어진다고 판단하여 라벨링되어있는 Start_position을 기준으로 이후 Context를 잘라내는 실험을 진행해보았다. 즉, 정답을 도출하는데 불필요한 정보를 노이즈 데이터로 고려하였고, 단순히 없애는 작업을 진행해보았다. 위와 같은 텍스트 처리 방법은 결과적으로 좋지 않은 방법이었는데, 이유는 아래와 같다. 예를 들어 Answer이 가장 앞에 나오는 데이터일 경우, Answer를 제외한 나머지 문장이 다 날아가게 된다.

```
애플,KT가 '쇼옴니아' 명칭을 사용하지 못하는 계기가 된 핸드폰은 어느 회사 제품인가?,"{'answer_start': [0], 'text': ['애플']}"
```

결과를 해석해보면, 정답을 도출하는데 불필요한 노이즈 데이터를 제거한다는 생각 자체는 틀리지 않지만, 무엇을 노이즈로 고려할지에 대한 판단이 미흡했다. 위와 같은 방식으로 Context를 아예 없애버린다면, 문맥 정보를 학습하지 못할 가능성이 매우 크다. 즉 일반화 성능이 떨어지게되는 것이다.

1.4.2 (WIKIPEDIA.json) 외국어 전처리

wiki.json에 대해 두가지 전처리 실험을 하였다. RoBERTa-large 모델을 사용하였으며 전처리 데이터를 제외하고 모든 환경 세팅을 동일하다. inference 후 Eval score(validation dataset)와 PUBLIC score(test dataset)를 기준으로 성능을 평가하였다.

(1) 줄바꿈 문자(\n), 줄바꿈 표현 문자(\n), 문자 열 내의 "#", 한국어/영어/한자/일본어와 일부 특수문자를 제외한 문자를 ""(빈칸)으로 치환하였다. (2) 줄바꿈 문자(\n), 줄바꿈 표현 문자(\n), 문자 열 내의 "#"을 ""(빈칸)으로 치환하였다.

데이터 셋	Eval EM / F1 Score (%)	PUBLIC EM / F1 Score (%)
대회 제공 base	60.0 / 68.9071	62.5000 / 74.9200
(1) 전처리	61.6667 / 69.795	58.7500 / 70.7300
(2) 전처리	56.6667 / 66.1271	

(1) 전처리가 TEST Score에서 약간의 성능 향상이 있었지만, PUBLIC Score에서는 오히려 성능이 떨어졌다. 그 이유는 지나친 전처리로 인해 중요한 텍스트 정보가 손실되었다고 생각한다. 따라서, 이후 실험에서는 wiki.json에 대해 별도의 전처리를 진행하지 않았다.

2. Reader

2.1. 모델 선정

ODQA는 다양한 도메인에서 질문에 대한 답을 정확히 추출해야 하므로, 한국어에 최적화 된 모델을 사용해야한다. 따라서, 한국어에 특화된 사전 학습 모델로 실험을 진행했다. 특히, KLUE는 한국어를 위한 데이터로 사전 학습된 언어 모델로, 한국어 텍스트의 문맥을 보다 잘 이해할 수 있을 것이라 생각하였다.

2.1.1. BERT vs RoBERTa

모델을 제외하고 모든 환경 세팅을 동일하다. train 후 Eval score(validation dataset)를 기준으로 성능을 평가하였다.

model	epoch	batch_size	Eval EM / F1 Score (%)
BERT-base	3	16*4	56.25 / 65.8074
RoBERTa-base	3	16*4	64.1667 / 72.0636

RoBERTa⁴는 BERT 기반으로 더 많은 데이터와 더 긴 훈련 시간을 통해 성능을 개선한 모델로, 대규모 데이터에 대한 일반화 능력이 우수하다. 특히, 문장 내에서 중요한 단어나 문맥을 더 잘 잡아낼 수 있기 때문에 RoBERTa를 선정했다.

2.1.2. RoBERTa-base vs RoBERTa-large

모델 버전을 제외하고 모든 환경 세팅을 동일하다. train 후 Eval score(validation dataset)를 기준으로 성능을 평가하였다.

model	epoch	batch_size	Eval EM / F1 Score (%)
RoBERTa-base	3	16*4	64.1667 / 72.0636
RoBERTa-large	3	16*4	68.3333 / 77.4049

base 버전보다 large 버전에서의 파라미터 수, 레이어 수, 헤드 수, 히든 사이즈 수 등이 더 크며 KLUE Baseline Scores⁵에서도 MRC TASK에 대해 large 버전의 점수가 더 높은 것을 확인할 수 있었다.

따라서, 이번 프로젝트에서 진행되는 대부분의 실험은 RoBERTa-large를 사용하였다. 별도의 표기가 없다면 본 레포트에서는 RoBERTa 모델은 KLUE/RoBERTa를 의미한다.

2.2. 개선

2.2.1. batch size 조절

RoBERTa⁶ 관련 논문 중 “모델을 더 오래 훈련하고, 더 큰 배치 크기로 더 많은 데이터를 처리하며, (중략) 성능이 크게 향상될 수 있다는 것을 발견했다.”는 내용을 확인할 수 있다. 본 실험 환경에서는 batch_size를 16이상으로 설정하면 학습이 되지 않는다. 따라서 gradient accumulate step(g), 배치사이즈(b)를 사용해서 배치사이즈 g*b로 작동하도록 하였다. RoBERTa-large 모델을 사용하였으며 batch_size를 제외하고 모든 환경 세팅을 동일하다. inference 후 PUBLIC score(Test dataset)를 기준으로 성능을 평가하였다.

batch_size	Public EM / F1 Score (%)
16	58.3300 / 70.0900
16*4	60.0000 / 70.6800
16*8	56.6700 / 70.3000

실험결과, gradient accumulate step(4), 배치사이즈(16)일 때 성능이 가장 좋았으며, gradient accumulate step(8), 배치사이즈(16)일 때는 배치사이즈(16)일 때보다 성능이 떨어졌다. 그 이유는 gradient accumulation step이 4일 때, 8일 때보다 업데이트가 더 자주 이루어져 노이즈가 누적되기 전에 기울기 업데이트가 이루어져 학습이 더 일관되게 진행됐을 가능성이 존재한다. 또한 기울기를 누적하는 과정이 길어지면, 미니 배치에서 얻어지는 세부적인 정보가 사라지거나 평균화될 수 있기 때문이다.

2.2.2. Model customizing

2020 삼성 테크토닉⁷에서 발표한 자료에 따르면, classifier에 CNN layer를 추가하여 성능 개선을 이루어 냈다는 자료가 있었다. 이에 근거하여, RobertaQuestionAnswering 모델은 마지막에 Linear layer만을 통과하는데 이 앞에 CNN layer를 추가해 근접 벡터간의 연관 정보까지 학습되도록 시도하였다.

(1) CNN Layer 추가

Conv1D(K=3) - Conv1D(K=1) - Relu - LayerNormalization으로 구성된 CNN Block을 5개 통과시키도록 하였다.

uomnf97/klue-roberta-finetuned-v2 모델 기준 eval,test 점수 모두 상승하였으나, 제출 점수는 하락하는 결과를 냈다.

(2) Dropout + CNN

오버피팅을 의심하여 Dropout 추가하고 통과하는 블록 수를3개로 줄였으나 눈에 띄는 성능 향상은 이루어지지 않았다.

model	CNN_Block	epoch	batch_size	EM / F1 Score (%)
uomnf97/klue-roberta-finetuned-v2	5	3	16	50.000 / 61.5000
uomnf97/klue-roberta-finetuned-v2	3	3	16	53.7500 / 65.7000

(3) Head Customizing
classification head 부분을 조금

⁴ Liu, Yinhan, et al. "RoBERTa: A Robustly Optimized BERT Pretraining Approach." arXiv, 2019, arxiv.org/abs/1907.11692.

⁵ [KLUE GitHub Repository](https://github.com/KLUE-benchmark/KLUE)

⁶ Liu, Yinhan, et al. "RoBERTa: A Robustly Optimized BERT Pretraining Approach." arXiv, 2019, arxiv.org/abs/1907.11692.

⁷ 삼성 SDS "[Techtonic 2020] Track 1. AI의 한국어 이해, 어디까지 왔나?(KorQuAD 1.0성능개선 Know-how) - 이현재 프로"

YouTube, uploaded by 삼성 SDS, 25 Nov. 2020, https://www.youtube.com/watch?v=ovD_87gHZO4

더 다양하게 custom 하여 모델의 예측 성능을 높이고자 하였다. 추가한 layer 는 MLP,LSTM,Bi-LSTM이다. test,eval 점수가 가장 높았던 Bi-LSTM을 제출해보았으나 여전히 제출 점수는 하락하여 성능향상을 이루어내지 못하였다. 모델의 복잡성 증가로 인한 과적합, RoBERTa 모델과의 구조적 부적합성, 그리고 데이터셋 특성과의 불일치 등이 원인일 것으로 추측하고 있다.

Custom_Layer	Model	epoch	batch_size	EM / F1 Score (%)
Bi_LSTM	RoBERTa-base	1	16*4	44.1667 / 51.0529
MLP	RoBERTa-base	1	16*4	41.2500 / 49.5490
CNN	RoBERTa-base	1	16*4	40.8333/49.3713
LSTM	RoBERTa-base	1	16*4	39.5833/47.8152
Bi_LSTM	RoBERTa-large	1	16*4	67.5000/76.5528

(4) dropout

기존 모델의 구조를 크게 변경하지 않으면서, 일반화 성능을 높이기 위해 dropout과 관련하여 3가지 실험을 진행했다. RoBERTa-large 모델을 사용하였으며 dropout을 제외하고 모든 환경 세팅을 동일하다. inference 후 Eval score(validation dataset)를 기준으로 성능을 평가하였다. (1) 최종 출력에 바로 dropout 적용 (2) 중간 레이어에 dropout 적용 (3) 최종 레이어에 가까운 레이어에 dropout 적용

dropout	Eval EM / F1 Score (%)
적용 X	60.0 / 68.9071
(1)	57.5 / 65.5365
(2)	60.0 / 68.9071

실험 결과에 대한 가설은 다음과 같다. (1)에 대해서 RoBERTa와 같은 사전 학습된 모델들은 이미 충분히 학습이 된 상태로, 파인튜닝 시에 drop out을 적용하면 미세한 정보까지 학습해야하는 상황에서 성능이 오히려 떨어질 수 있다. (2)에 대해서 dropout을 적용한 중간 레이어가 모델 학습에 크게 영향을 주지 않는 레이어일 수 있다. 트랜스포머 구조에서 상위 레이어에서 더 중요한 특성들이 학습되기 때문에, 중간 레이어에 dropout을 적용했을 때 영향이 적을 수 있다. (3)은 train_loss가 너무 커서 실험을 중단했다.

(5) cross entropy loss vs focal loss

RoBERTa 모델은 cross entropy loss를 사용한다. 이 손실함수는 모델의 예측 확률 분포와 실제 레이블 분포 간의 차이를 측정한다. 잘 분류된 예제(easy example)와 잘못 분류된 예제(hard examples)를 동등하게 취급한다. focal loss의 경우 cross entropy loss의 변형으로, 객체 탐지 분야에서 클래스 불균형 문제를 해결하기 위해 제안되었다.잘 분류된 예제(easy example)의 손실 기여도는 줄이고, 잘못 분류된 예제(hard examples)에 더 집중한다. 주로 분류 문제에서 사용되지만 특정 질문 유형이나 특정 위치에서 정답이 나오는 경우, 이러한 불균형으로 성능에 악영향을 준다면 focal loss가 유용할 수 있기 때문에 실험을 진행해보았다. RoBERTa-large 모델을 사용하였으며 loss function을 제외하고 모든 환경 세팅을 동일하다. inference 후 Eval score(validation)를 기준으로 성능을 평가하였다.

loss function	Eval EM / F1 Score (%)
cross entropy loss	60.0 / 68.9071
focal loss	57.9167 / 66.6132

실험 결과 cross entropy loss의 성능이 더 우수했으며, 이를 통해 데이터셋의 불균형으로 성능에 악영향을 주는 경우가 적다고 판단하였다.

2.2.3. 형태소 분석기 앙상블을 통한 후처리 작업

2020 삼성 테크토닉⁸에서 발표한 자료에 따르면, 조사를 탈락시키는 후처리 작업을 통해 성능의 개선을 시켰다고 한다. 예를 들어, 정답이 "толстои"일 때 모델의 예측이 "толсто이가"라면 F1 Score가 점수된다. 따라서 조사를 탈락시키는 후처리 작업이 필요하다. qa model의 prediction 값을 후처리하는 과정에서 4개의 형태소 분석기를 사용하여 2개 이상의 형태소 분석기가 정답의 마지막 단어를 조사로 분류하면 조사를 탈락시키는 후처리 작업을 진행했다. 실험 결과, 후처리 작업의 유무와 상관없이 결과는 같았다. 그 이유는 nbest_predictions를 살펴보면, 정답이 "복잡한 감염병"일 때, "감염병 환자를 진단하거나 관리하기 어려운 경우에" 보다는 "증상의 모호함 등으로 인해 감염병

⁸ 삼성 SDS "[Techtonic 2020] Track 1. AI의 한국어 이해, 어디까지 왔나?(KorQuAD 1.0성능개선 Know-how) - 이현재 프로"

YouTube, uploaded by 삼성 SDS, 25 Nov. 2020, https://www.youtube.com/watch?v=ovD_87gHZO4

환자를 진단하거나 관리하기 어려운 경우에는 감염내과 전문의들의 조언을 받아 진료” 또는 “감염병 환자를 진단하거나 관리하기 어려운 경우”와 같이 길게 예측하는 경우가 많았기 때문에 정답에는 큰 영향을 미치지 않았다.

2.2.4. Distillation

KakaoBrain이 KorQuAD 2.0에서 적용한 방법 중 하나로, 먼저 강력한 성능의 교사 모델을 학습시킨 후, 이를 기반으로 학생 모델을 훈련하는 방식이다. 이 과정에서 교사 모델의 "soft targets"를 학습함으로써 학생 모델은 단순히 정답만을 학습하는 것이 아니라 클래스 간의 관계도 파악할 수 있게 될 것이라 기대하였다.

구현 방법으로는 전체 손실 함수를 두 가지 요소로 구성했다. 첫 번째는 기존의 task-specific 손실인 cross-entropy 손실이며, 두 번째는 교사와 학생 모델 간의 KL divergence 손실이다. KL divergence는 교사와 학생 모델의 logit 값 차이를 최소화하기 위해 사용되며, 이는 학생 모델이 교사 모델의 예측 분포를 모방하도록 했다. 또한, 전체 손실 함수에서 KL divergence 손실에 적절한 가중치를 부여하여 task-specific 손실과의 균형을 맞췄다. RoBERTa-large를 korquad로 finetuning한 모델을 교사 모델로 삼고, CNN을 적용한 custom model을 학생 모델로 삼아 distillation을 진행하였다. 하지만 기존의 교사 모델보다 성능이 오르지 않았다.

2.2.5. Retro Reader

Machine Reading Comprehension task에서는 답을 찾을 수 없는 Query(NoAnswer)에 대해 답이 없다고 확실하게 판단하는 능력 또한 중요하다. Retro Reader는 Sketchy Reading과 Intensive Reading의 2-Stage로 해당 Query가 답변할 수 있는 Query인지 판단하고, 최종 예측을 결정한다. 이를 위해서는 데이터셋에 대한 증강(의도적으로 답변할 수 없는 Query를 삽입) 및 추가적인 Labeling(답변가능여부)이 필요하다. Sketchy Reading까지는 간단한 Test Code를 구현해보았으나, 시간상 최종 결과에 반영되진 못했다.

3. Retrieval

3.1. Sparse

3.1.1. TF-IDF

문서 내 단어의 중요도를 계산하기 위해 단어 빈도(TF)와 역문서 빈도(IDF)를 결합한 통계적 기법으로, 문서 내 자주 등장하지만 모든 문서에서 흔한 단어의 가중치를 낮추고, 특정 문서에서만 중요한 단어를 더 부각시켜 정보 검색 및 텍스트 마이닝에서 효과적으로 단어의 중요도를 평가할 수 있다.

3.1.2. BM25

BM25는 주어진 쿼리에 대해 문서와의 연관성을 평가하는 랭킹 함수로 사용되는 알고리즘으로, TF-IDF 계열의 검색 알고리즘 중 SOTA 인 것으로 알려져 있다. 위키피디아 문서를 불러온 뒤 각 문서의 단어 빈도와 역문서 빈도를 계산하였다. 토큰화된 쿼리를 사용하여 BM25 모델로 관련 문서를 검색하여 상위 k개의 문서를 반환하도록 구현하였다.

3.1.3. Elasticsearch (BM25)

Elasticsearch는 Apache Lucene 기반의 Java 오픈소스 분산형 RESTful 검색 엔진으로, 로그 분석, 웹사이트 검색, 빅데이터 처리 등 대용량 실시간 데이터 분석에 많이 사용된다. 역색인을 사용하여 가장 유사도가 높은 문서를 빠르게 검색할 수 있기 때문에, 이번 프로젝트에서 Retrieval로 사용하기 적합하다고 생각되었다. Elastic사에서 공식적으로 개발한 Nori 한글 형태소 분석기⁹를 Tokenizer로 사용하고, 검색 알고리즘으로 BM25를 기본값으로 사용한다.

3.1.4. Retrieval 성능 비교

각 Sparse Retrieval를 비교하기 위해 RoBERTa-large 모델을 사용하였으며 Retrieval를 제외하고 모든 환경 세팅을 동일하다. inference 후 Eval score(validation dataset)를 기준으로 성능을 평가하였다.

Retrieval	Eval EM / F1 Score (%)
TF-IDF	50.0 / 56.8592
BM25	60.8333 / 67.9476
Elasticsearch	59.5833 / 67.6911

실험 결과, TF-IDF보다 BM25와 Elasticsearch가 더 나은 성능을 보였고, BM25와 Elasticsearch의 성능 차이는 미미했다. 그러나 위 실험은 Retrieval를 Reader와 연결한 환경이기 때문에 Retrieval 자체의 성능을 측정하기 어렵다고 판단했다. 또한 top k별 EM Score / F1 Score의 차이가 크지 않았는데, 이는 top k가 늘어나며 Retrieval의 정확도는 향상되었으나 그만큼 Context의 길이가 길어졌기 때문에 Reader의

⁹ <https://esbook.kimjmin.net/06-text-analysis/6.7-stemming/6.7.2-nori>

성능이 하락하였기 때문이라고 추정했다. Retrieval 단독으로 실험을 구성할 필요성을 느끼고 실제로 top k가 늘어나는 만큼 정확도가 향상되는 지는 아래 실험을 통해 확인하였다.

BM25 (train set, 전체 3952개)

top_k	맞은 개수	틀린 개수	정확도 (%)	top_k	맞은 개수	틀린 개수	정확도 (%)
5	3280	672	82.9960	60	3729	223	94.3573
10	3454	498	87.3988	70	3746	206	94.7874
20	3592	360	90.8907	80	3758	194	95.0911
30	3652	300	92.4089	90	3766	186	95.2935
40	3681	271	93.1427	100	3773	179	95.4706
50	3712	240	93.9271				

top 50 부터는 정확도 증가 폭이 줄어들지만 꾸준히 상승하는 것을 확인할 수 있다.

BM25를 사용했을 때는 top 50까지 탐색했을 때 약 6%의 질문에 대해서 정답 context를 찾을 수 없다.

BM25 (validation set, 전체 240개)

top_k	맞은 개수	틀린 개수	정확도 (%)	top_k	맞은 개수	틀린 개수	정확도 (%)
5	205	35	85.4167	60	230	10	95.8333
10	216	24	90.0000	70	230	10	95.8333
20	223	17	92.9167	80	230	10	95.8333
30	228	12	95.0000	90	231	9	96.2500
40	228	12	95.0000	100	232	8	96.6667
50	228	12	95.0000				

top 30과 top 60에서 정확도가 정체되는 구간이 생겼으며 이후 정확도 증가량이 감소했다.

train set에 비해 validation set에서 약 1~3% 정도 정확도가 낮았다.

Elasticsearch (train set, 전체 3952개)

top_k	맞은 개수	틀린 개수	정확도 (%)	top_k	맞은 개수	틀린 개수	정확도 (%)
5	3411	541	86.3107	60	3783	169	95.7237
10	3543	409	89.6508	70	3793	159	95.9767
20	3667	285	92.7885	80	3803	149	96.2298
30	3722	230	94.1802	90	3810	142	96.4069
40	3741	211	94.6609	100	3817	135	96.5840
50	3764	188	95.2429				

top 60 부터는 정확도 증가 폭이 줄어들지만 꾸준히 상승하는 것을 확인할 수 있다.

Elasticsearch를 사용했을 때는 top 50까지 탐색했을 때 약 5%의 질문에 대해서 정답 context를 찾을 수 없다.

Elasticsearch (validation set, 전체 240개)

top_k	맞은 개수	틀린 개수	정확도 (%)	top_k	맞은 개수	틀린 개수	정확도 (%)
5	209	31	87.0833	60	234	6	97.5000
10	222	18	92.5000	70	234	6	97.5000
20	227	13	94.5833	80	234	6	97.5000
30	231	9	96.2500	90	235	5	97.9167
40	232	8	96.6667	100	235	5	97.9167
50	232	8	96.6667				

validation set에서는 top 60까지 성능이 오르다가 이후에는 거의 정체되었다.
train set에 비해 validation set에서 약 1~3% 정도 정확도가 높았다.

성능평가를 진행하며 내린 결론은 다음과 같다. 우선 top k가 커질수록 retrieval의 정확도가 커졌다. 앞서 추정하였던 “**top k에 대한 Reader의 성능과 Retrieval의 성능 사이의 trade-off**” 가정에 대한 근거를 바탕으로 **top k를 줄이면서 정확도를 높일 수 있는 방법을 찾아야겠다는 방향성을 설정했다**.(Rerank 구현 및 실험으로 이어진다.) 또한 BM25와 Elasticsearch가 성능 차이가 나는 이유는 tokenizer의 차이에서 왔을 가능성이 높는데, Elasticsearch가 내부적으로 BM25 알고리즘을 사용하기 때문이다. 따라서 적합한 tokenizer를 찾는다면 BM25도 Elasticsearch와 유사한 성능을 보일 것이다. 그러나 실행 속도 면에서 Elasticsearch가 더 훨씬 빠르기 때문에 BM25를 개선하여 Elasticsearch와 유사한 성능을 보이도록 튜닝하는 것은 비효율적일 가능성이 높아 우선순위를 낮게 측정했고, 결국 진행하지 않았다.

3.2 Dense

3.2.1. 구현

Dense Retrieval 실습 파트의 코드를 참고하였다. Dense Retrieval의 경우 학습이 먼저 필요하다. question(query)과 passage에 대한 각각의 Encoder를 생성하였다. 유사도 측정 및 softmax를 통해 passage마다의 확률을 확인한다. ground-truth의 확률을 NLL(Negative Log Likelihood)로 계산하여 Loss값으로 전달하는 방식이 사용되었다. 역전파는 question과 passage Encoder에 한 번에 전달된다. 모델을 pickle 저장 방식을 통해 .bin 파일로 각각의 Encoder를 저장하였고, 후에 inference를 위해 해당 Encoder를 불러와 문서를 찾는 방식의 구조를 택하였다. RoBERTa-large 모델을 사용하였으며 Retrieval을 제외하고 모든 환경 세팅을 동일하다. inference 후 Public score(test dataset)를 기준으로 성능을 평가하였다.

Retrieval	Public EM / F1 Score (%)
BM25	60.0000 / 70.8600
DPR	43.7500 / 54.0600

실험 결과 BM25의 성능이 DPR의 성능보다 우수했다. DPR은 사전 훈련된 임베딩 모델에 크게 의존한다. 학습된 데이터가 질문과 문서의 다양한 패턴을 충분히 반영하지 못할 경우 성능이 저하가 있을 수 있기 때문에, 임베딩 모델이 학습이 덜 되었다고 생각한다.

3.2.2. 튜닝

기존 train dataset을 이용하여 학습을 하였다. 처음에는 실습 파트의 파라미터를 사용하였고, 배치 사이즈를 크게 잡았으나 메모리 부족으로 반으로 줄여서 사용하였다. 후에 epoch를 늘렸으나 오히려 성능이 떨어져서 4로 고정하였고, learning_rate를 조절하면서 튜닝을 진행하였다. 우선은 3e-1, 3e-2, ... 와 같은 순서로 진행하여 괜찮은 값을 찾고, 그 뒤에는 1e-5, 2e-5, ... 순서대로 진행하여 결과적으로 epoch 4, lr 5e-5, batch 8을 고정값으로 사용하였다.

평가는 학습이 끝난 후 ipynb 파일에서 validation dataset을 가져와 topk를 저장하고, 그 중에 정답이 있는지 확인하여 정답률을 계산하였다. 해당 부분은 이미 어느 정도 튜닝을 마치고 진행했던 터라, 조금 아쉬웠다.

3.2.3. 외부데이터 학습

단순히 기존 데이터만이 아닌 외부의 데이터를 사용해보자는 의견이 나왔다. Reader 모델에서 사용한 외부데이터 중 KorQuAD가 가장 좋은 성능을 보였기에, Dense Encoder에도 해당 데이터를 사용하였다.

아래에서 batch size가 다른 이유는 Data 개수에서 batch size가 맞게 나뉘떨어지지 않기 때문이다. drop_last=True를 설정하여 마지막 batch를 무시할 수도 있으나, 그러기 보다는 모든 데이터를 학습할 수 있도록 약수로 설정하였다.

또한 KorQuAD를 사용 시 데이터가 너무 커져 epoch를 2로 낮추고, 대신 lr을 4e-5로 올려서 테스트하였다.

dataset	epoch	batch_size	learning_rate	top k	정확도(%)
train dataset	3	8	5e-5	20	92.0833
train dataset + KorQuAD	2	5	4e-5	20	96.2500
	2	5	4e-5	40	98.3333

3.2.4. 문제점

실제 추론에 적용할 때 문서를 찾는 시간이 너무 오래 걸린다는 문제가 발생했다. 하나의 query 당 13분, 총 130시간의 소요 시간이 예상되어 결국 제대로 적용하지 못했다는 아쉬움이 있다. 추가적으로 예상되는 원인, 해결 과정 등은 개인 회고에 작성할 예정이다.

3.3. ColBERT

colBERT는 dense retrieval에서 일반적으로 사용하는 벡터 임베딩 기법과 sparse retrieval의 효율성을 결합한 모델이다.

training-indexing-searching-inference 4단계로 구성되어있다. RoBERTa-large 모델을 사용하였으며 Retrieval를 제외하고 모든 환경 세팅을 동일하다. inference 후 Eval score(validation dataset)를 기준으로 성능을 평가하였다.

Retrieval	Eval EM / F1 Score (%)
DPR	45.8333 / 52.8724
ColBERT	62.5 / 69.4714
BM25	61.6667 / 69.6068
Rerank	64.5833 / 71.1346

실험 결과, DPR보다 colBERT의 성능이 더 우수했다. 이는 DPR이 문서나 쿼리를 단일 벡터로 표현하는 것과 달리, colBERT는 각 토큰마다 임베딩 벡터를 생성하여 문서와 쿼리를 여러 개의 임베딩 벡터로 표현함으로써 더 풍부한 문맥 정보를 반영할 수 있기 때문이라고 생각된다. 또한 BM25와 유사한 성능을 보였으며, Elastic Search를 사용한 Rerank를 Retrieval로 사용했을 때보다는 성능이 낮았다. Elastic Search의 Rerank가 colBERT보다 더 나은 성능을 보인 이유는, Rerank 과정에서 검색 결과의 품질을 추가로 향상시키고, Elastic Search가 대규모 데이터에서 더 효율적이고 최적화된 Retrieval을 제공하기 때문이라고 생각한다.

3.4. Reranker

top k가 늘어나면 Retrieval의 성능이 개선되지만 그만큼 passage의 수가 늘어나 Reader의 성능이 악화된다. 성능 개선에는 동일한 top k에서 Retrieval의 정확도를 개선하는 방법과 동일한 Retrieval 정확도에서 top k를 줄이는 방법 두 가지가 있다. Sparse Retrieval를 Dense Retrieval로 변경해도 정확도 향상 폭이 그다지 크지 않으며 GPU 메모리의 한계로 매우 작은 batch size로 학습할 수밖에 없는 점 등 때문에 개선이 매우 어려웠다. 따라서 Reranker를 통해 정확도 손실을 최소화하며 passage의 수를 대폭 줄이고자 했다. Reranker는 Dongjin-kr/ko-reranker¹⁰를 사용하여 구현했다. 해당 모델은 BAAI/bge-reranker-large를 한국어 데이터에 대해 fine-tuned model이다. context의 순서가 정확도의 영향을 준다는 논문¹¹에 기반하고, msmarco-triplets¹²데이터셋을 Amazon Translate 기반으로 번역하여 활용하였다.

¹⁰ <https://huggingface.co/Dongjin-kr/ko-reranker>

¹¹ <https://arxiv.org/pdf/2307.03172>

¹² <https://github.com/microsoft/MSMARCO-Passage-Ranking>

Rerank BM25 (validation set)

top k	rerank_5 정확도(%)	rerank_4 정확도(%)	rerank_3 정확도(%)	rerank_2 정확도(%)	rerank_1 정확도(%)
20	90.0000	89.1667	88.8333	88.8333	75.4267
30	90.4167	90.0000	89.5833	86.2500	77.0833
40	90.4167	90.0000	89.5833	85.8333	76.2500
50	90.4167	90.0000	89.5833	85.4167	76.2500
60	91.2500	90.8333	89.5833	86.2500	76.6667
70	91.2500	90.8333	89.1667	86.2500	76.6667
80	91.2500	90.4167	89.1667	86.2500	76.6667
90	91.2500	90.4167	89.5833	86.6667	76.6667
100	91.6667	90.4167	90.0000	86.6667	76.6667

- top k가 증가할 수록 정확도가 상승하다가 이후 소폭 떨어지거나 정체되는 경향성을 보인다.

- BM25 retrieval 결과를 rerank할 때는 top k가 60일 때 일반적으로 성능이 좋았다.

Rerank Elasticsearch (validation set)

top k	rerank_5 정확도(%)	rerank_4 정확도(%)	rerank_3 정확도(%)	rerank_2 정확도(%)	rerank_1 정확도(%)
20	91.6667	90.8333	89.5833	85.8333	79.1667
30	92.9167	91.6667	89.5833	87.0833	79.5833
40	92.5000	92.0833	89.1667	87.0833	79.5833
50	92.5000	91.2500	89.1667	86.6667	79.5833
60	93.3333	92.0833	90.0000	87.5000	79.5833
70	92.5000	91.6667	89.5833	86.6667	79.5833
80	92.5000	91.6667	89.1667	86.6667	79.5833
90	92.5000	91.6667	89.5833	86.2500	79.5833
100	92.5000	91.6667	89.5833	86.2500	79.5833

- Elasticsearch retrieval 결과를 rerank할 때는 top k가 60일 때 가장 성능이 좋았다.

Reader	Retrieval	top k	rerank	정확도(%)	Eval EM / F1 Score (%)	public EM / F1 Score (%)	private EM / F1 Score (%)
roberta-large(epoch 3)	Elasticsearch	20	X	94.5833	59.5833 67.6911	60.0000 70.1700	60.8300 72.8800
roberta-large(epoch 3)	Elasticsearch	20	5	91.6667	60.4167 67.7189	63.3300 73.2900	61.6700 73.8400
roberta-large(epoch 3)	Elasticsearch	30	5	92.9167	60.4167 67.6043	63.7500 73.8800	62.7800 75.0800
roberta-large(epoch 3)	Elasticsearch	40	5	92.5000	60.4167 68.2571	64.5800 74.7900	62.5000 74.9300
roberta-large(epoch 3)	Elasticsearch	50	5	92.5000	60.8333 68.6738	63.7500 73.8700	63.3300 75.7800
roberta-large(epoch 3)	Elasticsearch	60	5	93.3333	60.8333 68.6738	62.9200 73.3400	63.3300 75.9700

- Eval(validation dataset) 점수 기준으로는 topk=60을 rerank한 결과가 가장 좋았다.

- Public(test dataset) 제출 점수 기준으로는 topk=40을 rerank한 결과가 가장 좋았다.

Retrieval 정확도와 Reader + Retrieval 성능 사이의 상관관계가 validation set을 이용한 eval 점수에서는 일관성을 보인다. public 점수에서는 Best top k가 40으로 k가 커지면서 성능 점수도 커지다가 다시 꺾이는 경향을 보인다. 각 실험에서 reranker가 동일한 개수의 passage를 뽑기 때문에 retrieval 정확도가 높은 rerank 60 to 5가 public 점수에서도 높을 것이라는 추정과 상반된 결과였다.

이는 validation set과 test set의 분포가 다르기 때문에 편향에 의한 결과라고 생각했다. 여기서 아쉬웠던 점은 그럼 validation과 test의 question의 주제나 분야를 파악하여 분포를 추정하는 등의 실험으로 더 나아가지 못하고 단순히 리더보드 상에 당장 눈에 보이는 public 점수를 eval 점수보다 더 신뢰했다. 그래서 top k가 커지면 올바른 passage를 가져올 확률이 높아지지만 그만큼 negative passage들이 더 유사해지기 때문에 Reader의 성능이 더 낮아지는 결과를 초래한 것이라는 가정을 세웠다. 비록 가정을 세우는 과정은 엄밀하지 못했으나 이 가정으로부터 새로운 실험 아이디어를 도출했다. **Reader의 성능이 개선되는 만큼 더 정확도 높은 Retrieval을 사용할 수 있다. Reranker를 구현한 이후부터는 Reader의 성능이 Retrieval의 성능까지 Driven한다. Reader의 성능이 매우 중요한데, 현재 학습하는 방식은 original context 하나만 학습하기 때문에 Test task에 비해 난이도가 쉽다. Reader가 여러 Context에서 정확한 Context를 찾는 능력은 현재 학습으로는 얻을 수 없다. context에 negative passage를 추가하는 방식으로 학습을 어렵게 한다면 점수 향상을 기대할 수 있을 것이다.** (negative passage 데이터 증강으로 이어진다.)

4. Ensemble

4.1. hard ensemble

하드 보팅 방식의 앙상블 기법을 구현하였다. 여러 모델의 예측 결과가 담긴 predictions.json 파일들을 읽어와 각 질문에 대한 모든 모델의 답변을 수집한다. 이후, 각 질문에 대해 가장 많이 나온 답변을 선택하는 방식으로 하드 보팅을 수행한다.

동점인 경우, 즉 여러 답변이 동일한 빈도로 나타난 경우에는 사전에 정의된 모델 우선순위에 따라 답변을 선택한다. 이를 위해 모델의 우선순위를 정의하여, 우선 순위가 높은 모델일수록 답변이 채택될 가능성이 높도록 하였다. 실험 결과, 대부분의 경우에서 점수가 향상되었다.

4.2. soft ensemble

예측 결과를 앙상블하여 최종 예측을 생성하는 소프트 보팅 방식의 앙상블 기법을 구현하였다. 모델들의 예측 결과가 담긴 nbest_predictions.json 파일들을 읽어와 동일한 질문에 대한 여러 모델의 응답을 수집한다. 이후, SequenceMatcher를 활용해 유사한 답변들을 그룹화한다. 그룹화된 답변들에 대해 소프트 보팅을 적용하여 각 그룹 내의 답변 확률을 합산하고 가장 높은 점수를 받은 그룹의 답변을 최종 예측으로 선택하였다. 여기서 선택된 답변의 점수가 조정가능한 임계값보다 낮으면 빈 문자열을 반환하여, 답변을 찾을 수 없는 문제에 대해서는 응답을 하지 않도록 하였다.

실험 결과, 거의 대부분의 실험에서 점수가 향상됨을 확인하였으나 일정 갯수가 초과할 시에는 F1점수만 향상하였고 EM점수는 큰 차이가 보이지 않았다. 모델의 다양성 부족, 유사도 임계값의 문제등이 의심되었다.

4.3. weighted ensemble

Soft voting의 EM 점수가 일정 모델 수를 초과했을 때 더 이상 향상되지 않는 문제를 해결하고자 weighted ensemble을 시도하였다. 각 모델의 EM 점수 차이가 근소하여 단순히 EM 점수만으로 가중치를 부여하는 것은 soft voting과 큰 차이를 보이지 않았다. 따라서 모델 간 가중치 차이를 더 명확히 하기 위해 다음 세 가지 방법을 시도하였다.


1. 지수 가중치: 원래의 EM 점수를 지수 함수에 적용하여 가중치 간 차이를 확대하였다.
2. 순위 기반 가중치: 모델의 성능 순위에 따라 가중치를 할당하였다.
3. 상대적 차이 강조: EM 점수를 0과 1 사이로 정규화한 후 제곱하여 상위 성능 모델의 영향력을 강화하였다.

그러나 이 세 가지 방법 모두 기존의 앙상블 방식보다 성능 향상을 보이지 않아 최종적으로 채택하지 않았다. weighted voting은 모델간의 성능이 큰 차이를 보일 때 유용한 앙상블 기법인데, 각 모델의 EM 점수 차이가 근소하였기 때문에 점수 향상에 기여하지 못했다고 생각된다.

4.4. merged ensemble

Exact Match (EM)의 평가방식은 모델의 예측과, 실제 답이 정확하게 일치할 때만 점수가 주어진다. 따라서 정답 단어가 포함된 올바른 구절이라도 점수를 획득할 수 없다. 또한 앙상블을 시도할 때 정확히 일치하는지를 기준으로 voting 할 때도 예측들이 같은 단어를 포함하는지 고려되지 않는다. 따라서 A가 B를 포함하면서 B로 시작한다면 같은 예측을 하였다고 판단하고 A를 B로 병합하는 과정을 거쳤다. 예를 들어 “김부캠 캠퍼”와 “김부캠”가 있다면, 이는 “김부캠”, “김부캠”으로 병합된다. 이후 단계는 hard ensemble과 동일한 과정을 거친다. 모델 예측 선정의 기준은 Public EM score가 일정 점수 이상인 모델 전부를 가져오고, 동일 빈도에서의 우선순위 또한 Public EM을 기준으로 정렬했다. 그러나 앙상블은 모델이 서로 독립일 때 성능이 개선될 여지가 크기 때문에 일정 성능 이상의 점수 모델을 가져오는 방식은 적절하지 못하며, 더 짧은 예측이 정답이라는 근거 또한 없기 때문에 위 방식은 좋은 결과를 내지 못했다.

제출결과

1	NLP_04 조		72.2200%	82.8000%	135	8h
---	-------------	---	----------	----------	-----	----

개인회고

김민서

프로젝트를 시작하면서 세운 목표

지난 문장간 유사도 분석(STS)프로젝트에서는 베이스라인 개발, 모델리서치 및 실험, 앙상블 파트에 집중하였다. 프로젝트 시작 후 베이스라인 개발을 우선시하다 보니, 데이터분석과 활용에는 많이 신경 쓰지 못했던 아쉬움이 있었다. 그래서 이번 프로젝트는 데이터 분석과 데이터기반 성능향상에 초점을 맞추고자 했다.

프로젝트에서 시도한 것에 대한 설명

제공 데이터 분석

우선 제공된 데이터분석을 진행하였다. 그 과정에서 그래프를 그릴 때 데이터의 분포와 데이터간 상관관계를 직관적으로 파악하기 위해 seaborn라이브러리를 사용하였고, 최대한 분석 목적에 맞는 그래프의 종류를 사용하려고 노력했다.

우선 데이터의 구조를 판단하고, train, validation, test 데이터들의 분포를 확인하고 비교하였다. 이는 train 데이터와 validation, test 데이터 간의 분포 및 논리적 유사성이 높아야 모델이 올바르게 학습하고 일반화된 추론을 할 수 있다고 판단했기 때문이다. 만약 train 데이터가 특정 패턴이나 주제, 분포에 집중되어 있는데 validation, test 데이터가 그렇지 않다면 모델이 그 차이를 제대로 학습하지 못해 오차가 커질 있기 때문이다. 히스토그램과 KDE 곡선을 함께 사용하여 train, validation, question 데이터셋의 context, question, answer의 text 길이분포를 비교하였고, 결측치와 중복치 같은 특이사항까지 확인하였다.

Wikipedia 데이터셋 역시 동일한 방향으로 분석을 진행하였다. 추가로 train과 validation 데이터셋의 context가 Wikipedia 데이터셋에서 가져온 데이터셋이 맞는지 확인하는 과정을 거쳤다. 이때 train, validation 데이터셋의 context와 Wikipedia 데이터셋의 text간 정확히 일치하는 것을 확인하였지만, title과 같은 요소는 다른 경우가 있었다. 하지만 실제 학습시에는 title을 제외한 context만을 사용하기 때문에 추가적인 처리는 진행하지 않았다.

데이터기반 성능향상 전략

성능을 향상시키기 위해 데이터 증강 및 클렌징, train-validation 데이터 재배치 등 다양한 성능향상 전략을 진행했다. 그 중 성능에 향상이 있었고, 신경을 많이 쓴 항목은 GPT API 기반 데이터 증강이다. context데이터 갯수를 증강시키는 것은 retriever의 성능을 낮출 수 있기 때문에 question과 answer에 대한 증강을 진행하였다.

추가 질의를 생성하기 위해 GPT API를 불러온 뒤 few-shot learning을 진행했을 때 문제가 발생하였었다. GPT가 생성형 모델이기 때문에, context안에 없는 answer을 생성하는 문제가 있었다. 이를 해결하기 위한 방법으로 question과 answer을 동시에 만드는 것이 아니라 context의 핵심 키워드를 추출한 후 이를 정답으로 지정, 이 정답을 도출할 수 있는 question을 생성하는 방식을 사용하여 문제없는 question-answer 데이터셋을 추가적으로 생성할 수 있었다. 또한 동의어 증강기법도 고려하였다. 간단히 question의 동의어교환과 문맥변경을 진행하는 것 이였고, 이 역시 GPT API를 사용하여 진행하였다.

초반에 계획은 있었으나 진행하지 못한 것도 있는데, 로컬 LLM기반 데이터분석 및 클렌징이다. 데이터가 매우 많아지면 일일이 눈으로 모든 데이터를 확인하는 것은 불가능에 가까워진다. 하지만 모든 데이터를 고성능 LLM에 API형태로 확인시키는 것은 비효율적이라고 생각했다. 오류를 확인하는 것은 생성하는 것에 비해 쉽고, 성능이 부족하더라도 여러 번의 반복으로 이를 보완할 수 있기 때문에 로컬 LLM을 통해 진행해보면 어떨까 하는 생각을 했고, LLaMa모델의 작동까지 확인하였다. 하지만 시간적 여유가 부족하여 우선순위가 밀려 진행하지 못한 아쉬움이 있다.

앙상블

앙상블 전략을 고도화하는 방법에 대한 고민이 많았다. 앙상블은 어떤 모델의 결과를 얼마나 섞을지 고민하는 weighted voting 방법론을 많이 썼었는데, 이로 인해 단순히 리더보드의 public 성능만을 성능지표로 보고 파라미터를 변경하는 반복적인 작업이 이루어지는 것이 과정이 아니라고 아니라고 생각했다. 따라서 좀 더 고도화된 방법을 고민하다 LLM을 적용하는 방법을 시도해보았다. MRC의 Q-A과정에 LLM을 사용하는 것은 생성형 모델 특성상 EM성능이 높지 않아 부적합하다고 생각했지만, 각각의 질문에 대한 여러 모델의 출력을 기반으로 LLM이 최적의 답변을 선택하도록 하는 앙상블 전략을 고려해 보았다. 이를 통해 단순히 가중치 기반의 합산보다는 보다 정교하고 상황에 맞는 답변을 생성해낼 수 있을 것이라고 판단했다. 예를 들어, 각 모델이 제공한 답변을 LLM에 입력하여 최종적으로 가장 적합한 답변을 선택하는 방식이다. 이러한 방법은 단순히 특정 모델의 결과에 의존하는 것이 아니라 다양한 모델의 강점을 활용하여 최적의 성능을 얻을 수 있다는 점에서 의미가 있다고 생각했다. 높은 성능이 나왔지만 실제 제출에는 반영하지 않았다. 금지된 가중치인 MRC_KLUE 데이터셋을 반영하지 않았다고 보장할 수 없기 때문이다.

아쉬운 점 및 다음대회에 시도할 것

프로젝트 초반에는 데이터를 중점적으로 분석하다 보니 베이스라인 코드에 소홀했던 부분이 아쉬움으로 남았다. 특히, 팀원들이 각자 베이스라인 코드를 디벨롭해 나가는 과정에서 코드의 복잡도가 증가했고, 그 결과 후반에 실험을 진행하기 전에 코드의 동작 원리를 이해하는

데 많은 시간이 소요되었다. 이러한 문제로 인해 실험 진행 과정에서 효율성이 떨어졌던 점이 아쉬웠다. 따라서 다음 프로젝트에서는 데이터 분석과 베이스라인 코드의 이해를 나눠서 동시에 진행하는 것을 목표로 하고 있다. 데이터를 깊이 이해하는 동시에 코드의 구조와 기능도 철저히 이해하는 과정을 병행함으로써 팀원 간의 이해도 차이를 줄이고, 실험의 일관성과 효율성을 높이고자 한다.

협업

가장 잘한 점을 하나 뽑는다면 바로 협업이라고 생각한다. 지난 대회에서는 각자의 코드를 사용하고 진행하는 실험이 공유가 잘되지 않았었는데, 이번 대회에서는 하나의 베이스라인 코드를 사용하고 실험 내용의 공유가 잘되었다. 아쉬운 점은 학습한 모델 파일이 제대로 공유되지 못했고 학습 내용 기록에 대한 규칙이 없어서 재현이 어려웠다는 점이다. 내가 한 실험조차 기록을 정확히 해두지 않아 재현하는데 시간을 낭비한 경우도 있었다. 이번 대회를 통해 기록과 기록의 규칙의 중요성을 다시금 생각하게 되었다. 하지만 이번 프로젝트를 통해 우리의 손발이 척척 맞아가는 기분이 들어서 좋았다.^^

데이터

외부데이터를 학습에 사용해보았다. ETRI 데이터는 WIKI CONTEXT를 추가하면서 한땀한땀 정성스럽게 수정하였다. 300개였지만 내 기준 질 좋은 데이터였다. AI HUB의 일반 상식 데이터도 학습에 사용했다. 약 10만개의 데이터로 양이 많은 데이터였다. 데이터를 증강할 때 중요한 것이 많은 양의 데이터거나 좋은 질의 데이터라는 이야기를 많이 들었는데, 이번 실험에서는 적용되지 않는 이야기인가 고민하고 또 데이터란 무엇인가하고 혼란에 빠져있었다. 하지만 private score가 공개되고 두 데이터로 학습한 결과 모두 성능이 오른 것을 확인할 수 있었다. 많은 양의 데이터와 좋은 질의 데이터는 성능을 올린다. 하지만, 이는 task마다 다르고 또 test마다 다른 것일까? 여전히 의문으로 남아있다. 다음 대회에서는 public score에만 의존하는 것이 아닌 다양한 Test dataset을 사용해야겠다.

Retrieval

가장 큰 아쉬움으로 남아있는 부분이다. 첫번째 splade를 적용해보고 싶었는데 구현에 실패하였다. 두번째 ColBERT 실험을 너무 늦게 시작했다. 프로젝트의 마감일이 다가왔을 때 ColBERT를 사용하여 실험을 진행하였고 BM25만큼 성능이 잘 나오는 것을 확인해서 제출해보지 못한 것과 코드를 베이스라인 코드에 이식하지 못한 것이 아쉬움으로 남아있다. 그래도 잘한 점은 다양한 Retrieval로 실험을 해봤다는 것이다. 하지만, DPR과 BM25의 실험 결과를 비교해보지 않은 것이 아쉬웠다.

Reader

잘한 점은 모델의 성능을 높이기 위해 다양한 시도를 했다는 점이다. 모델의 forward 코드를 확인하며 dropout을 적용하기도 했고 Reader의 성능을 높이기 위해 다양한 데이터로 학습을 시키고 4개의 형태소 분석기 앙상블하여 후처리를 적용하기도 했다. 다만 아쉬운 점은 대부분이 전 기수나 논문의 실험 결과로 레퍼런스를 얻어서 진행한 실험이었다. 창의적인 아이디어로 실험을 하고 싶었는데 이 부분은 진행하지 못하였다.

베이스라인 코드

이번 대회에서는 템플릿화하는 것이 목표였는데 코드를 많이 수정하지 못했다. 베이스라인 코드에 중복되는 부분이 많은 걸 확인했지만 코드를 수정하는 것보다 실험하는 것에 우선순위를 둔 탓에 베이스라인 코드에 중복되는 코드가 많이 남아있다. 다음 대회부터는 시간이 오래 걸리더라도 코드의 중복을 줄이고 템플릿화하는 것이 목표이다.

다음 대회에서 할 시도들

1. 새로운 기능들 사용해보기

SFT 기법을 사용한 peft LoRA를 사용해서 일부 파라미터만을 튜닝함으로써 모델의 성능을 적은 자원으로도 높게 유지해보고 싶다.

2. Gradient Cache

이번 대회에서 배치 사이즈를 늘리기 위해 사용한 gradient accumulate step을 사용했다. 멘토 세미나를 들으면서 이 방법이 학습 속도를 저하하고 불균형한 업데이트 타이밍으로 누적 횟수가 많아지면 그 타이밍에 대한 불균형이 학습 품질에 영향을 미칠 수 있다는 것을 알게 되었다. 이에 대한 대안으로 gradient cache를 알게 되었고 다음 프로젝트에서는 gradient cache를 사용해보고싶다.

3. 더 다양한 논문 읽기

이번 대회에서는 실험과 관련된 논문을 많이 읽지 못했다. 2번의 내용도 관련 논문을 찾아봤다면 프로젝트 도중에 미리 알고 이를 실험에 적용할 수 있었을 것이다. 다음 프로젝트에서는 관련 논문을 많이 읽어야겠다.

4. 이론 공부를 소홀히 하지 않기.

매 프로젝트마다 다짐하지만 잘 되지 않는 것 같다. 이번 프로젝트에서는 이론 공부를 철저하게 해야겠다.

양가연

학습 목표

- 대회에서 높은 성적을 거두는 것도 중요하지만, 그보다 실험을 진행하고 나서의 분석에 치중하고자 했다. 성능이 개선된 이유와 하락한 이유를 찾는 과정이 공부가 될 것이라고 생각했다.

수행한 작업 및 느낀점

1. Retriever 개선

BM25

ODQA 질문들은 특정 키워드에 치중이 되어있는 경우가 있다. 이에 맞게 Sparse 리트리버는 특정 키워드 매칭에 특화되어 있어, 질문과 직접적으로 관련된 문서를 효과적으로 검색할 수 있어 이번 task에서 좋은 성능을 낼 수 있었다고 생각한다.

느낀점 및 개선 사항 TFIDF, BM25 에 대해 공부하며 코드를 구현한 점이 잘한 점이라고 생각한다. 하지만 Sparse와 Dense embedding을 함께 사용하는 Hybrid Retrieval를 순수 구현해보지 못한 것이 아쉽다.

2. Reader 모델 개선

모델의 기계 독해 능력을 향상 시키기 위해서 Custom Layer추가 전처리,데이터 증강, 외부 데이터를 이용해 fine-tuning, 하이퍼파라미터 개선, distillation 등을 시도 하였다.

Custom Layer 추가

기존 RobertaForQuestionAnswering의 경우 forward 단계에서 linear layer만을 통과하기에 CNN,LSTM,Bi-LSTM,MLP 등의 layer를 추가하여 복잡한 관계의 데이터들 또한 학습될 수 있도록 시도하였다. **느낀점 및 개선 사항** 성능 향상까지 이어지지는 못하였으나 모델 구조에 맞춰 custom Layer를 추가해보는 경험을 하였다. Custom Layer를 추가하는 과정에서 기존 모델의 구조와 작동 방식을 이해할 수 있었다. 앞으로 다양한 모델을 다루고 최적화하는 데 조금 더 어려움 없이 task를 수행할 수 있을 것 같다.

Distillation 시도

RoBERTa-large를 KorQuAD로 fine-tuning한 모델을 교사 모델로 선택하고 RoBERTa-large에 CNN 레이어를 추가한 custom 모델을 학생 모델로 선정하여 Distillation을 시도하였다.

느낀점 및 개선 사항 학생 모델의 구조가 distillation 성공에 큰 영향을 미친다는 것을 알게 되었다. 단순히 성능이 좋은 교사 모델을 선정하는 것만이 중요한 것이 아니고, 학생 모델이 지식을 효과적으로 전달받을 수 있는 구조를 설계하는 것이 중요하다는 것을 깨달았다.

3. 앙상블 코드 추가

하드 보팅, 소프트 보팅, weighted 보팅 등 여러 앙상블 방법을 구현해봄으로써 각 기법의 특성과 동작 방식을 이해하게 되었다.

느낀점 및 개선 사항 동점 처리나 유사도 임계값 설정 등을 추가하면서 조금 더 나은 앙상블 방식을 고민하며 구현하였다. 또한 소프트 보팅을 적용하고 모델 수 증가에 따른 성능 변화를 관찰하면서 한계를 극복하기 위해 weighted 보팅 구현을 시도한 점이 좋았다.

대회를 마무리하며 느낀점

- 실험에 대한 사후 분석을 철저히 해야겠다고 생각했으나 여전히 부족한 점이 많아 아쉬웠다. 점수 향상에 너무 치중하다 보니 철저히 사후분석을 하지 않고 적당히 하고 넘어갔던 것 같다. 특히, 저번 프로젝트 때도 데이터를 하나씩 뜯어보면서 개선 방향을 생각하자고 다짐했던 부분이 이루어지지 않아 쉽다.
- 베이스라인 코드를 하나하나 뜯어보는 것보다 기능을 덧붙여 가면서 이해하는 것이 더 효율적인 방향이라고 생각해 그런 방식으로 대회를 진행하였다. 베이스라인 코드의 일부는 내가 채택한 방식으로 이해가 되었으나, 일부는 이해도가 미흡했던 것 같아 아쉽다. 기능을 덧붙여 가면서 이해하는 방식은 좋으나, 사후 분석을 조금 더 철저히 해서 전체적인 코드의 구조에 대한 이해도를 더 높여야겠다.
- 수업과 대회의 비율을 비슷하게 맞추려고 다짐을 했지만, 이번에도 여전히 프로젝트에 더 많은 시간을 소비했다. 다음 프로젝트 때는 미리 강의를 들어버리고 프로젝트를 시작하는 것도 좋지만 대회를 진행하면서 부족한 개념들은 돌아가서 다시 듣고, 개선 방향을 찾는 도구로 사용하는 것이 더 나은 방향성이라고 생각한다.
- 허깅페이스를 사용하지 않아 팀원들의 모델을 자유자재로 사용하지 못한 점이 아쉽다. 다음 프로젝트나 대회에서는 꼭 모두 허깅페이스를 사용하여 협업의 효율성을 더 높이고 싶다.
- wandb 사용이 미흡해서 실험에대한 기록이 철저히 이루어지지 않은 점이 아쉽다. 다음 대회에서는 베이스라인에 wandb를 추가하여 실험에 대한 기록이 원활하게 이루어질 수 있도록 하고 싶다.

이에서

두번째 프로젝트인 ODQA(Open-Domain Question Answering)는 결과와 과정 모두 성공적이다. 이번 프로젝트에서 가장 중요하게 생각했던 점은 지난 프로젝트에서 아쉬웠던 부분을 개선하는 것이었다. [지난 프로젝트에서 배우고 개선하기]에서는 협업 관점에서 좋은 프로젝트를 만들기 위한 나름의 해결방안을 찾아가는 과정을 회고에 담았다. [프로젝트 성능 개선하기]에서는 실제 성능 관점에서 개선을 위한 작업 과정을, [아쉬웠던 점을 바탕으로 다음 프로젝트 계획하기]에서는 새로 발견한 아쉬웠던 점과 개선 목표를 담았다.

[지난 프로젝트에서 배우고 개선하기]

첫 번째는 지난 프로젝트에서 실험 및 작업 내역 공유와 협업이 원활하지 못하고 서로 실험하는 코드 환경이 달라 재현에 어려움이 있었다. 이런 문제를 극복하기 위해 매니저 역할의 필요성을 느꼈다. 나는 **PM 역할을 맡아 기본적인 협업에 대한 룰부터 작성하기 시작했다. 팀 프로젝트 노션 페이지를 정리하고, 역할 분배와 문서화, 깃허브 이슈 관리 및 브랜치 전략을 세웠다. 이후 하나의 프로젝트를 달성하기 위해 필요한 마일stones을 설정하고, 각 마일stones을 달성하기 위한 하위 작업카드를 구상했다.** 이렇게 협업을 위한 초석을 마련했고 우리는 작업카드를 통해 서로의 실험과 구현을 자연스럽게 공유할 수 있었고, Git 전략을 통해 하나의 코드로 작업할 수 있었다.

두번째는 지난 프로젝트에서 실험을 구상할 때마다 코드를 변경해야하는 일이 잦았기 때문에 이를 개선해보고자 이번에는 베이스라인 코드 템플릿화 작업에 참여했다. 최대한 실험하기 좋은 템플릿을 만들기 위해 내부 구현을 몰라도 되는 부분인 logging, seed 설정 등과 같은 부분을 util로 빼고, 다양한 Reader와 Retrieval을 구현하여 사용하기 위해 train과 inference에서 Model과 Retrieval을 호출하는 부분을 추상화했다. 또한 주피터 노트북을 관리하는 폴더를 만들어 데이터 분석 및 전처리 결과, 새로 구현한 코드 사용 방식에 대한 예시, 실험에 대한 간단한 재현 방법 등을 담아 마치 문서처럼 관리하였다.

세번째는 지난 프로젝트에서 GPU 서버를 새로 할당할 때마다 개발 환경 세팅 작업이 반복적으로 일어나는 점이 비효율적이고 각자 개발 환경을 세팅하다보니 마주치는 에러도 각각이었다. 그래서 내가 인프라 담당자 역할을 맡아 개발 환경 세팅 스크립트를 개발하여 팀원 모두가 동일한 환경에서 작업할 수 있도록 했다. 세팅에 걸리는 반복 작업도 사라졌고 에러도 한번만 해결하면 다른 사람들도 쉽게 해결할 수 있어 개발 환경에서 겪는 어려움을 대폭 개선할 수 있었다.

[프로젝트 성능 개선하기]

data cleansing

정량적 데이터 분석도 중요하지만 정성적으로 분석하는 것도 중요하다고 생각해서 데이터를 열어보며 직접 눈으로 확인하였다. 데이터를 하나씩 까보며 오류는 없는지, 특징적인 부분은 없는지, 신뢰할 만한 데이터인지 판단하였다. 그 중 몇 가지 오류를 발견할 수 있었다.

- 연도를 물어보는 질문에 대한 답변이 **1989**과 **1989년** 두가지 형태가 혼재되어 있었다. Context에 '년'까지 존재한다면 **1989년** 형태로 답변하도록 통일하고, Context에 숫자만 있는 경우에는 숫자 그대로 답변하도록 했다.
- 거의 모든 질문이 물음표로 끝나는데, 몇몇 질문에는 빠져있어 물음표를 붙여주었다.
- 답이 틀린 문제가 있어 수정했다. 데이터가 틀리지 않고 내가 틀렸을 가능성도 있기 때문에 ChatGPT를 이용하여 교차검증 후 틀린 이유와 고친 정답을 팀원들에게 공유하여 확인 후 수정했다. 그 중에는 답을 찾을 수 없는 문제도 있었는데, 이걸 일단 제거하기로 했다. 이후 답변 가능성 처리(No Answer)에 대한 논의를 따로 진행하며 데이터와 모델에 손봐야할 점이 너무 많고 점수 항상 폭이 크지 않을 것이라고 판단하여 이번 프로젝트에서 고려하지 않기로 결정했다.

Retrieval & Reranker 성능 분석 및 구현

Elastic Retrieval과 Reranker를 구현하고 성능 분석을 담당했다. Reader와 Retrieval을 함께 구성한 실험이 실제 테스트 결과와 유사하다는 장점이 있지만 순수 Retrieval의 성능을 확인하기 어렵다는 단점이 있다. 그래서 실험을 독립적으로 구상하고 각 retrieval마다, top k마다 실험 결과를 표로 정리했다. 성능 분석 결과를 바탕으로 다양한 가설을 세울 수 있었고, Retrieval 성능 분석 → Reranker 구현 → Reranker 성능 분석 → negative passage 데이터 증강 등 연쇄적으로 실험을 꾸려나갈 수 있었다. 근거를 가지고 출발한 실험들은 대부분 성공적이었으며, retrieval과 reranker 성능 분석 부분이 이번 프로젝트에서 가장 잘한 부분이라고 생각한다. 관련 내용은 팀 프로젝트에 더 자세히 작성하였다.

negative passage argument

Train 단계에서는 하나의 passage만으로 학습하는데 비해 실제 Test 단계에서는 여러 passage 중에서 정답을 찾아야 한다. 모델이 여러 passage, 즉 긴 context에서 필요한 정답을 추출하는 능력을 학습하지 못하는 상황이라고 생각했고, train dataset에 retrieval과 reranker로 가져온 negative passage를 추가한 데이터 셋을 구축했다. train 단계에서의 EM, F1 점수는 더 낮아졌지만 이후 Validation 및 Test 단계에서의 EM, F1 점수는 큰 폭으로 상승했다.

[아쉬웠던 점을 바탕으로 다음 프로젝트 계획하기]

추상화와 의존성 분리

아쉬웠던 점 중 하나는 프로젝트를 진행하며 Reader나 Retrieval의 구현체가 하나씩 추가될 때마다 전달해야 할 인자가 BaseModel과 달라졌는데 이런 부분이 추상화와 의존성 분리를 어렵게 만들었다. 만약 리팩토링을 진행한다면 새로운 구현체가 생겨도 train과 inference의 코드는 변경하지 않을 수 있는 방법을 고민할 것이다. 인자들을 dictionary 객체 내부에 숨겨 전달하고, 각각의 구현체에서 그 객체 내부에 필요한 인자들이 있는지를 체크하는 함수를 추가하는 방식 등을 적용해볼 것이다.

Public Score에 매몰되지 않기

negative passage argument로 성능 향상을 경험한 뒤 전이 학습을 통해 외부데이터 - 증강 데이터 - 원본 데이터 와 같은 순서로 학습을 구성하는 방식의 실험을 추가로 진행했다. 이때 validation 점수는 큰 차이가 없었으나 제출했을 때 Public Score가 크게 떨어졌다. 나는 당연히 해당 실험과 가설이 잘못되었을 것이라고 생각하고 실험을 마무리했지만 실제 대회 마무리 후 확인했을 때 Private Score는 올랐다. 다른 실험들에서도 Public Score를 train 점수, validation 점수와 함께 종합적으로 평가할 여러 지표 중 하나라고 생각하지 않고 100% 맹신했던 것 같다. 더 많은 실험을 진행할 수도 있었고 가능한 성능 개선의 폭도 높았는데 아쉬웠다. 이런 태도는 Public Score에 fitting한 모델을 만들게 될 가능성이 높기 때문에, 다음 프로젝트에서 항상 유념해야 할 부분이다.

Tokenizer와 Unknown Token 확인하기

Tokenizer의 성능이 Retrieval간 성능 차이에 영향을 주었을 것이라고 추정했다. 그렇다면 실제로 토크화된 문서를 직접 뜯어보면서 어떤 단어들이 UNK이 되었는지, 어떤 문서에 많은지 등을 직접 파악했어야 했다. 그러나 프로젝트 과정에서 이 부분을 생각하지 못하고 놓쳤다. 내가 이번 프로젝트에서 가장 실수했던 부분이고 가장 아쉬운 점이다. 오히려 정확도 분석보다 토큰들을 직접 확인해보는 것이 더 중요한 부분이라고 생각이 든다. 저번 프로젝트보다 더 깊이있는 실험과 가설 검증 과정을 거쳤지만 어느정도 성능이 나오면 쉽게 만족해버리거나 어느정도 성능이 나오지 않으면 쉽게 포기한 경향이 있다. 다음 프로젝트에서는 쉽게 결론 짓지 않고 끝까지 파고들고 싶고, 반드시 눈으로 확인할 수 있는 모든 부분을 확인하고 싶다.

[프로젝트]

이번 프로젝트는 MRC(Machine Reading Comprehension, 기계독해)라는 이루어진 QA 모델이었다. 이것은 주어진 대량의 텍스트 문서 중, 올바른 문서를 찾아 그 안의 텍스트 내용에서 답변을 탐색하는 과제였다.

Klue mrc dataset의 사용과 해당 데이터를 사전학습한 모델 외에는 크게 제한이 없는 대회였다. 우리는 데이터 증강, 외부 데이터 사용, 모델 튜닝, 다양한 retrieval의 사용과 튜닝, 마지막으로 앙상블을 통해 점수를 올릴 수 있었다. 우리는 End2End의 방식도 좋지만, 각자 하고 싶은 파트를 다양하게 나누어 진행하기로 하였고, 그 부분에는 제한을 두지 않기로 하였다.

그래서 나 또한 다양한 시도를 해보려 하였고, 이번엔 특히 Dense Retrieval 구현에 많은 시간을 쏟아부었다.

1. 나는 무엇을 수행하였는가?

가장 먼저 역할 분담을 하면서 나는 모델 쪽을 맡았다. 처음에는 모델의 사전 조사 및 리서치를 담당하였다. 그리고 조사한 모델에 대한 튜닝을 조금 진행하다가, Dense Retrieval 구현을 담당하게 되어 그쪽을 주로 작업하였다. Dense Retrieval의 경우, 제공된 baseline 코드에는 아예 없었기 때문에 실습 코드를 가져와서 알맞게 진행하여야 되었다. 개인적으로 직접 모델을 짜는 느낌이라 많이 어려움을 겪었던 것 같다.

2. 어떻게 수행하였는가?

우선 모델의 사전 조사 및 리서치 부분이다. 모델의 사전 조사는 여러 가지 모델의 종류를 먼저 찾아보았다. 대표적으로 BERT, ELECTRA, RoBERTa 등이 있다. 그리고 나는 ELECTRA 모델을 집중적으로 들어갔다. 구글의 논문과 해당 논문을 정리한 블로그들을 참고해 정리하고, 다른 팀원들에게 해당 내용을 간단히 설명해주는 방식으로 진행하였다. 최종적으로는 나는 electra의 2개의 모델 kykim/electra-kor-base와 멘토님의 Koelectra 모델을 사용하기로 하였다. 다만 아쉬운 점은 모델의 튜닝을 많이 해보지 못하고, Dense Retrieval에서 작업에 너무 매달렸다는 점이다.

위에서 말했듯 나는 Dense Retrieval의 구현과 튜닝을 또한 맡게 되었는데, 이 부분이 이번 프로젝트에서 가장 힘들었던 것 같다.

방식은 Dense Encoder를 학습하고, query & passage Encoder를 각각 저장한다. 학습이 끝난 모델을 평가하기 위해, top-k를 뽑아와 정답 개수, 오답 개수, 정답율을 확인하는 ipynb를 만들어 체크하였다. 후에 Retrieval에서 저장된 모델을 가져와 embedding하여 문서를 찾는 방식으로 진행하였다.

문제점은 학습과 retrieve에서 각각 존재했다.

Train

본래 데이터셋에서는 오류가 발생하지 않았으나, KorQuAD와 합친 데이터셋에서는 오류가 발생했다. 마지막 batch에서 query와 passage의 tensor 크기가 동일하지 않아 내적을 할 수 없다는 내용이었다.

이 부분은 데이터의 수가 달라지면서, 본래의 batch size로는 나누어 떨어지지 않아 크기가 달라지는 문제였다.

이에 대한 해결책은 둘이었다.

- DataLoader에서 drop_last=True로 설정해준다. → 마지막 batch를 무시해준다.
- batch size를 data 개수의 약수로 설정해준다.
- 나는 데이터가 일단 모두 학습했으면 하는 생각에 batch size를 약수로 설정해주고, 혹시 모를 오류를 대비해 drop_last=True로 설정해주었다.

Retrieve

validation의 데이터셋으로 추론할 때는 문제가 없었으나, wikipedia 데이터셋으로 추론할 때는 130시간이 걸리는 말도 안 되는 소요 시간이 걸렸다.

이 문제에 대하여 3가지를 문제점을 예상했었다.

- 중간에 list로 변환해준 뒤 데이터를 저장해주고, tensor를 생성하여 내적을 해준다. 이 tensor 생성 과정에서 시간이 소요된다고 추측하였다.
 - 하지만 이것은 오히려 소요 시간이 증가하였다.
 - 찾아보니 list.append()는 기존 리스트 끝에 데이터를 추가하는 작업만 한다.
 - 반면 stack은 단순히 기존 tensor에 데이터를 추가하는 것으로 끝이 아닌, 새로운 메모리 공간을 할당하여 새 tensor를 생성한다고 한다.
 - 따라서 이 과정은 O(n)의 시간 복잡도를 가진다. (여기서 n은 Tensor의 요소 수)
- 그렇다면 생성될 크기의 Tensor를 처음에 만들어주고, 그 자리에 넣어주는 방법을 택해보았다.
 - 이것은 기존보다 소요 시간이 2시간 정도 늘어났다.
 - 큰 차이는 나지 않더라도 시간이 줄어든 것이라 예상했으나, 되려 늘어났다.

- 아직 그 이유는 찾지 못하였다.
- (가장 큰 원인으로 파악되는 부분) 우리의 코드는 query를 하나씩 불러올 때마다 passage를 embedding해주고, wikipedia를 불러오는 과정으로 되어있었다.
 - 결국 passage의 embedding을 600번 하고, wikipedia를 600번 불러온 것이다.
 - 해당 dense retrieval 코드는 실습 코드를 채용한 것인데, 조교님이 코드를 짜셨을 것이라는 생각에 맹신한 결과였다.
 - 너무 맹신한 나머지 코드를 꼼꼼히 살펴보지 않았기에 발생했다.
 - 이 부분은 리더보드가 거의 마감하기 직전에 생각나 제대로 테스트 해보지 못하고 끝났다. 따라서 코드 제출도 해보지 못한 부분이기 때문에 추가로 수정하지는 않았다.

3. 대회의 소감

2번째 프로젝트가 끝났다. 가장 좋았던 점은 전 프로젝트에서 피드백한 부분이 수정되었다는 점이다. 팀원이 절차가 필요하다 생각하여 노션 페이지, git 규칙들을 재정립하였고 그것에 따라 진행하니 훨씬 편하게 진행되었다. 끝나고 나서도 서로에 대하여 어떤 점은 좋았고, 어떤 점은 아쉬웠다고 간단히 회고를 하였는데 이러한 과정이 너무 좋았다. 그리고 너무나 어색했던 Git에 조금 더 친숙해진 것 같다. commit 하나하나가 어색하고, PR, 리뷰 등 너무 어색했는데 이번 프로젝트로 조금은 친숙하게 된 것이 뿌듯하다.

4. 아쉬운 점

매일매일 정리하자! 내가 무엇을 했는지, 어떤 문제를 맞았는지, 어떻게 실패했는지, 해결했는지를 정말 간단하게라도 정리하는 과정의 필요성을 느꼈다. 사실 위에서의 해결 방법도 간단히 정리를 해보다 떠올랐기 때문에 그저 코드만 붙잡고 시간을 끄꽂 소요하기 보다는 이렇게 하는 게 더 좋다는 것을 느꼈다.

홍성재

- 모델 개선 방식 및 한계점, 배우게된 점

Reader의 성능을 개선하고자, NoAnswer에 대한 처리를 다루는 매커니즘을 직접 구현하려 시도해보았으나, 결과적으로는 구현해내진 못했다. 시도는 좋았다고 생각하기로 했다.

다만 구현을 끝까지 해내지 못한 이유는, 다음과 같다고 생각한다.

- 기본적인 구현 능력 및 흐름 파악 능력 부족
- 팀원들간 PR 및 변경사항 숙지 미흡

이론과 기본적인 코드 구현에 대해서는 평소에 꾸준히 노력하는수밖엔 없다. 당장 해결할 수 있는 문제가 아님. 그리고 베이스라인 및 팀원끼리 공유되고있는 코드는, 다음프로젝트부터는 아래와 같은 식으로 매꿀 것이다.

- 베이스라인 공개 시, 당일은 베이스라인 파악에만 집중
- 팀원들 공유 내용 및 PR 잘 파악해두기 > 한번정도는 직접 그대로 구현해보기

(귀찮다고 자꾸 미루면 안됨. 바로바로 해놔야 이후에 스무스하게 진행할 수 있음)

- 교훈 및 다음 프로젝트 목표

- 근거에 기반해서 고민을 많이 한다면, 그만큼 많은 인사이트를 얻을 수 있는것같다. 다음 프로젝트때는 가설 > 검증 > 근거 추론 이 사이클을 반복하고, 그 과정에서 알게된점을 잘 기록하자.

- 조금함을 좀 내려놓고, 프로젝트 자체에만 몰입하자. 이것저것 하다보면 둘 다 못하게되어있음
- 질문을 많이하자