

MRC Wrap-up Report

NLP-03조: 엔엘핑(NLPing)

팀원 이름: 육지훈 전진 정준한

허윤서 이수진 이금상

1. 프로젝트 개요

A. 개요

"서울의 GDP는 세계 몇 위야?", "MRC가 뭐야?" 우리는 궁금한 것들이 생겼을 때, 아주 당연하게 검색엔진을 활용하여 검색을 합니다. 이런 검색엔진은 최근 기계독해 (MRC, Machine Reading Comprehension) 기술을 활용하며 매일 발전하고 있는데요. 본 대회에서는 우리가 당연하게 활용하던 검색엔진, 그것과 유사한 형태의 시스템을 만들어 볼 것입니다.

MRC란?

- 우리는 궁금한 것들이 생겼을 때, 아주 당연하게 검색엔진을 활용하여 검색을 합니다. 이런 검색엔진은 최근 기계독해 (MRC, Machine Reading Comprehension) 기술을 활용하며 매일 발전하고 있는데요. 본 대회에서는 우리가 당연하게 활용하던 검색엔진, 그것과 유사한 형태의 시스템을 만들어 볼 것입니다.

ODQA란?

- Question Answering (QA)은 다양한 종류의 질문에 대해 대답하는 인공지능을 만드는 연구 분야입니다. 다양한 QA 시스템 중, Open-Domain Question Answering (ODQA) 은 주어지는 지문이 따로 존재하지 않고 사전에 구축되어있는 Knowledge resource 에서 질문에 대답할 수 있는 문서를 찾는 과정이 추가되기 때문에 더 어려운 문제입니다.

대회 특징	설명
대회 주제	네이버 부스트캠프 AI-Tech 7기 NLP 트랙의 level2 도메인 대회.
대회 설명	본 ODQA 대회에서 만들 모델은 two-stage로 구성되어 있다. 첫 단계는 질문에 관련된 문서를 찾아주는 "retriever" 단계이고, 다음으로는 관련된 문서를 읽고 적절한 답변을 찾거나 만들어주는 "reader" 단계다. 두 가지 단계를 각각 구성하고 그것들을 적절히 통합하게 되면, 어려운 질문을 던져도 답변을 해주는 ODQA 시스템을 직접 만들어야 한다. 본 대회는 더 정확한 답변을 내주는 모델을 만드는 팀이 좋은 성적을 거두게 된다.

대회 기간	9월 30일 (월) 10:00 ~ 10월 24일 (목) 19:00				
데이터 구성	분류(디렉토리 명)	세부 분류	샘플 수	용도	공개여부
	train_dataset	train	3952	학습용	모든 정보 공개 (id, question, context, answers, document_id, title)
		validation	240		
	test_dataset	validation	240 (Public)	제출용	id, question 만 공개
			360 (Private)		
평가지표	Exact Match (EM), F1 Score, (EM 기준으로 리더보드 등수가 반영되고, F1은 참고용으로만 활용됩니다.)				

B. 환경

(팀 구성 및 컴퓨팅 환경) 6인 1팀, 인당 V100 서버를 VSCode와 SSH로 연결하여 사용

(협업 환경) Notion, GitHub, Wandb

(의사 소통) 카카오톡, Zoom, Slack, Discord

2. 프로젝트 팀 구성 및 역할

팀원 명	역할
육지훈	EDA, 하이퍼파라미터 서치, 성능 검증 코드 제작, Inference 후처리
전진	EDA, 리트리버 실험 설계, 리트리버 성능 개선
정준한	EDA, 코드 모듈화, BM25 추가, 도메인 적응 코드 추가
허윤서	Retrieved data EDA, Cross, Bi-encoder DPR & Re-rank
이수진	BM25 Plus추가, Pre-trained 모델 실험, 리트리버 성능 실험
이금상	EDA, Pre-trained 모델 실험, 리트리버 성능 실험

3. 프로젝트 수행 내역

A. 베이스라인 코드 모듈화

아래와 같이 **config**파일을 지정하여 실험할 수 있도록 모듈화를 진행

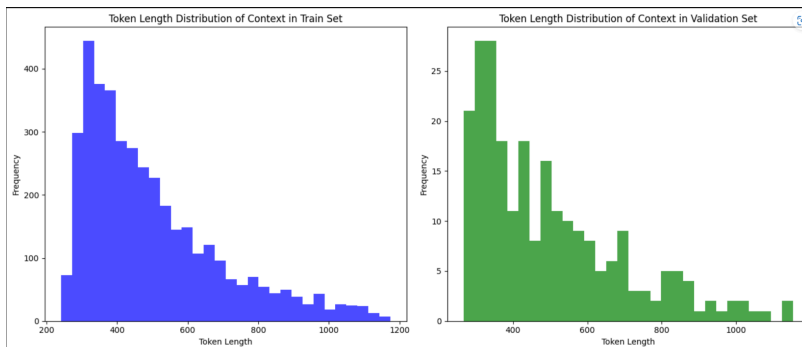
```
$ python train.py --config_path config/base_config.yaml
```

B. EDA

- Reader Data

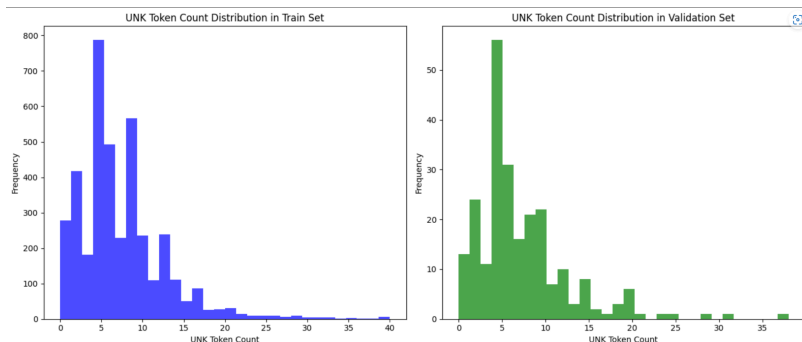
Reader 모델은 여러가지 모델로 테스트해본 결과, **roberta-large**가 가장 성능이 좋다고 판단하여 **roberta-large** 토크나이저로 토큰화하고 EDA를 진행하였다.

i. 토큰길이 분포



241~1174개의 토큰을 가지는 문서들로 구성되어있었고 평균적으로 **497**개의 토큰을 가짐을 확인하였다.

ii. 문서당 UNK 개수 분포



문서 1개당 **UNK**가 몇개 포함되었는지 확인하였고 평균적으로 문서 1개당 **7**개의 **UNK**토큰을 가짐을 확인했다. 최대 40개까지 있는 문서도 존재하는 것을 확인하였다.

iii. UNK 단어 확인

UNK토큰의 개수는 **train**에서 총 28279개, **validation**에서 1731개가 존재하였고, 무슨 단어가 **UNK**로

인식되었는지 확인해보았다.

확인결과, \가 28279개의 전체 UNK 중에 22808개를 차지하였고, 나머지는 -와 한자로 구성되어있었다.

train_unk_df.head(30)			validation_unk_df.head(30)		
	word	count		word	count
0	\	22808	0	\	1370
84	-	56	1	출라	7
24	李	30	206	μm	7
733	臺	26	122	-	4
720	頭	25	177	玄	4
485	臨	21	255	戟	3
255	呂	20	165	永	3
140	皇	19	24	像	3
729	半	18	2	출라의	3
1488	藩	18	20	。	3
623	郡	16	194	流	3
224	像	16	96	웃맨	3
73	国	16	39	스위 초는	3
439	里	16	66	济	2
746	領	16	91	프뢰티도르	2
665	衣	16	249	카르마놀라는	2
525	陳	15	227	트리베제	2
298	秦	15	50	肉	2
767	侯	14	54	樂	2
156	加	14	79	呂	2
359	塔	14	162	交	2
282	홋카이도	14	52	聯	2
1717	출라	14	51	黑	2
444	遷	13	82	李	2
250	座	13	222	遠	2
3	背	12	87	座	2
469	毛	12	77	妙	2
383	뫼라는	12	104	進	2
837	封	12	102	校	2
4	冠	12	103	尉	2

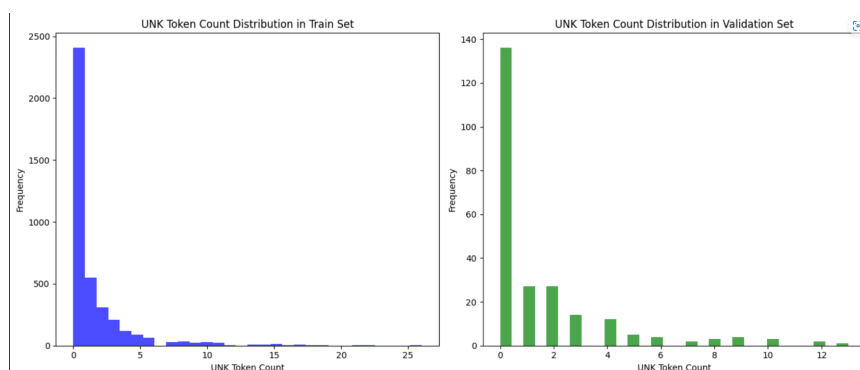
확인해본 결과, \n의 줄바꿈 기호가 \와 n으로 구분된다는 것을 확인하였다.

\n의 문제점

\n\n미국 → [UNK] n [UNK] n ##미 ##국

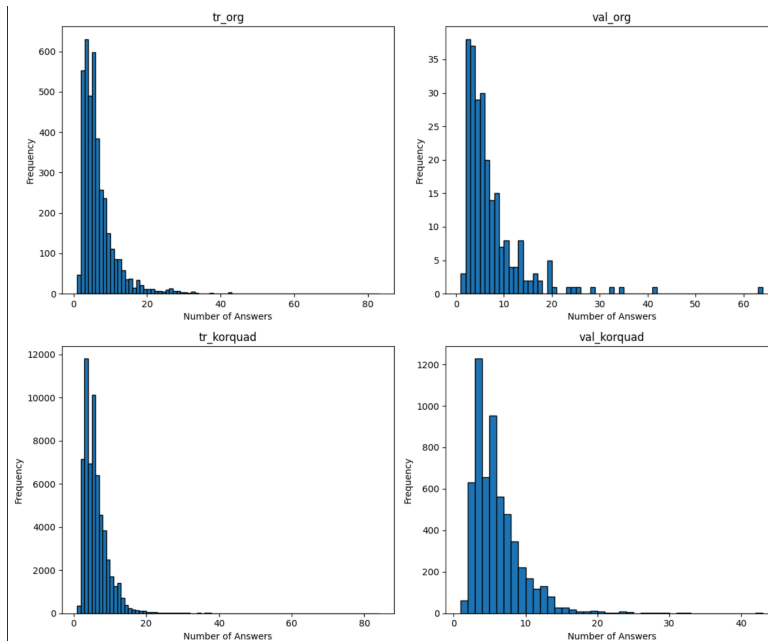
위와 같이 토큰화되는데, 쓸데없이 n이 생길뿐더러, '미'가 되었어야 할 토큰이 '##미'가 되어버린다.

iv. \n을 모두 공백으로 바꾼 후, 다시 UNK 개수 분포 측정



문서 1개당 UNK 토큰 개수가 대부분 5개 이하로 거의 줄었음을 알 수 있다.

v. Answer 길이 분포 (추후에 사용할 KorQuAD 데이터셋도 같이 분석하였음, 토큰이 아니라 길이로 분석)



특이값을 제외하면 대부분 0~30의 길이를 가졌다.

vi. 특이한 정답 확인 (UNK가 포함된 정답 등..)

```

: val_org_filtered['answers']

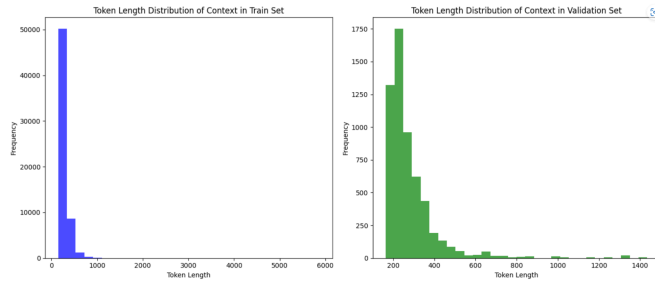
: [{'answer_start': [70], 'text': ['진전(陳田)이라 새겨진 기와조각'],
  {'answer_start': [212], 'text': ['가오슝 시'],
  {'answer_start': [135], 'text': ['훙카이도'],
  {'answer_start': [104], 'text': ['흑색육(黑色肉)],
  {'answer_start': [0], 'text': ['제서지전(齊西之戰)이후'],
  {'answer_start': [33], 'text': ['여정현(呂正鉉)],
  {'answer_start': [335], 'text': ['트레베제 부대'],
  {'answer_start': [524], 'text': ['트르피미로비치 왕조(Trpimirović) 출신의 문치미르'],
  {'answer_start': [592], 'text': ['양당(楊黨)],
  {'answer_start': [491], 'text': ['물적 성과(物的成果)']}]
```

괄호 안에 한자나 외국어가 들어있거나 특수기호가 정답 안에 포함되었는 것을 확인하였다. 이를 통해 함부로 전처리, 후처리를 하지 말고 고려를 하여 진행을 해야겠다고 판단했다.

vii. KorQuAD 1.0 데이터셋 context 분석 (추후에 같이 사용할 데이터셋)

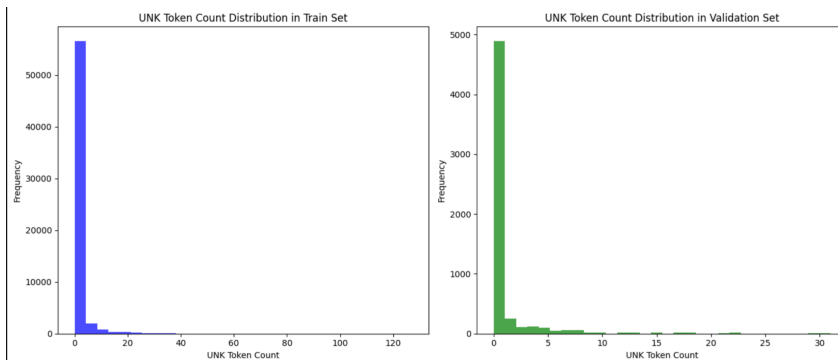
우리가 사용할 데이터셋이 3952개로 부족할 수 있다고 판단했고 현재 데이터셋과 유사한 데이터셋을 찾아서 분석해보았다. (KorQuAD 1.0은 train 60407개로 비교적 큰 데이터셋)

context 토큰 개수 분포(roberta-large 토큰나이저 기준)



train 데이터셋에서 평균적으로 문서 1개당 275개의 토큰을 가지며, 75% 기준선이 304개의 토큰에서 형성되었다. 단, 토큰 5874개의 context도 존재하여서 그래프가 저렇게 보인다.

unk 개수 분석

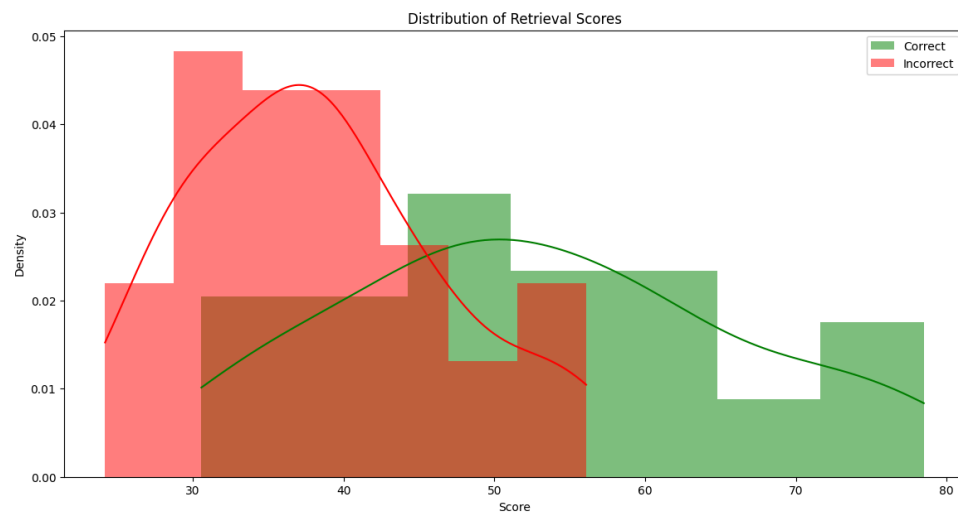


평균적으로 문서 1개당 1개의 UNK 토큰을 가지며, 기존 데이터셋과 비교하면 무척 적은 것을 확인할 수 있다. 대다수의 문서가 UNK 토큰이 없었다.

unk 토큰의 원본단어도 확인해보았는데 대다수가 한자였다.

- Retrieval Data(wikipedia_documents.json)

정답, 오답 Retrieved passage에 따른 question의 길이 분포



Question의 길이가 짧을 경우 Passage Retrieve를 정확하지 못할 가능성이 높았다. 이는 짧은 쿼리일수록 충분한 정보를 담지 못하기 때문일 수 있다. 따라서 Query Rewriting과 같이 Query에 정보를 추가하는 기법들을 적용해 볼 수 있을 것이다. 실제로 question의 핵심단어를 단순히 이어붙였을 때, retrieve가 되는 경우도 찾아볼 수 있었다.

C. Reader Model 훈련

Pretrain된 klue/roberta-large에 바로 기존 데이터셋을 finetuning하지 않고 도메인 적응을 하고 훈련시켰다. KorQuAD 1.0 데이터셋(60407개)으로 훈련후, 기존 데이터(3952개)로 finetuning을 하였다.

이렇게 한 이유는 기존 train 데이터셋이 3952개로 적은 편이어서, 작은 데이터셋에서 과적합 되기 쉬운 transformer 특성 상 성능이 안나올 수 있다고 생각했다. 따라서 바로 기존 데이터셋으로 roberta-large 모델을 fine-tuning하는 것이 아니라 task와 적합하면서 기존 데이터셋과 유사하지만 60407개로 상대적으로 큰 KorQuAD 1.0 데이터셋에서 1차로 훈련을 진행하고 기존 데이터셋에 2차로 fine-tuning함으로써 도메인 적응을 통해 성능을 향상시키고자 하였다.

i. 1차 fine-tuning

KorQuAD 1.0 데이터셋을 pre-trained 모델인 klue/roberta-large로 훈련하였다.

- 하이퍼파라미터

batch_size	32
max_epoch	1
learning_rate	3e-5
fp16	True
max_seq_length	384

- valid evaluation

eval_exact_match	88.5175
eval_f1	93.1273

ii. 2차 fine-tuning

1차 fine-tuning 모델에 이어서 기존 데이터셋으로 학습을 진행하였다.

총 4가지의 fine-tuning을 진행했다.

저장된 모델 명	batch_size	전처리유무	max_seq_length	epoch	fp16
finetune_fp16_nonewline	32	o	384	9e-6	True
finetune_fp16_withnewline	32	x	384	9e-6	True
finetune_fp16_nonewline_bs16	16	o	384	9e-6	True
finetune_fp16_withnewline_bs16	16	x	384	9e-6	True

- valid evaluation

저장된 모델 명	valid EM	valid F1
finetune_fp16_nonewline	69.5833	78.7052
finetune_fp16_withnewline	70.8333	79.2693
finetune_fp16_nonewline_bs16	68.75	77.0281
finetune_fp16_withnewline_bs16	69.1667	78.3896

- 추론 결과

추론 시 모두 bm25를 사용하였다. Retrieval Data(wikipedia_documents.json)는 전처리 하지 않았다.

저장된 모델 명	private EM	private F1
finetune_fp16_nonewline	68.75	80.50
finetune_fp16_withnewline	70.42	79.81
finetune_fp16_nonewline_bs16	67.50	78.04
finetune_fp16_withnewline_bs16	67.92	77.87

이외 **Reader**모델 훈련 진행 참고사항

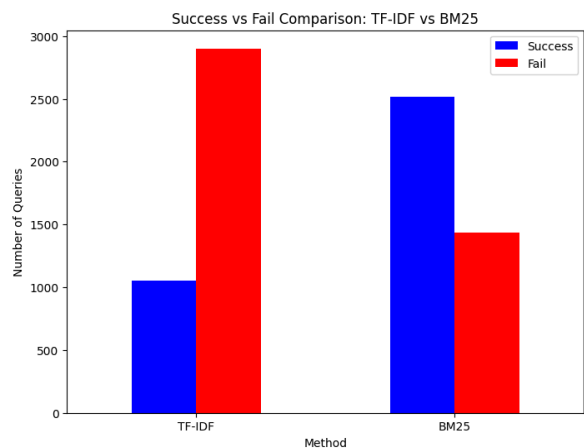
1. fp16 옵션을 사용하여 정확도를 떨어뜨리지 않으면서 훈련/추론 속도를 크게 올릴 수 있었다.
기존 훈련시간이 1시간이 소요되었는데 30분으로 줄일 수 있었다.
2. roberta-large의 max_sequence_length를 384로 고정하고 실험하였다. 512로 늘린다면, 성능이 늘어날 것으로 생각했지만 실험을 많이 하고 분석하는 것이 공부에 도움될 것 같아 전반적으로 384로 진행하였다. (마지막에 512로 훈련시키고 결과를 제출할까 생각했지만 하지 않았다.)
3. Reader모델을 훈련할 때, context를 다양하게 전처리한 모델을 훈련하고 Huggingface에서 공유했다. \n을 공백으로 바꿔서 훈련한 모델, 전처리하지 않은 모델, \n을 없애지 않고 '\n미국' -> '\n 미국' 같이 띄어쓰기만 뒤에 추가하고 훈련시킨 모델 등을 업로드해서 추론시에 가져와서 추론방식만 바꿔서 실험할 수 있도록 세팅했다.

D. Sparse Embedding Retriever

유사도 알고리즘 비교 실험

TF-IDF와 BM25의 Retrieval 결과를 명확하게 비교해보고자 진행하였다. 정답 문서 번호가 존재하는 train_dataset['train'] 을 활용하여 20개의 문서 내에 정답 문서를 찾을 경우 성공, 찾지 못할 경우 실패로 분류하여 두 유사도 알고리즘의 성능을 비교하였다.

실험 결과, TF-IDF는 약 27%의 성공률, BM25는 약 64%의 성공률이 나타났다. 따라서 유사도 알고리즘은 BM25를 활용하는 것으로 결정하였다.



Sparse Embedding 취약점 분석

위 실험에서 진행한 결과를 바탕으로 Sparse Embedding의 취약점을 분석하였고 취약점을 해결하여 리트리버 성능의 개선을 이루고자 하였다. .

첫번째로 단어 변형에 취약하였다. 현재 공백을 기준으로 토큰나이징 후 Sparse Imbedding을 진행하기 때문에 동사의 형태 변화나 조사의 차이가 존재할 경우 다른 단어로 인식하였다. 대표적인 예시로 질문에는 '강희제가' 의 형태로 존재하지만 정답 문서에는 '강희제는' 과 같이 다른 형태로만 존재할 경우 문서를 찾지 못하는 것을 발견하였다. 이러한 문제를 해결하기 위하여 조사를 제거하는 전처리를 수행하거나, 형태소 단위로 분리하는 토큰나이저, N-gram 등의 방식을 고려하였다.

두번째로 유의어를 인지하지 못한다. 예시로 질문에는 '남편을 잃은' 이라는 키워드가 있지만 정답 문서에는 '과부' 와 같은 단어로 표현하는 경우가 있었다. 이러한 문제를 해결하기 위하여 Dense Embedding과 조합한 Embedding 방식이 좋을 것이라 예상하였다.

리트리버 성능 비교 실험

위에서 고안한 아이디어를 실제로 테스트해보기 위하여 리트리버 성능 실험 코드를 새로 구성하였다. 데이터는 `train_dataset['train']` 를 활용하였고, BM25로 `topk` 내에 정답 문서를 찾는 비율을 기준으로 여러 방식의 성능을 비교하였다. 토큰나이징 방식과 `n-gram` 등을 파라미터로 활용하였다

실험 결과는 다음과 같다.

tokenize	Blank	Blank	Okt	Okt	Char	Char	Koelectra	Koelectra	Kiwi	Blank	Okt	Okt	Char	Kiwi
n-gram	1	2	1	2	2	3	1	2	1	1	1	1	2	1
remove										조사	조사+	조사	조사	불용어
topk=20	2516 (63.66%)	1103 (27.91%)	3567 (90.26%)	2829 (71.58%)	3636 (92.00%)	3443 (87.12%)	3585 (90.71%)	3243 (82.06%)	3443 (87.12%)	3449 (87.27%)	3570 (90.33%)	3542 (89.63%)	3637 (92.03%)	3502 (88.61%)
topk=10	2356 (59.62%)	1057 (26.75%)	3435 (86.92%)	2634 (66.65%)	3499 (88.54%)	3308 (83.70%)	3449 (87.27%)	3094 (78.29%)	3308 (83.70%)	3303 (83.58%)	3434 (86.89%)	3408 (86.23%)	3496 (88.46%)	3345 (84.64%)
topk=5	2166 (54.81%)	978 (24.75%)	3254 (82.34%)	2440 (61.74%)	3337 (84.44%)	3132 (79.25%)	3278 (82.95%)	2902 (73.43%)	3132 (79.25%)	3120 (78.95%)	3248 (82.19%)	3223 (81.55%)	3337 (84.44%)	3161 (79.98%)

글자 단위 토큰나이징 후 조사를 직접 제거하여 Bigram을 적용시킨 결과가 가장 높게 나타났다. 단일 토큰나이저 성능은 'monologg/koelectra-base-v3-finetuned-korquad' 의 성능이 가장 높았다.

E. Dense Embedding

Bi-encoder 방식으로 실습 코드를 프로젝트에 맞게 재가공해서 학습을 시키고 re-rank 결과를 확인했을 때 단일 Sparse Retriever 방식대비 10% 정도의 정확도 하락이 있었다. Encoder를 이용해 Question과 Contexts의 유사도를 측정할 때, Question과 Contexts의 정보의 비대칭성 때문일 가능성이 있다고 생각한다.

즉, Question에 비해 Contexts는 길이가 길어 다양한 정보를 담고 있어 의미적으로 유사하다고 보기 힘든 것이다. 따라서 DensePhrase와 같은 방법을 사용하여 비교적 여러 의미를 갖지않는 구문(Phrase)단위로 Question과의 유사도를 파악하는 방법이 더 어울릴 것이라 생각한다.

4. 프로젝트 수행 방법

github

main

[Go to file](#)
[Code](#)

Yunseo-Lab Merge pull request #35 from boostcampaitch7/ex... 5bbb3cf · 12 hours ago

.github	init: setup project setting	2 weeks ago
code	Merge pull request #35 from boost...	12 hours ago
profile	Add files via upload	last week
.gitignore	feat: modularize config and add B...	2 weeks ago
.pre-commit-config.yaml	init: setup project setting	2 weeks ago
README.md	modularize space preprocess traini...	last week

이전 프로젝트에서 실험 기록 공유가 부족하다는 피드백을 반영하여, 이번에는 실험 기록과 과정을 팀원들 간에 효과적으로 공유하는 데 중점을 두었다. 이를 위해 프로젝트의 컨벤션을 정하고, **commit**과 **issue** 등의 규칙을 체계화하여 팀원들이 작성하는 메시지를 통일시켜 쉽게 파악할 수 있도록 하였다.

Author ▾	Label ▾	Assignee ▾	Sort ▾
<div><div>🕒</div><div>[REFACTOR] 코드 좀 살펴보고 리팩토링 좀 하겠습니다</div><div>#34 opened last week by junhanjeong</div><div>🔖</div><div>💬 1</div></div>			
<div><div>🕒</div><div>[FEAT] wiki 파일 csv 저장에서 json 파일 저장 방식으로 변경하겠습니다.</div><div>#25 opened last week by GeumSangLEE</div><div></div><div>💬 3</div></div>			
<div><div>🕒</div><div>[FEAT] EDA 코드 추가!</div><div>#12 opened 2 weeks ago by junhanjeong</div><div></div><div></div></div>			

또한, **issue** 기능을 적극적으로 활용해 팀원들의 진행 상황과 예러 사항을 공유하고, 각 **task**의 수행 이유와 목표를 명확히 기록하여 다른 팀원들이 필요한 정보를 쉽게 확인할 수 있도록 했다.

huggingface

Reader 모델의 경우, **retrieval** 단계에서 사용하기 위해 매번 훈련 단계에서 **roberta-large**와 같은 대형 모델을 **fine-tuning**할 때마다 30분 정도의 시간이 소요되었다. 이러한 시간을 줄이고 팀원들 간에 모델을 쉽게 공유할 수 있도록, 학습된 모델을 **Hugging Face Models**에 업로드하여 팀원들이 모델을 효율적으로 관리하고 실험을 수행할 수 있었다.



















Models 4	🔍	↑↓ Sort: Recently updated
<div><div>NLP03/finetune_space_after_newline</div><div>private</div><div>Updated 8 days ago • 📄 3</div></div>		
<div><div>NLP03/finetune_fp16_withnewline</div><div>private</div><div>Updated 9 days ago • 📄 24</div></div>		
<div><div>NLP03/finetune_fp16_nonewline</div><div>private</div><div>Updated 9 days ago • 📄 4</div></div>		
<div><div>NLP03/korquad_fp16_384</div><div>private</div><div>Updated 9 days ago • 📄 2</div></div>		

Notion

노션(Notion)을 활용해 팀원들이 각자의 실험에 대한 미니 논문을 작성하고, 실험 세팅, 과정, 결과, 추후 개선 방향 등을 기록하도록 하였다. 이를 통해 모든 팀원이 서로의 실험 과정을 쉽게 확인할 수 있었으며, 동일한 실험을 여러 명이 진행할 때는 하나의 연구 log 페이지에서 공동 작업을 통해 결과를 공유할 수 있었다.

이 과정은 실험 기록의 일관성을 유지하고 협업 효율성을 높이는 데 기여했다. 문제점이나 개선 사항을 함께 기록함으로써, 추후 실험에 참고할 수 있는 자료를 제공하고 실험의 품질을 향상시켰다.

연구 log ...

-  Query 길이와 Retrieval 정확도의 관계
-  Sparse & Dense Hybrid: Rerank
-  전처리 (train_dataset)
-  후처리 (predictons.json)
-  curriculum learning (with CosineEmbeddingLoss)
-  Original context EDA
-  KorQuAD, original Answer EDA
-  Korquad context EDA
-  유사도 알고리즘에 따른 Retrieval 결과 분석
-  BM25 Retrieval 취약점 분석을 통한 성능 개선
-  TopK에 따른 Reader 성능 실험
-  wiki data EDA
-  문장단위 접근법
-  초기 Korquad 학습후 original 파인튜닝한 과정
-  max_seq_length 512로 하고 돌려본것 (하지만 fp16=True
-  fp16을 쓰면 빨라질까? + 전처리여부에 따른 성능비교
-  wiki 전처리에 따른 성능 실험
-  retriever 개선1_bm25Plus GPU를 사용

4. 프로젝트 수행 결과

Leaderboard [mid]

6	NLP_03조		70.4200%	79.8100%	38	1w
---	---------	---	----------	----------	----	----

Leaderboard [Final]

10	NLP_03조		64.4400%	76.9900%	38	1w
----	---------	---	----------	----------	----	----

5. 자체 평가의견

대회형 프로젝트에서는 앙상블이 성능에 높은 영향을 미친다는 것을 알지만 이번 프로젝트에서는

MRC와 Retrieval 과제에 집중하기 위해서 앙상블을 진행하지 않았다.

이는 지난 프로젝트에서의 프로젝트 중심적 운영에 대한 문제의식에 기인한 것이며, 이번 프로젝트에서는 깃허브 활성화와 허깅페이스를 이용한 모델 공유, 노션을 이용한 연구공유 등 다양한 협업 방식을 추가로 시도했다. 더불어, 지난 프로젝트에 비해 데이터의 이해와 Task가 작동하는 방식에 대해 더 깊은 이해를 중심으로 한 것에 의의가 있다.

추가적인 아이디어도 생각은 해보았지만 시간상 하진 못했었다. 생각한 아이디어는 아래와 같다.

1. Reader모델도 추론시와 비슷한 환경을 조성하기 위해, **hard negative sampling**을 하여 비슷한 문서를 가져오고 훈련을 진행한다.
2. 멘토링 때, DPR 구현에 도움이 될만한 아이디어를 얻었고 **hard negative sample**로 훈련시키거나 **simcse**를 사용하는 방법도 해볼 수 있었을 것 같다.

6. 개인 회고

육지훈_T7404

1. 나는 내 학습 목표 달성을 위해 무엇을 어떻게 했는가?

- 목표 설정: **MRC**(기계 독해)와 **ODQA**(개방형 도메인 질문 응답) 프로젝트에서 성능 향상을 목표로 설정했다. 특히 질의 응답 정확도 향상과 효율적인 검색 모델 구축을 목표로 두고 실험했다.
- 학습 전략:
 - **BERT** 기반의 모델을 학습하며 데이터 전처리와 하이퍼파라미터 튜닝에 집중했다.
 - 학습 진행 과정에서 **wandb**를 사용해 실시간으로 성능을 모니터링하며 문제를 빠르게 피드백했다.

2. 나는 어떤 방식으로 모델을 개선했는가?

- 하이퍼파라미터 최적화: **Batch size**, **learning rate** 등 여러 조합을 실험해 최적의 결과를 찾았다.
- 학습 데이터 증강: **Retrieval** 단계에서 파라미터(**top-k**)를 조정하여 질의 응답 정확도를 높였다.

3. 내가 한 행동의 결과로 어떤 지점을 달성하고, 어떤 깨달음을 얻었는가?

- 모델 정확도 향상: **TF-IDF**에서 **BM25**로 전환한 결과, **EM(Exact Match)**과 **F1** 스코어가 상승했다.
- 프레임워크 활용의 중요성: **wandb**를 활용해 모델 개선 과정을 체계적으로 관리한 것이 큰 도움이 되었다.

4. 전과 비교하여 새롭게 시도한 변화는 무엇이고, 어떤 효과가 있었는가?

- **BM25+** 개선: **BM25** 외에 **BM25+**를 도입해 더 정교한 검색을 시도했다.
- 추론 속도 개선: **Retrieval** 모델과 **Reader** 모델의 경량화를 시도하여 추론 시간을 줄였다. 이를 통해 실시간 응답 성능을 크게 향상시킬 수 있었다.

5. 마주한 한계는 무엇이며, 아쉬웠던 점은 무엇인가?

- 모델의 편향 문제: 일부 질문에서는 데이터 편향으로 인해 잘못된 답변이 도출되는 문제를 경험했다.
- **Reader** 성능: 질문이 길거나 복잡할 경우 리더기 모델이 적절한 답변을 추출하지 못하는 경우가 있었다.

6. 한계/교훈을 바탕으로 다음 프로젝트에서 시도해볼 것은 무엇인가?

- 파라미터 효율화 기법: **LoRA(Low-Rank Adaptation)**나 양자화(**Quantization**)와 같은 기법을 사용해 더 경량화된 모델을 구축하고자 한다.
- 대규모 데이터 활용: 다음 프로젝트에서는 더 다양한 출처에서 수집한 데이터로 모델을 학습하여 일반화 성능을 높일 예정이다.

정준한_T7437

1. 프로젝트에서 무엇을 시도했는가?

- `config`를 사용하여 모두가 프로젝트 진행할 수 있도록 모듈화 & `Readme`와 노션에 사용법 작성
- 협업 시 `github` 사용에 관해 컨벤션 작성 (`issue`, `PR` 템플릿 추가, 커밋, 브랜치 컨벤션 작성 등)
- 도메인 적응 아이디어를 떠올리고, `prepare_dataset.py`로 데이터셋 갈아낄 수 있도록 모듈화
- `bm25OKapi`를 사용하여 추론 진행할 수 있도록 코드 추가
- 팀원이 진행한 코드를 바탕으로 모듈화하여 결합
- `fp16` 옵션을 사용하여 훈련/추론 시간 단축
- `sparse retriever` 사용시 공백구분이 아닌 토큰나이저로 토큰화할 수 있도록 기능 추가
- 전처리 코드 추가 (`\n` 제거하거나 공백으로 바꾸거나, 안하거나)
- `Reader` 모델을 업그레이드하기 위해, `EDA` 진행 (`context` 길이분포, `UNK` 분석 등)
- 팀원의 `huggingface` 사용 아이디어를 듣고, 모델 업로드하는 방법 공유

2. 프로젝트에서 배운 것

- `github` 사용시 컨벤션을 정하고 `issue/PR`을 통해 협업하는 방식을 처음으로 진행해본 것 같다.
- `ODQA`의 전반적인 프로세스와 전처리코드, 후처리코드에 대해 자세히 알 수 있었다.
- `Huggingface`로 모델을 공유하고, 노션에 실험기록을 공유하거나 내가 만든 모듈화 코드 사용법을 적는 등 협업을 좀 더 체계적으로 하는 법을 배운 것 같다.

3. 아쉬운 점

- (개인적인 아쉬운점) 나 스스로 `Retriever` 쪽에서 `DPR`에 대해서 더 공부하고 구현했으면 더 많이 배울 수 있었을 것 같다.
- (협업적 아쉬운점) `PR`을 올리고, 팀원들의 확인을 완벽하게 받지 않고 `merge`할 때가 많았다. 또, 피어세션때 활발히 의견공유할 때도 있었지만 활성화되지 않았을 때도 많았던 것 같다. 그래서 차라리, 피어세션 때 `PR`에 대해서 설명하고 진행하는 것이 더 좋지 않을까 싶다.
- 성능을 올릴 아이디어가 없지 않았는데 적당히 만족하고 프로젝트를 진행한 감도 없지 않은 것 같다. (`hard negative sampling`한다거나, `DPR`을 정교하게 구현해 `Rerank`한다거나, 실험을 더 엄격하게 변인통제하여 실험한다던지)

허윤서_T7442

1. 프로젝트에서 무엇을 시도했는가?

- Retriever 결과 EDA
- Sparse Retriever 실험
- Cross-Encoding, Bi-Encoding DPR
- Sentence Transformer DPR
- TF-IDF to BertEncoder (Bi-Encoder DPR) Rerank

2. 프로젝트에서 배운 것

- github를 협업에 맞게 issue, PR을 사용해봤던 것.
- MRC task가 어떻게 이루어지는지 알게 되었다.
- 사실기반의 답변을 이끌어낼때 Retrieval의 역할과 동작을 알게 되었다.
- Huggingface의 Transformers에 대해 더 깊은 이해가 있었다.

3. 아쉬운 점

- 프로젝트와 공부 사이의 최적의 밸런스를 찾기 위해 노력해야한다.
- 더 깊은 이해와 새로운, 최신의 방법의 탐색 사이의 최적의 밸런스를 찾기 위해 노력해야한다.

전진_T7431

1. 프로젝트에서 무엇을 시도했는가?

- 리트리버 성능 향상을 위한 실험 설계
- EDA를 통한 리트리버의 취약점 분석 및 개선 방법 고안
- 효율적인 실험 수행을 위한 코드 작성 및 유지보수

2. 프로젝트에서 배운 것

- github으로 체계화된 협업을 하는 방법
- Huggingface를 활용한 모델 공유
- ODQA task에서 리트리버의 역할 및 작동 원리, 개선점
- ODQA task에서 리더의 역할 및 작동 원리

3. 아쉬운 점

- 리트리버 실험에서는 높은 성능을 보였으나 실제 제출 결과는 잘 반영되지 않음
- 리트리버 실험에서 잘 나온 방법에 대한 명확한 이유를 파악하지 못함
- 최적의 모델 성능을 내는 topk를 찾는 실험을 구상하였으나, 구현에 실패함
- 리트리버에 집중을 하다보니 리더 성능 개선에는 기여하지 못함

이수진_T7411

1. 프로젝트에서 무엇을 시도했는가?

- 모듈화된 코드에 `rank_bm25` 라이브러리로부터 `BM25Plus` 클래스를 통해 `retriever`에 `BM25Plus`를 사용할 수 있도록 코드를 추가하였다.
- `wikipedia_documents.json`의 전처리 코드에서 '\n, \n'의 유무, '특수기호의 유무' 등 전처리의 조건을 달리하여 `pretrained` 시켜놓은 `reader` 모델에 적용해 성능을 측정하였다.
- `retriever` 성능 실험에서 서로 다른 토큰나이저를(공백으로 `uni-gram` 분리한 후에 조사제거를 위해 `KoNLPy` 라이브러리의 `Okt` 형태소 분석기를 사용하였다.) 적용하여 `retriever`의 성능을 측정하고 비교하였다.
- `reader`의 개선을 위해 정답이 포함된 문장을 찾고, 문장 내에서 정답을 찾도록 하는 알고리즘을 만들려고 시도하였다.

2. 내가 한 행동의 결과로 어떤 지점을 달성하고, 어떤 깨달음을 얻었는가?

- `BM25`보다 `BM25plus`의 성능이 조금 더 좋아서 `retriever`의 성능을 측정할 때는 `BM25plus`를 default로 이용하게 되었다.
- `retriever` 성능 실험 방법론을 공부 하면서 `retriever`에서 성능을 측정하기 위해 `Hit Rate` 지표와 `Mean Reciprocal Rank(MRR)`를 사용한다는 것을 알게 되었고, 팀에서 사용한 `retriever` 성능 측정 방법론은 `Hit Rate`에 해당한다는 것을 알게 되었다.

3. 전과 비교하여 새롭게 시도한 변화는 무엇이고, 어떤 효과가 있었는가?

- `github`를 전 프로젝트에 비해서 원활하게 사용할 수 있게 되었다. `github` 컨벤션 및 규칙을 적용하고 `issue`, `PR`를 통해 협업하는 것을 처음 적용하게 되었다. 서로의 코드를 공유하기 수월했고, 적극적으로 자신의 코드를 브랜치에 올려둘 수 있었다.
- `Huggingface`를 활용해서 모델 공유하는 법에 대해 알게 되었고, 앞으로 이용할 수 있을 것 같다.

4. 마주한 한계는 무엇이며, 아쉬웠던 점은 무엇인가?

- `github`를 사용하면서 적극적으로 `issue`를 올린 다거나 올라온 `PR`에 대해서 적극적으로 피드백을 줄 필요가 있었다. 사용이 조금은 익숙해졌으니 그 부분을 다음 프로젝트에서 보완하고 싶다.
- `retriever`의 성능측정을 하면서 내가 시도했던 공백으로 `uni-gram` 분리한 후에 조사제거한 토큰나이징 방식은 `hit rate` 지표가 백분율로 87.27정도 나왔다. 추가적인 개선을 위해 시도할 수 있었는데, 위에서 시도했던 다른 방법들이 `bi-gram`에서 성능이 더 좋지 않은 것을 보았고, 성능이 가장 우수한 `tokenizer` 방식이 있었기 때문에 추가적인 보완을 하지 않았다. 좀 더 다양한 방법들을 주도적으로 시도해보았으면 좋았을 것 같다.
- `reader`의 개선을 위해 정답이 포함된 문장을 찾고, 문장 내에서 정답을 찾도록 하는 문장 단위 접근법을 구현하다가 실패하였다.

5. 한계/교훈을 바탕으로 다음 프로젝트에서 시도해볼 것은 무엇인가?

- `retriever`에서 `BM25`의 방법론에 집중한 것 같은데 `DPR`을 더 공부해서 프로젝트에 적용시켜보거나, `rerank`를 정교하게 구현해보는 시도를 못해봐서 아쉽다. 추가적인 공부가 필요할 것 같다.
- 프로젝트에서 배운 것은 `ODQA`의 전체적인 흐름과 `retrieval`의 다양한 방법론을 알게 되었다. `RAG`는 많이 쓰이는 기술이기 때문에 추가적으로 더 공부해 봐야겠다.
- 다른 팀의 프로젝트 `wrap-up` 발표를 통해 전처리 한 결과를 쉽게 비교할 수 있도록 `streamlit`을 이용하여 대시보드를 만든 팀을 보았다. 그렇게 결과를 비교한다는 아이디어가 인상깊었고, 그렇게 시도한 것이 배울 점이라 생각했다.
- 이번 프로젝트에서는 개인적으로 내가 배우는 과정, 실험에 하면서 배운 점을 기록을 하려고 노력했다. 내가 뭘 알았고, 뭘 깨달았는지 미리미리 기록해둔 것이 프로젝트 진행 기간 동안의 나에 대해서 회고하기 좋은 것 같아서 작은 기록이라도 남겨 놔야겠다.

이금상_T7407

1. 프로젝트에서 무엇을 시도했는가?

- wiki data EDA

- \n 및 공백 등의 데이터 전처리

- 리트리버 성능 실험

- Pre-trained 모델 실험

2. 프로젝트에서 배운 것

- hugging face에 fine-tuning 모델을 배포하고 해당 모델을 사용하는 방법을 배움
- github 사용을 통해 팀원들과 협업
- 데이터 전처리를 통한 실험 결과
- 프로젝트의 전반적인 실험 및 결과를 미니 논문 형식으로 팀원들과 공유하는 방법

3. 아쉬운 점

- 개인적으로 리트리버를 더 파보지 못한 점에서 아쉽다는 생각이 든다.
- 개인적으로 llm을 이용한 증강 방식을 구현해보고 싶었는데 이 부분을 수행하지 못하여 아쉬움이 든다.
- 프로젝트를 이해하는데 시간이 걸렸고 이 부분 때문에 프로젝트를 수행하는데 여러가지를 실험하고 아이디어를 내는 부분에서 많이 막혔다고 생각이 들어 아쉬움이 든다.