
MRC 모델을 통한 자연어 처리 프로젝트

NLP팀
왕준호

목차

01 프로젝트 개요

02 사용한 모델

03 프로젝트 수행 절차 및 방법

04 프로젝트 수행 결과

05 자체 평가 의견

01 프로젝트 개요

모델 선정 배경 및 동기

구글링을 통한 Search Model 탐색

DrQA, DPR, ODQA 논문을 읽고 관련 모델을 찾아봄

한국어 자료들을 읽을 수 있는 모델을 찾아 선정

성능과 모델이 나온 시기에 관계 없이 여러 모델을 선정

01 프로젝트 개요

프로젝트 구현 내용 및 컨셉

여러 모델을 구현하여 비교

한국어 처리에 특화된 모델 선

전처리를 통한 토크나이징

불용어 삭제

```
stop_words = set([
    "가", "가까스로", "가령", "악", "악악", "악자", "각종", "갖고말하자면",
    "간다", "걸이", "거의치않고", "거니와", "거비", "거외", "것", "것과같이", "것들",
    "이디기", "개우다", "겨우", "견지에서", "결과에이르다", "결국", "결론을낼수있다", "경",
    "고려하면서", "고로", "끌", "공통으로", "마", "마련", "만개기있다", "만개없이", "관련이
    "관하여", "관히", "관해서", "구", "구체적으로", "구도하다", "그", "그들", "그때", "그거",
    "그라도", "그리스", "그리나", "그리니까", "그리연", "그리므로", "그리한즉", "그",
    "그런데", "그런즉", "그걸", "그걸에도 불구하고", "그렇게홀으로써", "그렇지", "그렇지인",
    "그렇지않으면", "그렇지만", "그렇지않으면", "그리고", "그리하여", "그만이다", "그에따라",
    "그위에", "그것", "그중에서", "그치지않다", "근거로", "근거하여", "기대여", "기성으로",
    "기타", "까닭으로", "까악", "까지", "까지미치다", "까지도", "과장", "꼼꼼", "끼익", "나",
    "님들", "남짓", "네", "네학", "네학들", "네", "넷", "년", "논하지않다", "놀라다", "누가",
    "누구", "더흔", "더흔방법으로", "다면", "다섯", "다소", "다수", "다시말하자면", "다시말",
    "다음에", "다음으로", "단지", "답다", "당신", "당장", "대로하다", "대하던", "대하여",
    "다하서", "당그", "더구나", "더군다나", "더라도", "더들어", "더죽다", "더죽이는", "도달",
    "등지어", "동안", "된바이아", "된이상", "된바찌로", "돌", "동등", "뭐파라", "뭐이어", "은",
    "등", "등등", "당중", "따라", "따라서", "따워", "따직지않다", "작", "때", "내가되어",
    "또한", "푹푹", "리하도", "떻", "로", "로인하여", "로루타", "로씨", "룩", "톰", "미증",
    "마치", "막론하고", "만못하다", "만약", "만약에", "만은아니다", "만이아니다", "만일",
    "말할것도없죠", "매", "미번", "데쓰겁다", "몇", "도", "모두", "무렵", "무릅쓰고", "무수",
    "들흔", "및", "바꾸어말하자면", "바꾸어서말하면", "바꾸어서한다던", "바",
    "밖에안된다", "반대로", "반대로말하자면", "반드시", "바금", "보는데서", "보다더", "보나",
    "부류의시행들", "부터", "물구하고", "물문하고", "뭉뚱", "내려거리다", "비교적", "비길수",
    "비슷하다", "비주어보아", "비하던", "뿐만아니라", "준반아니라", "준이다", "비걱", "비해",
    "생각한다도", "설명", "설사", "셋", "소상", "소인", "최", "첫", "습니까", "습니",
    "시초에", "시킨다", "실로", "심지어", "아", "아내", "아내나다를까", "아내라면", "아내만",
    "아투도", "아이", "이출점", "아이", "아이하고", "아이구", "아이야", "아이코", "아하", "아",
    "왕기위해서", "알수있다", "알았어", "엇", "앞에서", "알의것", "아", "악간", "양자", "아",
    "어느년도", "어느것", "어느곳", "어느때", "어느쪽", "어느해", "어다", "어찌", "어여한",
    "어떻게", "어쫓아", "어째", "어께서", "어쳤든", "어께라고", "어께서", "어께면허드", "어",
    "어께맞든", "어께첫", "어께하든지", "어께하여", "언제", "언젠가", "얼마안되는",
    "얼마만큼", "얼마들", "영영", "애", "이거식", "이모경있다", "이데하", "이 있다", "이",
    "여덟", "여러분", "여보시오", "여우", "여섯", "여전히", "여자", "연관되다", "연이서", "이",
    "예를들자면", "예컨대", "여하던", "온", "온도자", "도르다", "온자마자", "온직", "온호",
    "온도드", "와아", "와", "왜나하면", "외에도", "여기마다", "여기마다", "여기만", "이",
    "인듯하다", "인전", "일", "일것이다", "일금", "일간", "일반적으로", "일지라도",
    "자", "자기", "자기침", "자마자", "자신", "장간", "정지", "자", "제것", "제것만들", "제",
    "전자", "전후", "정에서보아", "정도에이르다", "제", "제각기", "제외하고",
    "조금", "조차", "조차도", "조조", "즘", "祚아", "한한", "조록준족", "주제하지않고",
    "준은둔했다", "준은모든다", "중에서", "중의하나", "즈음하여", "恚", "죽시", "지른시",
    "지만", "지말고", "진짜로", "죽으로", "자라리", "참", "침니", "첫번째로", "첫", "총작으로",
    "총작으로말하자면", "총작으로보면", "실", "팔팔", "쿵", "탁다", "타인", "탕탕",
    "드하다", "동하여", "듬", "꿰", "듬다", "왁", "꼴", "왁", "厲경", "하", "하게될것이다",
    "하게하다", "하였느라", "하고있다", "하고있었다", "하고하였다", "하구나", "하기때문에",
    "하기위하여", "하기는한대", "하기만하면", "하기보다는", "하기에", "하나", "하느니",
    "하는경애", "하는편이낫다", "하는것도", "하는것만못하다", "하는것이낫다", "하는애",
    "하더라도", "하로다", "하드롭시키다", "하도록하다", "하든지", "하려고하다", "하마터면",
    "하면할수록", "하면된다", "하던서", "하물며", "하여금", "하여아", "하자마자", "하지않",
    "하지않도록", "하지마", "하지마라", "하지만", "하하", "한끼동에", "한이온", "한후",
    "한마디마음", "한마디마음의도", "한마디", "한마디", "한결이있다", "한건으로는", "한창뜻", "한"
])
```

```
# 모델별로 결과값을 추출하게함
for n, (model_name, search_function) in enumerate([
    ("TF-IDF", search_model.tfidf_search), # 기본 모델
    ("BM25", search_model.bm25_search), # 기본 모델
    ("Inverted Index", search_model.inverted_index_search), # 기본 모델
    ("RNN", search_model.rnn_search), # 기본모델, 투습 o
    ("DPR", search_model.dpr_search), # finetuned model, 투습 x
    ("Transformers", search_model.transformers_search), # finetuned model, 투습 x
    ("Sentence Transformers", search_model.sentence_transformers_search), # finetuned model, 투습 x
    ("KSCRoberta Search", search_model.KSCRoberta_search), # pretrained model, finetuned
]):
```

CONCEPT OF PROJECT

01 프로젝트 개요

모델 설명

- TF-IDF (Term Frequency-Inverse Document Frequency)
문서 내에서 단어의 빈도와 그 단어가 다른 문서들에 걸쳐 얼마나 희귀한지를 고려하여 가중치를 계산함
고전적인 방법론으로서 텍스트 검색에서 널리 사용되며 간단하고 효과적인 방법이지만 문맥이나 단어의 의미를 파악하지는 못함
- BM25 (Best Matching 25)
TF-IDF의 확장으로 검색 쿼리와 문서 간의 관련성을 평가하는 데 사용
정보 검색에서 매우 효과적이며 TF-IDF보다 더 진보된 기술로 여겨집니다. 문맥을 고려하지는 않지만 문서의 길이를 고려하여 보다 정교한 점수 지표를 제공함
- Inverted Index
단어와 그 단어가 포함된 문서의 목록을 맵핑하는 데이터 구조
대규모 문서 집합에서 빠른 검색을 가능하게 하며 검색 엔진의 핵심 구성 요소 단 자체적으로는 문맥이나 의미를 분석하지 않음
- RNN (Recurrent Neural Network)
시퀀스 데이터 처리에 적합한 신경망으로 이전의 출력이 다음의 입력으로 활용됨
자연어 처리에서 중요한 역할을 하며 문맥을 고려할 수 있지만 긴 시퀀스에서는 효율성이 떨어질 수 있다는 단점이 존재
본 프로젝트에서 유일하게 학습 레이어를 쌓고 학습을 시킨 모델
- Haystack
대규모 문서 집합에서의 검색과 QA 시스템 구축에 활용되는 모델
Retrieval과 Reader 역할을 동시에 수행 가능하며 Pipeline까지 구축이 가능한 유용한 프레임워크임
- DPR (Dense Passage Retrieval)
질문과 문서를 밀집 벡터로 변환하여 비슷한 순으로 내림차순 정렬 top k 만큼의 값을 뽑아냄
질문에 가장 잘 맞는 문서를 찾는 데 효과적이며 특히 자연어 질의에 대한 정확한 답변 찾기에 적합함
긴 문서를 읽고 이해하는 데 적합함
- Transformers
Attention 메커니즘을 사용하여 입력 데이터의 모든 부분 간의 관계를 고려함
자연어 이해 및 생성 작업에서 뛰어난 성능을 보이며 복잡한 문맥과 의미를 파악하는 데 매우 효과적
본 프로젝트에서는 document가 한국어 이므로 kcbert의 개정판인 kcelectra 모델을 base 모델로 사용함
- Sentence Transformers
문장 또는 문단 전체의 의미를 포착하여 벡터로 변환함
문장 간의 의미적 유사성을 파악하는 데 사용되며 문맥을 포함한 텍스트의 의미를 잘 이해할 수 있음
- KoSimCSE-roberta, paraphrase mpnet, MSMARCO DistillBERT
한국어에 특화된 최신 문장 임베딩 모델들로 Sentence Transformer 기반의 모델을 사용
한국어 텍스트의 의미적 유사성을 판단하는 데 매우 효과적이며 특히 한국어 자연어 이해에 적합함
본 프로젝트에서 SOTA 모델의 예시로 가져옴

01 프로젝트 개요

모델 간의 성능 비교

Golden document 추출

전처리 후 검색 단어가 ""로 감싸져 있다는 사실을 이용해 정답 문서 추출

자카드 유사도 사용

추출한 정답 문서와 모델이 추출한 문서와의 자카드 유사도를 계산함

```
# 유사도에는 코사인 유사도 함수를 사용
# 임베딩 간의 유사도를 측정하기에 가장 적합함
def jaccard_similarity(doc1, doc2):
    tokens1 = okt.nouns(doc1)
    tokens2 = okt.nouns(doc2)

    set1 = set(tokens1)
    set2 = set(tokens2)
    intersection = set1.intersection(set2)
    union = set1.union(set2)

    if not union:
        return 0

    jaccard_similarity = len(intersection) / len(union)

    return jaccard_similarity
```

```
# 파일은 주제인 txt 파일의 내용을 이용해 context (topic)으로 주제를 뜻하거나 문장을 이용하여 해당 document를 추출
# 100% 정확성으로는 광고 context가 document 내에 있는지 여부를 판별하는 것은 불가능하며 대신으로 아래에서는 try except 문을 사용함
def golden_document(documents, query):
    query_with_quotes = f'\"{query}\"'

    for document in documents:
        # 광고 element 를 걸러 skip
        if not document.strip():
            continue
        # '\"'로 둘러 context (여기서는 query 변수로 받음)가 담긴 경우 해당 document를 추출하고 loop 를 끝내 다음
        occurrence = document.count(query_with_quotes)

        if occurrence > 0:
            golden_document = document
            break

    return golden_document
```

01 프로젝트 개요

개발 환경

ssh 서버:

- Linux (ssh 서버)
- OS: Ubuntu 20.04
- Python Version: 3.9.5
- GPU: Nvidia Tesla K80
- ML Framework: torch 2.0.0
- CUDA version: 11.4

로컬 환경:

- OS: Windows 11
- RAM: 64GB
- GPU: Nvidia T1000
- CPU: i7-12700

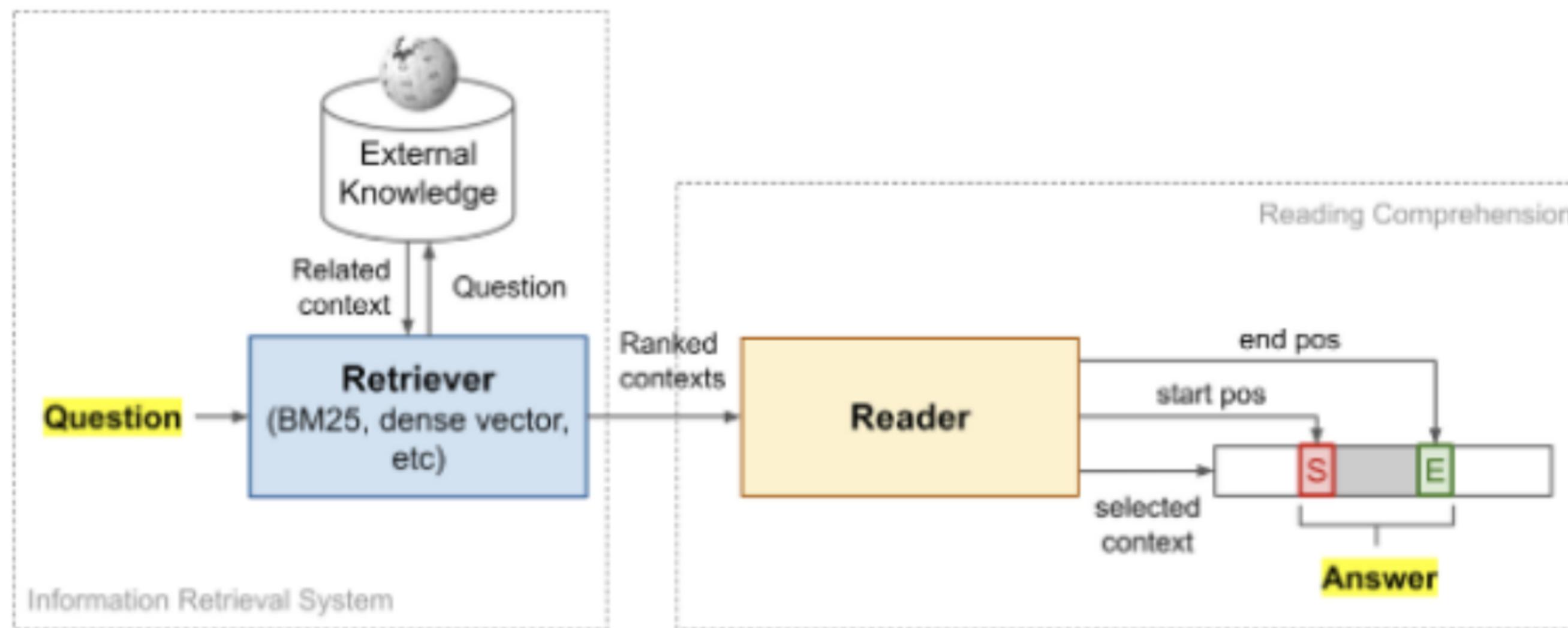
```
bogomips : 4400.10
clflush size : 64
cache_alignment : 64
address sizes : 46 bits physical, 48 bits virtual
power management:

processor : 39
vendor_id : GenuineIntel
cpu family : 6
model : 79
model name : Intel(R) Xeon(R) CPU E5-2630 v4 @ 2.20GHz
stepping : 1
microcode : 0xb000021
cpu Mhz : 1206.347
cache size : 25600 kB
physical id : 1
siblings : 20
core id : 12
cpu cores : 10
apicid : 57
initial apicid : 57
fpu : yes
fpu_exception : yes
cpuid level : 20
wp : yes
flags : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe sys
call nx pdpe1gb rdtsvp lm constant_tsc arch_perfmon pebs bts rep_good nopl xtstopology nonstop_tsc aperfnpmpf eagerfpn pn1 pcimulqdq dtes64 mon
itor ds_cpl vmx smx est tm2 ssse3 sdbe fma cx16 xtpri pdcm pdid dca sse4_1 sse4_2 x2apic movebe popcnt tsc_deadline_timer aes xsave avx f16c rd
rand lahf_lm abm 3dnowprefetch epb cat_3 cdp_l3 intel_pt tpr_shadow vmm1 flexpriority ept vpid fsgsbase tsc_adjust hmlt hle avx2 ssepm bm12 e
rms invpcid rtm cmn rdseed adx snap xsavesopt cmn_llc cmn_occur_llc cmn_nbm_total cmn_nbm_local dtherm ida arat pln pts
bogomips : 4405.54
clflush size : 64
cache_alignment : 64
address sizes : 46 bits physical, 48 bits virtual
power management:
```

NVIDIA-SMI 470.223.02 Driver Version: 470.223.02 CUDA Version: 11.4						
GPU	Name	Persistence-M	Bus-Id	Disp.A	Volatile Uncorr.	ECC
Fan	Temp	Perf	Pwr:Usage/Cap		Memory-Usage	GPU-Util Compute M.
						MIG M.
0	Tesla K80	Off	00000000:06:00.0	Off	0	
N/A	30C	P8	27W / 149W	3MiB / 11441MiB	0%	Default N/A
1	Tesla K80	Off	00000000:07:00.0	Off	0	
N/A	27C	P8	29W / 149W	3MiB / 11441MiB	0%	Default N/A
Processes:						
GPU	GI	CI	PID	Type	Process name	GPU Memory Usage
ID	ID					
No running processes found						

01 프로젝트 개요

프로젝트 구조



01 프로젝트 개요

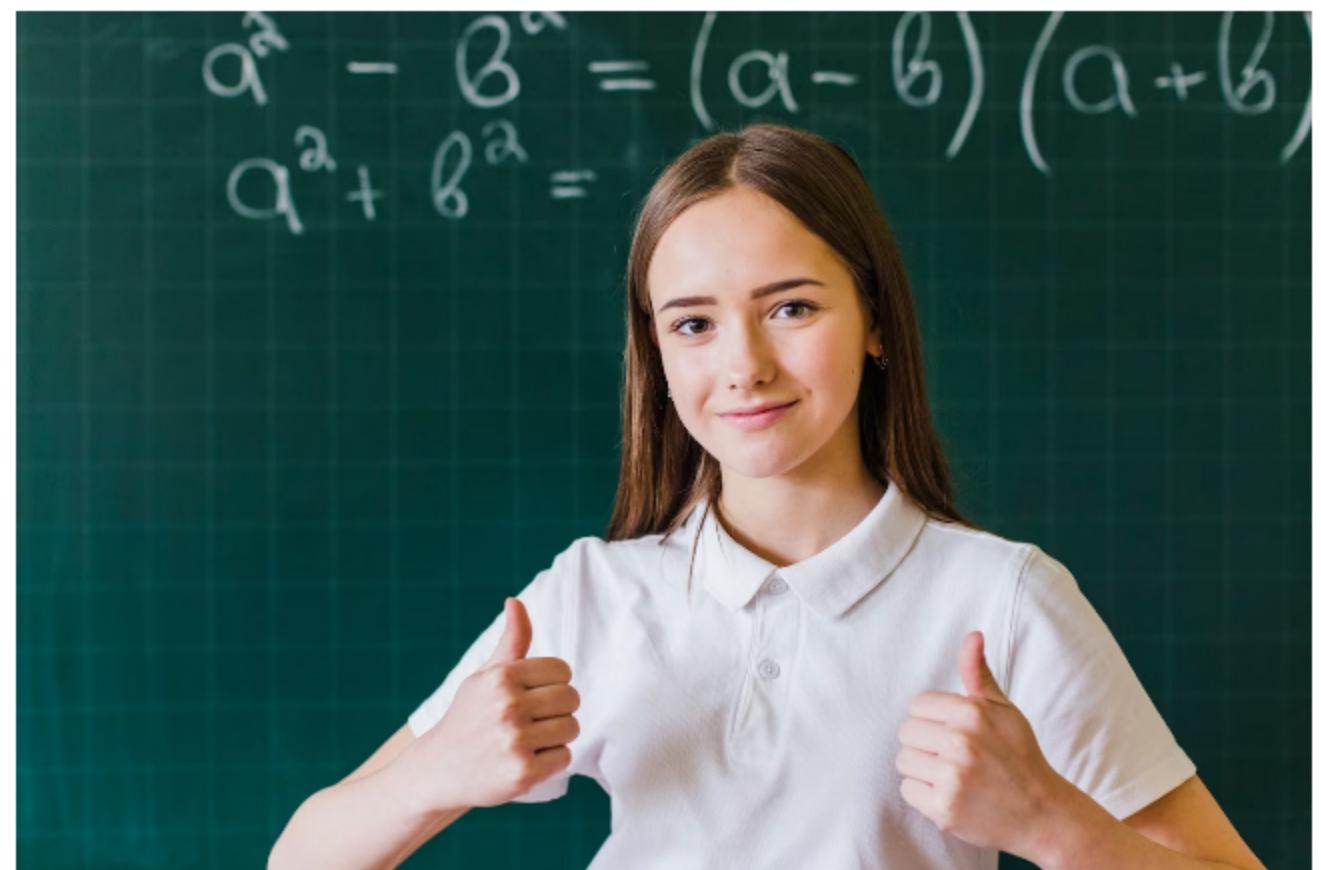
기대 효과

모델 간의 문서 추출 능력 및 reader로서의 능력 비교

최신모델과 구 모델과의 성능 비교

query와 context에 맞는 문구를 추출할 수 있는지 확인

top k를 통해 확률이 높은 순서대로 결과물을 확인 가능



03 프로젝트 수행 절차 및 방법

수행 절차 및 방법

1. document 텍스트 파일 탐색 및 전처리 계획 수립 (1일차)
2. txt 파일에서 정답 문서 추출 방법 확인 (2일차)
3. 관련 모델 논문 및 요약본 공부 (3~4일차)
4. 다른 사람이 짜놓은 모델 코드를 보고 공부 (DrQA) (5~6일차)
5. 모델 빌드 (7~9일차)
6. main 함수 빌드하고 주석 달기 (10일차)
7. 발표준비 (11일차)

총 개발기간: 12.12 ~ 12.26 (약 2주)

04 프로젝트 수행 결과

데이터 수집

1. [데이터 수집] Open Domain 데이터 수집

- 기본 제공 파일: kowiki-20220220-001.txt

노션에 나와있는 파일 사용

한국어 데이터 처리가 쉽지 않아 영어 문서를 이용하려고 했으나
한국어 자연어 처리를 한 번 도전해보기로 함

04 프로젝트 수행 결과

데이터 예시

```

Mathematics", in "The World of Mathematics", James K. Newman, editor, Dover, 2003, {{ISBN|0-486-43268-8}}.* Peterson, Ivars,
"Mathematical Tourist, New and Updated Snapshots of Modern Mathematics", Owl Books, 2001, {{ISBN|0-8050-7159-8}}.</div>{{참고 자
료 끝}}== 외부 링크 ==* {{위키공용분류-줄}}* {{프린-안라인|수학}}* {{언어링크|ko}} (http://www.kms.or.kr 대한수학회(KMS)* {{언어링크|ko}})
[https://web.archive.org/web/20150213234737/http://www.msquare.or.kr/ KAIST 수학문제연구회]* {{언어링크|en}}
[http://mathworld.wolfram.com(英) Mathworld]{개진 링크|url=http://mathworld.wolfram.xn--com().wt7n/ }* {{언어링크|en}}
[http://www-history.mcs.st-and.ac.uk/ The MacTutor History of Mathematics archive]* {{언어링크|ko}}
[https://web.archive.org/web/20130328230146/http://mathnet.or.kr/new_sub05/sub05_04.php 수리과학연구정보센터 용어 검색]((수학 분야))
[[전거 품제]](분류:수학) [[분류:형식과학]](분류:수학 용어)

***[[수학]]***에서 ***상수***란 그 값이 변하지 않는 불변량으로, [[빈수 (수학)|빈수]]의 반대말이다. [[율리 상수]]와는 달리, 수학 상수는
물리적 측정과는 상관없이 정의된다. 수학 상수는 대개 [[질수체]]나 [[복소수체]]의 원소이다. 우리가 이야기할 수 있는 상수는 (거의 대부분
[[계산 가능한 수|계산 가능]])한 [[정의 가능한 수]]이다. 특정 수학 상수, 예를 들면 [[골룸-刁曼 상수]], [[프랑세즈-로빈슨 상수]], [[제곱근
2| $\sqrt{2}$ ]](sqrt{2})[[math>]], [[리비 상수]]와 같은 상수는 다른 수학상수 또는 할수와 약한 상관관계 또는 깊은 상관관계를 갖는다.= 수학 상수표
==[] class="wikitable"- style="background: #a0e0a0;"| 기호| style="width: 24em;" | 값| 이름| style="width: 3.5em;" | 분류| N| 알려진
때| 알려진 소수점 시릿수| style="background: #d0f0d0; text-align: center;" | ''''[1]<math>i</math>''''| 1| 일. 하나| "[[수학|일반]]"
|[["정수"]]| 고대 BC 500년경| 약| style="background: #d0f0d0; text-align: center;" | ''''[0]<math>\theta</math>''''| 0| 영| "[[수학|일반]]" | "[[대수적
수]]"| 고대 BC 500년경| 약| style="background: #d0f0d0; text-align: center;" | ''''[1]<math>\pi</math>''''| 1| 일. 하나| "[[수학|일반]]"
|[["현수 단위|현수]]단위| 아이(<math>i</math>)| ''''[수학|일반]]''''|[["복소수"]]| 1500년경| 약| style="background: #d0f0d0; text-
align: center;" | ''''[원주율|<math>\pi</math>]''''| ≈ 3.14159 26535 89793 23846 26433 83279 50288| ''''원주율''''|[["수학|일반]]"
|[["조절수"]]| 고대| 1.241,177,300,000|- style="background: #d0f0d0; text-align: center;" | ''''[E (수학상수)|<math>e</math>]''''| ≈
2.71828 18284 59045 23536 02874 12844 901,000|- style="background: #d0f0d0; text-align: center;" | <math>\sqrt{2}</math>| ≈ 1.41421 35623 73095 04886 16887 24289
69807| [[2의 제곱근]], [[달음비]]|[["수학|일반]]|[["대수적 수"]], [[루리수]]| 고대| 137,438,953,444|- style="background: #d0f0d0; text-align:
center;" | <math>\gamma_{\text{自然}}

```

정제 되지 않은 HTML 문법 기반의 txt 파일

04 프로젝트 수행 결과

데이터 전처리

```
# 1차 텍스트 정제 함수
# html파일에서 쓸데없는 정보들을 떼낸 후 개행을 기준으로 텍스트를 요소 별로 분리하여 리스트로 추출
def preprocess_text(text):
    text = re.sub(r'\[\[.*?\]|^\[[^\]]+\]\]', '', text)
    text = re.sub(r'\{\{[^{}]+\}\}', '', text)
    text = re.sub(r'<math>[^<]+</math>', '', text)
    text = BeautifulSoup(text, 'html.parser').get_text(separator='', strip=True)
    tokens = text.split('\n')
    tokens = [token for token in tokens if stop_words not in okt.morphs(token)]

    return tokens

# txt 파일을 읽기 위한 함수
def read_korean_wikipedia_file(file_path):
    with open(file_path, 'r', encoding='utf-8') as file:
        text = file.read()

    processed_text = preprocess_text(text)

    return processed_text
```



TF-IDF Search in Document: ["***펜로즈 삼각형*** (또는)은 불가능한 물체의 일종이다. 1934년 스웨인의 회가 모스카르 하우터프스베르드가 개념을 쓰기 시작했고, 1958년대에 영국의 수학자 로저 펜로즈가 그의 특성을 고안하여, 널리 알렸다. 그 후에도 펜로즈 삼각형은 마우리츠 코르넬리스 매세의 판화에서 쓰이기 시작하여, 그의 작품 속에 등장하는 불가능한 물체에 영향을 주었다. 이 삼각형은 단면이 사각형인 일체인 것처럼 보이지만, 2차원 그림으로만 가능하다. 왜냐하면, 삼각형의 각 변을 이루는 평행한 선들은 각 꼭짓점에 이르면, 서로 다른 위치에서 본 적각의 모서리이기 때문이다. 각 변을 이루는 모서리는 모두 서로 적각을 이루며, 그럼에도 불구하고 삼각형을 만들고자 노력하는 시도는 펜로즈 삼각형으로 확대될 수 있다. 하지만 펜로즈 삼각형은 그 시각적 효과가 삼각형만을 출력적이진 않다. 파일:Penrose pentagon.svg|펜로즈 오각형파일:Penrose hexagon.svg|펜로즈 육각형파일:Penrose octagon.svg|펜로즈 팔각형펜로즈 삼각형처럼 보이는 임체를 만들 수는 있다. 하지만 이 파일은 표이거나, 끊어져야 한다. 파일:Deutsches Technikmuseum Berlin February 2008 0005.JPG|베를린의 독일 기술 박물관에 설치된 펜로즈 삼각형파일:Deutsches Technikmuseum Berlin February 2008 0004.JPG|나쁜 각도에서 본 모습== 같이 보기 ==* 로저 펜로즈* 착시분류:착시분류:위상수학분류:기하학분류:불가능한 물체"]

무의미한 토큰 제거, bs4를 통한 parsing 후 개행을 기준으로 리스트에 append

한국어 표현만 남기고 제거됨

04 프로젝트 수행 결과

모델 학습 과정

```
class RNNModel(nn.Module):
    def __init__(self, vocab_size, embed_dim, hidden_size, num_classes, batch_size):
        super(RNNModel, self).__init__()
        self.embedding = nn.Embedding(vocab_size, embed_dim)
        self.rnn = nn.RNN(embed_dim, hidden_size, batch_first=True)
        self.dropout = nn.Dropout(0.5)
        self.fc = nn.Linear(hidden_size, num_classes)
        self.batch_size = batch_size

    def forward(self, x):
        try:
            embedded = self.embedding(x)
            rnn_out, _ = self.rnn(embedded)
            # overfitting 방지용 dropout을 사용
            rnn_out = self.dropout(rnn_out)
            # output에서 last hidden state만을 뽑아서 return함 중간 과정은 필요없으므로 [batch_size, sequence_length, hidden_size]
            output = self.fc(rnn_out[:, -1, :])

            return output
        except RuntimeError as e:
            return torch.tensor(x)
```

RNN에만 학습을 적용하고 나머지는 base 모델이나
pretrained 모델을 적용함

04 프로젝트 수행 결과

모델 학습 과정

```

361 def _build_rnn_model(self, patience=3):
362     # document의 수열은 0, 1, 2, 3, 4, ... 순서로 indexing
363     self.document_labels = [1 for i in range(len(self.documents))] # Assign class labels
364
365     # RNN layer를 구현하는 코드입니다.
366     model = RNNModel(vocab_size=len(self.rnn_tokenizer.word_index) + 1, embed_dim=256, hidden_dim=128)
367     #交叉熵 손실을 사용하여 손실을 계산하는 데에 대해서는 손실 계산에 대한 자세한 내용은 나중에 다룬다.
368     criterion = nn.CrossEntropyLoss()
369     # weight decay를 사용하여 overfitting을 방지
370     optimizer = optim.Adam(model.parameters(), lr=0.001, weight_decay=0.001)
371     # 학습률 스케줄러를 사용하여 Loss가 줄 때마다 학습률을 조절하는 방법
372     scheduler = lr_scheduler.ReduceLROnPlateau(optimizer, mode='min', patience=2, factor=0.5)
373
374     x_train = torch.tensor(self.rnn_padded_sequences, dtype=torch.long)
375     y_train = torch.tensor(self.document_labels, dtype=torch.long)
376     device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
377     model.to(device)
378
379     try:
380         x_train, x_val, y_train, y_val = train_test_split(
381             self.rnn_padded_sequences,
382             self.document_labels,
383             test_size=0.2,
384             train_size=0.8,
385             random_state=42
386         )
387     except ValueError as e:
388         x_train, x_val, y_train, y_val = [], [], [], []
389     best_loss = float('inf')
390     current_patience = 0
391
392     for epoch in range(30):
393         model.train()
394         total_loss = 0
395         optimizer.zero_grad()
396         outputs = model(torch.tensor(x_train, dtype=torch.long).to(device))
397         loss = criterion(outputs, torch.tensor(y_train, dtype=torch.long).to(device))
398         loss.backward()
399         optimizer.step()
400         total_loss += loss.item()
401         avg_train_loss = total_loss / len(x_train)
402
403         model.eval()
404         total_val_loss = 0
405         with torch.no_grad():
406             val_outputs = model(torch.tensor(x_val, dtype=torch.long).to(device))
407             val_loss = criterion(val_outputs, torch.tensor(y_val, dtype=torch.long).to(device))
408             total_val_loss += val_loss
409             avg_val_loss = total_val_loss / len(x_val)
410
411         print(f'Epoch {epoch + 1}, Training Loss: {avg_train_loss}, Validation Loss: {avg_val_loss}')
412
413         if avg_val_loss < best_loss:
414             best_loss = avg_val_loss
415             current_patience = 0
416         else:
417             current_patience += 1
418
419         scheduler.step(avg_val_loss)
420         if current_patience >= patience:
421             print(f'Early stopping after {(epoch + 1)} epochs.')
422             break
423
424     return model

```

Overfitting을 방지하기 위해 여러가지 기법을 사용

04 프로젝트 수행 결과

서치 함수

```
def dpr_search(self, query, top_k=1):
    # search 함수에는 query와 document를 tokenize 한 다음 document의 element와 query 간의 코사인유사도가 가장 높은 값을 찾아내어 내림차순으로 sort하여 topk 만큼 추출함
    # 2개 빼고 모든 search 함수에는 이 방법을 사용함
    query_tokenized = self.dpr_tokenizer(query, return_tensors="pt")
    question_embedding = self.dpr_question_encoder(**query_tokenized)["pooler_output"]
    document_embeddings = self.dpr_context_encoder(input_ids=query_tokenized["input_ids"],
                                                    attention_mask=query_tokenized["attention_mask"])["pooler_output"]
    cosine_scores = util.pytorch_cos_sim(question_embedding, document_embeddings)[0]
    most_similar_indices = torch.argsort(cosine_scores, descending=True)[:top_k]

    return [self.documents[index] for index in most_similar_indices]
```

inverted index search와 RNN 모델을 제외하고는
유사한 기법을 사용

04 프로젝트 수행 결과

서치 함수

```
def inverted_index_search(self, query, top_k=1):
    # query 벡터를 0벡터로 초기화
    query_vector = np.zeros(len(self.inverted_index))
    feature_names = list(self.inverted_index.keys())

    # query에서 불용어를 제외하고 단어를 추출
    for term in query.split():
        if term in feature_names and term not in stop_words:
            term_index = feature_names.index(term)
            query_vector[term_index] = 1

    # 사전에 추출했던 document의 element와 query가 가장 유사한 케이스를 추출하여 그 정도와 함께 dict에 넣음
    matching_documents = defaultdict(float)
    for term_index, term in enumerate(feature_names):
        if query_vector[term_index] > 0:
            for doc_index in self.inverted_index[term]:
                matching_documents[doc_index] += 1

    # dict를 유사도 순으로 sort한 dictionary이므로 sorting이 끝남
    sorted_documents = sorted(matching_documents.keys(), key=lambda x: matching_documents[x], reverse=True)

    return [self.documents[index] for index in sorted_documents[:top_k]]
```

inverted index search

04 프로젝트 수행 결과

서치 함수

```
def rnn_search(self, query, top_k=1):
    # query를 tokenize
    sequence = self.rnn_tokenizer.texts_to_sequences(query)
    # padding
    padded_sequence = pad_sequences(sequence, maxlen=len(self.rnn_padded_sequences[0]))
    x_input = torch.tensor(padded_sequence, dtype=torch.long).to(next(self.rnn_model.parameters()).device)

    output = self.rnn_model(x_input)
    # 확률의 softmax를 취하여 정규화
    probabilities = nn.functional.softmax(output, dim=1).cpu().detach().numpy()[0]
    # 가장 가능성의 높은 순서대로 sort한 다음 top_k 값만을 추출
    most_probable_indices = probabilities.argsort()[-top_k:][::-1]

    return [self.documents[index] for index in most_probable_indices]
```

RNN search

04 프로젝트 수행 결과

메인 함수

```
if __name__ == "__main__":
    file_path = '/data/workspace/junow/haystack/small_example.txt' # for test
    # 시간이 너무 오래걸려서 원본 파일의 일부분만을 추출한 small example txt 파일을 사용
    # file_path = '/data/workspace/junow/haystack/kowiki-20220220-001.txt'
    documents = read_korean_wikipedia_file(file_path)
    search_model = SearchModel(documents)
```

원본 txt파일이 너무 커서 모델 서치 중 killed되는 현상이 생겨서
일부분만을 추출해서 test용으로 사용함

04 프로젝트 수행 결과

메인함수

특징

- while문을 통해 query와 context를 입력받음
- 0을 누르면 프로그램 종료
- query와 context를 사용자로부터 입력받고 search
- t5의 search result를 가장 먼저 보여주고 모델과 비교

```

while input('enter 0 to break or enter other word to move on: ') != '0':
    # 일단 query를 입력
    query = input("enter your search query: ")
    # query와 맞는 context (topic)를 입력
    contexts = input("enter your topic of search query: ")
    messages = [{"role": "context", "content": context} for context in contexts]
    messages.append({"role": "user", "content": query})

    # t5 search (reader)
    input_q = json.dumps({
        "model": "t5-small-fid",
        "mode": "",
        "messages": messages})
    response = requests.post('http://211.39.140.48:9090/predictions/temp', data=input_q)
    t5_prediction_result = response.json()
    print("\nT5-large Prediction Result:", t5_prediction_result)
    # 모델 별로 결과값을 출력하게 함
    for n, (model_name, search_function) in enumerate([
        ("TF-IDF", search_model.tfidf_search), # 기본 모델
        ("BM25", search_model.bm25_search), # 기본 모델
        ("Inverted Index", search_model.inverted_index_search), # 기본 모델
        ("RNN", search_model.rnn_search), # 기본모델, 학습 o
        ("DPR", search_model.dpr_search)]): # finetuned model, 학습 x
        
```

04 프로젝트 수행 결과

메인함수

특징

- context를 찾지 못하거나 너무 부정확한 결과가 나오는 것을 대비해 try except 문을 통한 error handling 을 함

```
try:  
    result = search_function(contexts, top_k=top_k)  
# context를 아예 찾지 못할 경우를 대비해서 try except문을 사용  
except TypeError as e:  
    print(e)  
    print("[]")  
    print('0%')  
    continue  
except ValueError as e:  
    print(e)  
    print("[]")  
    print('0%')  
    continue
```

04 프로젝트 수행 결과

메인함수 특징

- Reader 함수 대신 Retrieval 모델을 사용함
- 이를 통해 추출한 문서 내에서 query에 해당하는 부분을 보여줄 수 있음

```
# 추출된 result document element 그러니까 context에 해당하는 문서에서 query에 해당하는 부분을 다시 search하여 출력하는 부분입니다
if similarity * 10 > 1:
    candidate_list = deep_preprocess_text(result[0])
    # document에서 context 단어에 해당하는 문서를 찾고 그 문서에서 다시 query에 해당하는 part를 찾습니다
    # 처음에는 GPT랑 랜덤으로 통해 QA를 하려 했으나 잘 안되서 search model을 한번 더 써보기로 합니다
    # search 모델을 두번쓰는 것이 가장 효과가 좋았습니다
    deep_search_model = SearchModel(candidate_list)
    # 변수를 쓰려고 했더니 파일에서 이를 class의 함수로 인식을 해버려서 그냥 새로 deep search model을 만들었습니다
    # 구조는 같음
    try:
        # enumerate를 통해 인덱스를 추출합니다을 같은 모델을 중복하여 사용합니다
        # 1번: TF-IDF
        if n == 1:
            ans = deep_search_model.tfidf_search(query, top_k=top_k)
            print(f"\n{model_name}'s answer in Document:", ans)
            print('')
        # 2번 BM25
        elif n == 2:
            ans = deep_search_model.bm25_search(query, top_k=top_k)
            print(f"\n{model_name}'s answer in Document:", ans)
            print('')
        # 3번 Inverted Index
        elif n == 3:
            ans = deep_search_model.inverted_index_search(query, top_k=top_k)
            print(f"\n{model_name}'s answer in Document:", ans)
            print('')
        # 4번 RNN
        elif n == 4:
            ans = deep_search_model.rnn_search(query, top_k=top_k)
            print(f"\n{model_name}'s answer in Document:", ans)
```

04 프로젝트 수행 결과

모델 출력 결과 예시

학습 성능

query 별로 학습 성능을 구현해서 일일이 평가해야했으나 시간 상의 이슈로 실패

```
enter your search query: 펜로즈 삼각형의 넓이를 구할 수 있는가?
enter your topic of search query: 펜로즈 삼각형
```

TF-IDF Search in Document: [“‘펜로즈 삼각형’”(또는)는 불가능한 물체의 일종이다. 1934년 스웨덴의 화가 오스카르 레우테르스베르드가 처음 쓰기 시작했고, 1950년대에 영국의 수학자 로저 펜로즈가 그와는 독자적으로 고안하여, 널리 알렸다. 그 후에도 펜로즈 삼각형은 마우리츠 코르넬리스 에셔의 판화에서 쓰이기 시작하여, 그의 작품 속에 등장하는 불가능한 물체에 영향을 주었다. 이 삼각형은 단면이 사각형인 것처럼 보이지만, 2차원 그림으로만 가능하다. 왜냐하면, 삼각형의 각 변을 이루는 평행한 면들은 각 꼭짓점에 이르면, 서로 다른 위치에서 본 각각의 모서리이기 때문이다. 각 변을 이루는 막대는 모두 서로 직각을 이루며, 그럼에도 불구하고 삼각형을 만든다. 이 방법을 일반화 시켜서 펜로즈 다각형으로 확대할 수 있다. 하지만 펜로즈 사각형은 그 시작적 효과가 삼각형만큼 충격적이진 않다. 파일:Penrose square.svg|펜로즈 사각형파일:Penrose pentagon.svg|펜로즈 오각형파일:Penrose hexagon.svg|펜로즈 육각형파일:Penrose octagon.svg|펜로즈 팔각형펜로즈 삼각형처럼 보이는 입체를 만들 수는 있다. 하지만 이 때에 각 변은 꼬이거나, 끊어져야 한다. 파일:Deutsches Technikmuseum Berlin February 2008 0005.JPG|베를린의 독일 기술 박물관에 설치된 펜로즈 삼각형파일:Deutsches Technikmuseum Berlin February 2008 0004.JPG|다른 각도에서 본 모습== 같이 보기 ==* 로저 펜로즈* 착시분류:착시분류:위상수학분류:기하학분류:불가능한 물체”]

```
TF-IDF's similarity to a label document: 97.53%
Some weights of the model checkpoint at facebook/dpr-ctx_encoder-multiset-base were not used when initializing
    'bert_model.pooler.dense.bias', 'ctx_encoder.bert_model.pooler.dense.weight'
- This IS expected if you are initializing DPRContextEncoder from the checkpoint of a model trained on another
    architecture (e.g. initializing a BertForSequenceClassification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing DPRContextEncoder from the checkpoint of a model that you expect
    to initialize a BertForSequenceClassification model from a BertForSequenceClassification model.
Some weights of the model checkpoint at facebook/dpr-question_encoder-multiset-base were not used when initializing
    'question_encoder.bert_model.pooler.dense.weight', 'question_encoder.bert_model.pooler.dense.bias'
- This IS expected if you are initializing DPRQuestionEncoder from the checkpoint of a model trained on another
    architecture (e.g. initializing a BertForSequenceClassification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing DPRQuestionEncoder from the checkpoint of a model that you expect
    to initialize a BertForSequenceClassification model from a BertForSequenceClassification model).
Epoch 1, Training Loss: 0.22986366198613092, Validation Loss: 0.6721245050430298
Epoch 2, Training Loss: 0.1871013091160701, Validation Loss: 0.6749445199966431
Epoch 3, Training Loss: 0.15983005795112023, Validation Loss: 0.6785068511962891
Epoch 4, Training Loss: 0.11898108629079965, Validation Loss: 0.6816779971122742
Epoch 00004: reducing learning rate of group 0 to 5.0000e-04.
Early stopping after 4 epochs.
Some weights of ElectraForSequenceClassification were not initialized from the model checkpoint at beomi/KcELECTRA initialized: ['classifier.out_proj.weight', 'classifier.out_proj.bias', 'classifier.dense.bias', 'classifier.dense.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.

TF-IDF's answer in Document: ['svg|펜로즈 팔각형펜로즈 삼각형처럼 보이는 입체를 만들 수는 있다']
```

05 자체 평가 의견

메인:

- 좋았던 점: 1. 여러 모델 사이의 결과값을 직관적으로 비교할 수 있다는 점
2. NLP Retrieval을 두번 사용하여 Reader 모델을 생략하고 구현한 점
- 아쉬웠던 점: 1. 여러 모델 중에 정답을 제대로 내놓은 것도 있고 아닌 것도 있었는데 그 이유를 알 수 없어 아쉬웠다
2. Reader 모델을 정석으로 구현했을 때 성능이 제대로 나오지 않아 아쉬웠다

결론

여러가지 search 모델을 구현해서 이를 비교하고 직관적으로 결과값을 내놓는 데에는 성공함
평가를 위한 query 모델을 loop 문을 돌면서 직접 정답값과 비교하고 그에 따른 자카드 유사도를 내놓는 데에는
시간상의 문제로 실패하여 모델 간의 정확한 수치의 정확도를 내놓는 데에는 어려움을 겪음
그럼에도 Reader 모델을 Retrieval 모델로 대체한 점, 긴 코드를 구현하고 모델에 대한 이해도를 높인 점
최신 SOTA 모델을 구현한 점, 라이브러리 충돌 문제 등을 해결한 점은 고무적임
기회가 된다면 모델 간의 성능을 평가해보는 것까지 구현해보고 싶음

Thank you

이제 소스 코드를 간략하게 리뷰 하겠습니다