


MRC jobplz조 3rd Solution 발표

팀원 : 한현우, 최석환 , 장동재, 박상기, 김원배, 김명수


 Dongjae/mrc2reader
📄 Question Answering · Updated 9 days ago · 93

 thingsu/koDPR_context
Updated 2 hours ago · 212

 thingsu/koDPR_question
Updated 2 hours ago · 197

 CodeNinja1126/bert-p-encoder
Updated 9 days ago · 1

 CodeNinja1126/koelectra-model
Updated Apr 18

 CodeNinja1126/xlm-roberta-large-kor-mrc
📄 Question Answering · Updated 2 days ago · 13

목차

- ❑ Retrieval
 - Elasticsearch
 - Dense Passage Retrieval
 - Sparse Retrieval
- ❑ Pre-Processing
- ❑ Reader Model
- ❑ Post-Processing
- ❑ Ensemble
- ❑ 시도했으나 삽질한 것
 - 형태소 Tokenizer
 - Soft Voting

개요

Pre-Processing	Korean Sentence Splitter (KSS) Delete Trash ID, Special Characters Use Origin Datasets
Retrieval	Dense Passage Retrieval (ICT, In-Batch Negatives) Sparse Retrieval (BM25, TF-IDF) Elasticsearch - Thingsu/KoDPR_context(Hugging Face)
Reader	XLM-RoBERTa-large-squad2 XLM-RoBERTa-large KoElectra-v3 - Dongjae/mrc2reader(Hugging Face) - CodeNinja1126/xlm-roberta-large-kor-mrc(Hugging Face)
Post-Processing	조사 제거(konlpy-kkm) Retrieval Score 사용
Ensemble	Hard Voting (ratio)

- ❑ **Sparse Retrieval**
 1. **TF-IDF, BM25**
 2. **Elasticsearch**

- ❑ **Dense Passage Retrieval**

Sparse Retrieval with TF-IDF, BM25

I

● TF-IDF

- 정의 : 단어의 빈도와 역 문서 빈도(문서의 빈도에 특정식을 취함)를 사용하여 DTM 내의 각 단어들마다 중요한 정도를 가중치로 주는 방법
- Action : TfidfVectorizer 선언시, max_features 제한 해제

● BM25(a.k.a Okapi BM25)

- 정의 : 주어진 쿼리에 대해 문서와의 연관성을 평가하는 랭킹 함수
- Action : Custom으로 BM25 알고리즘 구현, 그러나, EM기준 48% 대에서는 한계가 있었음

48.33%

62.40%



46.25%

60.02%

	EM	F1
Before	14.17%	21.04%
After	28.75%	41.05%

10	2021-05-03(3) 전 제출에서 bin 파일 오버라이팅함	20.42%	29.86%
----	---------------------------------------	--------	--------



8

2021-05-03 submit(1)

16.25%

21.39%

Sparse Retrieval with Elastic Search(1)



elasticsearch

- 루씬(lucene) 기반의 검색 엔진
- Elastic Search는 **BM25** 기반의 알고리즘으로 주어진 context들을 검색
- 도입 결과 : 동일한 파라미터 및 topk=25, EM 기준 47.02% -> **50.82%**로 성능 향상이 있었음

* references

- 오피스 아워
- 김남혁 캠퍼님 토론 게시판

<http://boostcamp.stages.ai/competitions/31/discussion/post/314>

39	elastic 미적용	47.08%	68.51%	{"training": {}, "inference": {}}	<input type="checkbox"/>	
38	elastic search 적용	50.42%	71.26%	{"training": {}, "inference": {}}	<input type="checkbox"/>	

Sparse Retrieval with Elastic Search(2)

I

- Sparse Retrieval with E.S
 - Elastic Search Code(1)
 - Elastic Search Code(2)

```
def retrieve(self, query_or_dataset="", topk=100):
    es, es_server = self.connect()

    print(es.info())
    if not es.indices.exists(index="document"):
        try:
            self.indices_create(es)
        except:
            es.indices.delete('document')
            self.indices_create(es)

    with open('/opt/ml/code/dongjae/data/wikipedia_documents.json', 'r') as f:
        wiki_data = pd.DataFrame(json.load(f)).transpose()

    trash_ids = [973, 4525, 4526, 4527, 4528, 4529, 4530, 4531, 4532, 4533, 4534, 5527,
                 9079, 9080, 9081, 9082, 9083, 9084, 9085, 9086, 9087, 9088, 28989, 29028,
                 31111, 37157]

    new_wiki = [(vv[:10], vv.strip()) for vv in list(dict.fromkeys([v['text'] for v in wiki_data.values()]))]
    new_wiki_title = []
    new_wiki_text = []

    for current_title, current_text in tqdm(new_wiki):
        try:
            te = kss.split_chunks(current_text, max_length= 1280, overlap = True)
            for i in range(len(te)):
                new_wiki_title.append(current_title + str(i))
                new_wiki_text.append(te[i].text)
        except:
            new_wiki_title.append(current_title)
            new_wiki_text.append(current_text)
            continue
```

Sparse Retrieval with Elastic Search(3)

- Sparse Retrieval with E.S
 - Elastic Search Code(1)
 - Elastic Search Code(2)

```
for idx, example in enumerate(tqdm(query_or_dataset, desc="Sparse retrieval with Elastic Search: ")):

    question=example['question']
    query = {
        'query':{
            'bool':{
                'must':[
                    {'match':{'text':question}}
                ],
                'should':[
                    {'match':{'text':question}}
                ]
            }
        }
    }

    doc = es.search(index='document',body=query,size=topk)['hits']['hits']

    doc_scores=[]
    doc_contexts=[]
    doc_contexts_ids=[]
    for idx,i in enumerate(doc):
        doc_scores.append(i['_score'])
        doc_contexts.append(i['_source']['text'])
        doc_contexts_ids.append(i['_source']['context_id'])

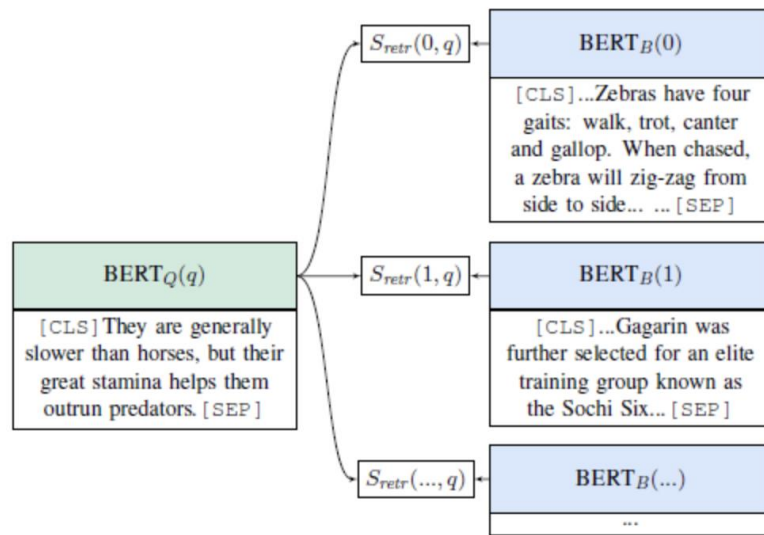
    tmp = {
        "question": example["question"],
        "id": example['id'],
        "context_id": doc_contexts_ids[0],
        'context' : ' '.join([ctx for ctx in doc_contexts]) # 하나로 잇기
    }
    if 'context' in example.keys() and 'answers' in example.keys():
        tmp["original_context"] = example['context'] # original document
        tmp["answers"] = example['answers'] # original answer
    total.append(tmp)

cqas = pd.DataFrame(total)
es_server.kill() # 서버 종료
return cqas
```


Dense Passage Retrieval

I

Inverse Cloze Task



Latent Retrieval for Weakly Supervised Open Domain Question Answering

- Context에서 문장을 랜덤으로 추출해 query로 활용해 학습
- Donggykim/Inverse-cloze-task를 아래와 같이 수정해 사용
(한국어 데이터셋에 맞게 수정, 문장 순서 셔플 추가, dual encoder 방식으로 변경)

×

In-Batch Negative



- 각 query에 대해 TF-IDF 점수가 높은 passage 랜덤으로 배치에 추가

Dense vs Sparse

I

(Top 30 기준)	Dense Retrieval	Sparse Retrieval	Mix(K=0.8)
Dense가 놓친 경우	76 -1 (>100)	5 1	7 2
Sparse가 놓친 경우	8 5	55 33	19 13
동시에 있는 경우	1 3	17 0	26 0

	Dense	Sparse	Mix
1	76	5	7
2	0	0	0
3	-1	1	2
	0	0	0
	1	1	1
	-1	1	2
	8	55	19
	3	0	0
	0	0	0
	-1	-1	-1
	2	1	1
	7	0	0
	1	17	26
	0	12	15
	47	4	14
	70	0	0
	1	0	0
	0	0	0
	0	0	0
	0	2	6
	9	0	0
	5	6	10
	5	33	13
	17	57	27

Dense Retrieval과 Sparse Retrieval의 결과가 겹치는 비율이 낮아 Ensemble했을 때 효과 있었음
 위 Mixed Retrieval로 추린 후 RECONSIDER 적용했으면 좋은 효과가 있을 것으로 예상

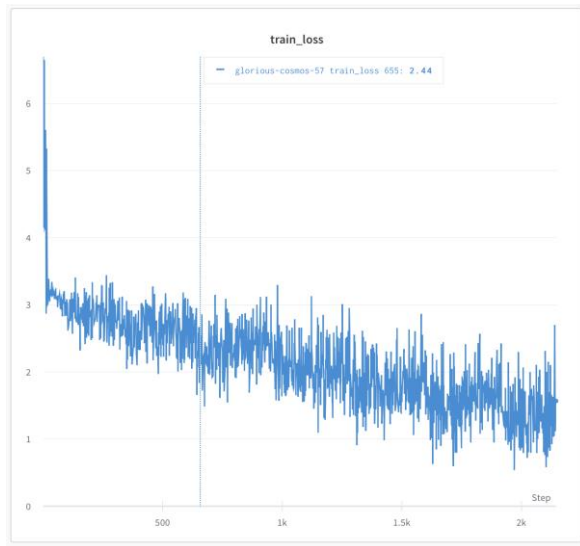
RECONSIDER

I

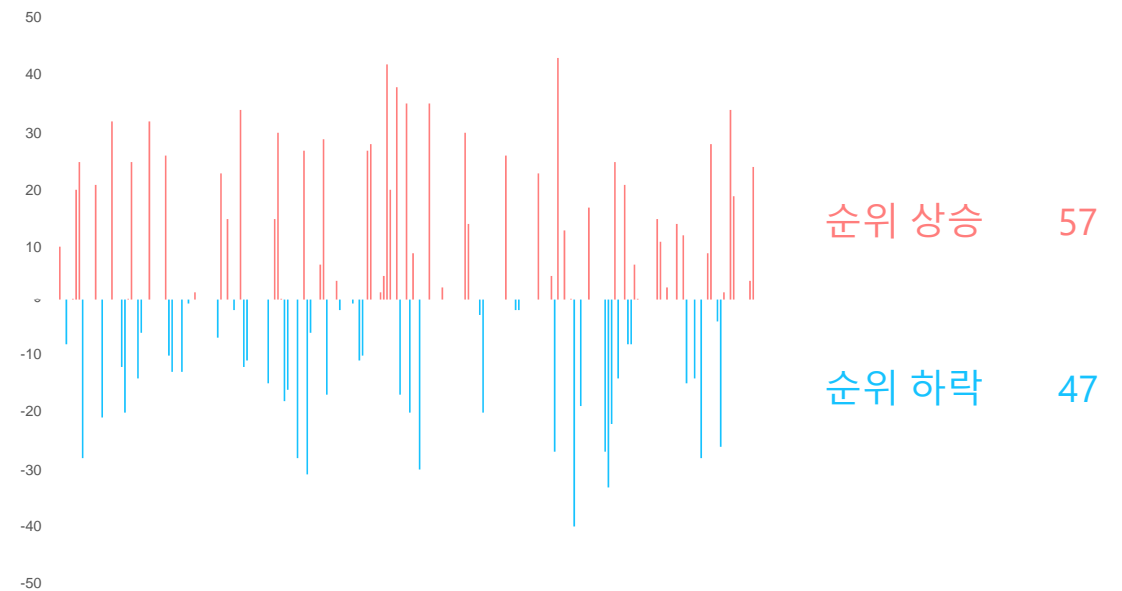
Train

베이스라인의 retrieval를 통해 관련
passage를 도출해 negative sample(M개)로 활용

Batch size = 1, M = 23, accumulation step = 16으로 학습

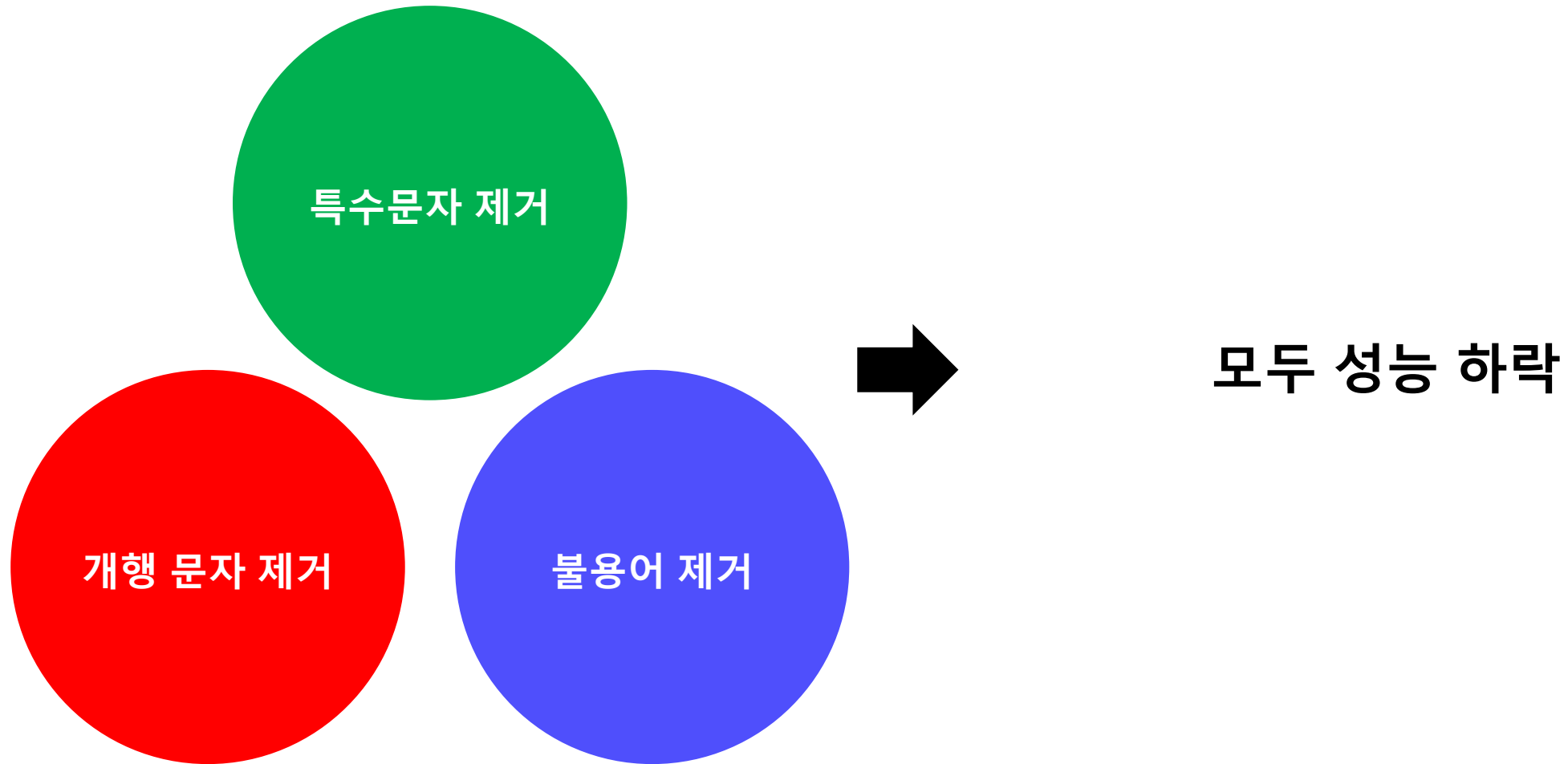


실패



실험 조건 상 1 epoch에 1시간 30분 소요
1, 3, 5 epoch의 모델을 비교했을 때 성능이 좋지 않아 폐기

Pre-processing



Pre-processing - KSS Library (Korean Sentence Splitter)

Context 길이를 짧게 통일하면
Embedding 및 Prediction에
도움되지 않을까?

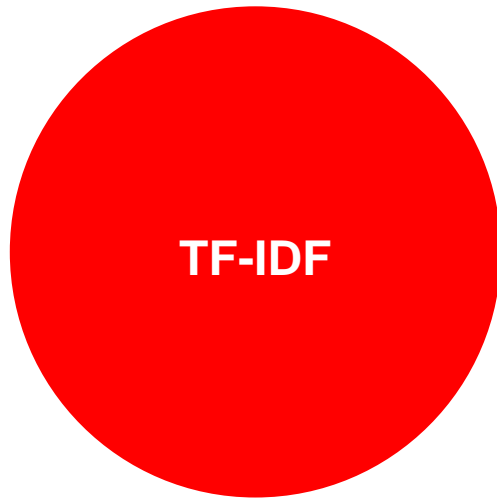


kss.split_chunk()

```
# 위키피디아 kss로 잘라서
wiki1280_name = 'wiki_1280_kss.pickle'

if os.path.isfile(os.path.join(self.data_path, wiki1280_name)) :
    with open(os.path.join(self.data_path, wiki1280_name), 'rb') as file :
        self.contexts = pickle.load(file)
else :
    # 글자수로 자를 때
    new_context = [v['text'] for v in wiki.values() if v['document_id'] not in trash_ids]
    self.contexts = []
    for current in new_context :
        try :
            te = kss.split_chunks(current, max_length= 1280, overlap = True)
            for v in te :
                self.contexts.append(v.text)
        except :
            self.contexts.append(current)
            continue
    # 위키피디아 피클 저장
    with open(os.path.join(data_path, wiki1280_name), "wb") as file:
        pickle.dump(self.contexts, file)
```

Pre-processing - KSS Library (Korean Sentence Splitter)



EM	F1
----	----

Baseline 52.50% 65.78%

Kss.split_chunk() 56.25% 72.33%

성능 향상



EM	F1
----	----

Baseline 58.33% 72.87%

Kss.split_chunk() 58.33% 71.72%

성능 하락

□ Model

1. xlm-roberta-large

- xlm-roberta-large
- xlm-roberta-large-squad2(deepset)

일반화 성능을 위하여 다르게 사전학습된 두 모델을 사용

2. KoElectra-v3(킹갓☆monologg☆)

□ Retrieval Score 활용

1. $\text{start_logit} * \text{end_logit} * \text{retrieval_score}$
각 점수를 softmax로 확률로 변환,
곱하기로 최종답안 점수 계산
2. $\text{start_logit} + \text{end_logit} + \text{retrieval_score}$
각 점수를 $[0, 1]$ 범위로 정규화,
모든 점수를 더해 최종답안 점수 계산

□ 조사 제거

Konlpy 라이브러리 사용, 마지막 형태소가 조사일 경우 제거했다.
꼬꼬마 형태소 분석기 사용

□ Hard Voting

- 지금까지 적용된 기술을 각자 조합해 결과물 출력
- 적절한 비율로 조합해 결과가 잘 나오기를 기도
- 50 중후반 정도였던 싱글 모델 EM 성능을 64.17까지 끌어올릴 수 있었다.



아이디어의 99.97%는 실패한다

- 형태소 tokenizer 적용하기

- (Baseline) prepare_train_features / prepare_validation_features
question + context

- > token_start_index + token_end_index + offset_mapping

- > decoded answer ≠ real answer

- decoded answer ≠ real answer인 case들의 특징 = 조사

- ex) real answer = '모스크바' / decoded answer = '모스크바로'

- 추측 : real answer를 정확하게 tokenize 하면 model이 더 정확한 문맥을 학습할 것이다.

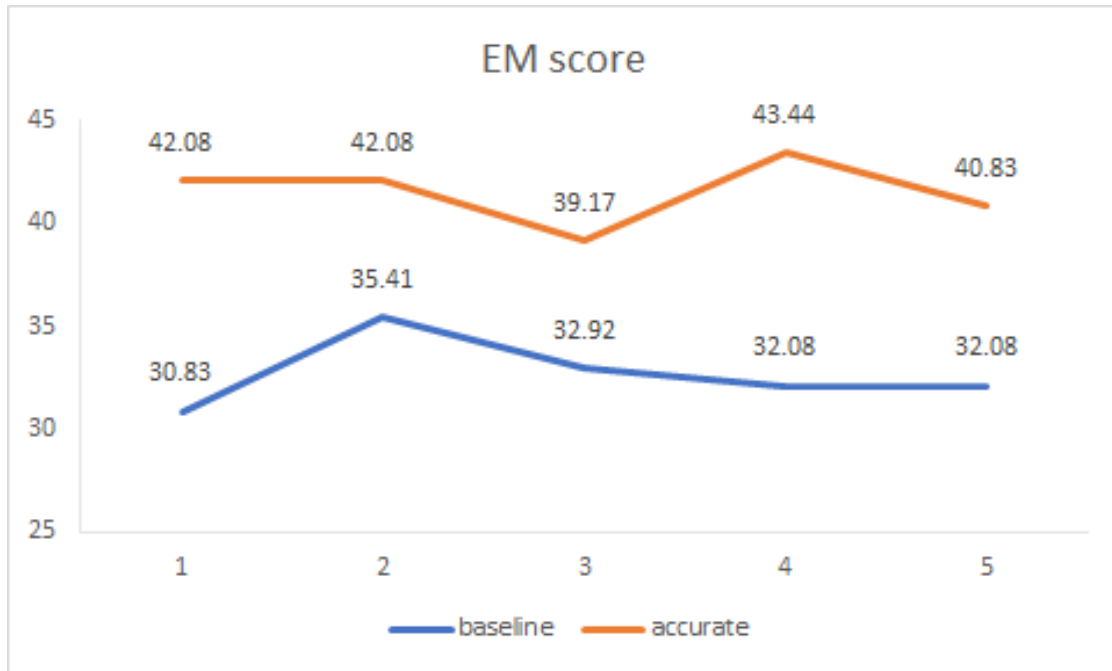
아이디어의 99.97%는 실패한다

- 형태소 tokenizer 적용하기

- (Accurate) prepare_train_features / prepare_validation_features
question + context
 - > mecab tokenizer
 - > bert tokenizer
 - > token_start_index + token_end_index + offset_mapping
 - > decoded answer = real answer
- klue['train']에서 발생한 decoded answer \neq real answer cases
baseline = 1033, accurate = 10 (형태소 tokenizer의 성능에 영향)

아이디어의 99.97%는 실패한다

● 형태소 tokenizer 적용하기



- `train_dataset = klue['train']`
- `valid_dataset = klue['validation']`
- `model = 'bert-base-cased'`
- tokenizer 마다 '##', '_' 등 prefix 가 붙는 방식이 달라서 reader 모델마다 다른 함수 필요
- baseline decoded answer에 후처리를 적용했을 때의 EM score와 비교 필요
- 구현 시점이 늦어 모델에 반영 실패

개선할 만한 것들

- Slack을 사용해서 소통했지만 더 많은 내용을 공유하며 체계적으로 실험관리를 했어야 한다.
- 협업과 분업 적절히 섞기
- Klue에서 사용했던 idea들을 도입하려는 시도
- 프로젝트 기간에 맞춰 일정 관리
- 파일 관리, 결과 관리를 통해 재현가능한 실험을 진행했어야 한다.
- 논문 서치를 통해 다양한 아이디어 얻기

References

- <https://blog.naver.com/PostView.nhn?blogId=duqrlwjddns1&logNo=221782081265&redirect=Dlog&widgetTypeCall=true&directAccess=false>
- <https://littlefoxdiary.tistory.com/12>
- <https://arxiv.org/abs/2004.04906>