DPR을 이용한 검색 솔루션
(with WISEnut)

다차원 벡터 활용을 통한 문서 유사도 검색

팀명 현명한호두
팀장 홍세일
팀원 정재훈 이정호 유예지 조은상

목차

table of contents

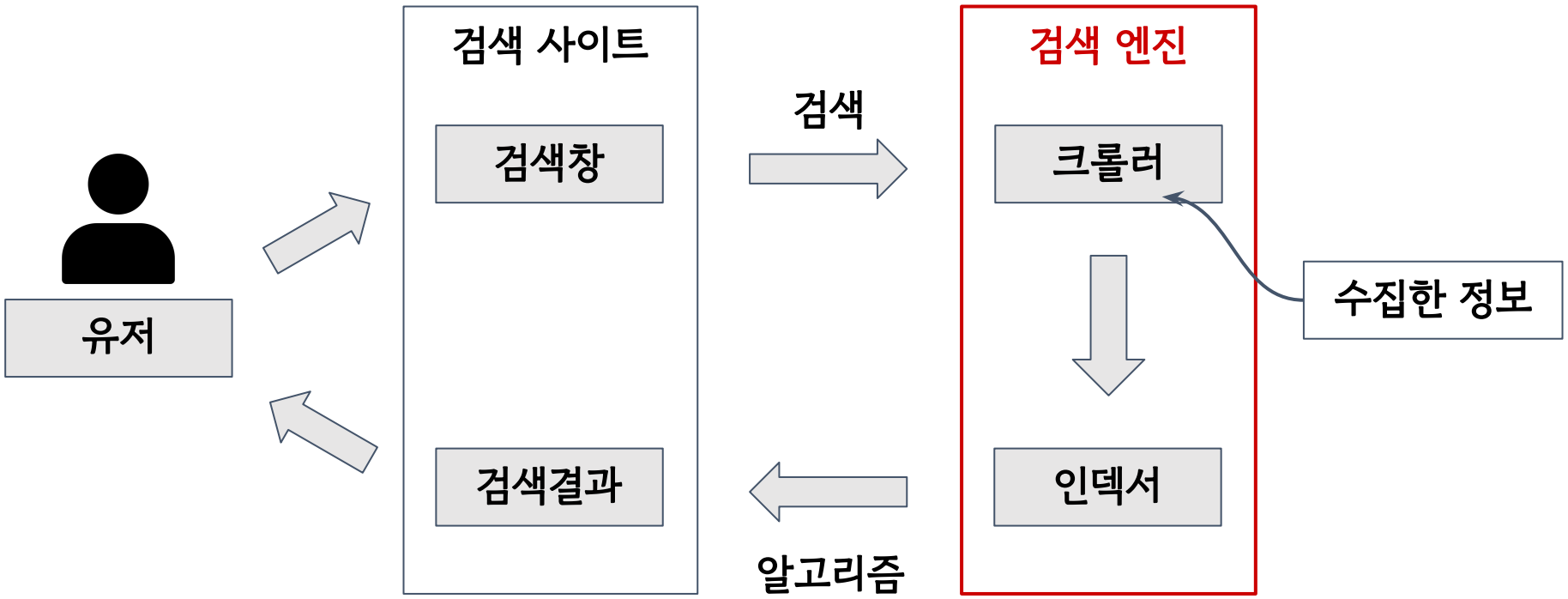
1

프로젝트 배경

‘검색 엔진’ 이란?

- 검색 엔진 (Search Engine)

➔ 사용자가 웹에서 특정 정보를 찾을 때, 검색어를 입력하면 그 검색어와 관련된 웹페이지를 찾아주는 프로그램



배경 및 문제점

BM25 / TF-IDF

키워드 파악은 가능하나 문맥 및 동의어를 구분하지 못함

ORQA

장점

- ICT를 사용함으로써 시간 단축
- 추가적인 pretraining 없이 적은 데이터로 dense vector/matrix를 활용하여 모델 생성 가능

단점

- Regular sentence 가 좋은지 명확하지 않음
- Fine-tuning 을 하지 않으므로 suboptimal 이 될 수 있음

DPR

in-batch negative sampling 사용

문맥 및 동의어 파악 가능

2

Dense Passage Retrieval

참고 논문

Dense Passage Retrieval for Open-Domain Question Answering

https://arxiv.org/pdf/2004.04906.pdf

‘DPR (Dense Passage Retrieval)’ 이란?

< Retrieval 의 Backbone Model >

‘사전훈련된 BERT model’ 로 이루어진 ‘2개의 Encoder’ (Question Encoder, Passage Encoder)’ 활용

< 유사도 측정 방법 >

question 벡터와 passage 벡터의 내적 (dot product)

$$\text{sim}(q, p) = E_Q(q)^\intercal E_P(p).$$

< Model Training >

loss 를 구하는 식 :

NLL (Negative Log Likelihood)

$$L(q_i, p_i^+, p_{i,1}^-, \cdots, p_{i,n}^-) = -\log \frac{e^{\text{sim}(q_i, p_i^+)}}{e^{\text{sim}(q_i, p_i^+)} + \sum_{j=1}^{n} e^{\text{sim}(q_i, p_{i,j}^-)}}.$$

< Negative sample >

loss 가 감소하는 방향으로 학습하기 위해 중요

- Random : 코퍼스 내의 random한 passage를 선택
- BM25 : 코퍼스 내에서 BM25 기준으로 top-k의 문서 사용
- Gold : 학습셋 내의 다른 질의의 positive passage 선택

< In-batch negatives >

모델 학습 시 같은 batch 내에서 (In-batch) gold passage 를 negative passage 로 활용하는 것 (재사용)
계산적인 효율 뿐만 아니라, 좋은 성능을 낼 수 있게 함

$$S = QP^T S = QP^T \text{ (BxB)}$$

DPR 학습 결과

< test set 의 retrieval 정확도 >

Training0	Retriever	top20					top-100				
		NQ	TriviaQA	WQ	TREC	SQuAD	NQ	TriviaQA	WQ	TREC	SQuAD
None	BM25	59.1	66.9	55.0	70.9	68.8	73.7	76.7	71.1	84.1	80.0
Single	DPR	78.4	79.4	73.2	79.8	63.2	85.4	85.0	81.4	89.1	77.2
	BM25+ DPR	78.4	79.8	71.0	85.2	71.5	83.8	84.5	80.5	92.7	81.3
Multi	DPR	79.4	78.8	75.0	89.1	51.6	86.0	84.7	82.9	93.9	67.6
	BM25+ DPR	78.0	79.9	74.7	88.5	66.2	83.9	84.4	82.3	94.1	78.6

- Single : 각각의 Dataset에 대하여 학습시킴
- Multiple : SQuAD를 제외한 4개의 Dataset을 합쳐서 학습시킴
- 40 epoch (NQ, TriviaQA, SQuAD)
- 100 epoch (TREC, WQ)
- dropout ratio : 0.1
- optimizer : Adam
- top k : 상위 k개의 retrieval 정확도를 가지고 측정

DPR 학습 결과

< NQ 데이터셋에 대해
서로 다른 학습 방법을 테스트한 결과 >

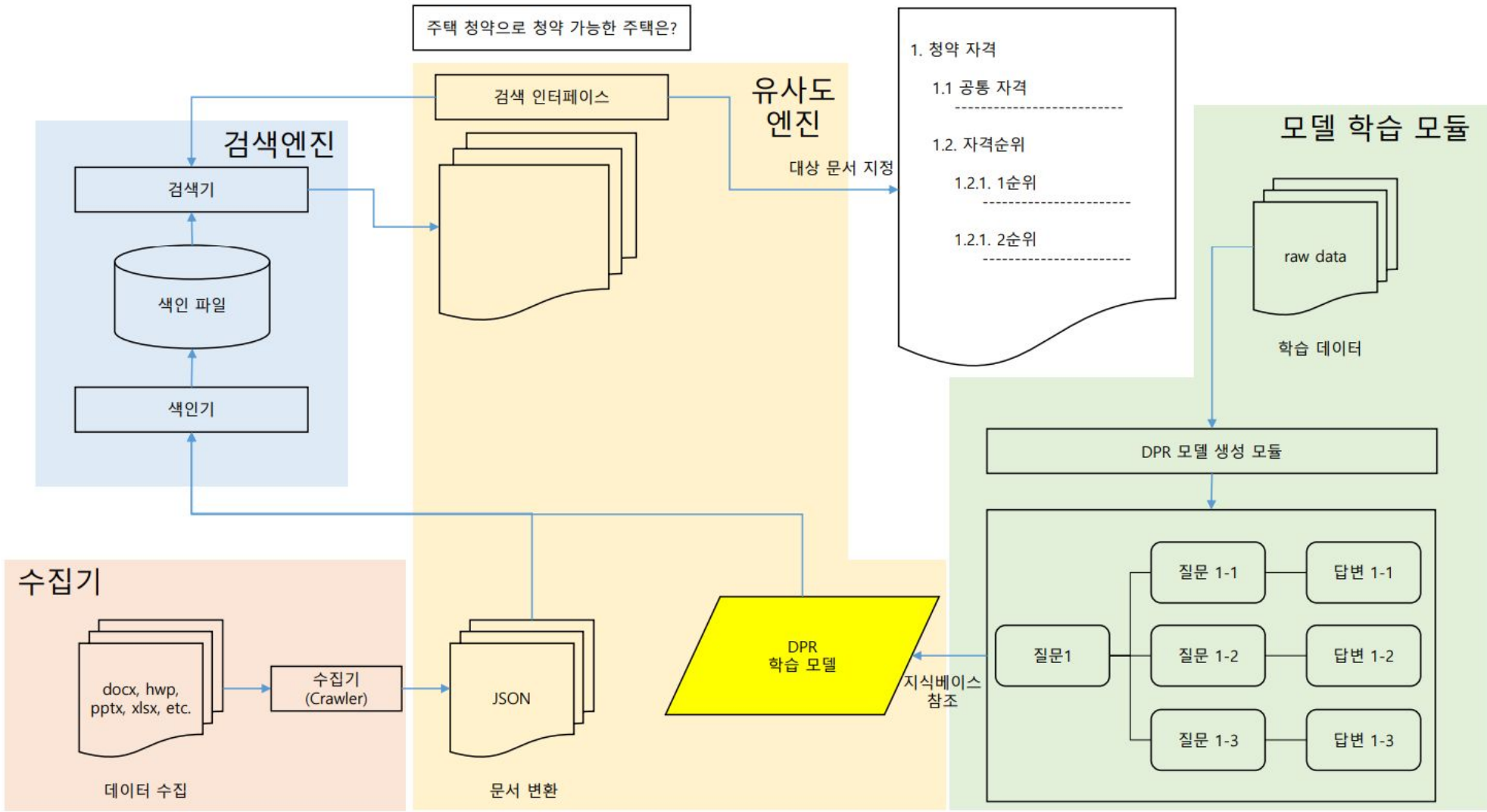
Type	#N	IB	Top-5	Top-20	Top-100
Random	7	✗	47.0	64.3	77.8
BM25	7	✗	50.0	63.3	74.8
Gold	7	✗	42.6	63.1	78.3
Gold	7	✓	51.1	69.1	80.8
Gold	31	✓	52.1	70.8	82.1
Gold	127	✓	55.8	73.0	83.1
G.+BM25$^{(1)}$	31+32	✓	65.0	77.3	84.4
G.+BM25$^{(2)}$	31+64	✓	64.5	76.4	84.0
G.+BM25$^{(1)}$	127+128	✓	**65.8**	**78.0**	**84.9**

#N : Nagative Sample 개수 , IB : In – Batch 적용여부

< end-to-end QA accuracy >

Training	Model	NQ	TriviaQA	WQ	TREC	SQuAD
Single	BM25+BERT (Lee et al., 2019)	26.5	47.1	17.7	21.3	33.2
Single	ORQA (Lee et al., 2019)	33.3	45.0	36.4	30.1	20.2
Single	HardEM (Min et al., 2019a)	28.1	50.9	-	-	-
Single	GraphRetriever (Min et al., 2019b)	34.5	56.0	36.4	-	-
Single	PathRetriever (Asai et al., 2020)	32.6	-	-	-	**56.5**
Single	REALM$_{Wiki}$ (Guu et al., 2020)	39.2	-	40.2	46.8	-
Single	REALM$_{News}$ (Guu et al., 2020)	40.4	-	40.7	42.9	-
Single	BM25	32.6	52.4	29.9	24.9	38.1
	DPR	**41.5**	56.8	34.6	25.9	29.8
	BM25+DPR	39.0	57.0	35.2	28.0	36.7
Multi	DPR	**41.5**	56.8	**42.4**	49.4	24.1
	BM25+DPR	38.8	**57.9**	41.1	**50.6**	35.8

프로젝트 구조도


주택 청약으로 청약 가능한 주택은?
검색 인터페이스
유사도 엔진
대상 문서 지정
1. 청약 자격
1.1 공통 자격
1.2. 자격순위
1.2.1. 1순위
1.2.1. 2순위
검색엔진
검색기
색인 파일
색인기
모델 학습 모듈
raw data
학습 데이터
DPR 모델 생성 모듈
질문 1-1
답변 1-1
질문1
질문 1-2
답변 1-2
질문 1-3
답변 1-3
DPR
학습 모델
지식베이스
참조
수집기
docx, hwp,
pptx, xlsx, etc.
수집기
(Crawler)
JSON
데이터 수집
문서 변환

사용한 tools

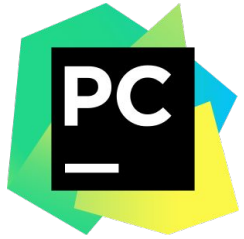

python


PyTorch


Flask
web development,
one drop at a time


Java


Elastic Search


PC


eclipse


GitLab

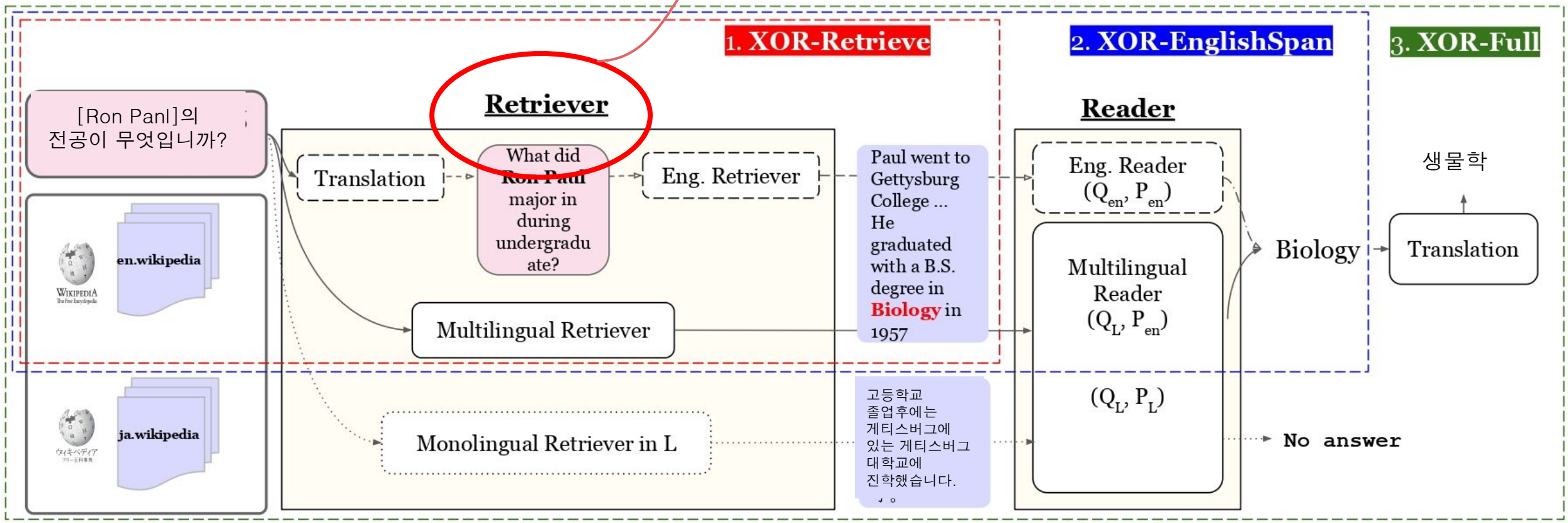

slack

3

Pretraining Model

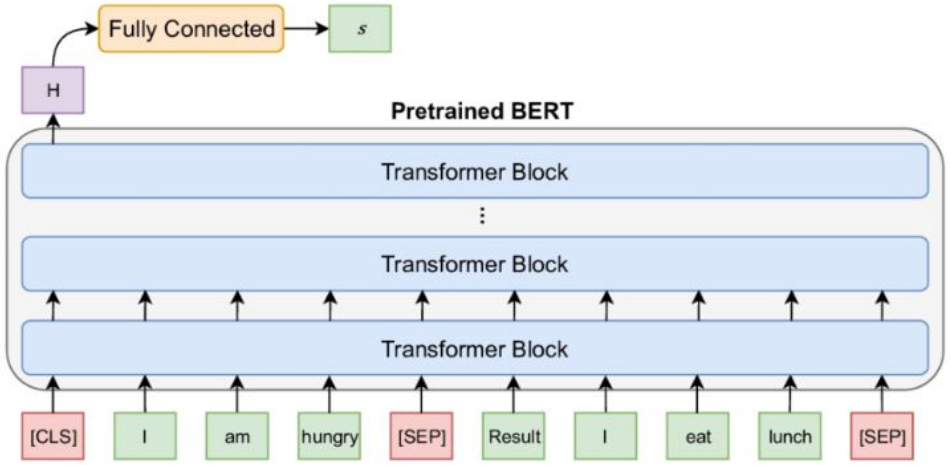
DPR 구조

Bert 모델 사용 → 한국어 처리를 위해 Kobert 로 변경



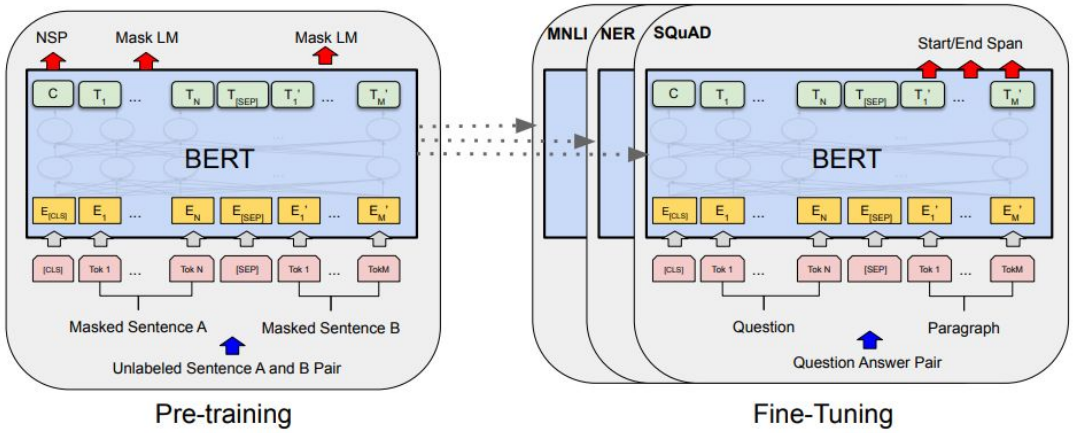
BERT , KoBERT Model

< BERT Model >



- ➔ 약 33억개의 단어로 pretrain 되어있는 기계번역 모델
- ➔ 사용목적에 따라 fine-tuning 이 가능
- ➔ 텍스트를 양방향 (앞뒤) 로 확인하여 자연어 처리
- ➔ **영어의 정확도는 높지만, 한국어의 정확도는 낮음**

< KoBERT Model >



- ➔ BERT Model 에서 **“한국어 데이터”** 를 추가로 학습시킨 모델 (한국어 위키에서 5백만개의 문장과 5,400만개의 단어를 학습시킨 모델)
- ➔ **한국어 버전 BERT Model**
  (한국어 데이터에서 높은 정확도)

Input data

< input_train.json >


```
[{'question': '바그너는 괴테의 파우스트를 읽고 무엇을 쓰고자 했는가?',
  'answers': ['교향곡'],
  'positive_ctxs': [{'title': '파우스트_서곡',
    'text': '1839년 바그너는 괴테의 파우스트을 처음 읽고 그 내용에 마음이 끌려 이를 소재로 해서 하나의 교향곡을 쓰려는 뜻을 갖는다. 이 시기 바그너는 1838년에 빚 독촉으로 산전수전을 다 겪은 상황이라 좌절과 실망에 가득했으며 메피스토펠레스를 만나는 파우스트의 심경에 공감했다고 한다. 또한 파리에서 아브네크의 지휘로 파리 음악원 관현악단이 연주하는 베토벤의 교향곡 9번을 듣고 깊은 감명을 받았는데, 이것이 이듬해 1월에 파우스트의 서곡으로 쓰여진 이 작품에 조금이라도 영향을 끼쳤으리라는 것은 의심할 여지가 없다. 여기의 라단조 조성의 경우에도 그의 전기에 적혀 있는 것처럼 단순한 정신적 피로나 실의가 반영된 것이 아니라 베토벤의 합창교향곡 조성의 영향을 받은 것을 볼 수 있다. 그렇게 교향곡 작곡을 1839년부터 40년에 걸쳐 파리에서 착수했으나 1악장을 쓴 뒤에 중단했다. 또한 작품의 완성과 동시에 그는 이 서곡(1악장)을 파리 음악원의 연주회에서 연주할 파트보까지 준비하였으나, 실제로는 이루어지지는 않았다. 결국 초연은 4년 반이 지난 후에 드레스덴에서 연주되었고 재연도 이루어졌지만, 이후에 그대로 방치되고 말았다. 그 사이에 그는 리엔치와 방황하는 네덜란드인을 완성하고 탄호이저에도 착수하는 등 분주한 시간을 보냈는데, 그런 바쁜 생활이 이 곡을 잊게 한 것이 아닌가 하는 의견도 있다.'}]},
. . .
```


< input_dev.json >


```
[{'question': '임종석이 여의도 농민 폭력 시위를 주도한 혐의로 지명수배 된 날은?',
  'answers': ['1989년 2월 15일'],
  'positive_ctxs': [{'title': '임종석',
    'text': '1989년 2월 15일 여의도 농민 폭력 시위를 주도한 혐의(폭력행위등처벌에관한법률위반)으로 지명수배되었다. 1989년 3월 12일 서울지방검찰청 공안부는 임종석의 사전구속영장을 발부받았다. 같은 해 6월 30일 평양축전에 임수경을 대표로 파견하여 국가보안법위반 혐의가 추가되었다. 경찰은 12월 18일~20일 사이 서울 경희대학교에서 임종석이 성명 발표를 추진하고 있다는 첩보를 입수했고, 12월 18일 오전 7시 40분 경 가스총과 전자봉으로 무장한 특공조 및 대공과 직원 12명 등 22명의 사복 경찰을 승용차 8대에 나누어 경희대학교에 투입했다. 1989년 12월 18일 오전 8시 15분 경 서울청량리경찰서는 호위 학생 5명과 함께 경희대학교 학생회관 건물 계단을 내려오는 임종석을 발견, 검거해 구속을 집행했다. 임종석은 청량리경찰서에서 약 1시간 동안 조사를 받은 뒤 오전 9시 50분 경 서울 장안동의 서울지방경찰청 공안분실로 인계되었다.'}]},
{'question': '1989년 6월 30일 평양축전에 대표로 파견 된 인물은?',
  'answers': ['임수경'],
  'positive_ctxs': [{'title': '임종석',
    'text': '1989년 2월 15일 여의도 농민 폭력 시위를 주도한 혐의(폭력행위등처벌에관한법률위반)으로 지명수배되었다. 1989년 3월 12일 서울지방검찰청 공안부는 임종석의 사전구속영장을 발부받았다.
. . .
```

KoBERT code

1. RESOURCE_MAP에 한국어 데이터 추가

```
"data.retriever.input-dev": {
    "s3_url": "https://dl.fbaipublicfiles.com/dpr/data/retriever/biencoder-squad1-train.json.gz",
    "original_ext": ".json",
    "compressed": False,
    "desc": "SQUAD 1.1 train subset with passages pools for the Retriever training",
},
"data.retriever.input-train": {
    "s3_url": "https://dl.fbaipublicfiles.com/dpr/data/retriever/biencoder-squad1-train.json.gz",
    "original_ext": ".json",
    "compressed": False,
    "desc": "SQUAD 1.1 train subset with passages pools for the Retriever training",
},
```

2. encoder train default.yaml과 retriever default.yaml 수정

```
input_train:
  _target_: dpr.data.biencoder_data.JsonQADataset
  file: data.retriever.input-train

input_dev:
  _target_: dpr.data.biencoder_data.JsonQADataset
  file: data.retriever.input-dev
```

3. hf_bert.yaml에서 하이퍼 파라미터 수정

```
# @package _group_

# model type. One of [hf_bert, pytext_bert, fairseq_roberta]
encoder_model_type: hf_bert

# HuggingFace's config name for model initialization
pretrained_model_cfg: skt/kobert-base-v1

# Some encoders need to be initialized from a file
pretrained_file:

# Extra linear layer on top of standard bert/roberta encoder
projection_dim: 0

# Max length of the encoder input sequence
sequence_length: 512

dropout: 0.1

# whether to fix (don't update) context encoder during training or not
fix_ctx_encoder: False

# if False, the model won't load pre-trained BERT weights
pretrained: True
```

KoBERT code

4. hf_model.py의 kobert 변환

```
def get_bert_tokenizer(pretrained_cfg_name: str, do_lower_case: bool = True):
    return KoBERTTokenizer.from_pretrained(pretrained_cfg_name,
do_lower_case=do_lower_case)


class BertTensorizer(Tensorizer):
    def __init__(self, tokenizer: KoBERTTokenizer, max_length: int, pad_to_max: bool =
True):
        self.tokenizer = tokenizer
        self.max_length = max_length
        self.pad_to_max = pad_to_max


if transformers.__version__.startswith("4"):
    from transformers import BertConfig, BertModel
    from transformers import AdamW
    from kobert_tokenizer import KoBERTTokenizer
    from transformers import RobertaTokenizer
```

5. hf_model.py의 truncation 전략 수정

```
class BertTensorizer(Tensorizer):
    def __init__(self, tokenizer: KoBERTTokenizer, max_length: int, pad_to_max:
bool = True):
        self.tokenizer = tokenizer
        self.max_length = max_length
        self.pad_to_max = pad_to_max

        if title:
            token_ids = self.tokenizer.encode(
                title,
                text_pair=text,
                add_special_tokens=add_special_tokens,
                max_length=self.max_length if apply_max_len else 10000,
                pad_to_max_length=False,
                truncation="only_second",
            )
        else:
            token_ids = self.tokenizer.encode(
                text,
                add_special_tokens=add_special_tokens,
                max_length=self.max_length if apply_max_len else 10000,
                pad_to_max_length=False,
                truncation="only_second",
            )
```

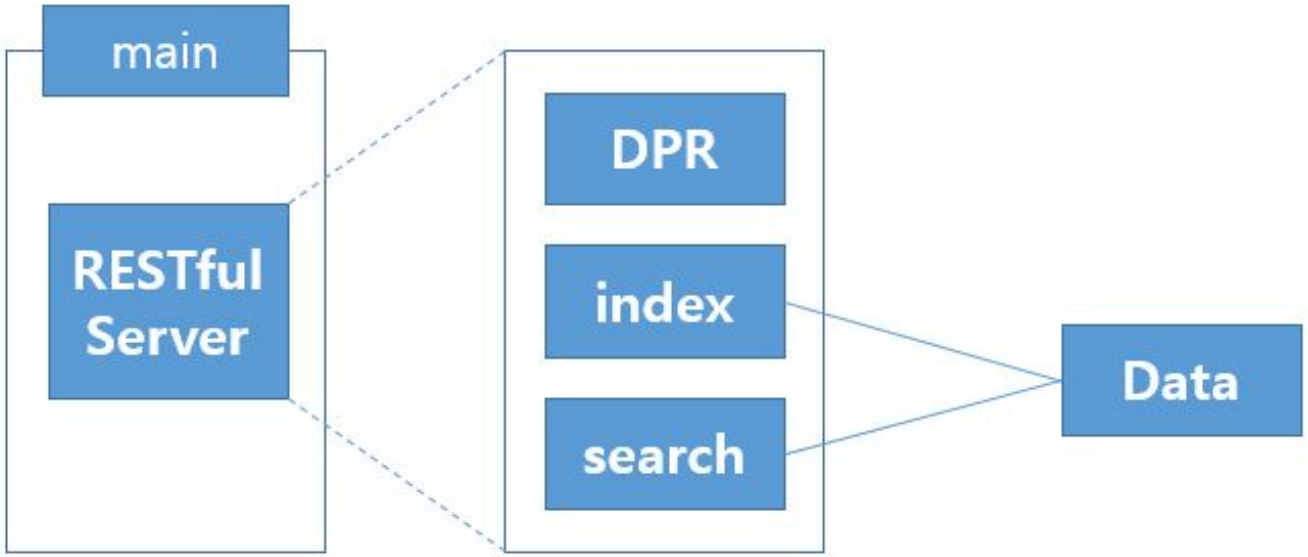
4

유사도 엔진

주요 기능

4. 유사도 엔진


Flask
web development,
one drop at a time


main
RESTful Server
DPR
index
search
Data

DPR code : similarity.py

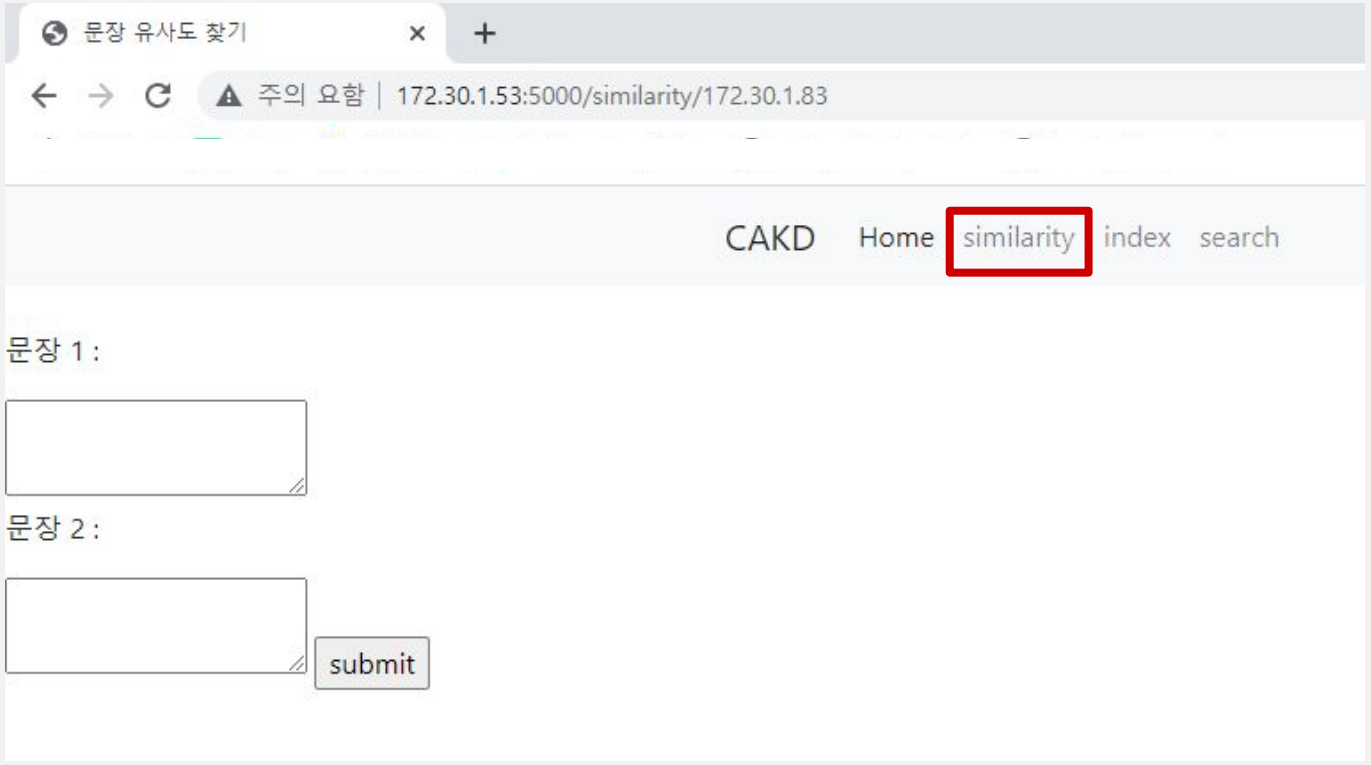
```
def get_idx(*args):
    idxs = []
    for _ in args:
        idxs.append(tokenizer(_,return_tensors="pt")["input_ids"])
    return idxs

def get_pooleroutput(List):
    embeddings = []
    for _ in List:
        embeddings.append(model(_).pooler_output)
    return embeddings

def dot_product_scores(q_vectors: T, ctx_vectors: T) -> T:
    """
    calculates q->ctx scores for every row in ctx_vector
    :param q_vector:
    :param ctx_vector:
    :return:
    """
    # q_vector: n1 x D, ctx_vectors: n2 x D, result n1 x n2
    r = torch.matmul(q_vectors, torch.transpose(ctx_vectors, 0, 1))
    return r
```

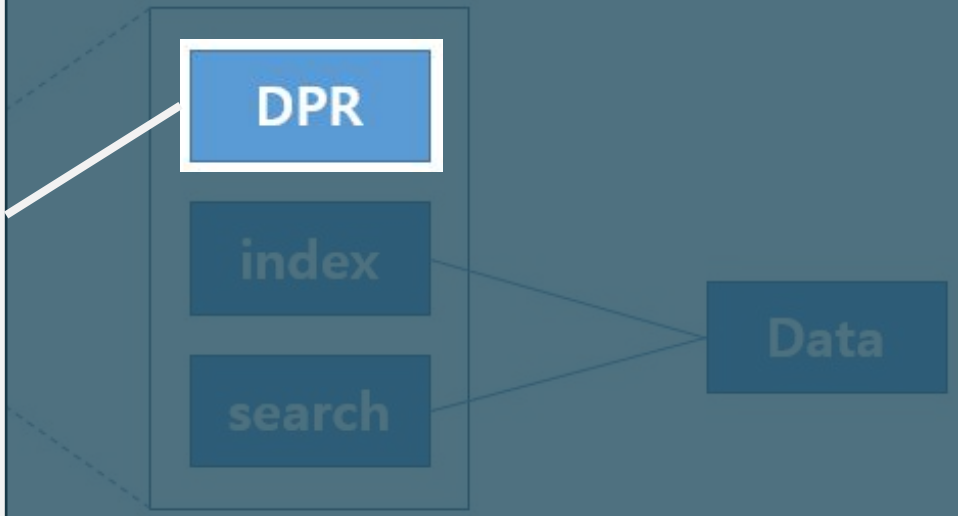
```
def cosine_scores(q_vector: T, ctx_vectors: T):
    # q_vector: n1 x D, ctx_vectors: n2 x D, result n1 x n2
    return F.cosine_similarity(q_vector, ctx_vectors, dim=1)

def get_total_scores(q_vector: T, ctx_vectors: T):
    cos_score=cosine_scores(q_vector, ctx_vectors)
    dot_pdt_score=dot_product_scores(q_vector, ctx_vectors)
    total_score = cos_score
    return round(total_score.item(),4)

def get_title_dpr(item):
    _ = tokenizer(item, return_tensors="pt")["input_ids"]
    input_data = _.to(device)
    res = model(input_data).pooler_output.detach().numpy().tolist()
    return res

def get_content_dpr(item):
    _ = tokenizer(item,
return_tensors="pt",truncation=True,max_length=512)["input_ids"]
    input_data = _.to(device)
    res = model(input_data).pooler_output.detach().numpy().tolist()
    return res
```

주요 기능

4. 유사도 엔진

1. DPR : 두 문장 간의 유사도 파악





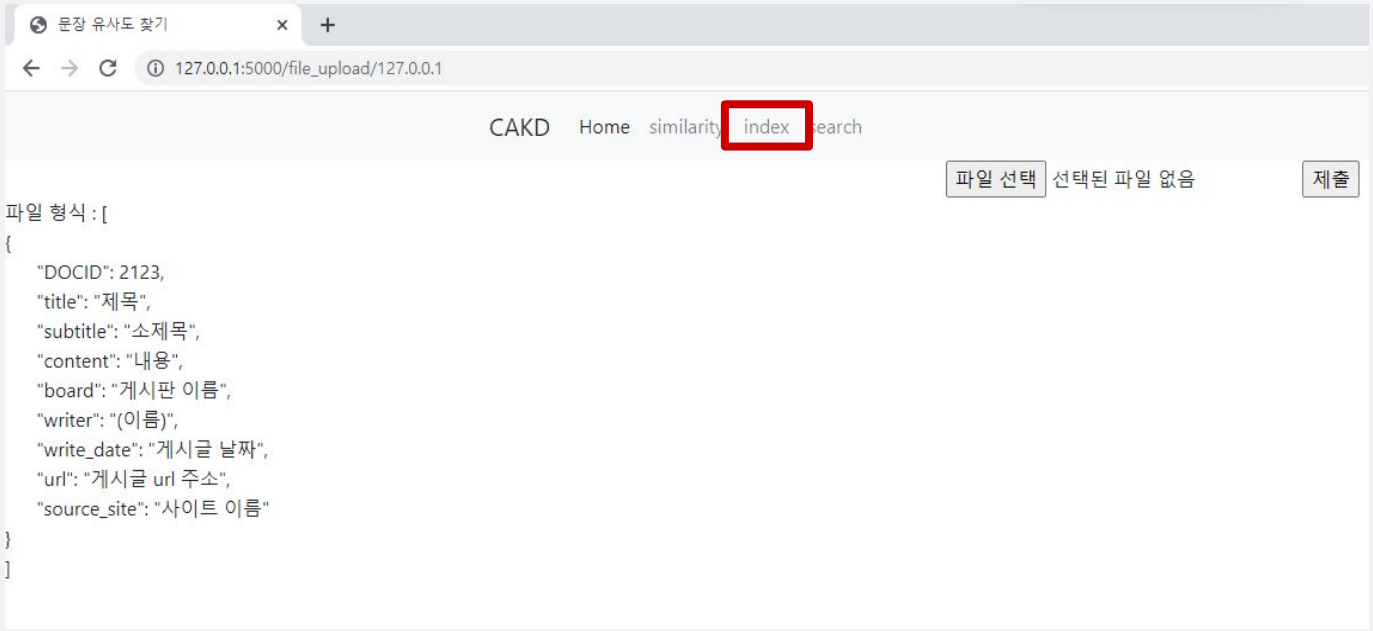
Index code : index.py

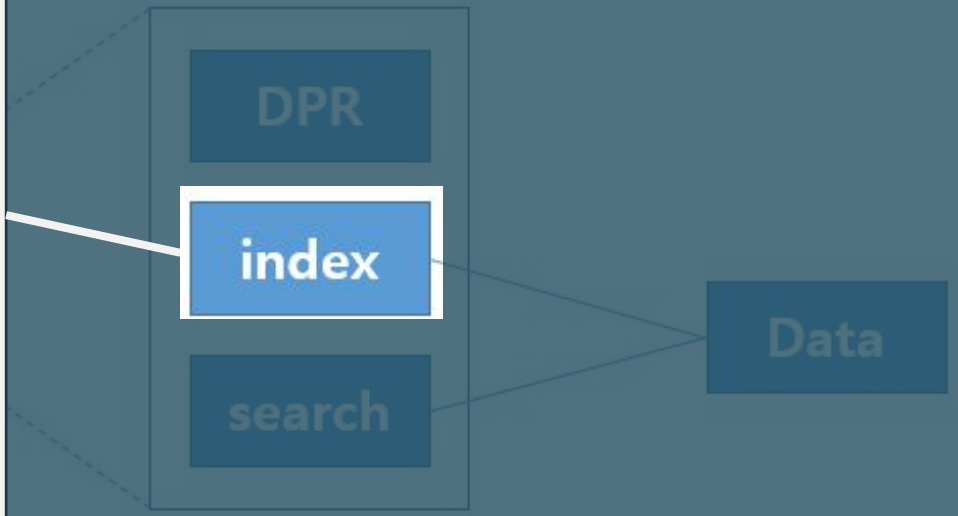
```
mecab = Komoran()
def file_save(f,fname,upload_folder):
    try :
        file_path = os.path.join(upload_folder, fname)
        print(file_path)
        f.save(file_path)
        if fname in os.listdir(upload_folder):
            print('success')
            return (True, file_path)
        else :
            print('failed')
            return (False, file_path)

    except Exception as e:
        print('-------------------------------------')
        print(e)
        return (False, file_path)
            new_dict['title'] = item['title']
            new_dict['title_dpr'] = title_dpr


return_tensors="pt",truncation=True,max_length=512)["input_ids"]).pooler_out
put.detach().numpy().tolist()
            content_dpr = get_content_dpr(item['content'])
            new_dict['content'] = item['content']
            new_dict['content_dpr'] = content_dpr
            new_dict['content_morphs'] = get_morphs(item['content'])
            new_dict['title_morphs'] = get_morphs(item['title'])
            new_dict['DOCID'] = item['DOCID']
            es.index(index='my_index', body=new_dict,id = new_dict['DOCID'])
            end = time.time()
            count += 1
            print(count,':',end-start)

        #bulk(es,json_file,index = 'my_index')
    return True
```

```
def file_upload_in_db(db, document_obj):
    try:
        db.session.add(document_obj)
        db.session.commit()
        return True
    except:
        return False
def get_morphs(item):
    res = mecab.morphs(item.replace('[^가-힣a-zA-Z0-9]',''))
    return res

def file_indexing(path):
    with open(path,'r',encoding='utf-8') as f:
        json_file=json.load(f)
        count = 0
        for item in json_file:
            start = time.time()
            new_dict = {}
            #title_dpr = model(tokenizer(item['title'],
return_tensors="pt")["input_ids"]).pooler_output.detach().numpy().tolist()
            title_dpr = get_title_dpr(item['title'])
            content_dpr = get_content_dpr(item['content'])
            new_dict['content'] = item['content']
            new_dict['content_dpr'] = content_dpr
            new_dict['content_morphs'] = get_morphs(item['content'])
            new_dict['title_morphs'] = get_morphs(item['title'])
            new_dict['DOCID'] = item['DOCID']
            es.index(index='my_index', body=new_dict,id = new_dict['DOCID'])
            end = time.time()
            count += 1
            print(count,':',end-start)


        #bulk(es,json_file,index = 'my_index')
    return True
```

주요 기능

2. Index : 파일 인덱싱

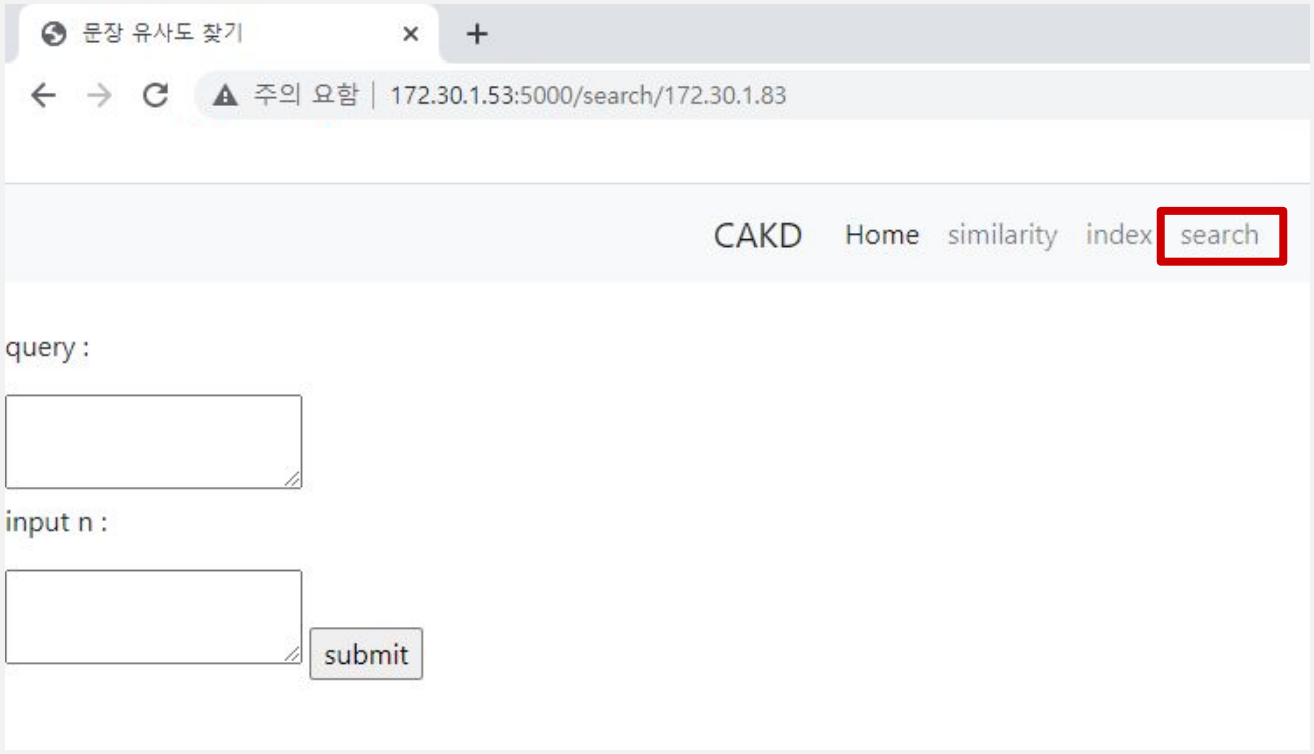


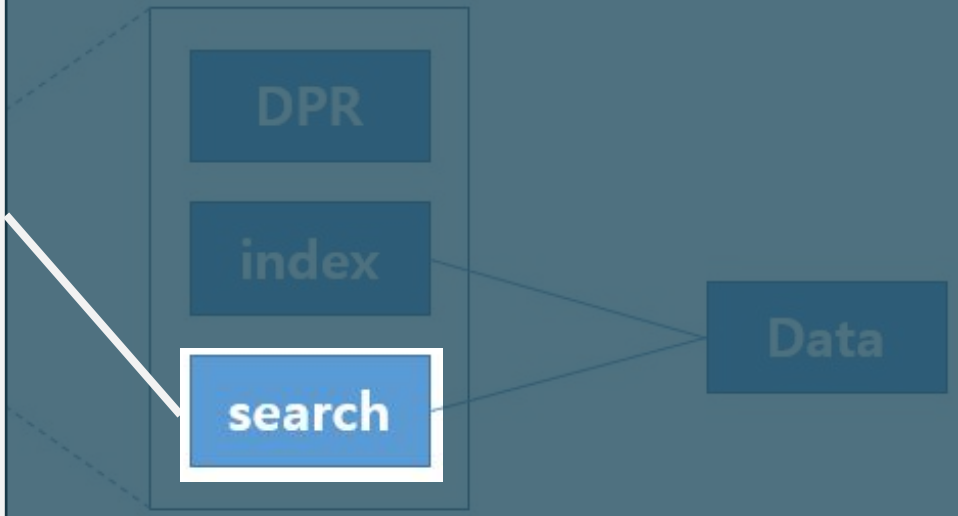


Search code : data_util.py

```
def get_similarity_in_document(input_text,input_nbest,json_objs):
    for json_obj in json_objs:
        indicies = get_idx(input_text, json_obj['text'])
        embeddings = get_pooleroutput(indicies)
        json_obj['similarity'] = get_total_scores(embeddings[0], embeddings[1])
    json_objs.sort(key=lambda x : -x['similarity'])
    for i,json_obj in enumerate(json_objs):
        json_obj['nbest'] = i+1
        if (i+1) == input_nbest: break;
    return json_objs
def search_in_es(q,n):
    que=q.replace('[^A-Za-z0-9가-힣]', '')
    query_with_tag=mecab.pos(que)
    print(query_with_tag)
    print(mecab.tagset)
    query = []
    for item in query_with_tag:
        if (item[1] == 'XR') | (item[1] == 'NNG') | (item[1] == 'VV') | (item[1] ==
'NNB') | (item[1] == 'NNP') | (item[1] == 'VA') | (item[1] == 'NP')  | (item[1] ==
'SN')| (item[1] == 'NA')| (item[1] == 'NR')| (item[1] == 'SL'):
            query.append(item[0])
    print(mecab.pos(que)[0][0])
    print(query)
    data = []  # 반환할 데이터를 담을 리스트를 초기화합니다
    for item in query:
        query = {
            "query": {
                "multi_match": {
                    "query": item,
                    "fields": ["title", "content","title_morphs","content_morphs"]
            }
        }
        }
        res =es.search(index = 'my_index',body = query)
        hits = res.get('hits', {}).get('hits', [])  # 실제 데이터가 있는 hits 리스트를
추출합니다
        # 각각의 문서에 대해서 필요한 정보만 추출하여 data 리스트에 추가합니다
        for hit in hits:
            source = hit.get('_source', {})
            item = {
                'title': source.get('title', ''),
                'content': source.get('content', ''),
                'title_dpr': source.get('title_dpr', []),
                'content_dpr': source.get('content_dpr', []),}
            if item not in data:
                data.append(item)
    for item in data:
        question=model(tokenizer(q,
return_tensors="pt",truncation=True,max_length=512)["input_ids"]).pooler_output
        ctx=torch.tensor(item['content_dpr'])
        title=torch.tensor(item['title_dpr'])
        score1=get_total_scores(question,ctx)
        score2=get_total_scores(question,title)
        item.setdefault('score',score1+score2)
    sorted_list = sorted(data, key=lambda x: x['score'], reverse=True)
    print(len(sorted_list))
    return sorted_list[:n]
```

3. Search : 유사도 top 문장 출력





5

검색 엔진 연동

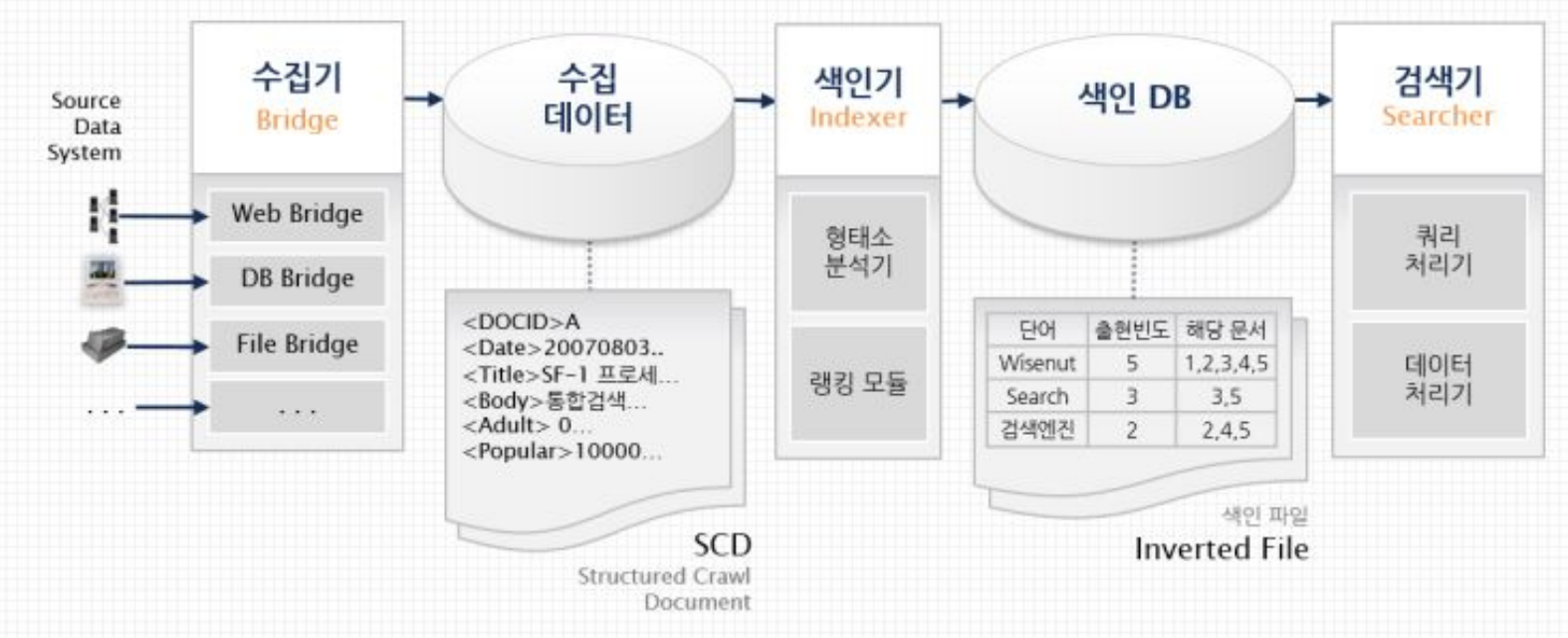
‘색인’

- 색인 (Indexing)

  ➔ 데이터베이스나 검색 엔진에서 검색 속도를 높이기 위해 사용되는 기술

  ➔ 특정 칼럼에 대한 정렬된 데이터 구조를 생성하여 검색 속도를 높임

- 색인 과정



'색인어 및 검색어 분석'

- **검색의 기본**

  ➔ "문서" 혹은 "검색어" 를 어떠한 방식으로 주요 단어를 추출할지 결정

  ➔ '어떻게 추출하는가' 에 따라 검색의 품질 및 정확성이 보장

- **색인어 및 검색어 분석 과정**

문서 → 레귤레이션 → 색인어 추출 → 색인 DB ⬅ 검색어 추출 ⬅ 레귤레이션 ⬅ 검색어

- **레귤레이션 (Regulation) - 구분자 및 병합자 지정**

  ➔ 특정 문자를 기준으로 문자열을 나누거나 병합하는 과정

  ➔ 분석 대상 문자열을 '토큰 (token)' 으로 분리하고, 해당 토큰을 가지고 색인어 및 검색어 추출기의 입력으로 사용

  ➔ 별도의 설정이 없을 때는, "기본 구분자 (대부분의 특수문자)" 를 이용하여 토큰 분리

'색인어 및 검색어 분석'

- **검색의 기본**

  ➔ "문서" 혹은 "검색어" 를 어떠한 방식으로 주요 단어를 추출할지 결정

  ➔ '어떻게 추출하는가' 에 따라 검색의 품질 및 정확성이 보장

- **색인어 및 검색어 분석 과정**

문서 → 레귤레이션 → 색인어 추출 → 색인 DB ⬅ 검색어 추출 ⬅ 레귤레이션 ⬅ 검색어

- **레귤레이션 (Regulation) - 구분자 및 병합자 지정**

예시) "안녕하세요? #중앙정보! 홍길동+입니다." → [안녕하세요], [중앙정보], [홍길동], [입니다]

‘역색인’

- 역색인 (Inverted index)

  ➔ 매우 빠른 풀텍스트 검색을 할 수 있도록 설계된 것

  ➔ 문서에 나타나는 모든 고유한 단어의 목록을 만들고, 각 단어가 발생하는 모든 문서를 식별

  ➔ java는 컴파일 언어이며, python보다 좀 더 빠르기 때문에 java를 사용

- 역색인 코드

```
.io.*;
.util.*;
.util.Map.Entry;
.nio.file.*;


s aufgabe1 {
ic static void main(String[] args) throws IOException {

ystem.out.println("loading "+args[0]+" file");
ong start = System.currentTimeMillis();
/버퍼사이즈 파일크기에 영향을 받음 파일크기는 약 35000byte이고 전체를 받으면 빠를것으로 생각됨
BufferedReader reader = new BufferedReader(
        new FileReader("C:\\Users\\admin\\Downloads\\"+args[0]),40000
    );
Path path = Paths.get("C:\\Users\\admin\\Downloads\\"+args[0]);
//문장의 개수를 미리 알면 속도향상에 도움이 될것으로 생각해서 문장의 갯수 확인
int lineCount = (int) Files.lines(path).count();
String[] saetze = new String[lineCount];
final int [] count = {0};
//읽은 문장들을 배열에 저장
Files.lines(path).forEach(line -> saetze[count[0]++] = line);
reader.close();

long end = System.currentTimeMillis();
double time = (end - start)/1000.0;
System.out.println(String.format("Complete! (%.3fs)", time));

//멀티쓰레딩 사용하면 기존보다 더 빨라질것으로 생각됨
//containskey 보다 get함수가 빠르지만 큰 차이 없음
long start1 = System.currentTimeMillis();


//해쉬맵을 2중으로 사용
HashMap<String, HashMap<Integer,Integer>> doc = new HashMap<String, HashMap<Integer,Integer>>();

//문장들을 병렬처리
Iterator<String> iterator = Arrays.stream(saetze).parallel().iterator();
while (iterator.hasNext()) {
    String satz = iterator.next();
    String[] temp = satz.toLowerCase().replaceAll("[^a-z0-9]", " ").split(" ");
    temp = Arrays.stream(temp).filter(s -> !s.isEmpty()).toArray(String[]::new);
    String satz = iterator.next();
    String[] temp = satz.toLowerCase().replaceAll("[^a-z0-9]", " ").split(" ");
    temp = Arrays.stream(temp).filter(s -> !s.isEmpty()).toArray(String[]::new);
    int doc_id = Integer.parseInt(temp[0]);
    for (int i = 1; i < temp.length; i++) {
        String word = temp[i];


            if(!doc.containsKey(word)) {

                doc.put(word, new HashMap<Integer,Integer>());

            }

            HashMap<Integer,Integer> innerMap = doc.get(word);
            innerMap.put(doc_id, innerMap.getOrDefault(doc_id, 0) + 1);




    }

}
Comparator<HashMap.Entry<Integer, Integer>> valueComparator = new Comparator<HashMap.Entry<Integer, Integer>>() {
    @Override
    public int compare(Entry<Integer, Integer> o1, Entry<Integer, Integer> o2) {
        int compare = o2.getValue().compareTo(o1.getValue()); // comparing values in descending order
        if (compare == 0) {
            return o1.getKey().compareTo(o2.getKey()); // comparing keys in ascending order
        }
        return compare;
    }
            };

            HashMap<String, LinkedHashMap<Integer,Integer>> sortedDoc = new HashMap<>();
            Set<String> keySet = doc.keySet();
            List<String> keyList = new ArrayList<>(keySet);
            Collections.sort(keyList);
            for (String key : keyList) {
                HashMap<Integer,Integer> innerMap = doc.get(key);
                List<Map.Entry<Integer, Integer>> list = new ArrayList<>(innerMap.entrySet());
                // comparator로 정렬
                Collections.sort(list,valueComparator);
                // comparator로 정렬된 결과를 새로운 해쉬맵에 저장
                LinkedHashMap<Integer,Integer> innerMapSorted = new LinkedHashMap<>();
                for(HashMap.Entry<Integer, Integer> innerEntry : list) {
                    innerMapSorted.put(innerEntry.getKey(), innerEntry.getValue());
                }
                sortedDoc.put(key, innerMapSorted);
            }
            /*
            Set<String> li = sortedDoc.keySet();
            for (String w : li) {
                LinkedHashMap<Integer, Integer> inn = sortedDoc.get(w);
                Set<Integer> inner = inn.keySet();
                for (Integer docId : inner) {
                    System.out.println(w+" "+ docId + " " + inn.get(docId) + " ");
                }
            }
            */


            BufferedWriter bw = null;
            try {
                bw = new BufferedWriter(new FileWriter("sortedDoc.txt"),1024);
                List<String> sortedDocKeySet = keyList;

                for (String word : sortedDocKeySet) {
                    LinkedHashMap<Integer, Integer> innerMap = sortedDoc.get(word);
                    Set<Integer> innerMapKeySet = innerMap.keySet();
                    for (Integer docId : innerMapKeySet) {
                        bw.write(word+" "+ docId + " " + innerMap.get(docId) + " ");
                    }
                    bw.newLine();
                    bw.flush();
                }
            } catch (IOException e) {
                e.printStackTrace();
            } finally {
                try {
                    if (bw != null)
                        bw.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
```

‘역색인’

- 역색인 (Inverted index)
  - ➔ 매우 빠른 풀텍스트 검
  - ➔ 문서에 나타나는 모든
  - ➔ java는 컴파일 언어이기

- 역색인 코드

< 역색인 결과 >

```
loading input.big file
Complete! (0,176s)
Complete! (5,762s)
total Complete! (5,938s)
input : frowning
Doc: 11396, Freuquency: 1
Doc: 82860, Freuquency: 1
Doc: 104083, Freuquency: 1
Doc: 249603, Freuquency: 1

input : happy
Doc: 64017, Freuquency: 3
Doc: 8929, Freuquency: 2
Doc: 68177, Freuquency: 2
Doc: 71409, Freuquency: 2
Doc: 122598, Freuquency: 2
Doc: 125899, Freuquency: 2
Doc: 143714, Freuquency: 2
Doc: 236782, Freuquency: 2
Doc: 262076, Freuquency: 2
Doc: 911, Freuquency: 1
Doc: 1214, Freuquency: 1
Doc: 1428, Freuquency: 1
Doc: 4452, Freuquency: 1
Doc: 6080, Freuquency: 1
Doc: 6859, Freuquency: 1
Doc: 7005, Freuquency: 1
Doc: 7902, Freuquency: 1
Doc: 8201, Freuquency: 1
Doc: 8333, Freuquency: 1
Doc: 9214, Freuquency: 1
Doc: 9686, Freuquency: 1
Doc: 11735, Freuquency: 1
Doc: 12324, Freuquency: 1
Doc: 14368, Freuquency: 1
Doc: 14374, Freuquency: 1
Doc: 14519, Freuquency: 1
Doc: 15320, Freuquency: 1
Doc: 16165, Freuquency: 1
Doc: 16361, Freuquency: 1
```

Elastic search 란?

- **텍스트, 숫자, 위치 기반 정보, 정형 및 비정형 데이터 등 모든 유형의 데이터를 위한 무료 검색 및 분석 엔진**

실시간 검색 플랫폼

문서가 색인될 때부터 검색 가능해질 때까지의 대기 시간이 매우 짧음

분산적

Elastic search에 저장된 문서는 '**샤드**' (여러 다른 컨테이너) 에 걸쳐 분산되며,

샤드는 복제되어 하드웨어 장애 시에 중복되는 데이터 사본을 제공함

강력한 기본기능

속도, 확장성, 복원력 뿐만 아니라 '데이터 롤업, 인덱스 수명 주기 관리' 등과 같이 데이터를 더욱 효율적으로 저장 및 검색할 수 있는 강력한 기본기능 탑재

데이터 처리 간소화

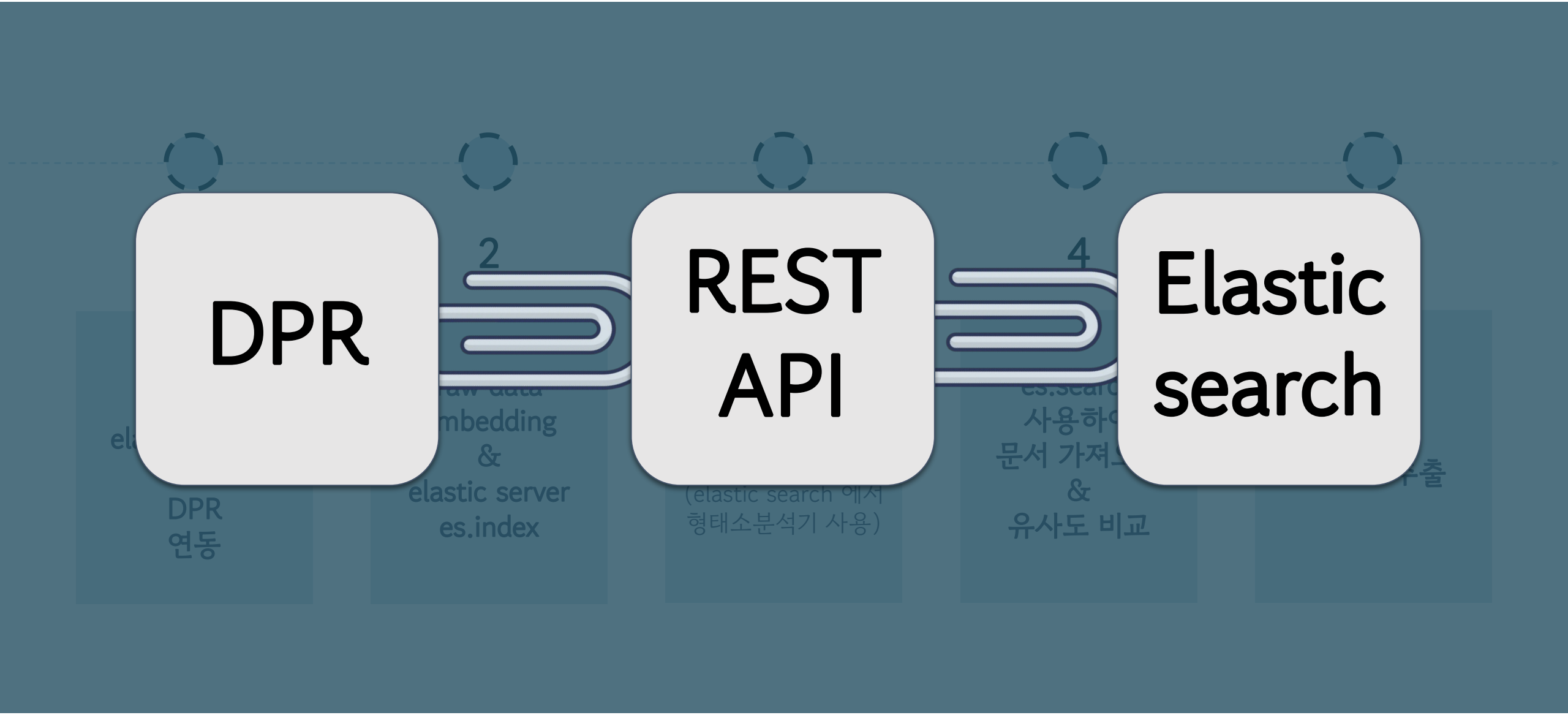
데이터 수집, 시각화, 보고를 간소화

Beats 와 Logstash 의 통합은 Elasic search 로 색인하기 전 데이터를 쉽게 처리할 수 있게 함

Elastic search 연동

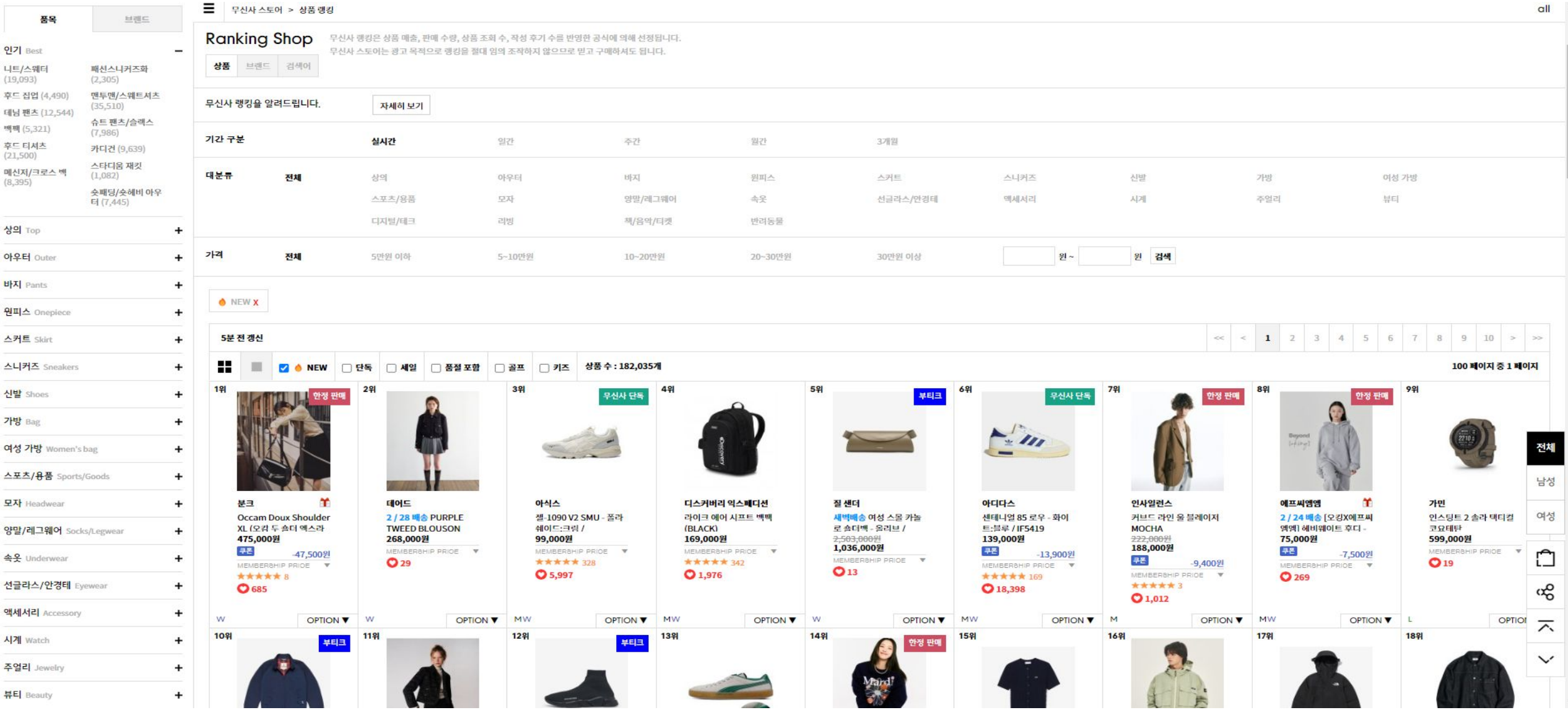
1	2	3	4	5
REST API & elastic search & DPR 연동	raw data embedding & elastic server es.index	question sentence 에 대한 주요 명사 추출 (elastic search 에서 형태소분석기 사용)	es.search 사용하여 문서 가져오기 & 유사도 비교	rank 로 n-best 추출

Elastic search 연동



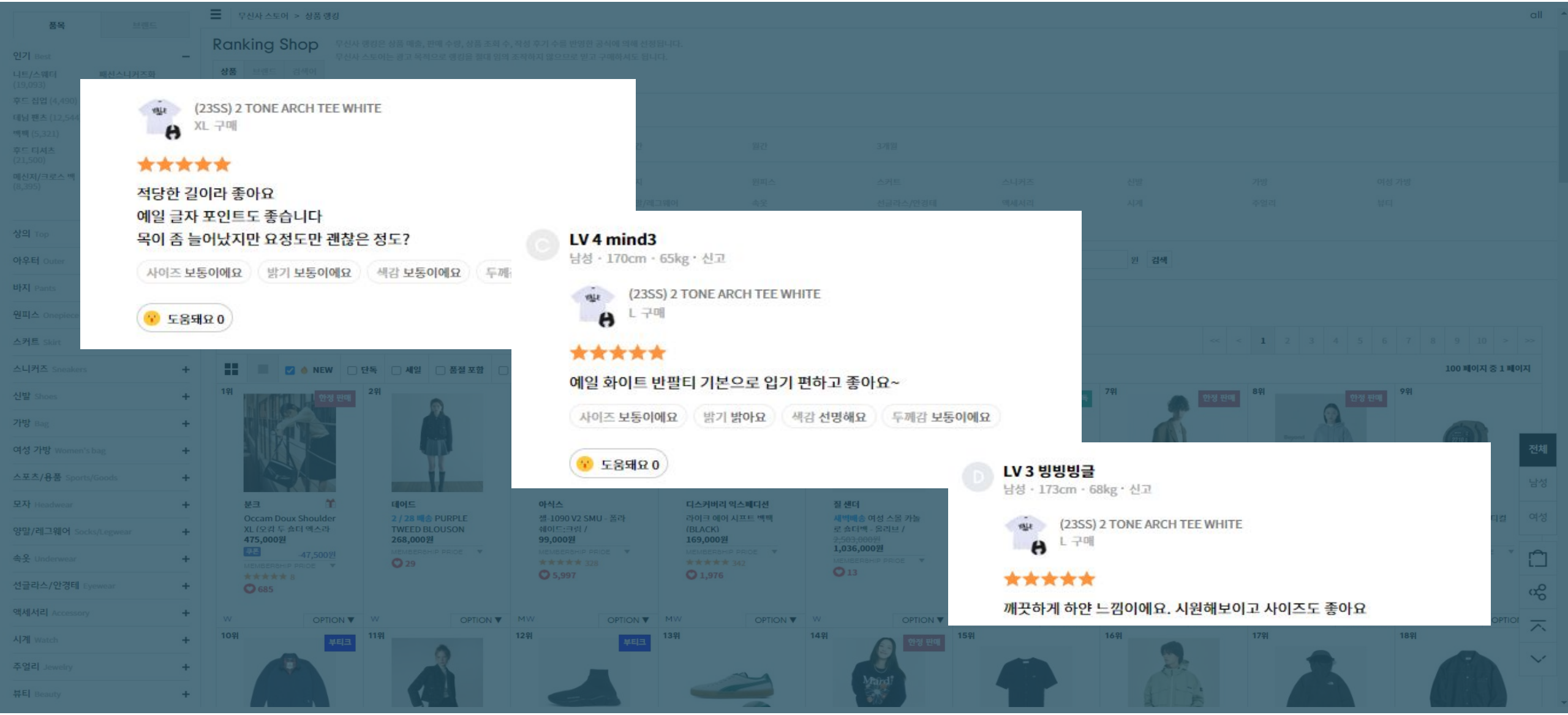
Web Crawling - www.musinsa.com

5. 검색 엔진 연동

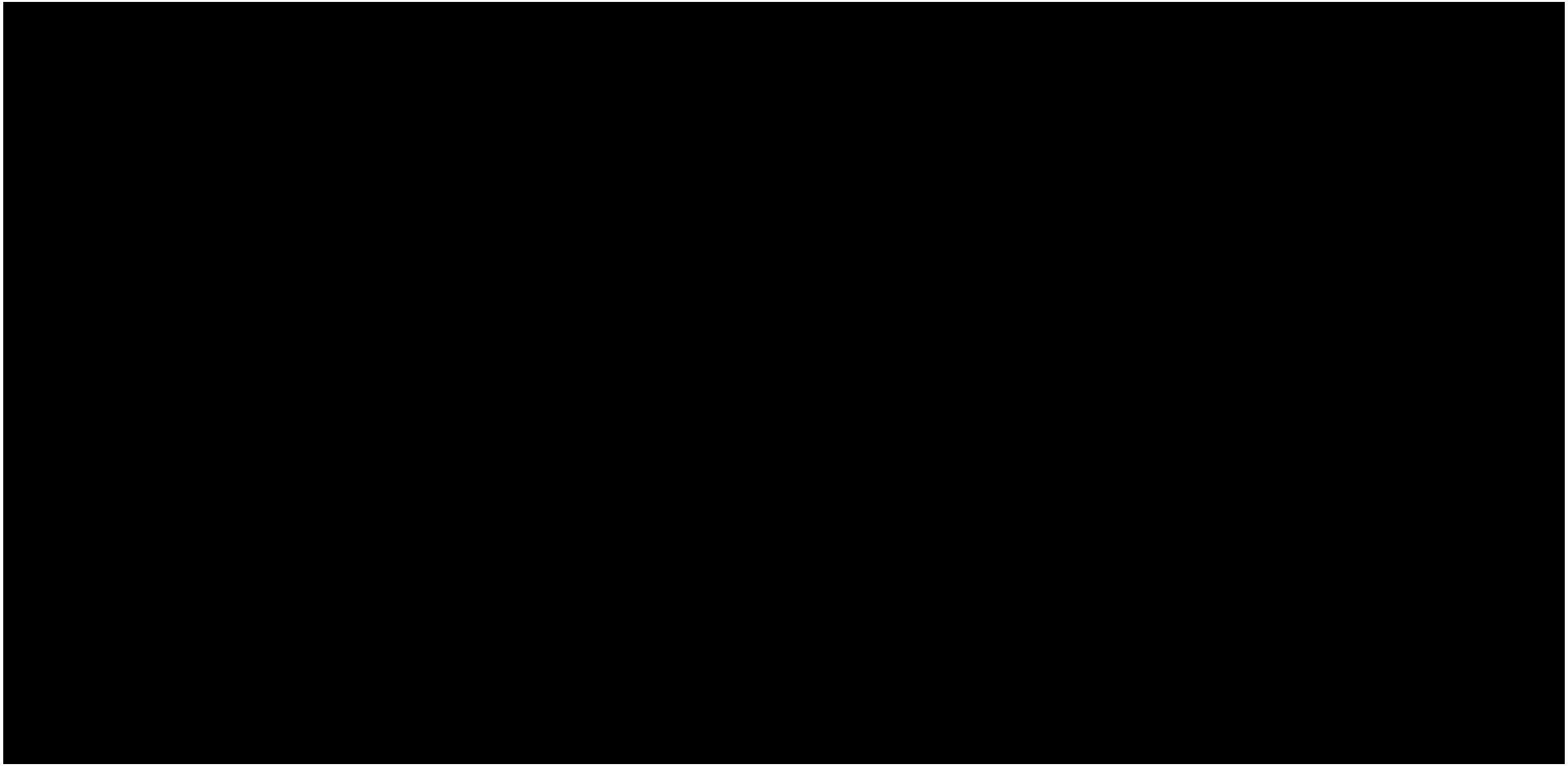


Web Crawling - www.musinsa.com

5. 검색 엔진 연동



Demonstration



6

기대효과 및 한계점

6. 기대효과 및 한계점

기대효과

- 검색 품질의 개선 : dense retrieval 모델을 이용한 품질개선
- 검색 효율성 향상 : 여러개의 문서를 병렬로 처리 가능
- 기존 시스템 개선 : 기존 시스템의 검색 품질 및 효율성 개선
- 새로운 비즈니스 모델 개발 : 검색 엔진 서비스 제공

Conclusion

한계점

- gpu 서버가 없어 모델 훈련에 많은시간이 소모됨
- 한국어 데이터셋 부족
- 인터프리터 언어인 파이썬은 컴파일 언어에 비해 느린 속도
- gpu 사용시에도 문장 임베딩 시 오랜시간 소요

개선점

- 로그인 기능을 구현 및 클라이언트 세션 구현
- 모델 하이퍼 파라미터 튜닝
- 엘라스틱 서치/플라스크 서버 보안설정
- 유사도 함수를 종합해서 사용하여 엔진 성능 개선

Q & A


Thank You