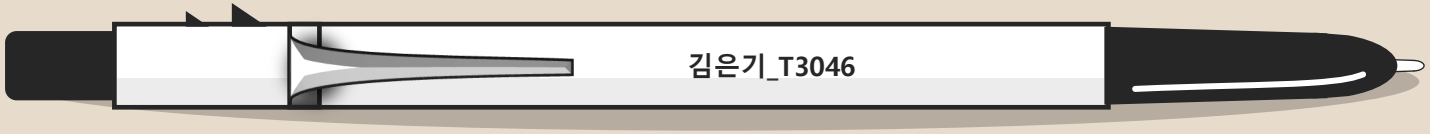
BM25 + DenseRetrieval



목차

1. BM25

BM25를 통한 tfidf code를 대체하고자함

Why?

1. BM25가 tfidf보다 성능이 좋다
    a. 왜 좋아? : tfidf는 여러 문서에서 단어가 중복적으로 등장할 경우 그것의 가중치를 줄여줌 ex) a, the, is 등
    b. BM25는 tfidf를 기반으로 하되, 문서의 길이도 반영을 해준다.
    c. 즉, 긴 문서에서 등장하는 단어일 수록 가중치를 적게 부여하고 짧은 문서일 수록 가중치를 부여
    d. 이렇게 하면, 짧은 문서에서 등장하는 단어가 중요하게 여겨진다.
    e. 즉, TF 부분을 덜 반영하고 IDF 부분을 더 반영하게 된다.
2. Elastic Search에서 사용되기 때문
    a. 하지만 아쉽게 사용하지 못함ㅎㅎ..

1. BM25

import rank_bm25 를 통해 모듈 설치

아래의 코드는 BM25 모듈을 불러와서 만든 것(doc 기반)

크게 어렵지 않다.

```python
class MyBm25(rank_bm25.BM25Okapi):

    def __init__(self, corpus, tokenizer=None, k1=1.5, b=0.75, epsilon=0.25):
        # 논문에 따르면 k1은 1.2~1.5가 적당하다. 또한 b의 경우에도 0.75~0.9가 적당하다.
        # GridSearchCV를 통해서 최적의 값을 찾아보고 정의하는 것이 좋을 듯 하다.
            super().__init__(corpus, tokenizer=tokenizer, k1=k1, b=b, epsilon=epsilon)

    def get_top_n(self, query, documents, n=5):
        assert self.corpus_size == len(documents), "The documents given don't match the index corpus!"

        scores = self.get_scores(query)
        # 이미 구현되어 있는 함수를 사용하여 점수를 구한다.

        top_n_idx = np.argsort(scores)[::-1][:n]
        doc_score = scores[top_n_idx]

        return doc_score, top_n_idx
```

1. BM25

BM25를 위한 설정들을 SpraseRetrieval에 추가

*** epsilon = Constant used for negative idf of document in corpus.**

$$\text{score}(D,Q) = \sum_{i=1}^{n} \text{IDF}(q_i) \cdot \frac{f(q_i, D) \cdot (k_1 + 1)}{f(q_i, D) + k_1 \cdot \left(1 - b + b \cdot \frac{|D|}{\text{avgdl}}\right)}$$

```python
class SparseRetrieval:
    def __init__(
        self,
        tokenize_fn,
        data_path: Optional[str] = "../data/",
        context_path: Optional[str] = "wikipedia_documents.json",
        k1=1.5, b=0.75, epsilon=0.25,
        q_encoder = None,
        p_encoder = None,
        is_bm25 = False
    ) -> NoReturn:
```

1. BM25

상단의 코드는 이전과 동일한 코드로 불러오는 것

하단의 코드는 bm25.bin파일을 생성하거나 불러오는 것

위의 코드와 동일하므로 크게 어렵지 않음

```python
if not self.is_bm25: # tfidf를 사용하는 경우
# Pickle을 저장합니다.
    pickle_name = f"sparse_embedding.bin"
    tfidfv_name = f"tfidv.bin"
    emd_path = os.path.join(self.data_path, pickle_name)
    tfidfv_path = os.path.join(self.data_path, tfidfv_name)

    if os.path.isfile(emd_path) and os.path.isfile(tfidfv_path):
        with open(emd_path, "rb") as file:
            self.p_embedding = pickle.load(file)
        with open(tfidfv_path, "rb") as file:
            self.tfidfv = pickle.load(file)
        print("Embedding pickle load.")
    else:
        print("Build passage embedding")
        self.p_embedding = self.tfidfv.fit_transform(self.contexts)
        print(self.p_embedding.shape)
        with open(emd_path, "wb") as file:
            pickle.dump(self.p_embedding, file)
        with open(tfidfv_path, "wb") as file:
            pickle.dump(self.tfidfv, file)
        print("Embedding pickle saved.")

else: # bm25
    bm25_name = f"bm25.bin"
    bm25_path = os.path.join(self.data_path, bm25_name)
    if os.path.isfile(bm25_path):
        with open(bm25_path, "rb") as file:
            self.bm25 = pickle.load(file)
        print("Embedding bm25 pickle load.")
    else:
        print("Building bm25... It may take 1 minute and 30 seconds...")
        # bm25 must tokenizer first
        # because it runs pool inside and this cuases unexpected result.
        tokenized_corpus = []
        for c in tqdm(self.contexts):
            tokenized_corpus.append(self.tokenize_fn(c))
        self.bm25 = MyBm25(tokenized_corpus, k1 = self.k1, b = self.b, epsilon=self.epsilon)
```

2. DenseRetriever.py

Dense Embedding을 만들기 위한 BERT Encoder를 만들어준다.

해당 코드는 실습기반 코드로 Pretrained를 불러와주는 것

그리고 기존에 제작한 SparseRetrieval을 상속한 DenseRetrieval을 만들어준다.

```python
class BertEncoder(BertPreTrainedModel):
  def __init__(self, config):
    super(BertEncoder, self).__init__(config)
    self.bert = BertModel(config)
    self.init_weights()

  def forward(self, input_ids,
              attention_mask=None, token_type_ids=None):

      outputs = self.bert(input_ids,
                          attention_mask=attention_mask,
                          token_type_ids=token_type_ids)
      pooled_output = outputs[1]
      return pooled_output
```

```python
class DenseRetrieval(SparseRetrieval):
    """ SparseRetreival을 활용해, 메소드를 DenseRetrieval에 맞춰 오버라이딩
        기존에서 p_embedding, contexts, tfidfv를 가져옵니다.
        arguments: train_data: 기존 wiki데이터가 아닌 특정데이터를 활용할때 추가
    """
    def __init__(self, tokenize_fn, data_path, context_path, dataset_path, tokenizer, train_data, num_neg, is_bm25=True,
    wandb=False):
        super().__init__(tokenize_fn, data_path, context_path)
        self.is_bm25 = is_bm25
        self.tokenize_fn = tokenize_fn
        self.org_dataset = load_from_disk(dataset_path)
        self.train_data = train_data
        self.num_neg = num_neg
        self.p_with_neg = []
        self.p_encoder = None
        self.q_encoder = None
        self.dense_p_embedding = []
        self.tokenizer = tokenizer
        self.wandb = wandb
        self.get_sparse_embedding()
```

2. DenseRetriever.py

negative sampling을 위한 코드를 제작한 것

dot product로 유사도가 계산되기 때문에 어떤 임베딩인지

상관 없이 유사도를 구할 수 있다.

즉, bm25기반으로도 코드가 돌아갈 수 있는 것

⇒ 이를 통해서 추후에 negative sample을 뽑을 때 정답은 아니나

최대한 유사도가 높은 문서를 후보로 선정해주게 된다.

```python
def get_topk_similarity(self, qeury_vec, k):
    result = qeury_vec * self.p_embedding.T
    result = result.toarray()

    doc_scores3 = np.partition(result, -k)[:, -k:][:, ::-1]
    ind = np.argsort(doc_scores3, axis=-1)[:, ::-1]
    doc_scores3 = np.sort(doc_scores3, axis=-1)[:, ::-1]
    doc_indices3 = np.argpartition(result, -k)[:, -k:][:, ::-1]
    r, c = ind.shape
    ind = ind + np.tile(np.arange(r).reshape(-1, 1), (1, c)) * c
    doc_indices3 = doc_indices3.ravel()[ind].reshape(r, c)

    return doc_scores3, doc_indices3

def get_resverse_topk_similarity(self, qeury_vec, k):
    """
    Arguments:
        queries (List): 하나의 Query를 받습니다.
        k (Optional[int]): 1 하위 몇개의 Passage를 반환할지 정합니다.
    Note:
        !주의사항! Sparse클래스와 달리 하위 k개의 Passage를 반환합니다!
    """
    result = qeury_vec * self.p_embedding.T
    result = result.toarray()

    doc_scores3 = np.partition(result, k)[:, :k][:, ::-1]
    ind = np.argsort(doc_scores3, axis=-1)[:, :]# ::-1]
    doc_scores3 = np.sort(doc_scores3, axis=-1)[:, :]# ::-1]
    doc_indices3 = np.argpartition(result, k)[:, :k][:, :]# ::-1]
    r, c = ind.shape
    ind = ind + np.tile(np.arange(r).reshape(-1, 1), (1, c)) * c
    doc_indices3 = doc_indices3.ravel()[ind].reshape(r, c)

    return doc_scores3, doc_indices3
```

2. DenseRetriever.py

negative sample이 추가된 train data 제작

negative sample의 10배만큼 후보를 뽑은 후에 거기서 topk개의 negative sample을 제작해주게 된다.

그리고 그것을 통해서 tokenizer에 넣어주고 dataset으로 들어가기 적합한 형태의 train_dataset 제작

즉, 여기서 Retrieval이 1차적으로 이루어지기 때문에 시간 소요가 꽤 오래된다.

그리고 밑에 있는 train에서 영차영차 학습된다.

```python
def make_train_data(self, tokenizer):
    """ Note: Dense Embedding학습을 하기 위한 데이터셋을 만듭니다. """
    print("make_train_data...")
    corpus = np.array(self.contexts)
    queries = self.train_data['question']
    top_k = self.num_neg * 10

    if self.is_bm25==True:
        global retriever
        retriever = self
        doc_scores, doc_indices = par_search(queries, top_k)
    else:
        query_vec = self.tfidfv.transform(queries)
        doc_scores, doc_indices = self.get_topk_similarity(query_vec, top_k)

    neg_idxs = []
    for idx, ind in enumerate(tqdm(doc_indices)): # 4000
        neg_idx = []
        for i in range(len(ind)): # 2~20 find negative
            if not self.contexts[ind[i]][:10] in self.train_data['context'][idx]:
                neg_idx.append(ind[i])
            if len(neg_idx)==self.num_neg: break
        neg_idxs.append(neg_idx)

    with open('../data/neg_idxs.pickle', "wb") as f:
        pickle.dump(neg_idxs, f)

    print(neg_idxs)
    for idx, c in enumerate(tqdm(self.train_data['context'])):
        p_neg = corpus[neg_idxs[idx]]
        self.p_with_neg.append(c)
        self.p_with_neg.extend(p_neg)

    with open('../data/p_with_neg.pickle', "wb") as f:
        pickle.dump(self.p_with_neg, f)

    print(self.p_with_neg)
```

2. DenseRetriever.py

문제의 그리고 애증의 topk_experiment(얘 경로 설정 때문에 2-3일이 날라감)

retrieve_dpr 즉, dense_retrieval의 inference를 한 번 진행하고

그것이 얼마나 일치하는 수치를 보이는지를 보여주는지에 대한 코드

쉬울 거라 생각했는데 이 부부에서 시간소요가 너무 많이 되었다..

```python
def topk_experiment(self, topK_list, dataset, datatset_name="train"):
    """ MRC데이터에 대한 성능을 검증합니다. retrieve를 통한 결과 + acc측정"""
    result_dict = {}
    for topK in tqdm(topK_list):
        # retrieve_dpr을 통해서 topk에 해당하는 임베딩을 list에 따라 만들어준다.
        result_retriever = self.retrieve_dpr(dataset = dataset, topk=topK, p_encoder = self.p_encoder, q_encoder = self.
        q_encoder)
        correct = 0
        for index in tqdm(range(len(result_retriever)), desc="topk_experiment"):
            # 정답이면 1씩 증가
            # 200이 아니라 topK개수만큼 retrieve를 하는 것
            if result_retriever['original_context'][index] in result_retriever['context'][index][:topK]:
                correct += 1
        result_dict[datatset_name + "_topk_" + str(topK)] = correct/len(result_retriever)
    return result_dict
```

3. dpr_score.py

matmul을 통해서 dpr score를 구하고 그것을 통해 앞선 dense_retriever.py에서 구현한 top K 가 돌아가게 된다.

여기서 많은 수의 질문과 context가 한 번에 연산이 이루어지기 때문에 많은 연산량이 필요

최종적으로 메모리가 터져서 어떻게 해야 할지 고민이 많았다.

```python
def get_dpr_score(query, contexts, tokenizer, p_encoder, q_encoder):
    # p_encoder = BertEncoder.from_pretrained("klue/bert-base")
    # q_encoder = BertEncoder.from_pretrained("klue/bert-base")

    # p_encoder.load_state_dict(torch.load(p_encoder_path))
    # q_encoder.load_state_dict(torch.load(q_encoder_path))
    # if torch.cuda.is_available():
    #     p_encoder.cuda()
    #     q_encoder.cuda()

    with torch.no_grad():
        p_encoder.eval()
        q_encoder.eval()
        q_seqs_val = tokenizer(query, padding="max_length", truncation=True, return_tensors='pt').to('cuda')
        q_emb = q_encoder(**q_seqs_val).to('cpu') #(num_query, emb_dim)
        p_embs = []
        for p in contexts:
            p = tokenizer(p, padding="max_length", truncation=True, return_tensors='pt').to('cuda')
            p_emb = p_encoder(**p).to('cpu').numpy()
            p_embs.append(p_emb)

        p_embs = torch.Tensor(p_embs).squeeze()  # (num_passage, emb_dim)
        dot_prod_scores = torch.matmul(q_emb, torch.transpose(p_embs, 0, 1))
    return dot_prod_scores
```

4. retrieval.py의 retrieve_dpr

일단 dpr_score를 get_dpr_score를 통해 구한다.

그리고 bm25 score를 라이브러리를 이용해서 가져온다.

그리고 비율을 정하고 top k 를 선정해준다.

```python
def retrieve_dpr(self, dataset, topk: Optional[int] = 20, p_encoder = None, q_encoder = None):
    print("dpr mode!")
    tokenizer = AutoTokenizer.from_pretrained("klue/bert-base")
    # 예측을 통해 생성된 score df를 얻는다.
    dpr_score = get_dpr_score(dataset['question'], self.contexts, tokenizer, p_encoder, q_encoder)

    bm25_score = []
    for query in dataset['question']:
        tok_q = self.tokenize_fn(query)
        bm25_score.append(self.bm25.get_scores(tok_q))
    bm25_score = torch.tensor(np.array(bm25_score))
    dpr_score = softmax(dpr_score,dim=1)
    bm25_score = softmax(bm25_score,dim=1)
```

```python
total_score = []
for idx in range(len(dataset['question'])):
    # grid search를 통해 적절한 값을 찾는다.
    total_score.append((dpr_score[idx]*0.1+bm25_score[idx]).tolist())
total_score = torch.tensor(np.array(total_score))
ranks = torch.argsort(total_score, dim=1, descending=True).squeeze()
context_list = []
for index in range(len(ranks)):
    k_list = []
    for i in range(topk):
        k_list.append(self.contexts[ranks[index][i]])
    context_list.append(k_list)

total = []
for idx, example in enumerate(
    tqdm(dataset, desc="Sparse retrieval: ")
):
    tmp = {
        # Query와 해당 id를 반환합니다.
        "question": example["question"],
        "id": example["id"],
        # Retrieve한 Passage의 id, context를 반환합니다.
        "context_id": ranks[idx][:topk],
        "context": " ".join(
            context_list[idx]
        ),
    }
    # if "context" in example.keys() and "answers" in example.keys():
    #     # validation 데이터를 사용하면 ground_truth context와 answer도 반환합니다.
    #     tmp["original_context"] = example["context"]
    #     tmp["answers"] = example["answers"]
    total.append(tmp)

cqas = pd.DataFrame(total)
return cqas
```

5. Inference.py

위의 모든 코드는 거의 동일

main이 실행되면 결국 아래의 두 코드가 실행되는 것

```
# True일 경우 : run passage retrieval
if data_args.eval_retrieval:
    datasets = run_sparse_retrieval(
        tokenizer.tokenize, datasets, training_args, data_args, tokenizer)

# eval or predict mrc model
if training_args.do_eval or training_args.do_predict:
    run_mrc(data_args, training_args, model_args, datasets, tokenizer, model)
```

retrieval 이후에 mrc가 이루어지는 과정은 기존과 동일하다.

5. Inference.py

run_sparse_retrieval 안에서 DenseRetrieval이 돌아갈 수 있도록 구현

load 모델이 달라질 때는

밑의 encoder를 수정해주면 된다.

그리고 get_dense_embedding으로

mrc를 위한 dataset 제작

```python
def run_sparse_retrieval(
    tokenize_fn: Callable[[str], List[str]],
    datasets: DatasetDict,
    training_args: TrainingArguments,
    data_args: DataTrainingArguments,
    tokenizer = None,
    data_path: str = "../data",
    context_path: str = "wikipedia_documents.json",
    p_encoder = None,
    q_encoder = None
) -> DatasetDict:

    # Query에 맞는 Passage들을 Retrieval 합니다.
    if not data_args.dpr_negative:
        retriever = SparseRetrieval(
            tokenize_fn=tokenize_fn, data_path=data_path, context_path=context_path, is_bm25=data_args.bm25,
            p_encoder= p_encoder, q_encoder=q_encoder, use_wiki_preprocessing=data_args.use_wiki_preprocessing
        )
        retriever.get_sparse_embedding()
    else:
        retriever = DenseRetrieval(tokenize_fn=tokenize_fn, data_path = data_path,
                                   context_path = context_path, dataset_path=data_path+"/train_dataset",
                                   tokenizer=tokenizer, train_data=datasets["validation"],
                                   num_neg=12, is_bm25=data_args.bm25)

        model_checkpoint = "klue/bert-base"
        retriever.load_model(model_checkpoint, "./outputs/dpr/p_encoder_14.pt", "./outputs/dpr/q_encoder_14.pt")
        retriever.get_dense_embedding()
        # with open("./data/dense_embedding.bin", "rb") as f: # dense_embedding 한번 실행후 진행
        # retriever.dense_p_embedding = pickle.load(f)
```

5. Inference.py

dpr를 사용하게 되면 bm25 + sparse를 같이 사용

그렇지 않으면 bin 파일에 따른 embedding 진행(기본이 bm25로 설정되어 있다.)

```python
if data_args.use_faiss:
    retriever.build_faiss(num_clusters=data_args.num_clusters)
    df = retriever.retrieve_faiss(
        datasets["validation"], topk=data_args.top_k_retrieval
    )
else:
    # if bm25, parallel is faster. ELSE, numpy in TFIDF outperforms the parallel. :/
    if data_args.bm25:
        start = time.time()
        print("Calculating BM25 similarity...")
        if data_args.dpr : # dpr + bm25
            df = retriever.retrieve_dpr(
                datasets["validation"], topk=data_args.top_k_retrieval, p_encoder = retriever.p_encoder, q_encoder =
                retriever.q_encoder
            )
        else:
            df = retriever.retrieve(
                datasets["validation"], topk=data_args.top_k_retrieval
            )
        end = time.time()
        print("Done! similarity processing time :%d secs "%(int(end - start)))
    else:
        df = retriever.retrieve(datasets["validation"], topk=data_args.top_k_retrieval)
```

결론

그리고 이렇게 Retrieval이 된 것들을 가지고 mrc가 진행된다.

고생 많으셨습니다ㅎㅎ..