NLP 5주차 스터디

작성일시	@2024년 2월 16일 오전 12:18
텍스트	7.2절 - 7.4절
복습	☐

7.2 QA 파이프라인 개선하기

리트리버는 전체 QA 시스템 성능의 상한선을 설정하므로 리트리버의 성능을 높이는 것이 중요

7.2.1 리트리버 평가하기

희소와 밀집 표현의 성능을 비교할 수 있는 일반적인 지표 “**재현율**”

▫ **Recall (재현율)**

추출된 관련 문서의 비율을 측정

여기서 ‘관련’이란 단순히 답이 텍스트 안에 있는지 없는지를 의미.

즉 질문이 주어지면 리트리버가 반환한 최상위 k개 문서에 답이 등장한 횟수 카운트해 재현율 계산

헤이스택에서 리트리버를 평가하는 방법

- 리트리버가 제공하는 `eval()` 메서드

  `eval()`은 오픈 도메인과 클로즈드 도메인 QA에 모두 사용 가능!

  하지만 각 문서가 하나의 제품과 쌍을 이루어 모든 쿼리에서

  제품 ID를 필터링해야하는 SubjQA 같은 데이터셋에서는 사용 XX

- `EvalDocuments` 클래스와 리트리버를 결합하는 사용자 정의 Pipeline 만들기

  이를 통해 사용자 정의 지표와 쿼리 흐름 구현

> 📢 **mAP (mean average precision)**
> 재현율을 보완하는 지표로 문서 순위에서 정답을 높은 위치에 놓은
> 리트리버에 보상을 제공

>> 제품마다 재현율을 평가하고 그 결과를 수집해야 하므로 우리는 두 번째 방법 사용

▼ 코드

```
class PipelineNode:
    def __init__(self):
        self.outgoing_edges = 1

    def run(self, **kwargs):
        ...
        return (outputs, "outgoing_edge_name")
```

Pipeline 그래프에 있는 각 노드는 어떤 입력을 받고

`run()` 메소드로 어떤 출력을 만드는 하나의 클래스를 나타냄

- `kwargs` : 그래프에 있는 이전 노드의 출력에 해당

- 다음 노드를 위해 출력의 튜플과 노드에서 나가는 edge의 이름을 반환하기 위해

  `run()` 메서드 안에서 처리됨

- 유일한 요구사항은 노드의 출력 개수를 나타내는 `outgoing_edges` 속성을 포함하는 것

  어떤 조건에 따라 입력을 라우팅하는 브랜치가 파이프라인에 없다면,

  대부분 `outgoing_deges = 1`

▼ 리트리버 평가할 노드 만들기

```
from haystack.pipeline import Pipeline
from haystack.eval import EvalDocuments

class EvalRetrieverPipeline:
    def __init__(self, retriever):
        self.retriever = retriever
        self.eval_retriever = EvalDocuments()
        pipe = Pipeline()
        pipe.add_node(component=self.retriever, name="Es
                      inputs=["Query"])
        pipe.add_node(component=self.eval_retriever, nam
                      inputs=["ESRetriever"])
        self.pipeline = pipe


pipe = EvalRetrieverPipeline(es_retriever)
```

`run()` 메소드에서 정답이 있는 문서를 추적하는 `EvalDocuments` 클래스 사용

그 다음 리트리버를 표현한 노드 뒤에 평가 노드를 추가해 `Pipeline` 그래프 만들기

- 각 노드는 name 매개변수와 리스트로 inputs 매개변수를 지정
- 대개 노드는 출력 edge가 하나

  >> 따라서 그냥 inputs에 이전 노드의 이름을 포함시키면 됨

▼ 평가 파이프라인을 만들었으니 쿼리와 이에 해당하는 답 전달

```
from haystack import Label

labels = []
for i, row in dfs["test"].iterrows():
    # 리트리버에서 필터링을 위해 사용하는 메타데이터
    meta = {"item_id": row["title"], "question_id": row
    # 답이 있는 질문을 레이블에 추가합니다
    if len(row["answers.text"]):
        for answer in row["answers.text"]:
            label = Label(
                question=row["question"], answer=answer
                meta=meta, is_correct_answer=True, is_c
                no_answer=False)
            labels.append(label)
    # 답이 없는 질문을 레이블에 추가합니다
    else:
        label = Label(
            question=row["question"], answer="", id=i,
            meta=meta, is_correct_answer=True, is_corre
            no_answer=True)
        labels.append(label)
```

문서저장소에 있는 전용 label 인덱스에 답을 추가

: 헤이스택은 표준 방식으로 답 범위와 메타데이터를 나타내는 Label 객체 제공

- label 인덱스를 채우기 위해 먼저 테스트 세트에 있는 질문을 순회하며
  답과 추가적인 메타데이터를 추출해 Laebel 객체의 리스트 만들기

▼ 위의 레이블 중 하나 확인

```
print(labels[0])
```

```
{'id': '336690e2-4398-4ba8-9744-b6363b373427', 'created_at': None, 'updated_at':
None, 'question': 'What is the tonal balance of these headphones?', 'answer': 'I
have been a headphone fanatic for thirty years', 'is_correct_answer': True,
'is_correct_document': True, 'origin': 'd0781d13200014aa25860e44da9d5ea7',
'document_id': None, 'offset_start_in_doc': None, 'no_answer': False,
'model_id': None, 'meta': {'item_id': 'B00001WRSJ', 'question_id':
'd0781d13200014aa25860e44da9d5ea7'}}
```

- 질문과 답변 쌍과 쿼리마다 문서 저장소를 필터링할 수 있는

  고유한 질문 ID가 담긴 origin 필드가 있음
- 제품으로 레이블을 필터링하기 위해 meta 필드에 제품 ID 추가

▼ 레이블이 준비됐으니 일래스틱서치의 label 인덱스에 저장

```
document_store.write_labels(labels, index="label")
print(f"""{document_store.get_label_count(index="label")}개
질문 답변 쌍을 로드했습니다.""")
```

```
358개의 질문 답변 쌍을 로드했습니다.
```

▼ 파이프라인에 전달하기 위해 질문 ID와 이에 상응하는 답변 매핑

```
labels_agg = document_store.get_all_labels_aggregated(
    index="label",
    open_domain=True,
    aggregate_by_meta=["item_id"]
)
print(len(labels_agg))
```

```
330
```