[주제]

retrieval-augmented generation with noisy documents

[구성원]

120240328 정가연, 120240269 김도현, 120240288 홍문기

[사용한 라이브러리]

spacy, symspellpy, pandas, numpy, json, tqdm, random, nltk, skicit-learn, torch, transformers

[구현 방식]

1. Preprocessing Wiki Data

- 'remove_special_characters' function : 주어진 문장에서 영어 알파벳, 숫자, 공백을 제외한 모든 특수 문자를 제거하는 전처리 작업을 수행하는 기능을 한다.

2. Construct RAG Architecture

- 'VectorDatabase' class : 위키 문서 데이터를 기반으로 벡터 데이터베이스를 생성하고 유사 문서 검색 기능을 한다. init 메서드에서는 문서 데이터를 입력받은 후 TF-IDF 를 통해서 벡터화하고 document_embeddings 변수에 저장한다. TF_IDF 란 Term Frequency-Inverse Document Frequency 의 약자로, 문서의 집합에서 단어의 중요도를 측정하여 수치화된 벡터로 변환한다. $TF(t,d) = \frac{f(t,d)}{n(d)}$ $IDF(t) = log(\frac{N}{d(t)})$
- search_similar_doc 메서드에서는 쿼리를 입력받은 후 TF-IDF 벡터화하여 query_embedding 변수에 저장한다. 그 후 데이터베이스에 저장된 문서 벡터와 쿼리 벡터 간의 코사인 유사도를 계산하고, 가장 유사한 문서의 인덱스를 도출한다. 도출된 인덱스를 사용해서 가장 유사한 문서를 최종 출력한다. $cosine\ similarity = cos(\theta) = \frac{A \cdot B}{|A||B|}$
- 'generate_with_prompt_tuning' function : 쿼리를 바탕으로 생성형 AI 모델을 활용해서 답변을 생성하는 기능을 한다. 쿼리 앞에 질문 유형과 답변 형식을 명시하는 프롬포트를 생성했다. 쿼리를 토큰화 한 후, max_length 와 padding 등의 파라미터를 설정하여 입력 데이터를 생성한다. 그 후 생성된 입력 데이터를 gemma 모델에 넣어 답변을 생성했다. 생성된 답변은 디코딩하여 문자열 형태로 반환했다. 최종적으로 답변 앞뒤 공백과 불필요한 문자열을 제거하여 최종 답변을 반환했다.
- 'rag' function : 문서 데이터와 쿼리 목록을 입력받아 쿼리 별 답변을 생성하고 관련 문서를 반환하는 기능을 한다.. 먼저 입력된 문서 데이터를 기반으로 vector_db 객체를 생성한다. 각 쿼리를 사용해서 vecotr_db 에서 유사 문서를 검색한다. 다음으로 generative_with_prompt_tuning 함수를 사용해서 질문에 대한 짧은 답변을 생성한다. 예측 결과 딕셔너리에 쿼리 인덱스를 키값으로 하여 생성된 답변과 관련 문서를 저장함으로써 최종적으로 쿼리 별 예측 결과 딕셔너리를 반환했다.

3. Running Rag

- rag 함수를 호수를 호출하여 질의 응답 시스템을 실행한다. document data 로는 processed_wikipedia.txt 데이터의 문장 100,000 개를 사용해주었고 쿼리 목록으로는 qa_test.json 데이터의 question 9,785 개를 사용했다.

4. Post Processing

- spacy, symspellpy : 이 모듈은 위키 데이터의 오탈자를 수정하기 위해 사용하였다. spacy 모델은 en_core_web_md 를 사용해 각 문장을 NER 토큰으로 구분하는데 사용하였다. 이렇게 사용한 이유는 오탈자 수정 시 고유 명사를 수정하는 경우를 배제하기 위해서이다. 오타 수정은 symspelly 의 frequency_dictionary_en_82_765.txt 를 사용하였다. 이 파일에는 자주 사용하는 영어 단어와 그 빈도수를 저장한 사전형 파일이다. 입력된 각 문장을 NER 단위로 분리한 다음 고유명사를 제외한 나머지 단어를 하나씩 사전 내 단어와 비교하면서 오탈자가 있는지 확인하는 과정을 거친다. 이때, 최대 수정 개수는 알파벳 3 개까지로 설정하였다. 일부 단어의 경우 고유명사로 취급되어 수정 가능한 단어임에도 수정되지 않는 경우가 있었다. 이 경우 해당 고유명사를 오타 수정한 다음 수정된 단어가 사전에 있는지 확인하였다. 사전에 있으면 수정된 단어를 사용하였고 사전에 없으면 해당 고유명사를 사전에 추가한 다음 사용하였다.

[실험 성능]

Submission File				
버전	Experiment	Score	진행자	파일명
v1	train_qa.json으로 양식만 지켜서 생성	0.0019	가연	submission_v1_0.0019.csv
v2	wikipedia 50,000개로 RAG 수행	0.01	가연	submission_v2_0.00100.csv
v3	wikipedia 500,000개로 RAG 수행	0.0103	가연	submission_v3_0.0103.csv
v4	Just Gemma Answer	0.0121	가연	submission_v4_0.0121.csv
v5	wikipedia 100,000개 +tfidf embedding + cosine similarity + postprocessing(team_v3)	0.0392	가연	submission_v5_0.0392csv
v6	embedding 방식 변경	0.0291	도현	submission_v6_0.0291.csv
v7	processing wikipedia 100,000개 + tfidf embedding + postprocessing gemma generate(only query)	0.1347	가연	submission_v7_0.1347.csv
v8	processing wikipedia 500,000개 + tfidf embedding + postprocessing gemma generate(context + query)	0.0871	가연	submission_v8_0.0871.csv
v9	processing wikipedia 1,000,000개 + tfidf embedding + postprocessing gemma generate(only query)	0.1343	가연	submission_v9_0.1343.csv

실험 성능은 위과 같이 표를 만들어서 관리했다. 앞서 기술한 구현 방식은 version 7 에 해당하는 방식으로 리더보드 상 가장 좋은 성능을 보였다. 이를 기반으로 submission 파일을 생성해서 최종 제출하였다.