CH07 - 텍스트 분할(Text Splitter)

문서분할

> 로드된 문서들을 효율적으로 처리하고, 시스템이 정보를 보다 잘 활용할 수 있도록 준비하는 중요한 과정

목적

- 크고 복잡한 문서를 LLM이 받아들일 수 있는 효율적인 작은 규모의 조각으로 나누는 작업
- 나중에 사용자가 입력한 질문에 대하여 보다 효율적인 정보만 압축/선별하여 가져오기 위함

분할의 필요성

1. **정확성**

→ 질문(Query)에 연관성이 있는 정보만 가져옴

2. **효율성**

→ 전체 문서를 LLM으로 입력하면 비용 과다

→ 할루시네이션 방지

문서분할 과정

1. **문서 구조 파악**

→ 문서의 헤더, 푸터, 페이지 번호, 섹션 제목 등을 식별

2. **단위 선정**

→ 문서를 어떤 단위로 나눌지 결정

→ 페이지별, 섹션별, 또는 문단별 등

→ 문서의 내용과 목적에 따라 다름

3. **단위 크기 선정(chunk size)**

→ 문서를 몇 개의 토큰 단위로 나눌 것인지 정함

4. **청크 오버랩**

→ 분할된 끝 부분에서 맥락이 이어지도록 일부를 overlap 분할하는 것이 원칙

I. 문자 텍스트 분할(CharacterTextSplitter)

CharacterTextSplitter

- 가장 간단한 방식
- “\n\n”을 기준으로 문자 단위로 텍스트를 분할하고, 청크의 크기를 문자 수로 측정함
  - 텍스트 분할 방식: 단일 문자 기준
  - 청크 크기 측정 방식: 문자 수 기준
- `separator` 매개변수로 분할할 기준을 설정함. 기본 값은 `"\n\n"`
- `chunk_size` 매개변수를 250 으로 설정하여 각 청크의 최대 크기를 250자로 제한함.
- `chunk_overlap` 매개변수를 50으로 설정하여 인접한 청크 간에 50자의 중복을 허용함.
- `length_function` 매개변수를 len으로 설정하여 텍스트의 길이를 계산하는 함수를 지정함.
- `is_separator_regex` 매개변수를 False로 설정하여 separator를 정규식이 아닌 일반 문자열로 처리함.

```
from langchain_text_splitters import CharacterTextSplitter

text_splitter = CharacterTextSplitter(
    # 텍스트를 분할할 때 사용할 구분자를 지정합니다. 기본값은 "\n\n"입니다.
    # separator=" ",
    # 분할된 텍스트 청크의 최대 크기를 지정합니다.
    chunk_size=250,
    # 분할된 텍스트 청크 간의 중복되는 문자 수를 지정합니다.
    chunk_overlap=50,
    # 텍스트의 길이를 계산하는 함수를 지정합니다.
    length_function=len,
    # 구분자가 정규식인지 여부를 지정합니다.
    is_separator_regex=False,
)
```

- `text_splitter`를 사용하여 `file` 텍스트를 문서 단위로 분할함.

- 분할된 문서 리스트 중 첫 번째 문서(`texts[0]`)를 출력함.

II. 재귀적 문자 텍스트 분할 (RecursiveCharacterTextSplitter)

- 일반적인 텍스트에 권장되는 방식
- 문자 목록을 매개변수로 받아 동작함.
- 분할기는 청크가 충분히 작아질 때까지 주어진 문자 목록의 순서대로 텍스트를 분할하려고 시도
- **단락 → 문장 → 단어** 순서로 재귀적으로 분할함
  - 텍스트가 분할되는 방식: 문자 목록(`["\n\n", "\n", " ", ""]`) 에 의해 분할됨.
  - 청크 크기가 측정되는 방식: 문자 수에 의해 측정됨.

III. 토큰 텍스트 분할(TokenTextSplitter)

** 언어 모델에는 토큰 제한이 있음! 따라서 토큰 제한을 초과하지 않아야 함.

- 텍스트를 토큰 수를 기반으로 청크를 생성할 때 유용함.

`tiktoken` 은 OpenAI에서 만든 빠른 `BPE Tokenizer`

```
# file 텍스트를 청크 단위로 분할합니다.
texts = text_splitter.split_text(file)
분할된 청크의 개수를 출력합니다.

print(len(texts))  # 분할된 청크의 개수를 출력합니다.
```

```
# texts 리스트의 첫 번째 요소를 출력함.
print(text[0])
```

SentenceTransformers

- `SentenceTransformersTokenTextSplitter`는 `sentence-transformer` 모델에 특화된 텍스트 분할기
- 기본 동작은 사용하고자 하는 sentence transformer 모델의 토큰 윈도우에 맞게 텍스트를 청크로 분할하는 것.

Hugging Face tokenizer

** Hugging Face는 다양한 토크나이저를 제공함.

Hugging Face의 토크나이저 중 하나인 GPT2TokenizerFast를 사용하여 텍스트의 토큰 길이를 계산함.

- 텍스트 분할 방식은 다음과 같음:
  - 전달된 문자 단위로 분할됨.
- 청크 크기 측정 방식은 다음과 같음:
  - Hugging Face 토크나이저에 의해 계산된 토큰 수를 기준으로 함.
  - `GPT2TokenizerFast` 클래스를 사용하여 `tokenizer` 객체를 생성함.
  - `from_pretrained` 메서드를 호출하여 사전 학습된 "gpt2" 토크나이저 모델을 로드함.

IV. 시멘틱 청커(SemanticChunker)

- 텍스트를 의미론적 유사성에 기반하여 분할함.
- 텍스트를 문장 단위로 분할한 후, 3개의 문장씩 그룹화하고, 임베딩 공간에서 유사한 단어들을 병합하는 과정을 거침.
- `SemanticChunker`는 LangChain의 실험적 기능 중 하나로, 텍스트를 의미론적으로 유사한 청크로 분할하는 역할을 함.

`create_documents()` 함수를 사용하여 청크를 문서로 변환할 수 있음.

```
# text_splitter를 사용하여 분할합니다.
docs = text_splitter.create_documents([file])
print(docs[0].page_content)  # 분할된 문서 중 첫 번째 문서의 내용을 출력합니다.
```

- 이 chunker는 문장을 "분리"할 시점을 결정하여 작동함.
- 두 문장 간의 **임베딩 차이를 살펴봄으로써 이루어짐.

(** 글자·단어·문장 같은 "의미"를 숫자 벡터로 바꿔서, 컴퓨터가 거리로 비교할 수 있게 만드는 것)

- 그 차이가 특정 **임계값**을 넘으면 문장이 분리됨
  - Percentile
  - Standard Deviation
  - Interquartile

V. 코드 분할(Python, Markdown, JAVA, C++, C#, GO, JS, Latex 등)

Split code

- `CodeTextSplitter`를 사용하면 다양한 프로그래밍 언어로 작성된 코드를 분할할 수 있음.
- 이를 위해서는 `Language` enum을 import하고, 해당하는 프로그래밍 언어를 지정해주면 됨! (Python, Markdown, Java...)

VI. 마크다운 헤더 텍스트 분할 (MarkdownHeaderTextSplitter)

- 마크다운 파일의 특정 부분, 즉 헤더별로 내용을 나누고 싶을 때가 있음.
- 서로 연관된 정보 덩어리, 즉 '청크'를 만들고 싶은 경우
- 텍스트의 공통된 맥락을 유지하면서도, 문서의 구조적 요소를 효과적으로 활용하려는 시도

`MarkdownHeaderTextSplitter`

- 문서를 지정된 헤더 집합에 따라 분할하여, 각 헤더 그룹 아래의 내용을 별도의 청크로 관리할 수 있게 함.
- 문서의 전반적인 구조를 유지하면서도 내용을 더 세밀하게 다룰 수 있게 됨.

- 마크다운 문서의 헤더(`#`, `##`, `###` 등)를 기준으로 텍스트를 분할하는 역할을 함.
- `markdown_document` 변수에 마크다운 형식의 문서가 할당됨.
- `headers_to_split_on` 리스트에는 마크다운 헤더 레벨과 해당 레벨의 이름이 튜플 형태로 정의됨.
- `MarkdownHeaderTextSplitter` 클래스를 사용하여 `markdown_splitter` 객체를 생성하며, `headers_to_split_on` 매개변수로 분할 기준이 되는 헤더 레벨을 전달함.
- `split_text` 메서드를 호출하여 `markdown_document`를 헤더 레벨에 따라 분할함.

- 이전의 `MarkdownHeaderTextSplitter` 로 분할된 결과를 다시 `RecursiveCharacterTextSplitter` 로 분할함.

```
chunk_size = 200  # 분할된 청크의 크기를 지정합니다.
chunk_overlap = 20  # 분할된 청크 간의 중복되는 문자 수를 지정합니다.
text_splitter = RecursiveCharacterTextSplitter(
    chunk_size=chunk_size, chunk_overlap=chunk_overlap
)

# 문서를 문자 단위로 분할합니다.
splits = text_splitter.split_documents(md_header_splits)
# 분할된 결과를 출력합니다.
```

```
for header in splits:
    print(f"{header.page_content}")
    print(f"{header.metadata}", end="\n======================\n")
```

VII. HTML 헤더 텍스트 분할 (HTMLHeaderTextSplitter)

- 텍스트를 요소 수준에서 분할하고 각 헤더에 대한 메타데이터를 추가하는 “구조 인식” 청크 생성기
- 각 청크와 관련된 메타데이터를 추가함
- 요소별로 청크를 반환하거나 동일한 메타데이터를 가진 요소를 결합할 수 있음
  - 관련 텍스트를 의미론적으로 그룹화
  - 문서 구조에 인코딩된 컨텍스트 풍부한 정보를 보존하는 것을 목표로 함

① HTML 문자열을 사용하는 경우

- `headers_to_split_on` 리스트에 분할 기준이 되는 헤더 태그와 해당 헤더의 이름을 튜플 형태로 지정함.
- `HTMLHeaderTextSplitter` 객체를 생성하면서 `headers_to_split_on` 매개변수에 분할 기준 헤더 리스트를 전달함.

② 다른 splitter와 파이프라인으로 연결하고, 웹 URL에서 HTML을 로드하는 경우

- 웹 URL로부터 HTML 콘텐츠를 로드한 후, 이를 다른 splitter와 파이프라인으로 연결하여 처리하는 과정

한계

- HTML 문서 간의 구조적 차이를 처리하려고 시도하지만, 때로는 특정 헤더를 누락할 수 있음
    - 예를 들어, 이 알고리즘은 헤더가 항상 관련 텍스트보다 "위"에 있다고 가정함. 즉, 이전 형제 노드, 조상 노드 또는 이들의 조합에 위치한다는 것임.
    - 다음 뉴스 기사(이 문서 작성 시점 기준)를 보면, 최상위 헤드라인이 "h1" 태그로 되어 있지만 예상되는 텍스트 요소와는 **별개의 하위 트리**에 있음.
    - 따라서 "h1" 요소와 관련 텍스트는 청크 메타데이터에 나타나지 않음. 반면 "h2"와 관련 텍스트는 나타남.

VIII. 재귀적 JSON 분할 (RecursiveJsonSplitter)

- JSON 분할기는 JSON 데이터를 깊이 우선 탐색하여 더 작은 JSON chunk를 생성함
- 중첩된 JSON 객체를 가능한 한 유지하려고 시도하지만, 청크의 크기를 min_chunk_size와 max_chunk_size 사이로 유지하기 위해 필요한 경우 객체를 분할함.
- 값이 중첩된 JSON이 아니라 매우 큰 문자열인 경우, 해당 문자열은 분할되지 않음.

분할하는 기준

1. 텍스트 분할 방식: JSON 값 기준
2. 청크 크기 측정 방식: 문자 수 기준