
- 출처: LangChain 공식 문서 또는 해당 교재명
- 원본 URL: https://smith.langchain.com/hub/teddynote/summary-stuff-documents

---

텍스트 분할 실습

문자 텍스트 분할 (CharacterTextSplitter)

- **가장 간단한 분할 방식**
- 기본적 기준
  - **"\n\n"** = 문자 단위로 텍스트 분할
  - **= 텍스트 분할 방식: 단일 문자 기준**
  - 청크 크기 = 문사 수로 측정
  - **= 청크 크기 측정 방식: 문자 수 기준**

사전 환경 설정

- 사전 `VS Code` 터미널에 설치할 것

```
pip install -qU langchain-text-splitters
```

```
# data/appendix-keywords.txt 파일을 열어서 f라는 파일 객체 생성하기
with open("../07_Text_Splitter/data/appendix-keywords.txt") as f:
    file = f.read()                                          # 파일의 내용을 읽어서 file 변수에 저장
```

```
print(type(file))                  # <class 'str'>
print(len(file))                   # 5733
```

- 파일로부터 읽은 파일의 일부 내용을 출력해보기

```
# 파일으로부터 읽은 내용 일부 출력하기

print(file[:500])
```

- 셀 출력

```
Semantic Search

정의: 의미론적 검색은 사용자의 질의를 단순한 키워드 매칭을 넘어서 그 의미를 파악하여 관련된 결과를 반환하는 검색 방식입니다.
예시: 사용자가 "태양계 행성"이라고 검색하면, "목성", "화성" 등과 같이 관련된 행성에 대한 정보를 반환합니다.
연관키워드: 자연어 처리, 검색 알고리즘, 데이터 마이닝

Embedding

정의: 임베딩은 단어나 문장 같은 텍스트 데이터를 저차원의 연속적인 벡터로 변환하는 과정입니다. 이를 통해 컴퓨터가 텍스트를 이해하고 처리할 수 있게 합
예시: "사과"라는 단어를 [0.65, -0.23, 0.17]과 같은 벡터로 표현합니다.
연관키워드: 자연어 처리, 벡터화, 딥러닝

Token
```

```
정의: 토큰은 텍스트를 더 작은 단위로 분할하는 것을 의미합니다. 이는 일반적으로 단어, 문장, 또는 구절일 수 있습니다.
예시: 문장 "나는 학교에 간다"를 "나는", "학교에", "간다"로 분할합니다.
연관키워드: 토큰화, 자연어
```

`CharacterTextSplitter` 사용

- **`CharacterTextSplitter`를 사용하여 `텍스트`를 `청크(chunk)`로 분할하는 코드**
  - **`separator`**
    - 매개변수로 분할할 기준을 설정
    - 기본 값 = **`"\n\n"`**
  - **`chunk_size`**
    - **`매개변수 = 250`** 으로 설정 → **각 `청크`의 `최대 크기`를 `250자`로 제한**
  - **`chunk_overlap`**
    - **`매개변수 = 50`** 으로 설정 → **`인접한 청크 간에 50자의 중복`을 허용**
  - **`length_function`**
    - **`매개변수 = len`** 으로 설정 → **`텍스트의 길이`를 `계산`하는 함수를 지정**
  - **`is_separator_regex`**
    - **`매개변수 = False`** 로 설정 → **`separator`를 정규식이 아닌 `일반 문자열`로 처리**

```
from langchain_text_splitters import CharacterTextSplitter

text_splitter = CharacterTextSplitter(
    # 텍스트를 분할할 때 사용할 구분자를 지정 (기본값은 "\n\n")
    # separator=" ",

    chunk_size=250,                                 # 분할된 텍스트 청크의 최대 크기 지정

    chunk_overlap=50,                               # 분할된 텍스트 청크 간의 중복되는 문자 수 지정

    length_function=len,                            # 텍스트의 길이를 계산하는 함수 지정

    is_separator_regex=False,                       # 구분자가 정규식인지 여부를 지정
)
```

- **`text_splitter` 사용 → `file 텍스트`를 `문서 단위`로 분할**
- **분할된 문서 `리스트` 중 `첫 번째 문서(texts[0])`를 출력**

```
# text_splitter를 사용하여 state_of_the_union 텍스트를 문서로 분할하기
texts = text_splitter.create_documents([file])

# 분할된 문서 중 첫 번째 문서 출력하기
print(texts[0])
```

- 셀 출력

```
page_content='Semantic Search

정의: 의미론적 검색은 사용자의 질의를 단순한 키워드 매칭을 넘어서 그 의미를 파악하여 관련된 결과를 반환하는 검색 방식입니다.
예시: 사용자가 "태양계 행성"이라고 검색하면, "목성", "화성" 등과 같이 관련된 행성에 대한 정보를 반환합니다.
연관키워드: 자연어 처리, 검색 알고리즘, 데이터 마이닝

Embedding'
```

메타데이터 함께 전달하기

- `메타데이터`가 문서와 함께 분할됨
- `create_documents` 메서드: `텍스트 데이터`와 `메타데이터 리스트`를 인자로 받음

```
# 문서에 대한 메타데이터 리스트를 정의하기
# 각 문서에 대한 메타데이터 = 딕셔너리 형태로 저장
metadatas = [
    {"document": 1},                            # 첫 번째 문서에 대한 메타데이터
    {"document": 2},                            # 두 번째 문서에 대한 메타데이터
]

# 텍스트 분할기(text_splitter)를 사용해 문서 분할하기
# create_documonets 메서드 호출해 문서 분할
"""
    create_documents()
    - 문서 분할
    - 분할할 텍스트 데이터 = 리스트로 전달
    - 각 문서에 해당하는 메타데이터 전달
"""

# text_splitter.create_documents(텍스트 데이터, 메타데이터)
documents = text_splitter.create_documents(
    [
        file,                                   # 첫 번째 분할할 텍스트 데이터
        file,                                   # 두 번째 분할할 텍스트 데이터
    ],
    metadatas=metadatas,                        # 각 문서에 해당하는 메타데이터 전달
)

# 분할된 문서 중 첫 번째 문서 출력하기
print(documents[0])
```

- 셀 출력

```
page_content='Semantic Search

정의: 의미론적 검색은 사용자의 질의를 단순한 키워드 매칭을 넘어서 그 의미를 파악하여 관련된 결과를 반환하는 검색 방식입니다.
예시: 사용자가 "태양계 행성"이라고 검색하면, "목성", "화성" 등과 같이 관련된 행성에 대한 정보를 반환합니다.
연관키워드: 자연어 처리, 검색 알고리즘, 데이터 마이닝

Embedding' metadata={'document': 1}
```

`split_text()`

- `split_text()` 메서드 사용 → 텍스트 분할
- `text_splitter.split_text(file)[0]`
  - `file` 텍스트를 `text_splitter`를 사용하여 `분할`한 후 → 분할된 텍스트 조각 중 `첫 번째 요소`를 반환

```
# text_splitter를 사용하여 file 텍스트를 분할 → 텍스트의 첫 번째 요소 반환

text_splitter.split_text(file)[0]
```

- 셀 출력

```
'Semantic Search\n\n정의: 의미론적 검색은 사용자의 질의를 단순한 키워드 매칭을 넘어서 그 의미를 파악하여 관련된 결과를 반환하는 검색 방식입니다
```

- *next:* ***`재귀적 문자 텍스트 분할 (RecursiveCharacterTextSplitter)`***
