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

---

`재귀적 문자 텍스트 분할 (RecursiveCharacterTextSplitter)`

- `일반적인 텍스트에 권장되는 방식`
- 재귀적 문자 텍스트 분할기
  - `문자 목록` 을 매개변수로 받아 동작
  - `청크가 충분히 작아질 때까지` 주어진 목자 `목록의 순서대로` 텍스트 `분할 시도`

`기본 설정`

- 기본 문자 목록
  - `["\n\n", "\n", " ", ""]`
  - `단락 > 문장 > 단어` 순서로 `재귀적으로 분할`
- 단락 (그 다음으로 문장, 단어) 단위로 의미적으로 가장 강하게 연관된 텍스트 조각으로 간주됨
- 가능한 한 함께 유지하려는 효과가 있음

`사전 환경 설정`

- 사전 `VS Code` 터미널에 설치할 것

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

- `텍스트 분할 방식`:
  - `문자 목록` 에 의애 분할됨

- `["\n\n", "\n", " ", ""]`

- **`청크 크기가 측정되는 방식`**: **`문자 수`** 에 의해 측정됨

```
# 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 fil
```

```
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 반환하
예시: 사용자가 "태양계 행성"이라고 검색하면, "목성", "화성" 등과 같이 관련된 행성에 대한 정보를 반환합
연관키워드: 자연어 처리, 검색 알고리즘, 데이터 마이닝

Embedding

정의: 임베딩은 단어나 문장 같은 텍스트 데이터를 저차원의 연속적인 벡터로 변환하는 과정입니다. 이를 통해
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
예시: 문장 "나는 학교에 간다"를 "나는", "학교에", "간다"로 분할합니다.
연관키워드: 토큰화, 자연어
```

`RecursiveCharacterTextSplitter` 사용

- **`RecursiveCharacterTextSplitter`** 사용 → 텍스트를 작은 청크로 분할하는 예제
  - **`chunk_size`**

        - `chunk_size = 250` 으로 설정 → 각 청크의 크기 제한
    - **`chunk_overlap`**
        - `chunk_overlap = 50` 으로 설정 → 인접한 청크 간에 50 개 문자의 중첩을 허용
    - **`length_function`**
        - `length_function = len` 함수 사용 → 텍스트의 길이 계산
    - **`is_separator_regex`**
        - `is_separator_regex = False` 로 설정 → 구분자로 정규식을 사용하지 않음

```
from langchain_text_splitters import RecursiveCharacterTextSplitter

text_splitter = RecursiveCharacterTextSplitter(
    chunk_size=250,                                    # 청크 크기 매우 작게 설정 (예시를 우
    chunk_overlap=50,                                  # 분할된 텍스트 청크 간의 중복되는 문
    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 문서는 **`texts` 리스트** 에 저장됨
    - **`print(texts[0])`**, **`print(texts[1])`** → 분할된 문서의 첫 번째와 두 번째 문서를 출력

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

print(texts[0])  # 분할된 문서의 첫 번째 문서 출력하기
print("\n", "===" * 50, "\n")
print(texts[1])  # 분할된 문서의 두 번째 문서 출력하기
```

- 셀 출력

```
page_content='Semantic Search

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

Embedding'

==========

page_content='Embedding
```

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

Token'
```

- **`text_splitter.split_text()`** 함수 사용 → **`file`** **텍스트 분할**

```
# 텍스트를 분할하고 분할된 텍스트의 처음 2개 요소를 반환하기

text_splitter.split_text(file)[:2]
```

- 셀 출력

```
['Semantic Search\n\n정의: 의미론적 검색은 사용자의 질의를 단순한 키워드 매칭을 넘어서 그 의미를
'Embedding\n\n정의: 임베딩은 단어나 문장 같은 텍스트 데이터를 저차원의 연속적인 벡터로 변환하는 과정
```

- *next:* ***`토큰 텍스트 분할 (TokenTextSplitter)`***
