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

---

`토큰 텍스트 분할 (TokenTextSplitter)`

- **`언어 모델 = 토큰 제한`** → **토큰 제한을 초과하지 않아야 함**
- **`TokenTextSplitter`** = **텍스트를 `토큰 수`를 `기반`으로 `청크`를 `생성`할 때 `유용`**

`tiktoken`

- **`tiktoken`** = **`OpenAI`에서 만든 빠른 `BPE Tokenizer`**
- 사전 `VS Code` 터미널에 설치할 것

```
pip install --upgrade --quiet langchain-text-splitters tiktoken
```

```
# 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
```

`RecursiveCharacterTextSplitter` 사용

- **`CharacterTextSplitter`** 사용 → 텍스트 분할
- **`from_tiktoken_encoder`** **메서드 사용 →** `Tiktoken` 인코더 기반의 텍스트 분할기를 `초기화`

```
from langchain_text_splitters import CharacterTextSplitter

text_splitter = CharacterTextSplitter.from_tiktoken_encoder(
    chunk_size=300,                                 # 청크 크기 300으로 설정
    chunk_overlap=0,                                # 청크 간 중복되는 부분이 없도록 설정하기
)


# file 텍스트를 청크 단위로 분할하기
texts = text_splitter.split_text(file)
```

- 셀 출력 (4.0s)

```
Created a chunk of size 358, which is longer than the specified 300
Created a chunk of size 315, which is longer than the specified 300
Created a chunk of size 305, which is longer than the specified 300
Created a chunk of size 366, which is longer than the specified 300
Created a chunk of size 330, which is longer than the specified 300
Created a chunk of size 351, which is longer than the specified 300
Created a chunk of size 378, which is longer than the specified 300
Created a chunk of size 361, which is longer than the specified 300
Created a chunk of size 350, which is longer than the specified 300
Created a chunk of size 362, which is longer than the specified 300
Created a chunk of size 335, which is longer than the specified 300
Created a chunk of size 353, which is longer than the specified 300
Created a chunk of size 358, which is longer than the specified 300
Created a chunk of size 336, which is longer than the specified 300
Created a chunk of size 324, which is longer than the specified 300
Created a chunk of size 337, which is longer than the specified 300
Created a chunk of size 307, which is longer than the specified 300
Created a chunk of size 361, which is longer than the specified 300
Created a chunk of size 354, which is longer than the specified 300
Created a chunk of size 378, which is longer than the specified 300
Created a chunk of size 381, which is longer than the specified 300
Created a chunk of size 365, which is longer than the specified 300
Created a chunk of size 377, which is longer than the specified 300
Created a chunk of size 329, which is longer than the specified 300
```

- 분할된 청크의 개수 출력해보기

```
print(len(texts))                       # 51
```

- `texts` 리스트의 첫 번째 요소 출력하기

```
# texts 리스트의 첫 번째 요소 출력하기

print(texts[0])
```

- 셀 출력

```
Semantic Search
```

`참고`

- **`CharacterTextSplitter.from_tiktoken_encoder`** **를 사용하는 경우**:
  - 텍스트: **`CharacterTextSplitter`** **에 의해서만 분할**
  - **`tiktoken 토크나이저`** = `분할된 텍스트`를 `병합`하는 데 사용

    - 즉, 분할된 텍스트가 tiktoken 토크나이저로 측정한 청크 크기보다 클 수 있음을 의미

- RecursiveCharacterTextSplitter.from_tiktoken_encoder를 사용하는 경우:
    - 분할된 텍스트가 언어 모델에서 허용하는 토큰의 청크 크기보다 크지 않도록 할 수 있음
        - 각 분할은 크기가 더 큰 경우 재귀적으로 분할함
    - tiktoken 분할기 직접 로드 → 각 분할이 청크 크기보다 작음을 보장

TokenTextSplitter

- TokenTextSplitter 클래스 사용 → 텍스트를 토큰 단위로 분할

```
from langchain_text_splitters import TokenTextSplitter

text_splitter = TokenTextSplitter(
    chunk_size=200,                              # 청크 크기를 10으로 설정
    chunk_overlap=0,                             # 청크 간 중복을 0으로 설정
)

# state_of_the_union 텍스트를 청크로 분할하기
texts = text_splitter.split_text(file)

# 분할된 텍스트의 첫 번째 청크를 출력하기
print(texts[0])
```

- 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"
```

---

spaCy

- spaCy = Python과 Cython 프로그래밍 언어로 작성된 고급 자연어 처리를 위한 오픈 소스 소프트웨어 라이브러리
- NLTK의 또 다른 대안 = spaCy tokenizer를 사용하는 것
    - 텍스트가 분할되는 방식: spaCy tokenizer에 의해 분할
    - chunk size가 측정되는 방법: 문자 수로 측정

- 사전 VS Code 터미널에 설치할 것

```
    pip install --upgrade --quiet spacy
```

- en_core_web_sm 모델 다운로드

```
    python -m spacy download en_core_web_sm --quiet
```

↓

```
    ✔ Download and installation successful
    You can now load the package via spacy.load('en_core_web_sm')
```

- appendix-keywords.txt 파일 읽기

```
```

```
# 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
```

- 일부 내용 출력해서 확인해보기

```
# 파일으로부터 읽은 내용 일부 출력해보기

print(file[:350])
```

- 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 처
```

- **`SpacyTextSplitter` 클래스 사용 → 텍스트 분할기 생성**

```
import warnings
from langchain_text_splitters import SpacyTextSplitter


# 경고 메시지를 무시하도록 설정하기
warnings.filterwarnings("ignore")

# SpacyTextSplitter 생성
text_splitter = SpacyTextSplitter(
    chunk_size=200,                          # 청크 크기를 200으로 설정
    chunk_overlap=50,                        # 청크 간 중복을 50으로 설정
)
```

- **`text_splitter` 객체의 `split_text` 메서드 사용 → `file` 텍스트 분할**

```
# text_splitter를 사용하여 file 텍스트를 분할하기
texts = text_splitter.split_text(file)

# 분할된 텍스트의 첫 번째 요소를 출력하기
print(texts[0])
```

- 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.
```

❌ `SentenceTransformers`

- **`SentenceTransformersTokenTextSplitter` = `sentence-transformer 모델`에 `특화`된 텍스트 분할기**
- 기본 동작: 사용하고자 하는 `sentence transformer 모델`의 `토큰 윈도우`에 맞게 `텍스트`를 `청크`로 `분할`하는 것

- 사전에 `VS Code` 터미널에 설치할 것

```
pip install sentence-transformers
```

- ❌ **`파이썬 버전 충돌로 해당 패키지 실제 실행 불가`**

```
from langchain_text_splitters import SentenceTransformersTokenTextSplitter

# 문장 분할기 생성
splitter = SentenceTransformersTokenTextSplitter(
    chunk_size=200,
    chunk_overlap=0,
)

# 분할할 텍스트
text = "The quick brown fox jumps over the lazy dog. This is another sentence. And a third one."

# 텍스트 분할
chunks = splitter.split_text(text)

# 결과 출력
for i, chunk in enumerate(chunks):
    print(f"청크 {i+1}: {chunk}")
```

```
from langchain_text_splitters import SentenceTransformersTokenTextSplitter

# 문장 분할기 생성
splitter = SentenceTransformersTokenTextSplitter(
    chunk_size=200,
    chunk_overlap=0,                        # 청크 간 중복 = 0 으로 설정하기
    )
```

```
# 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

# 파일로부터 읽은 내용을 일부 출력하기
print(file[:350])
```

- 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 처
```

- `file` 변수에 담긴 텍스트의 토큰 개수 세는 코드

```
# 토큰 개수 세기

count_start_and_stop_tokens = 2                    # 시작과 종료 토큰의 개수= 2로 설정
```

```
# 텍스트의 토큰 개수에서 시작과 종료 토큰의 개수 제외한 후 출력하기
text_token_count = splitter.count_tokens(
    text=file) - count_start_and_stop_tokens
print(text_token_count)                                  # 계산된 텍스트 토큰 개수 출력 = 7686
```

- `splitter.split_text()` 함수 사용 → `text_to_split` 변수에 저장된 텍스트를 `chunk` 단위로 분할하기

```
# 텍스트를 청크로 분할하기

text_chunks = splitter.split_text(text=file)
```

- 첫 번째 청크를 출력해 내용 확인하기

```
# 0번째 청크 출력하기

print(text_chunks[1])                                    # 분할된 텍스트 청크 중 두 번째 청크 출력함
```

- 셀 출력

```
. 이를 통해 컴퓨터가 텍스트를 이해하고 처리할 수 [UNK] 합니다. [UNK] : " 사과 " 라는 단어를 [ 0. 65, - 0. 23, 0. 17 ] 과 [UNK] 벡터
```

`NLTK`

- **`Natural Language Toolkit (NLTK)`**
  - `Python` 프로그래밍 언어로 작성된 `영어 자연어 처리(NLP)`를 위한 라이브러리와 프로그램 모음
  - **`텍스트 데이터의 전처리, 토큰화, 형태소 분석, 품사 태깅 등 다양한 NLP 작업 수행 가능`**

- 단순히 `"\n\n"`으로 분할하는 대신, **`NLTK tokenizers`**를 기반으로 텍스트를 분할하는 데 사용
  - **`텍스트 분할`** **방법:** **`NLTK tokenizer`** **에 의해 분할**
  - **`chunk 크기`** **측정 방법:** **`문자 수`** **에 의해 측정**

- 사전에 `VS Code` 터미널에 설치할 것

```
    pip install -qU nltk
```

```
# 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

# 파일로부터 읽은 내용을 일부 출력하기
print(file[:350])
```

- 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 처
```

- **`NLTKTextSplitter`** **클래스 사용 → 텍스트 분할기 생성**

- **`chunk_size` 매개변수를 1000으로 설정 → `텍스트`를 `최대 1000자 단위`로 `분할`하도록 지정**

```
from langchain_text_splitters import NLTKTextSplitter

text_splitter = NLTKTextSplitter(
    chunk_size=200,                                    # 청크 크기 = 200
    chunk_overlap=0,                                   # 청크 간 중복 = 0
)
```

- **`text_splitter` 객체의 `split_text 메서드`를 사용하여 `file 텍스트`를 `분할`**

```
# text_splitter 사용 → file 텍스트 분할
texts = text_splitter.split_text(file)

# 분할된 텍스트의 첫 번째 요소를 출력
print(texts[0])
```

- 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.
```

`KoNLPy의 Kkma 분석기를 사용한 한국어 토큰 분할`

- **`한국어 텍스트` : `KoNLPY`에 `Kkma`(Korean Knowledge Morpheme Analyzer)라는 형태소 분석기가 포함되어 있음**
- **`Kkma`**
  - `한국어 텍스트`에 대한 상세한 **`형태소 분석`을 제공**
  - `문장`을 `단어`로, `단어`를 **`각각의 형태소`로 분해 → 각 토큰에 대한 품사를 식별**
  - 텍스트 블록을 `개별 문장`으로 `분할` 가능 → **`긴 텍스트 처리에 특히 유용`**
- **`Kkma 사용시 주의사항`**
  - 장점: **`상세한 분석`** → *정밀성이 `처리 속도`에 영향을 미칠 수 있음*
  - **`Kkma`는 신속한 텍스트 처리 < `분석적 깊이 우선시`되는 애플리케이션에 가장 적합**

- 사전에 `VS Code`에 설치할 것

```
pip install -qU konlpy
```

*

- **`KoNLPy`**
  - **`한국어 자연어 처리를 위한 파이썬 패키지`**
  - **`형태소 분석`, `품사 태깅`, `구문 분석` 등의 기능 제공**

```
# 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

# 파일로부터 읽은 내용을 일부 출력하기
print(file[:350])
```

- 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 처
```

- **`KonlpyTextSplitter` 사용 → 한국어 텍스트 분할하기**

```
from langchain_text_splitters import KonlpyTextSplitter

# KonlpyTextSplitter를 사용하여 텍스트 분할기 객체 생성하기
text_splitter = KonlpyTextSplitter()
```

```
from langchain_text_splitters import KonlpyTextSplitter

text_splitter = KonlpyTextSplitter()  # 기본 초기화 (Kkma 호출 테스트)

test_text = "이 문장을 한국어로 분할해 보세요. Konlpy가 제대로 동작하나요?"
chunks = text_splitter.split_text(test_text)
print(chunks)  # 형태소/문장 단위 리스트 출력
```

```
['이 문장을 한국어로 분할해 보세요.\n\nKonlpy가 제대로 동작하나요?']
```

```
# Java Runtime 설치
# brew install openjdk

# KonlpyTextSplitter 임포트
from langchain_text_splitters import KonlpyTextSplitter

# KonlpyTextSplitter를 사용하여 텍스트 분할기 객체 생성하기
text_splitter = KonlpyTextSplitter()
```

- **`text_splitter` 사용 → `file`를 문장 단위로 분할**

```
# 한국어 문서를 문장 단위로 분할하기
texts = text_splitter.split_text(file)

# 분할된 문장 중 첫 번째 문장을 출력해보기
print(texts[0])
```

- 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", " 화 성" 등과 같이 관련된 행성에 대한 정보를 반환합니다.

연관 키워드: 자연어 처리, 검색 알고리즘, 데이터 마이닝 Embedding 정의: 임베딩은 단어나 문장 같은 텍스트 데이터를 저차원의 연속적인 벡터로 변환하

이를 통해 컴퓨터가 텍스트를 이해하고 처리할 수 있게 합니다.

예시: " 사과" 라는 단어를 [0.65, -0.23, 0.17] 과 같은 벡터로 표현합니다.

연관 키워드: 자연어 처리, 벡터화, 딥 러닝 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.

연관 키워드: 토큰 화, 자연어 처리, 구 문 분석 Tokenizer 정의: 토크 나이저는 텍스트 데이터를 토큰으로 분할하는 도구입니다.

이는 자연어 처리에서 데이터를 전처리하는 데 사용됩니다.
```

예시: "I love programming." 이라는 문장을 ["I", "love", "programming", "."]으로 분할합니다.

연관 키워드: 토큰 화, 자연어 처리, 구 문 분석 VectorStore 정의: 벡터 스토어는 벡터 형식으로 변환된 데이터를 저장하는 시스템 입니

다. 이는 검색, 분류 및 기타 데이터 분석 작업에 사용됩니다.

예시: 단어 임 베 딩 벡터들을 데이터베이스에 저장하여 빠르게 접근할 수 있습니다.

연관 키워드: 임 베 딩, 데이터베이스, 벡터화 SQL 정의: SQL(Structured Query Language) 은 데이터베이스에서 데이터를 관리하기 위한 프로그래

데이터 조회, 수정, 삽입, 삭제 등 다양한 작업을 수행할 수 있습니다.

예시: SELECT * FROM users WHERE age > 18; 은 18세 이상의 사용자 정보를 조회합니다.

연관 키워드: 데이터베이스, 쿼 리, 데이터 관리 CSV 정의: CSV(Comma-Separated Values) 는 데이터를 저장하는 파일 형식으로, 각 데이터 값은 쉼

표 형태의 데이터를 간단하게 저장하고 교환할 때 사용됩니다.

예시: 이름, 나이, 직업이라는 헤더를 가진 CSV 파일에는 홍길동, 30, 개발자와 같은 데이터가 포함될 수 있습니다.

연관 키워드: 데이터 형식, 파일 처리, 데이터 교환 JSON 정의: JSON(JavaScript Object Notation) 은 경량의 데이터 교환 형식으로, 사람과 기

예시: {" 이름": " 홍길동", " 나이": 30, " 직업": " 개발자"} 는 JSON 형식의 데이터 입니

다. 연관 키워드: 데이터 교환, 웹 개발, API Transformer 정의: 트랜스포머는 자연어 처리에서 사용되는 딥 러닝 모델의 한 유형으로, 주로 번역, 요

이는 Attention 메커니즘을 기반으로 합니다.

예시: 구 글 번역기는 트랜스포머 모델을 사용하여 다양한 언어 간의 번역을 수행합니다.

연관 키워드: 딥 러닝, 자연어 처리, Attention HuggingFace 정의: HuggingFace는 자연어 처리를 위한 다양한 사전 훈련된 모델과 도구를 제공하는

이는 연구자와 개발자들이 쉽게 NLP 작업을 수행할 수 있도록 돕습니다.

예시: HuggingFace의 Transformers 라이브러리를 사용하여 감정 분석, 텍스트 생성 등의 작업을 수행할 수 있습니다.

연관 키워드: 자연어 처리, 딥 러닝, 라이브러리 Digital Transformation 정의: 디지털 변환은 기술을 활용하여 기업의 서비스, 문화, 운영을 혁신하

이는 비즈니스 모델을 개선하고 디지털 기술을 통해 경쟁력을 높이는 데 중점을 둡니다.

예시: 기업이 클라우드 컴퓨팅을 도입하여 데이터 저장과 처리를 혁신하는 것은 디지털 변환의 예입니다.

연관 키워드: 혁신, 기술, 비즈니스 모델 Crawling 정의: 크롤링은 자동화된 방식으로 웹 페이지를 방문하여 데이터를 수집하는 과정입니다.

이는 검색 엔진 최적화나 데이터 분석에 자주 사용됩니다.

예시: 구 글 검색 엔진이 인터넷 상의 웹사이트를 방문 하여 콘텐츠를 수집하고 인 덱싱하는 것이 크롤링입니다.

연관 키워드: 데이터 수집, 웹 스크 래핑, 검색 엔진 Word2Vec 정의: Word2Vec 은 단어를 벡터 공간에 매 핑 하여 단어 간의 의미적 관계를 나타내는

이는 단어의 문맥적 유사성을 기반으로 벡터를 생성합니다.

예시: Word2Vec 모델에서 " 왕" 과 " 여 왕" 은 서로 가까운 위치에 벡터로 표현됩니다.

연관 키워드: 자연어 처리, 임 베 딩, 의미론적 유사성 LLM (Large Language Model) 정의: LLM은 대규모의 텍스트 데이터로 훈련된 큰 규모의 언어

이러한 모델은 다양한 자연어 이해 및 생성 작업에 사용됩니다.

예시: OpenAI의 GPT 시리즈는 대표적인 대규모 언어 모델입니다.

연관 키워드: 자연어 처리, 딥 러닝, 텍스트 생성 FAISS (Facebook AI Similarity Search) 정의: FAISS는 페이스 북에서 개발한 고속 유사성 검

예시: 수백만 개의 이미지 벡터 중에서 비슷한 이미지를 빠르게 찾는 데 FAISS가 사용될 수 있습니다.

연관 키워드: 벡터 검색, 머신 러닝, 데이터베이스 최적화 Open Source 정의: 오픈 소스는 소스 코드가 공개되어 누구나 자유롭게 사용, 수정, 배포할 수

이는 협업과 혁신을 촉진하는 데 중요한 역할을 합니다.

예시: 리눅스 운영 체제는 대표적인 오픈 소스 프로젝트입니다.

연관 키워드: 소프트웨어 개발, 커뮤니티, 기술 협업 Structured Data 정의: 구조화된 데이터는 정해진 형식이나 스키마에 따라 조직된 데이터입니다.

이는 데이터베이스, 스프레드 시트 등에서 쉽게 검색하고 분석할 수 있습니다.

예시: 관계 형 데이터베이스에 저장된 고객 정보 테이블은 구조화된 데이터의 예입니다.

연관 키워드: 데이터베이스, 데이터 분석, 데이터 모델링 Parser 정의: 파서는 주어진 데이터(문자열, 파일 등)를 분석하여 구조화된 형태로 변환하는 도

이는 프로그래밍 언어의 구문 분석이나 파일 데이터 처리에 사용됩니다.

예시: HTML 문서를 구 문 분석하여 웹 페이지의 DOM 구조를 생성하는 것은 파싱의 한 예입니다.

연관 키워드: 구 문 분석, 컴파일러, 데이터 처리 TF-IDF (Term Frequency-Inverse Document Frequency) 정의: TF-IDF는 문서 내에서 단어

이는 문서 내 단어의 빈도와 전체 문서 집합에서 그 단어의 희소성을 고려합니다.

예시: 많은 문서에서 자주 등장하지 않는 단어는 높은 TF-IDF 값을 가집니다.

연관 키워드: 자연어 처리, 정보 검색, 데이터 마이닝 Deep Learning 정의: 딥 러닝은 인공 신경망을 이용하여 복잡한 문제를 해결하는 머신 러닝의 한

이는 데이터에서 고수준의 표현을 학습하는 데 중점을 둡니다.

예시: 이미지 인식, 음성 인식, 자연어 처리 등에서 딥 러닝 모델이 활용됩니다.

연관 키워드: 인공 신경망, 머신 러닝, 데이터 분석 Schema 정의: 스키마는 데이터베이스나 파일의 구조를 정의하는 것으로, 데이터가 어떻게 저장되고 조직

`Hugging Face tokenizer`

- **`Hugging Face` = 다양한 토크나이저 제공**
- 다양한 토크나이저 中 하나인 **`GPT2TokenizerFast`** 사용해 텍스트의 토큰 길이 계산하기
  - **`텍스트 분할 방식` : 전달된 문자 단위로 분할**
  - **`청크 크기 측정 방식`: `Hugging Face 토크나이저` 에 의해 게산된 토큰 수를 기준으로 함**
- 사전에 `VS Code` 터미널에 설치할 것

```
pip install transformers

pip install --upgrade jupyter ipywidgets    # 오류 발생 시
```

- **`GPT2TokenizerFast` 클래스 사용 → `tokenizer` 객체 생성하기**
- **`from_pretrained` 메서드 호출 → 사전 학습된 `gpt2` `토크나이저 모델` 로드**

```
# transformers 라이브러리 임포트
from transformers import GPT2TokenizerFast

# GPT-2 모델의 토크나이저 불러오기
hf_tokenizer = GPT2TokenizerFast.from_pretrained("gpt2")
```

- 샘플 텍스트 확인하기

```
# 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
```

```
# 파일로부터 읽은 내용을 일부 출력하기
print(file[:350])
```

- 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 처
```

- `from_huggingface_tokenizer` 메서드 → **허깅페이스 토크나이저 (`tokenizer`) 사용 → 텍스트 분할기 초기화**

```
# 허깅페이스 토크나이저 사용 → CharacterTextSplitter 객체 생성하기
from langchain_text_splitters import CharacterTextSplitter

text_splitter = CharacterTextSplitter.from_huggingface_tokenizer(
    hf_tokenizer,
    chunk_size=300,
    chunk_overlap=50,
)


# state_of_the_union 텍스트를 분할하여 texts 변수에 저장하기
texts = text_splitter.split_text(file)
```

- 셀 출력

```
Created a chunk of size 358, which is longer than the specified 300
Created a chunk of size 315, which is longer than the specified 300
Created a chunk of size 305, which is longer than the specified 300
Created a chunk of size 366, which is longer than the specified 300
Created a chunk of size 330, which is longer than the specified 300
Created a chunk of size 351, which is longer than the specified 300
Created a chunk of size 378, which is longer than the specified 300
Created a chunk of size 361, which is longer than the specified 300
Created a chunk of size 350, which is longer than the specified 300
Created a chunk of size 362, which is longer than the specified 300
Created a chunk of size 335, which is longer than the specified 300
Created a chunk of size 353, which is longer than the specified 300
Created a chunk of size 358, which is longer than the specified 300
Created a chunk of size 336, which is longer than the specified 300
Created a chunk of size 324, which is longer than the specified 300
Created a chunk of size 337, which is longer than the specified 300
Created a chunk of size 307, which is longer than the specified 300
Created a chunk of size 361, which is longer than the specified 300
Created a chunk of size 354, which is longer than the specified 300
Created a chunk of size 378, which is longer than the specified 300
Created a chunk of size 381, which is longer than the specified 300
Created a chunk of size 365, which is longer than the specified 300
Created a chunk of size 377, which is longer than the specified 300
Created a chunk of size 329, which is longer than the specified 300
```

- 첫 번째 요소의 분할 결과 확인하기

```
# texts 리스트의 1 번째 요소 출력해보기

print(texts[1])
```

- 셀 출력

```
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
```

- *next:* ***`시멘틱 청커 (SemanticChunker)`***
