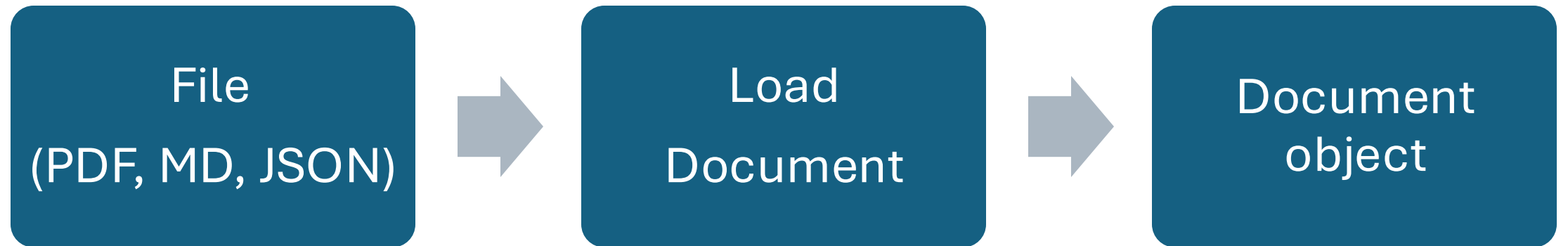


LangChain Document loader

신재익

2025.09.19

Load document



BaseLoader

https://python.langchain.com/api_reference/core/document_loaders/langchain_core.document_loaders.base.BaseLoader.html

- class langchain_core.document_loaders.base.BaseLoader
- ❖ async alazy_load() → AsyncIterator[Document]
- A lazy loader for Documents.
- ❖ aload() → list[Document]
- Load data into Document objects.
- ❖ lazy_load() → Iterator[Document]
- A lazy loader for Documents.
- ❖ load() → list[Document]
- Load data into Document objects.
- ❖ load_and_split(text_splitter: TextSplitter | None = None,) → list[Document]
- Load Documents and split into chunks. Chunks are returned as Documents.
- Do not override this method. It should be considered to be deprecated!
- Parameters:
- text_splitter (Optional[TextSplitter]) – TextSplitter instance to use for splitting documents. Defaults to RecursiveCharacterTextSplitter.

lazy: 필요할때 메모리에 하나씩

a(sync): 비동기 방식. 즉시 실행이 아니라 await를 사용해 실행

Document

https://python.langchain.com/api_reference/core/documents/langchain_core.documents.base.Document.html

❖ Example

```
from langchain_core.documents import Document
document = Document(
    page_content="Hello, world!",
    metadata={"source": "https://example.com"}
)
```

- class langchain_core.documents.base.Document

- ❖ param metadata: dict [Optional]

- Arbitrary metadata associated with the content.

- ❖ param page_content: str [Required]

- String text.

Document loaders

- https://python.langchain.com/docs/how_to/#document-loaders
- load PDF files
- load web pages
- load CSV data
- load data from a directory
- load HTML data
- load JSON data
- load Markdown data
- load Microsoft Office data
- write a custom document loader

Integration for Document loaders

- https://python.langchain.com/docs/integrations/document_loaders/
- https://python.langchain.com/api_reference/core/documents/langchain_core.documents.base.Document.html
- https://python.langchain.com/docs/integrations/document_loaders/#all-document-loaders

그외 문서 로더

- Upstage 에서 제공하는 문서 로더
- <https://python.langchain.com/docs/integrations/providers/upstage/#document-loader>
- OCR, 제목, 단락, 표 이미지 자동 인식 및 추출
- LlamaParse : LlamaIndex 에서 제공하는 문서 로더
- 멀티모달 파싱
- langchain_teddynote 에 추가된 문서 로더
- HWP loader
- https://github.com/teddylee777/langchain-teddynote/blob/main/langchain_teddynote/document_loaders/hwp.py

문서 로더

```
from langchain_community.document_loaders import  
PyPDFLoader
```

```
# 로더 설정
```

```
loader = PyPDFLoader(FILE_PATH)
```

```
# PDF 로더
```

```
docs = loader.load()
```


문자 분할기 사용

문자 분할기 설정

```
text_splitter = RecursiveCharacterTextSplitter(chunk_size=200,  
chunk_overlap=0)
```

예제 파일 경로

```
FILE_PATH = "./data/SPRI_AI_Brief_2023년12월호_F.pdf"
```

로더 설정

```
loader = PyPDFLoader(FILE_PATH)
```

문서 분할

```
split_docs = loader.load_and_split(text_splitter=text_splitter)
```

lazy_load()

lazy_load()

- generator 방식으로 문서를 로드합니다.

loader.lazy_load()

generator 방식으로 문서 로드

for doc in loader.lazy_load():

print(doc.metadata)

비동기 방식

- # 문서를 async 방식으로 로드
- adocs = loader.aload()
- # 문서 로드
- await adocs

LangChain Text splitter

신재익

2025.09.05

Split Text



Text splitters

- https://python.langchain.com/docs/how_to/#text-splitters
- recursively split text
- split HTML
- split by character
- split code
- split Markdown by headers
- recursively split JSON
- split text into semantic chunks
- split by tokens

- https://python.langchain.com/docs/concepts/text_splitters/
- https://python.langchain.com/api_reference/text_splitters/index.html#

TextSplitter

- https://python.langchain.com/api_reference/text_splitters/base/langchain_text_splitters.base.TextSplitter.html

```
class langchain_text_splitters.base.TextSplitter(  
    chunk_size: int = 4000,  
    chunk_overlap: int = 200,  
    length_function: ~typing.Callable[[str], int] = <built-in function len>,  
    keep_separator: bool | ~typing.Literal['start', 'end'] = False,  
    add_start_index: bool = False,  
    strip_whitespace: bool = True)
```

CharacterTextSplitter

- https://python.langchain.com/api_reference/text_splitters/character/langchain_text_splitters.character.CharacterTextSplitter.html

```
class langchain_text_splitters.character.CharacterTextSplitter(  
    separator: str = '\n\n',  
    is_separator_regex: bool = False,  
    **kwargs: Any, )
```



Chunk size


```
from langchain_text_splitters import CharacterTextSplitter
```


```
# CharacterTextSplitter를 사용하여 텍스트를 청크(chunk)로 분할하는 코드
text_splitter = CharacterTextSplitter(
# 텍스트를 분할할 때 사용할 구분자를 지정합니다. 기본값은 "\n\n"입니다.
separator="\n\n",
# 분할된 텍스트 청크의 최대 크기를 지정합니다 (문자 수).
chunk_size=210,
# 분할된 텍스트 청크 간의 중복되는 문자 수를 지정합니다.
chunk_overlap=0,
# 텍스트의 길이를 계산하는 함수를 지정합니다.
length_function=len,
)
```

<https://medium.com/@krishnahariharan/langchains-character-text-splitter-in-depth-explanation-5b0bf743121c>

<https://chunkviz.up.railway.app/>

Splitter: Character Splitter  

Chunk Size: 210 

Chunk Overlap: 0 

Total Characters: 5733
Number of chunks: 28
Average chunk size: 204.8

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 합니다.

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 처리, 구문 분석



Tokenizer


정의: 토큰라이저는 텍스트 데이터를 토큰으로 분할하는 도구입니다. 이는 자연어 처리에서 데이터를 전처리하는 데 사용됩니다.


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

연관키워드: 토큰화, 자연어 처리, 구문 분석

Chunk overlap

Splitter: Character Splitter  

Chunk Size: 210 

Chunk Overlap: 20 

Total Characters: 6333
Number of chunks: 31
Average chunk size: 204.3

Upload .txt

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 합니다.

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 처리, 구문 분석

Tokenizer

정의: 토크나이저는 텍스트 데이터를 토큰으로 분할하는 도구입니다. 이는 자연어 처리에서 데이터를 전처리하는 데 사용됩니다.

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

연관키워드: 토큰화, 자연어 처리, 구문 분석

<https://medium.com/@krishnahariharan/langchains-character-text-splitter-in-depth-explanation-5b0bf743121c>

<https://chunkviz.up.railway.app/>

파일을 문자 단위로 분할

- Create documents from a list of texts.

```
create_documents(  
    texts: list[str],  
    metadatas: list[dict[Any,  
Any]] | None = None,  
) → list[Document]
```

```
metadatas = [{"document": 1}, {"document": 2}, ]  
# 문서에 대한 메타데이터 리스트를 정의합니다.
```

```
documents = text_splitter.create_documents(  
    [ file, file, ],  
    # 분할할 텍스트 데이터를 리스트로 전달합니다.  
    metadatas=metadatas,  
    # 각 문서에 해당하는 메타데이터를 전달합니다.  
)  
print(documents[0])  
# 분할된 문서 중 첫 번째 문서를 출력합니다.
```

재귀적으로 분할

- https://python.langchain.com/api_reference/text_splitters/character/langchain_text_splitters.character.RecursiveCharacterTextSplitter.html

```
class
langchain_text_splitters.character.RecursiveCharacterTextSplitter(
separators: list[str] | None = None,
keep_separator: bool | Literal['start', 'end'] = True,
is_separator_regex: bool = False,
**kwargs: Any, )
```


텍스트를 분할, 문서를 분할

- Split the input text into smaller chunks based on predefined separators.
- `split_text(text: str,) → list[str]`
- Parameters: text (str) – The input text to be split.
- Returns: A list of text chunks obtained after splitting.
- Split documents.
- `split_documents(documents: Iterable[Document],) → list[Document]`
- Parameters: documents (Iterable[Document])
- Return type: list[Document]

Tiktoken 인코더 기반의 텍스트 분할

- https://python.langchain.com/api_reference/text_splitters/character/langchain_text_splitters.character.CharacterTextSplitter.html#langchain_text_splitters.character.CharacterTextSplitter.from_tiktoken_encoder
- Text splitter that uses tiktoken encoder to count length.
classmethod from_tiktoken_encoder(
encoding_name: str = 'gpt2',
model_name: str | None = None,
allowed_special: Literal['all'] | AbstractSet[str] = {},
disallowed_special: Literal['all'] | Collection[str] = 'all',
**kwargs: Any,
) → Self

Tiktoken 인코더 기반의 텍스트 분할

```
from langchain_text_splitters import CharacterTextSplitter

text_splitter = CharacterTextSplitter.from_tiktoken_encoder(
    # 청크 크기를 300으로 설정합니다.
    chunk_size=300,
    # 청크 간 중복되는 부분이 없도록 설정합니다.
    chunk_overlap=0,
)
# file 텍스트를 청크 단위로 분할합니다.
texts = text_splitter.split_text(file)
```

텍스트를 토큰단위로 분할

- https://python.langchain.com/api_reference/text_splitters/base/langchain_text_splitters.base.TokenTextSplitter.html
- Splitting text to tokens using model tokenizer.

```
class langchain_text_splitters.base.TokenTextSplitter(  
    encoding_name: str = 'gpt2',  
    model_name: str | None = None,  
    allowed_special: Literal['all'] | AbstractSet[str] = {},  
    disallowed_special: Literal['all'] | Collection[str] = 'all',  
    **kwargs: Any, )
```

텍스트를 토큰단위로 분할

```
from langchain_text_splitters import TokenTextSplitter
```

```
text_splitter = TokenTextSplitter(  
    chunk_size=300, # 청크 크기를 10으로 설정합니다.  
    chunk_overlap=0, # 청크 간 중복을 0으로 설정합니다.  
)
```

```
# state_of_the_union 텍스트를 청크로 분할합니다.  
texts = text_splitter.split_text(file)  
print(texts[0]) # 분할된 텍스트의 첫 번째 청크를 출력합니다.
```

SpacyTextSplitter

- <https://spacy.io/>
- <https://python.langchain.com/docs/integrations/providers/spacy/>
- https://python.langchain.com/docs/how_to/split_by_token/#spacy
[y](#)
- https://python.langchain.com/api_reference/text_splitters/spacy/langchain_text_splitters.spacy.SpacyTextSplitter.html

SentenceTransformersTokenTextSplitter

- https://python.langchain.com/docs/how_to/split_by_token/#sentence_transformers
- https://python.langchain.com/api_reference/text_splitters/sentence_transformers/langchain_text_splitters.sentence_transformers.SentenceTransformersTokenTextSplitter.html#
- <https://sbert.net/>

NLTKTextSplitter : 영어자연어처리기 이용

- <https://www.nltk.org/index.html>
- https://python.langchain.com/docs/how_to/split_by_token/#nltk
- https://python.langchain.com/api_reference/text_splitters/nltk/langchain_text_splitters.nltk.NLTKTextSplitter.html#langchain_text_splitters.nltk.NLTKTextSplitter

KoNLPy : 한국어용 토크나이저 이용

- <https://konlpy.org/>
- https://python.langchain.com/api_reference/text_splitters/konlpy/langchain_text_splitters.konlpy.KonlpyTextSplitter.html
- https://python.langchain.com/docs/how_to/split_by_token/#konlpy

Hugging Face의 다양한 토크나이저 이용

- <https://huggingface.co/docs/tokenizers/index>
- <https://huggingface.co/Ransaka/gpt2-tokenizer-fast>
- https://python.langchain.com/docs/how_to/split_by_token/#hugging-face-tokenizer
- https://python.langchain.com/api_reference/text_splitters/character/langchain_text_splitters.character.CharacterTextSplitter.html#langchain_text_splitters.character.CharacterTextSplitter.from_huggingface_tokenizer
- https://python.langchain.com/api_reference/text_splitters/character/langchain_text_splitters.character.RecursiveCharacterTextSplitter.html#langchain_text_splitters.character.RecursiveCharacterTextSplitter.from_huggingface_tokenizer

Semantic Chunker : 의미 단위로 분할

- https://python.langchain.com/docs/how_to/semantic-chunker/
- https://python.langchain.com/api_reference/experimental/text_splitter/langchain_experimental.text_splitter.SemanticChunker.html

- Split the text based on semantic similarity.

```
class langchain_experimental.text_splitter.SemanticChunker(
```

```
    embeddings: Embeddings,
```

```
    buffer_size: int = 1,
```

```
    add_start_index: bool = False,
```

```
    breakpoint_threshold_type: Literal['percentile', 'standard_deviation', 'interquartile', 'gradient'] = 'percentile',
```

```
    breakpoint_threshold_amount: float | None = None,
```

```
    number_of_chunks: int | None = None,
```

```
    sentence_split_regex: str = '(?<=[.?!])\\s+',
```

```
    min_chunk_size: int | None = None,)
```

Google Gemini embedding

- https://python.langchain.com/docs/integrations/text_embedding/google_generative_ai/
- <https://ai.google.dev/gemini-api/docs/embeddings?hl=ko#model-versions>

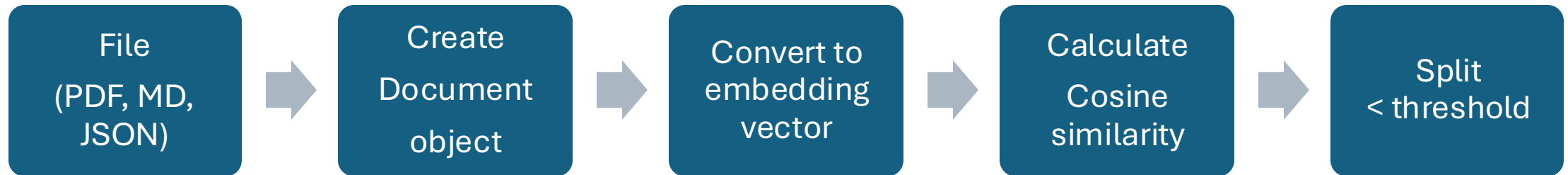
```
from langchain_google_genai import  
GoogleGenerativeAIEmbeddings
```

```
embeddings =  
GoogleGenerativeAIEmbeddings(model="models/gemini-  
embedding-001")  
vector = embeddings.embed_query("hello, world!")
```

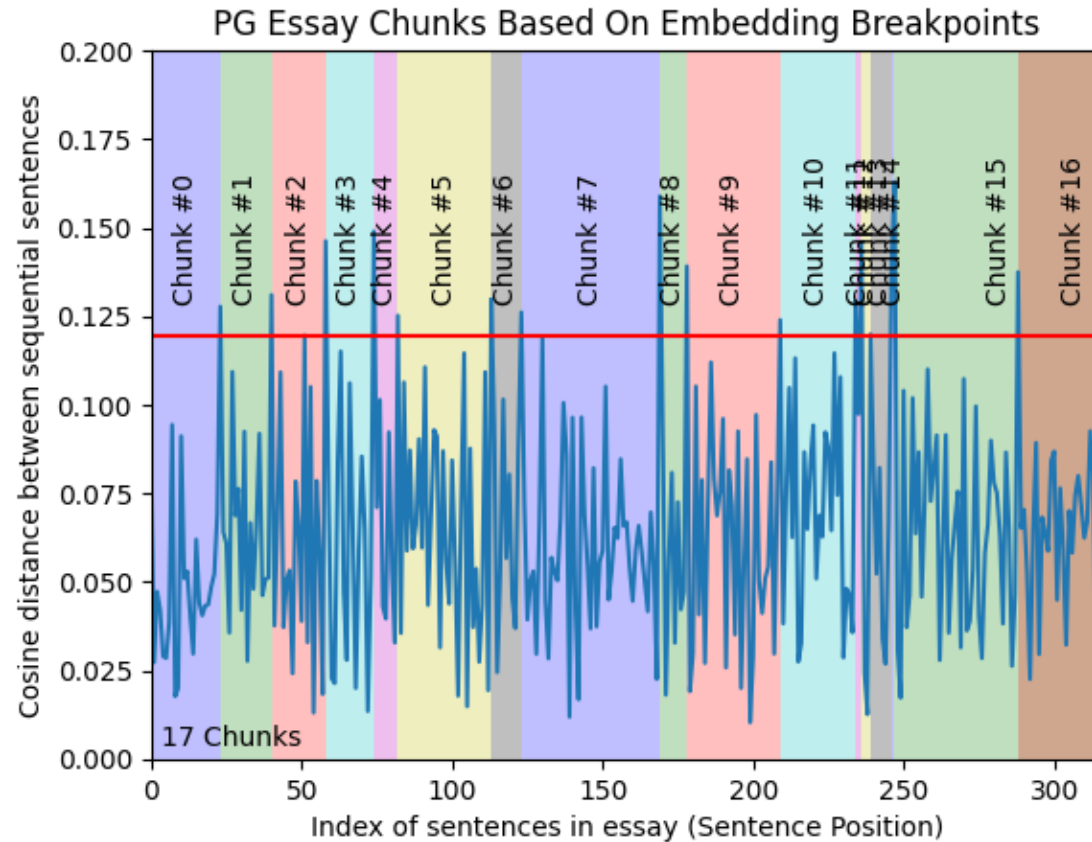
Embedding Gemma

- <https://ai.google.dev/gemma/docs/embeddinggemma/inference-embeddinggemma-with-sentence-transformers>
- <https://huggingface.co/google/embeddinggemma-300m>
- 300m parameters for Gemma 3
- Sentence transformer

Split Text using Semantic Chunker



Calculate cosine similarity



https://github.com/FullStackRetrieval-com/RetrievalTutorials/blob/main/tutorials/LevelsOfTextSplitting/5_Levels_Of_Text_Splitting.ipynb