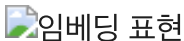
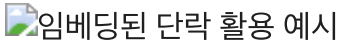


- 출처: LangChain 공식 문서 또는 해당 교재명
- 원본 URL: <https://smith.langchain.com/hub/teddynote/summary-stuff-documents>

✓ CH08 임베딩 (Embedding)

- 임베딩 = **RAG (Retrieval-Augmented Generation)** 시스템의 세 번째 단계
 - 문서분할 단계에서 생성된 문서 단위들을 기계가 이해할 수 있는 수치적 형태로 변환하는 과정
 - RAG 시스템의 핵심적인 부분 중 하나: 문서의 의미를 벡터 (숫자의 배열) 형태로 표현 → 사용자가 입력한 질문 (Query)에 대하여 DB에 저장한 문서 조각 / 단락 (Chunk)을 검색하여 가져올 때 유사도 계산 시 활용 가능
- 임베딩의 필요성
 - 의미 이해: 자연 언어 (매우 복잡, 다양한 의미 내포) → 정량화된 형태로 변화 → 컴퓨터가 문서의 내용, 의미 더 잘 이해하고 처리 가능
 - 정보 검색 향상
 - 수치화된 벡터 형태로의 변환 = 문서 간 유사성 계산의 필수
 - 관련된 문서 검색 or 질문에 가장 적합한 문서 찾는 작업 용이
- 예시: 임베딩 = 문장을 수치 표현으로 변경?

- 임베딩된 단락 활용 예시

 - 1번 단락: [0.1, 0.5, 0.9, ..., 0.1, 0.2]
 - 2번 단락: [0.7, 0.1, 0.3, ..., 0.5, 0.6]
 - 3번 단락: [0.9, 0.4, 0.5, ..., 0.4, 0.3]
- 질문: 시장조사기관 IDC가 예측한 AI 소프트웨어 시장의 연평균 성장률은 어떻게 되나요?
 - [0.1, 0.5, 0.9, ..., 0.2, 0.4]
- 유사도 계산 예시
 - 1번: 80% → ✓ 선택

- 2번: 30%
- 3번: 25%



코드

- 코드 예시

```
from langchain_openai import OpenAIEmbeddings

# 단계 3: 임베딩(Embedding) 생성
embeddings = OpenAIEmbeddings()
```



참고

- [Embedding](#)
- [LangChain Text Embeddings](#)



1. OpenAIEmbeddings

- 문서 임베딩 = 문서의 내용을 수치적인 벡터로 변환하는 과정임
 - 문서의 의미를 수치화 하고, 다양한 자연어 처리 작업에 활용 가능
 - 대표적인 사전 학습된 언어 모델: BERT, GPT → 문맥적 정보를 포착하여 문서의 의미를 인코딩
- 문서 임베딩 과정
 - 토큰화된 문서 → 모델에 입력 → 임베딩 벡터 생성 → 이를 평균 → 전체 문서의 벡터 생성
 - 문서 분류, 감성 분석, 문서 간 유사도 계산 등에 활용
- [더 알아보기](#)



1) 설정

- 먼저 **langchain-openai** 설치 → **필요한 환경 변수** 를 **설정**

```
# API 키를 환경변수로 관리하기 위한 설정 파일
from dotenv import load_dotenv
```

```
# API 키 정보 로드
load_dotenv()
```

```
# True
```

```
from langsmith import Client
from langsmith import traceable

import os

# LangSmith 환경 변수 확인

print("\n--- LangSmith 환경 변수 확인 ---")
langchain_tracing_v2 = os.getenv('LANGCHAIN_TRACING_V2')
langchain_project = os.getenv('LANGCHAIN_PROJECT')
langchain_api_key_status = "설정됨" if


if langchain_tracing_v2 == "true" and
    print(f"✅ LangSmith 추적 활성화됨 (")
    print(f"✅ LangSmith 프로젝트: '{langchain_project}'")
    print(f"✅ LangSmith API Key: '{langchain_api_key_status}'")
    print("    -> 이제 LangSmith 대시보드에")
else:
    print("❌ LangSmith 추적이 완전히 활성화되지 않음")
    if langchain_tracing_v2 != "true":
        print(f"    - LANGCHAIN_TRACING_V2: '{langchain_tracing_v2}'")
    if not os.getenv('LANGCHAIN_API_KEY'):
        print("    - LANGCHAIN_API_KEY가 설정되지 않음")
    if not langchain_project:
        print("    - LANGCHAIN_PROJECT가 설정되지 않음")
```

"@traceable" 주석은 허용되지 않습니다. 허용되는 값은 다음과 같습니다. [@param, @title, @markdown]

- 셀 출력

```
--- LangSmith 환경 변수 확인 ---
✅ LangSmith 추적 활성화됨 (LANGCHAIN_TRACING_V2='true')
✅ LangSmith 프로젝트: 'LangChain-prantice'
✅ LangSmith API Key: 설정됨
-> 이제 LangSmith 대시보드에서 이 프로젝트를 확인해 보세요.
```

- 지원되는 모델 목록

 지원되는 모델 목록

```
from langchain_openai import OpenAIEmbeddings

# OpenAI의 "text-embedding-3-small" 모델 → 임베딩 생성하기
embeddings = OpenAIEmbeddings(model="text-embedding-3-small")
```

```
text = "임베딩 테스트를 하기 위한 샘플 문장입니다."
```

2) 쿼리 임베딩

- `embeddings.embed_query(text)` = 주어진 텍스트를 임베딩 벡터로 변환하는 함수
 - 텍스트를 벡터 공간에 매핑하여 의미적으로 유사한 텍스트를 찾거나 텍스트 간의 유사도를 계산하는 데 사용

```
# 텍스트 임베딩해 쿼리 결과 생성하기
```

```
query_result = embeddings.embed_query(text)
```

- `query_result[:5]` = `query_result` 리스트의 처음 5개 요소를 슬라이싱(slicing)하여 선택

```
# 쿼리 결과 처음 5개 항목 선택하기
```

```
query_result[:5]
```

- 셀 출력

```
[-0.007747666910290718, 0.03676600381731987, 0.019548965618014336, -0.0197015218]
```

3) Document 임베딩

- `embeddings.embed_documents()` 함수 사용 → 텍스트 문서를 임베딩
 - `[text]`를 인자로 전달 → 단일 문서를 리스트 형태로 임베딩 함수에 전달
 - 함수 호출 결과로 반환된 임베딩 벡터 = `doc_result` 변수에 할당

```
doc_result = embeddings.embed_documents(  
    [text, text, text, text] # 텍스트를 임베딩하여 문서 벡터를  
)
```

- `doc_result[0][:5]` = `doc_result` 리스트의 첫 번째 요소에서 처음 5개의 문자를 슬라이싱하여 선택

```
# 문서 벡터의 길이 확인하기
```

```
len(doc_result)
```

문서 결과의 첫 번째 요소에서 처음 5개 항목 선택하기

```
doc_result[0][:5]
```

- 셀 출력

```
[-0.007747666910290718, 0.03676600381731987, 0.019548965618014336, -0.0
```

4) 차원 지정

- `text-embedding-3` 모델 클래스를 사용 시 → 반환되는 임베딩의 크기 지정 가능
- 기본적으로 `text-embedding-3-small` = 1536 차원의 임베딩 반환

문서 결과의 첫 번째 요소의 길이 반환하기

```
len(doc_result[0])
```

1536

5) 차원(dimensions) 조정

- `dimensions=1024` 전달 → 임베딩의 크기 = 1024로 줄일 수 있음

OpenAI의 "text-embedding-3-small" 모델 사용 → 1024차원의 임베딩을 생성하는 객체 초기화하기

```
embeddings_1024 = OpenAIEmbeddings(  
    model="text-embedding-3-small",  
    dimensions=1024)
```

차원 조정 (임베딩 크기를 1024로 설정)

주어진 텍스트를 임베딩하고 첫 번째 임베딩 벡터의 길이 반환하기

```
len(embeddings_1024.embed_documents([text])[0])
```

1024

6) 유사도 계산

```
sentence1 = "안녕하세요? 반갑습니다."  
sentence2 = "안녕하세요? 반갑습니다!"  
sentence3 = "안녕하세요? 만나서 반가워요."  
sentence4 = "Hi, nice to meet you."  
sentence5 = "I like to eat apples."
```

```
from sklearn.metrics.pairwise import cosine_similarity
```

```
sentences = [sentence1, sentence2, sentence3, sentence4, sentence5]
```

```
embedded_sentences = embeddings_1024.embed_documents(sentences)
```

```
def similarity(a, b):
    return cosine_similarity([a], [b])[0][0]
```

```
# sentence1 = "안녕하세요? 반갑습니다."
# sentence2 = "안녕하세요? 만나서 반가워요."
# sentence3 = "Hi, nice to meet you."
# sentence4 = "I like to eat apples."

for i, sentence in enumerate(embedded_sentences):
    for j, other_sentence in enumerate(embedded_sentences):
        if i < j:
            print(
                f"[유사도 {similarity(sentence, other_sentence):.4f}] {sentences[i]}
            )
```

- 셀 출력

[유사도 0.9644]	안녕하세요? 반갑습니다.	<=====>	안녕하세요? 반갑습니다!
[유사도 0.8375]	안녕하세요? 반갑습니다.	<=====>	안녕하세요? 만나서 반가워요.
[유사도 0.5043]	안녕하세요? 반갑습니다.	<=====>	Hi, nice to meet you.
[유사도 0.1362]	안녕하세요? 반갑습니다.	<=====>	I like to eat apples.
[유사도 0.8142]	안녕하세요? 반갑습니다!	<=====>	안녕하세요? 만나서 반가워요.
[유사도 0.4792]	안녕하세요? 반갑습니다!	<=====>	Hi, nice to meet you.
[유사도 0.1318]	안녕하세요? 반갑습니다!	<=====>	I like to eat apples.
[유사도 0.5128]	안녕하세요? 만나서 반가워요.	<=====>	Hi, nice to meet you.
[유사도 0.1409]	안녕하세요? 만나서 반가워요.	<=====>	I like to eat apples.
[유사도 0.2250]	Hi, nice to meet you.	<=====>	I like to eat apples.

2. gemini-embedding

```
# API 키를 환경변수로 관리하기 위한 설정 파일
from dotenv import load_dotenv
```

```
# API 키 정보 로드
load_dotenv()
```

```
# True
```

```
from langsmith import Client
from langsmith import traceable

import os

# LangSmith 환경 변수 확인

print("\n--- LangSmith 환경 변수 확인 ---")
langchain_tracing_v2 = os.getenv('LANGCHAIN_TRACING_V2')
langchain_project = os.getenv('LANGCHAIN_PROJECT')
langchain_api_key_status = "설정됨" if
```

"@traceable" 주석은 허용되지 않습니다. 허용되는 값은 다음과 같습니다. [@param, @title, @markdown]



```

if langchain_tracing_v2 == "true" and
    print(f"✅ LangSmith 추적 활성화됨 (I
    print(f"✅ LangSmith 프로젝트: '{la
    print(f"✅ LangSmith API Key: {la
    print("  -> 이제 LangSmith 대시보드에
else:
    print("❌ LangSmith 추적이 완전히 활성
    if langchain_tracing_v2 != "true"
        print(f"  - LANGCHAIN_TRACING
    if not os.getenv('LANGCHAIN_API_K
        print("  - LANGCHAIN_API_KEY기
    if not langchain_project:
        print("  - LANGCHAIN_PROJECT기

```

- 셀 출력

```

--- LangSmith 환경 변수 확인 ---
✅ LangSmith 추적 활성화됨 (LANGCHAIN_TRACING_V2='true')
✅ LangSmith 프로젝트: 'LangChain-prantice'
✅ LangSmith API Key: 설정됨
-> 이제 LangSmith 대시보드에서 이 프로젝트를 확인해 보세요.

```

- 임베딩 모델 생성하기

```

from langchain_google_genai import GoogleGenerativeAIEmbeddings

from dotenv import load_dotenv
import os

load_dotenv()

# API 키 확인
if not os.getenv("GOOGLE_API_KEY"):
    os.environ["GOOGLE_API_KEY"] = input("Enter your Google API key: ")

# Gemini 임베딩 모델 생성 (task_type 명시)
# embeddings1 = 쿼리 검색 임베딩 모델
embeddings1 = GoogleGenerativeAIEmbeddings(
    model="models/gemini-embedding-001",
    #task_type="retrieval_document",
    task_type="retrieval_query",
    google_api_key=os.getenv("GOOGLE_API_KEY")
)

# embeddings2 = 문서 검색을 위한 임베딩 생성
embeddings2 = GoogleGenerativeAIEmbeddings(
    model="models/gemini-embedding-001",
    task_type="retrieval_document",
    #task_type="retrieval_query",
    google_api_key=os.getenv("GOOGLE_API_KEY")
)

```

문서 검색을 위한 임베딩 생성
쿼리 검색을 위한 임베딩 생성

문서 검색을 위한 임베딩 생성
쿼리 검색을 위한 임베딩 생성

- 셀 출력

```
WARNING: All log messages before absl::InitializeLog() is called are written to
E0000 00:00:1758420472.472639 1563901 alts_credentials.cc:93] ALTS creds ignored
E0000 00:00:1758420472.475082 1563901 alts_credentials.cc:93] ALTS creds ignored
```

```
text = "임베딩 테스트를 하기 위한 샘플 문장입니다."
```

```
print(type(text)) # <class 'str'>
```

- 쿼리 임베딩 → **embeddings1** 사용하기

```
query_result2 = embeddings1.embed_query(text) # 0.8s 소요
```

```
print(type(query_result2)) # <class 'list'>
```

```
# 쿼리 벡터의 길이 확인하기
```

```
len(query_result2) # 3072
```

- **query_results[:5]** → 쿼리 결과 리스트의 첫 5개 요소를 슬라이싱해 선택하기

```
query_result2[:5]
```

- 셀 출력

```
[-0.030985329300165176,
 0.015072823502123356,
 0.011131638661026955,
 -0.07030871510505676,
 -0.0013758750865235925]
```

- **Document** 임베딩 → **embeddings2** 사용하기

```
from text_utils import save_sentences_to_data_folder
```

```
sentence1 = "안녕하세요? 반갑습니다."
```

```
sentence2 = "안녕하세요? 반갑습니다!"
```

```
sentence3 = "안녕하세요? 만나서 반가워요."
```

```
sentence4 = "Hi, nice to meet you."
```

```
sentence5 = "I like to eat apples."
```



```
sentences = [sentence1, sentence2, sentence3, sentence4, sentence5]
```

```
# sentences 리스트를 data 폴더에 sentences.txt 파일로 저장
save_sentences_to_data_folder(sentences, "sentences.txt")
```

```
query_result3 = embeddings2.embed_query("../08_Embedding/data/sentences.txt")

print(type(query_result3))          # <class 'list'>
```

```
# 문서 벡터의 길이 확인해보기
```

```
len(query_result3)                  # 3072
```

```
# 문서 결과의 첫 번째 요소에서 첫 5개 항목 선택하기
```

```
query_result3[:5]
```

- 셀 출력

```
[-0.010087567381560802,
 0.021242860704660416,
 0.01916883885860443,
 -0.07958440482616425,
 0.014998613856732845]
```

-
- **차원 지정**: `models/gemini-embedding-001` 모델 = **768** 차원 생성
 - **차원 조정**
 - **GoogleGenerativeAIEmbeddings** 클래스의 **dimensions** 매개변수 사용해야 함
 - **dimensions** 매개변수 = **1 ~ 2048** 사이
 - **차원 조정_1** - 문서 검색

```
# 차원 조정해보기
```

```
from langchain_google_genai import GoogleGenerativeAIEmbeddings

embeddings3 = GoogleGenerativeAIEmbeddings(
    model="models/gemini-embedding-001",
    task_type="retrieval_document",          # 문서 검색을 위한 임베딩 생성
    dimensions=1,                             # 임베딩 차원 조정
    google_api_key=os.getenv("GOOGLE_API_KEY")
)
```

```
E0000 00:00:1758425583.513673 1715309 alts_credentials.cc:93] ALTS creds ignored.
```

- 셀 출력

```
E0000 00:00:1758423695.696036 1563901 alts_credentials.cc:93] ALTS creds ignored
```

```
query_result4 = embeddings3.embed_query("../08_Embedding/data/sentences.txt")  
print(type(query_result4))          # <class 'list'>
```

```
<class 'list'>
```

```
# 차원 조정에 따른 임베딩 결과 비교해보기
```

```
from langchain_experimental.text_splitter import SemanticChunker
```

```
text_splitter1 = SemanticChunker(embeddings2)          # 차원 = 기본값
```

```
text_splitter2 = SemanticChunker(embeddings3)          # 차원 조정 (1)
```

```
# 임베딩 결과 확인
```

```
query_result3 = embeddings2.embed_query("../08_Embedding/data/sentences.txt")
```

```
query_result4 = embeddings3.embed_query("../08_Embedding/data/sentences.txt")
```

```
# 출력해보기
```

```
print(f"임베딩 결과의 길이: {len(query_result3)}")
```

```
print("\n", "="*50, "\n")
```

```
print(f"임베딩 결과의 길이: {len(query_result4)}")
```

- 셀 출력

```
임베딩 결과의 길이: 3072
```

```
=====
```

```
임베딩 결과의 길이: 3072
```

```
# 문서 결과의 첫 번째 요소에서 첫 5개 항목 선택하기
```

```
query_result4[:5]
```

- 셀 출력

```
[-0.010087567381560802,  
 0.021242860704660416,  
 0.01916883885860443,  
 -0.07958440482616425,  
 0.014998613856732845]
```

- 차원 조정_2 - 쿼리 검색

차원 조정해보기

```
from langchain_google_genai import GoogleGenerativeAIEmbeddings

embeddings4 = GoogleGenerativeAIEmbeddings(
    model="models/gemini-embedding-001",
    task_type="retrieval_query",
    dimensions=1,
    google_api_key=os.getenv("GOOGLE_API_KEY")
) # 문서 검색을 위한 임베딩 생성
# 임베딩 차원 조정
```

- 셀 출력

```
E0000 00:00:1758423423.919711 1563901 alts_credentials.cc:93] ALTS creds ignored
```

```
query_result5= embeddings4.embed_query(text) # 0.7s 소요
print(type(query_result2)) # <class 'list'>
```

쿼리 벡터의 길이 확인하기

```
len(query_result5) # 3072
```

쿼리 결과에서 첫 5개 요소 출력하기

```
query_result5[:5]
```

- 셀 출력

```
[-0.030985329300165176,
 0.015072823502123356,
 0.011131638661026955,
 -0.07030871510505676,
 -0.0013758750865235925]
```

- 차원 조정 이후 임베딩 결과의 길이 비교해보기

쿼리 검색

문서 검색

차원 = 기본값

[-0.030985329300165176, [-0.010087567381560802,

	쿼리 검색	문서 검색
	0.015072823502123356,	0.021242860704660416,
	0.011131638661026955,	0.01916883885860443,
	-0.07030871510505676,	-0.07958440482616425,
	-0.0013758750865235925]	0.014998613856732845]
차원 조정 (dimensions = 1)	[-0.030985329300165176,	[-0.010087567381560802,
	0.015072823502123356,	0.021242860704660416,
	0.011131638661026955,	0.01916883885860443,
	-0.07030871510505676,	-0.07958440482616425,
	-0.0013758750865235925]	0.014998613856732845]

- 쿼리 벡터 길이, 문서 벡터 길이 모두 동일
- **GoogleGenerativeAIEmbeddings** 클래스의 **dimensions** 매개변수를 조정해도 `기본적으로 생성되는 임베딩의 차원은 변하지 않음`
 - 공식 가이드 페이지에 따르면, **dimensions** 매개변수 는 임베딩의 차원을 조정하는 것이 아니라, **임베딩을 생성할 때 사용할 차원을 지정하는 것**
 - 조금 더 공부가 필요

• 유사도 계산

```
from langchain_google_genai import GoogleGenerativeAIEmbeddings
from sklearn.metrics.pairwise import cosine_similarity
import numpy as np

embeddings6 = GoogleGenerativeAIEmbeddings(
    model="models/gemini-embedding-001",
    task_type="semantic_similarity",
    google_api_key=os.getenv("GOOGLE_API_KEY")
) # 유사도 검색을 위한 임베딩 생성
```

• 셀 출력

```
E0000 00:00:1758426508.045810 1715309 alts_credentials.cc:93] ALTS creds ignored
```

```
# 문장 리스트
sentences = [
    "안녕하세요? 반갑습니다.",
    "안녕하세요? 반갑습니다!",
    "안녕하세요? 만나서 반가워요.",
    "Hi, nice to meet you.",
    "I like to eat apples."
]
```

```
# 문장 임베딩 생성
sentence_embeddings = embeddings6.embed_documents(sentences)
```

```
# 유사도 행렬 계산
similarity_matrix = cosine_similarity(sentence_embeddings)
```

```
# 유사도 출력
for i in range(len(sentences)):
    for j in range(i + 1, len(sentences)):
        print(f"[유사도 {similarity_matrix[i][j]:.4f}] {sentences[i]} \t <=====> \t {sentences[j]}")
```

[유사도 0.9940]	안녕하세요? 반갑습니다.	<=====>	안녕하세요? 반갑습니다!
[유사도 0.9751]	안녕하세요? 반갑습니다.	<=====>	안녕하세요? 만나서 반가워요.
[유사도 0.9221]	안녕하세요? 반갑습니다.	<=====>	Hi, nice to meet you.
[유사도 0.7319]	안녕하세요? 반갑습니다.	<=====>	I like to eat apples.
[유사도 0.9704]	안녕하세요? 반갑습니다!	<=====>	안녕하세요? 만나서 반가워요.
[유사도 0.9146]	안녕하세요? 반갑습니다!	<=====>	Hi, nice to meet you.
[유사도 0.7284]	안녕하세요? 반갑습니다!	<=====>	I like to eat apples.
[유사도 0.9269]	안녕하세요? 만나서 반가워요.	<=====>	Hi, nice to meet you.
[유사도 0.7295]	안녕하세요? 만나서 반가워요.	<=====>	I like to eat apples.
[유사도 0.7580]	Hi, nice to meet you.	<=====>	I like to eat apples.

-
- next: **캐시 임베딩 (CacheBackedEmbeddings)**
-