

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

2. FAISS

- Facebook AI Similarity Search (Fasiss)**
 - 밀집 벡터의 효율적인 유사도 검색과 클러스터링을 위한 라이브러리
 - RAM에 맞지 않을 수도 있는 벡터 집합을 포함하여 모든 크기의 벡터 집합을 검색하는 알고리즘을 포함
 - 평가와 매개변수 튜닝을 위한 지원 코드도 포함
- 참고 링크
 - [LangChain FAISS 문서](#)
 - [FAISS 문서](#)

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 os.getenv('LANGCHAIN_API_KEY') else "설정되지 않음" # API 키 값은 직접 출력하지 않음

if langchain_tracing_v2 == "true" and os.getenv('LANGCHAIN_API_KEY') and langchain_project:
    print(f"✅ LangSmith 추적 활성화됨 (LANGCHAIN_TRACING_V2='{langchain_tracing_v2}')
```

- 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.
```

- 샘플 데이터셋 로드하기

```

from langchain_community.document_loaders import TextLoader
from langchain.text_splitter import RecursiveCharacterTextSplitter

# 텍스트 분할
text_splitter = RecursiveCharacterTextSplitter(chunk_size=600, chunk_overlap=0)

# 텍스트 파일을 load -> List[Document] 형태로 변환
loader1 = TextLoader("../09_VectorStore/data/nlp-keywords.txt")
loader2 = TextLoader("../09_VectorStore/data/finance-keywords.txt")

# 문서 분할
split_doc1 = loader1.load_and_split(text_splitter)
split_doc2 = loader2.load_and_split(text_splitter)

# 문서 개수 확인
print(f"첫번째 문서의 개수: {len(split_doc1)}")
print("="*25)
print(f"두번째 문서의 개수: {len(split_doc2)}")

```

- 셀 출력 (0.3s)

```

첫번째 문서의 개수: 11
=====
두번째 문서의 개수: 6

```

▼ 2) VectorStore 생성

- **주요 초기화 매개 변수**
 - 인덱싱 매개변수: **embedding_function** (Embeddings) = 사용할 임베딩 함수
 - 클라이언트 매개변수
 - **index** (Any): 사용할 **FAISS** 인덱스
 - **docstore** (Docstore): 사용할 **문서 저장소**
 - **index_to_docstore_id** (Dict[int, str]): 인덱스에서 **문서 저장소 ID** 로의 매핑
- **참고**
 - 고성능 벡터 검색 및 클러스터링을 위한 라이브러리
 - **LangChain** 의 **VectorStore** 인터페이스와 통합
 - 임베딩 함수, FAISS 인덱스, 문서 저장소를 **조합** → **효율적인 벡터 검색 시스템** 을 구축

- **Sentence Transformers** 모델 사용해보기
 - 초경량화 모델: 로컬 실행 = API 불필요 = 네트워크 불필요
 - 384 차원 (google-embedding 모델의 1/8 메모리 사용량)
 - **FAISS** 와 완벽 호환
- 사전에 **VS Code** 터미널에 설치할 것

```

pip install -U langchain-huggingface sentence-transformers faiss-cpu

pip install -U langchain langchain-community

```

```

# 교재 내용으로 실습 시작해보기


import gc
import numpy as np
import time
from langchain_huggingface import HuggingFaceEmbeddings
from langchain_community.vectorstores import FAISS
import warnings

# 경고 무시
warnings.filterwarnings("ignore")

```

```
# 임베딩
embeddings = HuggingFaceEmbeddings()

# 임베딩 차원 크기 계산해보기
dimension_size = len(embeddings.embed_query("hello world"))
print(dimension_size) # 768
```

- 셀 출력 (49.6s)
 - 임베딩 차원 크기: **768**
 -  과정 및 결과

3) FAISS 벡터 저장소 생성 (from_documents)

- **from_documents** 클래스 메서드 = 문서 리스트 + 임베딩 함수 사용 → FAISS 벡터 저장소 생성
- 매개변수
 - **documents (List [Document])**: 벡터 저장소에 추가할 문서 리스트
 - **embedding (Embeddings)**: 사용할 임베딩 함수
 - **kwargs**: 추가 키워드 인자
- 동작 방식
 - 문서 리스트 → 텍스트 내용(**page_content**), 메타데이터 추출
 - 추출한 텍스트 + 메타데이터 → **from_texts** 메서드 호출
- 반환값
 - **VectorStore**: 문서와 임베딩으로 초기화된 벡터 저장소 인스턴스
- 참고
 - **from_texts** 메서드 내부적으로 호출 → 벡터 저장소 생성
 - 문서
 - **page_content** = 텍스트
 - **metadata** = 메타데이터
 - 추가적인 설정이 필요한 경우 **kwargs** 를 통해 전달 가능

```
# 초경량화 허깅페이스 모델 설치

import gc
import numpy as np
import time
from langchain_huggingface import HuggingFaceEmbeddings
from langchain_community.vectorstores import FAISS
import warnings

# 경고 무시
warnings.filterwarnings("ignore")

def ultra_light_faiss_updated():
    """최신 LangChain으로 수정된 초경량 FAISS"""

    print("🚀 LangChain으로 초경량 임베딩 시작...")

    try:
        # HuggingFace Embeddings 사용
        embeddings = HuggingFaceEmbeddings(
            model_name="sentence-transformers/all-MiniLM-L6-v2",
            model_kwargs={'device': 'cpu'},
            encode_kwargs={'normalize_embeddings': True}
        )

        print("✅ hugging-face 임베딩 모델 로딩 완료!")

        # FAISS 벡터스토어 생성
        print("🔄 FAISS 벡터스토어 생성 중...")
        start_time = time.time()

        db = FAISS.from_documents(
```

```

        from_documents=documents,
        documents=split_doc1,
        embedding=embeddings,
    )

    end_time = time.time()
    print(f"✅ FAISS 생성 완료! (소요시간: {end_time-start_time:.2f}초)")

    # 저장된 결과 확인하기
    print("📄 Document 확인: ", "\n")
    print(db.docstore._dict)
    print("\n", "문서 저장소 ID 확인: ", "\n")
    print(db.index_to_docstore_id)

    # 인덱스 저장
    db.save_local("../09_VectorStore/faiss_light_index")
    print("📁 인덱스 저장 완료!")

    # 메모리 정리
    del db, embeddings
    gc.collect()

    return True

except Exception as e:
    print(f"❌ 오류 발생: {e}")
    print("💡 패키지 설치를 다시 확인해보세요!")
    gc.collect()
    return False

```

확인해보기

```

success = ultra_light_faiss_updated()
if success:
    print(" from_documents으로 확인 성공 ")

```

• 셀 출력 (3.4s)

- 🚀 LangChain으로 초경량 임베딩 시작...
- ✅ hugging-face 임베딩 모델 로딩 완료!
- 🔄 FAISS 벡터스토어 생성 중...
- ✅ FAISS 생성 완료! (소요시간: 0.32초)
- 📄 Document 확인:

```
{'133d6b9b-6883-41cb-8ce2-8c414c259077': Document(id='133d6b9b-6883-41cb-8ce2-8c414c259077', metadata={'source':
```

- 문서 저장소 ID 확인:

```
{0: '133d6b9b-6883-41cb-8ce2-8c414c259077', 1: '04498a8b-1572-423c-8c02-3994c43b64f1', 2: '7f16d72b-a34a-455e-aa
```

- 📁 인덱스 저장 완료!
- from_documents으로 확인 성공

4) FAISS 벡터 저장소 생성 (from_texts)

- from_texts** 클래스 매서드 = 텍스트 리스트 + 임베딩 함수 사용 → FAISS 벡터 저장소 생성

매개변수

- texts (List[str]):** 벡터 저장소에 추가할 텍스트 리스트
- embedding (Embeddings):** 사용할 임베딩 함수
- metadatas (Optional[List[dict]]):** 메타데이터 리스트 (기본값 = None)
- ids (Optional[List[str]]):** 문서 ID 리스트 (기본값 = None)
- kwargs:** 추가 키워드 인자

동작 방식

- 제공된 임베딩 함수 사용 → 텍스트 임베딩
- 임베딩 벡터 + **_from** 메서드 호출 → **FAISS** 인스턴스 생성

반환값

- **FAISS**: 생성된 FAISS 벡터 저장소 인스턴스

- **참고**

- 사용자 친화적인 인터페이스 → 문서 임베딩, 메모리 내 문서 저장소 생성, FAISS 데이터베이스 초기화 를 한 번에 처리
- **빠르게 시작하기 위한 편리한 방법**

- **주의사항**

- 대량의 텍스트 처리 시 **메모리 사용량에 주의해야 함**
- **메타데이터**, **ID** 사용하려면 텍스트 리스트와 동일한 길이의 리스트 로 제공해야 함

- 교재 내용으로 실습해보기

```
# 초경량화 허깅페이스 모델 설치

import gc
import numpy as np
import time
from langchain_huggingface import HuggingFaceEmbeddings
from langchain_community.vectorstores import FAISS
import warnings

# 경고 무시
warnings.filterwarnings("ignore")

def ultra_light_faiss_updated():
    """최신 LangChain으로 수정된 초경량 FAISS"""

    print("🚀 LangChain으로 초경량 임베딩 시작...")

    try:
        # HuggingFace Embeddings 사용
        embeddings = HuggingFaceEmbeddings(
            model_name="sentence-transformers/all-MiniLM-L6-v2",
            model_kwargs={'device': 'cpu'},
            encode_kwargs={'normalize_embeddings': True}
        )

        print("✅ hugging-face 임베딩 모델 로딩 완료!")

        # 테스트 문서
        documents = [
            "안녕하세요. 정말 반갑습니다.", "제 이름은 엘리스입니다."
        ]

        # FAISS 벡터스토어 생성
        print("🔄 FAISS 벡터스토어 생성 중...")
        start_time = time.time()

        db2 = FAISS.from_texts(
            texts=documents,
            embedding=embeddings,
            metadatas=[{"source": "텍스트문서"}, {"source": "텍스트문서"}],
            ids=["doc1", "doc2"]
        )

        end_time = time.time()
        print(f"✅ FAISS 생성 완료! (소요시간: {end_time-start_time:.2f}초)")

        # 저장된 결과 확인하기
        print("🔍 검색 결과: ", "\n")
        print(db2.docstore._dict)

        # 인덱스 저장
        db2.save_local("../09_VectorStore/faiss_light_index")
        print("💾 인덱스 저장 완료!")

        # 메모리 정리
        del db2, embeddings
        gc.collect()

        return True

    except Exception as e:
        print(f"❌ 오류 발생: {e}")
        print("💡 패키지 설치를 다시 확인해보세요!")
```

```
gc.collect()
return False
```

확인해보기

```
success = ultra_light_faiss_updated()
if success:
    print(" from_texts로 확인 성공 ")
```

• 셀 출력 (3.2s)

- 🚀 LangChain으로 초경량 임베딩 시작...
- ✅ hugging-face 임베딩 모델 로딩 완료!
- 🔄 FAISS 벡터스토어 생성 중...
- ✅ FAISS 생성 완료! (소요시간: 0.03초)
- 🏁 검색 결과:

```
{'doc1': Document(id='doc1', metadata={'source': '텍스트문서'}, page_content='안녕하세요. 정말 반갑습니다.'), 'doc2': Doc
```

- 📁 인덱스 저장 완료!
- from_texts로 확인 성공

유사도 검색 포함해보기

```
import gc
import numpy as np
import time
from langchain_huggingface import HuggingFaceEmbeddings
from langchain_community.vectorstores import FAISS
import warnings

# 경고 무시
warnings.filterwarnings("ignore")

def ultra_light_faiss_updated():
    """최신 LangChain으로 수정된 초경량 FAISS"""

    print("🚀 LangChain으로 초경량 임베딩 시작...")

    try:
        # HuggingFace Embeddings 사용
        embeddings = HuggingFaceEmbeddings(
            model_name="sentence-transformers/all-MiniLM-L6-v2",
            model_kwargs={'device': 'cpu'},
            encode_kwargs={'normalize_embeddings': True}
        )

        print("✅ 최신 임베딩 모델 로딩 완료!")
        print("🔍 임베딩 차원: 384 (Google Gemini 대비 1/8 절약!)")

        # 테스트 문서
        documents = [
            "자연어 처리는 컴퓨터가 인간의 언어를 이해하는 기술입니다.",
            "머신러닝은 데이터로부터 패턴을 학습하는 방법입니다.",
            "딥러닝은 신경망을 이용한 머신러닝 기법입니다.",
            "FAISS는 효율적인 유사도 검색을 위한 라이브러리입니다.",
            "벡터 데이터베이스는 임베딩을 저장하고 검색하는 시스템입니다."
        ]

        # FAISS 벡터스토어 생성
        print("🔄 FAISS 벡터스토어 생성 중...")
        start_time = time.time()

        vectorstore = FAISS.from_texts(
            texts=documents,
            embedding=embeddings,
            metadatas=[{"id": i, "source": f"doc_{i}"} for i in range(len(documents))]
        )

        end_time = time.time()
        print(f"✅ FAISS 생성 완료! (소요시간: {end_time-start_time:.2f}초)")

        # 유사도 검색 테스트
        print("\n🔍 유사도 검색 테스트...")
        query = "딥러닝과 머신러닝의 차이점"

        results = vectorstore.similarity_search_with_score(query, k=3)
```

```

print("🔍 검색 결과:")
for i, (doc, score) in enumerate(results):
    similarity = 1 - score
    print(f" {i+1}. 유사도: {similarity:.4f}")
    print(f"     내용: {doc.page_content}")
    print(f"     메타데이터: {doc.metadata}\n")

# 인덱스 저장
vectorstore.save_local("../09_VectorStore/faiss_light_index")
print("📁 인덱스 저장 완료!")

# 메모리 정리
del vectorstore, embeddings
gc.collect()

return True

except Exception as e:
    print(f"❌ 오류 발생: {e}")
    print("💡 패키지 설치를 다시 확인해보세요!")
    gc.collect()
    return False

```

🚧 실행!

```

success = ultra_light_faiss_updated()
if success:
    print("🎉🎉🎉 최신 LangChain으로 FAISS 완료! 🎉🎉🎉")

```

• 셀 출력 (9.3s)

```

🚀 LangChain으로 초경량 임베딩 시작...
✅ 최신 임베딩 모델 로딩 완료!
🔗 임베딩 차원: 384 (Google Gemini 대비 1/8 절약!)
🔄 FAISS 벡터스토어 생성 중...
✅ FAISS 생성 완료! (소요시간: 1.48초)

🔍 유사도 검색 테스트...
🔍 검색 결과:
1. 유사도: 0.5892
   내용: 답변은 신경망을 이용한 머신러닝 기법입니다.
   메타데이터: {'id': 2, 'source': 'doc_2'}

2. 유사도: 0.4567
   내용: 머신러닝은 데이터로부터 패턴을 학습하는 방법입니다.
   메타데이터: {'id': 1, 'source': 'doc_1'}

3. 유사도: 0.4415
   내용: 벡터 데이터베이스는 임베딩을 저장하고 검색하는 시스템입니다.
   메타데이터: {'id': 4, 'source': 'doc_4'}

📁 인덱스 저장 완료!
🎉🎉🎉 최신 LangChain으로 FAISS 완료! 🎉🎉🎉

```

✓ 5) 유사도 검색 (Similarity Search)

- **similarity_search** 클래스 메서드: 주어진 쿼리와 가장 유사한 문서들을 검색하는 기능 제공
- **매개변수**
 - **query (str)**: 유사한 문서를 찾기 위한 검색 쿼리 텍스트
 - **k (int)**: 반환할 문서 수 (**기본값 = 4**)
 - **filter (Optional [Union [Callable, Dict [str, Any]]])**: 메타데이터 필터링 함수 or 딕셔너리 (**기본값 = None**)
 - **fetch_k (int)**: 필터링 전에 가져올 문서 수 (**기본값 = 20**)
 - **kwargs**: 추가 키워드 인자
- **반환값**
 - **List**: 쿼리와 가장 유사한 문서 리스트

- **동작 방식**
 - **similarity_search_with_score** 메서드 내부적으로 호출 → 유사도 점수와 함께 문서 검색
 - 검색 결과에서 점수를 제외 하고 **문서만 추출해 반환**
- **주의사항**
 - 검색 성능: **사용된 임베딩 모델의 품질** 에 크게 의존
 - 대규모 데이터셋: **k** 와 **fetch_k** 값을 적절히 조정 → 검색 속도와 정확도의 균형을 맞추는 것이 중요함
 - 복잡한 필터링이 필요한 경우: **filter** 매개변수에 **커스텀 함수** 를 전달 세밀한 제어가 가능함
- **최적화 팁**
 - 자주 사용되는 쿼리: **결과 캐싱** → 반복적인 검색 속도 향상 가능
 - **fetch_k** 를 너무 크게 설정 → 검색 속도 ↓ → **적절한 값** 을 실험적으로 찾는 것 권장

초경량화 허깅페이스 모델 설치

```
import gc
import numpy as np
import time
from langchain_huggingface import HuggingFaceEmbeddings
from langchain_community.vectorstores import FAISS
import warnings

# 경고 무시
warnings.filterwarnings("ignore")

def ultra_light_faiss_updated():
    """최신 LangChain으로 수정된 초경량 FAISS"""

    print("🚀 LangChain으로 초경량 임베딩 시작...")

    try:
        # HuggingFace Embeddings 사용
        embeddings = HuggingFaceEmbeddings(
            model_name="sentence-transformers/all-MiniLM-L6-v2",
            model_kwargs={'device': 'cpu'},
            encode_kwargs={'normalize_embeddings': True}
        )

        print("✅ hugging-face 임베딩 모델 로딩 완료!")

        # FAISS 벡터스토어 생성
        print("🔄 FAISS 벡터스토어 생성 중...")
        start_time = time.time()

        db3 = FAISS.from_documents(
            documents=split_doc1,
            embedding=embeddings,
        )

        end_time = time.time()
        print(f"✅ FAISS 생성 완료! (소요시간: {end_time-start_time:.2f}초)")

        # 유사도 검색 테스트
        print("\n🔍 유사도 검색 테스트...")
        query = "TF IDF 에 대하여 알려줘"

        results = db3.similarity_search_with_score(query, k=3)

        print("1. 기본 유사도 검색: ", db3.similarity_search(query), "\n")
        print("2. k값에 검색 결과 개수 지정하기: ", db3.similarity_search(query, k=2), "\n")
        print("3. metadata 정보로 filtering하기: ", db3.similarity_search(query, filter={"source" : "../09_VectorStore/data/nlp-key"}, k=20), "\n")
        print("4. metadata 정보로 filtering하기_2: ", db3.similarity_search(query, filter={"source" : "../09_VectorStore/data/finar"}, k=20), "\n")

        # 인덱스 저장
        db3.save_local("../09_VectorStore/faiss_light_index")
        print("💾 인덱스 저장 완료!")

        # 메모리 정리
        del db3, embeddings
        gc.collect()

        return True

    except Exception as e:
        print(f"❌ 오류 발생: {e}")
        print("💡 패키지 설치를 다시 확인해보세요!")
```



```
gc.collect()
return False
```

🗑️ 실행!

```
success = ultra_light_faiss_updated()
if success:
    print("🎉 from_documents로 유사도 검색 완료! 🎉")
```

- 셀 출력 (4.3s)
- 🚀 LangChain으로 초경량 임베딩 시작...
- ✅ hugging-face 임베딩 모델 로딩 완료!
- 🔄 FAISS 벡터스토어 생성 중...
- ✅ FAISS 생성 완료! (소요시간: 0.38초)
- 🔍 유사도 검색 테스트...
- 1. 기본 유사도 검색:

```
[Document(id='c5014297-da20-4f0f-b602-15506e0ef249', metadata={'source': '../09_VectorStore/data/nlp-keywords.txt'})]
```

- 2. k값에 검색 결과 개수 지정하기:

```
[Document(id='c5014297-da20-4f0f-b602-15506e0ef249', metadata={'source': '../09_VectorStore/data/nlp-keywords.txt'})]
```

- 3. metadata 정보로 filtering하기:

```
[Document(id='c5014297-da20-4f0f-b602-15506e0ef249', metadata={'source': '../09_VectorStore/data/nlp-keywords.txt'})]
```

===== 다른 source로 검색해보기 =====

- 4. metadata 정보로 filtering하기_2:

```
[]
```

- 📁 인덱스 저장 완료!
- 🎉 from_documents로 유사도 검색 완료! 🎉

6) 문서로부터 추가

- **add_documents** 메서드: 벡터 저장소에 문서를 추가 or 업데이트
- **매개변수**
 - **documents** (`List [Document]`): 벡터 저장소에 추가할 문서 리스트
 - **kwargs**: 추가 키워드 인자
- **반환값**
 - **List**: 벡터 저장소에 추가된 텍스트의 **ID** 리스트
- **동작 방식**
 - 문서에서 **텍스트 내용**, **메타데이터** 추출
 - **add_texts** 메서드 호출 → 실제 추가 작업 수행
- **주요 특징**
 - **편이성**: 문서 객체를 직접 처리할 수 있음
 - **고유성 보장**: **ID** 처리 로직 포함 → 문서의 고유성 보장
 - **코드의 재사용성 ↑**: **add_texts** 메서드 기반으로 동작

초경량화 허깅페이스 모델 설치

```
import gc
import numpy as np
```

```

import time
from langchain_huggingface import HuggingFaceEmbeddings
from langchain_community.vectorstores import FAISS
from langchain_core.documents import Document
import warnings

# 경고 무시
warnings.filterwarnings("ignore")

def ultra_light_faiss_updated():
    """최신 LangChain으로 수정된 초경량 FAISS"""

    print("🚀 from_document에서 문서 추가 시작...")

    try:
        # HuggingFace Embeddings 사용
        embeddings = HuggingFaceEmbeddings(
            model_name="sentence-transformers/all-MiniLM-L6-v2",
            model_kwargs={'device': 'cpu'},
            encode_kwargs={'normalize_embeddings': True}
        )

        print("✅ hugging-face 임베딩 모델 로딩 완료!")

        # FAISS 벡터스토어 생성
        print("🔄 FAISS 벡터스토어 생성 중...")
        start_time = time.time()

        db4 = FAISS.from_documents(
            documents=split_doc1,
            embedding=embeddings,
        )

        end_time = time.time()
        print(f"✅ FAISS 생성 완료! (소요시간: {end_time-start_time:.2f}초)")

        # 문서 추가해보기
        db4.add_documents(
            [
                Document(
                    page_content="안녕하세요! 이번엔 도큐먼트를 새로 추가해 볼게요",
                    metadata={"source": "mydata.txt"},
                )
            ],
            ids=["new_doc1"]
        )

        # 추가된 데이터 확인해보기
        print("1. 문서 확인해보기: ", "\n")
        print(db4.docstore._dict, "\n")

        print("2. 문서 저장소 ID 확인해보기: ", "\n")
        print(db4.index_to_docstore_id, "\n")

        print("3. 🔍 유사도 검색 테스트...", "\n")
        print(db4.similarity_search("안녕하세요", k=1), "\n")

        # 인덱스 저장
        db4.save_local("../09_VectorStore/faiss_light_index")
        print("💾 인덱스 저장 완료!")

    except Exception as e:
        print(f"❌ 오류 발생: {e}")
        print("💡 패키지 설치를 다시 확인해보세요!")
        gc.collect()
        return False

```

🏃 실행!

```

success = ultra_light_faiss_updated()
if success:
    print("\n", "🚀 from_documents로 문서 추가 완료! 🎉")

```

- 셀 출력 (3.2s)
- 🚀 from_document에서 문서 추가 시작...
- ✅ hugging-face 임베딩 모델 로딩 완료!
- 🔄 FAISS 벡터스토어 생성 중...
- ✅ FAISS 생성 완료! (소요시간: 0.34초)

- 1. 문서 확인해보기:

```
{'0430ed8e-c5c4-419a-939a-1cfbd88f1acb': Document(id='0430ed8e-c5c4-419a-939a-1cfbd88f1acb', metadata={'source': 'mydata.txt', 'page_content': '안녕하세요! 이번엔 도큐먼트를 새로 추가해 볼게요.'})}
```

- 2. 문서 저장소 ID 확인해보기:

```
{0: '0430ed8e-c5c4-419a-939a-1cfbd88f1acb', 1: 'c4bb632b-e1f9-4c44-a3e9-f8a005614998', 2: 'ab7ad5eb-30b2-4311-a095-8c4b6b6b6b6b'}
```

- 3. 🔍 유사도 검색 테스트...

```
[Document(id='new_doc1', metadata={'source': 'mydata.txt'}, page_content='안녕하세요! 이번엔 도큐먼트를 새로 추가해 볼게요.')]
```

- 📁 인덱스 저장 완료!
- 🎉 from_documents로 문서 추가 완료! 🎉

7) 문서 삭제 (Delete Documents)

- delete** 매서드: 벡터 저장소에서 지정된 ID에 해당하는 문서를 삭제하는 기능
- 매개변수**
 - ids** (Optional [List [str]]): 삭제할 문서의 ID 리스트
 - kwargs**: 추가 키워드 인자 (이 매서드에서는 사용되지 않음)
- 반환값**
 - Optional [bool]**: 삭제 성공 시 True, 실패 시 False, 구현되지 않은 경우 None
- 동작 방식**
 - 입력된 ID의 유효성 검사
 - 삭제할 ID에 해당하는 인덱스 찾기
 - FAISS 인덱스에서 해당 ID 제거
 - 문서 저장소에서 해당 ID 문서 삭제
 - 인덱스와 ID 매핑 업데이트
- 주요 특징**
 - ID 기반 삭제로 정확한 문서 관리 가능
 - FAISS 인덱스, 문서 저장소 양쪽에서 삭제 수행
 - 삭제 후 인덱스 재정렬 → 데이터 일관성을 유지함
- 주의사항**
 - 삭제 작업은 되돌릴 수 없음 → 신중하게 수행하기
 - 동시성 제어 미구현 → 다중 스레드 환경에서 주의 필요

초경량화 허깅페이스 모델 설치

```
import gc
import numpy as np
import time
from langchain_huggingface import HuggingFaceEmbeddings
from langchain_community.vectorstores import FAISS
from langchain_core.documents import Document
import warnings
```

```
# 경고 무시
warnings.filterwarnings("ignore")
```

```
# 위에서 문서를 추가한 db
db4 = FAISS.from_documents(
    documents=split_doc1,
    embedding=embeddings,
)
```

```
# 삭제용 데이터 추가하기
ids = db4.add_texts(
```

```

["삭제용 데이터를 추가합니다.", "2번째 삭제용 데이터입니다."],
metadatas=[{"source":"mydata.txt"}, {"source":"mydata.txt"}],
ids=["delete_doc1", "delete_doc2"],
)

# 인덱스 저장하기
try:
    db4.save_local("../09_VectorStore/faiss_light_index")
    print("📁 인덱스 저장 완료!")

except Exception as e:
    print(f"❌ 오류 발생: {e}")
    print("💡 패키지 설치를 다시 확인해보세요!")

```

- 셀 출력 (6.8s): 📁 인덱스 저장 완료!

```

# 추가된 데이터 확인해보기
print("1. 문서 확인해보기: ", "\n")
print(db4.docstore._dict, "\n")

print("2. 문서 저장소 ID 확인해보기: ", "\n")
print(db4.index_to_docstore_id, "\n")

print("3. 🔍 유사도 검색 테스트...", "\n")
print(db4.similarity_search("삭제용", k=2), "\n")

```

- 셀 출력 (1.3s)

1. 문서 확인해보기:

```
{'899dd90c-89a5-4e9d-81fd-6a22dcb66da1': Document(id='899dd90c-89a5-4e9d-81fd-6a22dcb66da1', metadata={'source': 'mydata.txt', 'page_content': '삭제용 데이터를 추가합니다.'})}
```

2. 문서 저장소 ID 확인해보기:

```
{0: '899dd90c-89a5-4e9d-81fd-6a22dcb66da1', 1: 'd9fd0c81-a887-47ce-bf46-739f25a80e1b', 2: '79e85496-2237-402e-b066-5592c78756ef', 3: '218eeb92-b29f-400d-bf0e-6c24b5a672b7', 4: '02b5f998-8d23-4d52-b826-85ab3977038f', 5: '0b179e4b-ff2a-4a5e-9a57-169f4d9e993e', 6: '69604ca8-9082-491b-8c89-ddf77b62e0ba', 7: '761ba4a0-3e47-4ae3-ad9c-aa4c49edc2cb', 8: '11cd306a-5984-4e6c-899d-cbfb899078cf', 9: '4c119fb2-0937-4bdd-b969-bca13b107445', 10: 'c1780d4f-cd20-40fa-9cd2-5987e93f39cf', 11: 'new_doc1'}
```

3. 🔍 유사도 검색 테스트...

```
[Document(id='delete_doc2', metadata={'source': 'mydata.txt', 'page_content': '2번째 삭제용 데이터입니다.'}), Document(id='delete_doc1', metadata={'source': 'mydata.txt', 'page_content': '삭제용 데이터를 추가합니다.'})]
```

```

# 삭제할 id 확인해보기

print(ids)                                # ['delete_doc1', 'delete_doc2']

```

```

# id로 삭제해보기

db4.delete(ids)                            # True

```

```

# 삭제된 결과 출력해보기

db4.index_to_docstore_id

```

- 셀 출력

```

{0: '899dd90c-89a5-4e9d-81fd-6a22dcb66da1',
1: 'd9fd0c81-a887-47ce-bf46-739f25a80e1b',
2: '79e85496-2237-402e-b066-5592c78756ef',
3: '218eeb92-b29f-400d-bf0e-6c24b5a672b7',
4: '02b5f998-8d23-4d52-b826-85ab3977038f',
5: '0b179e4b-ff2a-4a5e-9a57-169f4d9e993e',
6: '69604ca8-9082-491b-8c89-ddf77b62e0ba',
7: '761ba4a0-3e47-4ae3-ad9c-aa4c49edc2cb',
8: '11cd306a-5984-4e6c-899d-cbfb899078cf',
9: '4c119fb2-0937-4bdd-b969-bca13b107445',
10: 'c1780d4f-cd20-40fa-9cd2-5987e93f39cf',
11: 'new_doc1'}

```

```

# 메모리 정리
del db4, embeddings
gc.collect()                                # 5131

```

8) 텍스트로부터 추가

- **add_texts** 매서드: 텍스트를 임베딩, 벡터저장소에 추가하는 기능 제공
- **매개변수**
 - **texts** (`Iterable [str]`): 벡터 저장소에 추가할 이터러블 텍스트
 - **metadatas** (`Optional [List [dict]`): 텍스트와 연관된 메타데이터 리스트 (*선택적*)
 - **ids** (`Optional [List [str]`): 텍스트의 고유 식별자 리스트 (*선택적*)
 - **kwargs**: 추가 키워드 인자
- **반환값**
 - **List**: 벡터 저장소에 추가된 텍스트의 **ID** 리스트

```
# 초경량화 허깅페이스 모델 설치

import gc
import numpy as np
import time
from langchain_huggingface import HuggingFaceEmbeddings
from langchain_community.vectorstores import FAISS
from langchain_core.documents import Document
import warnings

# 경고 무시
warnings.filterwarnings("ignore")

def ultra_light_faiss_updated():
    """최신 LangChain으로 수정된 초경량 FAISS"""

    print("🚀 from_document에서 문서 추가 시작...")

    try:
        # HuggingFace Embeddings 사용
        embeddings = HuggingFaceEmbeddings(
            model_name="sentence-transformers/all-MiniLM-L6-v2",
            model_kwargs={'device': 'cpu'},
            encode_kwargs={'normalize_embeddings': True}
        )

        print("✅ hugging-face 임베딩 모델 로딩 완료!")

        # FAISS 벡터스토어 생성
        print("🔄 FAISS 벡터스토어 생성 중...")
        start_time = time.time()

        db5 = FAISS.from_documents(
            documents=split_doc1,
            embedding=embeddings,
        )

        end_time = time.time()
        print(f"✅ FAISS 생성 완료! (소요시간: {end_time-start_time:.2f}초)")

        # 문서 추가해보기
        db5.add_documents(
            [
                Document(
                    page_content="안녕하세요! 이번엔 도큐먼트를 새로 추가해 볼게요",
                    metadata={"source": "mydata.txt"},
                )
            ],
            ids=["new_doc1"]
        )

        # 추가된 데이터 확인해보기
        print("1. 문서 확인해보기: ", "\n")
        print(db5.docstore._dict, "\n")

        print("2. 문서 저장소 ID 확인해보기: ", "\n")
        print(db5.index_to_docstore_id, "\n")

        print("3. 🔍 유사도 검색 테스트...", "\n")
        print(db5.similarity_search("안녕하세요", k=1), "\n")
```

```
# 인덱스 저장
db5.save_local("../09_VectorStore/faiss_index")
print("📁 faiss_db, 인덱스 저장 완료!")
```

```
except Exception as e:
    print(f"❌ 오류 발생: {e}")
    print("💡 패키지 설치를 다시 확인해보세요!")
    gc.collect()
    return False
```

```
# 🚀 실행!
```

```
success = ultra_light_faiss_updated()
if success:
    print("\n", "🎉 from_documents로 문서 추가 완료!🎉")
```

```
# db5 불러오기
db5 = FAISS.from_documents(
    documents=split_doc1,
    embedding=embeddings,
)
```

```
# 문서 추가해보기
```

```
db5.add_documents(
    [
        Document(
            page_content="안녕하세요! 이번엔 도큐먼트를 새로 추가해 볼게요",
            metadata={"source": "mydata.txt"},
        )
    ],
    ids=["new_doc1"]
)
# ['new_doc1']
```

```
# 텍스트로 추가해보기
```

```
db5.add_texts(
    ["이번엔 텍스트 데이터를 추가합니다.", "추가한 2번째 텍스트 데이터 입니다."],
    metadatas=[{"source": "mydata.txt"}, {"source": "mydata.txt"}],
    ids=["new_doc2", "new_doc3"],
)
# ['new_doc2', 'new_doc3']
```

```
# 추가된 데이터 확인해보기
```

```
db5.index_to_docstore_id
```

• 셀 출력

```
{0: '4cdd0221-b9f7-4dc0-bc0c-4f4eaa3d0581',
1: '31e288f7-cb3f-4670-b1fb-8508b74a5deb',
2: '9f693bbc-2d28-4023-bb52-759d7f13cc77',
3: 'd52a9cfc-0535-4ac7-8baa-7df05c8eae9f',
4: '04b13d9c-b9ec-47a8-bf22-dc80c9ef845d',
5: 'c7c12db7-0209-477b-a860-634c5e9bcbf8',
6: 'f7c90bdc-4882-459c-a0a2-ea028df48072',
7: 'db6a2852-1519-4191-b180-47893bc986ff',
8: '30791600-ac08-441e-91f0-883418070537',
9: '24cdb5b9-e433-4af1-ac0f-1387cbce7be4',
10: 'f65d3bc9-329d-4d79-a4e6-47dff4dd1780',
11: 'new_doc1',           # 문서로 추가
12: 'new_doc2',           # 텍스트로 추가
13: 'new_doc3'}           # 텍스트로 추가
```

```
# 삭제해보기
```

```
ids = db5.add_texts(
    ["삭제용 데이터를 추가합니다.", "2번째 삭제용 데이터입니다."],
    metadatas=[{"source": "mydata.txt"}, {"source": "mydata.txt"}],
    ids=["delete_doc1", "delete_doc2"],
)
#
```

```
# 삭제할 id 를 확인
```

```
print(ids)
# ['delete_doc1', 'delete_doc2']
```

- `delete` = `ids` 입력해서 삭제 가능

```
# id 로 삭제
```

```
db5.delete(ids) # True
```

```
# 삭제된 결과 출력해보기
```

```
db5.index_to_docstore_id
```

- 셀 출력

```
{0: '4cdd0221-b9f7-4dc0-bc0c-4f4eaa3d0581',
1: '31e288f7-cb3f-4670-b1fb-8508b74a5deb',
2: '9f693bbc-2d28-4023-bb52-759d7f13cc77',
3: 'd52a9cfc-0535-4ac7-8baa-7df05c8eae9',
4: '04b13d9c-b9ec-47a8-bf22-dc80c9ef845d',
5: 'c7c12db7-0209-477b-a860-634c5e9bcfb8',
6: 'f7c90bdc-4882-459c-a0a2-ea028df48072',
7: 'db6a2852-1519-4191-b180-47893bc986ff',
8: '30791600-ac08-441e-91f0-883418070537',
9: '24cdb5b9-e433-4af1-ac0f-1387cbce7be4',
10: 'f65d3bc9-329d-4d79-a4e6-47dff4dd1780',
11: 'new_doc1',
12: 'new_doc2',
13: 'new_doc3'}
```

```
# 메모리 정리
```

```
del db5, embeddings
```

```
gc.collect()
```

```
# 4848
```

9) 저장 및 로드

① 로컬 저장 (Save Local)

- `save_local` 매서드: `FAISS` 인덱스, 문서 저장소, 인덱스-문서 `ID` 매핑을 로컬 디스크에 저장하는 기능 제공

- 매개변수

- `folder_path` (`str`): 저장할 폴더 경로

- `index_name` (`str`): 저장할 인덱스 파일 이름 (기본값: `"index"`)

- 저장 코드 예시

```
# 로컬 Disk 에 저장
```

```
db.save_local(folder_path="faiss_db", index_name="faiss_index")
```

- 동작 방식

- 지정된 폴더 경로 생성 (이미 존재하는 경우에는 무시)

- `FAISS` 인덱스를 별도의 파일로 저장함

- 문서 저장소, 인덱스-문서 `ID` 매핑을 `pickle` 형식으로 저장

- 사용 시 고려사항

- 저장 경로에 대한 쓰기 권한 필요

- 대용량 데이터의 경우 저장 공간과 시간이 상당히 소요될 수 있음

- `pickle` 사용으로 인한 보안 위험 고려해야 함

② 로컬에서 불러오기 (Load Local)

- `load_local` 매서드: 로컬 디스크에 저장된 `FAISS` 인덱스, 문서 저장소, 인덱스-문서 `ID` 매핑을 불러오는 기능

- **매개변수**
 - **folder_path** (str): 불러올 파일들이 저장된 폴더 경로
 - **embeddings** (Embeddings): 쿼리 생성에 사용할 임베딩 객체
 - **index_name** (str): 불러올 인덱스 파일 이름 (기본값: **"index"**)
 - **allow_dangerous_deserialization** (bool): pickle 파일 역직렬화 허용 여부 (기본값: **False**)
- **반환값**
 - FAISS: 로드된 FAISS 객체
- **동작 방식**
 - 역직렬화의 위험성 확인 → 사용자의 명시적 허가 요구
 - FAISS 인덱스를 별도로 불러옴
 - pickle 사용 → 문서 저장소, 인덱스-문서 ID 매핑 불러옴
 - 불러온 데이터로 FAISS 객체 생성 → 반환

```
import gc
import numpy as np
import time
from langchain_huggingface import HuggingFaceEmbeddings
from langchain_community.vectorstores import FAISS
from langchain_core.documents import Document
import warnings

# 경고 무시
warnings.filterwarnings("ignore")

embeddings = HuggingFaceEmbeddings(
    model_name="sentence-transformers/all-MiniLM-L6-v2",
    model_kwargs={'device': 'cpu'},
    encode_kwargs={'normalize_embeddings': True}
)

# 임베딩
embeddings = embeddings

# 임베딩 차원 크기를 계산
dimension_size = len(embeddings.embed_query("hello world"))
print(dimension_size) # 384
```

```
import faiss
from langchain_community.docstore.in_memory import InMemoryDocstore
from langchain_community.vectorstores import FAISS

# FAISS 벡터 저장소 생성
local_db = FAISS(
    embedding_function=embeddings,
    index=faiss.IndexFlatL2(dimension_size),
    docstore=InMemoryDocstore(),
    index_to_docstore_id={},
)
```

```
local_db.docstore._dict # {}
```

```
# 문서로 내용 추가하기
```

```
local_db = FAISS.from_documents(documents=split_doc1, embedding=embeddings)
```

```
# 문서 저장소 ID 확인하기
```

```
local_db.index_to_docstore_id
```

- 셀 출력 (0.1s)

```
{0: 'f5198267-57c7-4eee-84ba-ea673fc98220',
1: 'c1074e40-a0c0-4102-bd27-0a34e3700c98',
2: 'afeae953-2e73-4a96-992f-14c6acff2d82',
3: 'bc68efd4-d978-426c-97d8-07dafbe81c40',
4: 'fac55cde-5c51-44af-808b-fdb953391c44',
5: '6ca5c317-3e99-4e4b-bc0a-ca67fde8cce1',
6: '87a59986-6b7f-4d4a-94fb-ccb1fe03ebbf',
7: '51831ac6-2500-4128-b4b2-fbd45f423215',
8: '69a9df76-3a65-4a86-9dec-3fb0c4936728',
```



```
9: 'b4446a40-93a4-4234-b328-a533435ecabe',
10: 'a0695914-1413-47af-a295-633fd642b14f'}
```

메타데이터로 유사도 검색해보기

query = "TF IDF 에 대하여 알려줘"

```
local_db.similarity_search(query, filter={"source":"../09_VectorStore/data/nlp-keywords.txt"}, k=2)
```

- 셀 출력

```
[Document(id='87a59986-6b7f-4d4a-94fb-ccb1fe03ebbf', metadata={'source': '../09_VectorStore/data/nlp-keywords.txt'},
Document(id='51831ac6-2500-4128-b4b2-fbd45f423215', metadata={'source': '../09_VectorStore/data/nlp-keywords.txt'})]
```

- 로컬에 저장해보기

```
local_db.save_local(
    folder_path="faiss_db",
    index_name="faiss_index",
)
```

- 로컬에서 불러와보기

```
# 저장된 데이터를 로드
loaded_db = FAISS.load_local(
    folder_path="faiss_db",
    index_name="faiss_index",
    embeddings=embeddings,
    allow_dangerous_deserialization=True,
)
```

로드된 데이터 확인해보기

```
loaded_db.index_to_docstore_id
```

- 셀 출력: 앞의 `local_db` 와 같음을 확인할 수 있음

```
{0: 'f5198267-57c7-4eee-84ba-ea673fc98220',
1: 'c1074e40-a0c0-4102-bd27-0a34e3700c98',
2: 'afeae953-2e73-4a96-992f-14c6acff2d82',
3: 'bc68efd4-d978-426c-97d8-07dafbe81c40',
4: 'fac55cde-5c51-44af-808b-fdb953391c44',
5: '6ca5c317-3e99-4e4b-bc0a-ca67fde8cce1',
6: '87a59986-6b7f-4d4a-94fb-ccb1fe03ebbf',
7: '51831ac6-2500-4128-b4b2-fbd45f423215',
8: '69a9df76-3a65-4a86-9dec-3fb0c4936728',
9: 'b4446a40-93a4-4234-b328-a533435ecabe',
10: 'a0695914-1413-47af-a295-633fd642b14f'}
```

③ FAISS 객체 병합 (Merge From)

- `merge_from` 메서드: 현재 `FAISS` 객체에 다른 `FAISS` 객체를 병합하는 기능 제공
- 매개변수
 - `target` (`FAISS`): 현재 객체에 병합할 대상 `FAISS` 객체
- 동작 방식
 - 문서 저장소의 병합 가능 여부 확인
 - 기존 인덱스의 길이를 기준으로 새로운 문서들의 인덱스를 설정함
 - `FAISS` 인덱스 병합
 - 대상 `FAISS` 객체의 문서와 `ID` 정보 추출
 - 추출한 정보를 현재 문서 저장소와 인덱스-문서 `ID` 매핑에 추가함

- **주요 특징**
 - 두 **FAISS** 객체의 인덱스, 문서 저장소, 인덱스-문서 **ID** 매핑을 모두 병합
 - 인덱스 번호의 연속성을 유지하면서 병합
 - 문서 저장소의 병합 가능 여부를 사전에 확인
- **주의사항**
 - 병합 대상 **FAISS** 객체와 현재 객체의 구조와 호환되어야 함
 - 중복 **ID** 처리에 주의해야 함 → *현재 구현에서는 중복 검사를 하지 않음*
 - 병합 과정에서 예외가 발생하면 부분적으로 병합된 상태가 될 수 있음

```
# 저장된 데이터를 로드
loaded_db = FAISS.load_local(
    folder_path="faiss_db",
    index_name="faiss_index",
    embeddings=embeddings,
    allow_dangerous_deserialization=True,
)
```

```
# 새로운 FAISS 벡터 저장소 생성
```

```
local_db2 = FAISS.from_documents(documents=split_doc2, embedding=embeddings)
```

```
# 새로운 데이터 확인하기
```

```
local_db2.index_to_docstore_id
```

- 새로운 벡터 저장소 생성 (0.2s)

```
{0: 'd855dc98-7f2d-4478-b232-af6d2f2c631a',
1: 'b7ff103c-7c58-454f-8d38-ba00ebb42896',
2: '405211e4-5b13-4996-8819-b381ce912281',
3: 'f5a508a5-ac98-4ce4-89a5-41d9abf4c08f',
4: '8f17791a-6de3-4571-aaef-f0b8adef6b04',
5: 'e21fa123-d2d4-4f81-8b83-8f41a67c4835'}
```

- **merge_from** → 2개의 db 병합

```
# local_db + local_db2 를 병합
```

```
local_db.merge_from(local_db2)
```

```
# 병합된 데이터 확인해보기
```

```
local_db.index_to_docstore_id
```

- **local_db** → **local_db2** 순서로 병합되었음을 확인할 수 있음

```
{0: '167f7575-1235-429a-9972-55d71243e6af',
1: '8d1e84e7-ace9-49f3-9cda-3c4065dc72b9',
2: 'a9f4d320-03b1-4874-84e6-31c64b014e99',
3: '0e2ef7c6-f3f0-4916-91e4-8b9589514b0f',
4: '095de3de-bb83-4481-a130-6b78d0633625',
5: '00a45a90-b43d-48f4-aede-39ec39f27ef7',
6: 'e191027e-8024-4e65-9393-7dd415129149',
7: '73910cc7-c610-403f-9df4-7820e8e9aa6a',
8: '78477432-027d-4a2e-840d-8644e5ccf897',
9: 'e7133236-886d-4de1-83e1-8962d17eb527',
10: '4ddfe6f0-324c-4b20-a9c1-cf84cd02a011',
11: 'd855dc98-7f2d-4478-b232-af6d2f2c631a',
12: 'b7ff103c-7c58-454f-8d38-ba00ebb42896',
13: '405211e4-5b13-4996-8819-b381ce912281',
14: 'f5a508a5-ac98-4ce4-89a5-41d9abf4c08f',
15: '8f17791a-6de3-4571-aaef-f0b8adef6b04',
16: 'e21fa123-d2d4-4f81-8b83-8f41a67c4835'}
```

10) 검색기로 변환 (as_retriever)

- **as_retriever** 메서드: 현재 벡터 저장소를 기반으로 **VectorStoreRetriever** 객체를 생성하는 기능 제공
- **매개변수**
 - **kwargs**: 검색 함수에 전달할 키워드 인자
 - **search_type** (Optional[str]): 검색 유형
 - **similarity**
 - **mmr**
 - **similarity_score_threshold**
 - **search_kwargs** (Optional[Dict]): 검색 함수에 전달할 추가 키워드 인자
- **반환값**
 - **VectorStoreRetriever**: 벡터 저장소 기반의 검색기 객체
- **주요 기능**
 - **다양한 검색 유형 지원**
 - **similarity**: 유사도 기반 검색 (**기본값**)
 - **mmr**: Maximal Marginal Relevance 검색
 - **similarity_score_threshold**: 임계값 기반 유사도 검색
 - **검색 매개변수 커스터마이징**
 - **k**: 반환할 문서 수
 - **score_threshold**: 유사도 점수 임계값
 - **fetch_k**: MMR 알고리즘에 전달할 문서 수
 - **lambda_mult**: MMR 다양성 조절 파라미터
 - **filter**: 문서 메타데이터 기반 필터링
- **사용 시 고려 사항**
 - 검색 유형과 매개변수를 적절히 선택 → 검색 결과의 품질과 다양성을 조절할 수 있음
 - 대규모 데이터셋에서는 **fetch_k**, **k** 값 조절 → 성능, 정확도의 균형 맞출 수 있음
 - 필터링 기능 활용 → 특정 조건에 맞는 문서만 검색 가능
- **최적화 팁**
 - **MMR** 검색 시 **fetch_k**를 높이고 **lambda_mult**를 조절 → 다양성, 관련성의 균형 맞출 수 있음
 - 임계값 기반 검색을 사용 → 높은 관련성을 가진 문서만 반환 가능
- **주의사항**
 - 부적절한 매개변수 설정 → 검색 성능 or 결과의 품질에 영향을 줄 수 있음
 - 대규모 데이터셋에서 높은 **k** 값 설정 → 검색 시간 증가시킬 수 있음
 - 기본 값으로 설정된 **4개 문서**를 유사도 검색을 수행해 조회함

```
# 새로운 FAISS 벡터 저장소 생성
new_db = FAISS.from_documents(
    documents=split_doc1 + split_doc2,
    embedding=embeddings
)
```

- 기본 검색기(**retriever**) → **4개의 문서** 반환

```
# 검색기로 변환
```

```
retriever = new_db.as_retriever()
```

```
# 검색 수행
retriever.invoke("Word2Vec 에 대하여 알려줘")
```

```
huggingface/tokenizers: The current process just got forked, after parallelism has already been used. Disabling parallelism to av
To disable this warning, you can either:
- Avoid using `tokenizers` before the fork if possible
- Explicitly set the environment variable TOKENIZERS_PARALLELISM=(true | false)
[Document(id='98fab90f-f479-4139-8bab-95b810a93099', metadata={'source': '../09_VectorStore/data/nlp-keywords.txt'},
page_content='정의: Word2Vec은 단어를 벡터 공간에 매핑하여 단어 간의 의미적 관계를 나타내는 자연어 처리 기술입니다. 이는 단어의 문맥적 유사성을 기반으로 벡터를
```

생성합니다.\n\n예시: Word2Vec 모델에서 "왕"과 "여왕"은 서로 가까운 위치에 벡터로 표현됩니다.\n\n연관키워드: 자연어 처리, 임베딩, 의미론적 유사성\n\nLLM (Large Language Model)\n\n\n정의: LLM은 대규모의 텍스트 데이터로 훈련된 큰 규모의 언어 모델을 의미합니다. 이러한 모델은 다양한 자연어 이해 및 생성 작업에 사용됩니다.\n\n예시: OpenAI의 GPT 시리즈는 대표적인 대규모 언어 모델입니다.\n\n연관키워드: 자연어 처리, 딥러닝, 텍스트 생성\n\n\nFAISS (Facebook AI Similarity Search)\n\n\n정의: FAISS는 페이스북에서 개발한 고속 유사성 검색 라이브러리로, 특히 대규모 벡터 집합에서 유사 벡터를 효과적으로 검색할 수 있도록 설계되었습니다.\n\n예시: 수백만 개의 이미지 벡터 중에서 비슷한 이미지를 빠르게 찾는 데 FAISS가 사용될 수 있습니다.\n\n연관키워드: 벡터 검색, 머신러닝, 데이터베이스 최적화\n\n\n0Open Source'),\n\nDocument(id='5fa23d3c-7eb6-4851-809a-e955e681a2ac', metadata={'source': '..\n\n/09_VectorStore/data/nlp-keywords.txt'},\n\npage_content='정의: 토큰라이저는 텍스트 데이터를 토큰으로 분할하는 도구입니다. 이는 자연어 처리에서 데이터를 전처리하는 데 사용됩니다.\n\n예시: "I love programming."이라는 문장을 ["I", "love", "programming", "."]으로 분할합니다.\n\n연관키워드: 토큰화, 자연어 처리, 구문 분석\n\n\nVectorStore\n\n\n\n정의: 벡터스토어는 벡터 형식으로 변환된 데이터를 저장하는 시스템입니다. 이는 검색, 분류 및 기타 데이터 분석 작업에 사용됩니다.\n\n예시: 단어 임베딩 벡터들을 데이터베이스에 저장하여 빠르게 접근할 수 있습니다.\n\n연관키워드: 임베딩, 데이터베이스, 벡터화\n\n\nSQL\n\n\n\n정의: SQL(Structured Query Language)은 데이터베이스에서 데이터를 관리하기 위한 프로그래밍 언어입니다. 데이터 조회, 수정, 삽입, 삭제 등 다양한 작업을 수행할 수 있습니다.\n\n예시: SELECT * FROM users WHERE age > 18;은 18세 이상의 사용자 정보를 조회합니다.\n\n연관키워드: 데이터베이스, 쿼리, 데이터 관리\n\n\n\nCSV'),\n\nDocument(id='5e097e85-0144-4a3c-9bc0-87257e52d45b', metadata={'source': '..\n\n/09_VectorStore/data/nlp-keywords.txt'},\n\npage_content='정의: Semantic Search\n\n\n\n정의: 의미론적 검색은 사용자의 질의를 단순한 키워드 매칭을 넘어서 그 의미를 파악하여 관련된 결과를 반환하는 검색 방식입니다.\n\n예시: 사용자가 "태양계 행성"이라고 검색하면, "목성", "화성" 등과 같이 관련된 행성에 대한 정보를 반환합니다.\n\n연관키워드: 자연어 처리, 검색 알고리즘, 데이터 마이닝\n\n\n\nEmbedding\n\n\n\n\n정의: 임베딩은 단어나 문장 같은 텍스트 데이터를 저차원의 연속적인 벡터로 변환하는 과정입니다. 이를 통해 컴퓨터가 텍스트를 이해하고 처리할 수 있게 합니다.\n\n예시: "사과"라는 단어를 [0.65, -0.23, 0.17]과 같은 벡터로 표현합니다.\n\n연관키워드: 자연어 처리, 벡터화, 딥러닝\n\n\n\nToken\n\n\n\n\n정의: 토큰은 텍스트를 더 작은 단위로 분할하는 것을 의미합니다. 이는 일반적으로 단어, 문장, 또는 구절일 수 있습니다.\n\n예시: 문장 "나는 학교에 간다"를 "나는", "학교에", "간다"로 분할합니다.\n\n연관키워드: 토큰화, 자연어 처리, 구문 분석\n\n\n\nTokenizer'),\n\nDocument(id='62a1ee19-de34-4136-8e2e-54060dd1ee4f', metadata={'source': '..\n\n/09_VectorStore/data/nlp-keywords.txt'},\n\npage_content='정의: HuggingFace는 자연어 처리를 위한 다양한 사전 훈련된 모델과 도구를 제공하는 라이브러리입니다. 이는 연구자와 개발자들이 쉽게 NLP 작업을 수행할 수 있도록 돕습니다.\n\n예시: HuggingFace의 Transformers 라이브러리를 사용하여 감정 분석, 텍스트 생성 등의 작업을 수행할 수 있습니다.\n\n연관키워드: 자연어 처리, 딥러닝, 라이브러리\n\n\n\n\nDigital Transformation\n\n\n\n\n\n정의: 디지털 변환은 기술을 활용하여 기업의 서비스, 문화, 운영을 혁신하는 과정입니다. 이는 비즈니스 모델을 개선하고 디지털 기술을 통해 경쟁력을 높이는 데 중점을 둡니다.\n\n예시: 기업이 클라우드 컴퓨팅을 도입하여 데이터 저장과 처리를 혁신하는 것은 디지털 변환의 예입니다.\n\n연관키워드: 혁신, 기술, 비즈니스 모델\n\n\n\n\nCrawling\n\n\n\n\n\n정의: 크롤링은 자동화된 방식으로 웹 페이지를 방문하여 데이터를 수집하는 과정입니다. 이는 검색 엔진 최적화나 데이터 분석에 자주 사용됩니다.\n\n예시: 구글 검색 엔진이 인터넷 상의 웹사이트를 방문하여 콘텐츠를 수집하고 인덱싱하는 것이 크롤링입니다.\n\n연관키워드: 데이터 수집, 웹 스크래핑, 검색 엔진\n\n\n\n\nWord2Vec')]]

• 셀 출력 (0.2s)

```
[Document(id='98fab90f-f479-4139-8bab-95b810a93099', metadata={'source': '..\n\n/09_VectorStore/data/nlp-keywords.txt'},\n\nDocument(id='5fa23d3c-7eb6-4851-809a-e955e681a2ac', metadata={'source': '..\n\n/09_VectorStore/data/nlp-keywords.txt'},\n\nDocument(id='5e097e85-0144-4a3c-9bc0-87257e52d45b', metadata={'source': '..\n\n/09_VectorStore/data/nlp-keywords.txt'},\n\nDocument(id='62a1ee19-de34-4136-8e2e-54060dd1ee4f', metadata={'source': '..\n\n/09_VectorStore/data/nlp-keywords.txt'})]
```

• 다양성이 높은 더 많은 문서 검색해보기

- **k**: 반환할 문서 수 (*기본값: 4*)
- **fetch_k**: MMR 알고리즘에 전달할 문서 수 (*기본값: 20*)
- **lambda_mult**: MMR 결과의 다양성 조절 (*0~1, 기본값: 0.5*)

```
# MMR 검색 수행
retriever = new_db.as_retriever(
    search_type="mmr",                # 검색 유형
    search_kwargs={
        "k": 6,                       # 반환할 문서 수
        "lambda_mult": 0.25,          # MMR 결과의 다양성 설정하기
        "fetch_k": 10                 # MMR 알고리즘에 전달할 문서 수
    }
)

# 검색 결과 출력하기
retriever.invoke("Word2Vec 에 대하여 알려줘")
```

• 셀 출력

```
[Document(id='98fab90f-f479-4139-8bab-95b810a93099', metadata={'source': '..\n\n/09_VectorStore/data/nlp-keywords.txt'},\n\nDocument(id='5fa23d3c-7eb6-4851-809a-e955e681a2ac', metadata={'source': '..\n\n/09_VectorStore/data/nlp-keywords.txt'},\n\nDocument(id='62a1ee19-de34-4136-8e2e-54060dd1ee4f', metadata={'source': '..\n\n/09_VectorStore/data/nlp-keywords.txt'},\n\nDocument(id='b6e298a9-29a1-4fd5-961e-dc722597cadb', metadata={'source': '..\n\n/09_VectorStore/data/finance-keywords.txt'},\n\nDocument(id='5e097e85-0144-4a3c-9bc0-87257e52d45b', metadata={'source': '..\n\n/09_VectorStore/data/nlp-keywords.txt'},\n\nDocument(id='2baa4888-9507-4171-a68d-476d967d769d', metadata={'source': '..\n\n/09_VectorStore/data/nlp-keywords.txt'})]
```

• MMR 알고리즘을 위해 더 많은 가져오되 상위 2개만 반환

```
# MMR 검색 수행, 상위 2개만 반환
retriever = new_db.as_retriever(
    search_type="mmr",                # 검색 유형: MMR
    search_kwargs={
        "k": 2,                       # 반환 값: 2개
        "fetch_k": 10                 # MMR 알고리즘에 전달할 문서 수: 10개
    }
)

retriever.invoke("Word2Vec 에 대하여 알려줘")
```

- 셀 출력 (0.1s)

```
[Document(id='98fab90f-f479-4139-8bab-95b810a93099', metadata={'source': '../09_VectorStore/data/nlp-keywords.txt'})  
Document(id='5fa23d3c-7eb6-4851-809a-e955e681a2ac', metadata={'source': '../09_VectorStore/data/nlp-keywords.txt'})]
```

- 특정 임계값 이상의 유사도를 가진 문서만 검색해보기

```
# 임계값 기반 검색 수행  
  
retriever = new_db.as_retriever(  
    search_type="similarity",          # 검색 유형: similarity  
)  
  
retriever.invoke("Word2Vec 에 대하여 알려줘")
```

- 셀 출력

```
[Document(id='98fab90f-f479-4139-8bab-95b810a93099', metadata={'source': '../09_VectorStore/data/nlp-keywords.txt'})  
Document(id='5fa23d3c-7eb6-4851-809a-e955e681a2ac', metadata={'source': '../09_VectorStore/data/nlp-keywords.txt'})  
Document(id='5e097e85-0144-4a3c-9bc0-87257e52d45b', metadata={'source': '../09_VectorStore/data/nlp-keywords.txt'})  
Document(id='62a1ee19-de34-4136-8e2e-54060dd1ee4f', metadata={'source': '../09_VectorStore/data/nlp-keywords.txt'})]
```

```
# 임계값 기반 검색 수행  
retriever = new_db.as_retriever(  
    search_type="similarity_score_threshold",          # 검색 유형: similarity_score_threshold  
    search_kwargs={"score_threshold": 0.2}  
)  
  
retriever.invoke("Word2Vec 에 대하여 알려줘")
```

- `score_threshold = 0.2` 일 때 값이 나옴

```
[Document(id='98fab90f-f479-4139-8bab-95b810a93099', metadata={'source': '../09_VectorStore/data/nlp-keywords.txt'})]
```

- 교재: `score_threshold = 0.8` → `No relevant docs were retrieved using the relevance score threshold 0.8`

```
[]
```

- `score_threshold < 0.3` → `No relevant docs were retrieved using the relevance score threshold 0.3` 라고 나옴

- 가장 유사한 단일 문서만 검색해보기

```
# k=1 로 설정하여 가장 유사한 문서만 검색  
retriever = new_db.as_retriever(search_kwargs={"k": 1})  
  
# 검색 결과  
retriever.invoke("Word2Vec 에 대하여 알려줘")
```

- 셀 출력 (0.1s)

```
[Document(id='98fab90f-f479-4139-8bab-95b810a93099', metadata={'source': '../09_VectorStore/data/nlp-keywords.txt'})]
```

- 특정 메타데이터 필터 적용해서 검색해보기

```
# 메타데이터 필터 적용  
retriever = new_db.as_retriever(  
    search_kwargs={  
        "filter": {"source": "../09_VectorStore/data/finance-keywords.txt"},  
        "k": 2}  
    )  
  
# 메타데이터 활용  
# 반환할 문서 수: 2개  
  
retriever.invoke("ESG 에 대하여 알려줘")
```

- 셀 출력

```
[Document(id='b6e298a9-29a1-4fd5-961e-dc722597cadb', metadata={'source': '../09_VectorStore/data/finance-keywords'}, embedding=[0.1, 0.2, 0.3]), Document(id='58670945-65c8-4f38-94b2-628193ac2f9b', metadata={'source': '../09_VectorStore/data/finance-keywords'}, embedding=[0.4, 0.5, 0.6])]
```

-
- next: **Pinecone**
-