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

---

3. `Ensemble Retriever`

1) `앙상블 검색기`

- `Ensembel Retriever`
  - 여러 검색기를 결합해 더 강력한 검색 결과를 제공하는 Langchain의 기능
  - 다양한 검색 알고리즘의 장점을 활용해 단일 알고리즘보다 더 나은 성능을 달성할 수 있음

- `주요 특징`
  - `1. 여러 검색기 통합` : 다양한 유형의 검색기를 입력 → 결과를 결합함
  - `2. 결과 재순위화` : `Reciprocal Rank Fusion` 알고리즘 사용 → 결과의 순위 조정
  - `3. 하이브리드 검색` : 주로 `sparse retriever` (예시: `BM25`)와 `dense retriever` (예시: `임베딩 유사도`)를 결합해 사용

- `장점`
  - `sparse retriever` : *`키워드 기반`* 검색에 효과적
  - `Dense retriever` : *`의미적 유사성 기반`* 검색에 효과적
    - 상호보완적 특성 → `EmsembleRetriever` = 다양한 검색 시나리오에서 향상된 성능 제공 가능

- *참고:* *`LangChain 공식 문서`*

---

2) `설정`

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

# API 키 정보 로드
load_dotenv()                                 # True
```

```
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}')")
    print(f"✅ LangSmith 프로젝트: '{langchain_project}'")
    print(f"✅ LangSmith API Key: {langchain_api_key_status}")
    print("  -> 이제 LangSmith 대시보드에서 이 프로젝트를 확인해 보세요.")
else:
    print("❌ LangSmith 추적이 완전히 활성화되지 않았습니다. 다음을 확인하세요:")
    if langchain_tracing_v2 != "true":
        print(f"  - LANGCHAIN_TRACING_V2가 'true'로 설정되어 있지 않습니다 (현재: '{langchain_tracing_v2}').")
    if not os.getenv('LANGCHAIN_API_KEY'):
        print("  - LANGCHAIN_API_KEY가 설정되어 있지 않습니다.")
    if not langchain_project:
        print("  - LANGCHAIN_PROJECT가 설정되어 있지 않습니다.")
```

```
--- 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.
```

- 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.
```

- `EnsembleRetriever` 초기화 → `BM25Retriever`, `FAISS` 검색기 결합
  - *각 검색기의 가중치 설정됨*

```
from langchain_community.vectorstores import FAISS
from langchain_huggingface import HuggingFaceEmbeddings
from langchain_text_splitters import CharacterTextSplitter
from langchain_community.document_loaders import TextLoader
from langchain.retrievers import BM25Retriever, EnsembleRetriever
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}
    )

# 1단계: Embeddings 사용
embeddings = embeddings

# 임베딩 차원 크기를 계산
dimension_size = len(embeddings.embed_query("hello world"))
print(dimension_size)                                                    # 384 (12.0s 소요)
print("✅ HuggingFaceEmbeddings 초기화 완료!")                            # ✅ HuggingFaceEmbeddings 초기화 완료!
```

```
384
✅ HuggingFaceEmbeddings 초기화 완료!
```

```
# 2단계: 샘플 문서 리스트

doc_list = [
    "I like apples",
    "I like apple company",
    "I like apple's iphone",
    "Apple is my favorite company",
    "I like apple's ipad",
    "I like apple's macbook",
]

print(doc_list)
```

- 셀 출력

```
['I like apples', 'I like apple company', "I like apple's iphone", 'Apple is my favorite company', "I like apple
```

```
# bm25 retriever와 faiss retriever를 초기화하기
bm25_retriever = BM25Retriever.from_texts(
    doc_list,
)
bm25_retriever.k = 1                                        # BM25Retriever의 검색 결과 개수 = 1

# 허깅페이스 임베딩 사용하기
embedding = embeddings

faiss_vectorstore = FAISS.from_texts(
    doc_list,
    embedding,
)
```

```
faiss_retriever = faiss_vectorstore.as_retriever(search_kwargs={"k": 1})
```

```
# 앙상블 retriever를 초기화하기
ensemble_retriever = EnsembleRetriever(
    retrievers=[bm25_retriever, faiss_retriever],
    weights=[0.7, 0.3],
)
```

- `ensemble_retriever` 객체의 `get_relevant_document()` 메서드 호출 → 관련성 높은 문서 검색

- `query_1`

```
# 5단계: 검색 결과 문서 가져오기

query = "my favorite fruit is apple"

ensemble_result = ensemble_retriever.invoke(query)
bm25_result = bm25_retriever.invoke(query)
faiss_result = faiss_retriever.invoke(query)
```

```
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)
```

```
# 6단계: 가져온 문서 출력하기

print("[Ensemble Retriever]")
for doc in ensemble_result:
    print(f"Content: {doc.page_content}")
    print()


print("[BM25 Retriever]")
for doc in bm25_result:
    print(f"Content: {doc.page_content}")
    print()


print("[FAISS Retriever]")
for doc in faiss_result:
    print(f"Content: {doc.page_content}")
    print()
```

- 셀 출력

```
[Ensemble Retriever]
Content: Apple is my favorite company

Content: I like apples

[BM25 Retriever]
Content: Apple is my favorite company

[FAISS Retriever]
Content: I like apples
```

- `query_2`

```
# 5단계: 검색 결과 문서 가져오기

query = "Apple company makes my favorite iphone"

ensemble_result = ensemble_retriever.invoke(query)
bm25_result = bm25_retriever.invoke(query)
faiss_result = faiss_retriever.invoke(query)


# 6단계: 가져온 문서 출력하기
```

```
print("[Ensemble Retriever]")
for doc in ensemble_result:
    print(f"Content: {doc.page_content}")
    print()

print("[BM25 Retriever]")
for doc in bm25_result:
    print(f"Content: {doc.page_content}")
    print()

print("[FAISS Retriever]")
for doc in faiss_result:
    print(f"Content: {doc.page_content}")
    print()
```

- 셀 출력 (0.1s)

```
[Ensemble Retriever]
Content: Apple is my favorite company

[BM25 Retriever]
Content: Apple is my favorite company

[FAISS Retriever]
Content: Apple is my favorite company
```

---

3) `런타임 Config 변경`

- **`ConfigurableField`** 클래스: 런타임에서 `retriever` 속성 변경 가능
  - **`weights`** 매개변수 = **`ConfigurableField`** 객체로 정의
    - 필드의 `ID` = **`ensemble_weights`**

```
from langchain_core.runnables import ConfigurableField

# 앙상블 검색기 설정하기
ensemble_retriever = EnsembleRetriever(
    retrievers=[bm25_retriever, faiss_retriever],       # 리트리버 목록 설정: bm25, faiss_retriever
).configurable_fields(
    weights = ConfigurableField(
        id="ensemble_weights",                          # 검색 매개변수의 고유 식별자 설정
        name="Ensemble Weights",                        # 검색 매개변수의 이름 설정
        description="Ensemble Weights",                 # 검색 매개변수에 대한 설명 작성하기
    )
)
```

- 검색: **`config`** 매개변수를 통해 검색 설정 지정
- **`ensemble_weights`** 옵션의 가중치 = **`[1, 0]`** → **`모든 검색의 결과의 가중치와 BM25 retriever에 더 많이 보여되도록 함`**

```
config = {"configurable": {"ensemble_weights": [1, 0]}}

# config 매개변수 사용 → 검색 설정 지정하기
docs = ensemble_retriever.invoke("my favorite fruit is apple", config=config)

# 검색 결과인 docs를 출력하기
docs
```

- 셀 출력

```
[Document(metadata={}, page_content='Apple is my favorite company'),
Document(id='dc412634-c1ab-47d1-bec3-f5245c65f832', metadata={}, page_content='I like apples')]
```

- 검색시 모든 검색 결과의 가중치가 **`FAISS retriever`에 많이 부여되도록 해보기**

```
config = {"configurable": {"ensemble_weights": [0, 1]}}

# config 매개변수 사용 → 검색 설정 지정하기
docs = ensemble_retriever.invoke("my favorite fruit is apple", config=config)
```

```
docs = ensemble_retriever.invoke("my favorite fruit is apple", config=config)

# 검색 결과인 docs를 출력하기
docs
```

- 셀 출력 (0.1s 소요)

```
[Document(id='dc412634-c1ab-47d1-bec3-f5245c65f832', metadata={}, page_content='I like apples'),
Document(metadata={}, page_content='Apple is my favorite company')]
```

- **`가중치에 따른 결과 해석하기`**
  - **`BM25 Retriever`** vs **`FAISS Retriever`**
    - **`BM25 Retriever`**
      - 주로 **`키워드 기반`** 통계적 검색 방식
      - 질의, 문서 간의 단어 일치도와 빈도를 바탕으로 순위를 매김 → **`질이와 정확히 일치하는 단어가 포함된 문서를 찾는데 효과적`**
    - **`FAISS Retriever`**
      - **`의미 (Sementic) 기반`** 검색 방식
      - 문서, 질의를 벡터로 변환 → 벡터 공간에서 거리가 가까운 문서 찾음 = 유사도가 높은 문서를 찾음
      - 키워드가 직접 일치하지 않더라도 **`의미상 비슷한`** **문서를 찾는데 효과적**

- 결과 비교

Ensemble Retriever	가중치: BM25 > FAISS	가중치: BM25 < FAISS
코드	config = {"configurable": {"ensemble_weights": [1, 0]}}	config = {"configurable": {"ensemble_weights": [0, 1]}}
결과	[Document(metadata={}, page_content='Apple is my favorite company'), Document(id='dc412634-c1ab-47d1-bec3-f5245c65f832', metadata={}, page_content='I like apples')]	[Document(id='dc412634-c1ab-47d1-bec3-f5245c65f832', metadata={} Document(metadata={}, page_content='Apple is my favorite company')

- 해석

가중치 설정 (코드)	BM25 > FAISS ([1, 0])	BM25 < FAISS ([0, 1])
의미	BM25의 결과만 사용하고, FAISS의 결과는 무시함	FAISS의 결과만 사용하고, BM25의 결과는 무시함
순위 1위 문서	'Apple is my favorite company'	'I like apples'
해석	BM25가 이 문서를 질의에 가장 키워드적으로 관련 있다고 판단함	FAISS가 이 문서를 질의에 가장 의미적으로 관련 있다고 판단함

- **`앙상블 검색기의 역할`**
  - 두 검색기가 **`동일한 질의에 대해 다른 기준`** 으로 관련성을 판단 → **`다른 순위 부여할 수 있음`** 을 보여줌
  - 따라서 두 가중치(`BM25` =키워드 일치, `FAISS` =의미적 유사성)를 적절히 조합 (**`[0.5, 0.5]`** 혹은 **`[0.7, 0.3]`**)해서 사용할 때 장점이 드러남
  - 두 검색기의 결과가 다른 경우: 가중치에 따라 두 결과를 `합산` (`융합`) → `최종 순위 결정`

- *next:* ***`긴 문맥 재정렬 (LongContextReorder)`***
