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

10. 한글 형태소 분석기(Kiwi, Kkma, Okt) + BM25 검색기

1) 한글 형태소 분석기와 BM25Retriever의 결합

- 깔끔하게 출력결과를 확인하기 위한 함수를 정의하기

```
def pretty_print(docs):
    for i, doc in enumerate(docs):
        if "score" in doc.metadata:
            print(f"[{i+1}] {doc.page_content} ({doc.metadata['score']:.4f})")
        else:
            print(f"[{i+1}] {doc.page_content}")
```

2) Kiwi 토크나이저 사용 및 모델 만들어보기

- 사전에 `VS Code` 터미널에 설치할 것

```
pip install kiwipiepy                    # Kiwi 토크나이저 설치

python -m kiwipiepy download             # Kiwi 모델 다운로드
```

- 가상 환경을 사용하고 있다면 **해당 가상환경 활성화한 후 그곳에 설치할 것**

```
# kiwipiepy.Token 객체 구조 확인해보기

from kiwipiepy import Kiwi

kiwi = Kiwi()
tokens = kiwi.tokenize("안녕하세요, Kiwi 토크나이저를 사용해보겠습니다.")
for token in tokens:
    print(dir(token))
```

- kiwipiepy.Token 객체의 속성 확인

```
['__class__', '__delattr__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getit
['__class__', '__delattr__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getit
['__class__', '__delattr__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getit
['__class__', '__delattr__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getit
['__class__', '__delattr__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getit
['__class__', '__delattr__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getit
['__class__', '__delattr__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getit
['__class__', '__delattr__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getit
['__class__', '__delattr__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getit
['__class__', '__delattr__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getit
['__class__', '__delattr__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getit
['__class__', '__delattr__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getit
```

```
# Kiwi 기본 사용해보기

from kiwipiepy import Kiwi

print("=" * 60)
print("🚀 Kiwi 한글 형태소 분석기 테스트")
print("=" * 60)

# Kiwi 인스턴스 생성
kiwi = Kiwi()
```

```
# 텍스트 준비
text = "안녕하세요, Kiwi 토크나이저를 사용해보겠습니다."

print(f"📝 원본 텍스트: {text}")
print()

# 형태소 분석
tokens = kiwi.tokenize(text)

#print("🔍 형태소 분석 결과:")
#for i, token in enumerate(tokens):
#    print(f"[{i+1:2d}] {token.form:12s} | {token.tag:8s} | {token.prob:.4f}")   → prob 속성이 없어 오류 발생

print("🔍 형태소 분석 결과:")
for i, token in enumerate(tokens):
    print(f"[{i+1:2d}] {token.form:12s} | {token.tag:8s} | {token.score:.4f}")   # 속성 확인 후 score로 수정

print()
print("💡 주요 형태소만 추출:")
keywords = [token.form for token in tokens if token.tag in ['NNG', 'NNP', 'VV', 'VA']]
print(f"키워드: {keywords}")

print("\n✅ Kiwi 테스트 완료!")
```

- 형태소 분석 테스트

```
============================================================
🚀 Kiwi 한글 형태소 분석기 테스트
============================================================
📝 원본 텍스트: 안녕하세요, Kiwi 토크나이저를 사용해보겠습니다.

🔍 형태소 분석 결과:
[ 1] 안녕하세요         | NNP       | -24.9607
[ 2] ,                | SP        | -3.4208
[ 3] Kiwi             | SL        | -4.6044
[ 4] 토크나이저         | NNG       | -34.9031
[ 5] 를                | JKO       | -2.6550
[ 6] 사용              | NNG       | -6.3823
[ 7] 하                | XSV       | -1.0495
[ 8] 어                | EC        | -1.0495
[ 9] 보                | VX        | -2.1494
[10] 겠                | EP        | -3.5714
[11] 습니다            | EF        | -0.7610
[12] .                | SF        | -0.2585

💡 주요 형태소만 추출:
키워드: ['안녕하세요', '토크나이저', '사용']

✅ Kiwi 테스트 완료!
```

- **`Kiwi`** 의 `Score` = `log probability`
  - `Kiwi` = `log-likelihood` 방식 사용
  - 확률은 `0~1` 사이지만, `log` 를 취하면 `음수` 가 됨
    - 값이 `0` 에 가까울수록 `확률이 높음` **(좋음)**
    - 값이 `-무한대` 로 갈수록 `확률이 낮음` **(나쁨)**
    - 예시

```
확률 1.0   → log(1.0)   = 0.0      (최상)
확률 0.5   → log(0.5)   = -0.69    (중간)
확률 0.01  → log(0.01)  = -4.61    (낮음)
확률 0.0001 → log(0.0001) = -9.21   (매우 낮음)
```

- 위의 결과

```
[ 1] 안녕하세요    | NNP    | -24.9607  (낮음 - 고유명사로 잘못 분석)
[11] 습니다        | EF     | -0.7610   (높음 - 확실한 어미)
[12] .          | SF     | -0.2585   (매우 높음 - 확실한 구두점)
```

```
# → 코드 수정 필요
```

```
# 수정된 코드로 Kiwi 다시 사용해보기

from kiwipiepy import Kiwi

print("=" * 60)
print("🚀 Kiwi 한글 형태소 분석기 테스트")
print("=" * 60)

# Kiwi 인스턴스 생성
kiwi = Kiwi()

# 텍스트 준비
text = "안녕하세요, Kiwi 토크나이저를 사용해보겠습니다."

print(f"📝 원본 텍스트: {text}")
print()

# 형태소 분석
tokens = kiwi.tokenize(text)

print("🔍 형태소 분석 결과 (원본):")
print(f"{'번호':<4} {'형태소':<15} {'품사':<8} {'점수(로그확률)':<15}")
print("-" * 50)
for i, token in enumerate(tokens):
    print(f"[{i+1:2d}] {token.form:<15s} {token.tag:<8s} {token.score:>12.4f}")

print()

# 확률로 변환해서 보기
print("🔍 형태소 분석 결과 (확률 변환):")
print(f"{'번호':<4} {'형태소':<15} {'품사':<8} {'확률':<15}")
print("-" * 50)

import math
for i, token in enumerate(tokens):
    # log 확률 → 확률 변환
    probability = math.exp(token.score)
    print(f"[{i+1:2d}] {token.form:<15s} {token.tag:<8s} {probability:>12.8f}")

print()
print("💡 주요 형태소만 추출:")
keywords = [token.form for token in tokens if token.tag in ['NNG', 'NNP', 'VV', 'VA']]
print(f"키워드: {keywords}")

print()
print("📊 가장 확실한 토큰 TOP 3:")
# score 기준 정렬 (0에 가까울수록 확실)
sorted_tokens = sorted(tokens, key=lambda t: t.score, reverse=True)
for i, token in enumerate(sorted_tokens[:3]):
    prob = math.exp(token.score)
    print(f"[{i+1}] {token.form:<10s} | {token.tag:<6s} | score: {token.score:>8.4f} | prob: {prob:.6f}")

print("\n✅ Kiwi 테스트 완료!")
```

- 수정된 코드로 테스트한 `Kiwi test`

```
============================================================
🚀 Kiwi 한글 형태소 분석기 테스트
============================================================
📝 원본 텍스트: 안녕하세요, Kiwi 토크나이저를 사용해보겠습니다.

🔍 형태소 분석 결과 (원본):
번호   형태소           품사       점수(로그확률)
--------------------------------------------------
[ 1] 안녕하세요         NNP          -24.9607
[ 2] ,                SP           -3.4208
[ 3] Kiwi             SL           -4.6044
[ 4] 토크나이저         NNG          -34.9031
[ 5] 를                JKO          -2.6550
[ 6] 사용              NNG          -6.3823
[ 7] 하                XSV          -1.0495
[ 8] 어                EC           -1.0495
[ 9] 보                VX           -2.1494
[10] 겠                EP           -3.5714
[11] 습니다             EF           -0.7610
[12] .                SF           -0.2585
```

```
🔍 형태소 분석 결과 (확률 변환):
번호    형태소                 품사            확률
--------------------------------------------------
[ 1] 안녕하세요             NNP          0.00000000
[ 2] ,                     SP           0.03268630
[ 3] Kiwi                  SL           0.01000789
[ 4] 토크나이저              NNG          0.00000000
[ 5] 를                    JKO          0.07029892
[ 6] 사용                   NNG          0.00169127
[ 7] 하                    XSV          0.35009691
[ 8] 어                    EC           0.35009691
[ 9] 보                    VX           0.11655688
[10] 겠                    EP           0.02811701
[11] 습니다                 EF           0.46719469
[12] .                     SF           0.77223326

💡 주요 형태소만 추출:
키워드: ['안녕하세요', '토크나이저', '사용']

📊 가장 확실한 토큰 TOP 3:
[1] .            | SF       | score:  -0.2585   | prob: 0.772233
[2] 습니다         | EF       | score:  -0.7610   |  prob: 0.467195
[3] 하            | XSV      | score:  -1.0495   |  prob: 0.350097

✅ Kiwi 테스트 완료!
```

- `Score` 해석해보기
  - `로그 확률 (score)`의 의미

score 범위	의미	예시
0 ~ -1	매우 확실	구두점(.), 조사(를)
-1 ~ -5	확실	일반 동사, 명사
-5 ~ -10	보통	외래어, 복합어
-10 이하	불확실	미등록어, 드문 단어

- ◦ 위 결과 해석

```
-0.2585  (.)          →       99.7% 확신 (구두점은 확실함)
-0.7610  (습니다)      →       46.7% 확신 (어미는 확실함)
-24.9607 (안녕하세요)  →       0.0000...% (거의 불가능 – NNP로 분석한 게 잘못)
```

```python
# 실전: 신뢰도 기반 필터링

from kiwipiepy import Kiwi
import math

def analyze_with_confidence(text, confidence_threshold=-10.0):
    """
    신뢰도 기반 형태소 분석
    confidence_threshold: 이 값보다 높은(덜 음수인) 토큰만 반환
    """
    kiwi = Kiwi()
    tokens = kiwi.tokenize(text)

    # 신뢰도 높은 토큰만 필터링
    confident_tokens = [
        token for token in tokens
        if token.score > confidence_threshold                    # -10.0보다 크면 (덜 음수면)
        and token.tag in ['NNG', 'NNP', 'VV', 'VA', 'SL']        # 의미있는 품사
    ]

    print(f"📝 원본: {text}")
    print(f"✅ 신뢰도 높은 키워드 (threshold={confidence_threshold}):")

    for token in confident_tokens:
        prob = math.exp(token.score)
        print(f"  - {token.form:<12s} | {token.tag:<6s} | score: {token.score:>8.4f} | prob: {prob:.4f}")

    return [token.form for token in confident_tokens]
```

```
texts = [
    "Python은 AI 개발에 적합한 언어입니다.",
    "LangChain과 Kiwi를 활용한 검색 시스템",
    "형태소 분석은 자연어 처리의 기초입니다."
]
```

```
# 테스트
print("=" * 60)
print("🚀 신뢰도 기반 키워드 추출")
print("=" * 60)

for text in texts:
    keywords = analyze_with_confidence(text, confidence_threshold=-10.0)
    print(f"  → 키워드: {keywords}\n")

# 다른 속성도 추가해보기
print("🔍 Token 객체의 추가 속성:")
for token in tokens:
    print(f"\n형태소: {token.form}")
    print(f"  - tag: {token.tag}")                    # 품사 태그
    print(f"  - score: {token.score}")                # 로그 확률
    print(f"  - start: {token.start}")                # 시작 위치
    print(f"  - len: {token.len}")                    # 길이
    print(f"  - tagged_form: {token.tagged_form}")    # 형태소/품사

print("\n", "✅ 완료!")
```

- 신뢰도 기반 필터링 + 추가 요소 추출

```
============================================================
🚀 신뢰도 기반 키워드 추출
============================================================
📝 원본: Python은 AI 개발에 적합한 언어입니다.
✅ 신뢰도 높은 키워드 (threshold=-10.0):
- Python         | SL     | score:  -5.1598 | prob: 0.0057
- AI             | SL     | score:  -3.4959 | prob: 0.0303
- 개발           | NNG    | score:  -9.5165 | prob: 0.0001
- 적합           | NNG    | score:  -8.8320 | prob: 0.0001
- 언어           | NNG    | score:  -6.6020 | prob: 0.0014
→ 키워드: ['Python', 'AI', '개발', '적합', '언어']

📝 원본: LangChain과 Kiwi를 활용한 검색 시스템
✅ 신뢰도 높은 키워드 (threshold=-10.0):
- LangChain      | SL     | score:  -5.1598 | prob: 0.0057
- Kiwi           | SL     | score:  -0.4446 | prob: 0.6411
- 활용           | NNG    | score:  -5.6323 | prob: 0.0036
- 시스템         | NNG    | score:  -3.3437 | prob: 0.0353
→ 키워드: ['LangChain', 'Kiwi', '활용', '시스템']

📝 원본: 형태소 분석은 자연어 처리의 기초입니다.
✅ 신뢰도 높은 키워드 (threshold=-10.0):
- 분석           | NNG    | score:  -4.0561 | prob: 0.0173
- 기초           | NNG    | score:  -8.7458 | prob: 0.0002
→ 키워드: ['분석', '기초']

🔍 Token 객체의 추가 속성:

형태소: 안녕하세요
- tag: NNP
- score: -24.96066665649414
- start: 0
- len: 5
- tagged_form: 안녕하세요/NNP

형태소: ,
- tag: SP
- score: -3.4207992553710938
- start: 5
- len: 1
- tagged_form: ,/SP

형태소: Kiwi
- tag: SL
- score: -4.604381561279297
- start: 7
```

- len: 4
- tagged_form: Kiwi/SL

형태소: 토크나이저
- tag: NNG
- score: -34.90306091308594
- start: 12
- len: 5
- tagged_form: 토크나이저/NNG

형태소: 를
- tag: JKO
- score: -2.654998779296875
- start: 17
- len: 1
- tagged_form: 를/JKO

형태소: 사용
- tag: NNG
- score: -6.3822784423828125
- start: 19
- len: 2
- tagged_form: 사용/NNG

형태소: 하
- tag: XSV
- score: -1.0495452880859375
- start: 21
- len: 1
- tagged_form: 하/XSV

형태소: 어
- tag: EC
- score: -1.0495452880859375
- start: 21
- len: 1
- tagged_form: 어/EC

형태소: 보
- tag: VX
- score: -2.1493759155273438
- start: 22
- len: 1
- tagged_form: 보/VX

형태소: 겠
- tag: EP
- score: -3.571380615234375
- start: 23
- len: 1
- tagged_form: 겠/EP

형태소: 습니다
- tag: EF
- score: -0.7610092163085938
- start: 24
- len: 3
- tagged_form: 습니다/EF

형태소: .
- tag: SF
- score: -0.2584686279296875
- start: 27
- len: 1
- tagged_form: ./SF

✅ 완료!

- 재사용 가능한 클래스 모델로 만들기

- `클래스 모델` - 핵심 기능

메서드	설명	사용 예시
analyze()	기본 형태소 분석	`analyzer.analyze(text)`
extract_keywords()	신뢰도 기반 키워드 추출	`analyzer.extract_keywords(text, -10.0)`
extract_nouns()	명사만 추출	`analyzer.extract_nouns(text)`
extract_verbs()	동사만 추출	`analyzer.extract_verbs(text)`
extract_adjectives()	형용사만 추출	`analyzer.extract_adjectives(text)`
get_detailed_info()	상세 정보 조회	`analyzer.get_detailed_info(text)`
analyze_with_confidence()	통합 분석	`analyzer.analyze_with_confidence(text)`
print_analysis()	예쁜 출력	`analyzer.print_analysis(text, detailed=True)`

- ◦ 사용 예제
- ◦ ▪ a. 간단한 키워드 추출

```
from korean_analyzer import KoreanMorphologicalAnalyzer

analyzer = KoreanMorphologicalAnalyzer()

text = "Python과 LangChain을 활용한 AI 개발"
keywords = analyzer.extract_keywords(text)

print(keywords)
# ['Python', 'LangChain', '활용', 'AI', '개발']
```

- ◦ ▪ b. 문서 배치 처리

```
analyzer = KoreanMorphologicalAnalyzer()

documents = [
    "RAG 시스템 구축 방법",
    "형태소 분석의 중요성",
    "한국어 자연어 처리 기술"
]

all_keywords = []
for doc in documents:
    keywords = analyzer.extract_keywords(doc, confidence_threshold=-8.0)
    all_keywords.extend(keywords)

print(f"전체 키워드: {set(all_keywords)}")
```

- ◦ ▪ c. 품사별 분리 처리

```
analyzer = KoreanMorphologicalAnalyzer()

text = "Python은 강력하고 유용한 프로그래밍 언어입니다."

print(f"명사: {analyzer.extract_nouns(text)}")
print(f"동사: {analyzer.extract_verbs(text)}")
print(f"형용사: {analyzer.extract_adjectives(text)}")
```

- ◦ ▪ d. 파일로 저장 후 사용하기

```
# 10_Retriever/utils/korean_analyzer.py로 저장

from utils.korean_analyzer import KoreanMorphologicalAnalyzer

analyzer = KoreanMorphologicalAnalyzer()
result = analyzer.extract_keywords("안녕하세요")
```

```
# 10_Retriever/utils/Korean_analyzer.py 사용해보기
```

```
# 10_Retriever/utils/Korean_analyzer.py 사용해보기

from utils.korean_analyzer import KoreanMorphologicalAnalyzer

analyzer = KoreanMorphologicalAnalyzer()

text = "AI 기술이 빠르게 발전하고 있습니다."

print(f"명사: {analyzer.extract_nouns(text)}")
print(f"동사: {analyzer.extract_verbs(text)}")
print(f"형용사: {analyzer.extract_adjectives(text)}")
```

- 명사: ['기술', '발전']
- 동사: []
- 형용사: ['빠르']

- test_2 (`8.1s`)

```
============================================================
📝 원본: Python은 강력한 언어입니다
============================================================

번호   형태소              품사     품사설명            점수          확률
----------------------------------------------------------------------
[ 1] Python              SL      외국어              -5.1598     0.005743
[ 2] 은                  JX       기타               -1.5430     0.213736
[ 3] 강력                XR      어근               -8.0364     0.000323
[ 4] 하                  XSA     형용사 파생 접미사    -0.2585     0.772234
[ 5] ㄴ                  ETM      기타              -0.2585      0.772234
[ 6] 언어               NNG     일반 명사           -6.6236     0.001329
[ 7] 이                 VCP      기타              -2.7300     0.065219
[ 8] ㅂ니다              EF      종결 어미           -4.1887     0.015165

💡 요약:
- 총 토큰: 8개
- 명사: ['언어']
- 동사: []
```

```
# 10_Retriever/utils/Korean_analyzer.py 사용해보기

from utils.korean_analyzer import KoreanMorphologicalAnalyzer
analyzer = KoreanMorphologicalAnalyzer()
analyzer.print_analysis("Python은 강력한 언어입니다", detailed=True)
```

---

3) `Kiwi Analyzer + BM25 Retriever`

- `Kiwi Analyzer` + `BM25 Retriever` 결합 → `korean_bm25_retriever`

- `/10_Retriever/utils/korean_bm25_retriever.py` 실행 결과

```
/10_Retriever/utils/korean_bm25_retriever.py
============================================================
🚀 예제 1: 기본 BM25 검색
============================================================

🔍 검색어: '금융보험'

📊 검색 결과:
[1] 금융보험은 장기적인 자산 관리와 위험 대비를 목적으로 고안된 금융 상품입니다.
[2] 금융저축산물보험은 장기적인 저축 목적과 더불어, 축산물 제공 기능을 갖추고 있는 특별 금융 상품입니다.
[3] 금융단폭격보험은 저축은 커녕 위험 대비에 초점을 맞춘 상품입니다. 높은 위험을 감수하고자 하는 고객에게 적합합니다.
[4] 금융보씨 험한말 좀 하지마시고, 저축이나 좀 하시던가요. 뭐가 그리 급하신지 모르겠네요.

============================================================
🚀 예제 2: 점수 포함 검색
============================================================

🔍 검색어: '금융보험'
```

```
📊 점수 포함 결과:
[1] 금융보험은 장기적인 자산 관리와 위험 대비를 목적으로 고안된 금융 상품입니다. (1.0000)
[2] 금융저축산물보험은 장기적인 저축 목적과 더불어, 축산물 제공 기능을 갖추고 있는 특별 금융 상품입니다. (1.0000)
[3] 금융단폭격보험은 저축은 커녕 위험 대비에 초점을 맞춘 상품입니다. 높은 위험을 감수하고자 하는 고객에게 적합합니다. (1.0000)
[4] 금융보씨 험한말 좀 하지마시고, 저축이나 좀 하시던가요. 뭐가 그리 급하신지 모르겠네요. (0.5000)

============================================================
🚀 예제 3: k 값 설정 (상위 2개만)
============================================================

🔍 검색어: '금융보험'

📊 상위 2개 결과:
[1] 금융보험은 장기적인 자산 관리와 위험 대비를 목적으로 고안된 금융 상품입니다. (1.0000)
[2] 금융저축산물보험은 장기적인 저축 목적과 더불어, 축산물 제공 기능을 갖추고 있는 특별 금융 상품입니다. (1.0000)

============================================================
🚀 예제 4: 기본 BM25 vs Kiwi BM25 비교
============================================================

🔍 검색어: '금융보험'

📊 Kiwi BM25 1위:
금융보험은 장기적인 자산 관리와 위험 대비를 목적으로 고안된 금융 상품입니다.

📊 기본 BM25 1위:
금융단폭격보험은 저축은 커녕 위험 대비에 초점을 맞춘 상품입니다. 높은 위험을 감수하고자 하는 고객에게 적합합니다.

💡 차이점:
✅ Kiwi가 형태소 분석으로 더 정확한 결과 반환!

============================================================
✅ 모든 예제 완료!
============================================================
```

```
from utils.korean_bm25_retriever import KoreanBM25Retriever, pretty_print

# 문서 생성
documents = [
    "Python은 AI 개발에 적합한 언어입니다.",
    "LangChain은 RAG 시스템 구축 프레임워크입니다.",]

# 검색기 생성하기
retriever = KoreanBM25Retriever.from_texts(
    documents,
    k=1                     # 상위 1개만 반환
)

# 검색
query = "AI 개발 도구"
results = retriever.search_with_score(query)

print(f"🔍 검색어: {query}")
print(f"📊 결과: {len(results)}개\n")

for i, doc in enumerate(results):
    score = doc.metadata.get('score', 0)
    print(f"[{i+1}] 점수: {score:.4f}")
    print(f"    {doc.page_content}\n")
```

- 문서로 검색해보기 (`0.9s`)

```
🔍 검색어: AI 개발 도구
📊 결과: 1개

[1] 점수: 0.0000
    LangChain은 RAG 시스템 구축 프레임워크입니다.
```

```
from utils.korean_bm25_retriever import KoreanBM25Retriever, pretty_print

# 교재 예시로 해보기

documents = [
    "금융보험은 장기적인 자산 관리와 위험 대비를 목적으로 고안된 금융 상품입니다.",
    "금융저축산물보험은 장기적인 저축 목적과 더불어, 축산물 제공 기능을 갖추고 있는 특별 금융 상품입니다.",
```

```
    "금융보씨 험한말 좀 하지마시고, 저축이나 좀 하시던가요. 뭐가 그리 급하신지 모르겠네요.",
    "금융단폭격보험은 저축은 커녕 위험 대비에 초점을 맞춘 상품입니다. 높은 위험을 감수하고자 하는 고객에게 적합합니다.",
]

# 검색기 생성하기
retriever = KoreanBM25Retriever.from_texts(
    documents,
    k=2                         # 상위 2개만 반환
)

# 검색
query = "금융보험"
results = retriever.search_with_score(query)

print(f"🔍 검색어: {query}")
print(f"📊 결과: {len(results)}개\n")

for i, doc in enumerate(results):
    score = doc.metadata.get('score', 0)
    print(f"[{i+1}] 점수: {score:.4f}")
    print(f"    {doc.page_content}\n")
```

- 교재 속 텍스트로 test (`0.1s`)

```
🔍 검색어: 금융보험
📊 결과: 2개

[1] 점수: 1.0000
    금융보험은 장기적인 자산 관리와 위험 대비를 목적으로 고안된 금융 상품입니다.

[2] 점수: 1.0000
    금융저축산물보험은 장기적인 저축 목적과 더불어, 축산물 제공 기능을 갖추고 있는 특별 금융 상품입니다.
```

---

4) `KonlPy` (`Kkma`, `Okt`) 사용한 `BM25Retriever`

```
from konlpy.tag import Kkma, Okt
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity

# 문서 준비
documents = [
    "안녕하세요, 한국어 문서 검색을 테스트합니다.",
    "이 문서는 한국어 문서 검색을 위한 예제입니다.",
    "한국어 문서 검색은 BM25 알고리즘을 사용합니다."
]

# Kkma 형태소 분석기 사용
kkma = Kkma()
kkma_tokens = [kkma.morphs(doc) for doc in documents]

# Okt 형태소 분석기 사용
okt = Okt()
okt_tokens = [okt.morphs(doc) for doc in documents]

# TF-IDF 벡터화
kkma_vectorizer = TfidfVectorizer()
kkma_tfidf = kkma_vectorizer.fit_transform([' '.join(tokens) for tokens in kkma_tokens])

okt_vectorizer = TfidfVectorizer()
okt_tfidf = okt_vectorizer.fit_transform([' '.join(tokens) for tokens in okt_tokens])

# 쿼리 준비
query = "한국어 문서 검색"

# Kkma 형태소 분석기 사용
kkma_query_tokens = kkma.morphs(query)
kkma_query_tfidf = kkma_vectorizer.transform([' '.join(kkma_query_tokens)])

# Okt 형태소 분석기 사용
okt_query_tokens = okt.morphs(query)
okt_query_tfidf = okt_vectorizer.transform([' '.join(okt_query_tokens)])

# 코사인 유사도 계산
kkma_similarities = cosine_similarity(kkma_query_tfidf, kkma_tfidf)
okt_similarities = cosine_similarity(okt_query_tfidf, okt_tfidf)

# 결과 출력
```

```
print("KkmaBM25Retriever 결과:", [documents[i] for i in kkma_similarities.argsort()[0][::-1]])
print("OktBM25Retriever 결과:", [documents[i] for i in okt_similarities.argsort()[0][::-1]])
```

- `Kkma`, `Okt` (`14.1s`)

```
KkmaBM25Retriever 결과: ['이 문서는 한국어 문서 검색을 위한 예제입니다.', '안녕하세요, 한국어 문서 검색을 테스트합니다.', '한국어 문서 검색

OktBM25Retriever 결과: ['이 문서는 한국어 문서 검색을 위한 예제입니다.', '안녕하세요, 한국어 문서 검색을 테스트합니다.', '한국어 문서 검색은
```

```
from konlpy.tag import Kkma, Okt
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity

# 문서 준비
sample_texts = [
    "금융보험은 장기적인 자산 관리와 위험 대비를 목적으로 고안된 금융 상품입니다.",
    "금융저축산물보험은 장기적인 저축 목적과 더불어, 축산물 제공 기능을 갖추고 있는 특별 금융 상품입니다.",
    "금융보씨 험한말 좀 하지마시고, 저축이나 좀 하시던가요. 뭐가 그리 급하신지 모르겠네요.",
    "금융단폭격보험은 저축은 커녕 위험 대비에 초점을 맞춘 상품입니다. 높은 위험을 감수하고자 하는 고객에게 적합합니다.",
]

# Kkma 형태소 분석기 사용
kkma = Kkma()
kkma_tokens = [kkma.morphs(doc) for doc in documents]

# Okt 형태소 분석기 사용
okt = Okt()
okt_tokens = [okt.morphs(doc) for doc in documents]

# TF-IDF 벡터화
kkma_vectorizer = TfidfVectorizer()
kkma_tfidf = kkma_vectorizer.fit_transform([' '.join(tokens) for tokens in kkma_tokens])

okt_vectorizer = TfidfVectorizer()
okt_tfidf = okt_vectorizer.fit_transform([' '.join(tokens) for tokens in okt_tokens])

# 쿼리 준비
query = "금융보험"

# Kkma 형태소 분석기 사용
kkma_query_tokens = kkma.morphs(query)
kkma_query_tfidf = kkma_vectorizer.transform([' '.join(kkma_query_tokens)])

# Okt 형태소 분석기 사용
okt_query_tokens = okt.morphs(query)
okt_query_tfidf = okt_vectorizer.transform([' '.join(okt_query_tokens)])

# 코사인 유사도 계산
kkma_similarities = cosine_similarity(kkma_query_tfidf, kkma_tfidf)
okt_similarities = cosine_similarity(okt_query_tfidf, okt_tfidf)

# 결과 출력
print("KkmaBM25Retriever 결과:", [documents[i] for i in kkma_similarities.argsort()[0][::-1]])
print("OktBM25Retriever 결과:", [documents[i] for i in okt_similarities.argsort()[0][::-1]])
```

- 교재 속 텍스트로 테스트해보기 (`1.5s`)

```
KkmaBM25Retriever 결과: ['금융보험은 장기적인 자산 관리와 위험 대비를 목적으로 고안된 금융 상품입니다.', '금융저축산물보험은 장기적인 저축 목

OktBM25Retriever 결과: ['금융보험은 장기적인 자산 관리와 위험 대비를 목적으로 고안된 금융 상품입니다.', '금융저축산물보험은 장기적인 저축 목적
```

- `BM25 Retriever` vs `kkma`, `okt`

구분	형태소 분석기 / 검색 도구	역할 (중학생 눈높이 설명)	장점 (Pros) 👍
`BM25 Retriever`	검색 알고리즘 (Retrieval Algorithm)	질문(Query)에 대해 가장 관련성이 높은 문서를 찾아주는 '똑똑한 도서관 사서' 역할	성능이 안정적이고 전통적인 검색 방법 중에서 매우 뛰어남 / 키워드
`kkma` (꼬꼬마)	형태소 분석기 (Morphological Analyzer)	한국어 문장을 아주 꼼꼼하고 규칙에 맞게 단어(형태소) 단위로 자르는 '정교한 칼' 역할	정확도가 높고, 문법적인 분석을 세밀하게 잘해서 학술적인 연구나
`okt` (Open Korean Text)	형태소 분석기 (Morphological Analyzer)	한국어 문장을 빠르고 유연하게 단어(어절) 단위로 자르는 '빠른 칼' 역할 / 카카오에서 만듦	처리 속도가 매우 빠르고, 인터넷 용어나 오타 같은 비표준어를 비교

- 쉬운 설명

- `BM25 Retriever` (≒ 사서)
  - 구글, 네이버에 검색했을 때 관련성 높은 결과를 보여주는 방식과 비슷
  - **`질문과 겹치는 단어가 많을수록`**, **`그 단어가 다른 문서에는 흔하지 않을수록`** 그 문서를 가장 중요한 것으로 뽑아줌
  - **`앙상블 검색`** = 최근에는 `BM25 Retreiver` + `Vector Search` 함께 사용 → 검색 성능 ↑
- `kkma`, `okt` (≒ `칼`의 종류)
  - **`형태소 분석기`** = 둘 다 컴퓨터가 한국어 문장의 뜻을 이해하도록 **`단어 단위로 쪼개주는 도구`**
  - `kkma` ≒ 문법 규칙을 철저히 따르는 **`학자`** → 느리지만 정확 / 논문 분석에 유용
  - `okt` ≒ 실용적, 빠른 **`현장 전문가`** → 인터넷 용어 빠르게 처리 가능 / 수많은 댓글, SNS 데이터 분석에 유용

- next: **`Convex Combination(CC) 적용된 앙상블 검색기 (EnsembleRetriever)`**
