ch10

CH10. 임베딩 모델로 데이터를 압축하기

RAG

Text를 컴퓨터에게 인지시켜줘야 저장/학습/검색 등이 가능함

임베딩 벡터(수치)로 변환해야 컴퓨터가 인지 가능함
(임베딩 : 데이터의 의미를 압축한 숫자 배열(벡터))

1.통계/어휘 기반 (BoW, TF-IDF): 단순하고, 의미 이해나 순서 고려 없음.

2.단어 임베딩 평균: 단어 수준의 의미 이해, 여전히 순서 고려 미흡.

3.RNN (LSTM/GRU): 딥러닝의 시작, 시퀀스 포착 가능, 장기 의존성 한계.

4.어텐션 메커니즘: 장기 의존성과 병렬 처리 가능성의 핵심 전환점.

5.트랜스포머: 아키텍처 혁명, 병렬 처리 및 전역적인 문맥 파악.

6.문장 임베딩을 위한 트랜스포머 파인튜닝 (SBERT, SimCSE, E5):
의미 유사성 작업을 위해 최적화되어 고품질 임베딩을 효율적으로 생성.

7. LLM 자체 임베딩: 범용 LLM의 막대한 지식과 표현력을 활용.

10장

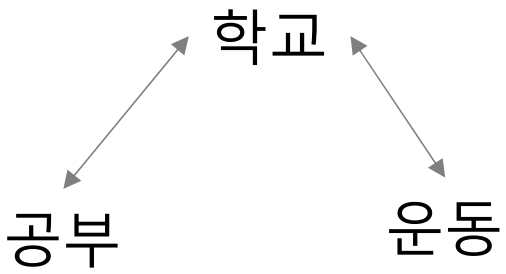
- ❑ 텍스트의 의미를 숫자로 변환하기 위한 다양한 기법 히스토리
- ❑ 문장 임베딩 방식 (바이정확도 / 교차연산량과 확장성 인코더)
- ❑ 문장 임베딩 모델을 활용해 의미 검색 방법 구현
  의미 검색은 구현 간단, 유사 검색 가능 but 정확도 낮을 수 있음
- ❑ 키워드 검색 방법 BM25
- ❑ 하이브리드 검색 구현 의미 검색 + 키워드 검색

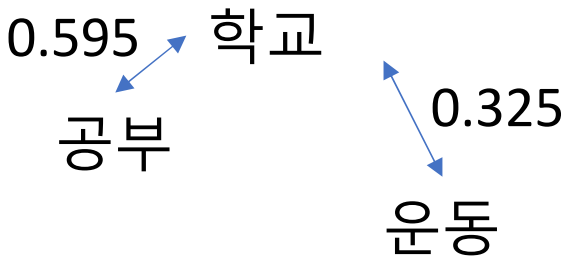
10.1 텍스트 임베딩 이해하기

텍스트 임베딩 : 여러 문장의 텍스트를 임베딩 벡터로 변환하는 방식
널리 사용되고 있지만 여전히 어려운 문제

❑ **문장 임베딩 활용**

- sentence_transformers 라이브러리 활용
- 단어나 문장 사이의 관계를 계산
- 텍스트 간 유사성, 관련성 판단 가능함





❑ **이전에는 어떤 방식으로 텍스트의 의미를 숫자로 표현하려고 했을까?**

- 넘버링
- 원핫인코딩
- 백오브워즈
- TF-IDF
- 워드투벡

10.1 텍스트 임베딩 이해하기

❑ 넘버링

- 학교 = 1, 공부 = 2, 운동 = 3
- 단점
  : 학교의 2배는 공부인가????

❑ 원핫 인코딩

- 학교 = [1,0,0], 공부 = [0,1,0], 운동 = [0,0,1]
- 단점
  : 단어 사이의 관계성 표현이 어려움
  : 학교와 공부의 유사도는 0

항목	원핫 인코딩	임베딩 벡터
의미 표현	없음 (단순 ID)	있음 (의미 기반)
벡터 유사도	대부분 0	유사도 기반 연산 가능
확장성	단어 수만큼 차원이 늘어남	고정된 저차원 벡터 사용
학습 가능성	불가능 (정적)	학습 가능 (의미 조정)

❑ 백오브워즈

- 비슷한 단어가 많이 나오면 비슷한 문장 or 문서로 정의
- 순서, 문법은 무시하고, 빈도수만 확인
- 장점
  : 구조와 연산이 간단하고, 직관적임
- 단점
  : 단어의 순서, 문맥이 고려 안됨
  : 단어의 의미를 충분히 이해하기 어려움

	가계대출	증시	AI	부동산	LLM	구글
경제 기사 1	3	3	0	2	0	0
경제 기사 2	0	5	3	0	0	0
IT 기사 1	0	0	3	0	4	2
IT 기사 2	0	0	2	1	2	0

경제+IT

경제 관련

IT 관련

10.1 텍스트 임베딩 이해하기

❑ **TF-IDF** (Term Frequency – Inverse Document Frequency)

- 단어의 빈도(TF)와 희귀성(IDF)을 함께 고려해서 중요한 단어를 찾는 기법
- 많은 문서에 등장하는 단어의 중요도를 낮춤

$$TF\text{-}IDF(w) = TF(w) \times \log(N/DF(w))$$

특정 단어가 여러 번 등장할수록 값이 작아짐

- TF : 특정 문서에 특정 단어의 등장 횟수
- N :전체 문서의 수
- DF("AI") : 특정 단어가 등장한 문서 수

	TF("이")	TF("LLM")	DF("이")	DF("LLM")	TF-IDF("이")	TF-IDF("LLM")
경제 기사 1	10	0	4	2	0	0
경제 기사 2	8	0	4	2	0	0
IT 기사 1	5	4	4	2	0	4 × log(4/2)
IT 기사 2	9	2	4	2	0	2 × log(4/2)

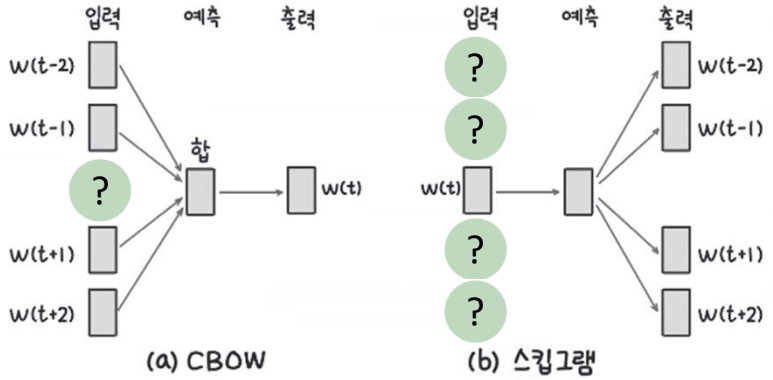
- 조사인 "이"는 "LLM"보다 자주 등장하지만, 모든 기사에 등장하므로 TF-IDF는 0으로 계산됨
- TF-IDF는 현재까지도 가장 보편적인 연관도 점수 계산 방식으로 사용 중
- 단점
  : 여전히 단어 순서/문맥 고려 안함, 문장 의미 파악 어려움

* 앞선 방식들은 사용된 단어 수만큼 차원이 늘어남
  대부분 값이 0이며, 단어의 의미를 압축해서 담지 못하므로 벡터간의 관계를 활용하기 어려움

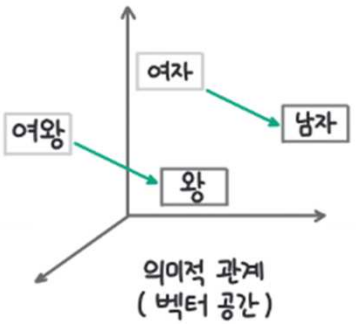
❑ **워드투벡** (Word2Vec, 인공신경망 등 활용)

- 단어의 의미를 벡터 공간에 담는 걸 목표로 함
- '함께 등장하는 빈도' 정보를 활용해서 의미를 압축함
  AI + ML, 한강 + 라면…
- 특정 단어 주변에 어떤 단어가 있는지 예측하는 모델
- 단점 : 연산시간, 문맥 이해x, 중의성 단어 구별 어려움

워드투벡의 두가지 학습 방식



워드투벡을 활용해서 단어 간 관계 계산



10.2 문장 임베딩 방식

여러 단어가 합쳐진 문장을 임베딩 벡터로 변환하는 방법 필요
→ 문장 사이의 유사도를 계산하는 두가지 방식

❑ 트랜스포머 인코더 구조를 활용한 BERT

(Bidirectional Encoder Representations from Transformers)

- 트랜스포머 인코더 구조
- 앞/뒤 문맥을 동시에 보면서 이해함
- 문장을 벡터로 바꿀 때, 의미를 잘 반영한 문장 임베딩을 만들어냄

- 장점

  문맥을 이해하는 능력이 뛰어남

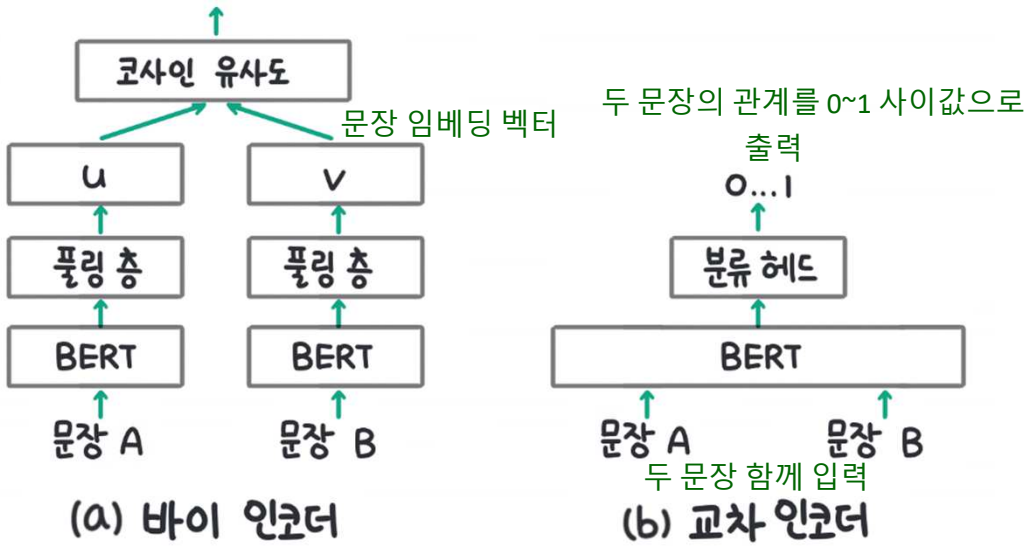
  fine-tuning 가능

❑ 문장 사이의 관계를 계산하는 두가지 방법

- 교차 인코더 방식 & 바이 인코더 방식
- 교차 인코더 방식의 장점

  : 두 문장 사이의 미세한 상호작용까지 학습 가능

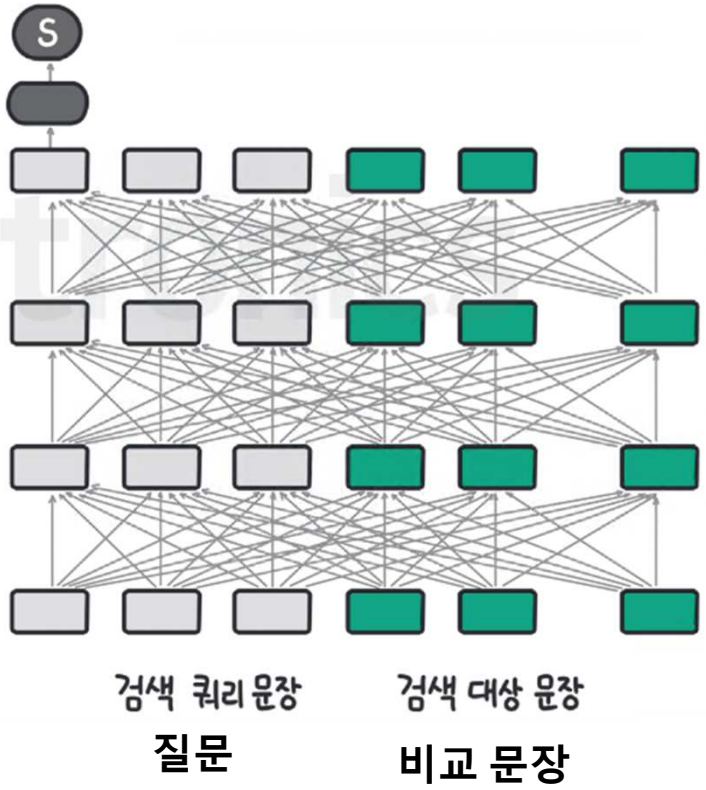
  : 정확도 매우 높음



10.2 문장 임베딩 방식

❑ 교차 인코더를 사용한 유사도 계산 프로세스

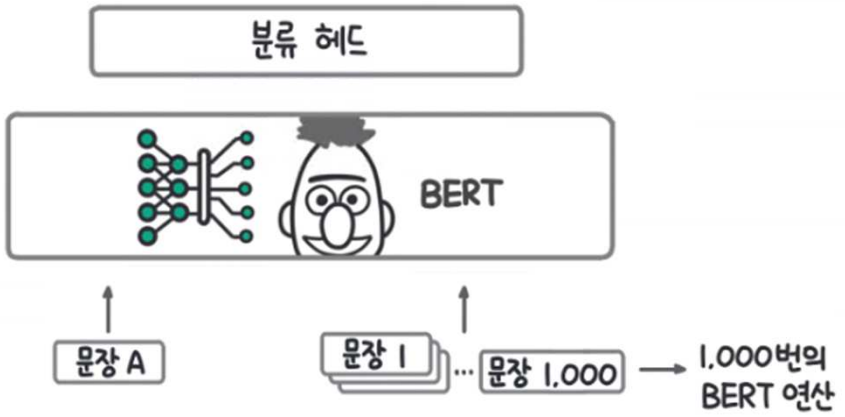
- 직접적으로 두 문장 인자의 관계를 모두 계산



❑ 교차 인코더 방식의 연산량

- 비교 문장이 10,000개이면 연산량 10,000번 실행
- 직접적인 연관성 추출 = 조건, 텍스트가 바뀌면 무의미
  즉, 질문이 변경되면 재학습이 필요함

문장 A와 유사한 문장을 찾으려면 저장된 문장 수만큼 연산해야함
= 비교적 정확하지만, 확장성이 떨어지는 방식



질문B가 새로 들어오면????

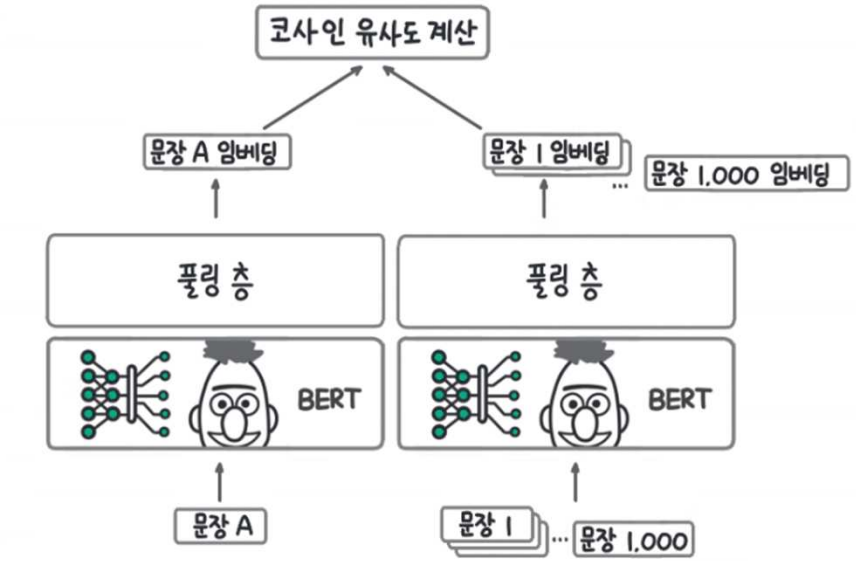
10.2 문장 임베딩 방식

❑ **바이 인코더를 활용한 문장 유사도 계산**

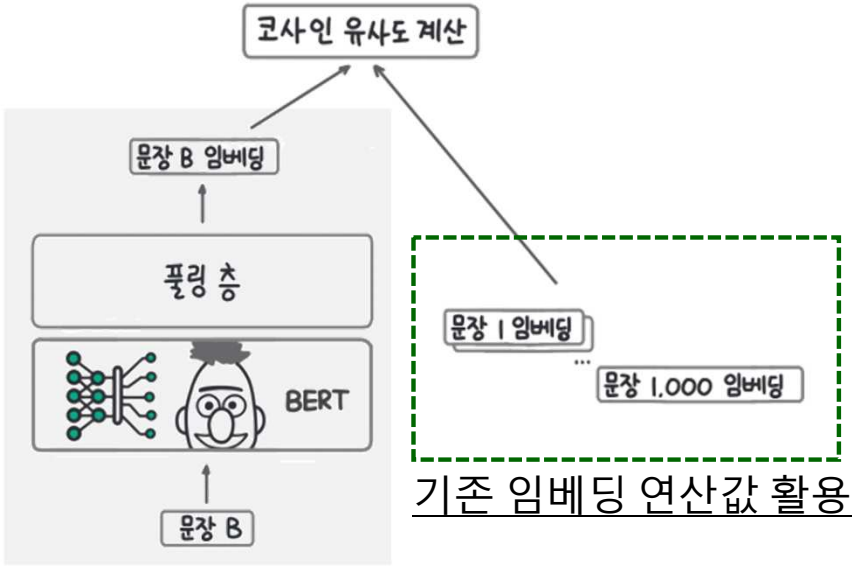
- 문장 A, 문장 B를 따로따로 임베딩한 후
  그 임베딩 벡터들 간의 코사인 유사도 등을 계산

- 장점
  : 문장 벡터를 미리 저장 → 빠른 비교 가능
  : 대규모 검색 시스템에 적합

바이 인코더에서 유사도 계산을 위한 임베딩



새로운 문장 B에 대한 유사도 계산 임베딩



기존 임베딩 연산값 활용

10.2 문장 임베딩 방식

정확도 vs 속도&확장성

항목	교차 인코더	바이 인코더
정확도	높음	중간~좋음
속도	느림 (매번 계산 필요)	빠름 (임베딩 미리 계산 가능)
문장 비교 방법	두 문장을 합쳐서 입력	각 문장을 따로 임베딩 후 비교
사용 예시	문장 관계 판단, 정밀 검색 등	대규모 검색, 문장 분류 등

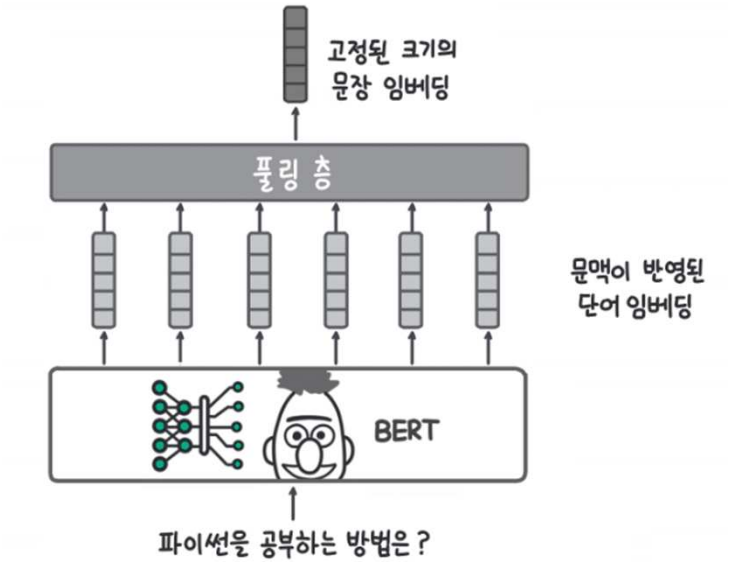
10.2 문장 임베딩 방식

❑ 바이 인코더 모델 구조

- 풀링층을 통해 비교하고자 하는 문장의 차원을 통일시킴
- 다양한 풀링 모드 중 평균 모드를 일반적으로 활용함

풀링 층을 통해 고정된 차원의 문장 임베딩 생성

입력되는 문장의 길이에 맞게 풀링층 설정



문장 임베딩 (1개의 768차원 벡터)

↑

Pooling (예: 평균 벡터)

↑

토큰별 벡터 (예: 768차원 × N개의 토큰)

↑

KLUE RoBERTa 모델 (Transformer)

↑

Tokenizer

↑

문장 입력

10.2 문장 임베딩 방식

❑ **(10.2.2) 바이 인코더 (코드 학습)**

- 텍스트
- 3가지 풀링 모드

❑ **(10.2.3) Sentence-Transformers로 텍스트와 이미지 임베딩 생성 (코드 학습)**

- 오픈소스 모델

10.3 실습: 의미 검색 구현하기

❑ 의미 검색 구현하기

-

❑ 라마인덱스에서 Sentence-Transformers 모델 사용하기

-

❑ 키워드 검색 vs 의미 검색

항목	키워드 검색	의미 검색 (Semantic Search)
검색 방식	정확한 단어 일치	의미와 문맥 기반
유사도 판단 기준	키워드 빈도수, 위치	단어 간 의미적 유사도
유의어 처리	불가능	가능
장점	빠르고 정확한 키워드 일치	문맥 이해, 자연스러운 결과
단점	유의어, 표현 다르면 검색 어려움	구현 복잡, 정확도 조절 어려움

의미 검색 방식의 한계와 극복 방안

1. 의미 검색 방식의 한계

① 문맥 및 세부 정보 손실 가능성

임베딩은 문장을 고정 길이 벡터로 압축하는 과정에서 일부 세부 의미나 문맥 정보가 손실될 수 있습니다.

예를 들어, "사과가 맛있다"와 "맛있는 사과"는 비슷하지만, 복잡한 문장은 차이가 미묘할 수 있음.

② 유사하지만 다른 의미 문장 혼동

벡터 거리가 가까운 문장이 반드시 정답이 아닐 수 있습니다.

예) "서울 날씨가 좋다" vs "부산 날씨가 좋다" → 날씨 관련 유사하지만 지역이 다름

③ 긴 문서와 짧은 쿼리 간 거리 문제

긴 문서 전체를 한 벡터로 만들면, 문서 내 여러 정보가 섞여 정밀한 검색이 어려워질 수 있음.

단락, 문장 단위 분할이 필요함

④ 도메인 특화 지식 부족

일반 임베딩 모델은 전문용어나 특정 도메인 의미를 잘 포착하지 못할 수 있음.

의료, 법률, 공학 분야에서 성능 저하

⑤ 계산 비용 및 인덱스 확장성 문제

대규모 데이터에서 벡터 유사도 계산 비용이 많이 들고, 인덱스 관리가 복잡함

의미 검색 방식의 한계와 극복 방안

2. 의미 검색 한계 극복 및 대책

① 문서 단위 세분화

문서 전체가 아닌 문장, 단락 단위로 임베딩하고 인덱싱

더 세밀하고 정확한 검색 결과 가능

② 도메인 특화 임베딩 사용

도메인 맞춤형 사전학습(pretrained) 모델 또는 파인튜닝된 임베딩 모델 사용

특정 분야 용어와 문맥을 잘 반영

③ 하이브리드 검색 도입

의미 검색 + 키워드 기반 전통 검색 결합

초기 후보군 필터링 후 의미 검색 적용 → 정확도 향상

④ 임베딩 정규화 및 유사도 방식 개선

코사인 유사도 사용 시 벡터 정규화(Normalization) 필수

유클리드 거리 대신 코사인 유사도 적용이 더 안정적

⑤ 검색 결과 후처리 및 재순위

유사도 점수 기반 후보군에 대해 별도 랭킹 알고리즘이나 LLM 활용해 재순위

⑥ 효율적 인덱스 및 검색 구조 활용

FAISS, Annoy, HNSW 같은 Approximate Nearest Neighbor(ANN) 인덱스 도입

대규모 데이터도 빠르게 검색 가능

의미 검색 방식의 한계와 극복 방안

한계점	대책
문맥 및 세부 정보 손실	문장/단락 단위 세분화 인덱싱
유사하지만 다른 의미 혼동	도메인 특화 임베딩, 후처리 랭킹
긴 문서 임베딩 문제	문서 쪼개기, 하이브리드 검색
도메인 지식 부족	도메인 맞춤형 모델 사용
계산 비용 및 확장성 문제	ANN 인덱스, 인프라 최적화

10.4 검색 방식을 조합해 성능 높이기

❑ **TF-IDP vs BM25**

- TF-IDF : 단어가 특정 문서에 얼마나 자주 등장하는가? + 단어가 전체 문서에서 얼마나 희귀한가?
- BM25는 - TF-IDF에 문서의 길이에 대한 가중치와 TF 포화를 추가한 방법론
  : 문서 길이 보정: 긴 문서일수록 불리하게 작용 (중복 키워드가 많을 수 있으므로)
  : TF 포화 함수 적용하여 단어가 아주 많이 나와도 점수가 일정 수준 이상은 안 올라감 (과도한 TF 상승 억제)

TF-IDF에서는 TF가 클수록 점수가 선형으로 계속 증가 (영향도 과다 측정 우려)
→ F(t,D)가 작을 때는 거의 선형 증가
→ F(t,D)가 클 때는점수 증가율이 둔화됨

$$\text{Score}(D,Q) = \sum_{t \in Q} IDF(t) \cdot \frac{f(t,D) \cdot (k_1 + 1)}{f(t,D) + k_1 \cdot (1 - b + b \cdot \frac{|D|}{\text{avgdl}})}$$

- $f(t,D)$: 단어 t의 문서 D에서의 등장 횟수
- $|D|$: 문서 D의 길이
- avgdl: 전체 문서의 평균 길이
- k_1, b: 조정 상수 (보통 $k_1 \in [1.2, 2.0]$, $b = 0.75$)

문장이 길면 그만큼 특정 단어 발생 확률이 높음
즉, 그 단어가 중요해서가 아니라 문장이 길어서 많이 등장
→ 문장 길이에 반비례

10.4 검색 방식을 조합해 성능 높이기

❑ 상호 순위 조합 (Reciprocal Rank Fusion, RRF)

- 서로 다른 방식으로 계산된 점수나 랭킹을 효과적으로 결합할 수 있는 기법
- 다양한 분포, 기준에도 결합이 다능하고, 계산이 단순함

문서	TF-IDF 점수	임베딩 유사도 (코사인 유사도 등)
D1	**0.89**	0.72
D2	0.55	**0.94**
D3	0.12	0.35

이 두 점수는 스케일(분포)이나 의미가 완전히 다르기 때문에
단순히 평균을 내거나 가중합을 하면 왜곡이 생길 수 있음

↓

점수가 아닌 "랭킹" 자체를 결합하는 방법

❑ 계산 방법

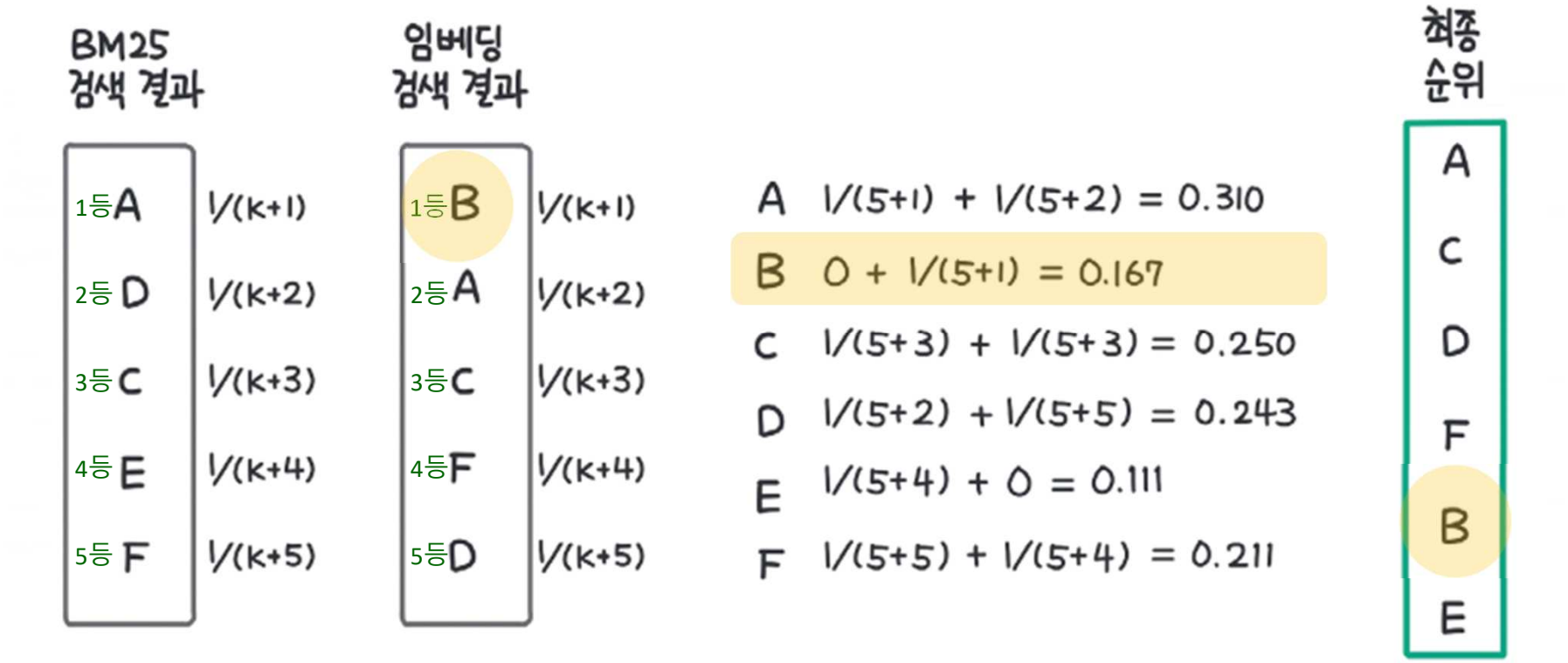
계산 순서1. 두 점수를 서로 독립적으로 랭킹(순위 매김)

계산 순서2. 각 문서가 받은 순위를 기반으로 점수를 다시 계산해서 결합 점수를 만듬

문서	통계 기반 순위	임베딩 기반 순위	RRF 점수 계산 (k=60으로 가정)
D1	1	3	1/61 + 1/63
D2	4	1	1/64 + 1/61
D3	5	5	1/65 + 1/65

← RRF 점수가 가장 높은 문서 D1이 최종 1위가 됨

10.4 검색 방식을 조합해 성능 높이기



10.5 실습: 하이브리드 검색 구현하기

❑ BM25 구현하기

문서	"안녕" 토큰 포함 여부	길이	점수
"안녕하세요"	포함 (토큰 공유)	짧음	0.447
"반갑습니다"	포함 안 됨	없음	0.000
"안녕 서울"	포함 (정확히 일치 가능)	2	0.524

- "안녕하세요"는 "안녕"과 유사한 의미이지만, 토큰화 결과가 다를 수 있어서 점수가 다소 낮음
- "반갑습니다"에는 "안녕" 관련 토큰이 없으므로 0점
- "안녕 서울"은 "안녕"이 정확히 들어 있으므로 가장 높음

1. 키워드 일치 기반 검색

BM25는 '언제', '이번 연도', '비', '많이', '올까' 같은 단어가 문서에 포함되면 점수를 부여하기 때문에
“언제”라는 단어가 있지만 질문과 연관성이 떨어져도 높은 점수를 줌

2. 의미 파악 불가능

BM25는 "비"라는 단어가 "장마"와 관련 있다는 것을 이해하지 못함
“장마” 문서가 “비”라는 단어를 포함하지 않는다면, 낮은 점수를 받을 수도 있음

3. 의미 기반 검색이 필요

"비가 많이 올까?" → "장마 시기", "강수량 예측", "기상청 전망" 등 의미 확장 필요

10.5 실습: 하이브리드 검색 구현하기

❑ 상호 순위 조합 구현하기

❑ 하이브리드 검색 구현하기

항목	설명
의미 이해	Dense 검색은 동의어, 문맥 유사성에 강함
키워드 정밀	BM25는 정확한 키워드 일치에 강함
융합(RRF)	서로 다른 모델의 순위를 균형 있게 결합하여 정확도와 다양성 모두 확보