IR Programming Assignment Report

- Boolean Retrieval Model과 Ranked Retrieval Model의 비교 -

Ⅰ. 서론

정보의 폭발적인 증가로 인해 우리는 수많은 데이터와 문서에 직면하게 되었다. 이러한 상황에서 효율적인 정보의 검색은 핵심적인 요구사항이 되었다. 인덱싱과 검색 시스템은 이러한 요구에 부응하여 정보를 구조화하고 효과적으로 검색할 수 있는 기능을 제공한다. 이를 통해 사용자는 복잡한 데이터 집합에서 원하는 정보를 신속하게 찾을 수 있으며, 조직은 데이터 관리와 탐색을 효율적으로 수행할 수 있다. 따라서 인덱싱과 검색 시스템은 현대 사회에서 정보 탐색과 데이터 관리에 있어서 매우 중요한 역할을 하며, 이 주제에 대한 연구와 이해는 정보 탐색과 데이터 관리에 관심이 있는 사람들에게 매우 중요하다.

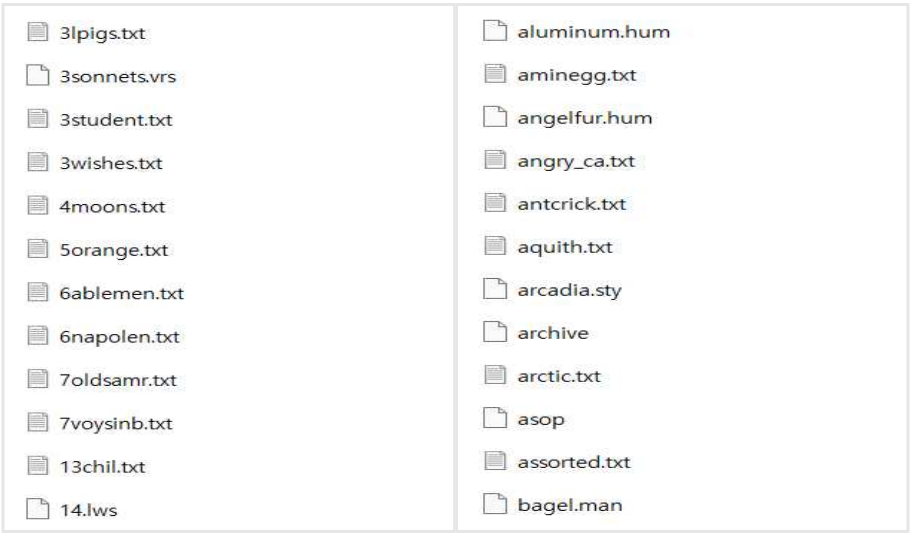
정보 검색에서 사용되는 두 가지 주요한 검색 모델은 Boolean retrieval model과 Ranked retrieval model이 있고, 두 모델은 문서 검색을 위한 접근 방식에 차이가 있다. 본 레포트에서는 두 정보 검색 모델을 비교하여 각 모델의 특징과 검색 결과를 비교하여 그 차이를 이해하고, 사용자 요구 상황에 따른 유용한 정보 검색 모델에 선택에 대한 정보를 제공하고자 한다.

또한 본 레포트의 주요 목적으로는 간단한 인덱싱과 검색 시스템에 대한 이해를 높이고, 해당 시스템의 설계, 구현, 기능, 성능 평가 등에 대한 통찰력을 제공하는 것이다. 한편 이 레포트는 기본적인 인덱싱과 검색 시스템에 대한 개념과 구현 방법을 다루며 보다 복잡한 기술이나 고급 주제는 다루지 않으며, 구축되는 시스템은 단순한 형태의 인덱싱 및 검색 시스템으로 제한한다.

정보 검색 모델 구현을 위해 http://archives.textfiles.com/stories.zip에서 제공된 텍스트 데이터 셋을 사용하였다. 이 데이터 셋은 영어로 작성된 다양한 이야기와 글들을 포함하고 있으며, 이 데이터들을 통해 인덱싱 및 검색 시스템을 구축하고 평가하는데 활용하고자 한다.
해당 데이터 셋은 압축 파일인 stories.zip으로 제공되었으며, 압축을 해제하면 다양한 텍스트 형식의 이야기들이 포함되어 있다. 데이터에는 제목과 본문 등의 정보가 들어있어 각 이야기를 식별하고 구성 요소별로 인덱싱하여 검색 가능한 형태로 만들 수 있다.

한편, 분석을 위해 데이터 셋에서 HTML로 형식화된 형태의 파일 등과 같은 일부 데이터를 제거하였다.

[그림1] stories.zip 파일 목록 예시



stories.zip 폴더의 경로에 있는 텍스트 문서 파일들을 불러왔으며, 각 파일에 대해 데이터 전처리를 수행하였다. 이때 문장 부호 제거, 숫자 제거, 소문자 변환, 토큰화, 불용어 제거, 표제어 추출, 어간 추출 등의 전처리 과정을 수행했으며 이렇게 함으로써 문서들을 표준화하고 검색의 일관성을 높일 수 있었다.
한편 각 문서에 대해 고유한 문서ID를 부여했으며, 해당 문서의 전처리 및 토큰화된 데이터와 파일명을 각각 딕셔너리로 저장하였다. 총 455건의 문서가 수집됐으며 [표1]은 확장자별 문서 수를 나타내는 통계표이다. 이때 txt 파일 확장자가 249건으로 가장 많은 빈도를 보였다.

[표1] stories.zip 파일 확장자별 문서 수

파일 확장자	문서 수
txt	249
hum	25
asc	6
poe	6
lws	6
이외	4건 이하

다음과 같은 구성으로 레포트를 진행하고자 하며, 이를 통해 검색 시스템의 기본 원리와 작동 방식을 이해하고 간단한 검색 시스템을 구축하는 데 필요한 개념과 기술을 학습할 수 있길 기대한다.

1. Unigram Inverted Index에 대한 내용과 SPIMI 알고리즘을 활용한 unigram inverted index 구현 방법에 대해서 소개합니다.
2. Boolean Retrieval Model에 대한 내용과 boolean query 작성법 및 unigram inverted index를 활용한 질의 방법에 대해서 소개합니다.
3. Ranked Retrieval Model에 대한 내용과 TF-IDF 계산 방법 및 이를 활용한 검색 결과의 순위화 방법에 대해서 소개합니다.
4. Boolean model과 Ranked model을 비교하며 상황에 따른 적절한 모델의 선택의 중요성에 대해서 논하고자 합니다.

본 리포트에서는 Unigram Inverted Index 구축, Boolean Retrieval 기능 구현, TF-IDF 스코어링을 활용한 Ranked Retrieval 기능 구현을 수행함으로써 간단한 인덱싱과 검색 시스템을 구축하고자 한다.

Ⅱ. Unigram Inverted Index

Inverted Index는 정보 검색에서 문서의 내용을 색인화하여 효율적인 검색을 가능하게 하는 자료 구조이다. 일반적으로, 단어와 해당 단어가 포함된 문서의 위치를 매핑한 데이터 구조이다.
이는 검색어 기반의 역방향 인덱스로서, 문서 검색을 빠르게 수행할 수 있도록 도와준다. Inverted index를 구축하면 검색어를 포함하는 문서를 찾는 데에 있어서 선형적인 검색 대신 효율적인 탐색을 가능하게 한다.

SPIMI(Single-Pass In-Memory Indexing) 알고리즘은 대용량 텍스트 문서를 처리하여 Inverted index를 구축하는 방법 중 하나이다. 이 알고리즘은 메모리 사용을 최적화하기 위해 모든 문서를 한 번에 올리지 않고, 작은 블록으로 나누어 처리한다. 따라서 SPIMI 알고리즘을 사용하여 대용량의 문서 집합을 효율적으로 다룰 수 있다. SPIMI 알고리즘의 핵심 단계는 다음과 같다.
1. 문서 집합을 블록 단위로 나눈다.
2. 각 블록을 메모리에 로드하고, 단어를 추출하여 inverted index를 구축한다.
3. 메모리에 저장된 inverted index 블록을 디스크에 기록한다.
4. 모든 블록에 대해 위 단계를 반복하고, 각 블록의 inverted index를 병합하여 최종적인 inverted index를 생성한다.

본 레포트에서 SPIMI 알고리즘을 이용하여 Unigram inverted index를 구축하였다. 다만, 이 구현에서는 전체 인덱스와 데이터 셋을 메모리에 유지할 수 있다는 가정하에 진행되었다. 따라서 인덱스와 문서 집합을 디스크에 기록하지 않는다.
먼저, Inverted index 딕셔너리를 생성한다. 그리고 전처리 완료된 토큰화된 문서에 대해서, 문서의 특정 단어가 inverted index 딕셔너리에 없는 경우 단어를 key로 추가하고 문서 ID를 값으로 할당한다. 만약 단어가 inverted index 딕셔너리에 있는 경우 문서 ID를 해당 단어 key의 값으로 추가한다. 이러한 과정을 모든 문서에 대해서 수행하면 unigram inverted index가 구축되며, 이는 각 단어에 대해서 문서의 위치 정보를 기록한다.

구축된 unigram inverted index를 확인하기 위해 몇 가지 용어에 대한 index와 문서를 검증하고자 한다. 이를 통해 SPIMI 기반의 Unigram inverted index가 정확하게 구축되었는지 확인하고, 검색 결과의 일관성을 보장할 수 있다.

[표2] inverted index 단어별 문서 수 예시

단어	문서 수
time	418
mind	182
wish	156
...	
survey	37
everyday	16

위 절차에 따라 Unigram inverted index를 구축하였고, 효율적이고 정확한 문서 검색을 위한 기반을 마련하였다. 다음으로 구축한 inverted index를 활용하여 Boolean Query를 통해 문서 검색을 수행하는 방법에 대해서 소개하고자 한다.

Ⅲ. Boolean Retrieval Model

Boolean Retrieval Model은 정보 검색에서 가장 간단하고 직관적인 모델 중 하나이다. 이 모델은 검색 결과를 참 또는 거짓으로 나타내는 불리언 쿼리(boolean query)를 사용하여 문서를 검색한다. 쿼리는 불리언 연산자(AND, OR, NOT)와 검색어로 구성된다. Boolean model은 주어진 쿼리와 정확히 일치하는 문서를 반환한다.

Boolean Query는 정보 검색에서 사용되는 질의 방법 중 하나로, 검색어와 boolean 연산자를 조합하여 문서를 검색하는 방식이다. 주로 AND, OR, NOT 연산자를 조합하여 쿼리를 작성한다.
1. AND 연산자는 주어진 두 개 이상의 용어를 모두 만족하는 문서를 검색하며, ‘X AND Y’로 작성한다. AND 연산자를 사용하면 문서의 범위를 좁혀 원하는 단어를 모두 포함하는 문서를 찾을 수 있다.
2. OR 연산자는 주어진 두 개 이상의 용어 중 하나 이상을 포함하는 문서를 검색하며, ‘X OR Y’로 작성한다. OR 연산자를 사용하면 여러 단어를 하나로 묶어 다양한 검색 결과를 얻을 수 있다.
3. NOT 연산자는 주어진 용어를 만족하지 않는 문서를 검색하며, ‘NOT X’로 작성한다. NOT 연산자는 단일 용어에 대해서만 적용되며, 특정 용어를 제외한 문서를 검색할 때 유용하게 사용될 수 있다.
4. AND NOT 연산자는 여러 개의 검색 조건을 결합하는데 사용된다. 이는 첫 번째 조건을 만족하면서, 동시에 나머지 조건을 만족하지 않는 문서를 검색한다. 예를 들어, ‘X AND NOT (Y OR Z)’의 질의는 X를 포함하면서 Y나 Z를 포함하지 않는 문서를 결과로 반환한다.
5. OR NOT 연산자도 여러 개의 검색 조건을 결합하는데 사용된다. 이는 첫 번째 조건을 만족하거나, 동시에 나머지 조건들을 만족하지 않는 문서를 검색한다. 예를 들어, ‘X OR NOT (Y AND Z)’의 질의는 X를 포함하거나 Y와 Z를 포함하지 않는 문서를 결과로 반환한다.

boolean query를 사용하여 문서를 검색할 때, 원하는 단어와 연산자를 조합하여 쿼리를 작성하고 이를 검색 모델에 전달한다. Boolean 검색 모델은 쿼리를 처리하여 해당 조건에 맞는 문서를 검색 결과롤 반환한다. 한편, 작성 시 단어 사이에 공백을 포함시켜야 한다는 점을 주의해야 하며, 괄호를 사용하여 연산자의 우선순위를 지정할 수 있다.

본 레포트에서는 boolean query 4가지에 대해서 SMIPI unigram inverted index를 활용하여 문서를 검색하는 함수를 구현했다. 이때 Boolean 검색 모델에서 inverted index를 활용하는 것은 다음과 같은 장점이 있다.

① 빠른 검색 속도

inverted index를 사용하면 단어가 포함된 문서를 빠르게 찾을 수 있다. 각 단어는 색인에 키로 매핑되고, 해당 키에는 해당 단어를 포함하는 문서 ID 또는 위치 정보가 저장되며, 이렇게 색인을 사용하면 검색 시 전체 문서를 탐색할 필요 없이 색인에 접근하여 검색 결과를 신속하게 얻을 수 있다.

② 메모리 효율성

inverted index는 단어의 출현 빈도와 상관없이 단어와 문서 간의 관계만을 저장합니다. 따라서 메모리를 효율적으로 사용할 수 있다. 특히, 대규모 문서 집합에서는 중복되는 용어가 많을 수 있기 때문에 inverted index는 메모리를 절약하는 데 도움이 된다.

③ 유연한 질의 기능

Boolean 검색 모델은 AND, OR, NOT 연산자를 사용하여 쿼리를 구성할 수 있는 간단하고 직관적인 방법으로 사용자가 쉽게 쿼리를 작성할 수 있다. 이를 통해 사용자는 검색 결과를 조정하고 원하는 문서를 세밀하게 필터링할 수 있어 효과적으로 검색하는 데 도움을 준다.

다음은 구현된 boolean 검색 모델의 검색 결과를 살펴보기 위한 예제로 16개의 문서에 포함된 “Everyday” 와 156개의 문서에 포함된 “Wish” 단어에 대한 4가지 boolean query 검색을 수행했으며, 그 결과는 [표3]~[표6]과 같다.

Boolean 검색은 Inverted Index를 효과적으로 활용하여 처리되며, 각 쿼리는 Inverted Index에서 논리적인 조합을 사용하여 해당 조건을 만족하는 문서를 찾는 방식으로 동작한다.

1. AND 연산자는 두 단어의 문서 ID의 교집합을 구하여 반환한다.
2. OR 연산자는 두 단어의 문서 ID의 합집합을 구하여 반환한다.
3. NOT 연산자는 오른쪽 단어가 포함된 문서 ID를 제외한 전체 문서 ID를 반환한다.

예제 쿼리 “Everyday Wish’ 를 각 boolean 검색 함수에 입력하면, 공백을 기준으로 쿼리를 토큰화하여 각 단어를 순회하면서 문서를 찾는다.

[표3] AND Boolean 검색 결과

Boolen Query	결과 건수	문서 ID	파일명
'Everyday' AND 'Wish'	11	130	darkness.txt
		131	day.n.mcdonald
		…	
		19	7voysinb.txt
		90	bulnoopt.txt

AND boolean 검색 함수는 첫 번째 단어가 inverted index에 있다면, 함수 결과 집합에 해당 단어의 문서ID 값을 넣는다. 그다음 두 번째 단어 또한 inverted index에 있다면, 해당 단어의 문서ID를 기존의 함수 결과 집합과 교집합을 구하여 결과 집합을 반환한다.

[표3] 예제의 AND 연산자는 전체 문서 중에서 'Everyday' 와 'Wish' 두 단어가 모두 포함된 문서를 찾는다. 'Everyday' 는 16개의 문서에 포함되고 'Wish' 는 156개의 문서에 포함된다. 이때 두 단어가 모두 포함된 문서를 찾는 AND 연산자 결과 문서는 총 11개로 나타났다.

[표4] OR Boolean 검색 결과

Boolen Query	결과 건수	문서 ID	파일명
'Everyday' OR 'Wish'	161	2	13chil.txt
		3	14.lws
		…	
		454	yukon.txt
		455	zombies.txt

OR boolean 검색 함수는 모든 단어에 대해서 inverted index에 있다면 함수 결과 집합에 문서ID 값을 넣는다. 최종적으로 중복된 문서ID 값을 처리한 후 결과 집합을 반환한다.

[표4] 예제의 OR 연산자는 전체 문서 중에서 'Everyday' 또는 'Wish' 두 단어가 포함된 문서를 찾는다. 'Everyday' 는 16개의 문서에 포함되고 'Wish' 는 156개의 문서에 포함되어 두 단어가 나타난 문서의 개수는 172개다. 그러나, 두 단어가 중복으로 포함된 문서가 11개가 있기 때문에 172개에서 11개의 문서를 제외해야 한다. 따라서 OR 연산자 결과 문서는 총 161개다.

[표5] AND NOT Boolean 검색 결과

Boolen Query	결과 건수	문서 ID	파일명
'Everyday' AND NOT 'Wish'	5	425	tinsoldr.txt
		426	toilet.s
		236	hotline3.txt
		45	assorted.txt
		46	bagel.man

AND NOT boolean 검색 함수는 두 단어가 모두 inverted index에 있다면, 첫 번째 단어가 포함된 문서ID 집합과 전체 문서ID 집합에서 두 번째 단어가 포함된 문서ID 집합을 제외한 결과 간의 교집합을 구하여 결과 집합을 반환한다. 그리고 첫 번째 단어만 inverted index에 존재하는 경우 첫 번째 단어가 포함된 문서ID 집합을 결과로 반환하고, 두 단어가 모두 inverted index에 존재하지 않는 경우 빈 결과 집합을 반환한다.

[표5] 예제의 AND NOT 연산자는 'Everyday'가 포함된 문서 중에서 'Wish'가 포함된 문서를 제외한 문서를 찾는다. 'Everyday'는 16개의 문서에 포함되고, 두 단어가 모두 포함된 문서는 11개이다. 따라서 16개에서 11개의 문서를 제외한 AND NOT 연산자 결과 문서는 총 5개다.

[표6] OR NOT Boolean 검색 결과

Boolen Query	결과 건수	문서 ID	파일명
'Everyday' OR NOT 'Wish'	310	1	100wes.txt
		10	31pigs.txt
		…	
		452	write
		453	wrt

OR NOT boolean 검색 함수는 두 단어가 모두 inverted index에 있다면, 첫 번째 단어가 포함된 문서ID 집합에서 두 번째 단어가 포함된 문서ID 집합을 제외한 결과를 반환한다. 그리고 inverted index에 첫 번째 단어만 존재하는 경우 첫 번째 단어가 포함된 문서ID 집합을 결과로 반환하고, 두 번째 단어만 존재하는 경우 전체 문서ID 집합에서 두 번째 문서가 포함된 문서ID 집합을 제외한 결과를 반환하고, 두 단어가 모두 존재하지 않는 경우 빈 결과 집합을 반환한다.

[표6] 예제의 OR NOT 연산자는 전체 문서 중에서 ‘Everyday’ 포함되거나 ‘Wish’ 가 포함되지 않는 문서를 찾는다. ‘Everyday’ 는 16개의 문서에 포함되고 ‘Wish’ 는 299개(455-156)의 문서에 포함되지 않아, ‘Everyday’ 가 포함되고 ‘Wish’ 가 포함되지 않은 문서는 315개다. 그러나 ‘Everyday’ 가 포함된 문서 중에서 ‘Wish’ 가 포함된 문서를 제외한 문서가 5개 있기 때문에 315개에서 5개의 문서를 제외해야 한다. 따라서 OR NOT 연산자의 결과 문서는 총 310개다.

예제 수행 결과 boolean retrieval model 검색 결과는 단순하고 직관적이며, 정확도가 높은 것으로 나타났다. 그러나 다음과 같은 단점이 있다.

① 부정확한 검색 결과

Boolean 검색 모델은 문서가 쿼리와 완전히 일치하는지 여부에만 기반하여 검색 결과를 반환하며, 이는 부정확한 결과를 초래할 수 있다. 예를 들어, 특정 단어를 포함하지 않는 문서를 제외하고자 할 때, 다른 중요한 단어를 포함하지 않는 문서가 제외되는 경우가 발생할 수 있다고 한다.

② 상세한 쿼리 작성 필요성

Boolean 검색 모델은 쿼리를 상세하게 작성해야 한다. 사용자가 모든 단어와 연산자를 정확하게 입력해야만 원하는 결과를 얻을 수 있으며, 이는 사용자가 검색 조건을 명확히 이해하고 쿼리를 올바르게 작성하는 것의 중요성을 강조한다.

③ 상호 연산자 제약

Boolean 검색 모델은 AND, OR, NOT 연산자를 사용하여 쿼리를 구성한다. 하지만 복잡한 쿼리를 표현하기에는 한계가 있다. 예를 들어, 더 복잡한 연산자(예: XOR, NEAR)를 사용하거나 다른 조건(예: 범위 검색)을 추가하려면 다른 검색 모델이 필요할 수 있다.

④ 검색의 유연성 제한

Boolean 검색 모델은 쿼리를 통해 문서를 필터링하는 데 적합하지만, 문서의 관련성 순위를 평가하는 데는 적합하지 않다. 따라서 검색 결과를 정확히 순위화하거나 사용자의 관심도에 따라 결과를 조정하는 유연성이 제한되며, 유연성이 필요한 경우 더 발전된 정보 검색 기법이 필요할 수 있다.

다음으로 Boolean Retrieval Model보다 검색 유연성이 높은 Ranked Retrieval model에 대해서 알아보고자 한다.

Ⅳ. Ranked Retrieval Model

Ranked retrieval model은 정보 검색에서 사용되는 모델로 사용자의 쿼리와 문서 간의 관련성을 평가하여 가장 관련성이 높은 문서를 상위에 랭크하는 방식이며, 대부분의 웹 검색 엔진과 정보 검색 시스템에서 사용된다. ranked 검색 모델의 목표는 사용자의 정보 요구에 최적화된 문서의 순위를 결정하는 것이다.

TAAT(Term-at-a-Time)는 검색 결과를 순위화하는 방법 중 하나로, 문서를 검색할 때 각 단어의 점수를 계산한 다음에 이를 문서 점수로 결합하여 검색 결과를 랭크하는 방법이다. TAAT 점수 계산 방법은 다음과 같다.
1. 쿼리에 포함된 모든 단어에 대해 각 문서의 점수를 계산한다.
2. 각 문서의 단어별 점수를 합산해 해당 문서의 총 점수로 나타낸다.
3. 문서의 총 점수에 따라 내림차순으로 정렬하여 결과를 반환한다.
TAAT 점수 계산 방법을 사용하면 쿼리와 문서 간의 상호 작용을 고려하여 검색 결과를 랭크할 수 있다. 이를 통해 쿼리에 가장 관련성이 높은 문서를 상위에 나타낼 수 있다.

한편 TAAT 방법은 기본적으로 단어와 문서의 빈도를 고려하는 TF-IDF(Term Frequency-Inverse Document Frequency) 점수를 사용하여 단어의 중요도를 평가한다. 단어의 빈도(TF)는 특정 문서에서 단어가 나타난 횟수를 나타내고, 문서의 빈도(IDF)는 전체 문서 집합에서 특정 단어가 나타난 문서의 수의 역수를 나타낸다. 이때 특정 단어가 더 많은 문서에서 나타나면 해당 단어의 IDF 값은 낮아지게 된다.

본 레포트에서는 자유 텍스트 쿼리에 대해서 TF-IDF 점수를 활용한 TAAT 랭크 검색 함수를 구현했다. 이때 Ranked retireval model에서 TF-IDF 점수를 활용하는 것은 다음과 같은 장점이 있다.
① 단어의 중요도 파악
TF-IDF 점수는 단어의 빈도와 문서의 빈도를 고려하여 단어의 중요도를 평가한다. 이를 통해 단어의 상대적인 중요도를 파악할 수 있다. 빈번하게 등장하는 단어는 해당 문서와 관련성이 높은 가능성이 높고, 드물게 등장하는 단어는 해당 문서와의 관련성이 낮을 가능성이 높다.
② 문맥 고려
TF-IDF 점수는 각 문서에서 단어의 빈도를 평가하기 때문에 문맥을 고려할 수 있다. 즉, 단어가 특정 문서에서 자주 등장한다면 해당 문서에서 해당 단어가 중요하다는 의미이다. 이는 검색 결과의 정확도를 높이는 데 도움이 된다.

③ 단어 가중치 조절
TF-IDF 점수는 IDF를 사용하여 문서 집합 전체에서 단어의 중요도를 조정한다. 흔한 단어일수록 IDF 값이 낮아지며, 이는 해당 단어가 널리 사용되는 단어일 경우 그 자체로는 특정 문서와의 관련성을 제대로 반영하지 못하기 때문이다. 따라서 TF-IDF 점수는 이러한 단어의 가중치를 조절하여 적절한 중요도를 반영할 수 있다.
④ 계산 효율성
TAAT 접근 방식을 사용하는 TF-IDF 점수 계산은 효율성이 높다. 각 문서에 대한 단어의 TF-IDF 점수를 계산하고 문서의 총 점수를 계산하여 정렬하는 단계가 비교적 간단하며, 대규모 문서 집합에서도 효과적으로 적용할 수 있다.

Ranked 검색은 TF-IDF를 효과적으로 활용하여 처리되며, 자유롭게 텍스트 쿼리를 입력하여 관련성이 높은 문서를 다음과 같이 검색할 수 있다.
1. 전처리 완료된 문서 집합에 대해 TF-IDF 점수를 계산한다. 각 문서의 단어의 TF-IDF 점수가 구해진다.
2. 사용자의 쿼리를 입력받은 후, 공백을 기준으로 쿼리를 토큰화하여 각 단어의 TF-IDF 점수를 계산한다.
3. 각 문서에 대해 단어의 TF-IDF 점수를 합산하여 문서의 총 점수를 계산한다.
4. 문서를 총 점수에 따라 내림차순으로 정렬하여 결과를 반환한다.

다음은 구현된 ranked retrieval model의 검색 결과를 살펴보기 위해, 앞서 boolean retrieval model의 예제로 사용한 쿼리 “Everyday Wish’ 에 대해 검색을 수행했으며 그 결과는 [표7]과 같다.

[표7] TF-IDF 기반 Ranked 검색 결과

Query	결과 건수	문서 ID	파일명	TF-IDF 점수
‘Everyday Wish’	455	13	3wishes.txt	0.0347
		264	lgoldbrd.txt	0.0240
		…		
		452	write	0.000
		453	wrt	0.000

예제 쿼리 ‘Everyday Wish’ 를 ranked 검색 함수 에 입력하면, 공백을 기준으로 쿼리를 토큰화하여 각 단어 ‘Everyday’ 와 ‘Wish’ 의 TF-IDF 점수를 계산한다. 그리고 문서별 두 단어의 TF-IDF 점수 총합을 계산 후 내림차순으로 정렬하여 검색 결과를 반환한다.

[표7]의 결과 건수는 총 455개로 수집된 문서의 총 개수와 동일하며, TF-IDF 점수를 기준으로 내림차순으로 정렬된 문서의 ID와 파일명을 확인할 수 있다.
자유 텍스트 쿼리 'Everyday Wish'와 가장 관련성이 높은 문서는 3wishes.txt로 이 문서의 TF-IDF 값은 소수점 넷째자리에서 반올림한 결과 0.0347점인 것으로 나타났다.

[표8] 3wishes.txt 문서 단어별 TF-IDF 점수 상위 5건

파일명	총 단어	단어명	TF-IDF 점수
3wishes.txt	231	woodcutt	0.1509
		sausag	0.0905
		elf	0.0722
		wife	0.0443
		wish	0.0347

[표8]은 가장 관련성이 높은 문서 3wishes.txt의 단어별 TF-IDF 점수를 나타내는 표이며, TF-IDF 점수를 기준으로 내림차순 정렬하여 상위 5건만 출력하였다. 이 문서는 총 231개의 단어를 포함하고 있으며, 주요 내용은 한 나문꾼이 요정을 만나게 되고 세 가지 소원을 주기로 약속한 후 마지막 소원을 통해 행복해지는 이야기이다.
한편 단어 문서 내에서 'wish'의 TF-IDF 값은 0.0347로 상위 5번째이며, 'everyday'는 문서 내에 포함되지 않은 것으로 나타났다. 따라서 ranked 검색 결과 3wishes.txt 문서의 TF-IDF 점수는 'wish' 단어의 점수만 포함되어있음을 확인할 수 있다.

예제 수행 결과 ranked retrieval model 검색 결과는 자유로운 텍스트 쿼리를 사용하여 모든 문서에 대해 쿼리와의 관련성을 TF-IDF 점수로 확인할 수 있었다. TF-IDF 점수를 활용한 TAAT 방식의 Ranked 검색은 일반적으로 효과적이지만, 다음과 같은 몇가지 단점이 있을 수 있다.
① 문서 길이의 영향
TF-IDF는 단어의 빈도와 역문서 빈도의 곱으로 계산되는데, 문서의 길이가 너무 긴 경우에는 단어의 빈도가 높아져서 높은 TF-IDF 점수를 가지게 된다. 따라서 긴 문서가 상위에 랭킹되는 경향이 있을 수 있으며, 짧은 문서의 중요한 정보가 상대적으로 무시될 수 있다.
② 단어의 중요도 평가
TF-IDF는 단어의 빈도와 역문서 빈도를 기반으로 단어의 중요도를 평가한다. 하지만 모든 단어가 동등한 중요도를 가지는 것은 아니다. 예를 들어, 불용어와 같은

일반적이고 빈번하게 등장하는 단어는 높은 빈도를 가지지만 문서 내에서 중요하지 않을 수 있다. 따라서 TF-IDF 계산 전 문서의 전처리 과정을 수행해야 한다.

③ 검색어의 다의성 처리

TAAT 방식은 쿼리의 단어를 개별적으로 처리하고, 단어 간의 상호작용을 고려하지 않는다. 이는 검색어에 특정 단어가 여러 가지 다른 의미를 가질 경우에 문제가 될 수 있다. TF-IDF 또한 단어의 독립적인 빈도를 고려하기 때문에 이러한 다의성을 처리하기 어렵다.

④ 메모리 요구량

TAAT 방식은 모든 문서에 대한 검색 결과를 동시에 계산하기 때문에 메모리 요구량이 상당할 수 있다. 특히, 큰 규모의 문서 집합에서는 많은 메모리를 필요로 할 수 있다.

다음으로 동일한 예제 쿼리를 기반으로 수행한 Boolean retrieval model과 Ranked retrieval model의 결과를 기반으로 두 검색 모델을 비교하고자 한다.

Ⅴ. 결론

검색 모델은 정보 또는 문서를 효과적으로 검색하고 정렬하는 데 사용되는 도구이다. 본 레포트는에서는 검색 모델 중 Boolean retrieval model과 Ranked retrieval model을 구현하고 동일한 예제 쿼리 'Everyday Wish'를 사용하여 검색을 수행하였다.

Boolean 검색 모델 예제 수행 결과, 예제 쿼리와 연산자의 조합에 따른 정확히 일치하는 문서들만 반환되었다. 따라서, 연산자 조합에 따라 'Everday' 또는 'Wish' 단어가 포함되지 않은 문서들은 결과에서 제외되었다.
Ranked 검색 모델 예제 수행 결과, 총 455개의 문서들이 TF-IDF 점수를 기반으로 평가되었으며 내림차순으로 정렬되어 반환되었다. 최상위 문서로 '3wishes' 문서가 나타났으며 예제 쿼리와 가장 관련성이 높은 것으로 판단되었다. 또한, 이외 문서들도 점수에 따라 순위화 되어 결과로 나타났다.
'Everyday Wish' 쿼리의 예제에서 Ranked 검색 모델은 더 다양한 관련 문서들을 반환하며, 상위에 가장 관련성이 높은 문서를 정확히 식별할 수 있었다. 반면 Boolean 검색 모델은 정확한 일치만을 고려하기 때문에 결과는 제한적이며, 단어의 순서나 빈도에 대한 고려 없이 일치 여부만을 고려하기 때문에 결과의 다양성이 부족하다고 할 수 있다.

두 검색 모델은 쿼리 표현, 유연성, 결과 정렬 및 정확도 등에서 차이를 보인다. 두 검색 모델의 기준별 비교는 [표9]와 같다.

[표9] Boolean 과 Ranked 검색 모델 비교표

비교 기준	Boolean 검색	Ranked 검색
쿼리 표현	논리 연산자	자유 텍스트
유연성	제한적	유연함
결과 정렬	동일 순위	관련성에 따라 순위화
결과 정확도	단순 일치 여부 고려	유사성 고려

쿼리 표현의 경우, Boolean 검색 모델은 주로 논리 연산자를 사용하여 쿼리를 표현하며, AND, OR, NOT 등의 연산자를 이용하여 단어의 조합을 나타낸다. 반면 Ranked 검색 모델은 일반적으로 텍스트 기반으로 쿼리를 사용하며, 자유로운 문장이나 구문으로 작성될 수 있다.

유연성의 경우, Boolean 검색 모델은 단순한 일치 여부만으로 고려하기 때문에 유연성이 상대적으로 제한된다. 반면 Ranked 검색 모델은 단어의 빈도, 위치, 용어의 중요도 등을 고려할 수 있으며, 쿼리와 문서 간의 유사성을 고려하여 결과를 제공한다.
결과 정렬의 경우, Boolean 검색 모델은 단순히 쿼리와 문서의 일치 여부에 따라 결과를 반환하며, 일치하는 문서는 모두 동등한 순위로 취급된다. 반면 Ranked 검색 모델은 각 문서에 점수를 할당하고 높은 점수를 가진 문서가 상위에 랭크되는 것처럼 검색 결과를 관련성에 따라 순위화하여 반환한다.
결과 정확도의 경우, Boolean 검색 모델은 일치하는 문서가 있는 경우 정확한 결과를 제공한다. 반면 Ranked 검색 모델은 쿼리와 문서 간의 유사성을 고려하여 결과를 평가하기 때문에 일치도에 따른 정확도가 높으며, 상위 순위의 문서는 관련성이 높은 경우가 많다.

본 레포트의 간단한 예제를 통해 결과적으로 Boolean retrieval model과 Ranked retrieval model의 두 모델은 각각 다른 장단점을 가지며 다른 결과를 반환하였다.

Ranked 검색 모델은 정보의 유연성과 다양성을 제공하며, 검색 모델 사용자가 원하는 정보를 효과적으로 찾을 수 있게 도와준다. 반면 Boolean 검색 모델은 단순한 일치 여부에 초점을 두고 결과를 반환하기 때문에 제한적이지만, 특정한 단어의 일치 여부에 집중하는 검색에 적합할 수 있다.
따라서 정보의 다양성와 문서와 쿼리 간 관련성을 중요시하는 경우 Ranked 검색 모델이 적합하여, 특정한 단어나 조건의 일치 여부에 초점을 두는 경우에는 Boolean 검색 모델이 적합할 수 있다.

이러한 검색 모델 비교 분석을 통해, 검색 모델 사용자의 요구 사항과 목적에 따라 최적화된 검색 모델의 선택이 중요하다는 것을 알 수 있었다.
또한, 정보화 시대에서 검색 모델의 개발자나 운영자는 검색 모델의 장단점을 구체적으로 파악하고 사용자들의 요구 사항을 면밀하게 분석하여 더 나은 검색 경험을 얻을 수 있게 도와주는 것이 필요하다는 것을 느꼈으며, 정확도 높게 효과적으로 검색을 수행하는 방법에 대한 고민을 해보는 시간이었다.