

# An Introduction to Information Retrieval

/ Christopher D. Manning, Prabhakar Raghavan, and Hinrich schutze. - cambridge, England. Cambridge Univ. Press., (c) 2009 - Online edition.

## I. Boolean retrieval

### 1. Boolean retrieval

Information retrieval이란 용어의 의미는 매우 광범위적일 수 있다. 카드번호를 입력하기 위하여 여러분의 지갑에서 신용카드를 꺼내는 것 또한 정보검색의 한 형태이다. 그렇지만 정보검색의 학술적 정의는 다음과 같다:

“IR is finding material (usually documents) of an unstructured nature(usually text) that satisfies an information need from within large collections(usually stored on computers.”

이런 정의에 따라, 정보검색은 단지 소수의 사람(reference librarians, paralegals, similar professional searchers)만의 activity로 여겨져 왔다. 그러나 이제 세상은 변하였고, 수 억 명의 사람이 매일매일 웹 탐색엔진을 이용하거나 전자우편을 탐색함으로써 정보검색에 참여하고 있다. 현재는 “search”란 단어가 “(information) retrieval”을 대체하여 사용되는 경향이 있다.

정보검색은 위에서 정의한 것 이외에도 또 다른 종류의 데이터와 정보문제를 다루기도 한다. “unstructured data(비정형화 데이터)”란 용어는 어의적으로 명확하게 컴퓨터에서 다루기 어려운 구조의 데이터를 말한다. 이것은 관계형 데이터베이스의 전통적 샘플인 structured data(정형화 데이터)의 반대이며, product inventories와 personnel records를 유지관리하기 위하여 일반적으로 사용하는 데이터이다. 현실에서 대부분의 데이터는 사실상 “unstructured” 하지 않다. 대부분의 텍스트는 일반적으로나 의도적으로나 문서에 headings, paragraphs, footnotes와 같은 구조를 가지고 있다. 정보검색은 또한 title에 *Java*를 포함하거나 body에 *threading* 을 포함하고 있는 “semi-structured” 텍스트를 탐색할 수도 있다.

정보검색에서는 다큐먼트 집단을 browsing하거나 filtering하거나 또는 더 나아가서 한 세트의 검색 다큐를 처리하려는 이용자를 지원한다. 한 세트의 다큐가 주어졌을 때, clustering이란 콘텐츠에 근거하여 다큐를 훌륭하게 grouping하는 것을 말한다. 이것은 서가에서 주제별로 책을 배열하는 것과 비슷하다. 그리고 서로 다른 집단의 텍스트가 주어졌을 때, 분류는 각각의 다큐가 어떠한 classes에 속하는가를 결정하는 일이다.

이제 정보검색문제의 간단한 예로 시작하여 a term-document matrix의 아이디어와 inverted index data structure를 소개한 다음에, 불리안 검색 모델과 불리안 쿼리를 처리하는 방법에 대해 알아보기로 한다.

## 1.1 정보검색문제의 간단한 예제

많은 사람이 갖고 있는 두툼한 책이 바로 Shakespeare's Collected Works(셰익스피어 전집)이다. 여러분이 셰익스피어의 어떤 희곡에 “Brutus AND Caesar AND NOT Calpurnia”가 포함되어 있는지를 알고자 한다고 가정해 보자. 이것을 아는 한 가지 방법은 처음부터 시작하여 모든 텍스트를 읽고, 각 희곡에 Brutus와 Caesar가 포함되어 있지만 Calpurnia는 포함 안된 것을 살펴보는 것이다. 다큐검색의 가장 간단한 형태는 컴퓨터를 가지고 다큐 전체를 이 같은 선형적 방식으로 스캔하는 것이다. 이 과정을 일반적으로 Grepping through text라고 부른다.

그렇지만, 여러분은 다음과 같은 여러 가지 목적으로 더 많은 기법을 필요로 한다:

1) To process large document collection quickly.

온라인 데이터의 양이 컴퓨터의 속도만큼 빠르게 증가하고 있으며, 우리는 지금 수십억 또는 수조의 단어들이 모두 질서정연하게 들어있는 collections를 탐색하고 싶다.

2) To allow more flexible matching operations.

예를 들어 ‘grep’ 방식에서 Romans NEAR countrymen 쿼리를 사용하는 것은 비실용적이다. 왜냐하면 NEAR는 “within 5 words” 또는 “within the same sentence”를 정의하기 때문이다.

3) To allow ranked retrieval:

다수의 경우에, 여러분은 어떤 단어가 포함된 많은 문서 중에서 정보요구에 대한 최상의 해답을 원한다.

쿼리에 맞춰서 텍스트를 선형적으로 스캐닝하는 것을 피하는 방법은 미리 그 다큐들을 색인(index)하는 것이다. 이 글에서는 Shakespeare's Collected Works를 대상으로 Boolean retrieval model의 기초를 소개하기로 한다. 우리는 셰익스피어가 사용한 모든 단어들 속에서 그의 특정한 희곡에서 어떤 단어가 사용되었는지를 체크한다고 가정해 보자(셰익스피어는 32,000개의 서로 다른 단어를 사용했다). 그 결과를 나타낸 것이 도 1.1의 binary term-document incidence (발생률) matrix 이다. 여기서 용어(terms)는 indexed units(색인단위)이다; 이것들은 늘 단어(words)이며 지금까지 여러분은 그것들을 단어로만 생각해 왔다. 그렇지만, 정보검색에서는 그것들 중 어떤 것은 항상 단어로만 취급되지 않기 때문에, 이

러한 단어를 용어라 부른다.

이제 아래의 매트릭스의 로우와 칼럼을 살펴보면, 여기에 표현된 다큐의 각각의 용어 vector를 알 수 있거나, 용어를 포함하고 있는 각각의 다큐 vector를 알 수 있다.

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth	...
Antony	1	1	0	0	0	1	
Brutus	1	1	0	1	0	0	
Caesar	1	1	0	1	1	1	
Calpurnia	0	1	0	0	0	0	
Cleopatra	1	0	0	0	0	0	
mercy	1	0	1	1	1	1	
worser	1	0	1	1	1	0	
...							

► **Figure 1.1** A term-document incidence matrix. Matrix element  $(t, d)$  is 1 if the play in column  $d$  contains the word in row  $t$ , and is 0 otherwise.

위에서 제시한 쿼리 **Brutus AND Caesar AND NOT Calpurnia**의 결과를 얻기 위하여, 우리가 Brutus, Caesar, Calpurnia의 vector를 근거하여 이진적으로 AND를 실행하면 그 표현식은 다음과 같다:

$$110100 \text{ AND } 110111 \text{ AND } 101111 = 100100$$

따라서 결과는 희곡 “Antony and Cleopatra”와 “Hamlet” 두 편뿐 이다.

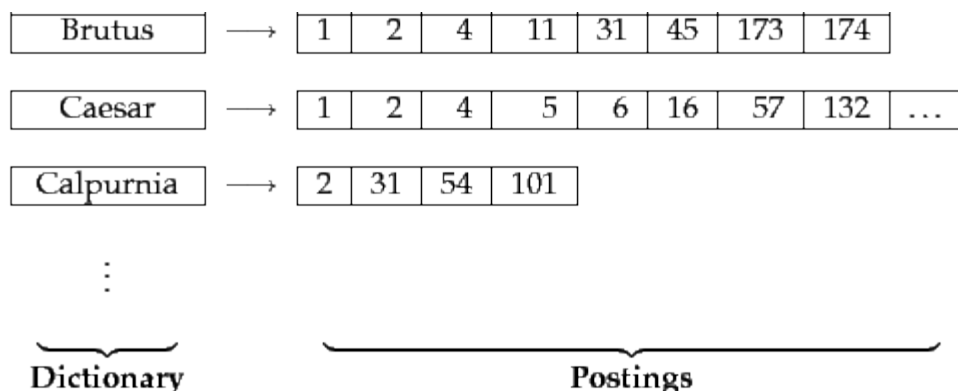
Boolean retrieval model은 정보검색용 모델이며, 우리는 용어들의 Boolean expression으로 어떠한 쿼리도 작성할 수 있으며, 용어들은 불리안 연산자 AND, OR, NOT 으로 결합시킨다. 이 모델은 다큐를 단지 단어들의 집단으로만 여긴다.

information need란 이용자가 알기 원하는 주제(topic)이며, query와는 다르다. 쿼리란 이용자가 정보요구를 처리하기 위하여 컴퓨터에 입력하는 어떤 무엇이다. 만일 이용자가 가치 있는 정보를 포함하고 있는 것으로 인식하는 다큐라면, 그것은 적합한 것이다. 정보검색 시스템의 효과 평가(i.e. the quality of its search results)를 위하여, 이용자는 쿼리와 시스템의 검색결과에 관한 두 가지 중요한 통계에 대하여 잘 알아야 한다:

- 1) Precision: What fraction of the returned results are relevant to the information need?
- 2) Recall: What fraction of the relevant documents in the collection were returned by the system?

우리는 대책없이 term-document matrix를 구축하진 않을 것이다. 그 이유는 500K x 1M matrix는 half-a-trillion의 0과 1을 갖기 때문에, 너무 커서 우리 컴퓨터의 메모리에 적합하지 않다. 그러나 중요한 것은 이러한 매트릭스는 매우 sparse하다는 것이다 - 즉, non-zero entries가 거의 없다는 것이다. 예를 들어 100만개의 다큐마다 1,000개씩의 단어를 갖고 있다면, 그 매트릭스는 단지 10억 개의 1이나 0 들을 가지고 있지만, 그 셀들의 최소한 99.8%는 0으로 되어있다. 한 가지 좋은 것은 1 만의 경우를 표현할 수 있다는 것이다.

이러한 아이디어가 바로 정보검색의 첫 번째 중요한 개념인 도치 색인(inverted index)의 핵심이 되었다. 도치색인(때때로 inverted file)은 정보검색의 표준어 이다. 도치색인의 기본적인 아이디어가 아래의 도 1.3에 나타나 있다. 우리는 용어 사전(dictionary of terms)을 가지고 있다(때때로 또한 vocabulary 또는 lexicon이라 부른다). 이 글에서는 data structure용으로는 dictionary를, 그리고 용어의 세트용으로 vocabulary를 사용한다. 각 용어와 관련해서, 우리는 그 용어가 나타나 있는 다큐가 어떤 것인지를 기록하고 있는 리스트를 갖고 있다. 그 리스트 - 다큐에 나타난 용어를 기록하고 있고, 나중엔 종종 그 다큐의 위치를 나타내기도 하는 - 에 있는 각 다큐들을 전통적으로 posting이라 부른다. 따라서 이것의 리스트를 postings list(또는 inverted list)라 부르며, 그 postings list에서 포함되어 있는 모든 것들을 postings라 부른다. 아래의 도 1.3에 있는 dictionary는 자모순으로, 그 postings list는 document ID로 정렬(sorted)되어 있다.



► **Figure 1.2** The two parts of an inverted index. The dictionary is commonly kept in memory, with pointers to each postings list, which is stored on disk.

## 1.2 A first take at building an inverted index

검색 시에 색인으로부터 신속한 도움을 얻기 위하여, 우리는 미리 색인을 구축해 놓아야 한다. 색인의 중요한 단계는 다음과 같다:

- 1) Collect the documents to be indexed:

Friends, Romans, countrymen. So let it be with Caesar ...

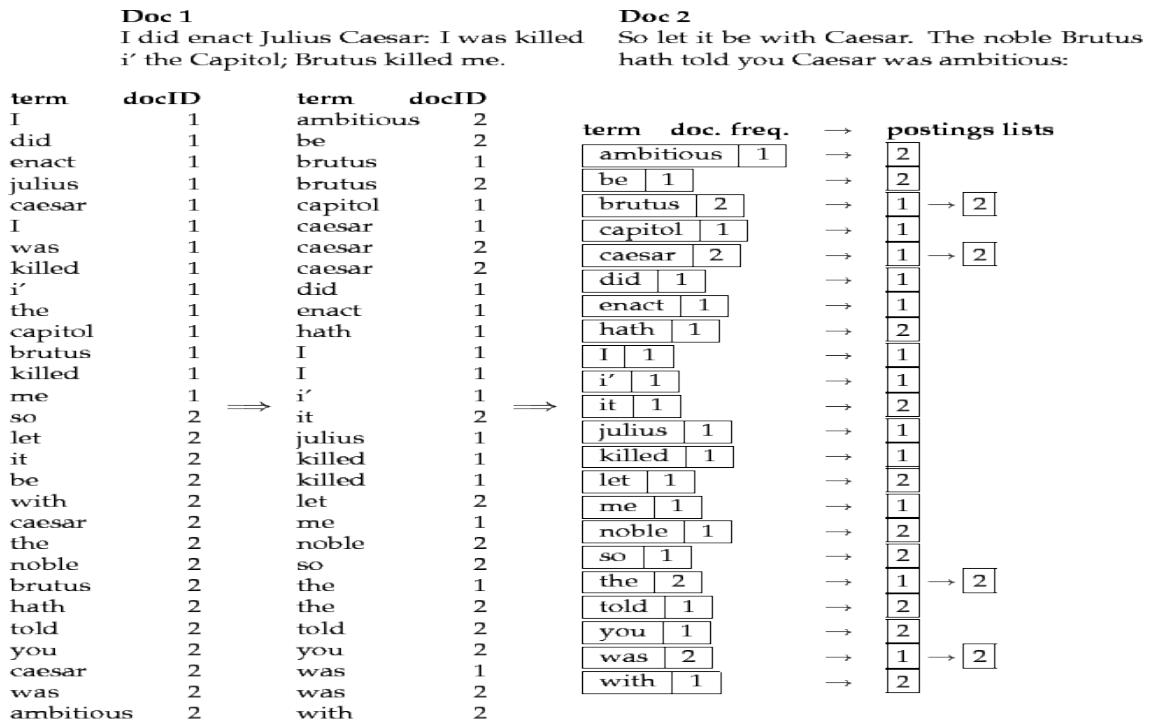
2) Tokenize the text, turning each document into a list of tokens:

Friends Romans countrymen So ...

3) Do linguistic preprocessing, producing a list of normalized tokens, which are the indexing terms:

friend roman countryman so ...

4) Index the documents that each term occurs in by creating an inverted index, consisting of a dictionary and postings.



► **Figure 1.3** Building an index by sorting and grouping. The sequence of terms in each document, tagged by their documentID (left) is sorted alphabetically (middle). Instances of the same term are then grouped by word and then by documentID. The terms and documentIDs are then separated out (right). The dictionary stores the terms, and has a pointer to the postings list for each term. It commonly also stores other summary information such as, here, the document frequency of each term. We use this information for improving query time efficiency and, later, for weighting in ranked retrieval models. Each postings list stores the list of documents in which a term occurs, and may store other information such as the term frequency (the frequency of each term in each document) or the position(s) of the term in each document.

tokens를 완곡하게 표현하면 words라 생각하고, 색인정렬작업을 통해 기본적인 도치색인을 작성한다.

document collection에서, 각 다큐는 document identifier(docID)라하는 유일한 일련번호를 가지고 있다고 가정하자. 색인작업을 하는 동안, 새로운 다큐가 처음 나타날 때마다, 그것에 연속적인 번호를 할당한다. indexing용 입력요소(input)는 각 다큐의 normalized tokens의 list(term과 docID의 pairs로 되어 있는 리스트) 이다. 중요한 indexing 단계는 용어가 자모순이 되도록 리스트를 정렬하는 것이다. 이 때, 동일한 다큐에서 동일한 용어가 복수로 발생하면, 통합시킨다(merge). 그런 다음에 동일한 용어의 빈도를 합산하여 그 결과를 dictionary 와 postings에 분리시켜 표현한다. 어떤 용어는 일반적으로 많은 다큐에 나타나므로, 이러한 데이터의 조직은 색인의 요구조건에 맞지 않다. dictionary 또한 각 용어를 포함하고 있는 다큐의 빈도수(document frequency)와 같은 어떤 통계들을 포함하고 있다. 이 정보는 기본적인 불리언 탐색엔진에 중요하지는 않지만, 탐색 시에 탐색엔진의 효율성을 증대시키는데 도움을 주며, 최근에는 많은 서열화 검색 모델에서 사용되고 있다. postings는 docID 다음에 두 번째로 정렬시킴으로써, 효율적인 쿼리과정을 위한 근거로 사용된다. 이러한 도치색인 구조는 특히 text search를 지원하는 데 있어서 가장 효율적인 구조이며, 어떠한 경쟁상대도 없다.

최종적으로 구축된 색인에서, 우리는 dictionary와 postings list 둘 다의 저장공간을 제공하여야 한다. 후자가 훨씬 더 크므로, dictionary는 일반적으로 memory 속에, postings list는 disk에 보관된다.

#### Exercises.

1) 다음의 컬렉션에 적합한 도치색인을 작성하라.

- Doc 1 new home sales top forecasts
- Doc 2 home sales rise in july
- Doc 3 increase in home sales in july
- Doc 4 july new home sales rise

2) 다음 문서를 생각해 보라

- Doc 1 breakthrough drug for schizophrenia
- Doc 2 new schizophrenia drug
- Doc 3 new approach for treatment of schizophrenia
- Doc 4 new hopes for schizophrenia patients

3) the term-document incidence matrix를 작성하라

4) the inverted index를 작성하라.

■ 예제 1.2의 컬렉션을 근거로 다음과 같은 쿼리의 결과가 무엇인지를 밝혀라.

5) schizophrenia AND drug

b. for AND not(drug OR approach)

### 1.3 Processing Boolean queries

도치색인과 기본적인 불리안 검색 모델을 사용하여 쿼리를 어떻게 처리하는가? 간단한 conjunctive query의 처리과정을 생각해 보자:

#### *Brutus AND Calpurnia*

위의 도 1.3에서 부분적으로 보여준 도치색인을 가지고서. 우리는:

1. Locate *Brutus* in the Dictionary
2. Retrieve its postings
3. Locate *Calpurnia* in the Dictionary
4. Retrieve its postings
5. intersect the two postings, as shown in Figure 1.5.

**교집합(intersection)**은 중요한 것이다: 우리는 두 가지 용어 모두를 포함하고 있는 다큐를 신속하게 발견하기 위하여 효율적으로 posting list를 교집합 시켜야 한다(이러한 오퍼레이션은 때때로 posting list를 *merging* 한다고 말하기도 한다).

위의 도 1.3에서 Brutus와 Calpurniz의 posting lists를 교집합 시키면 다음 도 1.5와 같다:

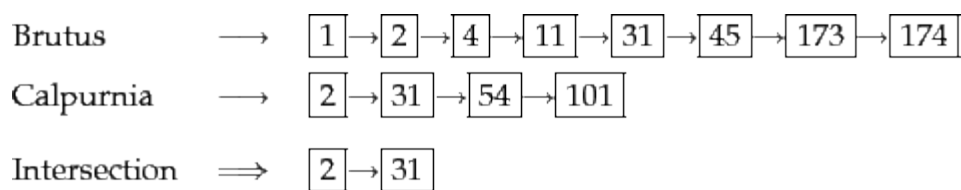


Figure: 1.5

우리는 교집합 연산을 다음과 같은 보다 복잡한 쿼리를 처리하도록 확장할 수 있다:

#### **(Brutus OR Caesar) AND NOT Calpurnia**

**Query optimization**이란 시스템에 의해 최소한의 총 문헌수(the least amount of total work)만을 얻도록 쿼리 작업을 최적화하는 방법이다. Boolean queries용으로 이것을 위한 한 가지 중요한 요소는 posting lists의 접근 순서 이다. 쿼리처리를 하는데 있어서 최상

의 순서는 무엇인가? 예를 들어, 여러 개의 용어들과 AND 연산자를 갖는 다음과 같은 쿼리에 대해 생각해 보자:

### **Brutus AND Caesar AND Calpurnia**

위의 용어들 각각에 대하여, 우리는 그것의 postings를 얻은 다음에 그것들을 AND로 묶어야 한다. 이상적인 방법은 용어들을 처리하여 다큐의 빈도수를 결정하는 것이다: 만일 우리가 두 개의 가장 작은 posting lists를 교집합하면서 시작한다면, 모든 중간 결과는 가장 작은 posting lists보다 결코 더 크지 않을 것이다. 그러므로 the least amount of total work를 얻게 될 것이다. 따라서 우리는 도 1.3에 있는 posting lists와 관련해서 위의 쿼리를 다음과 같이 수정하여야 한다:

### **(Calpurnia and Brutus) AND Caesar**

이것이 dictionary에 용어들의 빈도를 포함시켜야 하는 첫 번째 정당성이다. 이것은 우리에게 어떤 posting lists에 접근하기 전에 in-memory data를 근거로 위와 같은 순서를 결정하도록 한다.

## **II. The extended Boolean model versus ranked retrieval**

불리언 검색모델 그리고 벡터 스페이스 모델과 같은 서열화 검색모델은 서로 대조적이다. 벡터 스페이스 모델에서 사용자들은 대부분 free text queries를 사용한다. free text queries란 쿼리 표현식을 작성하기 위하여 연산자와 함께 정확한 언어를 사용하기보다는 한 개나 그 이상의 단어만을 자유롭게 입력시킨 쿼리를 말한다. 이러한 쿼리를 입력하면, 시스템에서는 그 쿼리를 가장 만족시키는 다큐를 결정한다. 서열화 검색의 장점에 대한 수십 년 동안의 학술적 연구에도 불구하고, 불리언 검색 모델을 적용하고 있는 시스템은 WWW이 도입 되던 1990년대 초까지 30년 동안 대형 상업정보회사에서 제공된 주요 탐색 옵션만을 사용하였다. 그렇지만 이들 시스템은 우리가 이제까지 표현해 왔던 기본적인 불리언 연산자(AND, OR, NOT)만을 가지고 있지는 않았다. 무순서적인 결과를 초래하는 불리언 표현식은 사람들에게 너무나 많은 결과 데이터를 제공하였기 때문에, 최근 시스템들은 term proximity operators와 같은 추가적인 연산자를 사용하는 확장 불리언 검색모델을 설치하였다.

### **1. proximity operator**



쿼리에 있는 두 개의 용어들이 다큐에서 서로 얼마나 근접해 있는가를 나타내는 방법이다. 이러한 근접성(closeness)은 중간에 개입할 수 있는 단어의 수를 제한함으로써, 또는 문장이나 문단과 같은 구성 단위(structural unit)를 참조함으로써, 측정될 수 있다.

- NEAR
- FBY (Followed BY)

proximity operators: 사용 방법:

- 1) 먼저 탐색어를 입력하라
- 2) proximity operator를 입력하라
- 3) 탐색어 사이에 들어갈 수 있는 단어의 수와 마침표를 입력하라:

예) sly near.6 fox  
young fby.5 in love

## 2. Truncation 및 wildcard operators

- \* (asterisk) truncation operator: zero or more terminal characters in a search term.
- ? (question) truncation operator: replace any character in a given search term, regardless of position (as in gr?y, hono?r, ?nquiry or gothic?).

## 3. Exact operators

The EXACT operator can be used to retrieve records that match your search term precisely. Simply type the word or phrase enclosed within double quotation marks.

## 4. parentheses with Boolean operators

Use parentheses to specify the order looks for any search terms in parentheses first.

예) (west indies NEAR pirates) OR (west indies NEAR bucaniers)

## 5. Relevance feedback

RF(*relevance feedback*)이란 검색과정에 이용자를 참여시켜서, 최종 결과를 개선시키는 것이다. 특히, 이용자는 제일 먼저 검색된 각각의 다큐에 대하여 적합성을 판단을 내린 다음, 그 결과를 시스템에 되돌려 보낸다(feedback). 이것의 기본적인 절차는 다음과 같다:

- 1) 이용자는 간단하고 짧은 쿼리를 제공한다.
- 2) 시스템에서는 최초의 검색결과를 보여준다.
- 3) 이용자는 검색결과 다큐에 대해 적합여부를 표기한다.
- 4) 시스템에서는 이용자의 피드백을 근거로 보다 나은 정보요구의 표현식을 계산한다.
- 5) 시스템은 수정된 결과를 보여준다.

RF에서는 이러한 일을 되풀이할 수 있다. 이것은 우리가 다큐 집단에 대해 잘 알지 못할 때 비록 양질의 쿼리를 작성하는 것이 어렵더라도, 특정한 문서를 주관적으로 판단하는 것은 쉽다는 아이디어를 활용한 것이다. 따라서 RF는 이러한 일의 반복을 통하여 기존의 쿼리를 정교화(refinement)한다. RF는 또한 이용자가 거듭해서 전개(evolution)시키는 정보요구를 추적하는데 효과적일 수 있다: 어떤 다큐를 평가함으로써 이용자는 자신들이 찾는 정보에 대한 이해를 정교화시킬 수 있다.