

AI기법과 활용

Week-04. Search Engine Advanced

2022-Summer 서종원

문서 검색

Boolean Retrieval (부울 검색)

- 쿼리 연산에 대해서 두가지 중 하나의 결과를 보여줌
 - True or False
 - Exact-match
- 일반적으로 Query는 부울 연산을 이용해 제공됨
 - AND OR NOT
- 기본 가정은, "검색된 모든 결과는 동일하게 관련된 내용이다"

문서 검색

Boolean Retrieval

- 아직도 많은 검색 시스템은 부울 연산을 활용
 - 이메일, 인스타그램
- 일부 도메인에 대해서는 매우 효과적인
 - 특허 검색
 - 법률 검색
 - 개발자 디버깅 검색

문서 검색

Boolean View

- 각 행은 특정한 단어(term)을 표현
 - 각 단어가 들어간 문서는?
- 쿼리 실행
 - 검색어로 들어온 Term을 고르고
 - Boolean 연산을 적용
 - My
 - Doc1, Doc3, Doc5

Term	Doc1	Doc2	Doc3	Doc4	Doc5
Hello	0	0	0	1	0
My	1	0	1	0	1
Love	0	0	1	0	0
It	1	1	1	1	1
Is	1	1	1	1	1
Very	0	1	0	0	1
Cold	0	0	0	1	0
On	1	0	1	1	1
This	0	1	1	1	1
Island	0	1	0	0	0

문서 검색

Boolean 검색 예제

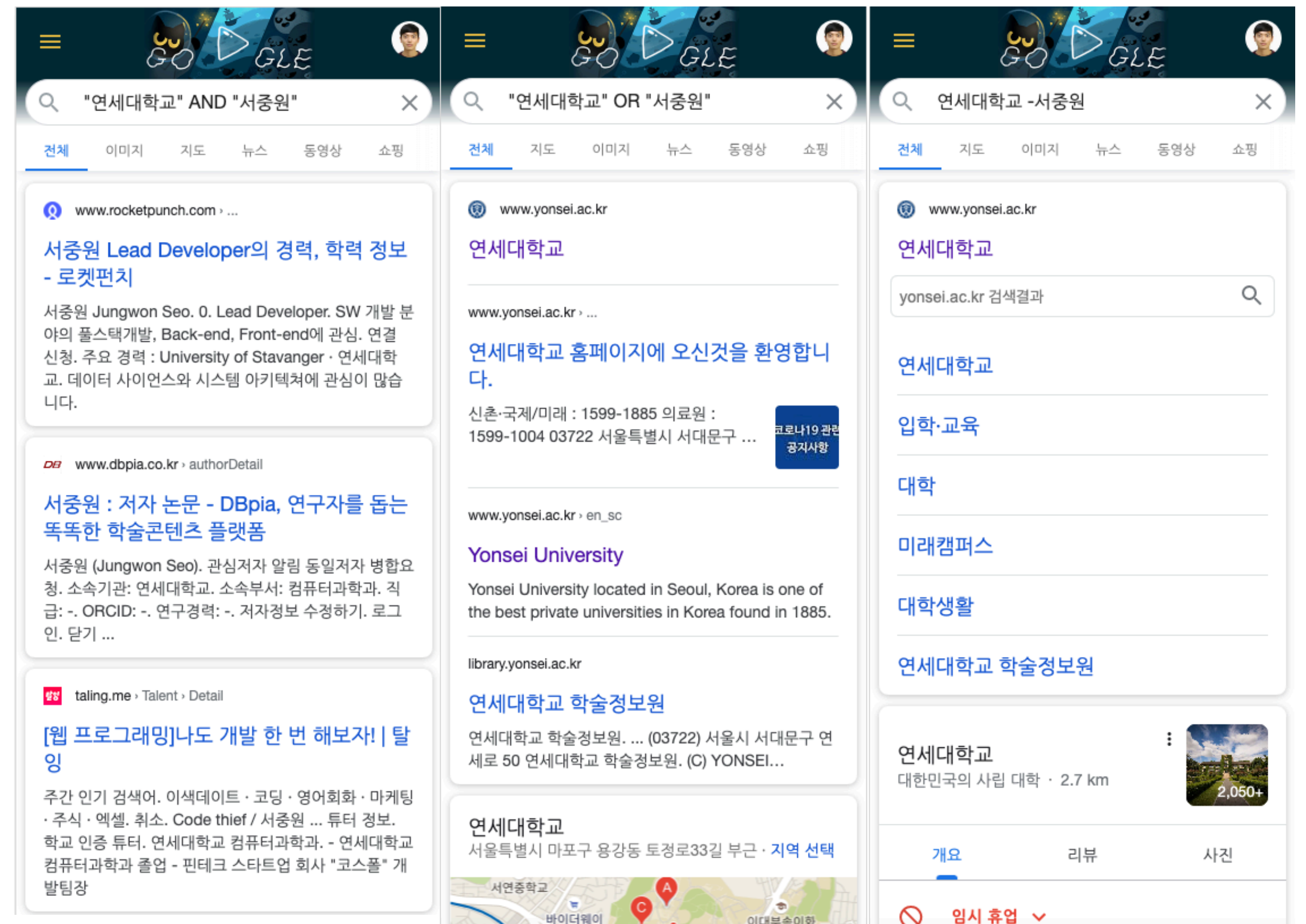
Term	Doc1	Doc2	Doc3	Doc4	Doc5
This	0	1	1	1	1
On	1	0	1	1	1

- This **AND** On : Doc3, Doc4, Doc5
- This **OR** ON : Doc1, Doc2, Doc3, Doc4, Doc5
- This **AND NOT** On : Doc2
- On **AND NOT** This : Doc1

문서 검색

구글에서 Boolean 검색 예제

- AND 연산자: "연세대학교" **AND** "서중원"
- OR 연산자: "연세대학교" **OR** "서중원"
- NOT 연산자: 연세대학교 -학술정보원



문서 검색

Boolean 검색 장/단점

- 장점

- 검색 결과에 대한 설명이 쉬움 (포함/미포함)
- 다양한 요소들이 검색에 함께 포함될 수 있음 (이미지가 포함되어있냐 아니냐 등등)
- 효율적인 연산 (시작과 동시에 많은 문서들이 제외 될 것이기 때문에)
- 관련된 문서를 절대 놓치지 않음

- 단점

- 검색 결과의 퀄리티는 사용자의 쿼리 작성에 의해 달려있음
- 문서간 랭킹이 X
- 단어가 포함되어 있지는 않지만, 관련된 문서를 검색 할 수 없음

문서 검색

Rank Retrieval

- $\text{score}(d, q)$
 - 주어진 쿼리 q 에 대해서 각각의 문서의 점수를 계산
 - Query = "Hello world"
- How?
 - $\omega_{t,d}$: 문서(d)와 Term(t)과의 **가중치** 계산
 - $\omega_{t,q}$: 쿼리(q)와 Term(t)과의 **가중치** 계산
 - 그리고 그 둘의 내적을 통한 **유사도** 계산
 - 각각의 문서에 대한 점수 반환

$$\text{score}(d, q) = \sum_{t \in q} \omega_{t,d} \cdot \omega_{t,q}$$

Scoring

Example 1 - Term Frequency Weighting

- $f_{t,d}$: 문서 d에서 term t가 등장한 횟수
- $f_{t,q}$: 쿼리 q에서 term t가 등장한 횟수
- Example
 - 쿼리 : Hello, Hello world
 - 텀 : Hello
 - 문서 : "Hello, Hello, Hello world, programming is very fun"
 - $f_{t,d} : 3$
 - $f_{t,q} : 2$

$$score(d, q) = \sum_{t \in q} \omega_{t,d} \cdot \omega_{t,q}$$

$$\omega_{t,d} = \begin{cases} 1, & f_{t,d} > 0 \\ 0, & \text{otherwise} \end{cases}$$

$$\omega_{t,q} = f_{t,q}$$

Scoring

Example 2 - Log frequency Weighting

- 문서에서 단어의 등장 회수를 "적절"하게 반영하기 위해서는?

$f_{t,d}$	$W_{t,d}$
0	0
1	1
2	1.3
10	2
1000	4

$$\omega_{t,d} = \begin{cases} 1 + \log f_{t,d}, & f_{t,d} > 0 \\ 0, & \text{otherwise} \end{cases}$$

$$\begin{aligned} \text{score}(d, q) &= \sum_{t \in q} \omega_{t,d} \cdot \omega_{t,q} & \omega_{t,q} &= f_{t,q} \\ &\quad \downarrow \\ \text{score}(d, q) &= \sum_{t \in q} (1 + \log f_{t,d}) \cdot \omega_{t,q} \end{aligned}$$

벡터 공간 모델

단어와 문서를 표현하기 위한 방법

- 1960-70년대 정보검색 연구분야의 기반이 되는 컨셉
- 아직도 많이 활용됨
- 다음과 같은 Framework를 구현하기에 간단하고 직관적임
 - Term weighting
 - Ranking
 - Relevance feedback

벡터 공간 모델

단어와 문서를 표현하기 위한 방법

- 문서와 쿼리는 term의 weight들의 벡터로 표현됨

$$D_i = (d_{i1}, d_{i1}, \dots, d_{it})$$

- 쉽게 표현하면, 문서와 쿼리는 단어들의 중요도에 대한 벡터임

$$Q = (q_1, q_2, \dots, q_t)$$

- Term과 Document의 매트릭스로 표현

	Term ₁	Term ₂	...	Term _t
Doc ₁	d ₁₁	d ₁₁	...	d _{1t}
Doc ₂	d ₂₁	d ₂₂	...	d _{2t}
.	.			
Doc _n	d _{n1}	d _{n2}	...	d _{nt}

우리가 보관하고 있는 모든 문서를 문서-Term 매트릭스로 표현

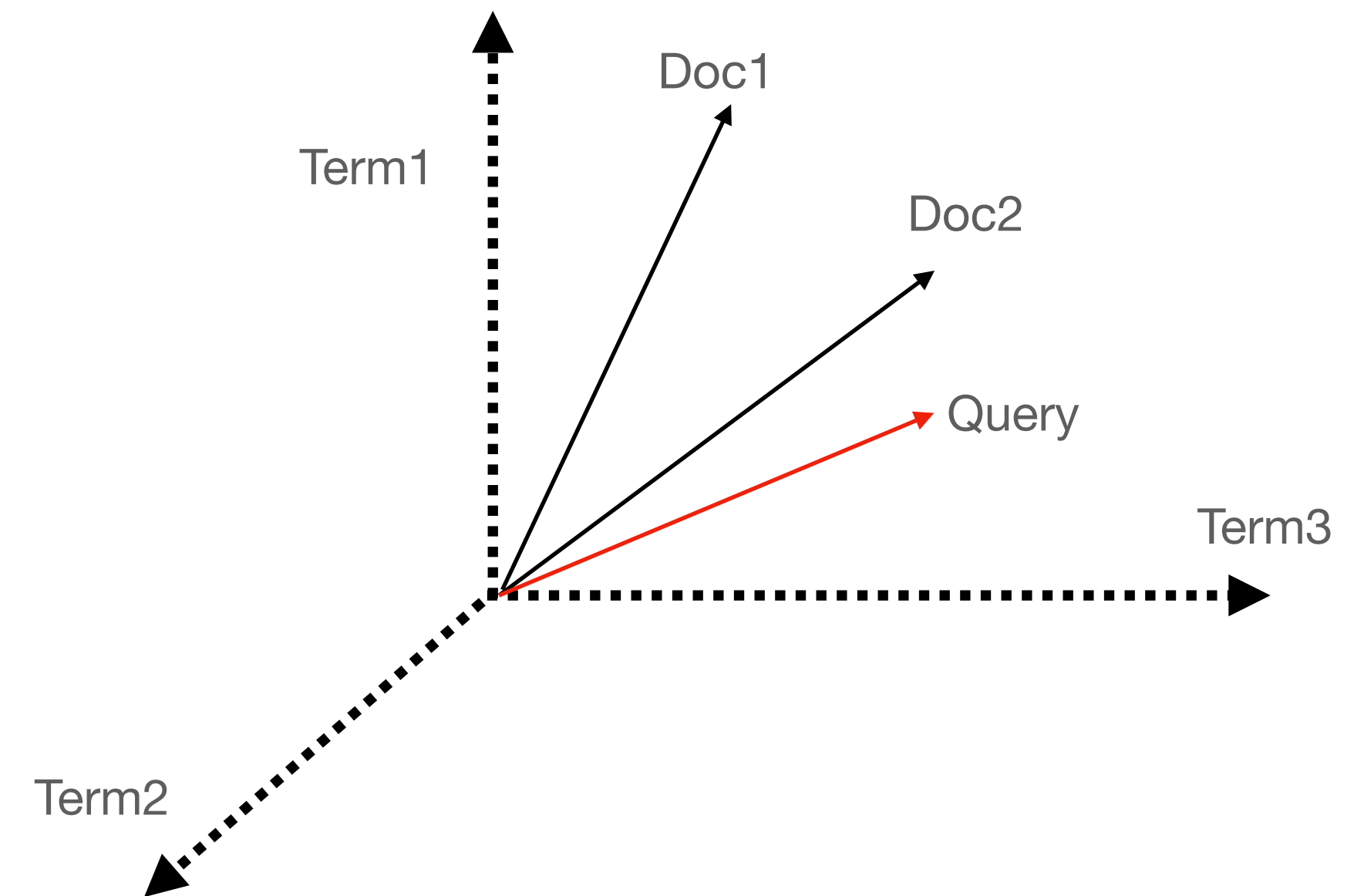
문서 점수화

이 쿼리와 가장 비슷한 문서는 무엇일까?

- 이전 슬라이드에서, 쿼리와 다큐먼트 둘 다 벡터로 표현을 했으므로, 문제를 재정의 가능
 - 두 벡터 간의 유사도는 어떻게 구할 수 있을까?

- Cosine 유사도

- $$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2} \sqrt{\sum_{i=1}^n B_i^2}},$$
- θ 가 작을 수록 1에 가까워 진다.



단어 가중치

어떤 단어가 얼마나 중요한가?

- 직관적으로 생각해보면,
 - 문서에 많이 등장하는 단어는 높은 가중치를 가져야 한다 (1개의 문서)
 - 빅데이터 라는 단어가 많이 등장하면, 빅데이터에 관련된 문서일 확률이 높다.
 - 많은 문서에서 등장하는 단어는 낮은 가중치를 가져야 한다
 - (나 / 너 / 우리 / 그리고) 등의 불용어들 (stopwords)
- 수학적으로 계산을 하려면?
 - 단어 빈도수 (Term frequency) : TF
 - 역문서 빈도수 (Inverse document frequency) : IDF

TF-IDF

너무 흔하지도 않지만, 너무 희귀하지도 않음

- TF

- Binary TF = {0, 1}
- Raw frequency TF = 빈도수
- Normalized TF = 빈도수/문서길이
 - 문서길이: 문서내의 전체 단어수
- Log-normalized TF = $1 + \log(\text{빈도수})$

weighting scheme	document term weight	query term weight
1	$f_{t,d} \cdot \log \frac{N}{n_t}$	$\left(0.5 + 0.5 \frac{f_{t,q}}{\max_t f_{t,q}}\right) \cdot \log \frac{N}{n_t}$
2	$1 + \log f_{t,d}$	$\log \left(1 + \frac{N}{n_t}\right)$
3	$(1 + \log f_{t,d}) \cdot \log \frac{N}{n_t}$	$(1 + \log f_{t,q}) \cdot \log \frac{N}{n_t}$

<https://en.wikipedia.org/wiki/Tf%E2%80%93idf>

- IDF

- $idf_t = \log \frac{N}{n_t}$
 - N: 전체 문서수, n_t : 단어 t를 포함하고 있는 문서의 수
 - 예를 들어 N=100, $n_t=50 \rightarrow idf = \log(2)$
 - N=100, $n_t=100 \rightarrow idf = \log(1) = 0$
- 해석: IDF가 높다? 단어가 등장하는 문서가 적다

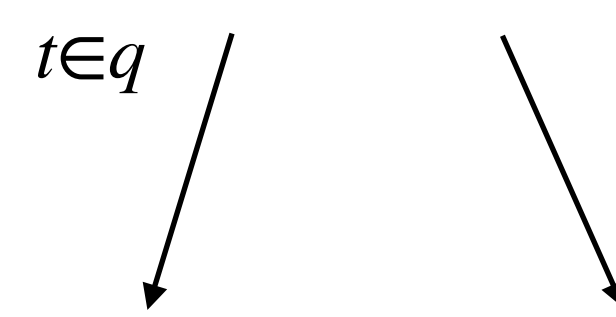
- TF-IDF

- TF와 IDF를 동시에 고려하기 위한 수식
 - $tf-idf = tf \cdot idf$
- TF는 해당 문서에서의 단어의 중요도를 나타내고
- IDF는 해당 단어의 문서 전체에서의 중요도를 나타냄

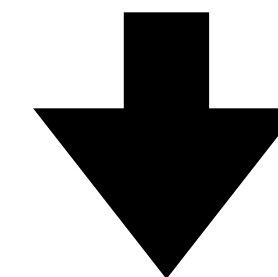
Boolean vs Term Weighting

Boolean 검색과의 차이점

- 단어의 수가 유사도에 영향을 미친다
- 단어의 가중치가 유사도에 영향을 미친다
- 무엇보다도! 문서의 Rank를 계산할 수 있다.

$$Score(q, d) = \sum_{t \in q} \omega_{t,q} \cdot \omega_{t,d}$$

$$\omega_{t,q} = \frac{tfidf_{t,q}}{\sqrt{\sum_t tfidf_{t,q}^2}} \quad \omega_{t,d} = \frac{tfidf_{t,d}}{\sqrt{\sum_t tfidf_{t,d}^2}}$$

$$cosine(d, q) = \frac{\sum_t \omega_{t,d} \cdot \omega_{t,q}}{\sqrt{\sum_t \omega_{t,d}^2} \sqrt{\sum_t \omega_{t,q}^2}}$$



$$cosine(d, q) = \frac{\sum_t tfidf_{t,d} \cdot tfidf_{t,q}}{\sqrt{\sum_t tfidf_{t,d}^2} \sqrt{\sum_t tfidf_{t,q}^2}}$$

BM25

Ranking function

- 엘라스틱 서치에서 기본적으로 사용하는 랭킹 알고리즘
 - 1980~1990년도 사이에 London's City University 의 Okapi information Retrieval system에 적용된 랭킹함수로 Full name은 Okapi BM25
 - 여러 실험을 통해 찾아낸 수식: Best Match! (아마 25번째 실험?)
- Term 가중치를 위한 세가지 핵심 원리
 - Inverse document frequency
 - Term frequency
 - Document length normalization

$$score(d, q) = \sum_{t \in q} idf_t \cdot \frac{f(t, d) \cdot (k_1 + 1)}{f(t, d) + k_1 \cdot \left(1 - b + b \cdot \frac{|d|}{avgdl} \right)}$$

BM25

BM25에서 직접 변경 가능한 파라미터

- k1: Term Frequency의 Scaling을 조절
 - 0: binary model: 문서에 단어가 있나? 없나?
 - 숫자가 커지면, 실제 term frequency를 사용하겠다는 의미
 - 일반적으로 1.2에서 2.0사이
- b: 문서 길이 Normalization
 - 0: No normalization
 - 1: Full length normalization
 - 일반적으로 0.75를 사용

$$score(d, q) = \sum_{t \in q} idf_t \cdot \frac{f(t, d) \cdot (k_1 + 1)}{f(t, d) + k_1 \cdot \left(1 - b + b \cdot \frac{|d|}{avgdl} \right)}$$

Elasticsearch

오픈소스 검색엔진

- Lucene 라이브러리 기반의 검색엔진
- REST API 형태로 접근
 - HTTP : PUT/DELETE/GET/POST
- 가장 대중적인 엔터프라이즈 검색엔진
- 최근 **ELK(Elasticsearch, Logstash, Kibana)** 스택이라는 빅데이터 수집 및 분석에 많이 사용됨



Elasticsearch

Elasticsearch는 어떻게 시작되었나?

런던의 아파트에서, Shay Banon은 일자리를 찾고 있었습니다. 그때 그의 아내는 요리 학교에 다니는 중이었습니다. Shay는 남은 시간을 이용해 점점 늘어나는 아내의 요리법 목록을 위한 검색 엔진을 만들기 시작했습니다. 최초의 버전은 컴퍼스(Compass, 2004)라고 불렸습니다. 두번째 버전이 (아파치 루신(Apache Lucene)을 기반으로 한) Elasticsearch(2010)였습니다.



Elasticsearch

이 이야기의 교훈은?

ESTC · NYSE

엘라스틱

+ 팔로우

< 공유

\$171.87 ↑ 145.53% +101.87 최대

시간외: \$171.87 (0.00%) 0.00

마감일: 10월 15일, 오후 4시 40분 47초 UTC-4 · USD · NYSE · 연복조항

1일 5일 1개월 6개월 YTD 1년 5년 최대



주식

미국 상장 증권

NL에 본사 소재

전일 증가 **\$173.29**

일일 변동폭 **\$171.02 - \$175.42**

52주 변동폭 **\$97.48 - \$177.74**

시가총액 **158.20억 USD**

수량 **82.46만**

주가수익률 **-**

배당수익률 **-**

기본 거래소 **NYSE**

Elasticsearch Overview

Elasticsearch와 관련된 용어들

- Cluster
- Node
- Shard
- Replica

시스템 아키텍트가 신경써야 할 부분
(Scalability, High availability, Fault tolerance and Disaster recovery)

- Index
- Documents
- Mappings
- Analyzer
- Scoring

데이터 과학자(우리)가 신경써야 할 부분

Elasticsearch Overview

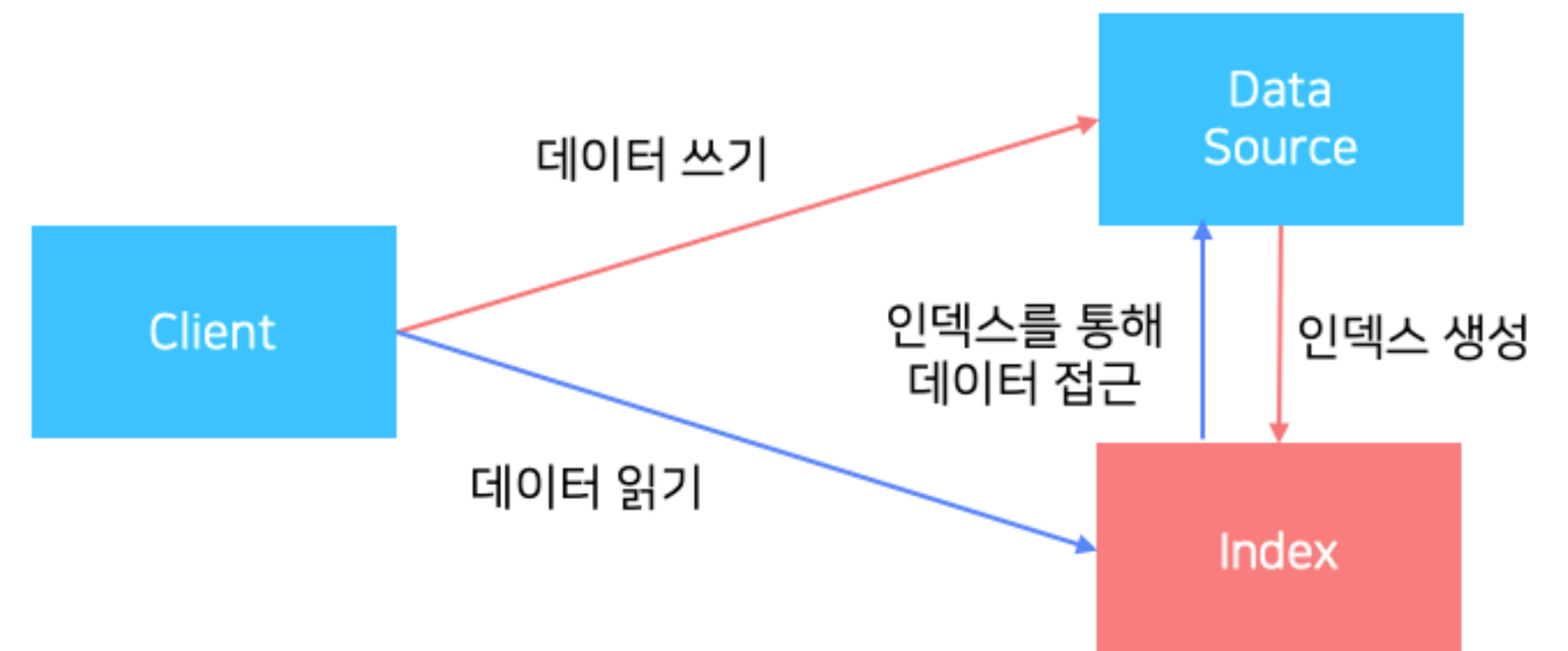
RDB vs ElasticSearch

관계형 데이터베이스 (mysql)	엘라스틱서치
Database	Index
Table	Type
Row	Document
Column	Field
Schema	Mapping
Index	모두 Index되어있음
SQL	Query DSL

Index

Index란?

- 원본 데이터에 빠르게 접근하기 위한 추가적인 데이터 집합
- 데이터가 저장될 때 그 데이터를 위한 인덱스 데이터를 자동으로 생성
- 데이터의 타입이나 목적성에 따라 구현 방식만 다를 뿐 결국에는:
 - 추가적인 저장용량을 쓰면서 검색 속도를 향상시키는 방법
 - B-tree, B+tree, Hash Table, Inverted index



Index

RDB vs ElasticSearch

Term	Document
Big	Doc1, Doc2, ..
Data	Doc1, Doc3, ...
...	...

Elasticsearch: $O(1)$

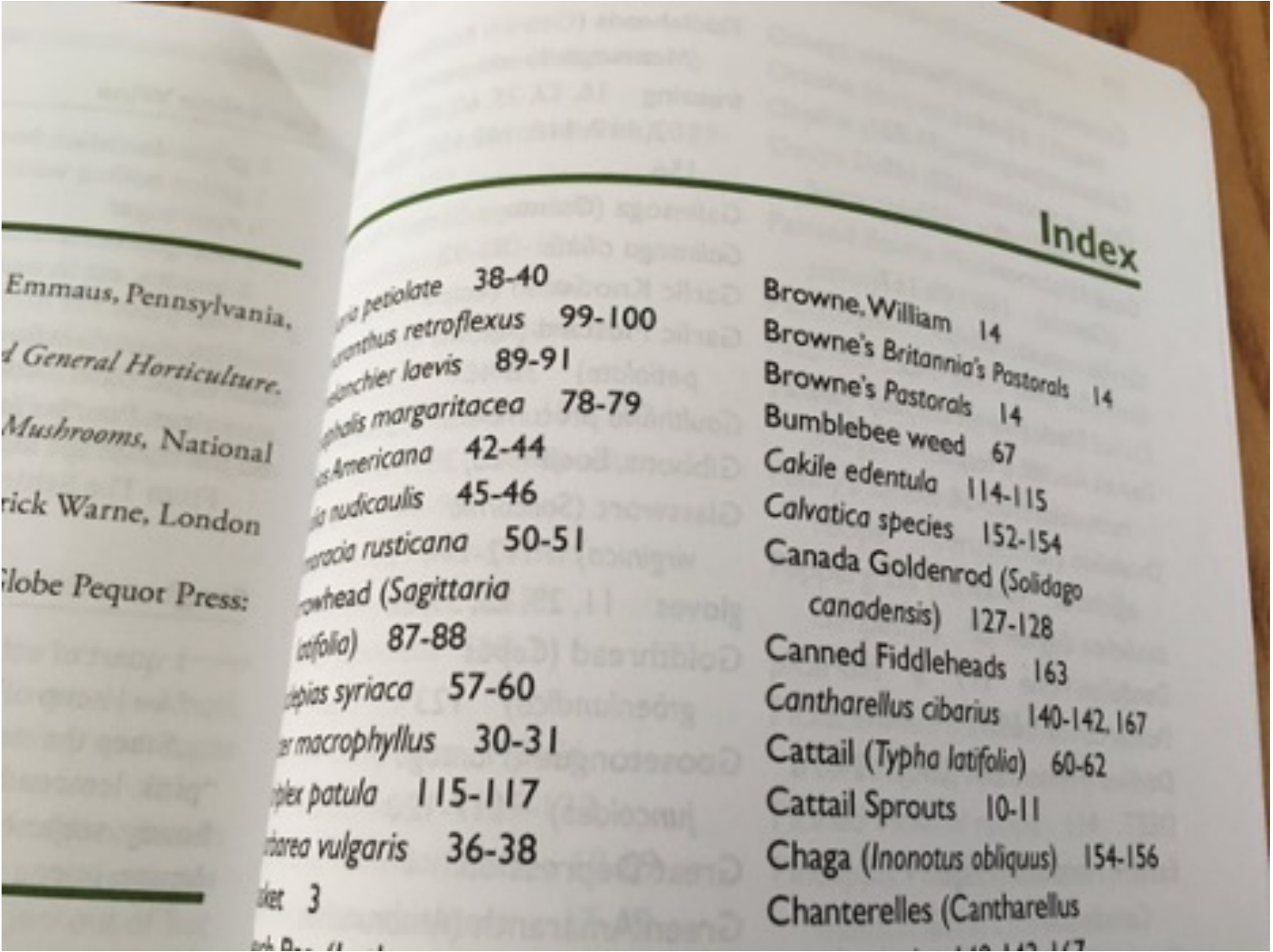
Document_id	Content
Doc1	Big data is very big
Doc2	Data science is science
...	...

RDB: $O(n)$

Seach : "Big"

Index

Index란?

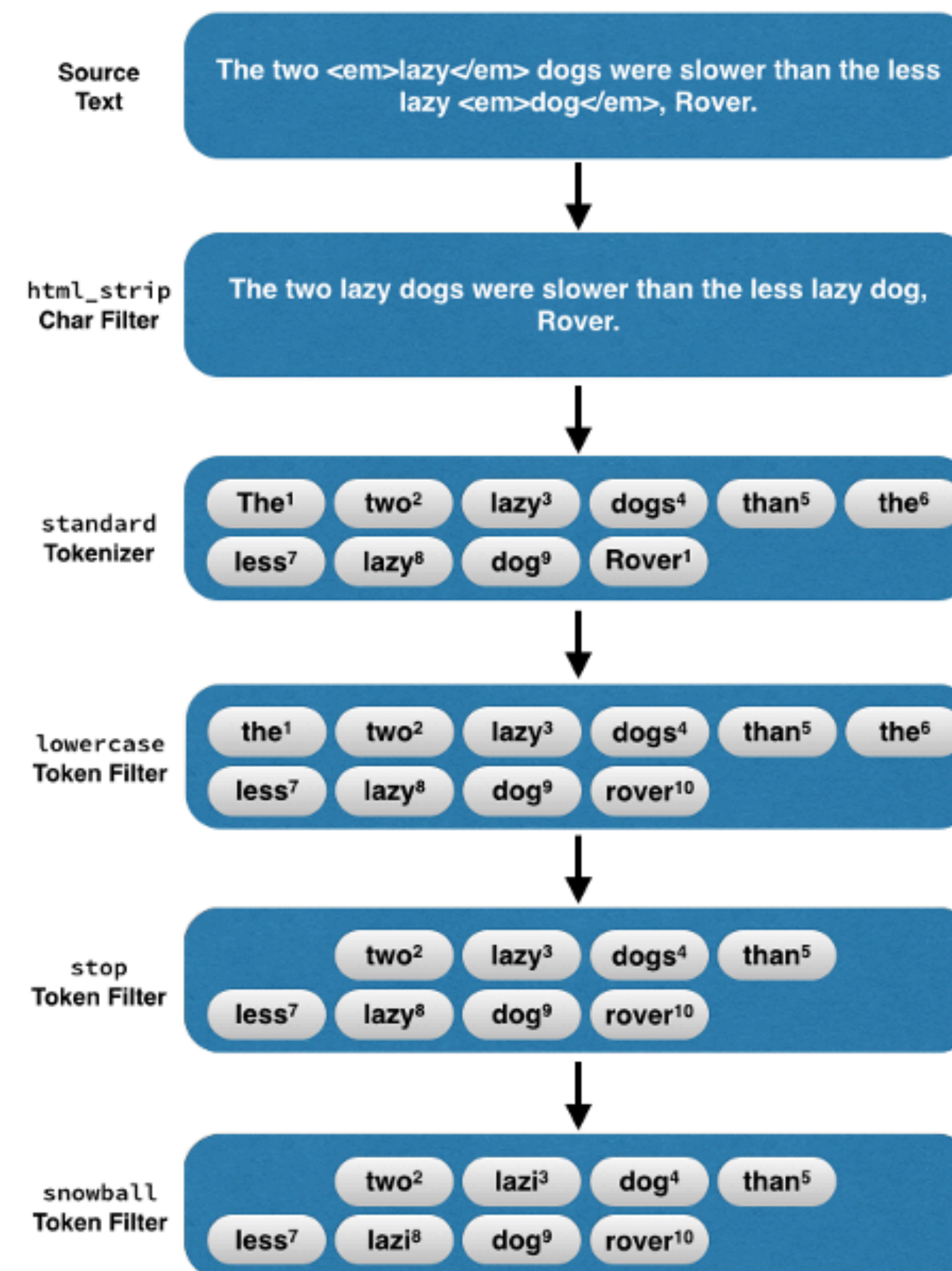


Term	Docs (doc_id, offset)
`The`	`{(1, 0), (1, 32), (2, 0)}`
`big`	`{(1, 4)}`
`brown`	`{(1, 8), (2, 4)}`
`fox`	`{(1, 10), (2, 14)}`
`jumped`	`{(1, 18)}`
`over`	`{(1, 27)}`
`lazy`	`{(1, 36)}`
`dog`	`{(1, 41)}`
`is`	`{(2, 14)}`
`Firefox`	`{(2, 17)}`

Elasticsearch Setting

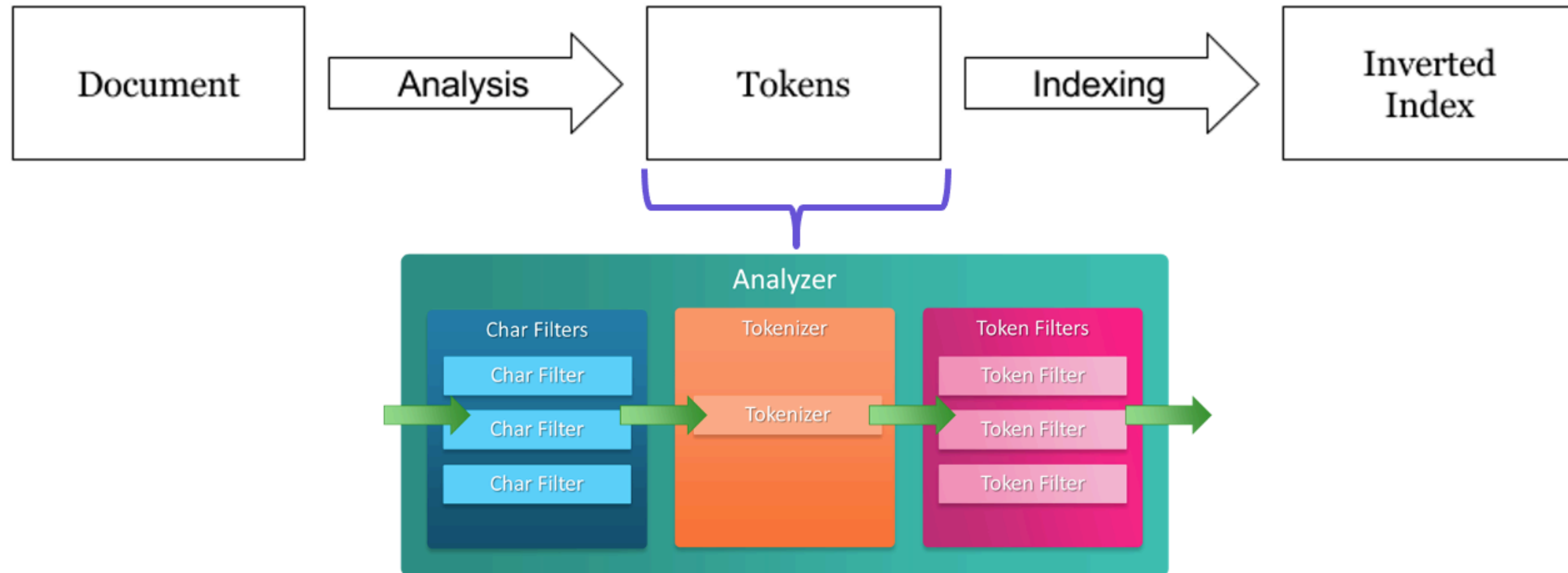
어떻게 Term을 추출해야 좋을까?

- Text mining 101스타일로!
 - 특수문자 제거, 원형복원, 불용어제거 등
- Analyzer를 활용
 - Tokenizer: 문장 자르기
 - Filter: 변형, 제거하기



Elasticsearch Setting

Analyzer



Elasticsearch Setting

Tokenizer: 어떤 기준으로 단어를 자를 것인가?

- Word Oriented Tokenizer
 - Standard, Letter, Whitespace
- Partial Word Tokenizer
 - N-gram, Edge N-gram
- Structured Text Tokenizer
 - Keyword, Pattern

Elasticsearch Setting

Filter : 어떤 단어를 어떻게 바꿀 것인가?

- Char Filter: Before tokenizing
 - HTML Strip Character Filter
 - Mapping Character Filter
 - Pattern Replace Character Filter
- Token Filter: After tokenizing
 - 우리가 알고 있는 대부분의 텍스트 전처리 기법들
 - stemmer, n-gram, stop words, shingle, uppercase, lowercase...

Elasticsearch Setting

Scoring : 쿼리와 문서간의 점수계산

- Query 또한 analyzer에 의해 분해가 되고, 이렇게 분해된 term들을 활용하여, 이전에 index된 문서와 점수 계산
- BM25가 기본이지만, 다양한 옵션을 제공
 - BM25
 - DFR
 - DFI
 - IB
 - LM Dirichlet, LM Jelinek Mercer

```
"similarity": {  
  "my_similarity": {  
    "type": "DFR",  
    "basic_model": "g",  
    "after_effect": "l",  
    "normalization": "h2",  
    "normalization.h2.c": "3.0"  
  }  
}
```

Elasticsearch Setting

Mappings

- 문서가 어떠한 필드들을 갖고 이들을 어떻게 인덱싱 할지를 정의하는 과정
- 최초에 인덱스를 생성할 때 설정
- 필요에 따라 각 field에 Analyzer도 정의

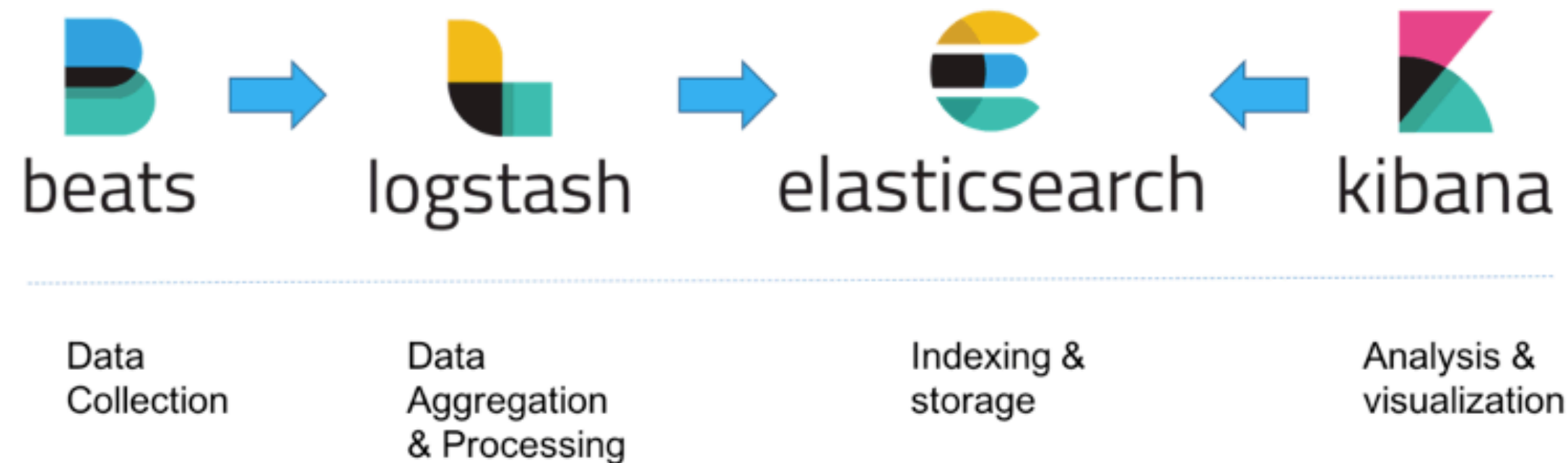
```
{
  "mappings": {
    "properties": {
      "age": { "type": "integer" }, ①
      "email": { "type": "keyword" }, ②
      "name": { "type": "text" } ③
    }
  }
}
```

```
"mappings":{
  "properties":{
    "title": {
      "type":"text",
      "analyzer":"my_analyzer", ③
      "search_analyzer":"my_stop_analyzer", ④
      "search_quote_analyzer":"my_analyzer" ⑤
    }
  }
}
```


Elasticsearch Setting

Elastic Stack

- 단순히 Search Engine으로써 이렇게 큰 회사가 됐을까?
 - “Elasticsearch는 비정형 데이터 검색에 최적화가 되어 있다!”
 - 응? 로그도 비정형 데이터인데?
 - 로그 분석, 모니터링, 이상탐지 등등에 활용



E.O.D