

P R E S E N T A T I O N


33주차 주제 **엘라스틱서치**

~Elasticsearch~

by mun

Elasticsearch


엘라스틱 서치



elastic

3.3k followers <https://www.elastic.co/> info@elastic.co

Pinned

 **elasticsearch** Public

Free and Open, Distributed, RESTful Search Engine

Java ☆ 68.5k 🍴 24.3k

- 오픈 소스 검색 엔진
- 역 인덱스 (역색인) 구조
- 분산 구성 가능
- HTTP기반의 RESTful API를 활용

색인과 역색인

색인

CHAPTER 1	1.1	인공지능, 머신러닝 그리고 딥러닝	018
딥러닝과	1.2	왜 텐서플로인가?	022
텐서플로의 만남			
CHAPTER 2	2.1	파이썬 및 필수 라이브러리 설치하기	026
텐서플로 설치와	2.2	텐서플로 예제 내려받고 실행해보기	028
주피터 노트북	2.3	주피터 노트북	029
CHAPTER 3	3.1	텐서와 그래프 실행	034
텐서플로		전체 코드	038
프로그래밍 101	3.2	플레이스홀더와 변수	038
		전체 코드	043
	3.3	선형 회귀 모델 구현하기	044
		전체 코드	050

- 문서에서 키워드를 찾아
보기 쉽도록 정렬 및 나열

역색인

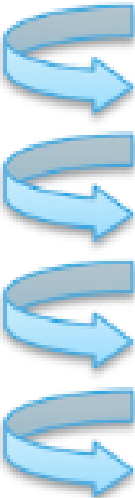
가중치 046, 054	머신러닝 018
강화학습 225	명령 프롬프트 029
강화학습 강화 비디오 목록 257	모델 재사용 074
경사하강법 047	목표 신경망 227
고수준 API 131	문서 요약 207
과적합 077, 099, 105, 108, 111, 126	물체 감출 API 221
교차 엔트로피 063	미니배치 096
구분자 149	
규칙 기반 인공지능 019	
군동분포 045	
그래프 037	
기계번역 174	
기본 신경망 227	
뉴런 052	
다층 신경망 068	
단어 자동 완성 184	
드롭아웃 106	
	변수 038
	분류 059
	비용 046
	비용 함수 063
	비지도 학습 138
	생성자 149
	선형 회귀 044
	셀 176
	세이프 035
	손실값 046

- 키워드를 통해 문서를 찾음

색인과 역색인

색인을 사용한 검색

CHAPTER 1	1.1	인공지능, 머신러닝 그리고 딥러닝	018
딥러닝과 텐서플로의 만남	1.2	왜 텐서플로인가?	022
CHAPTER 2	2.1	파이썬 및 필수 라이브러리 설치하기	026
텐서플로 설치와 주피터 노트북	2.2	텐서플로 예제 내려받고 실행해보기	028
	2.3	주피터 노트북	029
CHAPTER 3	3.1	텐서와 그래프 실행	034
텐서플로 프로그래밍 101		전체 코드	038
	3.2	플레이스홀더와 변수	038
		전체 코드	043
	3.3	선형 회귀 모델 구현하기	044
		전체 코드	050

	ID	Text	
	doc1	The quick brown fox	fox (O) → 선택
	doc2	The quick brown fox jumps over the lazy dog	fox (O) → 선택
	doc3	The quick brown fox jumps over the quick dog	fox (O) → 선택
	doc4	Brown fox brown dog	fox (O) → 선택
	doc5	Lazy jumping dog	fox (X) → 제외

- fox 키워드로 검색
- 테이블 데이터에서 LIKE '%fox' 검색을 통해 검색
- row 안의 내용을 모두 읽어야 해 속도가 느림
- 데이터가 늘어날수록 시간이 오래 걸린다

색인과 역색인

역색인을 사용한 검색

	
가중치 046, 054	머신러닝 018
강화학습 225	명령 프로그래밍 029
강화학습 강화 비디오 목록 257	모델 재사용 074
경시하강법 047	목표 신경망 227
고수준 API 131	문서 요약 207
과적합 077, 099, 105, 108, 111, 126	물체 감출 API 221
교차 엔트로피 063	미니배치 096
구분자 149	
규칙 기반 인공지능 019	
군동분포 045	변수 038
그래프 037	분류 059
기계번역 174	비용 046
기본 신경망 227	비용 함수 063
	비지도 학습 138
	
뉴런 052	
	
다층 신경망 068	생성자 149
단어 자동 완성 184	선형 회귀 044
드롭아웃 106	셀 176
	세이프 035
	순실값 046

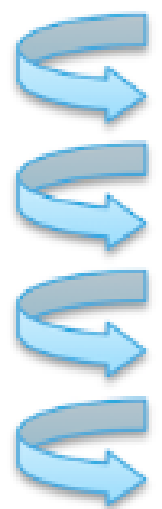
텀(Term)	ID	텀(Term)	ID
The	doc1, doc2, doc3	quick	doc1, doc2, doc3
brown	doc1, doc2, doc3, doc4	fox	doc1, doc2, doc3, doc4
jumps	doc2, doc3	er	doc2, doc3
the	doc2, doc3	razy	doc2
dog	doc2, doc3, doc4, doc5	Brown	doc4
Lazy	doc5		

텀(Term)	ID
fox	doc1, doc2, doc3, doc4

- 검색 엔진에서 추출된 각 단어(텀)를 조회
- fox를 포함하고 있는 문서의 id를 바로 얻을 수 있다
- 데이터가 늘어나도 행이 아닌 역색인 안의 id 배열에 값이 추가되므로 큰 속도 저하가 없다

색인과 역색인

색인

	ID	Text	
	doc1	The quick brown fox	fox (O) → 선택
	doc2	The quick brown fox jumps over the lazy dog	fox (O) → 선택
	doc3	The quick brown fox jumps over the quick dog	fox (O) → 선택
	doc4	Brown fox brown dog	fox (O) → 선택
	doc5	Lazy jumping dog	fox (X) → 제외

역색인

텀(Term)	ID	텀(Term)	ID
The	doc1, doc2, doc3	quick	doc1, doc2, doc3
brown	doc1, doc2, doc3, doc4	fox	doc1, doc2, doc3, doc4
jumps	doc2, doc3	over	doc2, doc3
the	doc2, doc3	lazy	doc2
dog	doc2, doc3, doc4, doc5	Brown	doc4
Lazy	doc5	jumping	doc5

역색인 방식으로 더 빠른
검색이 가능하다

색인과 역색인

Document 1

MSA 환경에서 데이터
관리를 위한 필수 사항

Document 2

서비스 경량화를 위한
MSA 설계 시 고려 사항

Document 3

MSA 서비스 아키텍처링을
위한 설계 방법

일반적인 RDB 행기반 검색

MSA	
ID	Text
1	MSA 환경에서 데이터 관리를 위한 필수 사항
2	서비스 경량화를 위한 MSA 설계 시 고려 사항
3	MSA 서비스 아키텍처링을 위한 설계 방법

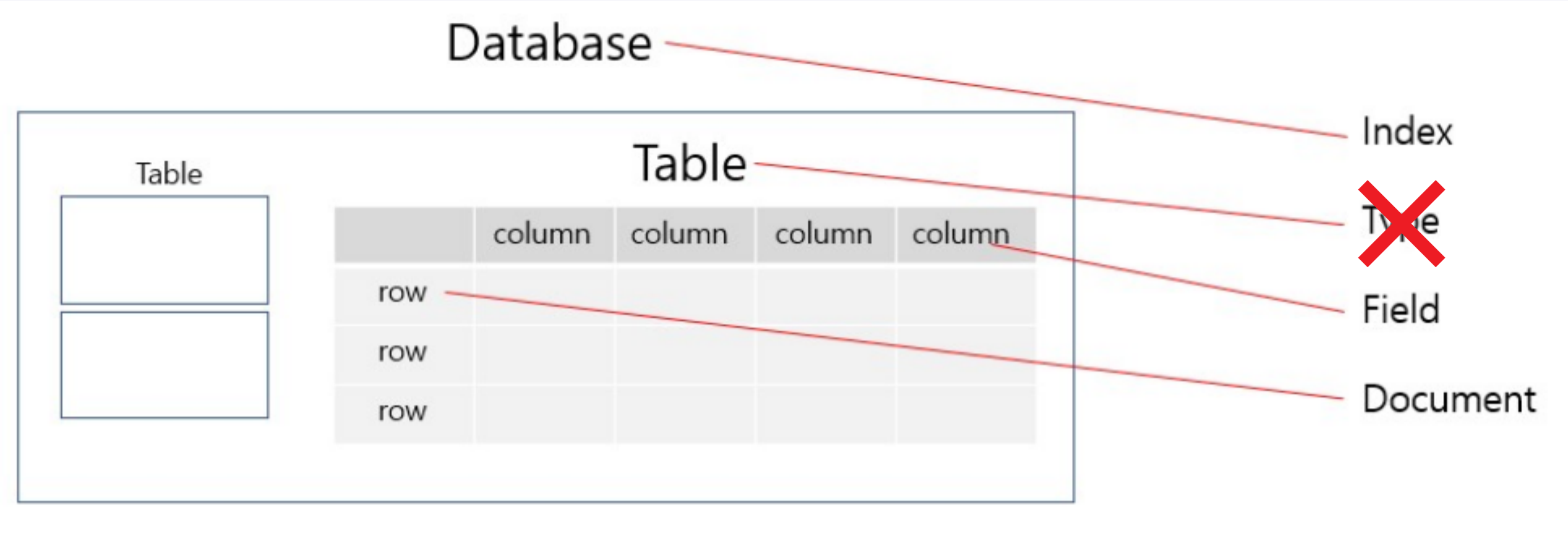
역인덱스 구조 검색

MSA		
ID	Term	Document
1	MSA	1, 2, 3
2	환경	1
3	데이터	1
4	관리	1
5	위한	1, 2, 3
6	필수	1
7	사항	1, 2
8	서비스	2, 3
9	경량화	2
10	설계	2, 3
11	고려	2
12	아키텍처링	3
13	방법	3

엘라스틱서치의 구조

ES와 RDBMS

ES	RDBMS
Index	Database
Shard	Partition
Type	Table
Document	Row
Field	Column
Mapping	Schema
Query DSL	SQL



엘라스틱서치의 구조



- **인덱스**
데이터의 저장 공간
클러스터 환경으로 구성했을 경우 하나의 인덱스가 여러 노드에 분산 저장된다
- **도큐먼트**
데이터의 최소 단위
Json 포맷으로 저장
하나의 도큐먼트는 다양한 필드로 구성
- **필드**
도큐먼트를 구성하는 속성
RDBMS의 Column과 유사하지만
여러 타입의 데이터가 존재할 수 있어 보다 동적이다

엘라스틱서치에서의 데이터 조작

ES와 RDBMS

ES HTTP Method	RDBMS SQL
GET	SELECT
PUT	INSERT
POST	UPDATE, SELECT
DELETE	DELETE
HEAD (인덱스 정보확인)	

- Document 관리 API (= single document api)
- 기본적으로 search api를 제공하지만, 색인된 문서의 ID 기준으로 한건 한건의 문서를 다룰 경우 document 관리 api를 이용
- REST API를 통해 데이터 조작을 지원

엘라스틱서치의 저장 방식

```
# Request
POST /my_index/_doc/1
{
  "message": "안녕하세요 ES"
}
```



```
# Response
{
  "_index" : "my_index",
  "_type" : "_doc",
  "_id" : "1",
  "_version" : 1,
  "result" : "created",
  "_shards" : {
    "total" : 2,
    "successful" : 1,
    "failed" : 0
  },
  "_seq_no" : 0,
  "_primary_term" : 1
}
```

- **_doc api**를 통해 단일데이터의 **CRUD**를 진행
- 메타데이터(문서의 내부 식별자, 버전 등)와 소스 데이터(문자열)를 **key-value**형태의 **JSON**으로 변형하여 저장

엘라스틱서치의 저장 방식

```
# Request
POST /my_index/_doc/1
{
  "message": "안녕하세요 ES"
}
```

id로 조회

- **_doc api**를 통해 단일데이터의 **CRUD**를 진행
- 메타데이터(문서의 내부 식별자, 버전 등)와 소스 데이터(문자열)를 **key-value**형태의 **JSON**으로 변형하여 저장

```
# Request
GET /my_index/_doc/1
```



```
# Response
{
  "_index" : "my_index",
  "_type" : "_doc",
  "_id" : "1",
  "_version" : 1,
  "_seq_no" : 0,
  "_primary_term" : 1,
  "found" : true,
  "_source" : {
    "message" : "안녕하세요 ES"
  }
}
```

엘라스틱서치의 저장 방식

```
# Request
POST /my_index/_doc/1
{
  "message": "안녕하세요 ES"
}
```

id로 조회

```
# Request
GET /my_index/_doc/1
```

```
# Response
{
  "_index" : "my_index",
  "_type" : "_doc",
  "_id" : "1",
  "_version" : 1,
  "seq_no" : 0,
  "primary_term" : 1
}
```

- `_doc` api를 통해 단일데이터의 CRUD를 진행
- 메타데이터(문서의 내부 식별자, 버전 등)와 소스 데이터(문자열)를 key-value형태의 JSON으로 변형하여 저장

그냥 NoSQL
아닌가요

엘라스틱서치의 저장 방식

```
# Request
POST /my_index/_doc/1
{
  "message": "안녕하세요 ES"
}
```

형태소 분할



Tokenizer, Filter 또는 Analyzer

안녕 ES

텍스트 타입은 형태소 분석되어
역색인 자료구조에 저장

- 유의어, 동의어, 전문검색을 위해 텍스트 형태의 데이터를 형태소 분할
- 단어 단위로 자르고 조사는 삭제, 문서별로 의미있는 데이터만 추출

엘라스틱서치의 저장 방식

단순 텍스트 형태의 메시지

```
POST /my_index/_doc/1
{"message": "wedge potato so yammy"}
```

```
POST /my_index/_doc/2
{"message": "potato better than sweet potato"}
```

형태소 분할



Term	id
potato	1,2
wedge	1
better	2
...	

역색인 자료구조에 저장

그대로 저장도 됨

```
# Response
{
  "_index" : "my_index",
  "_type" : "_doc",
  "_id" : "1",
  "_version" : 1,
  "result" : "created",
  "_shards" : {
    "total" : 2,
    "successful" : 1,
    "failed" : 0
  },
  "_seq_no" : 0,
  "_primary_term" : 1
}
```

- 특정한 토큰이 어떤 문서와 연관있는지 매핑됨

엘라스틱서치의 저장 방식

```
POST /my_index/_doc/1
{"message": "wedge potato so yammy"}
```

```
# Request
GET /my_index/_analyze
{
  "tokenizer": "standard",
  "text": "wedge potato so yammy"
}
```



**_analyze api 를 통해
분할 방식 확인 가능**

```
POST /my_index/_doc/2
{"message": "potato better than sweet potato"}
```

```
#Reponse
{
  "tokens" : [
    {
      "token" : "wedge",
      "start_offset" : 0,
      "end_offset" : 5,
      "type" : "<ALPHANUM>",
      "position" : 0
    },
    {
      "token" : "potato",
      "start_offset" : 6,
      "end_offset" : 12,
      "type" : "<ALPHANUM>",
      "position" : 1
    },
    {
      "token" : "so",
      "start_offset" : 13,
      "end_offset" : 15,
      "type" : "<ALPHANUM>",
      "position" : 2
    }
  ],
}
```


엘라스틱서치의 저장 방식

(위 생략)

potato와 일치하는 문서를 조회하는 쿼리

```
GET /my_index/_search
{
  "query": {
    "match": {
      "message": "potato"
    }
  }
}
```



_search api 를 이용해 전문 검색

- potato가 두 번 포함되어 있던 Document가 우선 검색됨
- 검색어와의 유사도를 0~1 사이인 score로 표현, 이에 따라 내림차순 정렬

```
"max_score" : 0.24309544,
"hits" : [
  {
    "_index" : "my_index",
    "_type" : "_doc",
    "_id" : "2",
    "_score" : 0.24309544,
    "_source" : {
      "message" : "potato better than sweet potato"
    }
  },
  {
    "_index" : "my_index",
    "_type" : "_doc",
    "_id" : "1",
    "_score" : 0.19100355,
    "_source" : {
      "message" : "wedge potato so yammy"
    }
  }
]
```

더 알아볼 점...

```
# Request
POST /my_index/_doc/1
{
  "message": "안녕하세요 ES"
}
```

형태소 분할



Tokenizer, Filter 또는 Analyzer

안녕 ES

텍스트 타입은 형태소 분석되어
역색인 자료구조에 저장

엘라스틱서치...
흥미롭다!!

텍스트를 분석해주는 Analyzer
Tokenizer, Stemming 등등...

**엘라스틱서치를 사용해
검색 성능을 높여보아요!**

감사합니다



참고

- <https://sihyung92.oopy.io/database/elasticsearch/1>
- https://www.youtube.com/watch?v=fBfUr_8Pq8A
- <https://jaemunbro.medium.com/elastic-search-%EA%B8%B0%EC%B4%88-%EC%8A%A4%ED%84%B0%EB%94%94-ff01870094f0>
- <https://www.elastic.co/kr/blog/found-elasticsearch-from-the-bottom-up>
- <https://steady-coding.tistory.com/581>
- <https://ksb-dev.tistory.com/311>
- https://www.samsungsds.com/kr/insights/elastic_data_modeling.html
- <https://danidani-de.tistory.com/52>
- <https://sihyung92.oopy.io/database/elasticsearch/1#2ce35ff3-c41c-45f1-a8aa-61b35bd45a7c>