



## 실무에 활용하는 Elasticsearch 검색엔진 구축

4일차 : ElasticSearch

1. Elasticsearch의 검색 질의 방식 및 각각의 질의 방식 별 특징
2. 검색 결과를 가져오는 순서의 변경 방법
3. Elasticsearch 랭킹 구조의 설계

## 오늘의 아젠다



- 1. Elasticsearch의 검색 질의 방식 및 각각의 질의 방식 별 특징
- 2. 검색 결과를 가져오는 순서의 변경 방법
- 3. 검색 결과를 가져오는 순서의 변경 방법

## 엘라스틱서치의 검색질의

# URI 요청 질의

주로 간단한 테스트 질의에 사용

예) curl -XGET 'localhost:9200/index1/book/\_search?q=title:엘라스틱&pretty=true'

## 주요 파라미터

q (query) : 필드명: 질의어 형태 루씬 기본 파서 사용

df(default field) : 검색할 필드 지정

default\_operator : 질의어 사이 공백값에 대한 AND/OR 처리

explain : 스코어 랭킹 계산식 출력

fields : 출력 결과를 표시할 필드 지정

sort : 검색 결과를 출력할 순서 결정 기본값 \_score기준 (필드명:desc)

from : 검색결과를 몇번째 값부터 출력할 지 여부 (페이징에 사용)

size : 검색결과를 몇개까지 표시 할것인지 여부 (페이징에 사용)

search\_type :

1. query\_then\_fetch : 전체 샤드검색이 모두 수행된후 출력(default)
2. query\_and\_fetch : 샤드별로 검색되는 대로 결과를 받아 출력 (size값만큼 각 샤드에서 검색함)
3. dfs\_query\_then\_fetch : 글로벌 문서 빈도를 계산하기위한 사전 쿼리를 수행
4. dfs\_query\_and\_fetch : 글로벌 문서 빈도를 계산하기위한 사전 쿼리를 수행
5. count - 검색된 문서를 배제하고 hit수만 출력
6. scan - scroll과 같이 사용하여 검색 결과를 바로 보여주지 않고 scroll에 저장 후 \_scroll\_id를 사용하여 나중에 출력

# Request Body 검색

## QueryDSL 사용

### 기본 형식

[http://localhost:9200/인덱스/타입/\\_search](http://localhost:9200/인덱스/타입/_search) -d ‘

```
{
  "_source" : {
    "include" : "c*",
    "exclude" : "*ry"
  },
  "from" : 1,
  "size" : 10,
  "fields" : ["title","category"],
  "sort" : [{"category":{"order":"desc","mode":"min"}}, {"pages"}, {"title"}],
  "highlight" : {
    "pre_tags" : ["<strong>"],
    "post_tags" : ["</strong>"],
    "fields" : {"author" : { }}
  },
  "query" : {
    "term" : { "_all" : "time" }
  }
}
```

옵션 영역

질의 영역

mode: 배열로 여러 값을 가진 경우 처리

1. min
2. max
3. avg
4. sum

# 검색 쿼리의 종류

텀레벨 쿼리	폴텍스트용 쿼리	복합쿼리	조인쿼리	GEO 쿼리	기타
term query <b>terms query</b> range query exists query prefix query wildcard query regexp query fuzzy query type query ids query	Match Query Match Phrase Query Match Phrase Prefix Query Multi Match Query Common Terms Query <b>Query String Query</b> Simple Query String Query	Constant Score Query Bool Query Dis Max Query Function Score Query Boosting Query Indices Query	Nested Query Has Child Query Has Parent Query Parent Id Query	GeoShape Query Geo Bounding Box Query Geo Distance Query Geo Distance Range Query Geo Polygon Query	More Like This Query Template Query Script Query

# TERM 쿼리 ,TERM 필터

## Term쿼리

[http://localhost:9200/인덱스/타입/\\_search](http://localhost:9200/인덱스/타입/_search) -d ‘

```
{
  "query" : {
    "term" : {
      "title" : "문재인"
    }
  }
}
```

## Term 필터

[http://localhost:9200/인덱스/타입/\\_search](http://localhost:9200/인덱스/타입/_search) -d ‘

```
{
  "query" : {
    "filtered" : {
      "query " : {
        "match_all" : {}
      }
    },
    "filter" : {
      "term": {
        "tags": "elastic"
      }
    }
  }
}
```

## Terms 쿼리

[http://localhost:9200/인덱스/타입/\\_search](http://localhost:9200/인덱스/타입/_search) -d ‘

```
{
  "query" : {
    "terms" : {
      "title" : ["문재인","대통령"],
      "minimum_should_match": 2
    }
  }
}
```

# RANGE 쿼리, RANGE 필터

## Range 쿼리

[http://localhost:9200/인덱스/타입/\\_search](http://localhost:9200/인덱스/타입/_search) -d ‘

```
{
  "query" : {
    "range" : {
      "created_on" : {
        "gt": "2017-01-01",
        "lt": "2017-10-01"
      }
    }
  }
}
```

GET \_search

```
{
  "query": {
    "range": {
      "date": {
        "gte": "now-1d/d",
        "lt": "now/d"
      }
    }
  }
}
```

## Range 필터

[http://localhost:9200/인덱스/타입/\\_search](http://localhost:9200/인덱스/타입/_search) -d ‘

```
{
  "query" : {
    "filtered" : {
      "query " : {
        "match_all" : {}
      }
    },
    "filter" : {
      "range": {
        "created_on" : {
          "gt": "2017-01-01",
          "lt": "2017-10-01"
        }
      }
    }
  }
}
```



# Prefix 쿼리, Wildcard 쿼리

## Prefix 쿼리

GET /\_search

```
{ "query": {  
  "prefix" : { "user" : { "value" : "ki", "boost" : 2.0 } }  
}
```

## Wildcard 쿼리

GET /\_search

```
{  
  "query": {  
    "wildcard" : { "user" : { "value" : "ki*y", "boost" : 2.0 } }  
  }  
}
```

# QUERY\_STRING 쿼리

## 기본 형식

[http://localhost:9200/인덱스/타입/\\_search](http://localhost:9200/인덱스/타입/_search) -d ‘

```
{
  "query": {
    "query_string" : {
      "fields" : ["content", "name"],
      "query" : "this AND that OR thus"
    }
  }
}
```

혹은

```
{
  "query": {
    "query_string": {
      "query": "(content:this OR name:this) AND (content:that OR name:that)"
    }
  }
}
```

루씬이 사용하는 boolean query parser를 그대로 이용함

name:search^ AND (tags:lucene OR tages:"big data"~2) AND description:analytics

# MATCH 쿼리, PHRASE 쿼리

## Match 쿼리

[http://localhost:9200/인덱스/타입/\\_search](http://localhost:9200/인덱스/타입/_search) -d ‘

```
{
  "query" : {
    "match" : {
      "title" : {
        "query": "Elastic Search",
        "operator": "and"
      }
    }
  }
}
```

## Phrase 쿼리

[http://localhost:9200/인덱스/타입/\\_search](http://localhost:9200/인덱스/타입/_search) -d ‘

```
{
  "query" : {
    "match" : {
      "title" : {
        "type": "phrase",
        "query": "elastic search",
        "slop": 1
      }
    }
  }
}
```

## Multi Match 쿼리

[http://localhost:9200/인덱스/타입/\\_search](http://localhost:9200/인덱스/타입/_search) -d ‘

```
{
  "query" : {
    "multi_match" : {
      "query": "Elastic Search",
      "fields": ["title^3", "author"],
      "tie_breaker": 0.3
    }
  }
}
```

## tieBreaker

0 (default) : 가장 높은 점수를 기록한 필드의 score만 최종 score에 반영

1 : 전체 필드의 score를 합산하여 반영

$(\text{맥스 필드 score}) + (\text{tieBreaker}) * (\text{나머지 필드 score의 합})$

# Bool 쿼리

POST \_search

```
{
  "query": {
    "bool" : {
      "must" : {
        "term" : { "user" : "kimchy" }
      },
      "filter": {
        "term" : { "tag" : "tech" }
      },
      "must_not" : {
        "range" : {
          "age" : { "gte" : 10, "lte" : 20 }
        }
      },
      "should" : [
        { "term" : { "tag" : "wow" } },
        { "term" : { "tag" : "elasticsearch" } }
      ],
      "minimum_should_match" : 1,
      "boost" : 1.0
    }
  }
}
```

bool 식이 스코어에 영향을 미치지 않으려면

bool filter

GET \_search

```
{
  "query": {
    "bool": {
      "filter": {
        "term": {
          "status": "active"
        }
      }
    }
  }
}
```

# Combined 쿼리

Bool 쿼리와 Match 쿼리의 조합을 많이 사용

```
{
  "query": {
    "bool": {
      "must": { "match": { "title": "quick" } },
      "must_not": { "match": { "title": "lazy" } },
      "should": [
        { "match": { "title": "brown" } },
        { "match": { "title": "dog" } }
      ]
    }
  }
}
```

```
{
  "query": {
    "bool": {
      "must": {
        "match": {
          "content": {
            "query": "full text search",
            "operator": "and"
          }
        }
      },
      "should": [
        { "match": {
          "content": {
            "query": "Elasticsearch",
            "boost": 3
          }
        } },
        { "match": {
          "content": {
            "query": "Lucene",
            "boost": 2
          }
        } }
      ]
    }
  }
}
```

# GEO Spatial 쿼리

GET /example/\_search

```
{
  "query":{
    "bool": {
      "must": {
        "match_all": {}
      },
      "filter": {
        "geo_shape": {
          "location": {
            "shape": {
              "type": "envelope",
              "coordinates" : [[13.0, 53.0], [14.0, 52.0]]
            },
            "relation": "within"
          }
        }
      }
    }
  }
}
```

INTERSECTS : 필드가 geometry 쿼리와 교차하는 모든 문서  
DISJOINT: geometry 쿼리 범위밖의 모든 문서  
WITHIN: geometry 쿼리 범위내의 모든 문서  
CONTAINS: geometry 쿼리가 포함된 모든 문서

GET /my\_locations/location/\_search

```
{
  "query": {
    "bool" : {
      "must" : {
        "match_all" : {}
      },
      "filter" : {
        "geo_distance" : {
          "distance" : "12km",
          "pin.location" : [-70, 40]
        }
      }
    }
  }
}
```

GET /\_search

```
{
  "query": {
    "bool" : {
      "must" : {
        "match_all" : {}
      },
      "filter" : {
        "geo_polygon" : {
          "person.location" : {
            "points" : [
              "40, -70",
              "30, -80",
              "20, -90"
            ]
          }
        }
      }
    }
  }
}
```

# NESTED 쿼리(중첩쿼리)

Nested 쿼리 : 중첩 매핑을 실시 할 경우 중첩쿼리가능

PUT /my\_index

```
{
  "mappings": {
    "type1" : {
      "properties" : {
        "obj1" : {
          "type" : "nested"
        }
      }
    }
  }
}
```

GET /\_search

```
{
  "query": {
    "nested" : {
      "path" : "obj1",
      "score_mode" : "avg",
      "query" : {
        "bool" : {
          "must" : [
            { "match" : { "obj1.name" : "blue" } },
            { "range" : { "obj1.count" : { "gt" : 5 } } }
          ]
        }
      }
    }
  }
}
```

# PARENT, CHILD 쿼리

## parent-child relationship

```
PUT my_index
{
  "mappings": {
    "my_parent": {},
    "my_child": {
      "_parent": {
        "type": "my_parent"
      }
    }
  }
}

PUT my_index/my_parent/1
{
  "text": "This is a parent document"
}

PUT my_index/my_child/2?parent=1
{
  "text": "This is a child document"
}

PUT my_index/my_child/3?parent=1&refresh=true
{
  "text": "This is another child document"
}
```

## Has\_Child 쿼리

```
GET my_index/my_parent/_search
{
  "query": {
    "has_child": {
      "type": "my_child",
      "query": {
        "match": {
          "text": "child document"
        }
      }
    }
  }
}
```

## children aggregation

```
GET my_index/_search
{
  "query": {
    "parent_id": {
      "type": "my_child",
      "id": "1"
    }
  },
  "aggs": {
    "parents": {
      "terms": {
        "field": "_parent",
        "size": 10
      }
    }
  },
  "script_fields": {
    "parent": {
      "script": {
        "inline": "doc['_parent']"
      }
    }
  }
}
```

## 제약조건

1. 부모와 자식의 타입은 달라야함.
2. \_parent.type 설정은 아직 존재하지 않은 타입만가능 (타입을 만든 이후에는 상위 유형을 만들수 없음)
3. 동일한 샤드에 색인되어야함 (parent id 라우팅에 사용)



# SPAN 쿼리 (근접 연산자 쿼리)

span\_term query

- span 쿼리의 기본 단위 , term query 와 동일

span\_multi query

- term, range, prefix, wildcard, regexp, or fuzzy query의 래핑 쿼리

span\_first query

- 필드 시작 부근에서 일치

span\_near query

- 질의된 텀들이 서로 가까이 있어야 검색되는 쿼리

span\_or query

- 질의된 텀을 OR로 검색

span\_not query

- 질의된 텀을 NOT으로 검색

span\_containing query

- 두개의 span 쿼리를 묶음

span\_within query

- 두개의 span 쿼리를 묶음

# SPAN 쿼리 (근접 연산자 쿼리)

span\_near query

```
{
  "query": {
    "span_near" : {
      "clauses" : [
        { "span_term" : { "field" : "value1" } },
        { "span_term" : { "field" : "value2" } },
        { "span_term" : { "field" : "value3" } }
      ],
      "slop" : 12,
      "in_order" : false
    }
  }
}
```

span containing query

```
{
  "query": {
    "span_containing" : {
      "A" : {
        "span_term" : { "field1" : "foo" }
      },
      "B" : {
        "span_near" : {
          "clauses" : [
            { "span_term" : { "field1" : "bar" } },
            { "span_term" : { "field1" : "baz" } }
          ],
          "slop" : 5,
          "in_order" : true
        }
      }
    }
  }
}
```

A의 일치 항목에 포함된 B의 일치 항목 반환

span\_within query

```
{
  "query": {
    "span_within" : {
      "A" : {
        "span_term" : { "field1" : "foo" }
      },
      "B" : {
        "span_near" : {
          "clauses" : [
            { "span_term" : { "field1" : "bar" } },
            { "span_term" : { "field1" : "baz" } }
          ],
          "slop" : 5,
          "in_order" : true
        }
      }
    }
  }
}
```

B의 일치 항목에 포함된 A의 일치 항목 반환

# Function Score 쿼리

## 1) function score 쿼리

```
{
  "query": {
    "function_score": {
      "query": { "match_all": {} },
      "boost": "5",
      "functions": [
        {
          "filter": { "match": { "test": "bar" } },
          "random_score": {},
          "weight": 23
        },
        {
          "filter": { "match": { "test": "cat" } },
          "weight": 42
        }
      ],
      "max_boost": 42,
      "score_mode": "max",
      "boost_mode": "multiply",
      "min_score": 42
    }
  }
}
```

## 2) score script 쿼리

```
{
  "query": {
    "function_score": {
      "query": {
        "match": { "message": "elasticsearch" }
      },
      "script_score": {
        "script": {
          "inline": "Math.log(2 + doc['likes'].value)"
        }
      }
    }
  }
}
```

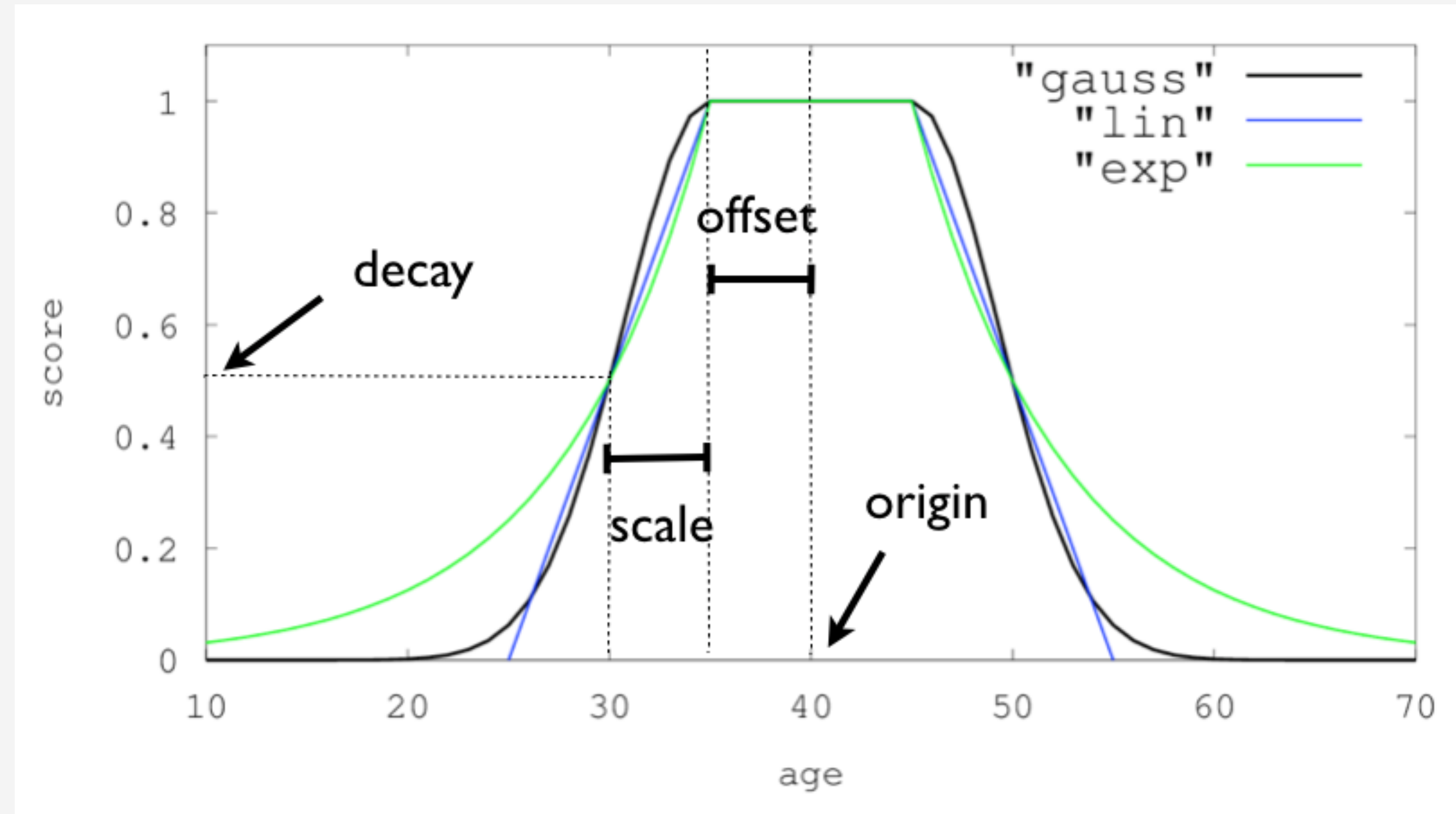
## 3) Field Value Factor

```
{
  "query": {
    "function_score": {
      "field_value_factor": {
        "field": "likes",
        "factor": 1.2,
        "modifier": "sqrt",
        "missing": 1
      }
    }
  }
}
```

# Decay functions 쿼리

특정 기준점에서 가까워지거나 멀어질때 스코어를 일정 값으로 쇠퇴(Decay) 시키는 함수

```
{
  "query": {
    "function_score": {
      "gauss": {
        "date": {
          "origin": "2013-09-17",
          "scale": "10d",
          "offset": "5d",
          "decay": 0.5
        }
      }
    }
  }
}
```



# MORE LIKE THIS 쿼리

주어진 문서 세트와 가장 유사한 문서를 텀벡터 기반으로 검색

```
PUT /imdb
{
  "mappings": {
    "movies": {
      "properties": {
        "title": {
          "type": "text",
          "term_vector": "yes"
        },
        "description": {
          "type": "text"
        },
        "tags": {
          "type": "text",
          "fields": {
            "raw": {
              "type": "text",
              "analyzer": "keyword",
              "term_vector": "yes"
            }
          }
        }
      }
    }
  }
}

GET /_search
{
  "query": {
    "more_like_this": {
      "fields": ["title", "description"],
      "like": "Once upon a time",
      "min_term_freq": 1,
      "max_query_terms": 12
    }
  }
}
```

→

```
"like" : [
  {
    "_index" : "imdb",
    "_type" : "movies",
    "_id" : "1"
  },
  {
    "_index" : "imdb",
    "_type" : "movies",
    "_id" : "2"
  },
  "and potentially some more text here as well"
],
"min_term_freq" : 1,
"max_query_terms" : 12
}
```

---



데이터 사이언스

# 루씬에서의 스코어 계산 (TF-IDF)

Lucene은 정보 검색 모델중 Boolean 모델과 벡터스페이스 모델이 결합된 형태로 결과를 도출  
Boolean 모델로 결과를 한정하고 벡터스페이스 모델로 스코어를 부여

벡터스페이스 모델에서의 문서와 쿼리는 다차원 공간에서 가중치 벡터로 표현됨  
각 텀은 차원이고 가중치는 TF-IDF 값

가중치 벡터의 내적(스칼라의 곱)  $\sum_{i=1}^n A_i \times B_i$  길이가 정규화된 가중 벡터의 내적으로 간주

$$\text{similarity} = \cos(\theta) = \frac{A \cdot B}{\|A\| \|B\|} = \frac{\sum_{i=1}^n A_i \times B_i}{\sqrt{\sum_{i=1}^n (A_i)^2} \times \sqrt{\sum_{i=1}^n (B_i)^2}}$$

두 벡터의 정규화된 길이

루씬이 추가한 컨셉

- 1) 단위벡터로 정규화 한것에 문제가 있다. (doc-len-norm(d))
- 2) 색인을 생성할때 특정 문서가 더 중요할 수 있다. (doc-boost(d))
- 3) 루씬은 필드기반이므로 필드별 부스트도 따로 있어야 한다.
- 4) 쿼리에서도 용어별로 가중치가 다르다 (query-boost(q))
- 5) 문서는 다차원 쿼리와 일치 할수 있다(coord-factor(q,d))

$$\text{score}(q, d) = \text{coord-factor}(q, d) \cdot \text{query-boost}(q) \cdot \frac{V(q) \cdot V(d)}{|V(q)|} \cdot \text{doc-len-norm}(d) \cdot \text{doc-boost}(d)$$

Lucene 개념적 채점 공식



# 루씬에서의 스코어 계산

TF(Term Frequency) :  $\text{sqrt}(\text{freq})$   
문서에서 해당 Term이 나온 횟수

IDF(Inverse Document Frequency) :  $\log(\text{numDocs}/(\text{docFreq}+1)) + 1$   
전체 문서에서 해당 Term이 나온 횟수의 역수

Coord :  $\text{overlap} / \text{maxOverlap}$

검색된 문서에서 쿼리의 Term이 몇 개 들어있는지에 대한 값  
OR 검색에서만 효과

예) 나이키 운동화 검색 => 나이키 매장 = coord 값은 1/2

lengthNorm :  $1/\text{sqrt}(\text{numTerms})$   
문서길이 대비 중요도

queryNorm :  $1/\text{sqrt}(\text{sumOfSquaredWeights})$

문서간의 비교에서 직접적 영향 없는 값 (하나의 쿼리에서는 동일한 값)  
쿼리간의 비교를 위한 정규화 값

$$\text{score}(q,d) = \text{coord}(q,d) \cdot \text{queryNorm}(q) \cdot \sum_{t \text{ in } q} ( \text{tf}(t \text{ in } d) \cdot \text{idf}(t)^2 \cdot t.\text{getBoost}() \cdot \text{norm}(t,d) )$$

Lucene Practical Scoring Function

$\text{score}(q,d) =$

$\text{Sigma}(t \text{ in } q) ( \text{tf}(t \text{ in } d) * \text{idf}(t)^2 * \text{getBoost}(t \text{ in } q) * \text{getBoost}(t.\text{field} \text{ in } d) * \text{lengthNorm}(t.\text{field} \text{ in } d) * \text{coord}(q, d) * \text{queryNorm}(q)$

$\text{lengthNorm}(t.\text{field} \text{ in } d) = 1/\text{sqrt}(\text{numTerms}) * f.\text{getBoost} * d.\text{getBoost}$

$\text{queryNorm}(q) = \text{queryNorm}(\text{sumOfSquaredWeights})$

$\text{sumOfSquaredWeights} = \text{Sigma}(t \text{ in } q)(\text{idf}(t) * \text{getBoost}(t \text{ in } q))^2$

# 검색 스코어 계산 예제

문서1: title필드 : 현대/카드/각/좋다 (텀갯수 3개)  
content필드 : 현대/카드/는/매우/좋은/카드/입니다.(전체 텀갯수 5개)  
문서가중치 : 3  
문서2: title필드 : 현대/자동차 (텀갯수 2개)  
content필드 : 현대/자동차/는/자동차/를/만듭니다.(전체 텀갯수 4개)  
문서가중치 : 5

필드별 가중치 : title(10) , content(1)

질의어 : 현대^3 OR 카드

## 문서1의 스코어 계산

텀의 빈도

전체문서수

텀의문서출현빈도

텀가중치

텀의갯수

문서가중치

필드가중치

title에서 “현대”

$\text{sqrt}(1) \times (\log(2/(2+1)) + 1)^2 \times 3 \times (1/\text{sqrt}(3) \times 3 \times 10)$

+

title에서 “카드”

$\text{sqrt}(1) \times (\log(2/(1+1)) + 1)^2 \times (1/\text{sqrt}(3) \times 3 \times 10)$

+

content에서 “현대”

$\text{sqrt}(1) \times (\log(2/(2+1)) + 1)^2 \times 3 \times (1/\text{sqrt}(5) \times 3 \times 1)$

+

content에서 “카드”

$\text{sqrt}(2) \times (\log(2/(1+1)) + 1)^2 \times (1/\text{sqrt}(5) \times 3 \times 1)$

35.2728135569

텀가중치가 “현대”가 높기 때문에 “카드”보다 더 높은 점수 받음

17.3205080757

2.73222038959

필드가중치가 낮기때문에 title보다 낮은 점수를 받음

1.8973665961

coord

$2/2 \times 1 \times 35.2728135569 + 17.3205080757 + 2.73222038959 + 1.8973665961 = 57.2229086183$



# 검색 스코어 계산 예제

문서1: title필드 : 현대/카드/각/좋다 (텀갯수 3개)  
content필드 : 현대/카드/는/매우/좋은/카드/입니다.(전체 텀갯수 5개)  
문서가중치 : 3  
문서2: title필드 : 현대/자동차 (텀갯수 2개)  
content필드 : 현대/자동차/는/자동차/를/만듭니다.(전체 텀갯수 4개)  
문서가중치 : 5

필드별 가중치 : title(10) , content(1)

질의어 : 현대^3 OR 카드

문서2의 스코어 계산

텀의 빈

전체문서

텀의문서출현빈

텀가중

텀의갯

문서가중

필드가중

Σ

title에서 “현대”  
sqrt(1) X (log(2/(2+1)) +1)<sup>2</sup> X 3 X (1/sqrt(2) \* 5\* 10)

+

title에서 “카드”  
카드에서 히트된 텀이 없음

+

content에서 “현대”  
sqrt(1) X (log(2/(2+1)) +1)<sup>2</sup> X 3 X (1/sqrt(4) \* 5\* 1)

+

content에서 “카드”  
카드에서 히트된 텀이 없음

72.000329172  
문서의 가중치가 문서1보다 높기 때문에 스코어 높


0

5.0911921005

0

coord

1/2 \* 1 \* 117.57604519+57.735026919+11.757604519 +6.32455532034 = 38.5457606364

Fast campus

데이터 사이언스

문서2가 문서 가중치에 의해서 높은 점수를 받았으나 TF-IDF 계산과 coord 계산으로 인해 문서1이 좀더 적합만 문서라고 스코어링됨

## 검색 스코어 계산 요약정리

- 1) 검색어(term)가 많이 일치할수록 랭킹이 높아진다. (Term Frequency 증가)
- 2) 검색어중에 흔한 단어의 경우 가중치가 낮아진다. (두 단어 이상으로 검색될시 의미 있음)
- 3) 여러 필드를 검색시에 가중치를 높게준 필드에 일치 할수록 랭킹이 높아진다.(필드별 가중치 적용)
- 4) 데이터가 짧은 필드에서 일치된 것과 데이터가 긴 필드에서 일치된 것중 짧은 쪽이 랭킹이 높아진다. (length Norm 적용)
- 5) 두개 이상의 단어 검색시에 Term 가중치를 줄 경우 Term 가중치를 받은 Term이 일치할 경우 랭킹이 높아진다.  
(Term Weight 적용)
- 6) 검색어가 반복적으로 입력될경우 Term 가중치와 비슷한 효과가 있다.  
  
예) love love love hate 라는 검색어는 love라는 단어에 3배의 가중효과를 부여한다.
- 7) 색인 시점에서 문서에 대한 가중치를 부여할경우 부여된 문서의 랭킹이 높아진다(문서 가중치 적용)

# BM25

## 2) BM(Best Match)25 순위 부여 알고리즘

$$score(Q, D) = \sum_{t \in Q} \log\left(\frac{N}{df_t}\right) \cdot \frac{tf_{td} \cdot (k_1+1)}{tf_{td} + k_1 \cdot ((1-b) + b \cdot \frac{L_d}{L_{avg}})}$$

×

확률 모델값

$$\log \frac{r/(R-r)}{(n-r)/(N-n-R+r)}$$

$\sum_{t \in Q}$

term에 대한 모든 것을 합산

$\log\left(\frac{N}{df_t}\right)$

IDF에 해당 함  
term을 가진 문서/전체 문서의 갯수(DF)를 역수 취하고 로그를 씌운 값 (N이 매우 크기 때문에)

$$\frac{tf_{td} \cdot (k_1+1)}{tf_{td} + k_1 \cdot ((1-b) + b \cdot \frac{L_d}{L_{avg}})}$$

TF는 텀의 빈도  
k1은 옵션 파라미터 (tf의 영향도를 얼마나 줄것인지)  
L<sub>d</sub> 현재 문서의 길이  
b는 옵션 파라미터 (문서의 길이의 영향도를 얼마나 줄것인지)  
  
실험적으로 k1은 1.2~2.0 사이, b는 0.75 이 부분은 튜닝의 영역

# 정보 검색의 모델 (확률 모델)

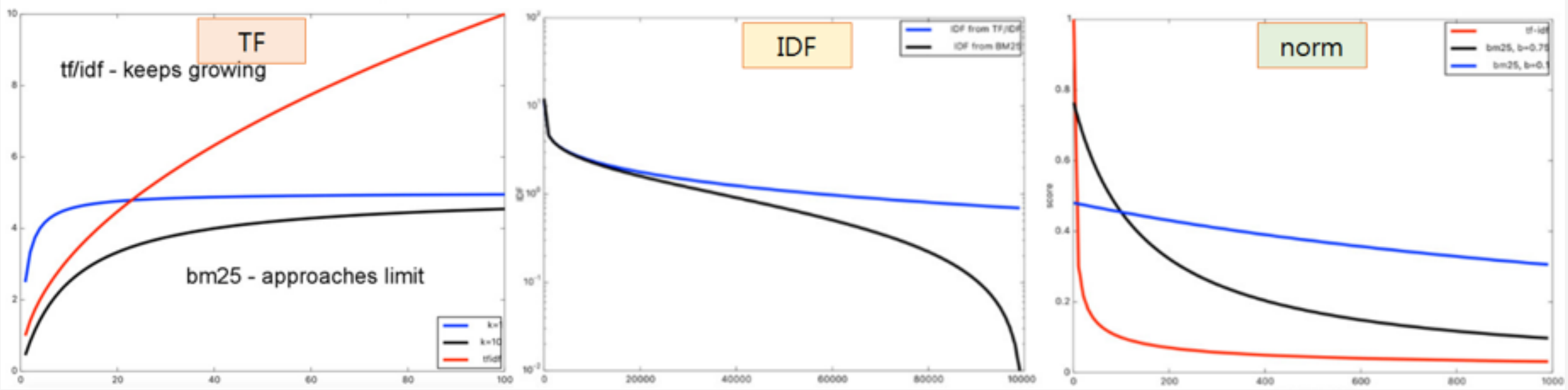
- 2) BM(Best Match)25 순위 부여 알고리즘
- 2진 독립 모델에서 문서와 질의 단어 가중치를 추가함.

$$bm25(d) = \sum_{t \in q, f_{t,d} > 0} \log \left( 1 + \frac{N - df_t + 0.5}{df_t + 0.5} \right) \cdot \frac{f_{t,d}}{f_{t,d} + k \cdot (1 - b + b \frac{l(d)}{avgdl})}$$

TF-IDF와 비교

$$score_{q,d} = norm(q) \times \sum_{t \text{ in } q} \sqrt{tf_{t,d}} \times idf_t^2 \times norm(d, field) \times boost(t)$$

$$bm25(d) = \sum_{t \in q, f_{t,d} > 0} \log \left( 1 + \frac{N - df_t + 0.5}{df_t + 0.5} \right) \cdot \frac{f_{t,d}}{f_{t,d} + k \cdot (1 - b + b \frac{l(d)}{avgdl})}$$



- TF의 영향 감소
- IDF 영향 증대
- 불용어가 검색 점수에 영향 덜 미침
- 문서 길이의 영향 줄어듦
- TREC에서 매우 좋은 성능 결과
- 엘라스틱 서치 5.0 이상 버전에서 기본으로 채택

# Okapi BM25 설정 방법

```
{
  "book" : {
    "properties" : {
      "title" : { "type" : "text", "similarity" : "my_similarity" }
    }
  }

  "similarity" : {
    "my_similarity" : {
      "type" : "BM25",
      "k1" : 1.2,
      "b" : 0.75,
      "discount_overlaps" : false
    }
  }
}
```

감사합니다.

다음 주제 : 7일차 Elasticsearch Chapter 3  
Aggregation  
기타 고급 검색 기법