실무에 활용하는 Elasticsearch 검색엔진 구축

5일차 : ElasticSearch
Aggregation
기타 고급검색기법

오늘의 아젠다

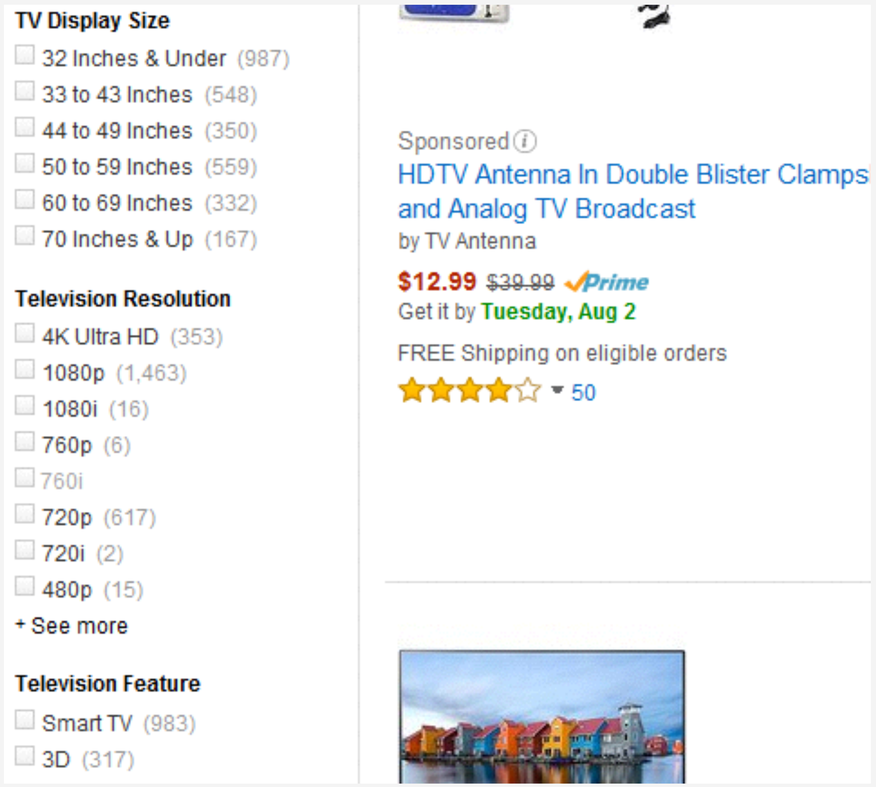
1. Aggregation

2. 기타 고급검색 기법
   - Suggesters (스펠체커)
   - 연관검색어 추출 기법

1. Aggregation

Aggregation의 개요

Facet
- Term Facet
- Range Facet
- Date Facet
- Statistical Facet
- geo distance Facet



Aggregations
버켓 단위의 도큐먼트 셋에 대해서 지속적인 추적과 연산 수행
-최소(min),최대(max),합(sum),평균(avg),개수(value count)

- Term aggregations
- Range aggregations
- Data aggregations

aggregation은 “facet 의 재 탄생” 입니다.

Aggregation의 구성요소

Bucketing

기준을 만족하는 문서 집합

조건에 해당하는 도큐먼트를 버킷이라는 저장소에 저장하는 단위

Metric

우리가 알고 싶은 정보

Bucket에 포함된 문서에 대한 통계정보 (min,max,avg, etc ..)

Matrix

여러 필드에서 작동하고 요청된 문서 필드에서 추출한 값을 기반으로 행렬 결과를 생성하는 집합

Pipeline

집계의 결과를 다른 집계에 활용한다.

레벨에 따라 부모/자식 집계로 나뉜다.

buckets_path 지정 필수

ex)

metric bucket

SELECT count(hit) FROM table GROUP BY hit

Aggregation의 기본 문법

```
"aggregations" : {
   "<aggregation_name>" : {
      "<aggregation_type>" : {
         <aggregation_body>
      }
      [,"meta" : {  [<meta_data_body>] } ]?
      [,"aggregations" : { [<sub_aggregation>]+ } ]?
   }
   [,"<aggregation_name_2>" : { ... } ]*
}
```

ex)

```
{
      “aggs” : {
            “terms” : {
                  “field” : “color”
            }
      }
}
```

```
select color,count(*) as colors
from cars
group by color
```

Aggregation의 Scope

1. 어그리게이션을 검색 요청과 함께 실행하면 질의 결과내(Scope) 안에서 수행
2. 질의 가 생략된 어그리게이션은 모든 문서에 대한 질의로 간주(MatchAllQuery)
3. 글로벌 버킷을 통해 질의가 포함된 어그리게이션도 모든 문서에 대한 어그리게이션 계산 가능

예)

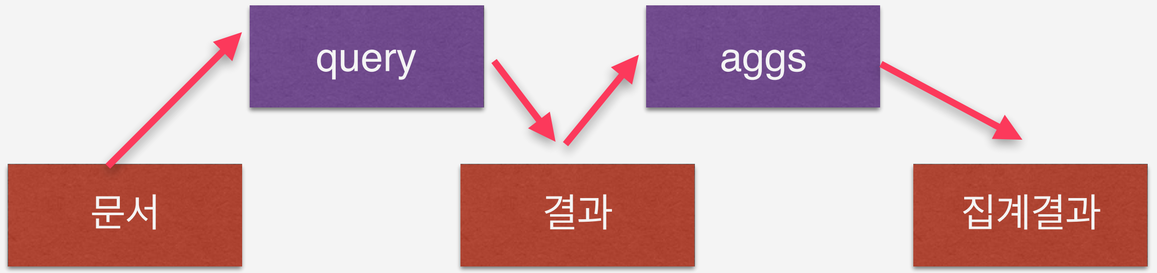
```
"aggs" : {
        "avg_price" : {
              "avg" : { "field" : "price"}
        },
       "all_price" : {
          "global" : {},
          "aggs" : {
             "avg_price" : {
                "avg" : { "field" : "price"}
             }
          }
       }
}
```

Aggregation의 필터

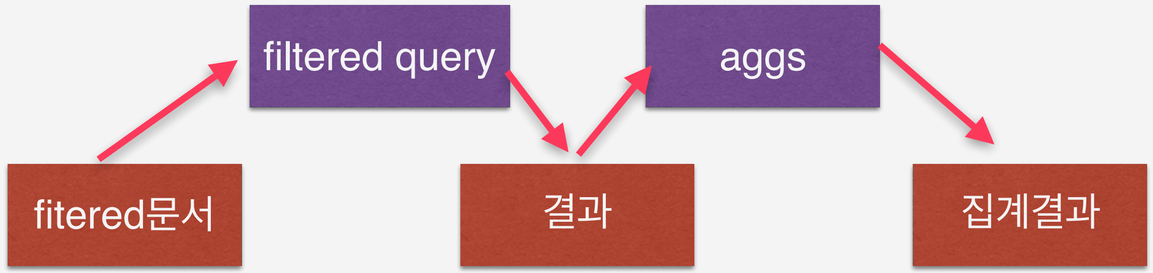
검색 질의가 아니라 에그리게이션 내부에서 직접 필터링 하여 적용 할수 있음

예)

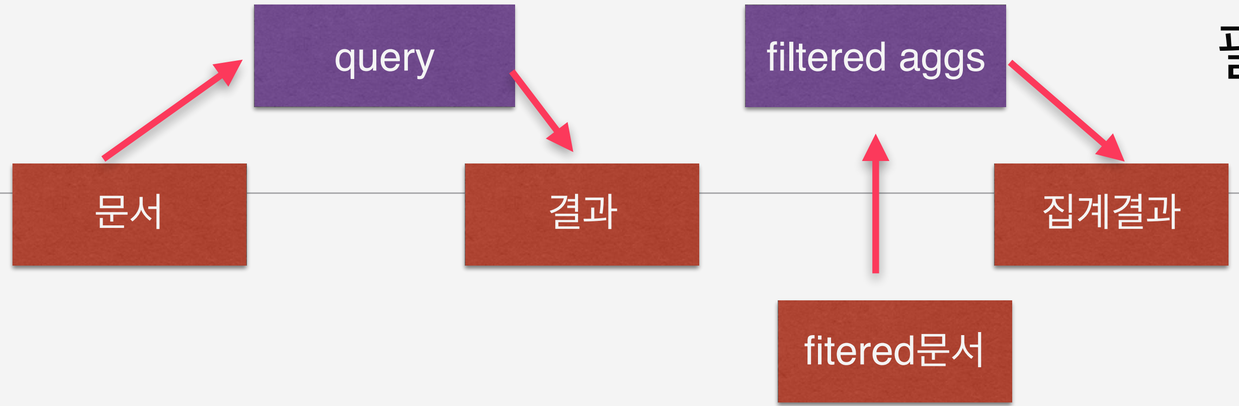
```
"aggs" : {
    "recent_sales" : {
        "filter" : {
            "range" : "{
                "sold" : {
                    "from" : "now-1M"
                }
            }
        }
    },
    "aggs" : {
        "avg_price" : {
            "avg" : { "field" : "price"}
        }
    }
}
```



일반 질의



필터 질의



필터 버킷

Metric의 종류

Avg Aggregation
Cardinality Aggregation
Extended Stats Aggregation
Geo Bounds Aggregation
Geo Centroid Aggregation
Max Aggregation
Min Aggregation
Percentiles Aggregation
Percentile Ranks Aggregation
Scripted Metric Aggregation
Stats Aggregation
Sum Aggregation
Top hits Aggregation
Value Count Aggregation

Metric 문법 구조

```
GET /{index_name}/_search?pretty
{
  "size": 0,
  "aggs": { // 집계
    "my_aggs": { // 집계 명 (사용자 정의)
      "{metric_arrgs_type}": { // 집계 타입
        "field": "{file_name}" // 집계 대상 필드
         "order" : { "{field_name}" : "desc" } // 정렬
      }
    }
  }
}
```

Bucket의 종류

Adjacency Matrix Aggregation
Children Aggregation
Date Histogram Aggregation
Date Range Aggregation
Diversified Sampler Aggregation
Filter Aggregation
Filters Aggregation
Geo Distance Aggregation
GeoHash grid Aggregation
Global Aggregation
Histogram Aggregation
IP Range Aggregation
Missing Aggregation
Nested Aggregation
Range Aggregation
Reverse nested Aggregation
Sampler Aggregation
Significant Terms Aggregation
Terms Aggregation

Bucket 문법 구조

```
GET /{index_name}/_search?pretty
{
  "size": 0,
  "aggs": { // 집계
    "my_aggs": { // 집계 명 (사용자 정의)
      "{bucket_arrgs_type}": { // 집계 타입
        "field": "{file_name}" // 집계 대상 필드
        "order" : { "{field_name}" : "desc" } // 정렬
      }
    }
  }
}
```

Significant Terms Aggregation

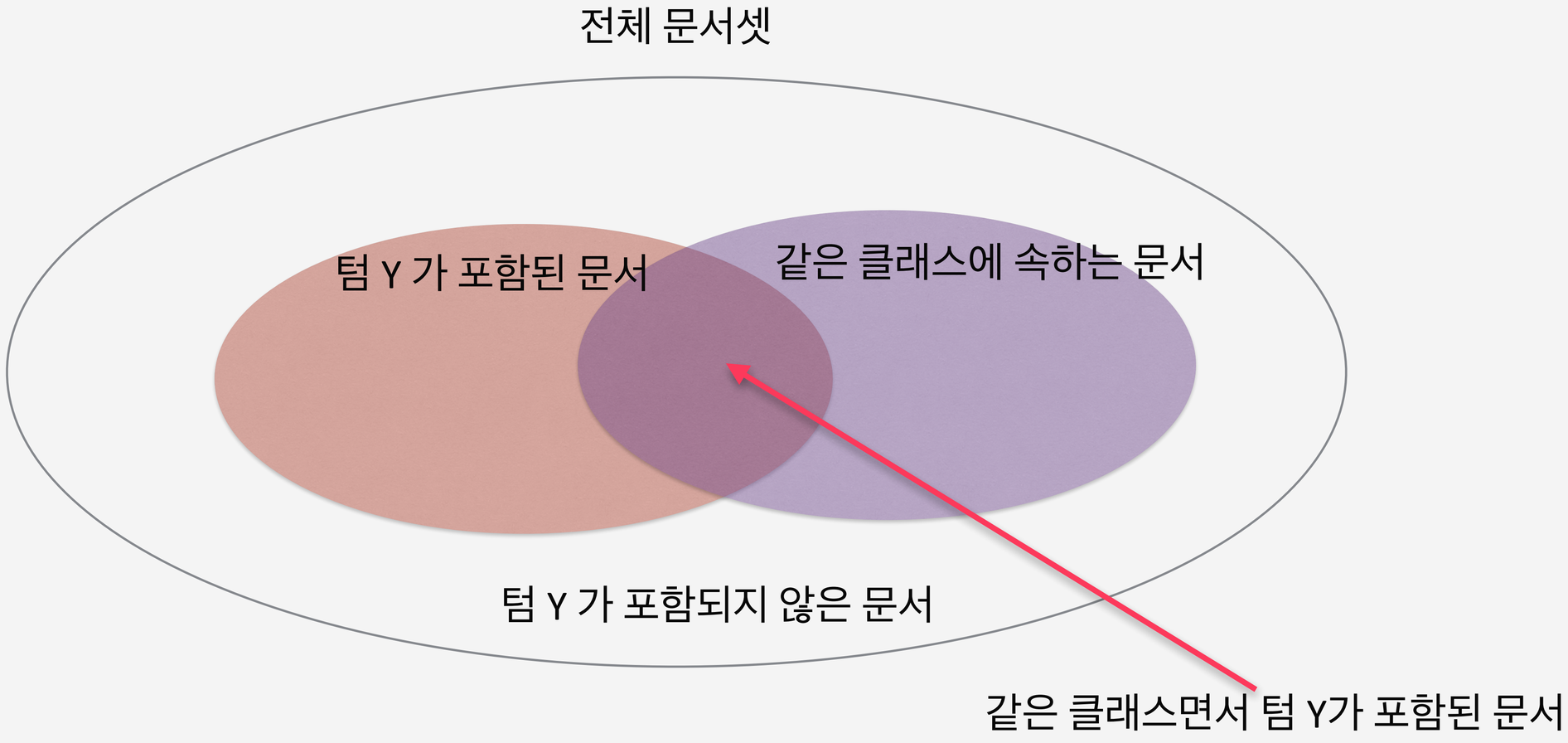
통계적으로 의미 있는 데이터를 찾아냄
데이터 집합에서 Uncommonly common 텀즈를 찾는것이 목적

Uncommonly common 데이터

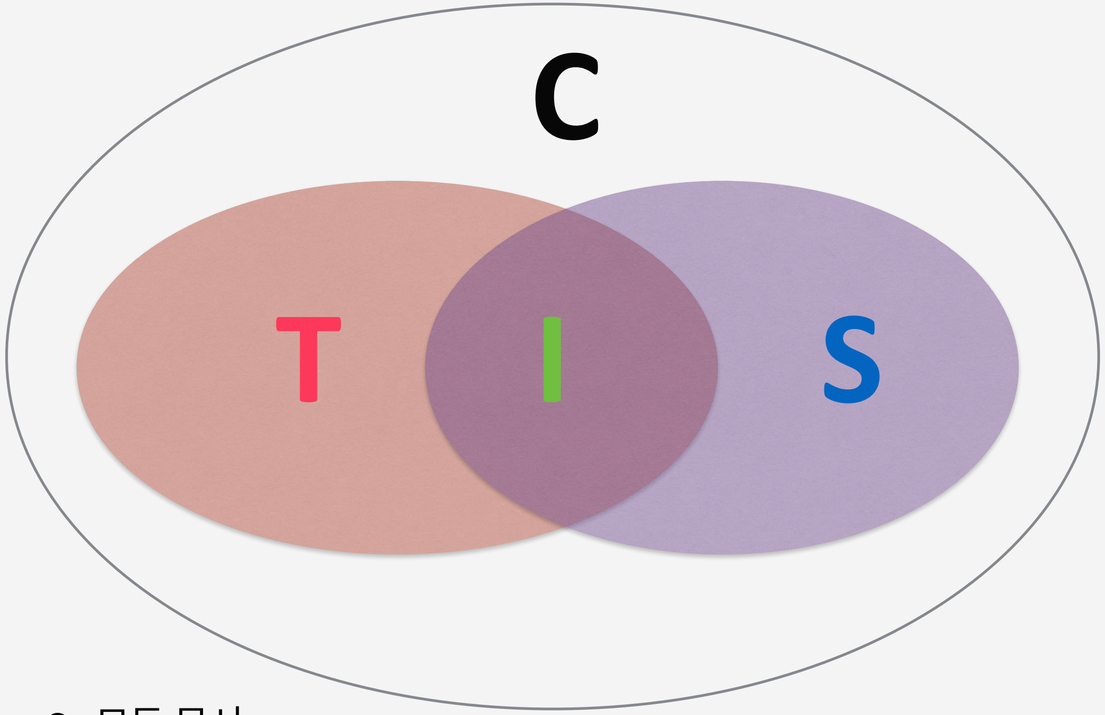
- 비정상적으로 자주 발생하는 데이터
- 일반적인 통계적 배경 오류보다 더 빈번한 데이터
- 통계적으로 예외적인 데이터
- 관측자가 주목해야 하는 관심데이터

주) 내부적 통계치 계산을 수반하기 때문에 큰 색인에서는 매우 무거울수 있음

Significant Terms Aggregation



Significant Terms Aggregation



C : 모든 문서
T : 텀이 포함된 문서
S : 검색매칭된 서브셋의 문서
I : 텀이 포함된 문서중 서브셋 문서

background rate = 배경문서(텀이 포함된 문서)/ 전체문서
foreground rate = 텀이포함된 서브셋문서/ 전체 서브셋 문서
score 계산
= (absolute change) * (relative change)
= |foreground rate - background rate|
*(foreground rate/background rate)

$$s(term) = \left(\frac{I}{S} - \frac{T}{C}\right) \cdot \left(\frac{I}{S} \Big/ \frac{T}{C}\right)$$

Significant Terms Aggregation

일반적인 빈도 기반 계산으로는
A 회사가 비정적인 회사로 계산됨

하지만

- A회사는 많은 트랜잭션이 발생하므로 분석하지 않아도 비정상 쉽게 드러남 (commonly common)
- D 회사는 희박하게 발생되지만 비정상 트랜잭션이 없음 (commonly uncommon)
- X 회사는 소수의 트랜잭션이 발생하지만 그 안에서 비정상 트랜잭션도 X 회사의 트랜잭션대비 높게 나타남 (uncommoly common)

Significant Terms Aggregation (2)

결과 예)

```
"aggregations" : {
      "significantCrimeTypes" : {
        "doc_count": 47347,
        "buckets" : [
          {
            "key": "Bicycle theft",
            "doc_count": 3640,
            "score": 0.371235374214817,
            "bg_count": 66799
          }
          ...
        ]
      }
  }
```

일반적인 집합 셋

배경 집합 셋

background rate = 66799 (bg_count)/ 100000 (total)

foreground rate = 3640(doc_count)/ 47347 (그룹의 갯수)

score 계산

= (absolute change) * (relative change)

= |foreground rate - background rate|*(foreground rate/background rate)

foreground 그룹 : 특정한 조건에 대한 사용자 그룹에 대해서 높은 빈도 분석

background 그룹 : 모든 그룹에 대해서 높은 빈도 분석

Aggregation의 정렬

1) 버킷의 경우

_count : 결과 문서의 갯수 기반 정렬 (기본값)

terms.histogram,data_histogram 버킷 적용

_term : 텀의 알파벳 값으로 정렬

_key : 버킷의 키값으로 정렬

2) 메트릭의 경우

계산된 계산값에 의해 정렬

예)

```
"aggs" : {
    "color" : {
      "terms" : {
        "field" : "color",
        "order" : "{
           "avg_price" : "asc"
        }
      }
    },
   "aggs" : {
     "avg_price" : {
        "avg" : { "field" : "price"}
     }
```

Pipeline의 Aggregation

집계의 결과를 다른 집계에 활용
레벨에 따라 부모/형제 집계로 구분
buckets_path 지정 필수

buckets_path 문법

AGG_SEPARATOR = '>'
METRIC_SEPARATOR = '.'
AGG_NAME = 어그리게이션의 이름
METRIC = 메트릭 이름
PATH = <AGG_NAME> [<AGG_SEPARATOR>, <AGG_NAME>]* [<METRIC_SEPARATOR>, <METRIC>]

예)
"my_bucket>my_stats.avg"

Pipeline의 종류

Avg Bucket Aggregation
Derivative Aggregation
Max Bucket Aggregation
Min Bucket Aggregation
Sum Bucket Aggregation
Stats Bucket Aggregation
Extended Stats Bucket Aggregation
Percentiles Bucket Aggregation
Moving Average Aggregation
Cumulative Sum Aggregation
Bucket Script Aggregation
Bucket Selector Aggregation
Serial Differencing Aggregation

파이프 라인 집계 구조

```
{
  "size": 0,
  "aggs": {
    "{parent_aggs_name}": { // 부모 집계 이름 (사용자 정의)
      "{bucket_type_name}": { // 버킷 집계 타입 이름
        "field": "{field_name}" // 버킷 집계 대상 필드
      },
      "aggs": {
        "{aggs_name}": { // 집계 이름 (사용자 정의)
          "{metric_aggs_type}": { // 매트릭 집계 타입
            "field": "{field_name}" // 매트릭 집계 대상 필드
          }
        }
      }
    },
    "{pipe_arrgs_name}": { // 파이프라인 집계 이름(사용자정의)
      "{pipe_arrgs_type}": { // 파이프라인 집계 타입
        "buckets_path": "{parent_aggs_name} >{aggs_name}" // 경로
      }
    }
  }
}
```

Aggregation의 근사화 (Cardinality Aggregation)

성능을 위해 정확도를 100% 보장하지 않음

- Cardinality Aggregation (HLL 근사화 알고리즘 사용)
- 유일하고 고유한(unique, distinct) 카운트를 제공함
  SELECT COUNT(DISTINCT color) FROM cars
- precision_threshold 옵션으로 정확도를 조절
  0~40000까지 조절
  카디널러티 수가 위 값보다 적으면 정확도 100%
  메모리 사용량은 대략 precision_threshold * 8 byte
- 카디널러티 계산시 입력값을 해싱하여 넣으면 성능 향상

```
....
   "color" : {
      "type" : “string",         {
       "fields" : {
            "hash" : {
                 type : “murmur3”
            }
       }
   }
}
....
```

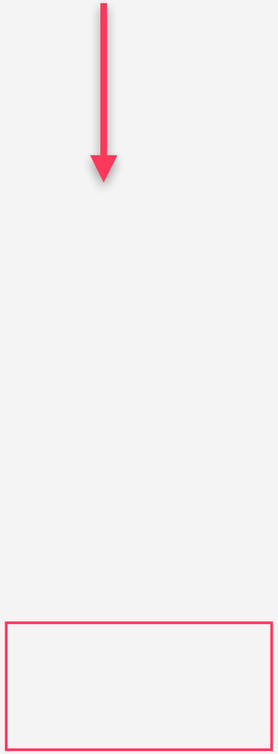
Aggregation의 근사화 (Percentiles(백분위) Aggregation)

아웃라이어(outlier, 이상치) 를 찾을 때 사용

전체 데이터에 대해 평균(mean) 또는 중앙값(median)만을 보는 경우

```
{
   "aggs" : {
      "load_time_outlier" : {
         "percentiles" : {
            "field" : "load_time"
         }
      }
   }
}
```

```
"aggregations": {
   "load_time_outlier": {
      "values" : {
         "1.0": 15,
         "5.0": 20,
         "25.0": 23,
         "50.0": 25,
         "75.0": 29,
         "95.0": 60,
         "99.0": 150
      }
   }
}
```



Aggregation 사용시 주의점

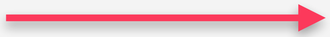
1. aggregation은 정확한 값을 보장하지 않는다. (size, shard_size 조절)
   Calculating Document Count Error 값을 살피자

```
{
  ...
  "aggregations" : {
    "products" : {
      "doc_count_error_upper_bound" : 46,
      "buckets" : [
        {
          "key" : "Product A",
          "doc_count" : 100
        },
        {
          "key" : "Product Z",
          "doc_count" : 52
        },
        ...
      ]
    }
  }
}
```

doc_count_error_upper_bound : 실제 계산에 포함되지 못한 값
sum_other_doc_count : 리턴되지 않은 나머지 값

Aggregation 사용시 주의점

순위	shard1	shard2	shard3
1	*Product A (25)*	*Product A (30)*	*Product A (45)*
2	*Product B (18)*	*Product B (25)*	*Product C (44)*
3	*Product C (6)*	*Product F (17)*	*Product Z (36)*
4	*Product D (3)*	*Product Z (16)*	*Product G (30)*
5	*Product E (2)*	*Product G (15)*	*Product E (29)*
6	*Product F (2)*	*Product H (14)*	*Product H (28)*
7	*Product G (2)*	*Product I (10)*	*Product Q (2)*
8	*Product H (2)*	*Product Q (6)*	*Product D (1)*
9	*Product I (1)*	*Product J (8)*	
10	*Product J (1)*	*Product C (4)*	



Product A (100)
Product Z (52)
Product C (50)
Product G (45)
Product B (43)

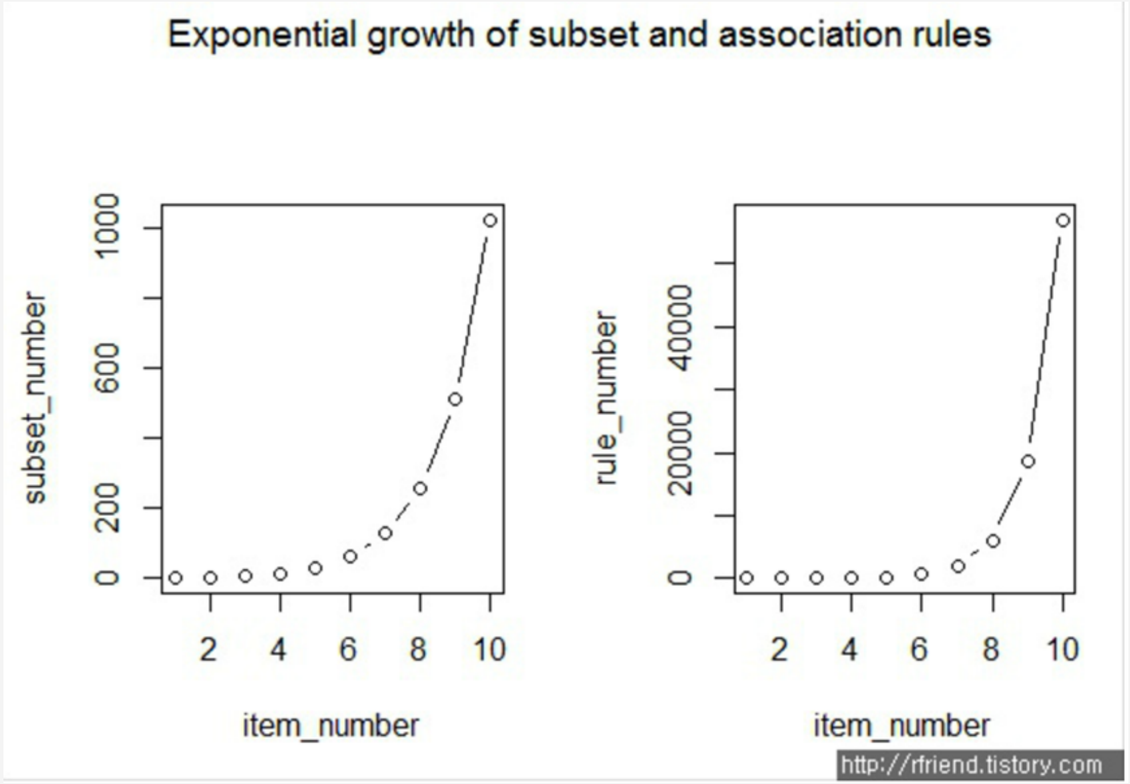
2. 기타 고급 검색기법

연관규칙 탐색

키워드 혹은 거래(Transaction)에서 나타나는 데이터에서 빈발 집합 셋을 찾아 내는 것

모든 항목의 집합은 $number\ of\ rules = 3^k - 2^{k+1} + 1$

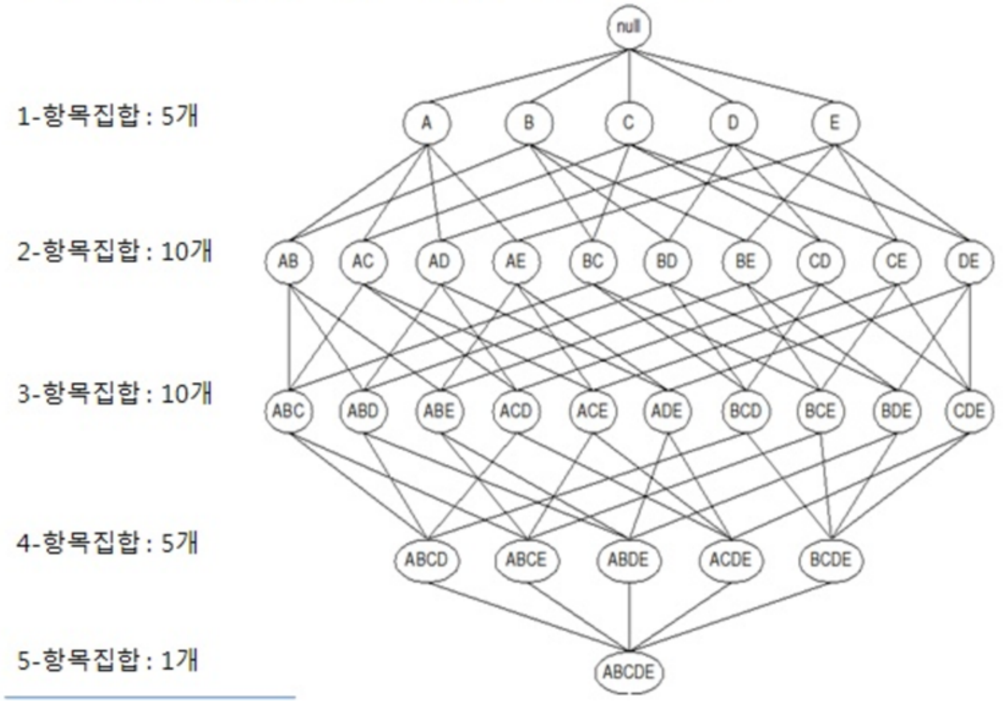
효율적인 연관규칙 탐색 알고리즘 필요



연관규칙 탐색

5개의 원소 항목 집합 $I = \{A, B, C, D, E\}$일 때,

모든 가능한 항목집합 = $2^k - 1 = 2^5 - 1 = 32 - 1 = 31$



1-항목집합 : 5개

2-항목집합 : 10개

3-항목집합 : 10개

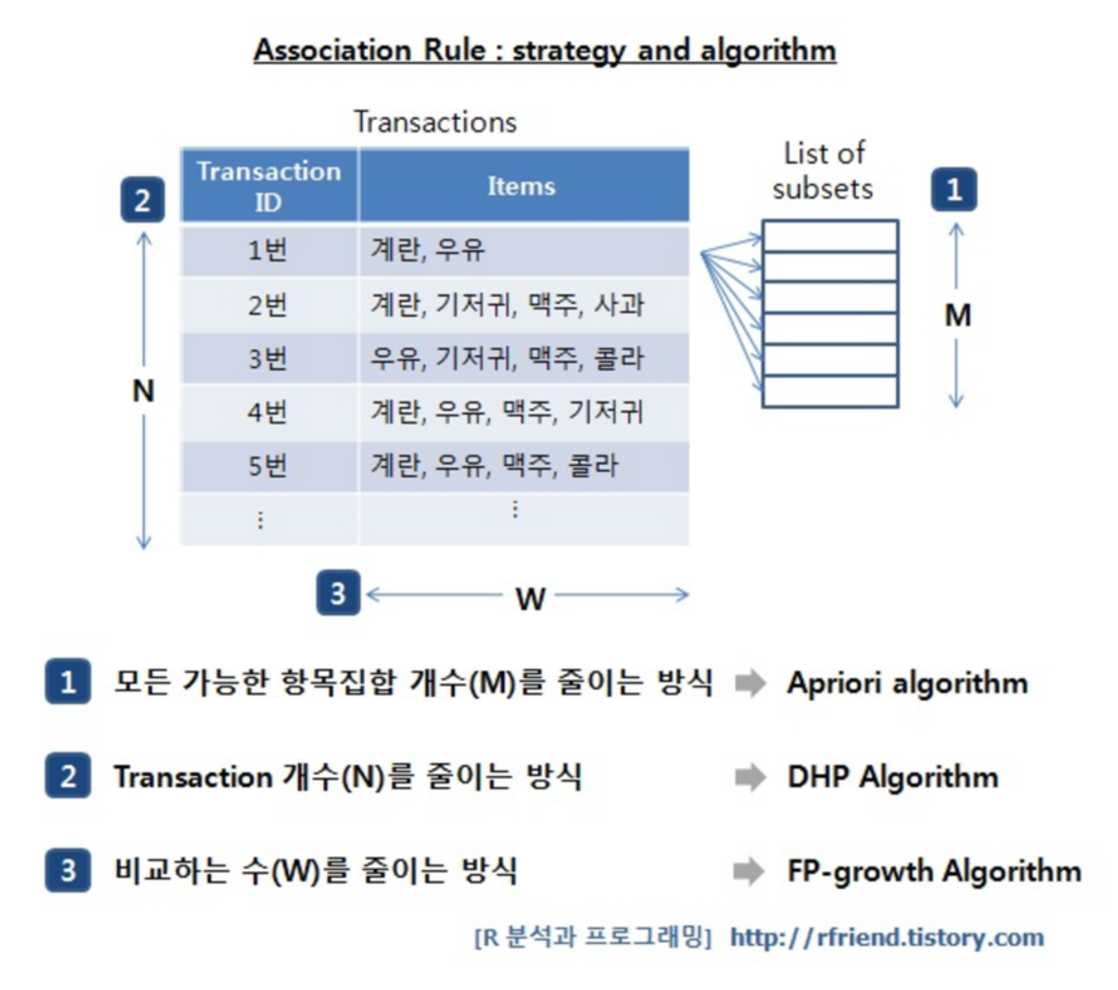
4-항목집합 : 5개

5-항목집합 : 1개

총 계 31개

[R 분석과 프로그래밍] http://rfriend.tistory.com

연관규칙 탐색 알고리즘의 종류



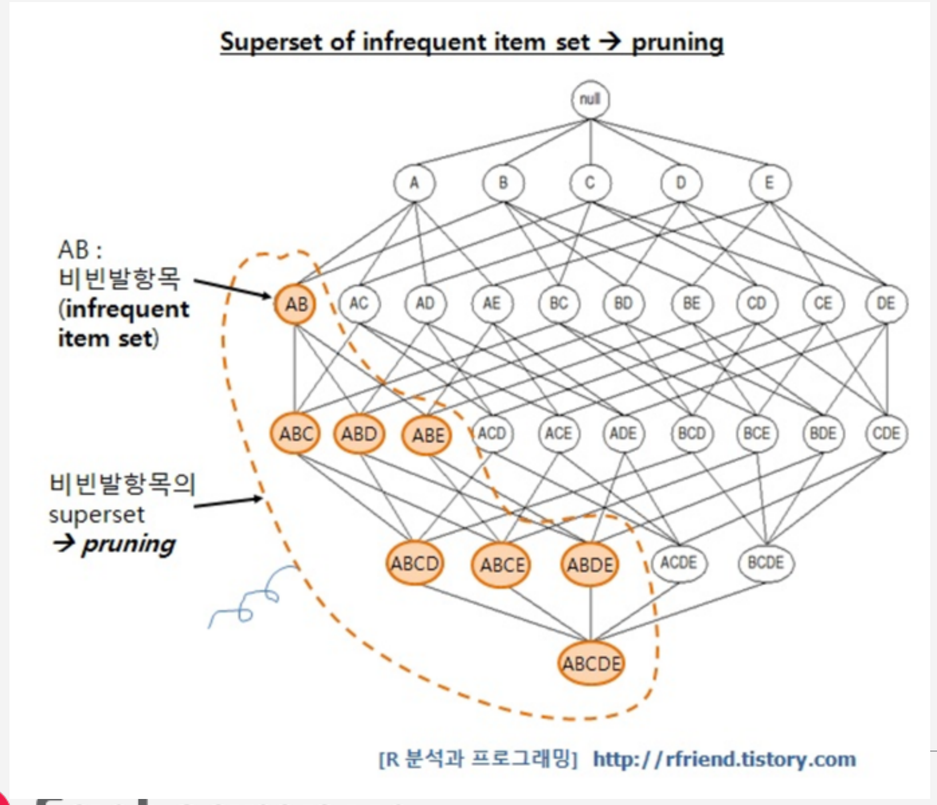
FP-Growth

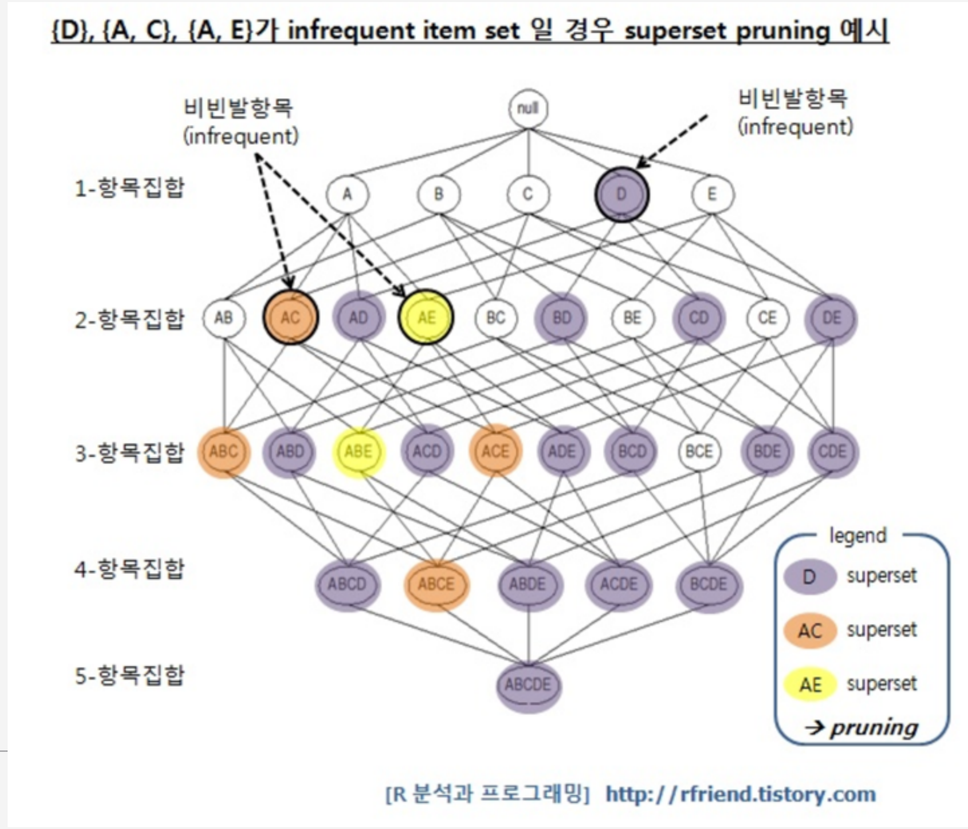
https://www.youtube.com/watch?v=vPcJEFFWN_k

Apriori 알고리즘

1) 한 항목집합이 빈발(frequent)하다면 이 항목집합의 모든 부분집합은 역시 빈발항목집합이다.
(frequent item sets -> next step)

2) 한 항목집합이 비빈발(infrequent)하다면 이 항목집합을 포함하는 모든 집합은 비빈발항목집합이다. (superset -> pruning)





검색엔진을 이용한 손쉬운 계산

필요 데이터

: 하나의 검색 세션을 구분 가능한 검색 쿼리 로그

하나의 검색 세션은 아이디 혹은 세션아이디와 시간을 기준으로 정의

세션	키워드
1	삼성
1	애플
1	엘지
2	삼성
2	애플
2	구글
3	김연아
3	피겨
3	동계올림픽
4	김태희

삼성과 연관있는 빈발 집합셋을 찾고자 한다면?

1. 삼성을 검색하여 나온 세션을 찾는다.
2. 세션값을 기준으로 검색하여 모든 키워드 셋을 찾는다.
3. Facet혹은 Aggregation을 이용하여 가장 많이 언급된 빈발셋을 순서대로 찾는다.
4. 최소 지지도를 설정하여 최소 지지도 미만의 데이터는 버린다.

검색엔진을 이용한 손쉬운 계산

Solr Example (Self JOIN 연산 기능 이용)

http://{검색엔진호스트}:{검색엔진포트}/solr/querylog/select?q={!join+from=signature%20to=signature}keyword_s:”삼성”&facet=true&facet.field=keyword_s&facet.mincount=1&facet.count=5&rows=0

ElasticSearch 의 경우

term aggregation 이후의 session값으로 재 aggregation 수행

엘라스틱서치에서 연관검색어 추출 기법

연관규칙추출실습.txt

오타 교정 (Spellchecker)

Levenshtein edit distance

어떤 문자열을 삽입,삭제,변경을 몇번이나 해서 바꿀수 있는가 계산
그 최소값을 구해 유사도 판단의 척도 사용

Jaro–Winkler distance

문자열 s_1과 s_2 사이의 Jaro distance d_j 의 정의

$$d_j = \begin{cases} 0 & \text{if } m = 0 \\ \frac{1}{3}\left(\frac{m}{|s_1|} + \frac{m}{|s_2|} + \frac{m-t}{m}\right) & \text{otherwise} \end{cases}$$

m 은 두 문자열에서 공통적으로 존재하는 문자의 수
t 는 두 문자열에 공통적으로 존재하나, 인덱스(순서)가 다른 문자의 수의 반

Jaro–Winkler distance d_w

$$d_w = d_j + (\ell p(1 - d_j))$$

p 는 고정된 scaling actor
공통 접두사가 있을경우 점수를 상향조정하기위해 사용하는 상수
p 는 0.25를 넘으면 안된다(넘으면 거리값이 1을 초과할수도 있음)
Winkler's work 에서 표준적인 값은 p = 0.1 이다
ℓ 문자열의 공통 접두사의 길이로 최대 4개

주어진 문자열 s_1 *MARTHA* 와 s_2 *MARHTA* 가 있을 때:

- $m = 6$
- $|s_1| = 6$
- $|s_2| = 6$
- s_1 T/H 와 s_2의 H/T가 공통적으로 존재하나 순서가 다른 문자이므로, $t = \frac{2}{2} = 1$

따라서 Jaro score 는:

$$d_j = \frac{1}{3}\left(\frac{6}{6} + \frac{6}{6} + \frac{6-1}{6}\right) = 0.944$$

Jaro-Winkler score 를 계산하기 위해 $p = 0.1$로 설정하고, 공통 접두사는 MAR 이므로:

- $\ell = 3$

따라서:

$$d_w = 0.944 + (3 * 0.1(1 - 0.944)) = 0.961$$

한글에서의 오타 교정 (Spellchecker)

한글의 음절을 자소단위로 분해 후 재결합하여 사용함 (정준 분해한 뒤 다시 정준 결합)

색인과 검색시
ICUNormalizer2Filter를 이용하여 방식으로 nfc 분해

출력 결과물 사용시
Normalizer.normalize({결과스트링), Normalizer.Form.NFC)

http://d2.naver.com/helloworld/76650
유니코드 정규화 방법 참조

Term Suggesters (스펠체커)

기본 문법

```
{
  "query" : {
    "match": {
      "message": "tring out Elasticsearch"
    }
  },
  "suggest" : {
    "my-suggestion" : {
      "text" : "trying out Elasticsearch",
      "term" : {
        "field" : "message"
      }
    }
  }
}
```

주요 옵션

text : 스펠체킹을 하기 위한 텍스트

field : 스펠체킹을 하기 위한 필드

analyzer : 스펠체킹 검색을 질의하기 위한 분석기 (기본값은 매핑값)

size : 출력되는 후보 갯수

sort : 정렬 기준 (기본 score, frequency)

string_distance : 문자열 비교를 위한 알고리즘 (damerau_levenshtein 기본)

“스펠체커만들기예제.txt” 실습

감사합니다.

다음 주제 :
ELK 로 구축하는 데이터 시각화 및 분석
기타 유용한 팁들