DEVIEW 2020

일본어 까막눈이 만드는 일본 주소검색
(검색 엔지니어는 무슨일을 할까요?)

조재용 NAVER

CONTENTS

1.검색 서비스 정의

1.1 검색 요구사항 특징

- 검색의 목표
  - 사용자 쿼리(검색어)를 입력으로 받아서, 사용자가 의도한 검색 결과를 최상위에 노출하는 것
- 주요 기능도 1개, 주요 입력도 1개
- 상세 요구사항 및 스펙이 눈에 보이지 않는다.
- 사용자가 인지하기 어렵다.
- 대부분의 검색 스펙은 어떤 타입의 쿼리까지 지원하는가에 있다.

1.2 검색 요구사항의 예

- 음차변환 (rkdskadur -> 강남역)
- 오타교정 (간남역 -> 강남역)
- 우편번호 검색 (13561 -> 그린팩토리)
- 빌딩명 검색 (퍼스트타워 -> 경기 성남시 분당구 분당로 55)
- 유의어 검색 (네이버 -> 그린팩토리)
- 부분쿼리 검색 (강 -> 강남역)
- 다국어 쿼리 검색 (gangnam -> 강남역)
- 카테고리 검색 (지하철 -> 강남역 2호선)

1.3 검색 데이터 분석

DEVIEW 2020

➢ 검색 엔지니어 업무에서 가장 많은 시간을 차지

- 양질의 데이터와 데이터의 양은 검색 품질과 직결된다.
- 저품질 데이터는 양질의 데이터로 정제해야 한다.
- 데이터의 품질이 개발 난이도를 결정한다.

➢ 분석해야 할 검색 데이터

- 사용자 쿼리
- 검색 대상이 되는 문서

1.4 일본 주소 쿼리 유형 분석의 예

DEVIEW 2020

- 영어/히라가나/카타가나/한자 문자가 혼재되어 검색된다.
  - 로마자 영어 발음으로 검색 (oishimise)
  - 히라가나/카타가나 발음으로 검색 (おいしいレストラン)
  - 한자로 검색 (おいしい店)
  - 영어 의미 자체로 검색 (delicious restaurant)
- 주소는 일반적으로 띄어쓰기를 하지 않는다.
- 우편번호만으로 검색하는 경우도 많다.
- 구어(쿠지타이)와 신어(신지타이)가 존재한다. (卷 => 巻, 寬 => 寛)
- 전각 문자와 반각 문자가 존재한다. (１ ０ => 10)
- 숫자를 한자로 표현하는 경우도 많다. (一 二 七 => 127)
- 스테가나(작은 문자)로 입력하는 경우도 많다. (ぁ -> あ, ァ => ア)
- 주소 입력 패턴(UX)은 한국과 유사하다.

1.5 일본 주소 체계 분석의 예

DEVIEW 2020

- ➢ 대부분의 도시에서 지번주소를 사용한다.
  - 교토: 옛날 주소 체계에 따른 도로명 주소를 사용
  - 삿포르(홋카이도): 전철을 기준으로 주소를 만들어 사용
- ➢ 일본 주소는 주로 전각 문자를 사용해서 표현한다.
- ➢ 주소의 일부는 히라가나/카타가나를 활용한 발음으로 표기한다.
- ➢ 규칙을 무시하는 예외가 매우 많고 다양하다.
  - 전통을 중요하게 생각해서 현주소체계와 맞지 않는 몇백 년 전의 주소도 그대로 사용
  - 캠퍼스와 같이 토지 단위가 큰 경우 번지가 없음
  - 하나의 토지에 두 개의 집이 이어진 경우, 번지도 연결되어 표현(1-2-3 -> 2-C-2-B)

2.검색 서비스 원리

2.1 색인과 검색어에 대해서

DEVIEW 2020

- 책에서 어떤 단어가 나온 페이지를 찾고 싶다면?

1. 원하는 단어가 나올 때까지 첫 페이지부터 읽는 방법
2. 책의 부록에 있는 색인에서 원하는 단어의 페이지를 알아내는 방법

- 검색의 핵심은 검색어(쿼리)에 대한 색인(인덱스)을 만드는 것이다.

<색인>

네이버	1
지도	3, 20
검색	4, 5, 14

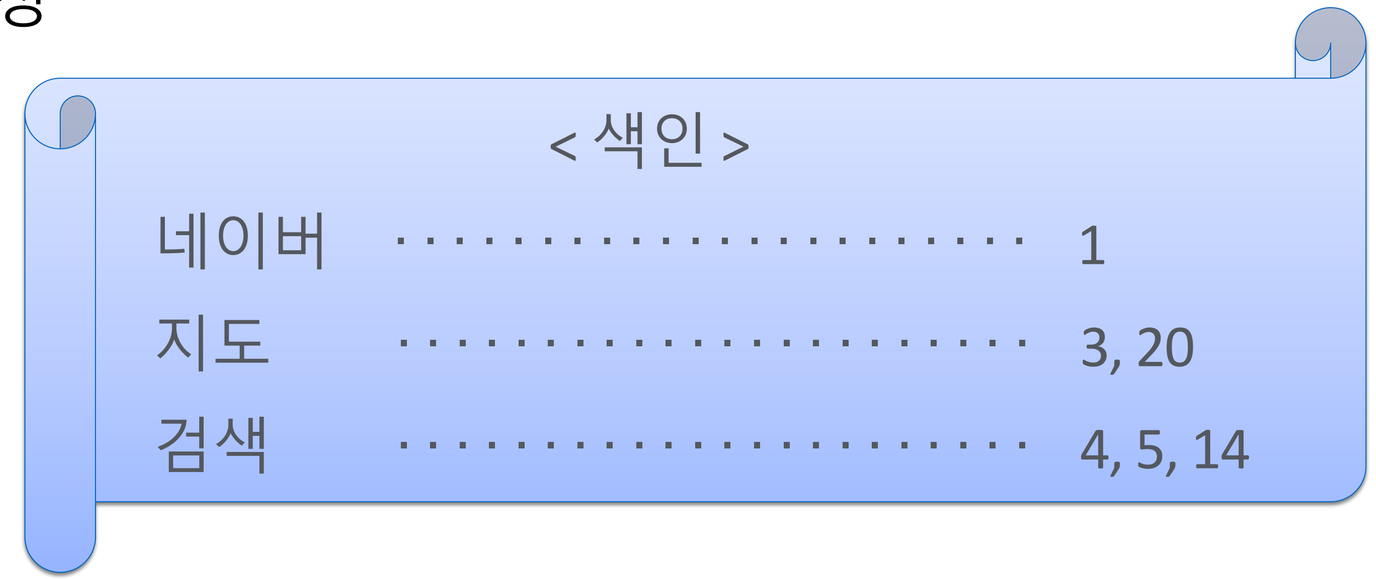
2.2 인덱스 구축 단계

1. 인덱스텀 추출(Term Extraction)

- 문서에서 가장 중요하고 많이 나오는 텀을 추출하는 작업

2. Inverted File 생성

- 인덱스텀과 인덱스텀이 존재하는 문서의 DocId 목록을 매핑
- 일반적인 DB와 반대의 구조로 인덱싱



2.2.1 인덱스텀 추출(term extraction)

- 적절한 인덱스텀이 없으면, 예상되는 쿼리에 대해서 검색이 되지 않는다.
- 적절한 인덱스텀이란?

- 문서의 언어와 문자 환경에 적합
- 예상되는 사용자 쿼리의 특징에 적합
- 검색 결과를 구성하기에 적합

- 텀을 추출하는 방법

- 공백 단위 구분 추출
- n-gram 추출
- 형태소 분석기 추출
- NLU를 사용한 추출

2.2.2 인덱스텀 추출의 예 (1)

DEVIEW 2020

➢ 공백 단위 구분 추출

- [서울에서, 부산가는, 길]

➢ n-gram 추출

- n이 2일때: [서울, 울에, 에서, 서_, _*부*, *부산*, *산가*, *가는*, *는*_, _길]
- n이 3일때: [서울에, 울에서, 에서_, 서_부, _*부산*, *부산가*, *산가는*, *가는*_, 는_길]

➢ Edge n-gram 추출

- min: 2, max: 4, sep: “ ”일때: [서울, 서울에, 서울에서, 부산, 부산가, 부산가는]

2.2.3 인덱스텀 추출의 예 (2)

DEVIEW 2020

➢ 형태소 분석기 추출

- 형태소 분석기의 종류나 설정에 따라서 결과가 다름
- 품사에 따른 분리 또는 사전 기반에 따른 분리가 가능 (텀, 품사, 사전)
- [(서울 , 명사 , 지역), (에서 , 조사), (부산 , 명사 , 지역), (가는 , 동사), (길 , 명사)]
- 명사만: [서울, 부산, 길]
- 지역만: [서울, 부산]

➢ 적절한 인덱스텀 추출의 중요성

- 공백 분리: “서울에서 부산가는”는 되는데, “서울 부산”은 안된다.
- 형태소 분석기: “서울에서 부산까지”와 “서울 부산”이 모두 된다.

2.2.4 Inverted File 생성

DocId	Document	Index Terms
1	서울 중랑구	서울, 중랑
2	경기 성남시	경기, 성남
3	서울 송파구 방이동	서울, 송파, 방이
4	경기 수원시 정자2동	경기, 수원, 정자, 2
5	경기 성남시 정자동	경기, 성남, 정자

➢ Inverted File

- 인덱스텀과 인덱스텀이 존재하는 문서의 DocId 목록 매핑의 집합

➢ Lexicon

- 인덱스텀을 빨리 찾을 수 있도록 텀들의 목록을 모아서 관리하는 사전
- 빠른 탐색에 최적화되어야 함

➢ Posting

- 인덱스텀이 위치하는 문서 식별자(DocID) 1개
- 메모리와 보조기억장치를 최대한 효율적으로 사용

Inverted File

Lexicon			Posting List
Term	Postings count	Offset	
서울	2	→	1, 3
경기	3	→	2, 4, 5
중랑	1	→	1
성남	2	→	2, 5
송파	1	→	3
방이	1	→	3
정자	2	→	4, 5
수원	1	→	4
2	1	→	4

2.3 쿼리 처리(query processing)

➢ 쿼리의 종류

- Boolean 쿼리, 근접(proximity) 쿼리, 자연어 쿼리 등이 있다.
- 대부분의 사용자들은 알아서 처리해주는 자연어 쿼리를 원한다.

➢ 쿼리 처리 단계

1. 쿼리텀 추출(Term Extraction)
2. 추출된 쿼리텀으로 Inverted File에서 포스팅 찾기
3. 쿼리 연산 수행 및 검색 결과 목록 생성
4. 문서 점수화 및 정렬

2.3.1 쿼리텀 추출

Inverted File

Lexicon			Posting List
Term	count	Offset	
서울	2	→	1, 3
경기	3	→	2, 4, 5
중랑	1	→	1
성남	2	→	2, 5
송파	1	→	3
방이	1	→	3
정자	2	→	4, 5
수원	1	→	4
2	1	→	4

DocId	Address
1	서울 중랑구
2	경기 성남시
3	서울 송파구 방이동
4	경기 수원시 정자2동
5	경기 성남시 정자동

쿼리텀 추출

Query	Query Term
경기도 정자동	경기
	정자

➢ 인덱스텀 추출과 거의 유사한 기준으로 추출해야 한다.

- 다른 기준으로 텀을 추출하면 문서와 동일한 쿼리로 검색해도 검색이 안될 수 있다.

➢ “경기 성남시 정자동”의 예

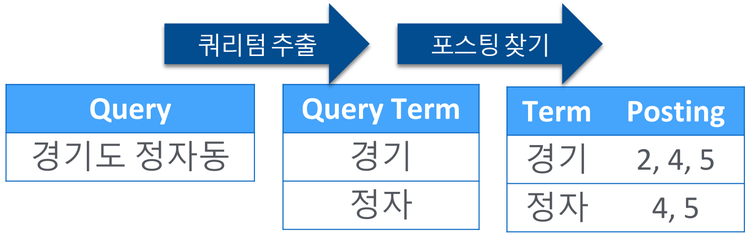
- 인덱스텀은 형태소 분석기로 분리해서 “경기”, “성남”, “정자”로 생성
- 쿼리텀은 공백으로 분리해서 “경기”, “성남시”, “정자동”로 생성
- 이 경우, “경기 성남시 정자동”로 검색해도 해당 문서가 검색되지 않는다.

2.3.2 포스팅 찾기

Inverted File

Lexicon			Posting List
Term	count	Offset	
서울	2	→	1, 3
경기	3	→	2, 4, 5
중랑	1	→	1
성남	2	→	2, 5
송파	1	→	3
방이	1	→	3
정자	2	→	4, 5
수원	1	→	4
2	1	→	4

DocId	Address
1	서울 중랑구
2	경기 성남시
3	서울 송파구 방이동
4	경기 수원시 정자2동
5	경기 성남시 정자동



- Inverted File에서 각 쿼리텀에 대한 포스팅을 찾는다.
- Inverted File은 메모리에 있고, 탐색 속도가 매우 빠르다.

2.3.3 쿼리 연산

Inverted File

Lexicon			Posting List
Term	count	Offset	
서울	2	→	1, 3
경기	3	→	2, 4, 5
중랑	1	→	1
성남	2	→	2, 5
송파	1	→	3
방이	1	→	3
정자	2	→	4, 5
수원	1	→	4
2	1	→	4

DocId	Address
1	서울 중랑구
2	경기 성남시
3	서울 송파구 방이동
4	경기 수원시 정자2동
5	경기 성남시 정자동

쿼리텀 추출 → 포스팅 찾기 → 쿼리 연산(AND)

Query
경기도 정자동

Query Term
경기
정자

Term	Posting
경기	2, 4, 5
정자	4, 5

DocId
4
5

➢ 쿼리텀에 매칭되는 포스팅로 쿼리 연산을 수행한다.

- Boolean 쿼리, 근접 쿼리 등 각 쿼리의 성격에 따라서 다른 연산 과정을 수행한다.
- 일반적으로 자연어 쿼리는 모든 쿼리텀을 AND 검색한 문서를 찾는다.

➢ 최적화된 쿼리 처리

- 한개의 쿼리텀에 대한 포스팅 리스트의 크기는 최악의 경우 전체 문서의 집합의 크기와 같다. 이 경우, 쿼리 처리 속도가 급격하게 느려진다.
- 메모리 사용과 수행 속도를 최적화한 쿼리 처리가 중요하다.

2.3.4 점수화 및 정렬

Inverted File

Lexicon			Posting List
Term	count	Offset	
서울	2	→	1, 3
경기	3	→	2, 4, 5
중랑	1	→	1
성남	2	→	2, 5
송파	1	→	3
방이	1	→	3
정자	2	→	4, 5
수원	1	→	4
2	1	→	4

DocId	Address
1	서울 중랑구
2	경기 성남시
3	서울 송파구 방이동
4	경기 수원시 정자2동
5	경기 성남시 정자동

쿼리텀 추출 → 포스팅 찾기 → 쿼리 연산(AND) → 점수계산 → 정렬 → 검색 결과 생성

Query
경기도 정자동

Query Term
경기
정자

Term	Posting
경기	2, 4, 5
정자	4, 5

DocId
4
5

DocId	Score
4	1.25
5	1.37

DocId	Score
5	1.37
4	1.25

Search Result
경기 성남시 정자동
경기 수원시 정자2동

➢ Relevancy (정확도)

- 입력된 쿼리텀들과 문서가 얼마나 일치하는지를 나타내는 점수
- Relevancy를 구하는 알고리즘은 검색의 특성이나 도메인에 따라서 다양하다.
- 기본적인 Relevancy 알고리즘으로는 문서 유사도를 구하는 TF-IDF와 BM25가 있다.
- 이외에도 벡터 모델, 확률 모델 등 이 있다.

➢ 랭킹 모델링

- Relevancy 점수를 계산하는 방법을 정하는 과정이 랭킹 모델링이다.
- 사용자의 의도에 가장 근접한 점수를 모델에 반영하기 위해서 가지고 있는 데이터를 최대한 활용한다.

➢ 지도 장소검색 랭킹 모델의 예

- 베이지언 확률을 기반으로 한 모델을 사용한다.
- 텀-문서 유사도, 장소 인기도, 사용자의 위치와 대상 장소의 거리

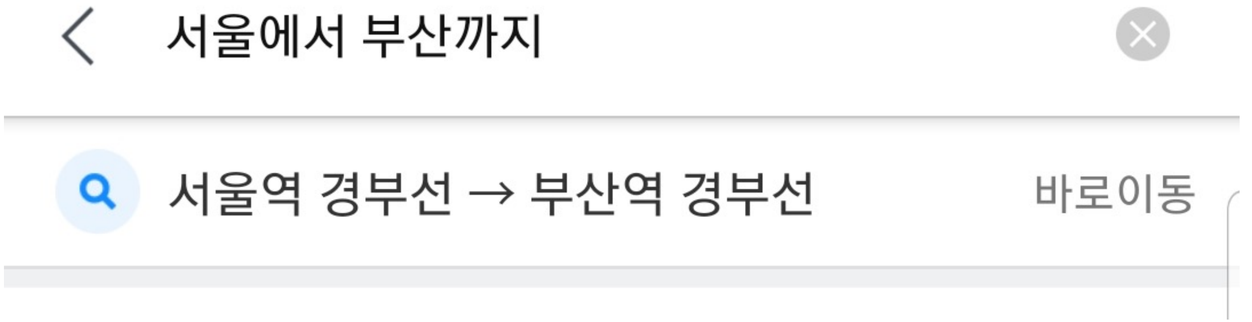
2.3.5 NLU를 사용한 쿼리 분석(참고)

➢ NLU(Natural Language Understanding)

- 문장이 실제로 의미하는 바를 추론할 수 있다.
- 자연어 분석, 음성 검색에서 많이 사용된다.

➢ 지도 검색에서 NLU를 활용한 예

- “서울에서 부산까지” 쿼리를 NLU로 분석하면 다음과 같은 결과가 나온다.
- from: 서울, to: 부산, by: 대중교통
- 네이버 지도에서는 NLU가 인지하는 쿼리가 들어왔을때, 바로 대중교통 검색으로 이동할 수 있는 자동완성을 제공한다.



2.3.6 TF-IDF, BM25 (참고)

$$\text{score}(D,Q) = \sum_{i=1}^{n} \text{IDF}(q_i) \cdot \frac{f(q_i, D) \cdot (k_1 + 1)}{f(q_i, D) + k_1 \cdot \left(1 - b + b \cdot \frac{|D|}{\text{avgdl}}\right)}$$

BM25 계산식 - 출처: https://en.wikipedia.org/wiki/Okapi_BM25

➢ TF (Term Frequency)

- 문서에 인덱스텀이 포함된 개수다.
- 문서에 매칭된 텀의 개수가 많을수록 사용자가 원하는 문서일 가능성이 높다.
- TF가 높을수록 높은 점수를 준다.

➢ IDF (Inverse Document Frequency)

- 인덱스텀이 포함된 문서의 개수다. (포스팅 리스트의 크기)
- 매칭된 텀이 희소할수록, 특정 문서를 잘 대표할 것이다.
- IDF가 낮을수록 높은 점수를 준다.

➢ Field Length

- 문서의 길이다.
- 문서의 길이가 짧은 필드에 매칭된 텀일수록 사용자의 의도에 더 부합하는 문서일 것이다.
- 예를 들어 쿼리텀이 문서의 내용 필드보다는 길이가 짧은 제목 필드에 매칭될수록 높은 점수를 준다.
- Filed Length가 낮을수록 높은 점수를 준다.

DEVIEW 2020

3. 일본 주소검색 서비스 개발

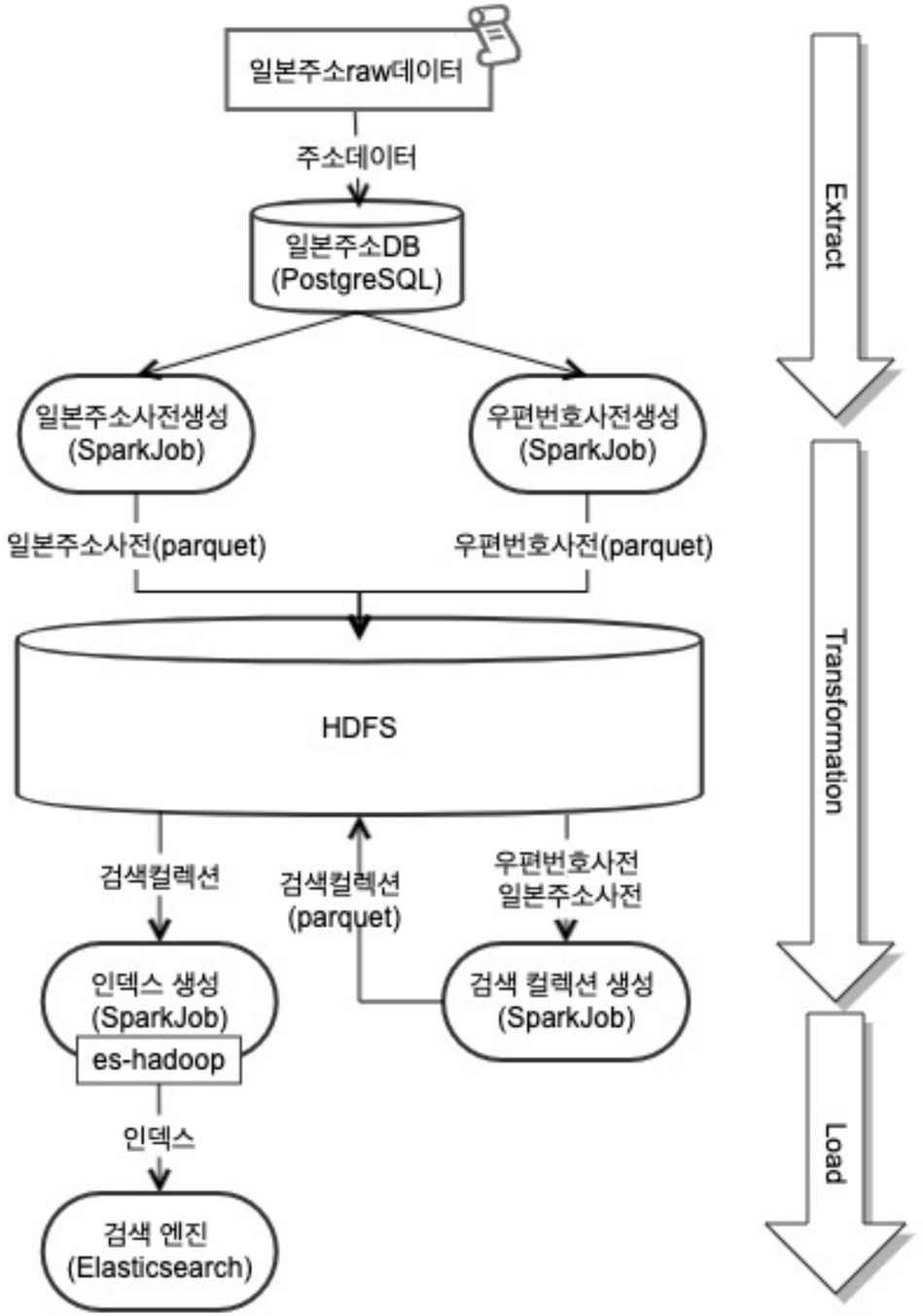
3.1 기술 스택

DEVIEW 2020

- ➢ 검색 엔진
- **Lucene**
- ➢ 검색 프레임워크
- **Elasticsearch**, Solr
- ➢ 데이터베이스
- **PostgreSQL**, MongoDB 등
- ➢ 대용량 처리
- **Hadoop + Spark + Scala**, pandas 등
- ➢ 형태소 분석기
- 한국어: **linguist2**, nori, 코모란, 꼬꼬마, 은전한닢 등
- 일본어: **linguist2**, kuromoji, Mecab 등
- ➢ API 서비스
- **spring boot 2 + kotlin**
- ➢ 검색 인프라
- **ncc(docker + kubernetes)**

3.2 데이터 파이프라인 구축

- ➢ 원본 데이터 준비
- ➢ 데이터 추출/변환/적재(ETL)
  - 추출: 원본 데이터에서 필요한 데이터만 뽑아서 저장
  - 변환: 추출된 데이터를 인덱스를 만들기에 적절한 형태로 가공
  - 적재: 생성된 인덱스를 검색엔진에 로딩
- ➢ 모든 작업에 사람의 개입을 최소화
- ➢ 데이터의 최신성 목표에 따른 구축
  - 일주일에 한 번씩 메일로 데이터를 전달받는다면, 새로운 주소는 최소 일주일이 지나야 반영된다.
  - 2시간마다 시스템적으로 배치 처리된다면, 새로운 주소는 2시간이 지나면 반영된다.
  - 실시간 반영을 목표로 한다면, 증분 Job을 구축해야 한다.



3.3 일본 주소검색 - 정규화

DEVIEW 2020

➢ 일본어는 표현 방식의 다양성과 많은 예외로 정규화가 불가피

- 예를들어 한국 주소의 번지에 해당하는 초메반고만 해도 다음과 같이 다양한 형태의 쿼리가 있을 수 있다.

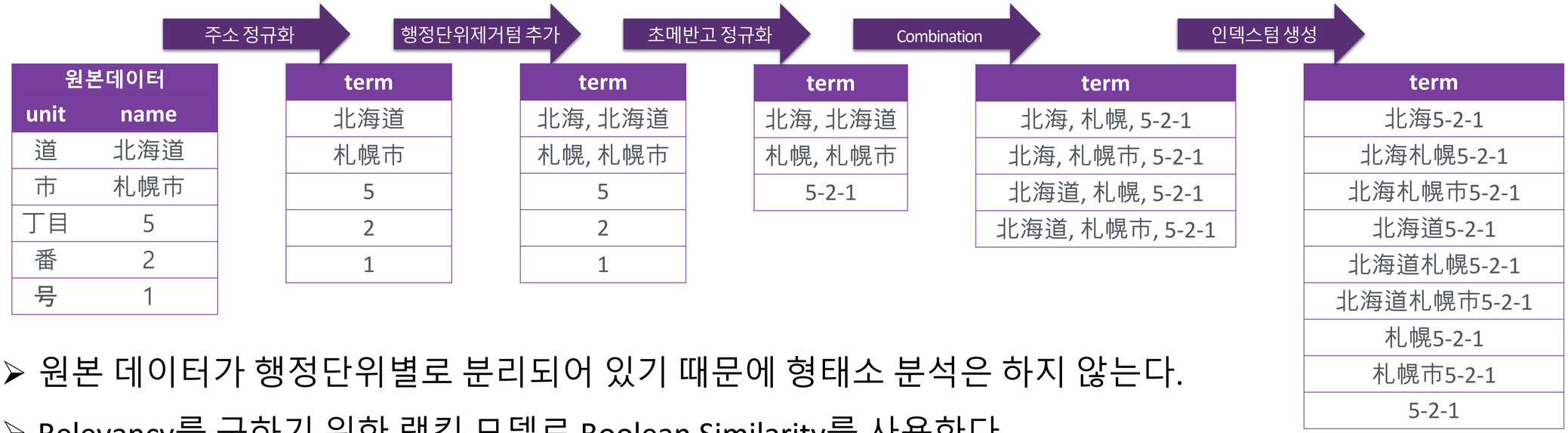
“1丁目3番地5号”, “1丁目3番地の5”, “1丁目3番5”, " 一丁目三番地5”, “一丁目三番地の5”, “一丁目三番5”, “1-3-5”, “1丁目3-5”

- 쿼리텀과 인덱스텀을 모두 “1-3-5”로 정규화 한다면 인덱스의 크기를 줄일 수 있다.
- 정규화로 인해서 쿼리텀과 인덱스텀이 의미상 충돌나지 않도록 해야 한다.
- 인덱스텀과 쿼리텀은 반드시 같은 기준으로 정규화 해야 한다.

➢ 일본 주소 / 사용자 쿼리 정규화(normalize)

- 대문자 -> 소문자 (Cafe => cafe)
- 쿠지타이 -> 신지타이 (卷 => 巻, 寬 => 寛)
- 스테가나 -> 가나 (ぁ -> あ, ァ => ア)
- 전각 -> 반각 (３　丁目　４　１０ => 3 丁目 4 10)
- 히라가나 -> 카타가나 (でかい => デカい)
- 한문숫자 -> 아라비아숫자 (一 二 七 => 127)
- 초메반고 정규화 (3 丁目　4 - １０ => 3-4-10)

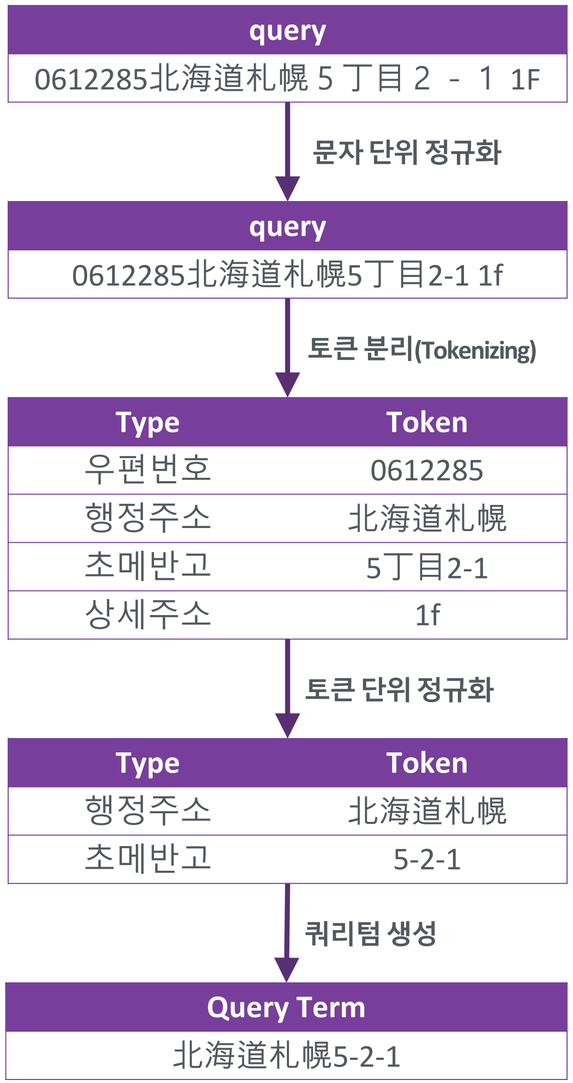
3.3 일본 주소검색 – 인덱스텀 생성



- 원본 데이터가 행정단위별로 분리되어 있기 때문에 형태소 분석은 하지 않는다.
- Relevancy를 구하기 위한 랭킹 모델로 Boolean Similarity를 사용한다.
- 예상 가능한 쿼리텀의 모든 인덱스텀을 미리 생성해 놓는다.
  - 일본어(한자), 일본어발음, 로마자발음, 중국어 텀 추가
  - 행정단위를 제거한 텀 추가
  - 주소 접두사와 점미사를 제거한 텀 추가
- 생성되는 인덱스 크기가 매우 크다.
- 쿼리 처리 시간이 빠르다.

3.4 일본 주소검색 – API 개발

- ➢ 쿼리 정규화
- ➢ 쿼리 토큰 분리
- ➢ 랭킹 모델
  - Boolean Similarity (Rule Based)
  - 거리 점수
  - String Similarity (Algorithm Based)
- ➢ 쿼리 처리 속도 vs 검색 커버리지
  - Boolean Similarity: 빠르지만 커버리지가 낮음
  - String Similarity: 느리지만 커버리지가 높음
  - Boolean Similarity 검색결과가 없으면, String Similarity 사용
  - (TODO) Log Base Boolean Similarity: 빠르고 커버리지가 높음



Index Term
北海5-2-1
北海札幌5-2-1
北海札幌市5-2-1
北海道5-2-1
北海道札幌5-2-1
北海道札幌市5-2-1
札幌5-2-1
札幌市5-2-1
5-2-1

4.검색 품질 측정 및 개선

4.1 테스트 데이터와 정답셋

- 데이터가 많을수록 검색 품질이 향상된다.
- 양질의 테스트 데이터
  - 경험에 기반한 쿼리 타입 추출
  - 검색 로그
  - 서비스되고 있는 주소 검색과 비교 (구글, 야후)
  - 클릭 로그
- 확보된 데이터는 주기적으로 생성되도록 자동화하는 것이 좋다.
  - 지원되지 않는 쿼리 타입을 중심으로 개선 점을 도출하는데 사용
  - A/B 테스트, 회귀 테스트 등 검색 품질 측정에 활용
  - 학습 데이터로 활용해서 검색 품질 개선을 자동화

4.2 품질 측정 기준 지표(KPI) 정의

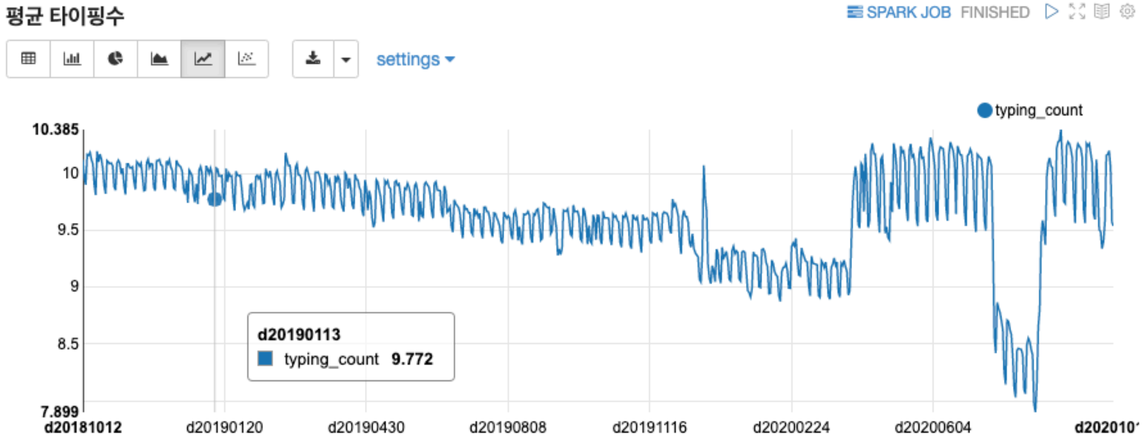
DEVIEW 2020

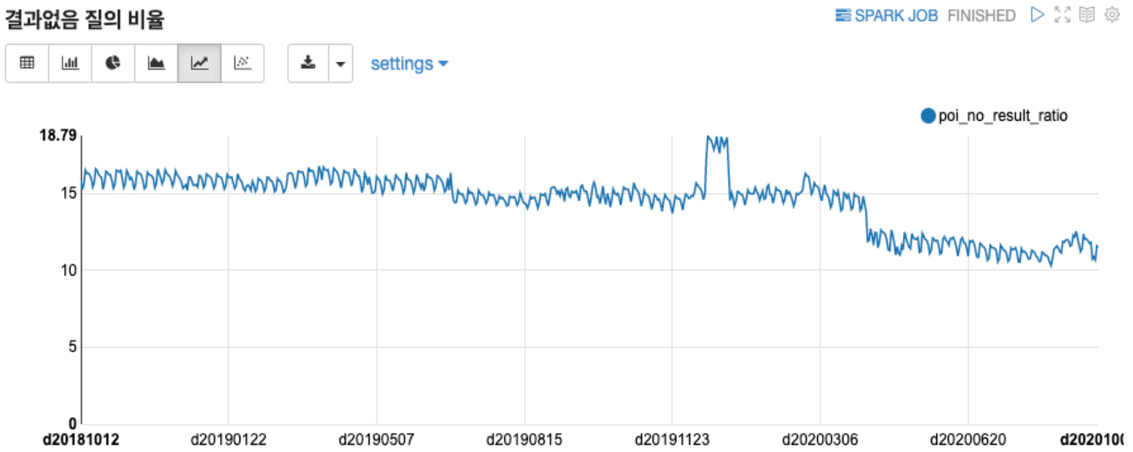
➢ 검색 품질을 평가하기 위한 기준 지표를 만든다 .

- 검색 품질을 개선하려면, 평가할 수 있어야 한다.
- 매일/매주/매달 KPI 지표를 뽑을 수 있는 파이프라인을 구축한다.
- KPI를 통해서 검색 품질 개선점을 도출한다.

➢ 품질 측정 기준 지표의 예

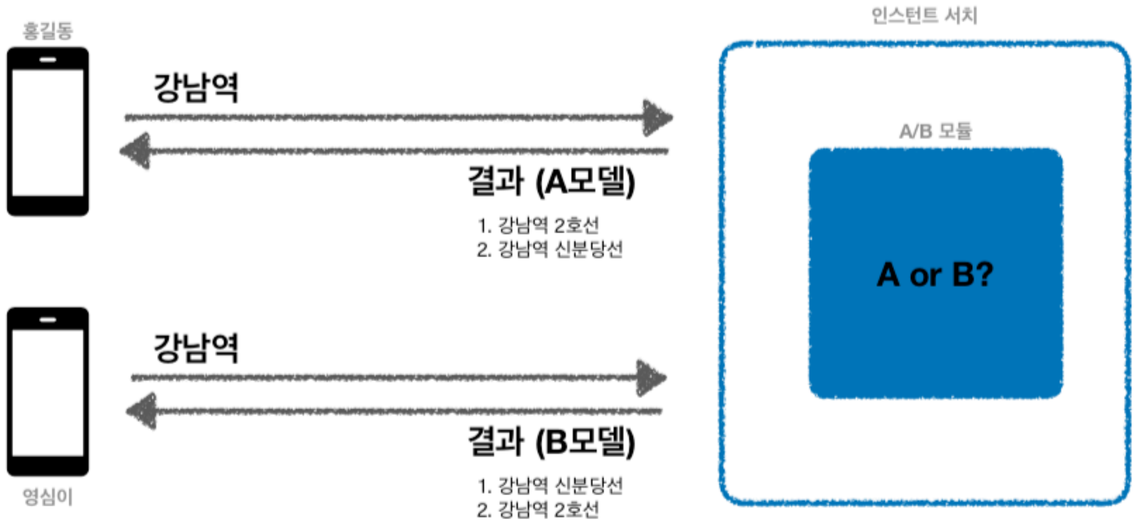
- 사용자 클릭 비율
- 검색어 삭제 및 재검색 비율
- 사용자가 상위랭크를 클릭한 비율 점수화
- 검색결과 없음 비율
- 클릭이 발생한 시점에 검색어의 길이





4.3 A/B 테스트 구축

- AS-IS 서비스를 A, TO-BE 서비스를 B로 한다.
- 출시 전 A/B 테스트
  - 준비된 정답셋으로 A와 B를 직접 실행해서 비교한다.
  - A보다 B의 점수가 낮으면, 출시하지 않는다.
- 출시 후 A/B 테스트
  - 서비스의 일부만 부분배포할 수 있는 매커니즘이 필요하다.
  - 실서비스에 B를 부분배포한 후, 품질 측정 기준 지표로 비교한다.



4.4 일본 주소검색 품질 개선(...ing)

DEVIEW 2020

- 일본내 서비스에 적용 예정
- 출시 전 개선

- 구글, 야후 등의 서비스와 품질을 비교하면서, 부족한 부분을 보완
- 일본내 신규 주소 및 업데이트 반영 주기 개선

- 출시 후 개선

- 사용자 쿼리/클릭 로그를 기반으로 검색 품질이 자동 개선되도록 구축
- 랭킹 모델에 인기도 반영

4.5 검색 엔지니어가 하는 일...

DEVIEW 2020

- 벡엔드 엔지니어
- 데이터 엔지니어
- 데이터 분석가
- 검색/랭킹 모델링
- 검색 엔진 개발

Q & A

Thank You