

Neo4j(APOC, GDS, Cypher)를 이용한 추천 시스템

2019038277 김피비]

May 11, 2023

Contents

1 리모트 서버 db 구현 형태	3
1.1 Pagerank algorithm	4
1.2 Force directed graph	4
1.3 Louvain 알고리즘(community detection)	5
2 Neo4j 테스크탑 db 생성 과정	6
2.1 페이지 랭크 알고리즘	7
2.1.1 페이지 랭크 생성과정(company)	8
2.1.2 페이지 랭크 생성과정(review)	8
2.2 community 찾기 (louvain)	8
3 추가사항 & 향후 보완 가능성	9

모든 설계는 Neo4j에서 cypher 쿼리로 작성되었음

1 리모트 서버 db 구현 형태

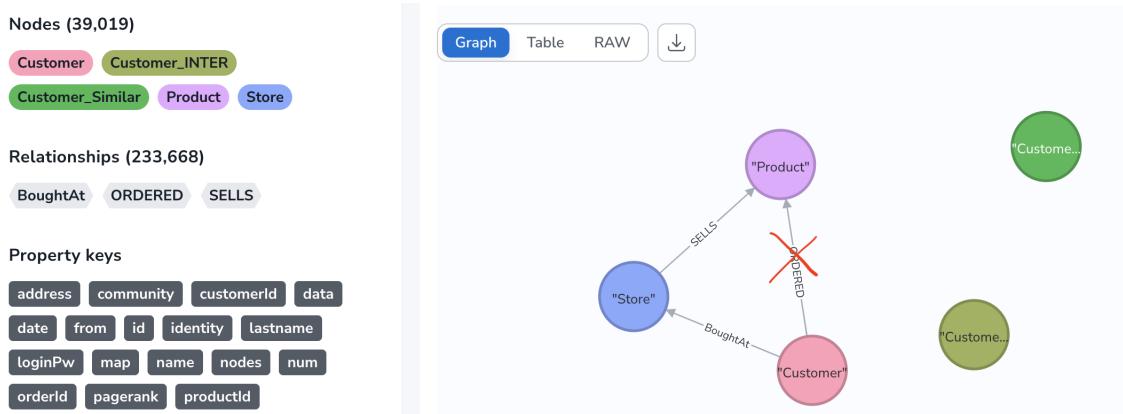


Figure 1: aura db에서의 company 메타 그래프

선택한 데이터베이스 인스턴스의 최대 노드수와 관계 수에 제한이 있기 때문에 처음 로컬에서 생성한 db 인스턴스와 형태가 달라졌기 때문에 서버 db 쿼리문에 차이가 있다. 먼저, 초기 company csv 파일로부터 Customer, Product, Store 노드와 SELLS 관계를 단순 열 삽입으로 생성했고 BoughtAt and 'Customer_INTER'는 graph ds algorithm과 apoc library를 이용해 생성한 관계이다. 리모트에서는 노드 간 커플링을 최소화하고 추천 모델 학습의 용이성을 위해 Customer, Product, Store, Customer_Inter 노드 SELLS, BoughtAt 관계를 사용했다.

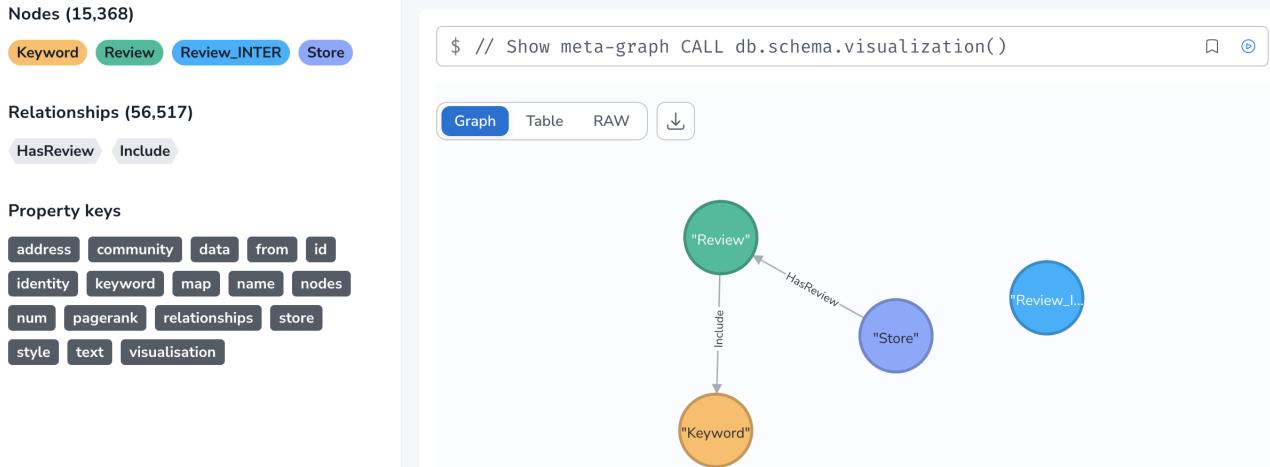


Figure 2: aura db에서의 review 메타 그래프

마찬가지로 'Review_INTER'는 그래프 알고리즘에 학습에 이용하기 위해 생성한 관계이다. 여기서는 Review 노드를 중심으로 그래프 알고리즘을 적용했고 모든 노드 관계를 사용한다.

1.1 Pagerank algorithm

```

1 MATCH (c:Customer)
2 WITH c ORDER BY c.pagerank DESC LIMIT 10
3 with COLLECT(c.identity) AS c_ids
4 UNWIND c_ids AS c_id with c_id, c_ids
5 MATCH (i:Customer_INTER) WHERE i.from = c_id
6 UNWIND keys(apoc.convert.fromJsonMap(i.map)) AS c2_id
7 MATCH (c2:Customer) WHERE c2.identity = toInteger(c2_id) AND
    c2.identity <> c_id and c2.identity in c_ids WITH c_id,
    c2_id,apoc.convert.fromJsonMap(i.map)[c2_id] AS weight
8 RETURN c_id, toInteger(c2_id) as c2_id, weight

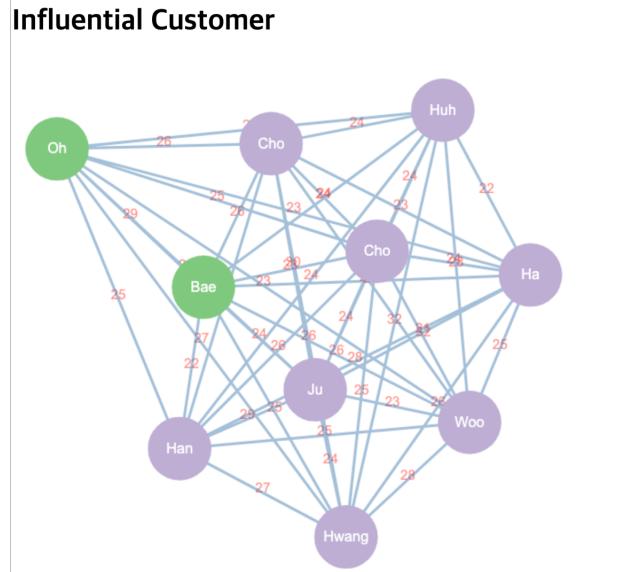
```

Figure 3: 상위 10 pagerank의 Customer 노드

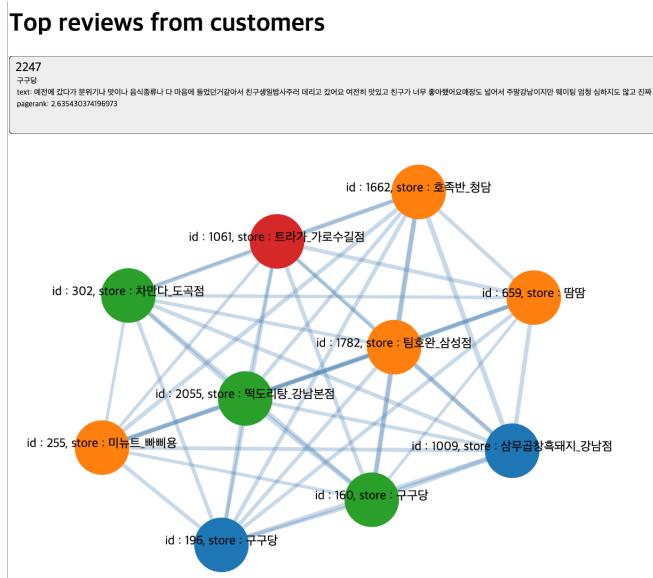
top 10 인플루엔서 customer nodes 와 Inter 레이션 페이지 랭크 알고리즘을 통해 다른 노드와의 관계적 영향이 큰 노드를 찾을 수 있었고 이를 인플루엔서 노드로 지정했다. 페이지 랭크를 학습시키는데 사용한 Inter relationship (서로 다른 customer 간에 공통으로 방문한 가게의 수를 프로퍼티로 가짐) 의 정보를 Customer_INTER 노드에 identity, 시작 노드, 끝 노드의 identity의 키와 둘 사이의 weight value로 이루어진 map property를 넣어 대신하기로 했다. 따라서 상위 페이지 랭크 수치를 10개의 customer 노드와 각각의 관계를 구하기 위해 위의 쿼리를 실행했다. 위의 정보를 불러오기 위한 쿼리문의 구조는 review 노드에서도 동일하게 적용된다. top 10 인플루엔서 Review nodes 도 같은 구조로 설계되었고 자세한 과정은 Desktop db 생성 섹션참조.

1.2 Force directed graph

최상위 pagerank인 customer 노드와 weight로 표시된 노드간 Inter 관계를 d3.js를 사용하여 force directed graph로 나타냈다.



그래프의 노드는 상위 페이지랭크의 10명 Customer로 이루어져 있고, 에지는 서로 다른 Customer 노드 사이에 공통으로 구매 이력이 있는 Store의 수이다. 페이지랭크 알고리즘의 기반이 되는 그래프 관계가 서로 다른 두 고객이 공통으로 방문한 가게 수를 나타내는 Inter 관계이기 때문에 대체적으로 서로 긴밀히 연결되어있음을 확인할 수 있다.



Review 노드를 그래프의 노드로 각 정제된 리뷰 텍스트의 공통된 키워드 개수를 가중치로 설정한 관계를
에지로 하여 그래프로 나타내면 다음과 같다. 리뷰 데이터는 가게 이름, 그리고 텍스트로 이루어져 있기 때문에
노드를 클릭하면 그 정보를 볼 수 있도록 만들었다.

1.3 Louvain 알고리즘(community detection)

위 force directed graph에서 같은 색깔로 표시된 노드는 같은 커뮤니티에 속한다. 알고리즘에 사용한 인메모리
그래프는 페이지 랭크에서 사용하기 위해 만들었던 것과 동일하다.

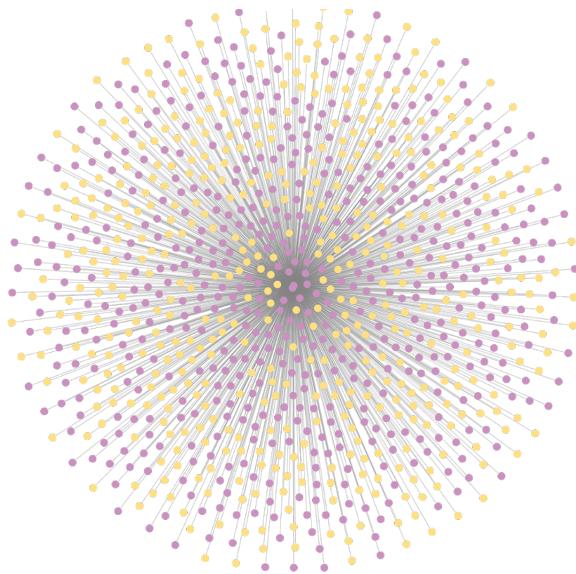


Figure 4: company 커뮤니티 시각화

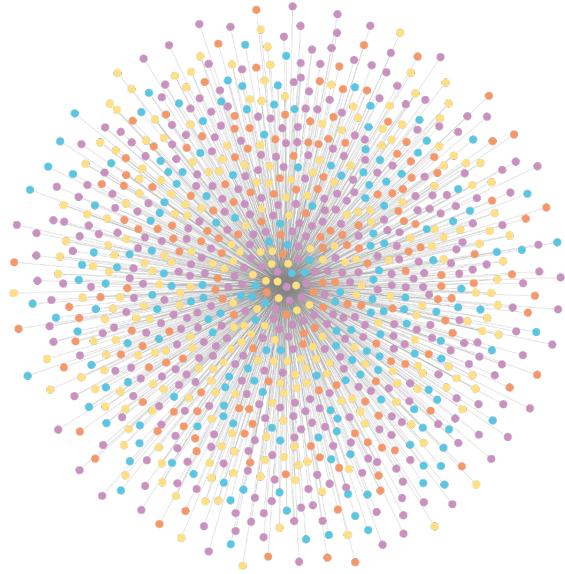
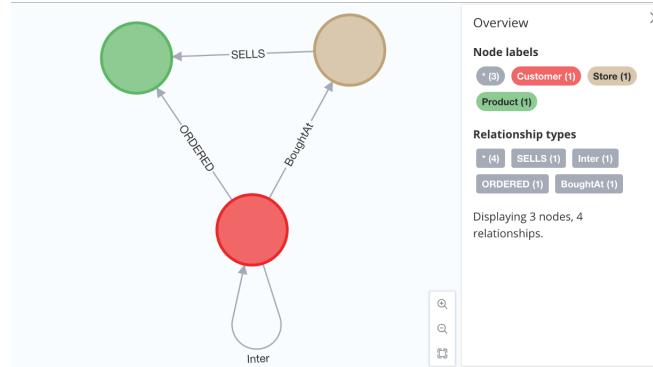


Figure 5: review 커뮤니티 시각화

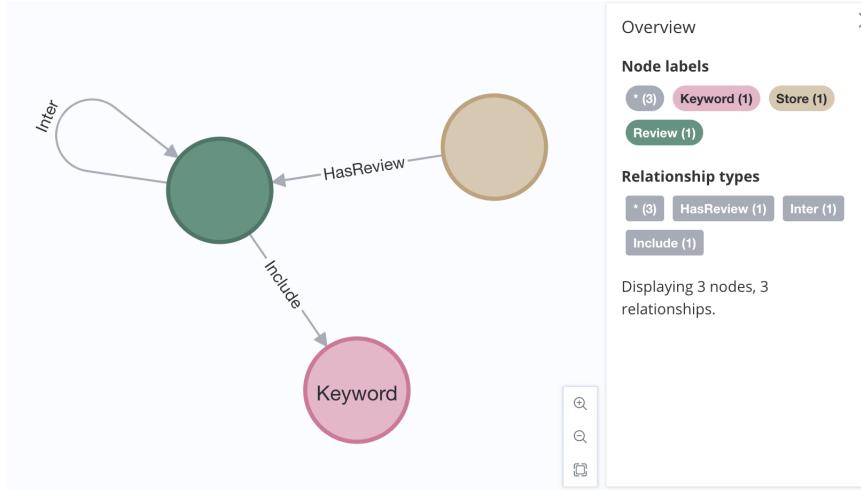
bloom 을 통해 전체 고객과 리뷰 노드의 커뮤니티 속성을 고유키로 설정해 시각화한 모습이다. Customer 노드는 총 2개, Review 노드에서는 10개의 커뮤니티가 감지된다. 두 데이터 모두 가중 그래프를 바탕으로 학습시켰고 알고리즘의 modularity 최적화 과정에서 가중치가 높은 에지가 선택되므로 커뮤니티가 영역으로 구분되기 보다는 골고루 퍼져 있는 형태임을 알 수 있다.

2 Neo4j 데스크탑 db 생성 과정

neo4j 데스크탑에서 생성한 쿼리의 원본 메타 그래프로 더 직관적이고 이해하기 쉬운 것을 알 수 있다.



만들고자 하는 추천 시스템은 고객이 어떤 가게를 몇 번 방문했는지를 바탕으로 설계되었으므로 여기서는 ORDERED 관계를 제외한 노드, 관계를 사용했다.



만들고자 하는 추천 리뷰는 각 리뷰는 어떤 키워드를 가지고 있고 서로 다른 리뷰 사이에 공통된 키워드를 바탕으로 유사성을 감지하고 그 리뷰와 가게 목록을 보여주는 것이 목적이므로 모든 노드, 관계를 사용했다.

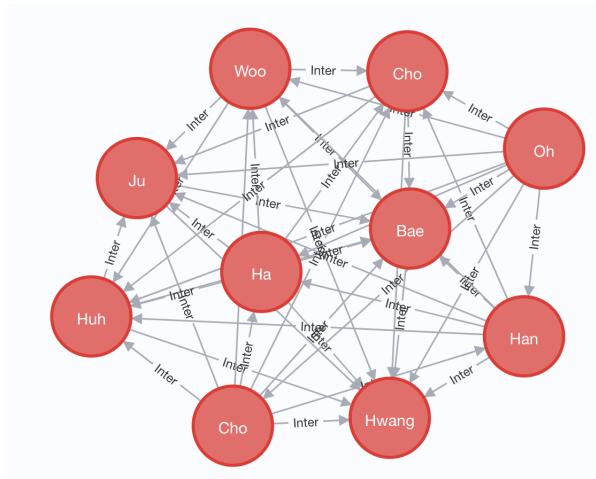
2.1 페이지 랭크 알고리즘

```

match (c:Customer)
return ID(c),c.customerId,c.lastname,c.pagerank
order by c.pagerank desc
limit 10

match (r:Review)
return ID(r),r.store,r.text,r.pagerank
order by r.pagerank desc
limit 10

```



리모트에서와 같은 결과를 보여주는 쿼리이다. 바로 더 직관적인 결과를 확인 할 수 있다.

2.1.1 페이지 랭크 생성과정(company)

- 먼저 서로 다른 조합의 customer 노드 사이에 공통으로 주문한 store를 바탕으로 Inter relationship을 만든다.

```
CALL apoc.periodic.iterate(
  'MATCH (a:Customer)-[:BoughtAt]→(s:Store)←[:BoughtAt]-(b:Customer)
  WHERE id(a) > id(b)
  return a, b, count(*) as weight',
  'MERGE (a)-[r:Inter]-(b)
  ON CREATE SET r.w = weight',
  {batchSize : 10000})
YIELD batch, operations
```

두 노드 사이에 중복된 관계 생성을 피하기 위해, 화살의 방향은 항상 노드 id가 높은 곳에서 낮은 곳으로 흐른다.

- 다음으로, 그래프 이름, 노드를 어떻게 매칭할지 알려주는 노드 쿼리, 사용할 관계&가중치를 설정하는 예지 쿼리를 인자로 하여 in-memory 그래프를 생성한다. in-memory 그래프의 범위는 전체 노드로 설정 한다.

```
CALL gds.graph.project.cypher('Customer_Inter', 'MATCH
  (c:Customer) RETURN id(c) AS id', 'MATCH (n:Customer)-
  [e:Inter]-(m:Customer) RETURN id(n) AS source, e.w AS w, id(m)
  AS target')
```

- 마지막으로 생성한 Customer_Inter graph를 바탕으로 pagerank를 계산하고 Customer 노드의 속성으로 추가한다.

```
CALL gds.pageRank.write('Customer_Inter', {maxIterations: 20,
dampingFactor: 0.85, relationshipWeightProperty: 'w',
writeProperty: 'pagerank'}) YIELD nodePropertiesWritten,
ranIterations
```

2.1.2 페이지 랭크 생성과정(review)

- 전체적인 쿼리 구조는 company db 와 같지만 여기서는 서로 다른 2개의 Review 노드가 공통으로 가지는 keyword를 바탕으로 Inter relationship을 만든다.

```
CALL apoc.periodic.iterate(
  'MATCH (a:Review)-[:Include]→(k:Keyword)←[:Include]-(b:Review)
  WHERE id(a) > id(b)
  return a, b, count(*) as weight',
  'MERGE (a)-[r:Inter]-(b)
  ON CREATE SET r.w = weight',
  {batchSize : 10000})
YIELD batch, operations
```

2.2 community 찾기 (louvain)

- 같은 in-memory 그래프를 이용하여 루바인 알고리즘을 통해 community를 노드 속성으로 추가할 수 있다.

```
CALL gds.louvain.write('Review_Inter',
{relationshipWeightProperty: 'w', writeProperty: 'community' })
YIELD communityCount, modularity, modularities
```

	communityCount	modularity	modularities
1	2	0.037707863855890165	[0.037707863855890165]

Started streaming 1 records in less than 1 ms and completed after 85630 ms.

modularity는 커뮤니티 구조를 평가하는 척도로 커뮤니티 내의 노드가 다른 커뮤니티의 노드보다 서로 더 밀접하게 연결되어 있는 정도를 측정한다. -1 to 1 사이의 값을 가질 수 있으며 0은 무질서한 랜덤 상태를 나타낸다. 여기서는 0.3에 가까운 수가 customer 노드 사이에 community가 존재함을 보여준다.

3 추가사항 & 향후 보완 가능성

- 구매 정보, 날씨 데이터를 바탕으로 만든 추천 시스템 결과를 neo4j에 넣었다. 총 9개의 날씨 카테고리마다 가게 추천리스트가 존재한다.
- 이를 이용해 각 가게의 리뷰 데이터(크롤링 결과)도 같이 웹페이지로 보여주기에는(store id를 바탕으로 조인하여) 먼저 리뷰데이터는 총 380개 가게 목록에서 50개밖에 존재하지 않아 가게 이름과 사진만 웹페이지에서 나타내기로 했다. 후에 리뷰데이터가 누적된다면 같이 보여줄 계획이다.
- 또한 Neo4j에서 학습시킨 Customer 노드는 1651개로 apriori 알고리즘의 고객 수와 동일하지만 구매 날짜를 기준으로 제외된 구매 기록이 존재하기 때문에 Store(402 → 380) Product(9049 → 5978) apriori 결과 (Customer_Similar에 저장된)에서 다른 노드로의 조인은 가능하지만 역방향은 불가능하다. 이 부분에서 보완 필요성을 느꼈다.