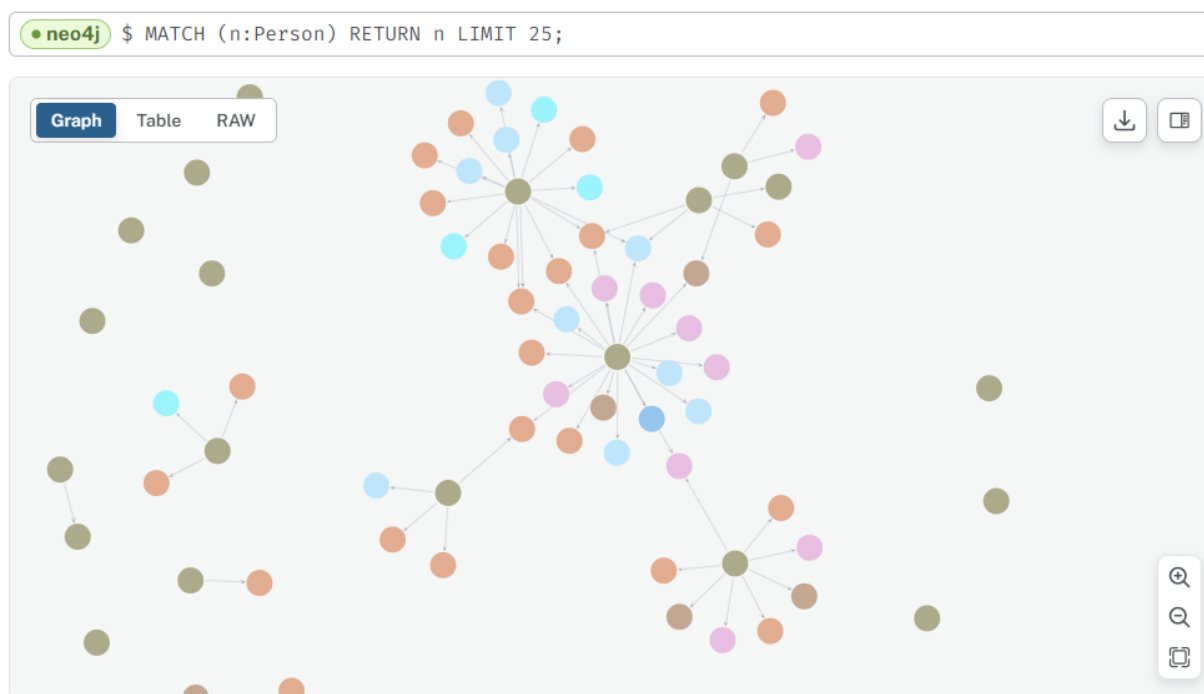


추천시스템 밑바닥부터 설계

시작하기에 앞서, KGAT에 대한 포기를 선언하겠다.

개발환경 세팅까지 해도 자원 문제로 로컬에서 돌아가지 않는다. 그렇다고 코랩에 올려서까지 돌려도 코드가 너무 길어서 이해하려 시간을 쏟을 여유가 없다. 그리고 애당초 학습 데이터(구매 로그라든가)가 전무하다.. 그래서 기존 모델을 가져와서 학습시키거나 그럴 수가 없다.

그래서 전환한 방향은 더이상 논문이나 모델 리서치가 아니라 은희가 만든 neo4j 그래프를 직접 탐색하면서 아이템 추천의 아이디어를 직접 떠올리는 것이다. 분명 시각화된 지식그래프를 보면 힌트가 있을 것.



대화 데이터를 지식그래프로 만들면 이런 그림이 나온다.

지식그래프는 노드와 엣지로 이루어지는데, 중요한 건 노드의 label(노드 색상으로 표현, type이라고 생각하면 됨)이 다르다는 점이다. 위 이미지 상에선 갈색 노드가 person label에 해당하고 가장 많은 엣지를 보유한다.

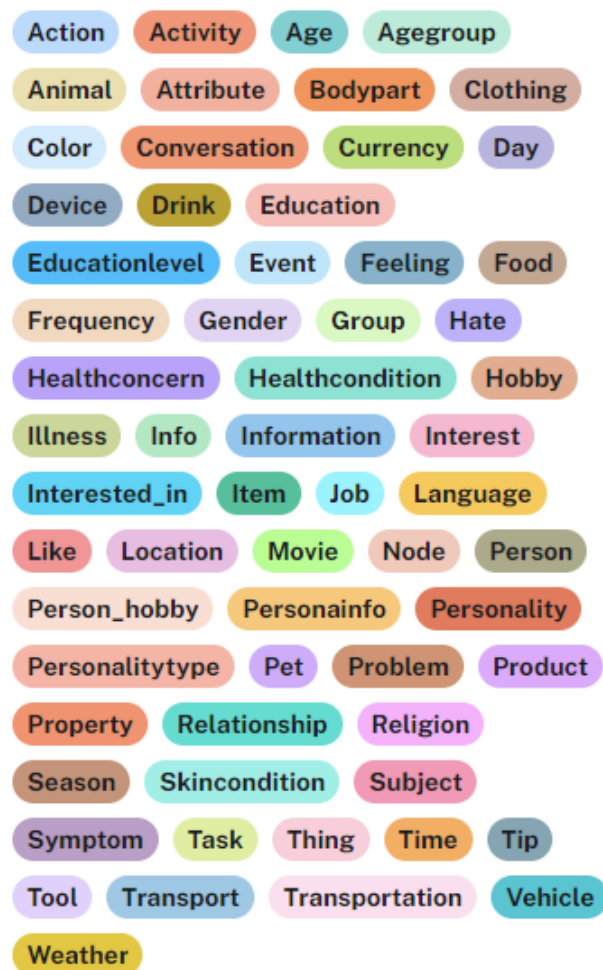
각각의 노드는 {id, 각종 metadatas}에 대한 정보를 가지고 있다.

1. 지식그래프 분석

label에 대한 분포가 분명 균일하지 않고 쏠려있을 것이고, 모든 label의 노드가 추천에 사용되는 것이 아니라 추천에 크게 도움이 되는 label이 있을 것이다.

총 7472개의 노드가 가진 label 종류는 다음과 같다.

Nodes (7,472)



그런데 db에서 person 25명을 뽑아내어 해당 person들과 관계(1-order 연결)되어있는 다른 노드의 label 종류를 보면 event, food, hobby, information, job, location 밖에 없음을 확인할 수 있다.

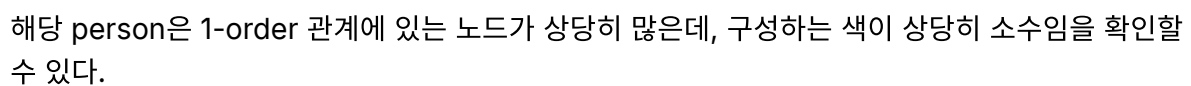
Results overview

Nodes (99)

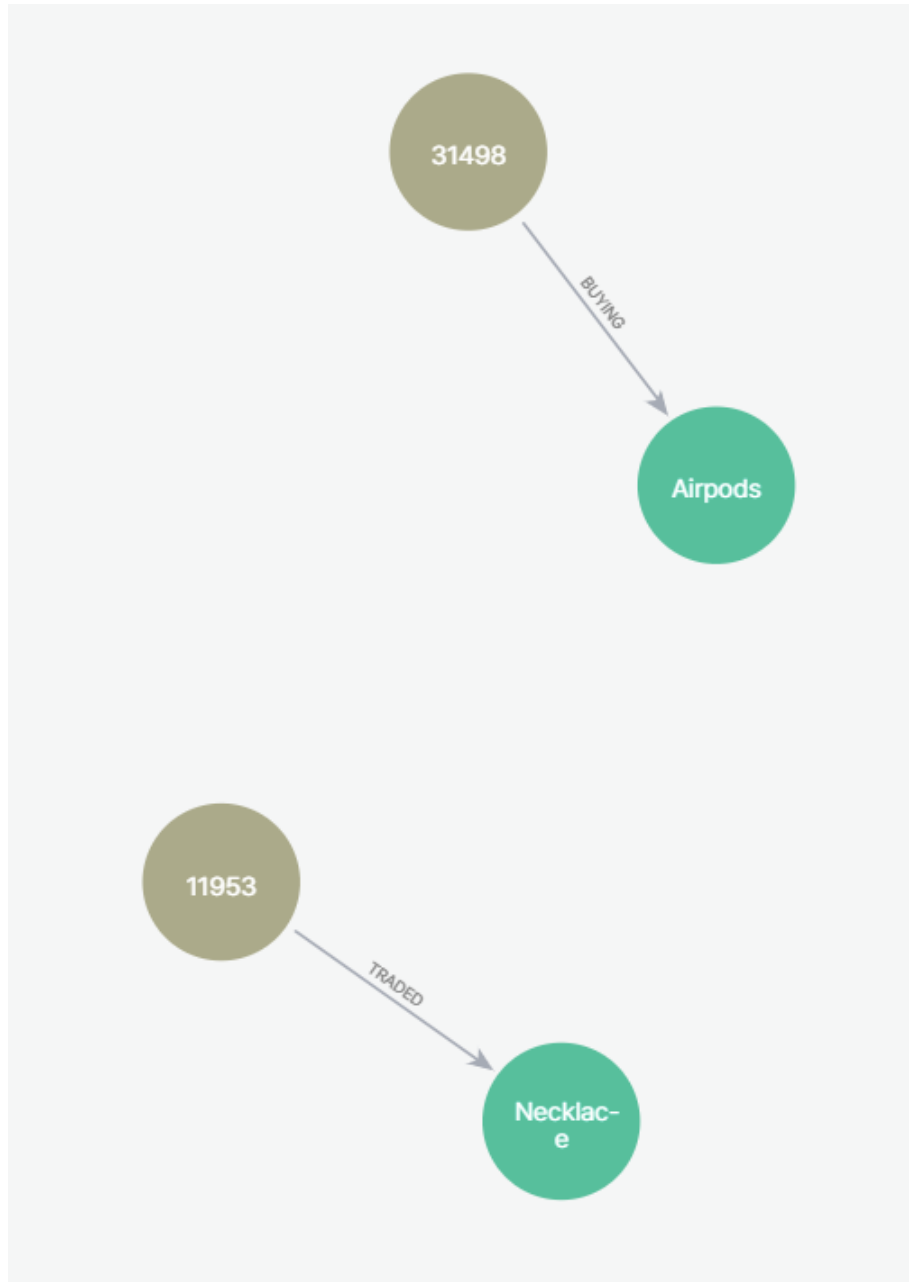


Relationships (87)





추천시스템 밑바닥부터 설계



2. 추천 시스템 구상

대화 데이터로 만들어진 지식그래프는 다음 특징을 가졌다.

- 1) 발화자 person 노드로부터 edge가 빠져나와 다른 노드를 가르키는 1-order 그래프 형성 (마이셀 구조)
- 2) 특정 발화자 person의 이웃에 다른 발화자 person이 존재하지 않는다.

특정 아이템을 특정 발화자에게 추천해야하는 상황인데, 우리가 알 수 있는 정보는 1)특정 아이템과 특정 노드와의 관계(유사성) + 2)특정 노드와 특정 발화자의 관계(지식그래프)이다.

이 두 가지 정보를 조합하여 각각의 아이템에 대해 전체 발화자들을 대상으로 하는 추천확률벡터(확률이라는 단어가 약간 부적합하긴 한데 1로 normalized된 추천 가중치 벡터)를 출력하는 것을 본 추천시스템의 목적으로 한다.

이를 위해 1)특정 아이템과 특정 노드와의 유사성은 pretrained word embedding을 떠올리는 게 자연스럽고, 2)특정 노드와 특정 발화자의 관계그래프는 adjacent matrix를 떠올릴 수 있다. 이 두 파트 각각의 구현 방안에 대해 구체화하겠다.

1) 특정 아이템과 특정 노드와의 유사성

가장 naive한 방식은 cos similarity.

당연히 이 방식에는 문제가 있다.

문제 1: 음수 유사성은 존재하지 않는다. 관련이 있거나 없거나, 즉 0~1 사이의 값으로 mapping해야 한다.

이 문제의 해결방안은 sigmoid같은 치역이 양수 영역에 있는 매핑 함수를 사용하는 것이다.

그러나 필자는 exponential(cos similarity)를 추천하고싶다. (softmax라고 봐도 됨)

노드 '사과'는 추천아이템 '닌텐도'와 유사도 0.2을 가지고, 노드 키위는 유사도 0.3를 가진다고 하자. 둘의 유사도 차이는 0.1이다. 반면 노드 '전자레인지'와 추천아이템 '닌텐도'는 유사도 0.7을 가지는데 노드 'PC'는 유사도 0.8 을 가진다고 하자. 이 또한 유사도 0.1의 차이이지만 실제로는 사과-키위의 차이 보다 엄청난 유사도 차이라고 생각한다. 즉, 요지는 유사도가 커질수록 mapping function의 기울기가 커지는(=이계도함수가 양수인=convex) 함수를 채택하는 것이 옳다.

문제 2: 유사도가 어느정도 있는 노드더라도 '추천'을 하기엔 적합하지 않은 상황이 많다.

이 문제의 해결방안은 노드의 label 정보를 활용한다는 것이다.

어떤 단어(노드)에 label이 붙는 방식은 그 단어의 의미적 상위(추상화된) 레벨의 범주를 지칭하는 단어가 label이 되는 것으로 추측한다. 이를 이용하면 추천할 아이템에 부여된 label과 임의의 노드에 부

여된 label간의 유사성을 비교하면 측정된 임베딩 유사도와 결은 비슷하지만 추가적인 정보를 얻을 수 있을 것으로 기대한다.

이 label 유사도에 대한 정보는 점수 자체로 활용하기보다, label간 유사도가 threshold 이하로 낮다면 사전에 노드를 거르는 filter로 사용이 가능하겠다.

→ 이걸 정말 중요한 문제지만, 밑에서 더 나은 방식으로 해결

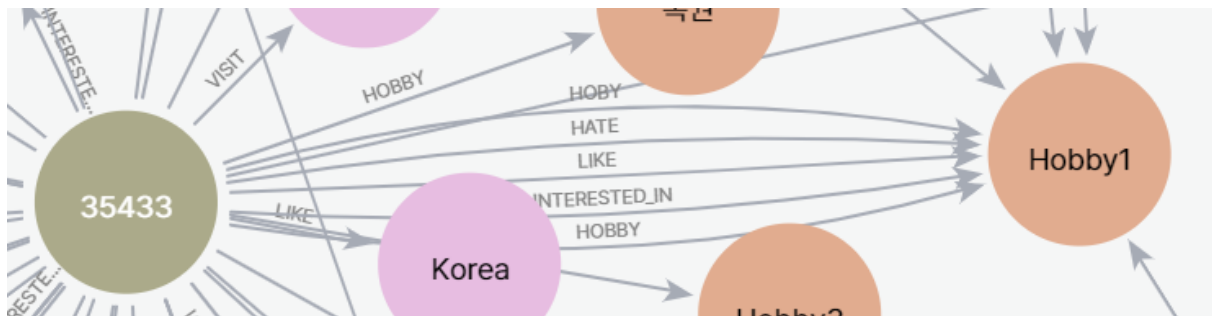
2) 특정 노드와 특정 발화자의 관계

이는 총 발화자 수*총 노드 수인 adjacent matrix로 표현 가능하다. 물론 이것만 구현하는 것보다 현명한 방법이 있다.

생각해볼 부분은 해당 발화자와 특정 이웃노드가 '얼마나 강하게' 이어져있는지이다. (이는 KGAT에서 self attention training으로 해결한 부분이기도 하다.)

즉, edge intensity를 정량화해야하고, 그에 대한 인자는 다음과 같다.

1. edge의 수, 얼마나 발화자의 관심을 드러내는 동사인지 측정



이런식으로 하나의 이웃노드를 다른 동사(edge)로 말하면 여러 edge가 생성되는 것 같다.

잘 보면 LIKE같은 긍정적 edge도 있고, HATE같은 부정적 edge도 있기에 동사의 sentiment 분석으로 가중치를 부여할수도 있겠지만, 부정적 발화라고해서 해당 노드에 '관심이 없다'고 단언할 수 없다. (e.g. 나 사과 안먹어봤어, 우리 집엔 닌텐도가 없어)

그것보다 동사 토큰이 얼마나 발화자의 관심과 기호를 반영하는지 수치화하는 게 중요하다.

너는 주어진 동사 토큰이 얼마나 발화자의 관심과 기호를 반영하는지를 0~10점 사이의 정수 점수를 부여해야해.

'immersed in', 'take', 'visit', 'interested in' 4개의 토큰에 대해 점수를 부여해줘



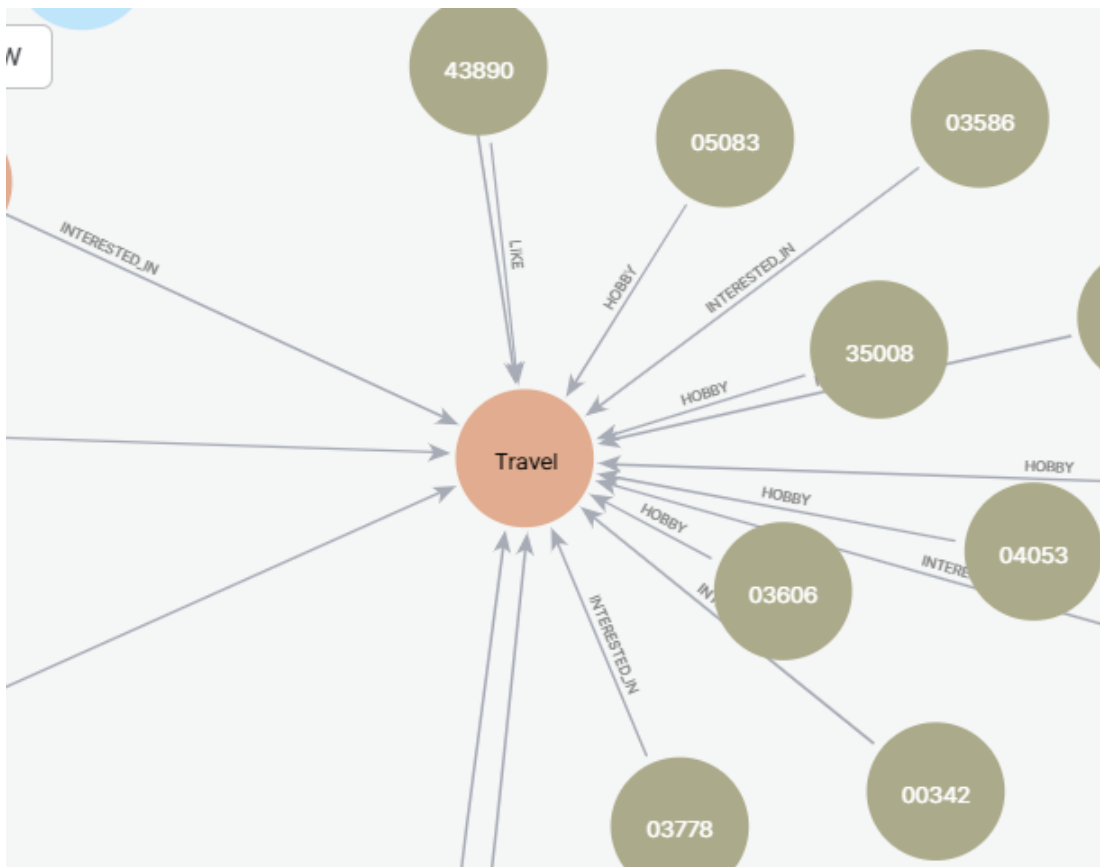
주어진 동사 토큰이 발화자의 관심과 기호를 얼마나 잘 반영하는지를 0부터 10까지의 정수로 점수를 매길 수 있어요. 제 생각에는 아래와 같은 점수를 부여할 수 있겠네요.

- "immersed in": 8
- "take": 5
- "visit": 6
- "interested in": 9

"immersed in"은 몰입하고 깊게 관심을 가진다는 의미를 갖고 있어서 발화자의 관심과 기호를 잘 반영할 것으로 생각합니다. "interested in"은 직접적으로 관심을 나타내는 표현이므로 높은 점수를 주었습니다. "take"와 "visit"은 상대적으로 중립적인 동사로, 관심을 나타내지만 몰입도가 높지 않을 수 있기 때문에 중간 정도의 점수를 주었습니다.

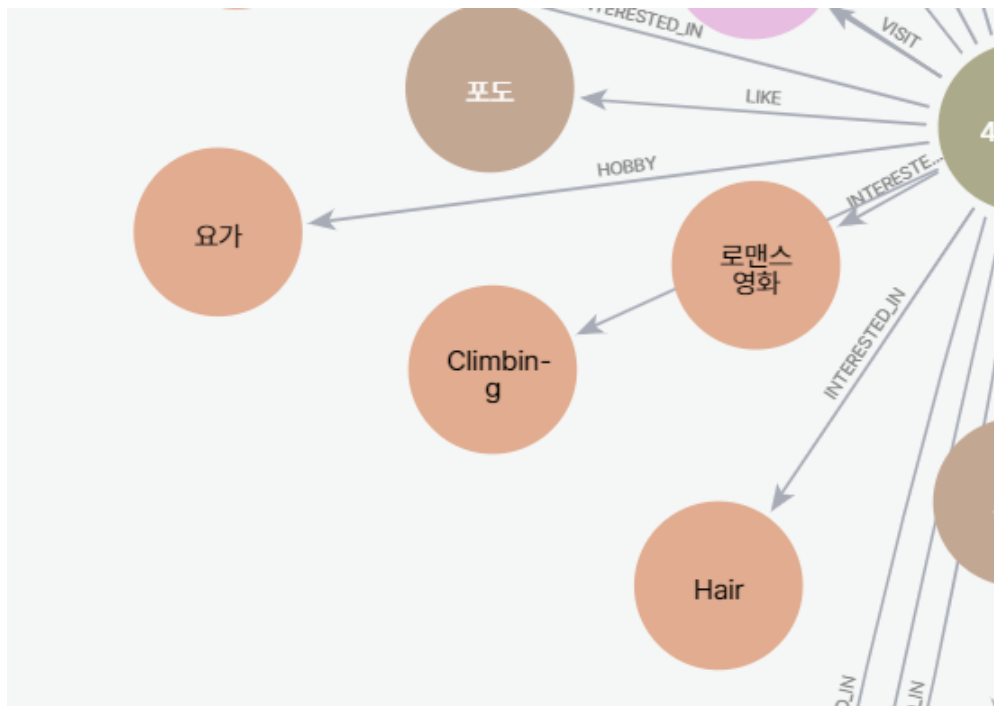
이런 방식으로 edge 별 강도를 측정할 수 있다.

2. 얼마나 해당 노드가 발화자에게 특별한가?



여행은 누구에게나 좋은 hobby이다.

그런데 어떤 hobby는 특정한 사람만 가지는 것이기도 하다. 예를 들어 클라이밍이라고 하자.



이런 상황에서 '해외로 여행을 가서 클라이밍 체험을 하는 패키지 상품'을 추천해야한다고 하자. 즉, 이 추천상품은 travel과 climbing 둘 다 유사도가 0.8로 같다고 가정하자.

해당 상품을 한정된 인원에게만 추천할 수 있다고 하면, 클라이밍을 취미로 하는 발화자에게 추천하는 게 현명하다. 유튜브로 음악 추천 재생을 틀었는데, 모두가 좋아하는 대중가요가 들려나오는 순간보다 나만의 마이너한 장르가 딱 들려나왔을 때 사람은 홀리기 쉽다.

정리하면 어떤 노드로 들어오는 edge 수가 많을수록 그 edge의 강도는 약화되어야한다.

방법1에서는 특정 유저가 해당 노드로 뻗는 edge의 수에 비례하게 강도를 설정하자고 주장했고, 방법2에서는 특정 노드로 들어오는 edge 수에 반비례하게 edge 강도를 설정하자고 주장했다. 이는 NLP task에서 TF-IDF와 의미적으로 동일하다.

$$TF(t, d) = \frac{\text{문서 } d \text{ 에서 단어 } t \text{ 가 등장한 횟수}}{\text{문서 } d \text{ 에 등장한 모든 단어의 수}}$$

$$IDF(t, D) = \log \left(\frac{\text{총 문서의 개수}}{\text{단어 } t \text{ 를 포함하는 문서의 수}} \right)$$

$$TF-IDF(t, d, D) = TF(t, d) * IDF(t, D)$$

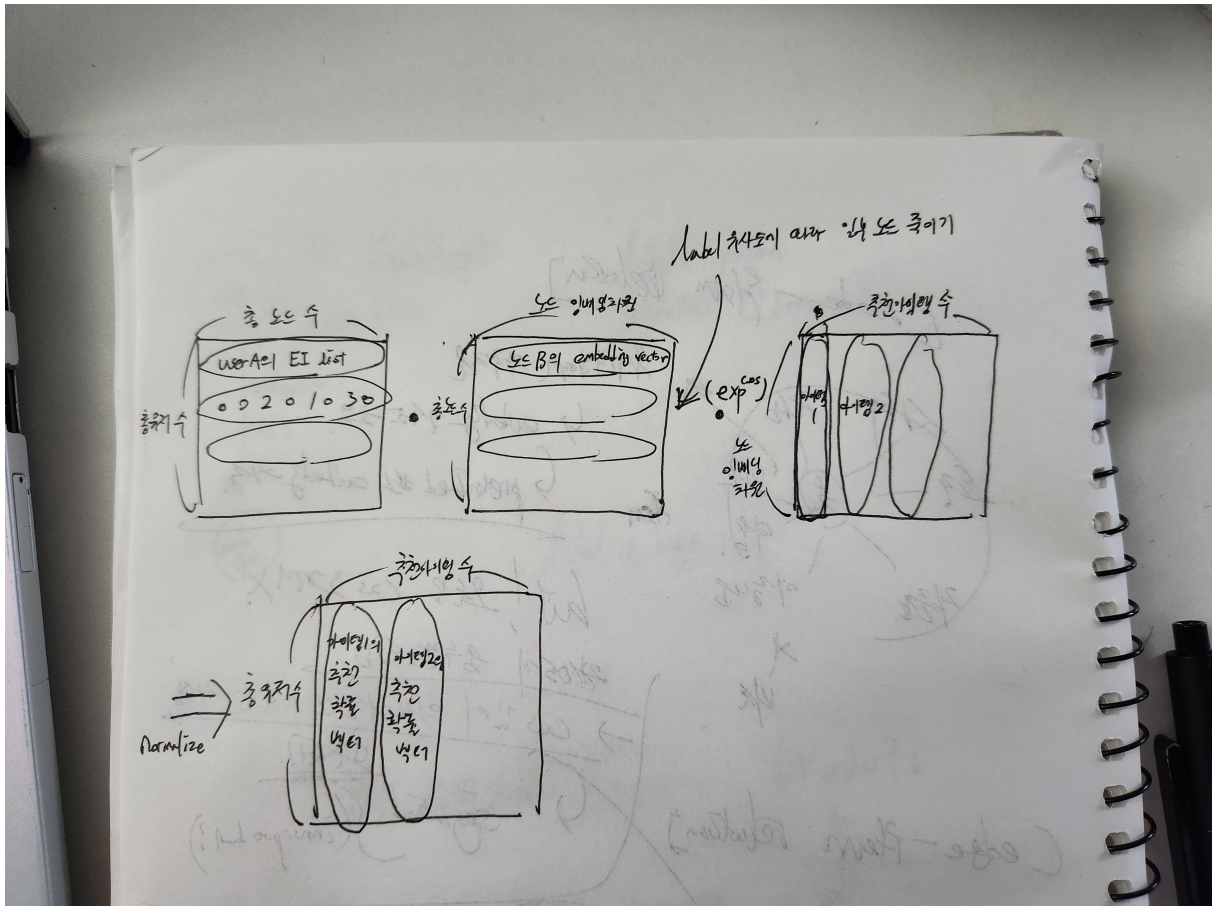
즉, 특정 edge의 intensity는 다음과 같이 표현된다. (이하 EI라고 부르겠다)

edge_intensity = edge토큰의 기호반영수치 * log(edge가 향하는 이웃노드의 in-degree)

이 EI를 이용해 user-node 간 adjacent matrix를 구성할 수 있다. (edge가 존재하면 EI, 존재하지 않으면 0)

3) 특정 아이템과 특정 유저의 관계

이제 위에서 구체화한 2가지 파트를 수합하자. 위 모든 과정은 간단한 행렬곱으로 표현이 가능하다.



총 3개의 행렬이 있는데, 왼쪽부터 각각 1) 유저-노드 adjacent matrix, 2) 노드 임베딩 행렬(행벡터 기반), 3) 추천아이템 임베딩 행렬(열벡터 기반) 이다.

위에서 말했듯, 노드 임베딩 행렬은 추천아이템과의 label 유사도를 이용하면 일부 노드(행벡터)를 drop할 수 있다.

또 노드 임베딩 행렬의 각 행벡터와 추천아이템 임베딩행렬의 각 열벡터는 단순 행렬곱(inner product)이 아니라 $\text{np.exp}(\text{cosine_sim}(v1, v2))$ 의 형태로 연산된다.

마지막으로 결과물 행렬의 각 열벡터를 $\text{norm}=1$ 로 정규화를 거치면 추천아이템 별 유저들에 대한 확률 분포를 얻는다. 해당 분포에 따라 아이템 별 적합한 추천 대상을 선정할 수 있다.

1차 구현 결과

위에서 구상한 과정을 코드로 구현했다.

```
target_users=list[str] = get_unique_target_people(graph, minimum_neighbors_for_target = 5)
target_users_adj_list=dict[str, list[dict[str,str]]] = get_adj_rel_to_node(target_users, graph)

unique_relationships = get_unique_relationships(graph)
prompt = [{ 'role': 'system', 'content': 'json 형식으로 출력해줘.' }, { 'role': 'user', 'content': '추어진 동사가 얼마나 발화자의 관심과 기호를 반영하는지 0~1점 사이의 \
점수를 부여해서 {단어:점수, 단어:점수, ...} 형태로 출력해줘. 예시는 다음과 같아. \
[예시] \
input: ["immersed in", "take", "visit", "interested in"], \
output: {"immersed in":0.8, "take":0.5, "visit":0.6, "interested in":0.9}'
+ f'input으로 사용될 단어 리스트는 다음과 같아: {unique_relationships}' ]
rels_to_interest_score = get_json_from_prompt(prompt)

unique_nodes=list[str] = get_unique_nodes(graph)
nodes_to_IDF=dict[str,float] = get_IDF_of_node(unique_nodes, graph)

nodes_to_embeddings=dict[str, list[float]] = get_words_to_embeddings(unique_nodes)
#EMBEDDING_LEN = 1536

items = ['coke', 'travel_package', 'playstation', 'cookies', 'education service']
items_to_embeddings=dict = get_words_to_embeddings(items)
#EMBEDDING_LEN = 1536
```

특정 활성유저(조건: 이웃노드 10개 이상) 한 명을 대상으로 추천 아이템 리스트와의 유사도 측정 결과는 다음과 같다.

추천 아이템 리스트: [coke, travel_package, playstation, cookies, education service]

```
test_user = target_users[3]
recommendation_items_score = calculate_similarity(target_users_adj_list[test_user], rels_to_interest_score)

{'coke': 11.592217557331889,
 'travel_package': 11.568624559979908,
 'playstation': 11.048422982830663,
 'cookies': 11.80904000788163,
 'education service': 11.537511011364401}
```

그렇다. 문제가 있다는 걸 나도 안다.

유저와 이웃한 모든 노드가 모든 item과 가지는 모든 유사도를 더하는 방식으로 각 item에 대한 유사도를 구했다.

그러니 정보를 많이 가진(=주변 노드가 많은) 유저의 경우, 추천아이템 리스트와 무관한 노드가 90% 이상인데 그 무관한 노드에 의해 더해지는 score이 전체 score을 지배해버린 상황이다.

해결책은 다음과 같다.

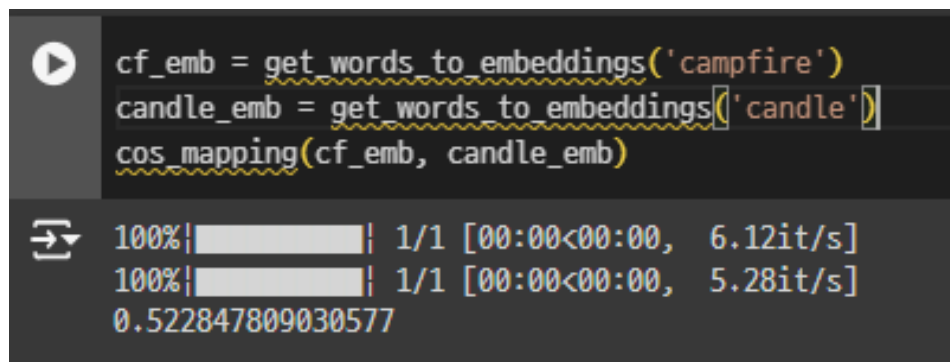
1) 추천 아이템 리스트와 관련있는 이웃 노드만 고려하고 나머지 노드는 버린다.

직관적으로 생각해도 대부분의 노드는 분명 추천아이템과 관련이 없는데 score 책정 과정에 쓰이고 있다. 그런데 추천 아이템의 워드 임베딩과의 유사도가 낮다고 정보를 버리면 남아있는 정보가 거의 없다는 문제가 있다.

2) 그래서 추천아이템에 대한 정보를 워드 임베딩만으로 추출하면 안된다.

워드임베딩이 추천아이템을 대표하지 못한다는 주장에 대한 사례 2가지를 보이겠다.

사례1. 양초라는 추천 아이템을 추천하고자 한다고 하자. 유저 A는 최근 캠핑을 다녀와서 이웃노드로 '캠프파이어'이 잡혔다. 양초와 캠프파이어는 임베딩 측면에서 유사도가 높지만, 그렇다고 양초를 추천하는 것은 부적합하다.



```
cf_emb = get_words_to_embeddings('campfire')
candle_emb = get_words_to_embeddings('candle')
cos_mapping(cf_emb, candle_emb)
```

100%|██████████| 1/1 [00:00<00:00, 6.12it/s]
100%|██████████| 1/1 [00:00<00:00, 5.28it/s]
0.522847809030577

[유사도] ~0.3: 유사x, 0.3~0.4: 직관적으로 유사, 0.5~: 단어 자체가 유사함

또한 단어 유사도만으로는 '추천을 위한' 유사도를 반영하지 못할 수 있다.

사례2. 꽃꽂이를 좋아하고 핸드크림을 자주 바르는 사람이 있다고 하자. 이 사람에게 양초를 선물하면 좋아할 가능성이 높다고 생각했다. 그 추측의 이유에는 [향기, 단아한 라이프스타일]처럼 '꽃꽂이'와 '핸드크림'이 가진 추상적 속성이 '양초'가 가지는 속성과 공통적이기 때문이다. 하지만 [양초]와 [꽃꽂이, 핸드크림] 간의 단어적(임베딩) 유사성을 찾긴 힘들다. 단어 자체가 유사한 맥락에서 자주 쓰이지는 않기 때문이다.

따라서 추천할 아이템의 '속성'을 여러 부류로 구분하고, 유저의 이웃 노드가 해당 속성에 얼마나 부합하는지를 알 수 있다면 더 정확한 추천이 가능할 뿐더러 정보의 희소성(즉, 이웃 노드가 별로 없는 유저를 향한 추천) 문제도 해결할 수 있다고 생각했다.

2차 접근: 추천 아이템에 대한 정보 증강

llm을 이용하여 추천 아이템에 대한 정보를 지식그래프로 증강하여 유저 지식그래프와 아이템 지식그래프 간의 유사도를 측정해야한다.

[Item_KnowledgeGraph 추출 결과]

```
item = 'candle'

item_KG = get_item_KG([item])
item_KG

{'candle': {'Aesthetic Appeal': ['Decorative purposes',
    'Ambiance enhancement',
    'Aromatherapy'],
    'Functional Use': ['Lighting', 'Scented atmosphere', 'Emergency lighting'],
    'Variety': ['Different scents', 'Colors', 'Shapes and sizes'],
    'Eco-Friendly Options': ['Soy wax candles',
    'Beeswax candles',
    'Recyclable packaging'],
    'Seasonal Offerings': ['Holiday-themed candles', 'Seasonal scents'],
    'Health Benefits': ['Stress relief', 'Improved focus', 'Mood enhancement'],
    'Gift Giving': ['Housewarming gifts', 'Hostess gifts', 'Thank you gifts']}}
```

추천아이템 candle의 properties은 'Aesthetic Appeal', 'Functional Use', 'Variety'...임을 확인할 수 있다.

KG 구성 형식은 {'아이템의 속성을 대표하는 대분류': ['구체적인 예시1', '구체적인 예시2', '구체적인 예시3']} 이 되도록 prompt를 짰다.

프롬프트:

```
"Make knowledge graph of properties of "
+ item_name
+ " which are the reasons expected customers consume "
+ item_name
+ ". Here is an example of the knowledge graph of flower
+ '{"Freshness": ["Same-day delivery", "Locally sourced
+ '[detailed explanation of the example] Output dict of
```

[item_KG와 user_KG간의 유사도 계산 방식]

위에서 문제였던 '관련없는 노드가 추천스코어에 개입하는 현상'을 최소화하기 위해 다음 알고리즘을 구상했다.

핵심은 각 (관계+노드)와 추천아이템 간의 유사도는

추천아이템의 **속성** 중 가장 유사한 것을 고르고, 그 속성의 **사례** 중에서 가장 유사한 것을 한 번 더 골라서 유사도 점수를 계산한다는 점이다.

- 1) 관계(동사)+노드(명사) concatenate를 embedding한 벡터1과 추천아이템의 properties(여러개)를 embedding한 벡터2(여러개)간의 유사도 계산
→ 가장 유사도가 높은 best_related_property 선정
- 2) 선정된 best_related_property의 하위에 있는 instances(여러개)를 embedding한 벡터3과 다시 벡터1의 유사도 계산
→ 가장 유사도가 높은 best_matched_instance 선정
- 3) 해당 '**노드**'가 지니는 해당 '**추천아이템**' 대상 점수 = best_related_property 점수 * best_matched_instance 점수
- 4) 지정한 threshold 이상의 점수를 지닌 노드만 추출 (quantile 기준으로 잘라도 될 것 같다.)
- 5) 추출된 노드 점수를 총합하여 해당 '**유저**'가 지니는 해당 '**추천아이템**' 대상 점수 산정

[유사도 계산 알고리즘 핵심 고려 요소]

Q1) 왜 관계와 노드를 concatenate?

→ 단순 명사(노드)가 아니라 동사+명사(관계+노드)를 임베딩하면 해당 벡터에는 '**이벤트**'의 정보가 담긴다고 생각했다.

'전자제품' 그 자체에는 ['첨단과학', '높은 가치'] 같은 속성이 연상되지만, 'interested_in 전자제품'은 ['nerdy', 'creative']같은 속성이 연상된다. 그리고 무엇을 추천할 때 고려해야 하는 점은 **추천아이템**이 제공하는 [가치, 상황, 분위기, 통념] 등이 얼마나 **사용자**의 [가치관, 습관] 등과 유사한지이기 때문에 노드의 이벤트 기반 embedding이 합리적이라 생각했다.

Q2) 왜 item이 가진 모든 property와 모든 instance간의 유사도를 전부 합하는 게 아니라 '가장' 적합한 property의 '가장' 적합한 사례만 고려?

→ 유저가 어떤 제품을 사용하는 이유는 그 제품이 가진 모든 속성을 exploit하고 싶기 때문이 아니라 특정 속성과 유저의 핏이 잘 맞기 때문이라 생각했다.

커피만 봐도 나같은 경우엔 (0.7 카페인 수혈용 + 0.2 시원함 + 0.1 맛)으로 이용하는 반면 내 어머니는 (1.0 맛)이다. 서비스에서 착취하는 효용의 측면이 서로 다르지만 나와 엄마는 둘 다 커피 상품의 지지자이다.(label=True)

그런데 커피라는 상품을 그다지 좋아하지 않는(label=False) 내 친구A는 커피라는 상품에서 얻는 효용이 (0.4 카페인, 0.4 시원함, 0.2 맛)이다. 그니까 커피의 속성 각각에서 얻는 효용이 전반적으로 균

등한 분포이고, 이 친구 입장에서 커피는 '이도저도 아닌 상품'일 것이다.

이런 인간심리를 생각했을 때, 상품의 모든 속성과의 유사도를 전부 고려해서 더하는 방식은 실제로는 그 사람에게 이도저도 아닐 상품인데도 나쁘지 않은 스코어를 책정하게 된다.

이런 맥락에서 유사도 선형결합보다는 maxpooling이, 그리고 maxpooling보다는 softmax같은 convex function을 이용한 downscaling이 본 문제에 적합하다고 봤다.

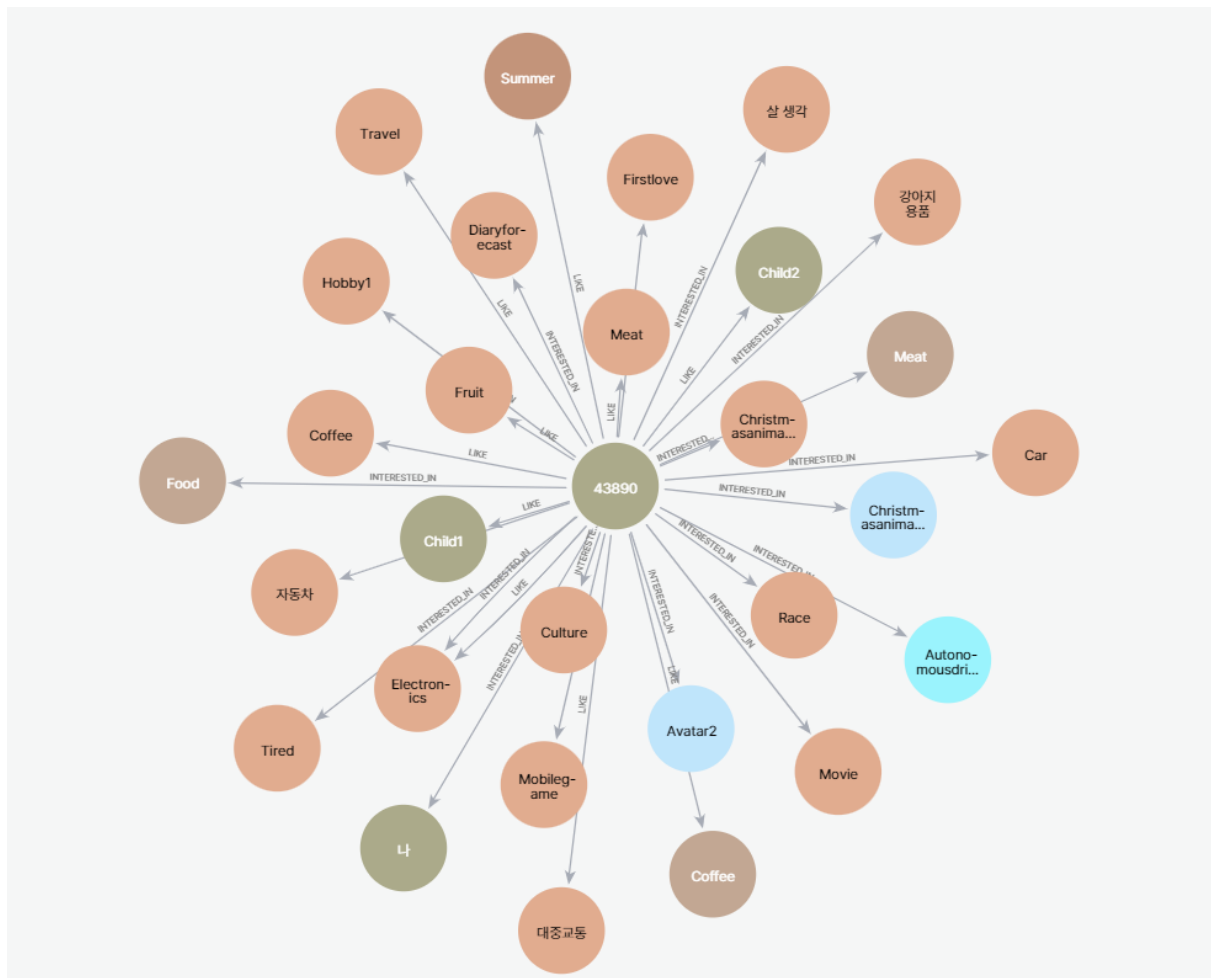
(cf. 추천시스템은 지연시간문제를 해결하는 게 꽤 중요한데, 알고리즘 time-complexity를 고려할 때 softmax를 사용하면 추천아이템의 모든 property의 모든 instance를 전부 traverse해야하기에 maxpooling을 유사도 결정 함수로 택하여 property단에서 한 번 가지치기했다.)

[테스트에 사용할 활성유저 선정]

반복하지만, label 없이 추천시스템을 만드는 상황이므로 모델 성능 평가에는 출력물 정성평가가 지배적이다.

꽤 많은 정보가 수집된 유저 한 명 선정해서 얼마나 모델이 괜찮은지 확인하자.

id: 43890, 이웃노드가 너무 많아 relationship = INTERESTED_ID or LIKE 만 filtering한 결과이다.



[추천아이템 'candle' 대상으로 해당 유저의 scoring 결과]

유저의 이웃 노드마다 어떤 경로를 통해서 추천아이템을 향한 유사도를 측정했는지 출력되도록 했다.
(threshold = 0.08)

```

< Recommended Item > :: [Node Name] --- [Best Related Property] --- [Best Matched Instance]

< candle > :: HOBBY Uniquename --- Personalization --- Personalized labels : 0.1035229488615975
< candle > :: INTERESTED_IN Food --- Health Benefits --- Improved focus : 0.0960209742194255
< candle > :: INTERESTED_IN Culture --- Personalization --- Personalized labels : 0.0848069428212268
< candle > :: INTERESTED_IN Christmasanimationmovie --- Seasonal Themes --- Holiday scents : 0.10118684798115372
< candle > :: INTERESTED_IN Christmasanimation --- Seasonal Themes --- Holiday scents : 0.11300167161000735
< candle > :: INTERESTED_IN Diaryforecast --- Seasonal Themes --- Seasonal designs : 0.08471074637241582
< candle > :: WORK_AS Work_Overtime --- Functional Use --- Emergency backup light : 0.09358323803055676
< candle > :: VISIT Dinner --- Seasonal Themes --- Gift sets : 0.08017670588305664
< candle > :: LIKE Fruit --- Variety --- Different scents : 0.0983536061506354
< candle > :: LIKE Summer --- Seasonal Themes --- Seasonal designs : 0.17766564335986626
< candle > :: DISLIKE Winter --- Seasonal Themes --- Seasonal designs : 0.11943811941990651
< candle > :: EXPERIENCED Weather --- Seasonal Themes --- Seasonal designs : 0.11058303545015417
{'candle': 1.2630504801600027}

```

추천의 이유가 합리적이라고 느낀 부분을 뽑아보면 다음과 같다.

추천 아이템	노드명(relationship + node)	(추천아이템) 가장 관련된 속성	(추천아이템) 가장 관련된 예시	유사도
< candle >	WORK_AS Work_Overtime	Functional Use	Emergency backup light	0.093
< candle >	INTERESTED_IN Christmasanimation	Seasonal Themes	Holiday scents	0.113
< candle >	LIKE Summer	Seasonal Themes	Seasonal designs	0.177

MATCH (p:Person)-[verb]→(person_trait), (item:RecommendedItem)-[hasprop]→(prop)-[hasinst]→(inst), (person_trait)-[similar]→(prop) WHERE p.id = '43890' AND item.id = 'candle' RETURN p, verb, person_trait, item, hasprop, prop, hasinst, inst, similar

