분석 23기 미니프로젝트2

도서 데이터 기반 추천시스템 구현

23기 분석 미니프로젝트2 4조 심심한 사과
윤왕규, 김경민, 백다은, 오영민

BOAZ

CONTENTS

사용 데이터셋

Amazon Reviews'23

Field	Type	Explanation
rating	float	Rating of the product (from 1.0 to 5.0).
title	str	Title of the user review.
text	str	Text body of the user review.
images	list	Images that users post after they have received the product. Each image has different sizes (small, medium, large), represented by the small_image_url, medium_image_url, and large_image_url respectively.
asin	str	ID of the product.
parent_asin	str	Parent ID of the product. Note: Products with different colors, styles, sizes usually belong to the same parent ID. The "asin" in previous Amazon datasets is actually parent ID. **Please use parent ID to find product meta.**
user_id	str	ID of the reviewer
timestamp	int	Time of the review (unix time)
verified_purchase	bool	User purchase verification
helpful_vote	int	Helpful votes of the review

사용 데이터셋

	rating	parent_asin	user_id	timestamp
20687409	1.0	0073378151	AE2222HXKLMAEK4SG56OW23V3LTA	1393429135000
25670196	1.0	B01719Y5XK	AE2222WEGI5MUXLFZW6UKKUSI74A	1461552308000
2205090	5.0	B004H4XI0Y	AE22236AFRRSMQIKGG7TPTB75QEA	1326679671000
2205085	4.0	B008P3JNHU	AE22236AFRRSMQIKGG7TPTB75QEA	1370651952000
2205093	5.0	B002PEP4SC	AE22236AFRRSMQIKGG7TPTB75QEA	1259456701000
...	...	...	...	...
14285733	4.0	B06WD8GJ3P	AHZZZYHANWL2OW5PGXDOBUTJXCTA	1516135047122
23959628	5.0	1633224635	AHZZZZ76NI5YF4RP5TKCQRGRQAGA	1521607388514
20325413	5.0	1628603283	AHZZZZUMHA57YCBU4WFINWREUBKQ	1545150722210
9100476	3.0	0894803123	AHZZZZUZCRIUWGMXKFAJO3T5S45A	1379960837000
3290827	5.0	1604863315	AHZZZZYA4NULLM5PNACIU3GA5OQQ	1653164753579

24281918 rows × 4 columns

1.rating (평점)

- **사용자가 상품에 대해 남긴 평점**으로, 일반적으로 1에서 5까지의 값을 가집니다.
- 1: 매우 불만족
- 5: 매우 만족

2.parent_asin (상품 ID)

- 리뷰가 작성된 **상품의 고유 ID**를 나타냅니다.
- ASIN(Amazon Standard Identification Number)은 아마존에서 사용되는 고유 식별자입니다.

3.user_id (사용자 ID)

- 리뷰를 남긴 **사용자의 고유 ID**를 나타냅니다.
- 특정 사용자의 리뷰 히스토리를 확인하거나 분석할 수 있습니다.

4.timestamp (리뷰 작성 시각)

- **리뷰가 작성된 날짜와 시간**을 나타냅니다.
- UNIX 타임스탬프 형식으로 기록되어 있습니다.

데이터 정제

1. 빈도 계산

```
# 사용자별 등장 빈도 계산
user_freq = train['user_id'].value_counts()

# 상품별 등장 빈도 계산
item_freq = train['parent_asin'].value_counts()

# 빈도 확인
print("상위 5명의 사용자 빈도:")
print(user_freq.head())

print("\n상위 5개의 상품 빈도:")
print(item_freq.head())
```

✓ 10.8s

```
상위 5명의 사용자 빈도:
user_id
AGSP6LSQK32SQEJO3YVVNACPWMSQ    2916
AGY5JM4JIEL6RX6X6NLULXZB7DOQ    2214
AG473D27NAWRW3N2RMWVYZ6ICBTA    2204
AF65CA4WVYV6HQI7WQOGR757JHRA    2079
AGWDYYVVWM3DC3CASUZKXK67G6IA    2022
Name: count, dtype: int64

상위 5개의 상품 빈도:
parent_asin
B00L9B7IKE    35985
B006LSZECO    25633
B00JO8PEN2    25399
B00DPM7TIG    16731
B016ZNRC0Q    12884
Name: count, dtype: int64
```

2. 상위 1000명의 사용자 & 1000개의 상품

```
# 상위 1000명의 사용자와 상품 선택
top_n_users = user_freq.head(1000).index  # 상위 1000명의 사용자
top_n_items = item_freq.head(1000).index  # 상위 1000개의 상품
```

✓ 0.0s

```
filtered_train = train[train['user_id'].isin(top_n_users) & train['parent_asin'].isin(top_n_items)]
```

✓ 2.7s

```
num_users = filtered_train['user_id'].nunique()
num_users
```

✓ 0.0s

```
804
```

```
num_items = filtered_train['parent_asin'].nunique()
num_items
```

✓ 0.0s

```
829
```

BOAZ

데이터 정제

3. 테스트셋

```python
# 테스트셋 필터링
filtered_test = test[test['user_id'].isin(top_n_users) & test['parent_asin'].isin(top_n_items)]

# 필터링된 테스트셋의 사용자 및 상품 수 확인
num_test_users = filtered_test['user_id'].nunique()
num_test_items = filtered_test['parent_asin'].nunique()

print(f"테스트셋 사용자 수: {num_test_users}")
print(f"테스트셋 상품 수: {num_test_items}")
```

✓ 0.6s

```
테스트셋 사용자 수: 585
테스트셋 상품 수: 602
```

```python
# 훈련 데이터 사용자와 상품
train_users = set(filtered_train['user_id'])
train_items = set(filtered_train['parent_asin'])

# 테스트 데이터에서 제거: 훈련 데이터에 없는 사용자와 상품을 모두 포함한 경우
filtered_test = filtered_test[
    (filtered_test['user_id'].isin(train_users)) |
    (filtered_test['parent_asin'].isin(train_items))
]

# 결과 확인
print(f"제거 후 테스트 데이터 크기: {filtered_test.shape[0]} 행")
```

✓ 0.0s

```
제거 후 테스트 데이터 크기: 1810 행
```

BOAZ

GLocal-K

- 협업 필터링 기반 추천시스템에서 **사용자-아이템 행렬은 희소하며**, 이는 **고차원 행렬의 결측값을 완성하는 문제**(matrix completion)로 표현할 수 있음

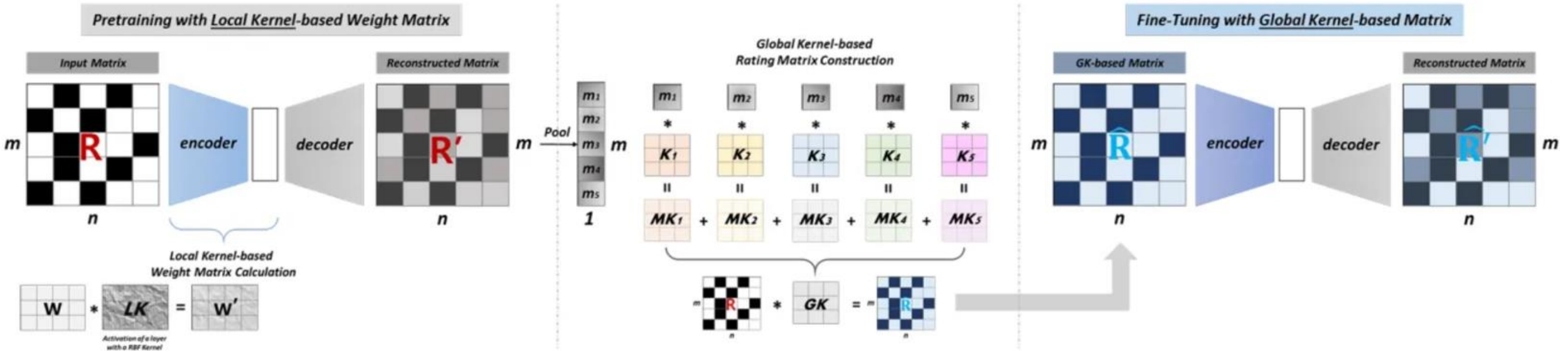
- 최근 연구들은 **Side Information(사용자 속성, 의견 정보 등)을 활용하는 것**에 집중하고 있지만 **실제 환경에서는 충분치 않음**

- 본 논문은 Side Information 대신, **고차원의 Rating Matrix를 저차원으로 일반화**하여 표현하는 것을 목표로 함

- **사용자 및 아이템의 잠재적 특징을 추출**하는데 중점을 둔 **Local & Global Kernel 기반 Auto-Encoder**

BOAZ

Glocal-K

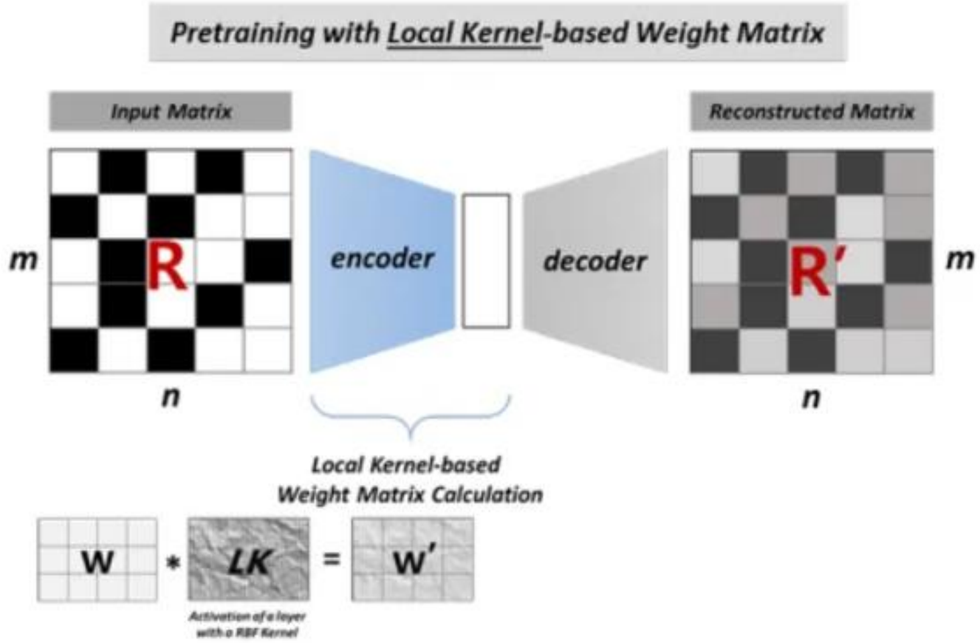


1) **Pre-training** : **2차원 RBF 커널**을 사용하여 데이터를 하나의 공간에서 특징 공간으로 변환하는 **local kernelised weight matrix**로 AutoEncoder pre-train

2) **Fine-tuning** : 각 아이템의 특성을 포착하는 **convolution-based global kernel**에 의해 생성된 평점 행렬로 pre-train된 AutoEncoder fine-tuning

Glocal-K

01 Pre-training : Local Kernel을 이용해 새로운 rating matrix 생성



Raw rating matrix

$$train_r = \begin{bmatrix} 5 & ? & 3 & 4 \\ 3 & 4 & ? & 5 \\ ? & 2 & 5 & 3 \end{bmatrix}$$

AutoEncoder

Pre-training rating matrix

$$train_r = \begin{bmatrix} 5 & 3.5 & 3 & 4 \\ 3 & 4 & 4.2 & 5 \\ 3.8 & 2 & 5 & 3 \end{bmatrix}$$

- 입력 matrix에 빈 평점을 생성

- 학습 과정에서, W(e)와 W(d)로 표현되는 인코더/디코더의 가중치는 2d-RBF kernel을 통해서 Reparameterised

$$K_{ij}(U,V) = \max(0,\ 1 - \|u_i - v_j\|_2^2), \qquad (2)$$

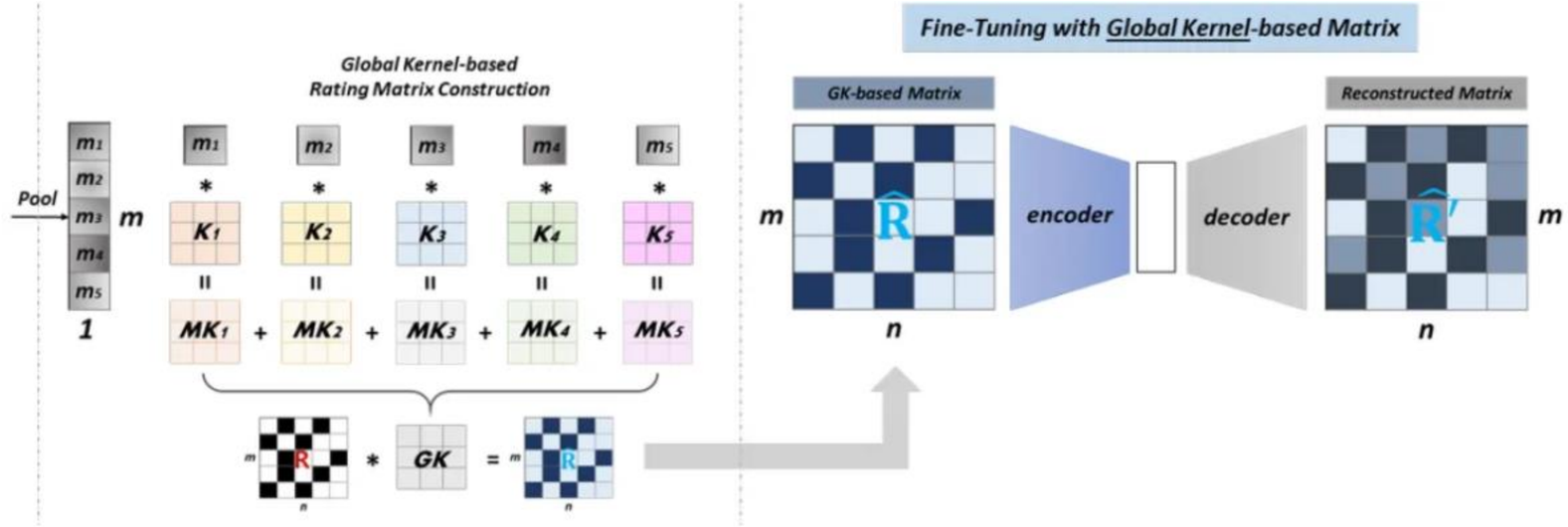
u_i, v_j : U(User)에 속한 벡터, V(Item)에 속한 벡터(둘 사이의 거리가 멀어질수록 0, 동일하면 1)

```
# local kernel 구현 공식
def local_kernel(u, v):
    dist = torch.norm(u - v, p=2, dim=2)
    hat = torch.clamp(1. - dist**2, min=0.)
    return hat
```

Local Kernelised Weight

Glocal-K

02 Fine-tuning : global kernel(GK)을 이용해 최종 rating matrix 생성



1) 앞서 생성된
Pre-training rating matrix

$$train_r = \begin{bmatrix} 5 & 3.5 & 3 & 4 \\ 3 & 4 & 4.2 & 5 \\ 3.8 & 2 & 5 & 3 \end{bmatrix}$$

- 아이템 A의 평균 평점: $\frac{5+3.5+3+4}{4} = 3.88$
- 아이템 B의 평균 평점: $\frac{3+4+4.2+5}{4} = 4.05$
- 아이템 C의 평균 평점: $\frac{3.8+2+5+3}{4} = 3.45$

```
def global_kernel(self, input, gk_size, dot_scale):
  # 각 아이템 평균 구하기
  avg_pooling = torch.mean(input, dim=1)
  avg_pooling = avg_pooling.view(1, -1)
  # (각 아이템 평균 X conv kernel) X 값 스케일(dot scale)
  gk = torch.matmul(avg_pooling, self.conv_kernel) * dot_scale
  gk = gk.view(1, 1, gk_size, gk_size)

  return gk
```

$$\text{conv\_kernel} = \begin{bmatrix} 0.1 & 0.2 & 0.3 & 0.4 & 0.5 & 0.6 & 0.7 & 0.8 & 0.9 \\ 0.9 & 0.8 & 0.7 & 0.6 & 0.5 & 0.4 & 0.3 & 0.2 & 0.1 \\ 0.2 & 0.3 & 0.4 & 0.5 & 0.6 & 0.7 & 0.8 & 0.9 & 0.1 \end{bmatrix}$$

$$GK = \begin{bmatrix} 0.4723 & 0.5051 & 0.5379 \\ 0.5707 & 0.6035 & 0.6363 \\ 0.6691 & 0.7019 & 0.4242 \end{bmatrix}$$

2) 각 아이템의 평균 평점과 conv_kernel 곱하기 (conv_kernel은 랜덤으로 초기화)

3) 최종 GK에 dot_scale 적용 (dot_scale은 하이퍼파라미터)

4) 사전 훈련된 평점 행렬에 GK 커널을 컨볼루션 연산을 수행하여 최종적으로 예측된 평점 행렬 생성

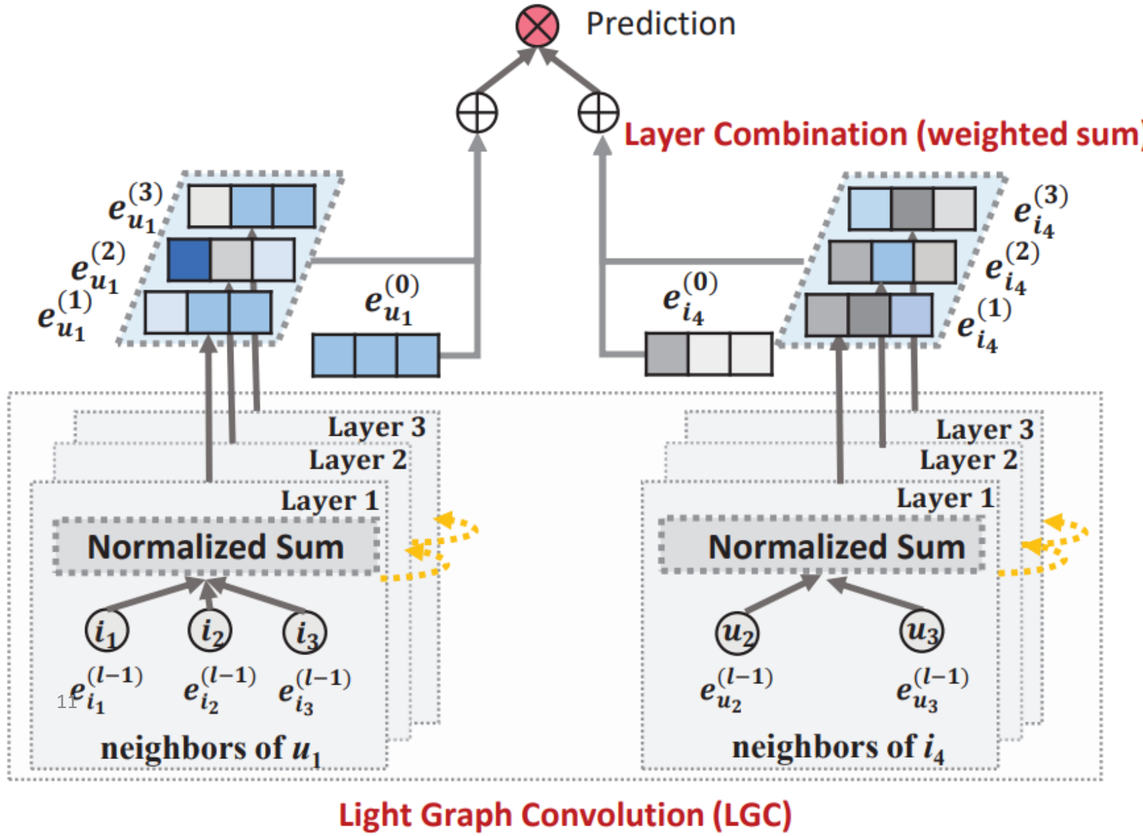
$$\hat{R} = R \otimes GK$$

BOAZ

LightGCN

Model Architecture

Light Graph Convolution 연산:



Light Graph Convolution (LGC)

최종 임베딩 벡터는 다음과 같이 각 layer로부터 얻은 임베딩들의 선형결합으로 표현할 수 있다:

$$\mathbf{e}_u = \sum_{k=0}^{K} \alpha_k \mathbf{e}_u^{(k)}; \quad \mathbf{e}_i = \sum_{k=0}^{K} \alpha_k \mathbf{e}_i^{(k)},$$

LightGCN

Model Architecture

- Model prediction은 ranking score로 다음과 같다:

$$\hat{y}_{ui} = \mathbf{e}_u^T \mathbf{e}_i,$$

where $\mathbf{e}_u = \sum_{k=0}^{K} \alpha_k \mathbf{e}_u^{(k)}; \quad \mathbf{e}_i = \sum_{k=0}^{K} \alpha_k \mathbf{e}_i^{(k)},$

- 최종 임베딩 벡터의 행렬 표현:

$$\begin{aligned}\mathbf{E} &= \alpha_0 \mathbf{E}^{(0)} + \alpha_1 \mathbf{E}^{(1)} + \alpha_2 \mathbf{E}^{(2)} + \ldots + \alpha_K \mathbf{E}^{(K)} \\ &= \alpha_0 \mathbf{E}^{(0)} + \alpha_1 \tilde{\mathbf{A}} \mathbf{E}^{(0)} + \alpha_2 \tilde{\mathbf{A}}^2 \mathbf{E}^{(0)} + \ldots + \alpha_K \tilde{\mathbf{A}}^K \mathbf{E}^{(0)},\end{aligned}$$

where $\tilde{\mathbf{A}} = \mathbf{D}^{-\frac{1}{2}} \mathbf{A} \mathbf{D}^{-\frac{1}{2}}$ is the symmetrically normalized matrix.

BOAZ

LightGCN

Model Architecture

전체 데이터의 user의 수를 M, item의 수를 N이라고 하자.User-Item interaction 행렬 $\mathbf{R} \in \mathbb{R}^{M \times N}$ 의 각 원소는 학습 데이터의 user i가 학습데이터의 item j 에 리뷰를 남겼을 경우, $\{\mathbf{R}\}_{ij} \in \mathbb{R}^{M \times N}$ = 1 이고, 나머지는 0이다.

```
def _make_user_item_interaction(self):
  R = sp.dok_matrix((self.n_users, self.n_items), dtype=np.float32) # remark. Initialize to train-test dataset
  R[self.train_df[self.col_user], self.train_df[self.col_item]] = 1.0 # store the information about train data
  return R # stored elements in Dictionary Of Keys format
```

인접 행렬(adjacency matrix) $\mathbf{A} = \begin{pmatrix} \mathbf{0} & \mathbf{R} \\ \mathbf{R}^{\mathbf{T}} & \mathbf{0} \end{pmatrix} \in \mathbb{R}^{(M+N)\times(M+N)}$

```
# Step 1: Construct adjacency matrix
R_csr = self.R.tocsr()  # Convert R to CSR format for faster operations
adj_mat = sp.vstack([
    sp.hstack([sp.csr_matrix((self.n_users, self.n_users)), R_csr]),
    sp.hstack([R_csr.T, sp.csr_matrix((self.n_items, self.n_items))])
]).tocsr()  # Convert full adjacency matrix to CSR format
```

BOAZ

LightGCN

Model Architecture

정규화된 인접 행렬(normalized adjacency matrix) : $\tilde{\mathbf{A}} = \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$, $\mathbf{D}$는 각 대각원소 d_{ii} 는 i번째 행 벡터에서 0이 아닌 원소의 수인 대각 행렬, 즉 $\mathbf{D} = diag(d_{11}, \ldots, d_{M+N})$

대각 행렬 계산

```
# Step 2: Calculate row sums and inverse square root diagonal values
rowsum = np.array(adj_mat.sum(axis=1)).flatten()  # Sum rows
d_inv_sqrt = np.clip(rowsum, 1e-9, None) ** -0.5  # Calculate inverse square root safely
```

제한된 메모리 자원내에서의 계산을 위해 chunk 단위로 쪼개어 행렬 연산을 수행

```
for i in tqdm(range(0, adj_mat.shape[0], chunk_size)):
    start, end = i, min(i + chunk_size, adj_mat.shape[0])

    # Process in sub-chunks to reduce memory usage
    chunk_rows = []
    for j in range(start, end, sub_chunk_size):
        sub_start, sub_end = j, min(j + sub_chunk_size, end)
        d_chunk = sp.diags(d_inv_sqrt[sub_start:sub_end])  # Diagonal matrix for the sub-chunk
        normalized_sub_chunk = d_chunk.dot(adj_mat[sub_start:sub_end]).dot(sp.diags(d_inv_sqrt))
        chunk_rows.append(normalized_sub_chunk)

    # Concatenate sub-chunks and save to disk
    normalized_chunk = sp.vstack(chunk_rows).tocsr()
    sp.save_npz(f"{norm_adj_mat_path}/chunk_{i}.npz", normalized_chunk)

# Step 4: Combine all chunks into the final matrix
print("Combining chunks into the final normalized adjacency matrix...")
chunk_files = sorted(glob.glob(f"{norm_adj_mat_path}/chunk_*.npz"))
norm_adj_mat = sp.vstack([sp.load_npz(f) for f in chunk_files]).tocsr()
```

BOAZ

LightGCN

Model Architecture

1. Xavier Initialization를 통한 임베딩 벡터 $\mathbf{E}^{(0)} = [\mathbf{E}_u^{(0)}, \mathbf{E}_i^{(0)}] \in \mathbb{R}^{(M+N)\times H}$ 초기화 (H는 임베딩 공간의 차원)

```
E_0 = nn.Embedding(data.n_users + data.n_items, 32)
nn.init.xavier_uniform_(E_0.weight)
E_0.weight = nn.Parameter(E_0.weight)
```

2. 각 $k = 1, \ldots, K$마다, layer의 임베딩 벡터를 $\mathbf{E}^{(k)} = \tilde{\mathbf{A}}\mathbf{E}^{(k-1)}$ 를 통해 계산

- all_layer_embedding: $[\mathbf{E}^{(0)}, \mathbf{E}^{(1)}, \ldots, \mathbf{E}^{(K)}]$ where K is the number of layers and $\mathbf{E}^{(k)} = \tilde{\mathbf{A}}\mathbf{E}^{(k-1)}$ for $k = 1, \ldots, K$

```
all_layer_embedding = [E_0.weight]
E_k = E_0.weight

for k in range(K):
  E_k = torch.sparse.mm(norm_adj_mat_sparse_tensor, E_k)
  all_layer_embedding.append(E_k)

all_layer_embedding = torch.stack(all_layer_embedding)
```

BOAZ

LightGCN

Model Architecture

3. 최종 임베딩 벡터 계산:

- $\mathbf{e}_u = \sum_{k=0}^{K} \alpha_k \mathbf{e}_u^{(k)} = \frac{1}{K+1} \sum_{k=0}^{K} \mathbf{e}_u^{(k)} = ave(\{\mathbf{e}_u^{(k)}\}_{k=0}^{K})$
- $\mathbf{e}_i = \sum_{k=0}^{K} \alpha_k \mathbf{e}_i^{(k)} = \frac{1}{K+1} \sum_{k=0}^{K} \mathbf{e}_i^{(k)} = ave(\{\mathbf{e}_i^{(k)}\}_{k=0}^{K})$

즉, 각 layer의 임베딩 벡터의 평균을 취한 후, user-item 임베딩 벡터 각각으로 분할

```
mean_layer_embedding = torch.mean(all_layer_embedding, axis = 0)
final_user_Embed, final_item_Embed = torch.split(mean_layer_embedding, [n_users, n_items])
```

BPR Loss

$$L_{BPR} = \sum_{u=1}^{M} \sum_{i \in N_u} \left(\sum_{j \notin N_u} \ln \sigma(\hat{y}_{ui} - \hat{y}_{uj}) \right) + \lambda \| \mathbf{E}_0 \|^2$$

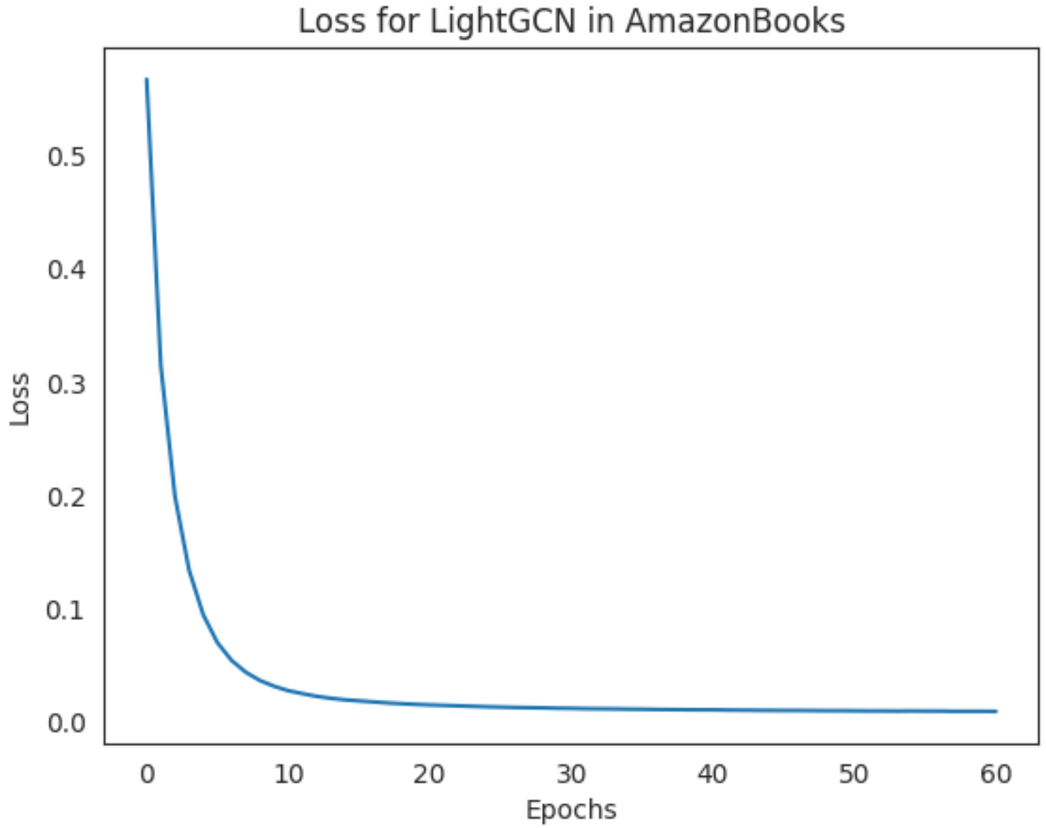
- $\hat{y}_{ui} = \mathbf{e}_u^T \mathbf{e}_i$: positive sample for user u (훈련 데이터에서 실제 리뷰가 있는 아이템 대상)
- $\hat{y}_{uj}$: negative sample for user u (유저가 리뷰하지 않은 아이템 중에 랜덤으로 샘플링)
- $\sigma(\cdot)$: softmax function

BOAZ

LightGCN

Model Architecture

- Amazon Book 데이터로부터 2,526,568명의 사용자, 1,917,359개의 책 데이터를 추출하여 모델 학습:



(평가지표)

```
Recall@10:  0.00011242
Precision@10:  1.653e-05
Ndcg@10:  4.648e-05
Map@10:  2.251e-05
```

평가지표

1. Precision@k
- 상위 k개의 추천 중 사용자가 실제로 좋아한 항목의 비율.
- 예: k=10일 때, 상위 10개 추천 중 3개가 사용자의 선호 항목이라면, Precision@10 = 0.3.

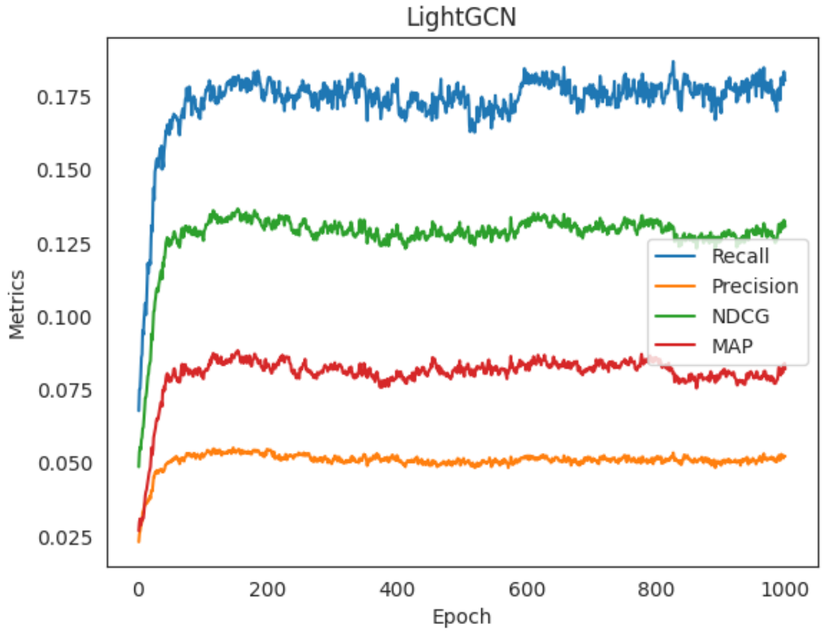
2. Recall@k
- 사용자가 좋아한 항목 중 상위 k개의 추천에 포함된 항목의 비율.
- 예: 사용자가 좋아한 항목이 5개이고, 상위 10개 추천에 2개가 포함되었다면, Recall@10 = 0.4.

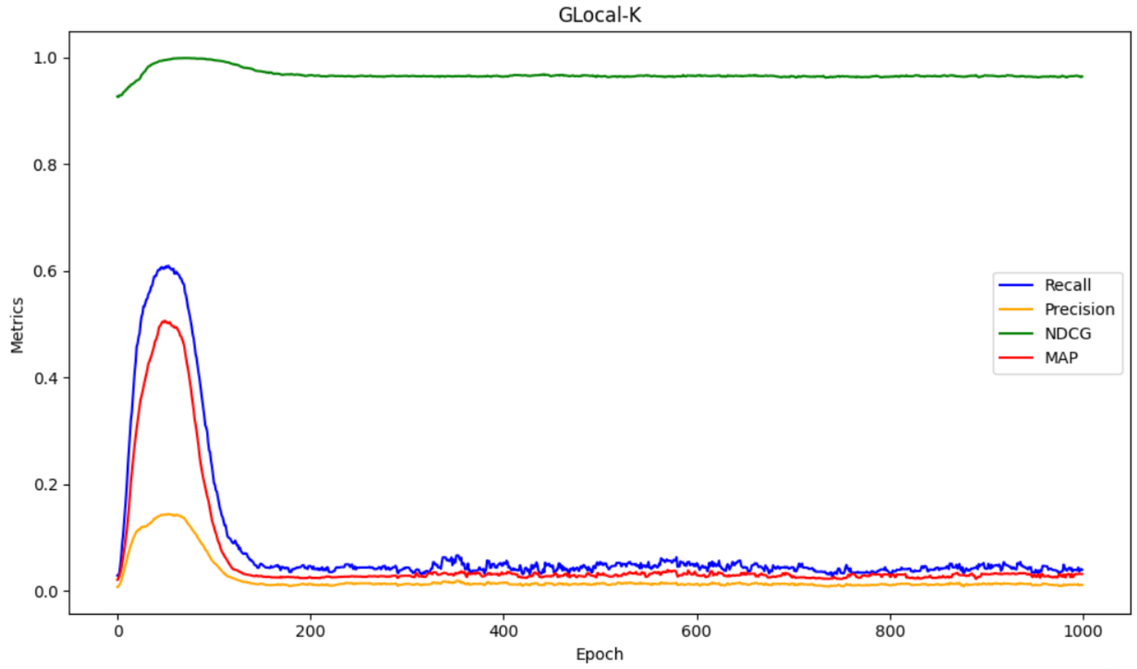
3. NDCG@k
- 상위 k개의 추천 리스트의 순서를 고려해 평가.
- 예: 상위 10개 중 가장 선호도가 높은 항목이 1위에 있으면 더 높은 점수를 부여.

4. MAP@k
- 각 사용자에 대해 상위 k개의 Precision을 계산하고 평균을 구함.
- 모든 사용자가 상위 k개의 추천에 대해 만족했는지 확인.

성능비교

	LightGCN	Glocal-K
Recall	0.1804	0.6093
Precision	0.0523	0.1443
NDCG	0.1306	0.9985
MAP	0.1306	0.5065





분석 23기 미니프로젝트2

감사합니다

도서 데이터 기반 추천시스템 구현

분석 23기 윤왕규, 김경민, 백다은, 오영민