Chapter 7. Recommender Systems

Outline

A. Recommender System

a. Architectures

b. Evaluation metrics

c. Loss functions

d. Representative models

A. Recommender System

1) 추천 시스템 개요

(1) Candidate Generation

Huge corpus로부터 시작해서 작은 candidate의 subset을 생성
(Starts from a huge corpus and generates a smaller subset of candidates)

거대한 사이즈의 코퍼스가 주어졌을 때, 모델은 쿼리들을 빠르게 평가해야 함.
(Given the enormous size of the corpus, the model needs to evaluate queries quickly)

(2) Scoring

유저에게 보여주기 위한 아이템들의 집합을 고르기 위해 후보자들의 점수를 매기고 랭킹을 매긴다.
(Another model scores and ranks the candidates to select the set of items to display to the user.)

모델은 비교적 작은 항목 부분을 평가하기 때문에 시스템은 추가 쿼리에 의존하는 보다 정확한 모델을 사용할 수 있다.
(Since this model evaluates a relatively small subset of items, the system can use a more precise model relying on additional queries.)

(3) Re-ranking

Final ranking을 위해 추가적인 조건들을 확인, score에 근거해 유저가 싫어하는 아이템들 제거

2) Candidate Generation

두 타입 모두 임베딩 공간에 $E = R^d$ 에 query(또는 context)와 item을 임베딩 벡터로 mapping. 임베딩 공간은 low-dimensional하다. Item과 query의 latent structure를 포착한다. 같은 사용자가 본 유튜브 영상 같은 것은 임베딩 공간에서 ending point가 가깝다. 여기서 **closeness**는 similarity로 정의된다.

(1) Similarity(Cosine, Dot-product, Euclidian Distance)

Given two vectors q and x:

Cosine: $s(q,x) = \cos(q,x)$

Dot Product: $s(q,x) = \langle q,x \rangle = \Sigma_{i=1}^{d} q_i x_i = \|x\|\|q\|\cos(q,x)$

Euclidean distance: $s(q,x) = \|q-x\| = [\Sigma_{i=1}^{d}(q_i - x_i)^2]^{\frac{1}{2}}$.

Cos 유사도는 두 벡터의 각도차에 집중한다. 반면 내적은 각도와 더불어 벡터의 크기도 고려한다. 유클리디언 거리는 반대로 오직 거리만들 이용한다. 만약 두 벡터 임베딩들이 Normalization되어 있으면 유클리디언 거리 제곱은 상수까지 내적(및 코사인)과 일치가게 된다. 따라서 수식이 아래처럼 바뀐다.

$$\frac{1}{2}\|q-x\|^2 = 1 - \langle q,x \rangle$$

Depending on the similarity measure used, the ranking of the items can be different.

Dot-Product

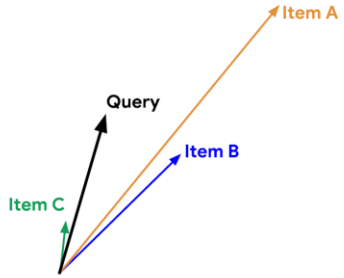
- Query: Item A > Item B > Item C

Cosine

- Query: Item C > Item A > Item B

Euclidean Distance

- Query: Item B > Item C > Item A



코사인 유사도보다는 내적이 좀 더 임베딩의 norm에 sensitive하다 (크기까지 고려하기 때문). Training set에서 빈도수가 높은 아이템은 large norm 임베딩을 가지는 경향성 있다.

또한, 빈도수가 적은 아이템은 training중 업데이트가 거의 일어나지 않는데, 만약 이 아이템들이 large norm을 가지고 있을 경우 시스템은 이 rare item들을 연관성 있는 아이템들보다 더 많이 추천해 줄 수 있다.

(2) Content based Filtering

사용자와 연관된 아이템을 추천

similarity metric을 고른다 -> 각각의 후보 아이템들의 score를 similarity를 통해 매긴다. 내적 값이 클수록 더 common feature를 나타내게 되고 이는 더 큰 similarity임을 말해준다.

장점

a. 구체적인 유저에 대해 아이템을 추천해주기 때문에 모델이 다른 유저들에 대한 데이터를 필요로 하지 않는다.

b. 유저와 관련된 specific interest를 포착한다.

단점

a. 아이템의 feature들이 수작업으로 만들어짐(hand-engineered) -> 방대한 도메인 지식을 필요로 함.

b. 다른 유저들의 아이템을 추천하지 못한다는 제한이 있음. (도메인이 있는 구체적인 유저에 대한 아이템만을 추천)

(3) Collaborative filtering

a. 유저와 아이템의 유사도를 동시에 측정

b. Serendipitous recommendations이라고도 불림

-> 유저 A의 아이템을 추천해 줄 때 비슷한 유저인 B의 관심사를 바탕으로 추천해주기 때문

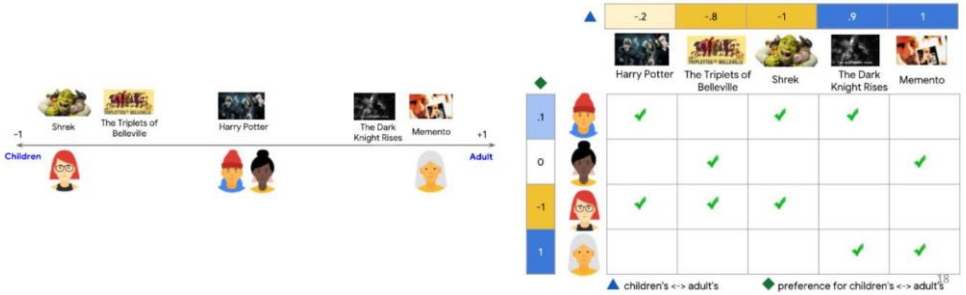
c. 임베딩들은 automatically 학습됨. Hand-engineering feature들에 의존 X

d. Example of collaborative filtering – Movie recommendation

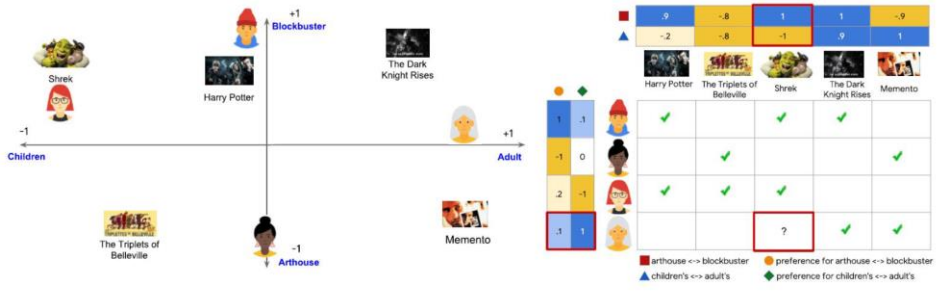
- row = user, column = item
- feedback은 두 카테고리 중 하나로 분류됨
- Explicit: : users specify how much they liked a particular movie by providing a numerical rating.
- Implicit: : if a user watches a movie, the system infers that the user is interested.
- feedback 행렬은 binary하다.
- 유저가 홈페이지를 방문했을 때 시스템이 영화를 1)유저가 과거에 좋아했던 영화들의 similarity와 2) 비슷한 유저가 좋아하는 영화들을 기반으로 추천해준다.

1D Embedding

- 모든 영화가 scalar [-1,1], 각각의 값들을 유저에게 배치



2D Embedding



학습

a. 모델이 임베딩을 학습할 때는 영화들의 임베딩 벡터는 고정되어 있다고 가정한다.

i) 모델은 사용자가 선호도를 가장 잘 설명할 수 있도록 임베딩 벡터를 학습한다.

ii) 결과적으로, 유저의 임베딩과 선호도의 유사도는 가까워진다.

b. 유저의 임베딩이 고정된 경우

i)모델은 영화 임베딩이 feedback matrix를 가장 잘 설명할 수 있도록 학습한다.

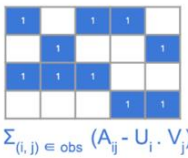
c. 비슷한 사용자가 좋아하는 영화의 임베딩은 임베딩 공간에서 가까워진다.

(4) Matrix Factorization

가장 간단한 임베딩 모델이다. M x N 사이즈의 feedback matrix가 주어졌을 때, 유저의 임베딩 행렬은 U $\in R^{m \times d}$, 아이템 임베딩 행렬은 V $\in R^{n \times d}$

Objective function: 관측되는 모든 entry의 모든 쌍의 에러의 합을 최소화

Observed Only MF

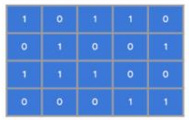


$\Sigma_{(i,j) \in obs} (A_{ij} - U_i \cdot V_j)^2$

$$\min_{U \in \mathbb{R}^{m \times d}, V \in \mathbb{R}^{n \times d}} \Sigma_{(i,j) \in obs} \left(A_{ij} - \langle U_i, V_j \rangle\right)^2$$

(1)

SVD



$|A - UV^T|_F^2$

$= \Sigma_{(i,j)} (A_{ij} - U_i \cdot V_j)^2$

$$\min_{U \in \mathbb{R}^{m \times d}, V \in \mathbb{R}^{n \times d}} \Sigma_{(i,j) \in obs} \left(A_{ij} - \langle U_i, V_j \rangle\right)^2$$

(2)

Left figure.

관측되는 쌍 (i, j)만 합한다 = feedback 행렬의 non-zero값들만 합한다.

-> Bad idea: A matrix of all 1s will have a minimal loss and produce a model that can't make effective recommendations and that generalizes poorly.

Right figure.

관측되지 않는 값들은 zero로 두고 행렬의 모든 entry를 합한다.

-> Frobenius distance: 행렬 norm between A and approximation UV^T

-> Singular value decomposition(SVD)를 이용해서 풀 수 있음

-> 하지만, 실제로 매우 Sparse하기 때문에 SVD는 좋은 solution이 아님.

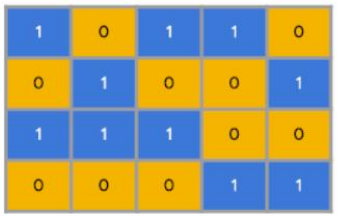
∴ Weighted Matrix Factorization은 두개의 합 형태로 분해시킴.

-> Sum over observed entries

-> Sum over Unobserved entries (treated as zeros)

$$\min_{U\in\mathbb{R}^{m\times d},V\in\mathbb{R}^{n\times d}} w_{i,j}\,\Sigma_{(i,j)\in obs}\left(A_{ij}-\langle U_i,V_j\rangle\right)^2+w_0\Sigma_{(i,j)\notin obs}\left(\langle U_i,V_j\rangle\right)^2$$

Weighted MF



$$\Sigma_{(i,j)\in obs}\,(A_{ij}-U_i\cdot V_j)^2+\;w_0\,\Sigma_{(i,j)\notin obs}\,(0-U_i\cdot V_j)^2$$

(5) Minimizing Optimization

SGD: Stochastic Gradient Descent = Loss를 최소화하는 가장 generic한 방법 (Singular function 전용)

WALS: Weighted Alternating Least Squared = 특정 objective에 대해 특화됨 (Quadratic function 전용)

Type	Advantage	Disadvantage
SGD	• Very flexible (can use other loss functions). • Can be parallelized.	• Slower (does not converge as quickly) • Harder to handle the unobserved entries (need to use negative sampling).
WALS	• Can be parallelized. • Converges faster than SGD. • Easier to handle unobserved entries.	• Reliant on Loss Squares only.

(6) Collaborative Filtering

장점

a. No domain knowledge: 임베딩이 automatically learned

b. Serendipity: 유저가 새로운 관심사를 찾게 도와주며, 모델은 유저의 관심사를 몰라도 비슷한 유저의 관심사를 통해 아이템을 추천해줄 수 있다.

c. Great Starting Point

- 행렬을 factorization하기 위해 오직 feedback 행렬만을 필요로 함.
- contextual features가 불필요
- 실제로, multiple candidate generator로 사용될 수 있음.

단점

a. Cannot handle fresh items

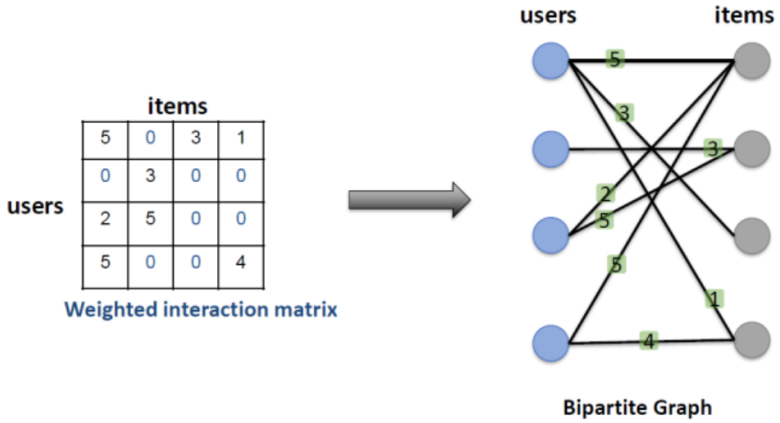
- 주어진 유저-아이템 쌍에 대해서만 유사도를 구해 추천
- 만약 학습 중 한 아이템이 보이지 않는다면 시스템은 임베딩을 만들 수 없고, 이 아이템에 대해 쿼리할 수 없다. (Cold-Start Problem)
- 물론 Projection in WALS나 Heuristics to generate embeddings of fresh items 를 통해 어느 정도 이 문제를 완화 가능

b. Hard to include side feature for query/item

- Side features는 쿼리의 또 다른 feature가, 또는 아이템의 ID가 될 수 있음
- WALS에 사이트 feature를 넣는 게 어렵더라도, generalization을 통해 넣을 수 있음.

B. GNN Recommendation System

(1) 정의



Weighted Interaction Matrix를 인접행렬 (Adjacency Matrix)로 간주

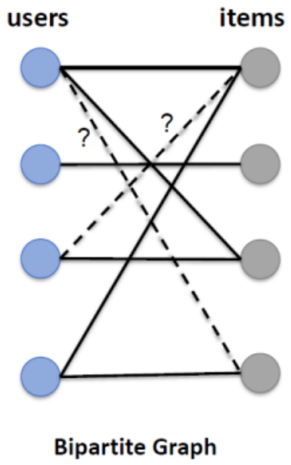
Weighted Interaction Matrix <-> Bipartite Graph with Weight

Given

- Past user-item interactions

Task

- Predict new items each user will interact in the future.
- Can be cast as link prediction problem.
  - Predict new user-item interaction edges given the past edges.



User-Item interaction이 주어졌을 때, Top-k개의 아이템을 출력. 하나의 아이템을 추천해주는 것이 아닌 여러 개의 아이템을 추천

(Positive item = Items that user will interact with in the future)

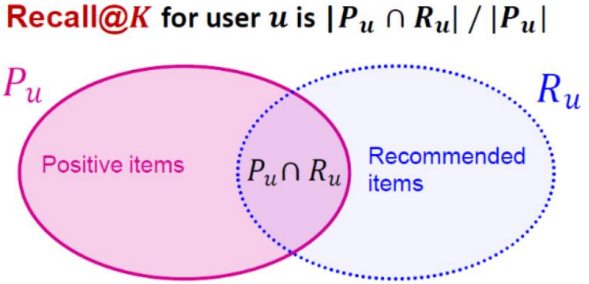
(2) Evaluation Metric

a. Recall@K

i. Pu = Positive item set

ii. Ru = 모델이 추천해준 item set

iii. |Ru| = K, Top-K recommendation



b. Accuracy

i. Accuracy = $\frac{TP+TN}{TP+FP+FN+TN}$

ii. 장점: 전체 중 정답으로 나온 비율을 직관적으로 보기 편함

iii. 단점: Symmetric한 dataset에서만 좋게 나옴

c. Precision

i. Precision (P) = $\frac{TP}{TP+FP}$

ii. 장점: False Positive 비율이 중요하면 좋은 방법임.

iii. 단점: False Negative 비율 측정 불가

d. Recall (R)

i. Recall = $\frac{TP}{TP+FN}$

ii. 장점: False Negative 비율이 중요하면 좋은 방법임.

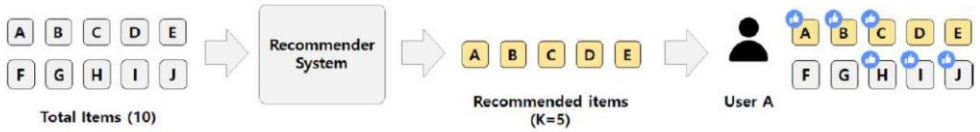
iii. 단점: False Positive 비율 측정 불가

e. F1 Score

i. F1 score = $\frac{2}{\frac{1}{P}+\frac{1}{R}} = \frac{2PR}{P+R}$

ii. 장점: FN과 FP 모두를 고려. Uneven한 class distribution에 좀 더 효과적인 방법

iii. 단점: 직관적이지 않음



$$Precision@5 = \frac{3}{5} = 0.6 \qquad Recall@5 = \frac{3}{6} = 0.5$$

* Precision@5 = 선택/추천 = 3/5

* Recall@5 = 추천/선택 all = 3/6

f. AUC ROC

i. Area Under Curve (AUC): the area under Receiver Operating Characteristic (ROC) curve (TPR against FPR). Normalized Discounted Cumulative Gain

ii. Receiver Operating Characteristic Curve (ROC)

- Sensitivity(민감도) + Specificity(특이도)

표기는 이후에 TP, TN, FP, FN으로 함		예측값 (Positive / Negative)	
		양성(1)	음성(0)
실제값(진단결과) (Ture / False)	양성(1)	①True Positive(TP)	②False Negative(FN)
	음성(0)	③False Positive(FP)	④True Negative(TN)

- TP: 양성이라고 예측 & 진단도 양성 = 일치(True)
- FN: 음성이라고 예측 & 진단은 양성 = 불일치(False)
- FP: 양성이라고 예측 & 진단은 음성 = 불일치(False)
- TN: 음성이라고 예측 & 진단도 음성 = 일치(True)

Sensitivity (민감도)는 실제 Positive 대비 정답으로 맞춘 True Positive 비율이다.

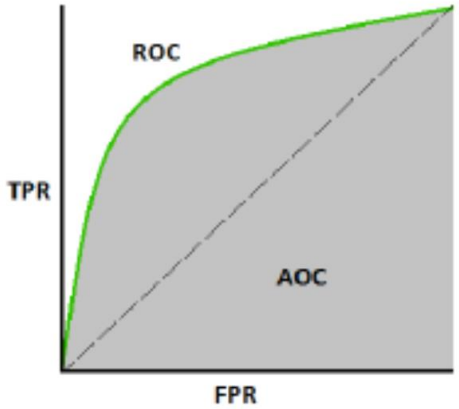
True Positive Rate (TPR) = $\frac{TP}{TP+FN}$ -> 민감도는 클수록 Good

Positive으로 예측해서 맞췄거나 음성으로 예측했는데 틀린 경우를 합친 것이 실제 양성인 값이다.

Specificity (특이도)는 실제 Negative 대비 맞춘 Negative 비율이다.

False Negative Rate (FNR) = $\frac{FP}{FP+TN}$ -> 특이도는 클수록 Good

Negative로 예측해서 맞췄거나 양성으로 예측했는데 틀린 경우를 합친 것이 실제 음성인 값이다.



ROC Curve는 좌상단으로 붙어 있을수록 더 좋은 이진 분류기이다.

g. Mean Reciprocal Rank (MRR)

i. Mean Rank의 역수와 수식이 유사

ii. MRR은 사용자의 질문에 대한 정답이 시스템이 제공하는 결과 목록의 어디에 위치하는지를 평가하는 데 사용된다. MRR은 전체 질의 집합에 대한 단일 질의의 최초 정확한 답변의 역순위(즉, 순위의 역수)의 평균값으로 정의된다.

iii. Ex) 사용자가 어떤 질문을 하고 시스템이 그 질문에 대한 답변 목록을 제공하는데, 첫 번째로 올바른 답변이 나오면 그 질의의 reciprocal rank은 1 이다. 만약 올바른 답변이 두 번째 위치에 있다면 reciprocal rank은 1/2 = 0.5가 된다. 따라서, 여러 질의에 대한 이러한 reciprocal ranks의 평균값을 취하면 MRR이 된다.

$$MRR = \frac{1}{|U|}\sum_{u \in U}\frac{1}{r_u}$$

iv. 장점: Low Computational Cost. 리스트에서 가장 관련이 있는 첫번째 요소에 집중한다. 따라서 유저를 위한 가장 좋은 item을 뽑는데 적합하다.

v. 단점: 리스트에 남은 item들을 평가하지 못한다. 유저가 여러가지 아이템을 원할 경우 부적합하다.

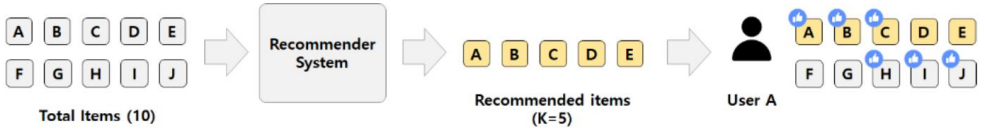
h. Precision, Hits, Recall

i. Precision, Hits, Recall

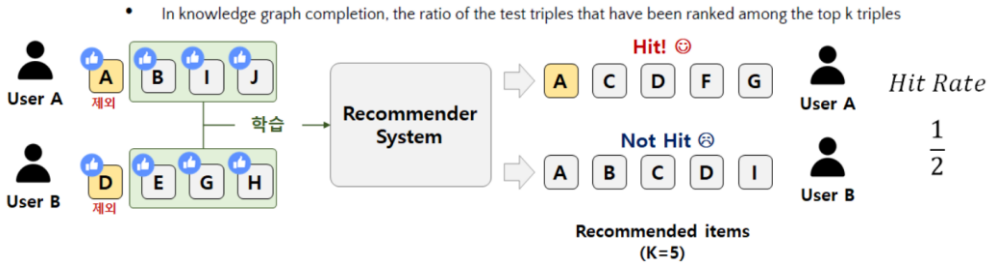
ii. Precision@K = 내가 추천한 아이템 K개 중에 실제 사용자가 관심이 있는 아이템의 비율

iii. Recall@K = 유저가 관심있는 모든 아이템 중에서 내가 추천한 아이템 K개가 얼마나 포함되어 있는지 비율

iv. Hits@K = 전체 유저 수 대비 적중한 유저 수 (적중률)



$$Precision@5 = \frac{3}{5} = 0.6 \qquad Recall@5 = \frac{3}{6} = 0.5$$



Ground truth for two triples

Jack born_in Italy
Jack friend_with Thomas

Top 5 results for two triples

s	p	o	score	rank	
Jack	born_in	Ireland	0.789	1	
Jack	born_in	Italy	0.753	2	*
Jack	born_in	Germany	0.695	3	
Jack	born_in	China	0.456	4	
Jack	born_in	Thomas	0.234	5	

s	p	o	score	rank	
Jack	friend_with	Thomas	0.901	1	*
Jack	friend_with	China	0.345	2	
Jack	friend_with	Italy	0.293	3	
Jack	friend_with	Ireland	0.201	4	
Jack	friend_with	Germany	0.156	5	

- Hits@3 = 위의 top 3안에 정답이 둘 다 들어가 있음. 따라서 1/2 + 1/2 = 1
- Hits@1 = top 1안에 정답이 아래만 있음. 따라서 0/2 + 1/2 = 1/2

i. Mean Average Precision (MAP)

i. MAP는 여러 질의에 대한 평균적인 precision 값
ii. Precision은 검색 결과 중 올바른 항목의 비율을 나타내는 측정치이다. 정보 검색에서는 특정 질의에 대해 여러 개의 결과가 반환되므로 각 질의에 대한 평균 precision을 계산할 때 "average precision"이라는 측정치를 사용한다.
iii. Average Precision(AP)는 특정 질의에 대한 precision의 평균값으로, 모든 관련 문서가 검색되는 순서대로의 precision 값을 평균 낸 것이다. MAP는 여러 질의에 대한 각각의 AP 값들의 평균이다.

$$\boldsymbol{MAP@k} = \frac{1}{|U|}\Sigma_{u\in U} AP@k(u)$$

$$\boldsymbol{AP@k(u)} = \frac{1}{m}\Sigma_{i=1}^{k} Precision@i \cdot \mathbb{I}[kth\ is\ true]$$ where m is the number of true candidates

R: relevant item, NR: non-relevant item		
User	User 1	User 2
1	R	NR
2	NR	NR
3	R	NR
4	R	R
5	NR	R
AP@5	(1/1+2/3+3/4)/3=0.8	(1/4+2/5)/2=0.325
MAP@5	(0.8+0.325)/2 = 0.56	

Ex) 정답이 등장한 부분에서 정답 개수를 전체 개수로 나눔. 모두 더해 평균 냄.

- User1: $\frac{1}{3}(\frac{1}{1} + \frac{2}{3} + \frac{3}{4})$
- User2: $\frac{1}{2}(\frac{1}{4} + \frac{2}{5})$

iv. 장점: 리스트 안의 추천 아이템들의 랭킹을 handling할 수 있다. 또한, 추천 목록 중 높은 위치에서 발생하는 오류에 더 많은 가중치를 부여하여 업데이트 가능하다.
v. 단점: fine-grained numerical rating (미세 수치 평가)에는 적합하지 않고, Binary(Relevant / Irrelevant) 평가에 적합하다.

j. Normalized Discounted Cumulative Gain (NDCG@K)

이 평가지표는 1에 가까울수록 좋다. 순서별로 가중치 값을 다르게 적용하여 계산한다.

NDCG를 이해하기 위해서는 먼저 DCG를 알아야 한다.

Relevance (관련성)는 사용자가 특정 아이템과 얼마나 관련이 있는지를 나타내는 값이다. 이 값은 정해진 것이 아니고, 추천의 상황에 맞게 정해야 한다. 예를 들어, 신발 추천인 경우 사용자가 해당 신발을 얼마나 클릭했는지, 혹은 클릭 여부(Binary) 등 다양한 방법으로 선정할 수 있다.

* Cumulative Gain (CG)

CG는 추천한 아이템의 Relevance 합이다. 두 추천 모델이 순서에 관계없이 동일한 아이템 셋을 추천한 경우 두 모델의 CG는 같아진다. 즉, 순서를 고려하지 않은 값이다.

$$CG_K = \sum_{i=1}^{K} rel_i$$

* Discounted Cumulative Gain (DCG)

결과 목록에 표시되는 관련성 높은 항목에 높은 가중치를 부여하고, 낮은 순위의 항목에는 낮은 가중치를 부여합니다.

$$DOG = \sum_{i=1}^{n} \frac{rel_i}{log_2(i+1)}$$

reli는 i번째 결과의 관련성 점수이다.

* Ideal DCG (IDCG)

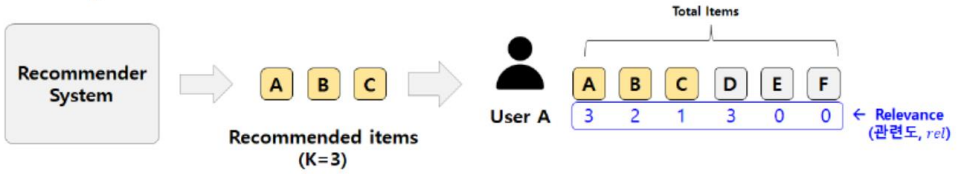
가능한 최대의 DCG 값이다. 즉, 결과 목록이 완벽하게 순위가 매겨졌을 때의 DCG 값을 나타낸다.

* NDCG (Normalized DCG)

DCG의 값을 IDCG 값으로 나누어 정규화한 값이다. 이렇게 하면 NDCG의 값은 0~1 사이의 값으로 제한된다.

$$DOG = \frac{DCG}{IDCG}$$

따라서 NDCG@K는 결과 목록의 순위와 관련성 둘 다를 고려하는 평가 지표로, 검색 결과의 상위 부분만 고려할 때 유용한다. 관련성이 높은 항목이 상위에 올라갈수록 NDCG@K 값이 커진다.



$$CG_3 = \sum_{i=1}^{K} rel_i = rel_1 + rel_2 + rel_3 = 3 + 2 + 1 = 6$$

$$DCG_3 = \sum_{i=1}^{K} \frac{rel_i}{log_2(i+1)} = \frac{3}{log_2(1+1)} + \frac{2}{log_2(2+1)} + \frac{1}{log_2(3+1)} = \frac{3}{1} + \frac{2}{1.58} + \frac{1}{2} = 4.78$$

$$IDCG_3 = \sum_{i=1}^{K} \frac{rel_i^{opt}}{log_2(i+1)} = \frac{3}{log_2(1+1)} + \frac{3}{log_2(2+1)} + \frac{2}{log_2(3+1)} = \frac{3}{1} + \frac{3}{1.58} + \frac{2}{2} = 5.89$$

$$NDCG_3 = \frac{DCG}{IDCG} = \frac{4.78}{5.89} = 0.81$$

결론적으로, NDCG@K는 가장 이상적인 추천 조합 대비 현재 모델의 추천 리스트가 얼마나 좋은지를 나타내는 지표이다. DCG는 K가 증가함에 따라 지속적으로 증가하는 반면, NDCG는 어느정도 K에 독립적이어서 어떤 K가 적절한지 판단이 가능하다. 또한 NDCG가 작은 스케일을 가지기 때문에 비교가 더 용이하다.

DCG vs NDCG

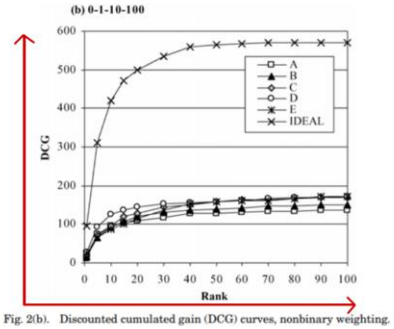


Fig. 2(b). Discounted cumulated gain (DCG) curves, nonbinary weighting.

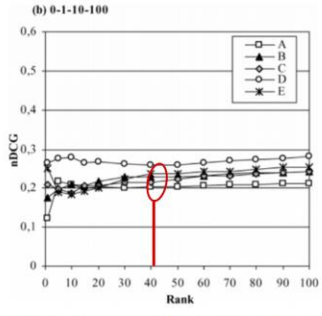
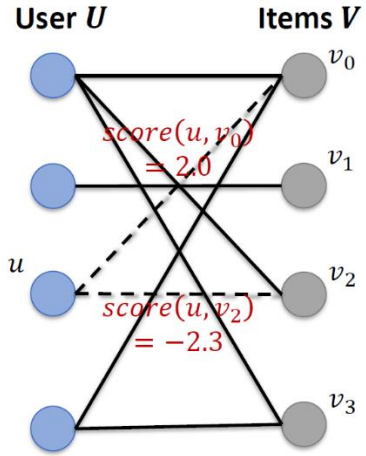


Fig. 4(b). Normalized discounted cumulated gain (nDCG) curves, nonbinary weighting.

C. Embedding-Based Model

(1) 정의

- Top-K개의 아이템을 얻기 위해, user-item interaction의 score function이 필요하다.
- 각 유저 $u \in U$, 아이템 $v \in V$의 실제 가치가 있는 scalar score(u,v)를 얻어야 한다.
- 주어진 사용자를 위해 가장 높은 점수를 가진 K개의 항목이 추천된다. (이미 상호 작용한 항목은 제외)



For K = 2, recommended items for user u

즉, User-Item interaction을 scoring하기 위해 Embedding-Based model을 사용한다. 유저(u)와 item(v)는 모두 D차원을 가지 임베딩으로 representation한다.

Let $f_\theta(\cdot,\cdot): R^D \times R^D \rightarrow R$ be a parameterized function

$score(u,v) \equiv f_\theta(u,v)$

(2) Training Objective

a. Encoder for user embedding $\{u\}_{u \in U}$

b. Encoder for item embedding $\{v\}_{v \in V}$

c. Score function f_θ

Training Objective

: 보이는, 주어진 (Seen) 유저-아이템 interaction에 대한 Recall@K값을 최대화하도록 파라미터를 최적화하자!!

MAX(Recall@K)

(3) Surrogate Loss function

Vanilla training objective (Training Recall@K)는 미분이 불가능하다. 따라서, gradient를 기반한 최적화 방식은 적용이 불가능하다.

이를 위해 두 가지 surrogate loss function을 제안한다.

a. Binary Loss

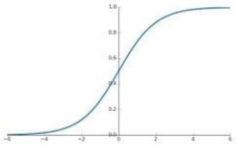
b. Bayesian Personalized Ranking(BPR) Loss

Surrogate Loss들은 미분가능하며 반드시 original training objective에 잘 일치해야 한다.

Binary Loss

- Define **positive/negative edges**
  - A set of **positive edges** E (i.e., observed/training user-item interactions)
  - A set of **negative edges** $E_{\text{neg}} = \{(u, v) | (u, v) \notin E, u \in U, v \in V\}$

- Define **sigmoid function** $\sigma(x) = \frac{1}{1+\exp(-x)}$
  - Maps real-valued scores into binary likelihood scores, i.e., in the range of [0,1].



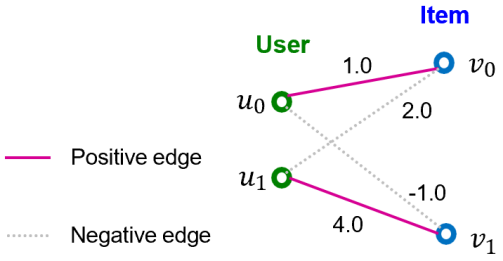
- **Binary loss**: Binary classification of **positive/negative** edges using $\sigma(f_\theta(\boldsymbol{u}, \boldsymbol{v}))$:

$$\underbrace{-\frac{1}{|E|}\Sigma_{(u,v)\in E}\log\left(\sigma\left(f_\theta(\boldsymbol{u},\boldsymbol{v})\right)\right)}_{} \underbrace{-\frac{1}{|E_{\text{neg}}|}\Sigma_{(u,v)\in E_{neg}}\log(1-\sigma(f_\theta(\boldsymbol{u},\boldsymbol{v})))}_{}$$

During training, these terms can be approximated using mini-batch of positive/negative edges

- Binary loss pushes the scores of **positive edges** higher than those of **negative edges**.
  - This aligns with the training recall metric since positive edges need to be recalled.

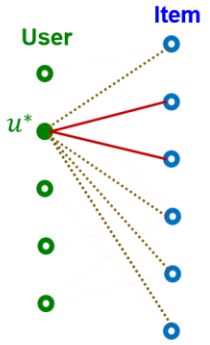
- **Key insight**: The binary loss is **non-personalized** in the sense that the **positive/negative edges are considered *across ALL users at once***.
- However, the recall metric is inherently **personalized (defined for each user)**
  - The non-personalized binary loss is overly-stringent for the personalized recall metric.
- Surrogate loss function should be defined in a **personalized** manner.
  - **For each user**, we want the scores of positive items to be higher than those of the negative items



BPR Loss

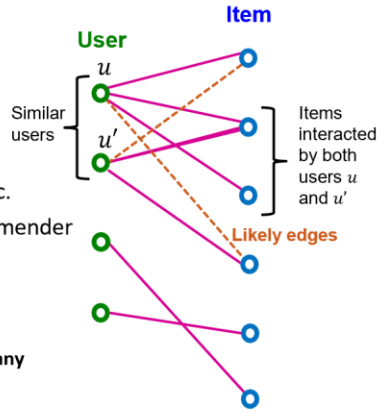
- **Bayesian Personalized Ranking (BPR) loss** is a personalized surrogate loss that aligns better with the recall@K metric.
- For each user $u^* \in U$, define the **rooted positive/negative edges** as
  - Positive edges rooted at u^*: $E(u^*) = \{(u^*, v) | (u^*, v) \in E\}$
  - Negative edges rooted at u^* : $E_{neg}(u^*) = \{(u^*, v) | (u^*, v) \in E_{neg}\}$
- Training objective:
  For each user $u^* \in U$, the scores of rooted positive edges $E(u^*)$ to be higher than those of rooted negative edges $E_{neg}(u^*)$.

$$\frac{1}{|U|} \Sigma_{u^* \in U} Loss(u^*)$$



Takeaways

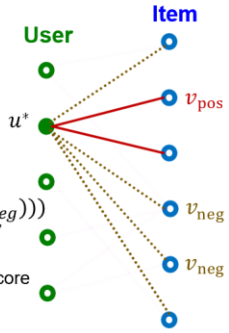
- Recall@K as a metric for personalized recommendation
- Embedding-based models
  - Three kinds of parameters to learn
    - **user encoder** to generate user embeddings
    - **item encoder** to generate item embeddings
    - **score function** to predict the user-item interaction likelihood.
- Surrogate loss functions to achieve the high recall metric.
- Embedding-based models have achieved SoTA in recommender systems.
- Why do they work so well?
  - **Underlying idea: Collaborative filtering**
    - Recommend items for a user by **collecting preferences of many other similar users.**
    - **Similar users tend to prefer similar items.**



- **Mini-batch training for the BPR loss:**
  - In each mini-batch, we sample a subset of users $U_{\text{mini}} \subset U$.
    - For each user $u^* \in U_{\text{mini}}$, we sample one positive item v_{pos} and a set of sampled negative items $V_{\text{neg}} = \{v_{\text{neg}}\}$.
- The mini-batch loss is computed as

$$\underbrace{\frac{1}{|U_{mini}|}}_{\text{Average over users in the mini-batch}} \Sigma_{u^* \in U_{mini}} \frac{1}{|V_{neg}|} \Sigma_{v_{neg} \in V_{neg}} \underbrace{-\log(\sigma(f_\theta(u^*, v_{pos})) - \sigma(f_\theta(u^*, v_{neg})))}_{\text{Encouraged to be positive for each user}}$$

= positive edge score is higher than negative edge score



Conventional Collaborative Filtering

- Conventional collaborative filtering model is based on **Matrix Factorization (MF)**.
  - Use shallow encoders for users and items
    - For every $u \in U$ and $v \in V$, we prepare shallow learnable embeddings $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^D$.
  - Score function for user u and item v is $f_\theta(u, v) \equiv u^T v$.

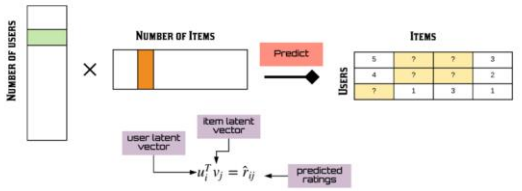
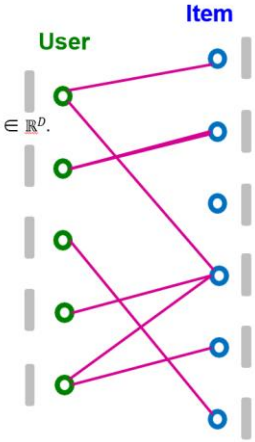


Image credit: Recommendation System Series Part 4: The 7 Variants of Matrix Factorization For Collaborative Filtering



* Limitation of Matrix Factorization (MF)

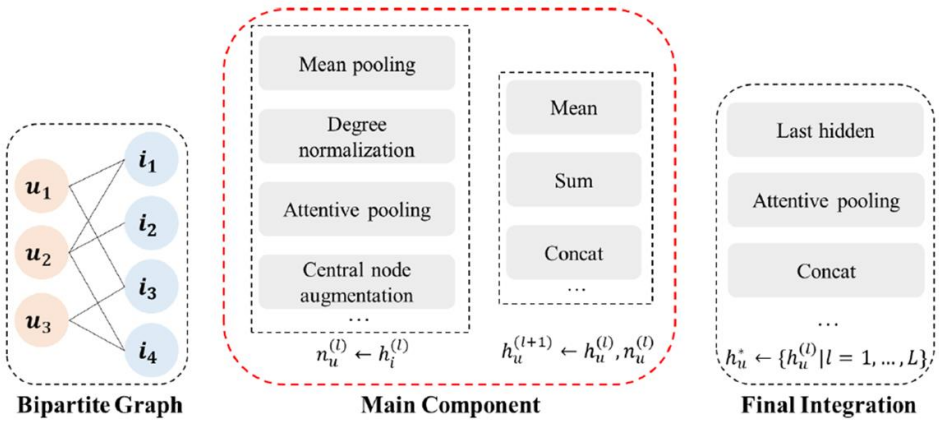
a. 모델이 그래프의 구조를 포착하지는 못한다.

b. training objective가 오직 일차 graph structure를 포착한다.

참고로 일차 graph는 이웃들의 직접적인 이웃 고려하며 보통 그래프의 구조 정보를 인접 행렬을 통해 구현한다. 고차원 그래프는 이웃 노드들의 연결성뿐만 아니라, 다양한 연결 패턴을 고려한다. 예를 들어, A노드의 이웃의 이웃까지 연결성을 고려해 인접행렬을 만들거나, 더 멀리 떨어진 이웃들의 연결성까지 고려하는 것이다.(인접행렬의 거듭 제곱으로 표현된다.)

(4) General Recommendation

Without side information



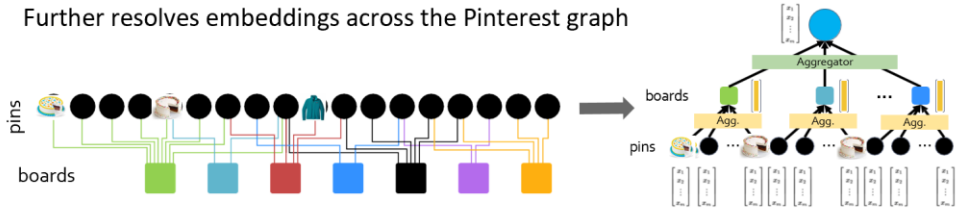
(5) Pinterest

- **Goal:** Generate embeddings for nodes in a large-scale Pinterest graph containing billions of objects
- **Key Idea:** Borrow information from nearby nodes
  - e.g., bed rail Pin might look like a garden fence, but gates and beds are rarely adjacent in the graph

- Pin embeddings are essential to various tasks like recommendation of Pins, classification, ranking
  - Services like “Related Pins”, “Search”, “Shopping”, “Ads”

(6) PinSAGE

- Graph has tens of billions of nodes and edges
- Further resolves embeddings across the Pinterest graph



Algorithm 1: CONVOLVE

Input : Current embedding $\mathbf{z}_u$ for node u; set of neighbor embeddings $\{\mathbf{z}_v | v \in \mathcal{N}(u)\}$, set of neighbor weights $\boldsymbol{\alpha}$; symmetric vector function $\gamma(\cdot)$

Output: New embedding $\mathbf{z}_u^{\text{NEW}}$ for node u

$\mathbf{n}_u \leftarrow \gamma\left(\{\text{ReLU}\left(\mathbf{Q}\mathbf{h}_v + \mathbf{q}\right) \mid v \in \mathcal{N}(u)\}, \boldsymbol{\alpha}\right)$;

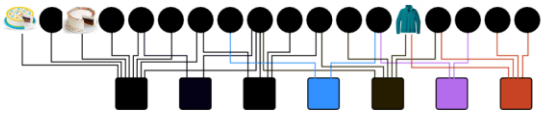
$\mathbf{z}_u^{\text{NEW}} \leftarrow \text{ReLU}\left(\mathbf{W} \cdot \text{CONCAT}(\mathbf{z}_u, \mathbf{n}_u) + \mathbf{w}\right)$;

$\mathbf{z}_u^{\text{NEW}} \leftarrow \mathbf{z}_u^{\text{NEW}} / \|\mathbf{z}_u^{\text{NEW}}\|_2$

74

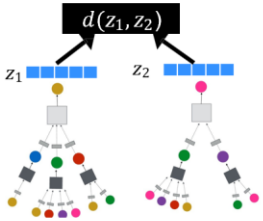
In addition to the GNN model, the PinSAGE paper introduces several methods to scale the GNN to a billion-scale recommender system (e.g., Pinterest).

- **Shared negative samples across users in a mini-batch**
- **Hard negative samples**
- **Curriculum learning**
- Mini-batch training of GNNs on a large-graph



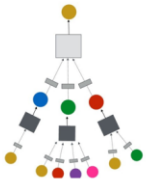
- **Task: Recommend related pins to users**
- Learn node embeddings z_i such that

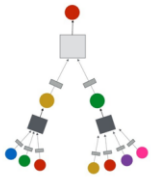
$$d(z_{cake1}, z_{cake2}) < d(z_{cake1}, z_{sweater})$$



* Shared Negative Sample

- **Recall**: In BPR loss, for each user $u^* \in \boldsymbol{U}_{\text{mini}}$, we sample one positive item v_{pos} and a set of sampled negative items $\boldsymbol{V}_{\text{neg}} = \{v_{\text{neg}}\}$
- Using more negative samples per user improves the recommendation performance, but is also expensive.
- We need to generate $|\boldsymbol{U}_{\text{mini}}| \cdot |\boldsymbol{V}_{\text{neg}}|$ embeddings for negative nodes.
- We need to apply $|\boldsymbol{U}_{\text{mini}}| \cdot |\boldsymbol{V}_{\text{neg}}|$ GNN computational graphs (see right), which is expensive.





- **Key idea**: We can share the same set of negative samples $\boldsymbol{V}_{\text{neg}} = \{v_{\text{neg}}\}$ ***across all users*** $\boldsymbol{U}_{\text{mini}}$ in the mini-batch.
- This way, we only need to generate $|\boldsymbol{V}_{\text{neg}}|$ embeddings for negative nodes.
  - This saves the node embedding generation computation **by a factor of** $|\boldsymbol{U}_{\text{mini}}|$!
  - Empirically, the performance stays similar to the non-shared negative sampling scheme.

(7) Curriculum Learning

Key insight: It is effective **to make the negative samples *gradually harder* in the process of training**.

At n-th epoch, we add $n - 1$ hard negative items.

- #(Hard negatives) gradually increases in the process of training.

The model will gradually learn to make finer-grained predictions.

Idea: use harder and harder negative samples

Include more and more hard negative samples for each epoch


Source pin


Positive


Easy negative

Hard negative

* Hard Negative

Challenge: Industrial recsys needs to make **extremely fine-grained predictions**.

- #Total items: Up to billions.
- #Items to recommend for each user: 10 to 100.

Issue: The shared negative items are randomly sampled from all items

- Most of them are "**easy negatives**", i.e., a model does not need to be fine-grained to distinguish them from positive items.

We need a way to sample "**hard negatives**" to force the model to be fine-grained!

- **For each user node**, the **hard negatives** are item nodes that are close (but not connected) to the user node in the graph.
- Hard negatives for user $u \in \boldsymbol{U}$ are obtained as follows:

1. Compute personalized page rank (PPR) for user u.
2. Sort items in the descending order of their PPR scores.
3. Randomly sample item nodes that are ranked high but not too high, e.g., 2000$^{\text{th}}$ —5000$^{\text{th}}$.
   - Item nodes that are close but not too close (connected) to the user node.

- The hard negatives for each user are used in addition to the shared negatives.