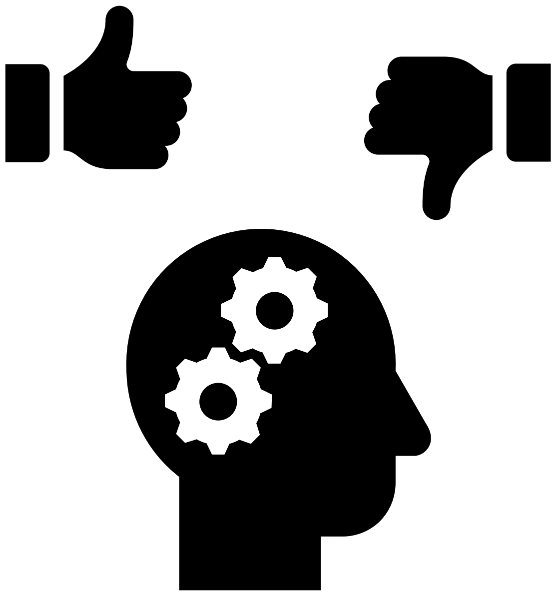
Bayesian Personalized Ranking from Implicit Feedback

홍진영

A Table of Contents.

Part 1, **Introduction**

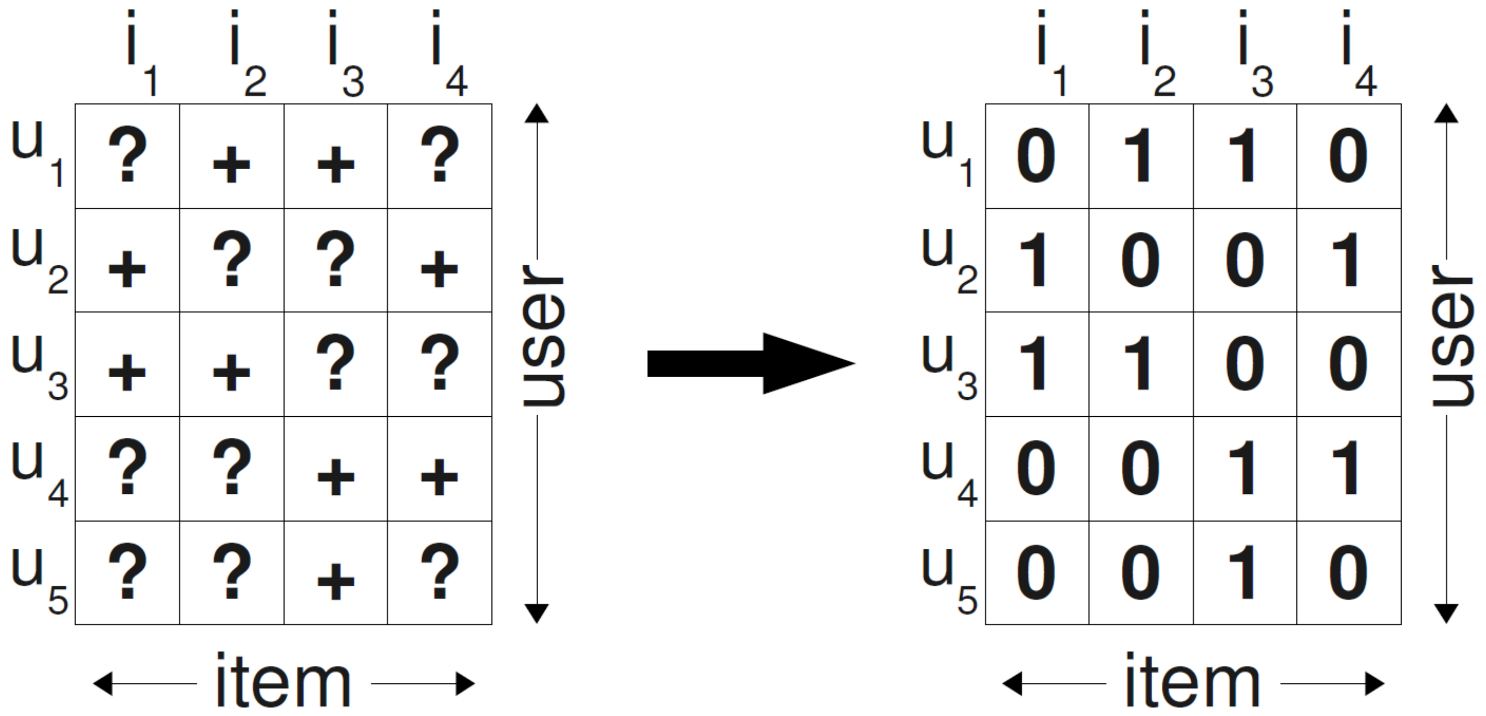


Implicit Feedback의 분류

Explicit Feedback		Implicit Feedback
- Item에 대한 사용자의 직접적인 피드백 - 별점, 좋아요 등	<	- Item에 대한 사용자의 내재적인 피드백 - 클릭, 머무는 시간 등

Part 1,
Introduction

Implicit Feedback의 분류



Missing value와 negative를 구분할 수 없음
→ 전통적인 방법에선 모두 0으로 취급
→ 나중에 예측해야 할 곳들이 학습 단계에서는 0으로 제공됨

기존 model들의 한계

- kNN

: similarity matrix를 이용해 user가 item을 선택할지 예측

- Matrix Factorization

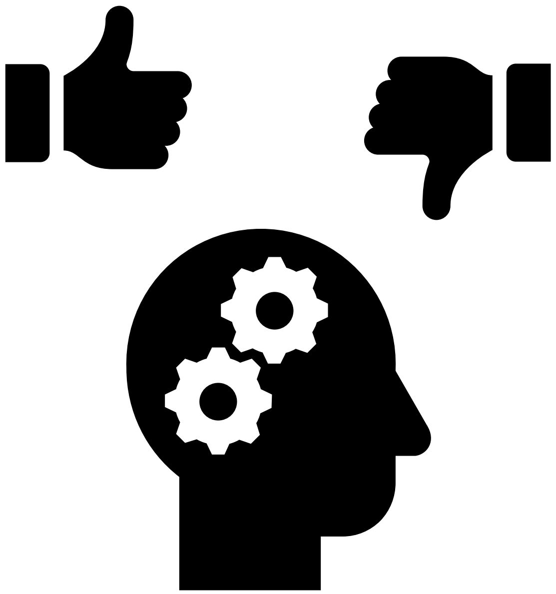
: user X item matrix를 user matrix, item matrix로 분해하여 예측

→ ranking을 목적으로 최적화를 하는 것이 아닌, item을 선택할 확률에 최적화된 model

∴ Missing value/negative를 구분하며, item 간의 ranking에 최적화된 model

Part 2,

Formalization



Formalization

용어 정의

U : 모든 user

I : 모든 item

$S \subseteq U \times I$: implicit feedback이 관측된 (u, i)의 집합

$I_u^+ := \{i \in I : (u, i) \in S\}$: user u가 implicit feedback을 준 i의 집합

$U_i^+ := \{u \in U : (u, i) \in S\}$: item i에 대해서 implicit feedback을 준 u의 집합

Formalization

용어 정의

$>_u$: user u의 item 선호 순서

$$\forall i, j \in I : i \neq j \Rightarrow i >_u j \vee j >_u i \qquad (totality)$$

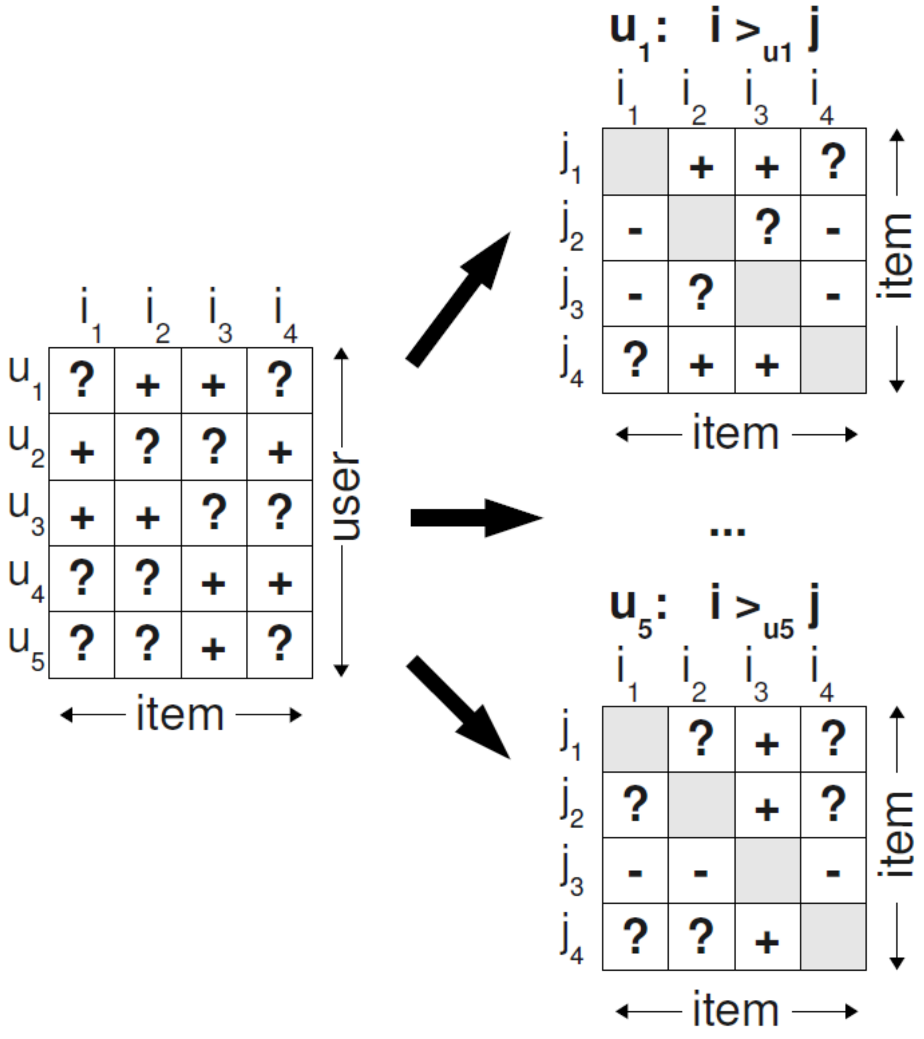
$$\forall i, j \in I : i >_u j \wedge j >_u i \Rightarrow i = j \qquad (antisymmetry)$$

$$\forall i, j, k \in I : i >_u j \wedge j >_u k \Rightarrow i >_u k \qquad (transitivity)$$

+ Implicit feedback이 관측되지 않은 item보다 관측된 item을 선호한다.

Formalization

용어 정의



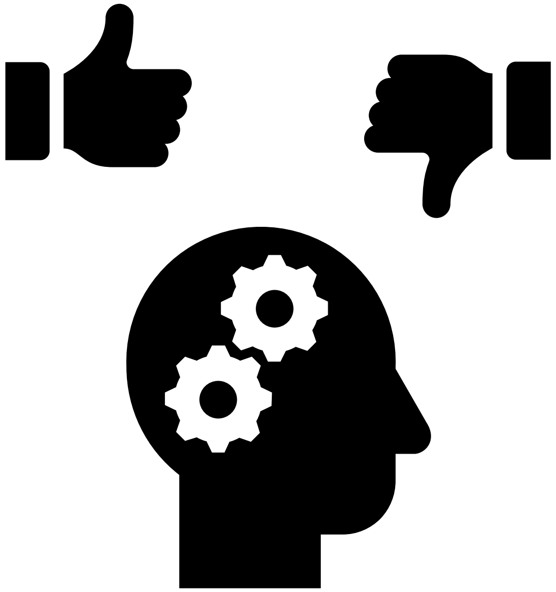
$$D_S := \{(u, i, j) | i \in I_u^+ \wedge j \in I \setminus I_u^+\}$$

관측된 item 관측되지 않은 item

$$\rightarrow \quad i >_u j$$

Part 3,

BPR Optimization Criterion



BPR Optimization Criterion

Maximum a posteriori estimation

$$p(\theta|y) = \frac{p(y|\theta)p(\theta)}{p(y)}$$

$$= \frac{p(y|\theta)p(\theta)}{\int_\theta p(y|\theta)p(\theta)d\theta} \qquad \left(\because p(y) = \int_\theta p(y|\theta)p(\theta)d\theta\right)$$

$$\propto p(y|\theta)p(\theta)$$

Posterior ∝ Likelihood X Prior

$p(\theta)$: Prior - subjective belief about θ

$p(y|\theta)$: Likelihood – observation (data) regarding θ

$p(\theta|y)$: Posterior - Updated belief about θ with the data

BPR Optimization Criterion

Likelihood

$$p(\Theta | >_u) \propto p(>_u | \Theta)\, p(\Theta)$$

- 가정

- 각 user는 서로 독립적으로 작용한다.
- 각 item pair의 비교는 서로 독립적으로 작용한다.

- User u의 item i, j의 비교는 $i >_u j$ 혹은 $j >_u i$ (Totality)

→ $>_u$ 는 Bernoulli 분포를 따름

- User u의 item i, j가 $i >_u j$ 이면 $j <_u i$ (Antisymmetry)

$$\therefore \prod_{u \in U} p(>_u | \Theta) = \prod_{(u,i,j) \in D_S} p(i >_u j | \Theta)$$

BPR Optimization Criterion

Likelihood

$$p(\Theta | >_u) \propto p(>_u | \Theta)\, p(\Theta)$$

- $p(i >_u j | \Theta) := \sigma(\hat{x}_{uij}(\Theta))$

→ totality, antisymmetry, transitivity를 만족하기 위해 sigmoid 함수 사용

→ $\hat{x}_{uij}(\Theta)$: parameter Θ를 가지는 model이 예측한 user u의 item i, j 사이의 관계

Prior

$$p(\Theta | >_u) \propto p(>_u | \Theta)\, p(\Theta)$$

$$p(\Theta) \sim N(0, \Sigma_\Theta)$$

Part 3,

BPR Optimization Criterion

Posterior

$$p(\Theta | >_u) \propto p(>_u | \Theta)\, p(\Theta)$$

$$\begin{aligned}
\text{BPR-OPT} &:= \ln p(\Theta | >_u) \\
&= \ln p(>_u | \Theta)\, p(\Theta) \\
&= \ln \prod_{(u,i,j) \in D_S} \sigma(\hat{x}_{uij})\, p(\Theta) \\
&= \sum_{(u,i,j) \in D_S} \ln \sigma(\hat{x}_{uij}) + \ln p(\Theta) \\
&= \sum_{(u,i,j) \in D_S} \ln \sigma(\hat{x}_{uij}) - \lambda_\Theta ||\Theta||^2
\end{aligned}$$

BPR Optimization Criterion

BPR-OPT와 AUC 사이의 관계

- AUC

$$\mathrm{AUC}(u) := \frac{1}{|I_u^+|\,|I \setminus I_u^+|} \sum_{i \in I_u^+} \sum_{j \in |I \setminus I_u^+|} \delta(\hat{x}_{uij} > 0)$$

$$\delta(x > 0) = H(x) := \begin{cases} 1, & x > 0 \\ 0, & \text{else} \end{cases}$$

$$\mathrm{AUC} := \frac{1}{|U|} \sum_{u \in U} AUC(u) = \sum_{(u,i,j) \in D_S} z_u\, \delta(\hat{x}_{uij} > 0)$$

$$= \sum_{(u,i,j) \in D_S} z_u\, H(\hat{x}_{uij})$$

$$z_u = \frac{1}{|U|\,|I_u^+|\,|I \setminus I_u^+|}$$

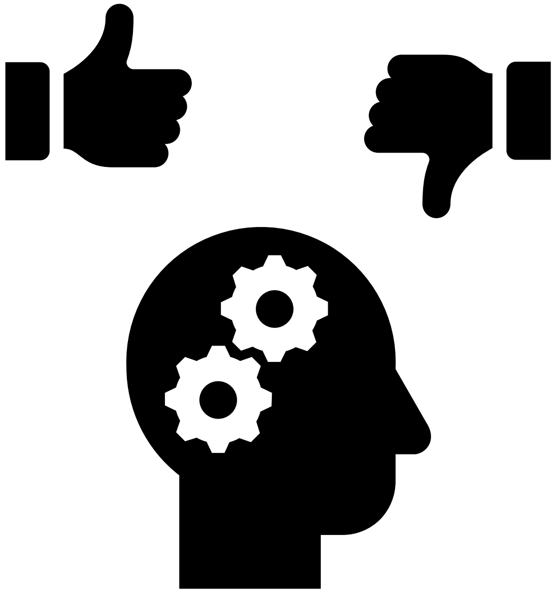
- BPR-OPT

$$\text{BPR-O}\textsc{pt} = \sum_{(u,i,j) \in D_S} \ln \sigma(\hat{x}_{uij}) - \lambda_\Theta ||\Theta||^2$$

∴ BPR-OPT를 최적화하는 것은 AUC를 최적화하는 것과 같다.

Part 4,

BPR Learning Algorithm



Part 4,

BPR Learning Algorithm

Optimizing BPR

- BPR optimization criterion이 미분가능하므로 gradient descent를 통해 parameter를 update

$$\frac{\partial \text{BPR-O}\textsc{pt}}{\partial \Theta} = \sum_{(u,i,j)\in D_S} \frac{\partial}{\partial \Theta} \ln \sigma(\hat{x}_{uij}) - \lambda_\Theta \frac{\partial}{\partial \Theta} ||\Theta||^2$$

$$\propto \sum_{(u,i,j)\in D_S} \frac{-e^{-\hat{x}_{uij}}}{1+e^{-\hat{x}_{uij}}} \cdot \frac{\partial}{\partial \Theta} \hat{x}_{uij} - \lambda_\Theta \Theta$$

→ model의 선택에 따라 update rule이 달라진다. ($\hat{x}_{uij}$, $\frac{\partial}{\partial \Theta}\hat{x}_{uij}$ term)

BPR Learning Algorithm

Optimizing BPR

$$
\begin{array}{ll}
1: & \textbf{procedure } \textsc{LearnBPR}(D_S, \Theta) \\
2: & \quad \text{initialize } \Theta \\
3: & \quad \textbf{repeat} \\
4: & \qquad \text{draw } (u,i,j) \text{ from } D_S \\
5: & \qquad \Theta \leftarrow \Theta + \alpha \left(\frac{e^{-\hat{x}_{uij}}}{1+e^{-\hat{x}_{uij}}} \cdot \frac{\partial}{\partial \Theta} \hat{x}_{uij} + \lambda_\Theta \cdot \Theta \right) \\
6: & \quad \textbf{until } \text{convergence} \\
7: & \quad \textbf{return } \hat{\Theta} \\
8: & \textbf{end procedure}
\end{array}
$$

Q: 왜 full gradient descent가 아닌 stochastic gradient descent를 사용하는가?

A: Implicit feedback이 관측된 i 집단의 크기가 커지면 i 집단이 기울기를 지배한다. (learning rate를 낮춰야 함)

Part 4,

BPR Learning Algorithm

Matrix Factorization의 update rule

- Parameter $\Theta = (W, H) \quad (\because \hat{X} := WH^t)$
- $\hat{x}_{uij} := \hat{x}_{ui} - \hat{x}_{uj} = \langle w_u, h_i \rangle - \langle w_u, h_j \rangle$

$$= \sum_{f=1}^{k} w_{uf} \cdot h_{if} - \sum_{f=1}^{k} w_{uf} \cdot h_{jf} \rightarrow \frac{\partial}{\partial \theta}\hat{x}_{uij} = \begin{cases} (h_{if} - h_{jf}) & \text{if } \theta = w_{uf}, \\ w_{uf} & \text{if } \theta = h_{if}, \\ -w_{uf} & \text{if } \theta = h_{jf}, \\ 0 & \text{else} \end{cases}$$

BPR Learning Algorithm

Adaptive kNN의 update rule

- Parameter $\Theta = C : I \times I$

$$\hat{x}_{uij} := \hat{x}_{ui} - \hat{x}_{uj}$$

$$= \sum_{l \in I_u^+ \wedge l \neq i} c_{il} - \sum_{l \in I_u^+ \wedge l \neq j} c_{jl} \rightarrow \frac{\partial}{\partial \theta} \hat{x}_{uij} = \begin{cases} +1 & \text{if } \theta \in \{c_{il}, c_{li}\} \wedge l \in I_u^+ \wedge l \neq i, \\ -1 & \text{if } \theta \in \{c_{jl}, c_{lj}\} \wedge l \in I_u^+ \wedge l \neq j, \\ 0 & \text{else} \end{cases}$$

$$c_{i,j}^{\text{cosine}} := \frac{|U_i^+ \cap U_j^+|}{\sqrt{|U_i^+| \cdot |U_j^+|}}$$

BPR Learning Algorithm

기존 model들의 한계

- Weighted Regularized Matrix Factorization

$$\sum_{u \in U} \sum_{i \in I} c_{ui}(\langle w_u, h_i \rangle - 1)^2 + \lambda ||W||_f^2 + \lambda ||H||_f^2$$

→ item pair를 고려하지 않음, square loss

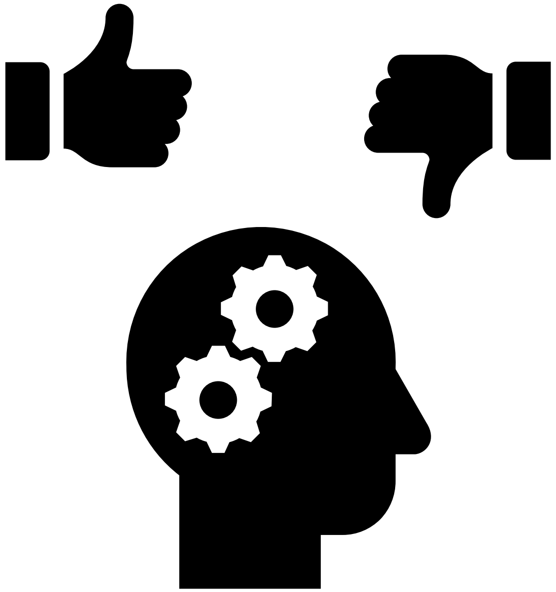
- Maximum Margin Matrix Factorization

$$\sum_{(u,i,j) \in D_s} \max(0, 1 - \langle w_u, h_i - h_j \rangle) + \lambda_w ||W||_f^2 + \lambda_h ||H||_f^2$$

→ matrix factorization에만 사용할 수 있는 criterion

Part 5,

Evaluation



문제 정의

- Dataset: Rossman dataset, Netfilx dataset(rating data는 제거)

→ user마다 한 개의 item을 빼서 train dataset을 만듦(10 fold)

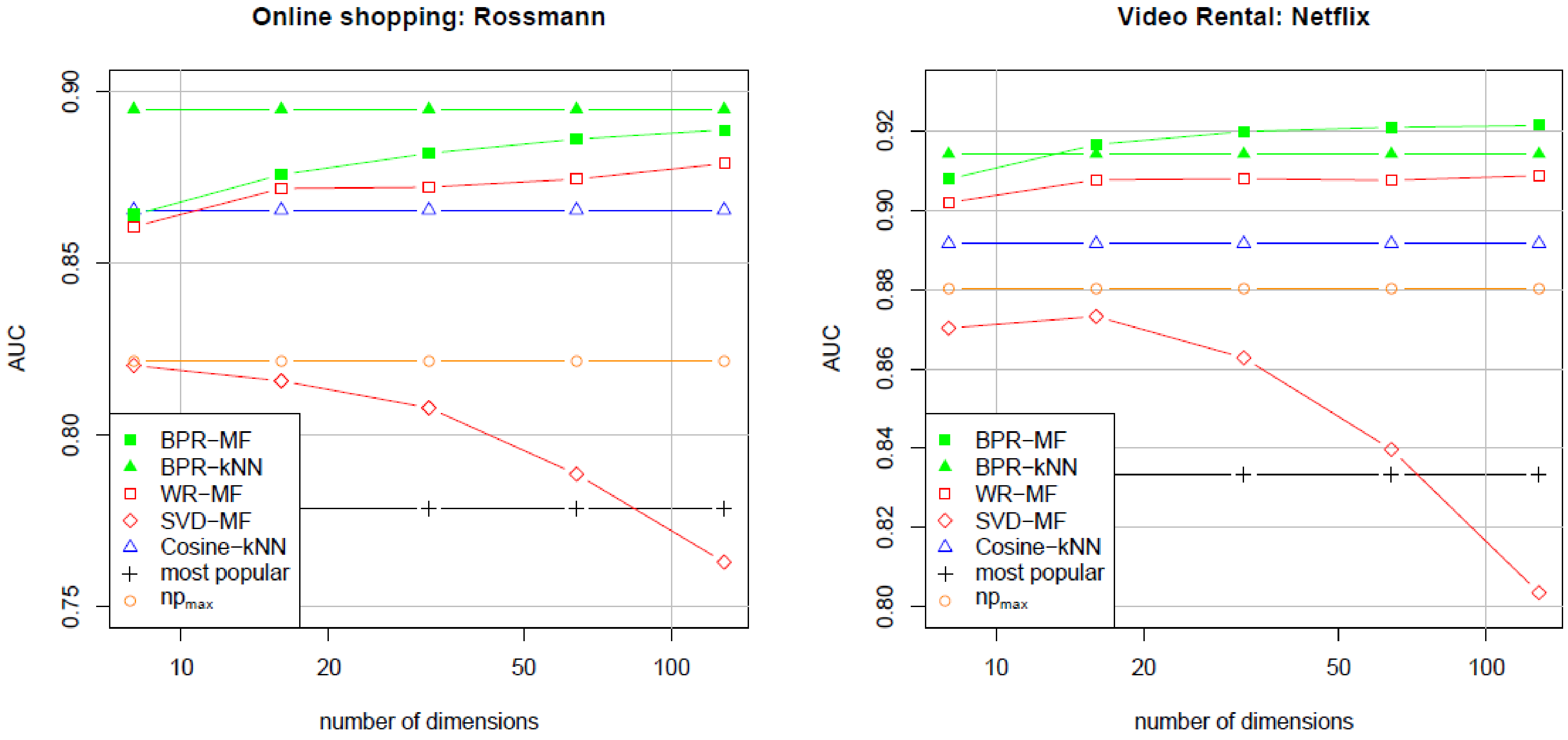
→ 제거된 sample들은 test dataset으로 사용

- 평가산식: AUC

$$\mathrm{AUC} = \frac{1}{|U|} \sum_{u} \frac{1}{|E(u)|} \sum_{(i,j) \in E(u)} \delta(\hat{x}_{ui} > \hat{x}_{uj})$$

$$E(u) := \{(i, j) | (u, i) \in S_{\text{test}} \wedge (u, j) \notin (S_{\text{test}} \cup S_{\text{train}})\}$$

결과



Online shopping: Rossmann
Video Rental: Netflix
AUC
number of dimensions
10
20
50
100
0.75
0.80
0.85
0.90
0.80
0.82
0.84
0.86
0.88
0.90
0.92
BPR-MF
BPR-kNN
WR-MF
SVD-MF
Cosine-kNN
most popular
np$_{max}$

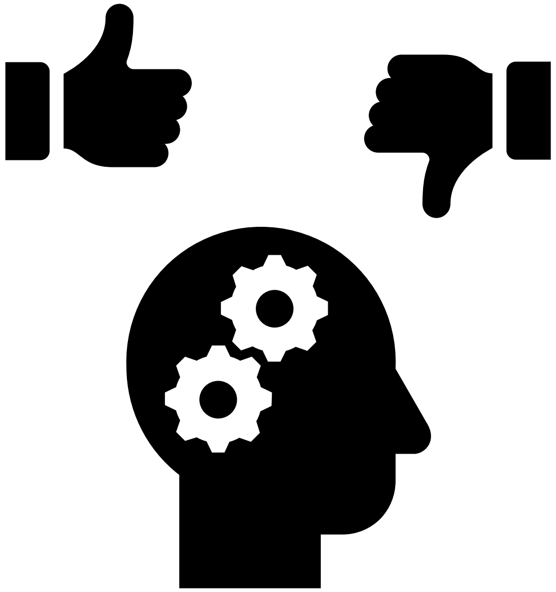
결론

- BPR-OPT는 implicit feedback을 이용해서 user에 따라 item 선호에 순서를 매기는 personalized ranking model

- Learning algorithm은 기존 pair-wise model에 적용할 수 있고(generic), 기존 model의 성능을 훨씬 뛰어넘는 성능을 보여줌

Part 6,

Implementation



Implementation

전처리

- Dataset: MovieLens

→ Netflix dataset과 같이 rating을 삭제하여 implicit feedback만으로 model 학습

→ 10000 users, 5000 movies (I_u^+ , U_i^+ ≥ 10)

	userId	movieId	rating	timestamp
	1132	8	4.0	2001-07-04 07:02:29
	1132	376	2.0	2001-07-04 08:00:49
	1132	93	4.0	2001-07-04 07:41:51
	1132	284	3.0	2001-07-04 08:01:38
	1132	73	3.0	2001-07-14 07:57:24
	...	...	...	...
	1957	2130	4.5	2009-10-17 21:52:53
	1957	1016	4.5	2009-12-07 18:15:20
	1957	328	4.5	2009-11-13 15:42:00
	1957	365	5.0	2009-11-13 15:42:24
	1957	3756	2.5	2009-10-17 20:25:36

전처리

- User – movie pair가 존재하면 해당 원소에 1 표시

```
userid_mat = np.zeros((10000, 5000))
```

```
for i in dataset.index:
    userid_mat[dataset.loc[i, "userId"]][dataset.loc[i, "movieId"]] = 1
userid_mat
```

```
array([[1., 1., 1., ..., 1., 1., 1.],
       [1., 1., 1., ..., 0., 0., 0.],
       [1., 1., 1., ..., 0., 0., 0.],
       ...,
       [1., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [1., 0., 1., ..., 0., 0., 0.]])
```

Part 6, Implementation

Train, test dataset 분리

- User마다 한 개의 item을 빼서 train dataset을 만듦
- 제거된 sample들은 test dataset으로 사용

```
def train_test_split(data_mat):
    np.random.seed(42)
    data_mat = data_mat.copy()
    test_list = []
    for i in range(data_mat.shape[0]):
        temp = data_mat[i]
        test_idx = np.random.choice(np.where(temp == 1)[0])
        compare_idx = np.random.choice(np.where(temp == 0)[0])
        data_mat[i][test_idx] = 0
        test_list.append((test_idx, compare_idx))
    return data_mat, test_list

train_mat, test_list = train_test_split(userid_mat)
```

Bootstrap 함수 구현

- D_S의 원소 중 하나를 복원 추출

→ stochastic gradient descent에 사용

```
def bootstrap(mat):
    user = np.arange(mat.shape[0])
    u = np.random.choice(user)
    temp = mat[u]
    i = np.random.choice(np.where(temp == 1)[0])
    j = np.random.choice(np.where(temp == 0)[0])
    return (u, i, j)
```

Implementation

Matrix factorization

- Parameter $\Theta = (W, H) \quad (\because \hat{X} := WH^t)$
- Number of dimension: 128

```
k = 128

user_mat = np.random.random((train_mat.shape[0], k))
item_mat = np.random.random((train_mat.shape[1], k))
```

Part 6,

Implementation

Matrix Factorization의 update rule

```
n_epoch = 10000 * 50
learning_rate = 1e-2
alpha = 1e-4

for epoch in range(n_epoch):

    u, i, j = bootstrap(train_mat)
    W_u = user_mat[u,:]
    H_i = item_mat[i,:]
    H_j = item_mat[j,:]

    x_uij = np.dot(W_u, H_i) - np.dot(W_u, H_j)
    sigmoid = 1 / (1 + np.exp(x_uij))

    user_mat[u,:] += learning_rate * (sigmoid * ( H_i - H_j ) - alpha * W_u)
    item_mat[i,:] += learning_rate * (sigmoid * W_u - alpha * H_i)
    item_mat[j,:] += learning_rate * (-sigmoid * W_u - alpha * H_j)
```

$$\frac{\partial}{\partial\theta}\hat{x}_{uij} = \begin{cases} (h_{if} - h_{jf}) & \text{if } \theta = w_{uf}, \\ w_{uf} & \text{if } \theta = h_{if}, \\ -w_{uf} & \text{if } \theta = h_{jf}, \\ 0 & \text{else} \end{cases}$$

Implementation

결과

- Timestamp를 고려하여 가장 최근의 영화를 제거한 train dataset을 사용했으면 조금 더 좋은 결과를 얻었을 것이라 추측

```
Epoch: 0, Test AUC: 0.4864
Epoch: 50000, Test AUC: 0.8158
Epoch: 100000, Test AUC: 0.8497
Epoch: 150000, Test AUC: 0.8606
Epoch: 200000, Test AUC: 0.8624
Epoch: 250000, Test AUC: 0.8652
Epoch: 300000, Test AUC: 0.8701
Epoch: 350000, Test AUC: 0.8687
Epoch: 400000, Test AUC: 0.8690
Epoch: 450000, Test AUC: 0.8707
```

감사합니다