Collaborative Filtering for Implicit Feedback Datasets

Y. Hu, Y. Koren and C. Volinsky

Paper Review

오지환

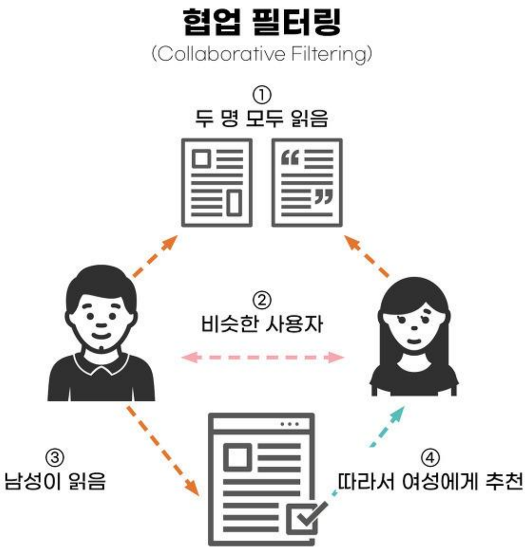
2023.01.10

Contents

1. Background
2. Previous task
3. Our model
4. Experiment in paper
5. Implement

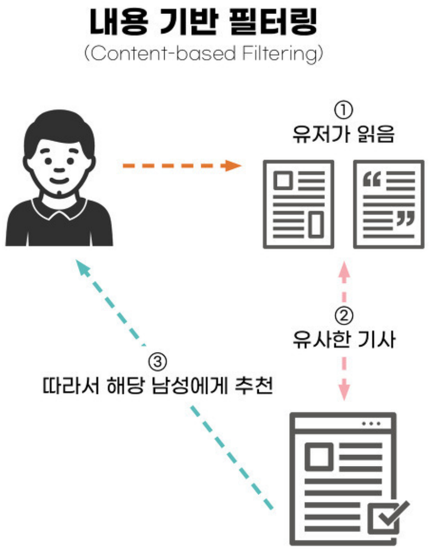
Background

1. Recommender System



- 사용로부터 얻은 선호도 정보에 기반하여 사용자에게 추천하는 방법
- 대표자적으로 **Neighborhood model** 과 **Latent factor model**이 있음
- **Latent factor model** 에는 **User-oriented** 와 **Item-oriented** 가 있음

단점
- 데이터가 부족하면 추천을 잘 하지 못함. (Cold start problem)



- 콘텐츠의 유사도를 기반으로 사용자에게 추천하는 방법
- 간단하고 쉬운 모델

단점
- 추천 콘텐츠가 다양하지 않음(비슷한 아이템을 계속해서 추천)

Background

2. User feedback data types

1. Explicit-feedback data

- 사용자가 선호도를 직접적으로 표현한 데이터
- 예시: 평점, 리뷰 등

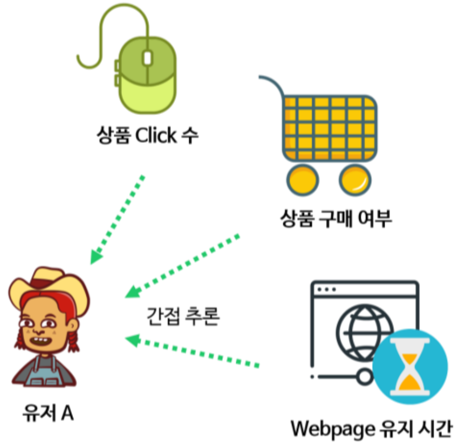
- 장점: 유저 선호도를 정확하게 알 수 있음.
- 단점: 데이터를 구하기 힘들고 양이 적음.

	 다만 악에서 구하소서	 강철비2 : 정상회담
유저 A	★★★★★ 역시 이정재 존재감, 대체불가 황정민, 김정민 캐릭터 연기폭이 무궁무진하네. 배우들 조합만으로도 1등이네	★★☆☆☆ 미국대통령이 너무나 쉽고 허술하게 납치되는 과정을 보면서 어이상실 한번 하고 백악관이 이 사실조차 모른다는데서 두번 어이상실..
유저 B	★★★★☆ 오랜만에, 진짜 간만에 좋은 영화 한 편을 봤습니다. 액션, 감동, 재미 세 마리 토끼를 다 잡았다고 생각합니다.	★★★★☆ 색감이 너무 예뻤어요! 재밌는 영화 감사합니다 :D
유저 C	★★★★★ 박정민이 왜 포스터, 예고에 없는 지 알았음. 어쩜 그리 연기를 잘 하는지	★★★★★ 재미 없다 해서 기대 안 했는데, 그건 그냥 그런 분들 의견인듯요. 영화 보면서 웃다가 긴장했다가 충분히 재미있게 즐겼습니다.
유저 D	★☆☆☆☆ 정말 너무 뻔해서 놀람…	★★★☆☆ 중간중간 유머나, 위트는 참신했음. 각 국에 대한 풍자도 불만했고, 근데... 스토리의 긴박감이 좀 부족하지 않았나 싶네요..

2. Implicit-feedback data

- 사용자가 선호도를 간접적으로 표현한 데이터
- 예시: 검색 기록, 클릭 횟수, 구매 내역, 열람 내역 등

- 장점: 비교적 데이터를 구하기 쉬움.
- 단점: 유저의 선호도를 유추해야 하는 번거로움이 있음.



Background

3. Main characteristic of Implicit-feedback

1. No negative feedback

- 해당 아이템에 대한 부정적인 행동을 추론하기 어렵다.
- 예를 들어, 유저 A가 Item 1을 구매하지 않았을 때, 선호하지 않아서 사지 않은 것인지, 혹은 상품을 알지 못했기 때문에 구매를 하지 않은 것인지, 확실하지 않다.

2. Inherently noisy

- Implicit-feedback 데이터는 본질적으로 노이즈가 많다.
- 예를 들어, 유저 A가 Item 1을 10시간 동안 열람하였다는 데이터가 있을때, 그 아이템이 마음에 들어서 열람한 것인지, 혹은 페이지를 닫지 않고 다른 일을 하러 갔는지, 확실하지 않다.

3. Numerical value of implicit feedback indicates confidence not preference

- Explicit Feedback에서는 수치가 클수록 높은 선호도를 나타내지만, Implicit Feedback에서는 항상 그렇지만은 않다.
- 반복적인 접속, 일정한 열람 시간 등은 preference(선호도) 가 아니라 confidence(신뢰도) 가 높다고 해석해야 한다.

4. Requires appropriate measures to evaluate

- Implicit Feedback에서는 적절한 평가지표를 고민해 봐야 한다.
- 예를 들어, 유저 A가 예능 TV프로그램을 시청하고 있을 때는 동 시간에 방영하는 드라마를 시청하지 못한다. 즉, 드라마는 기록에 시청 기록에 남지 않는다. 하지만 유저 A는 드라마도 좋아할 수 있다.

Previous Work

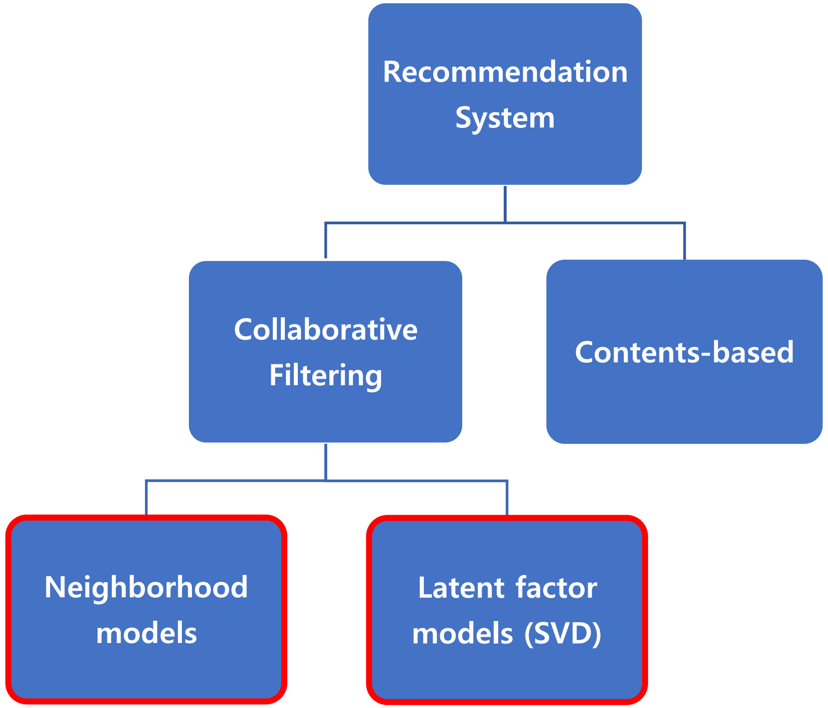
1. Notation

1. User/ Item
   - Users: u, v
   - Items, i, j

2. Input data: r_{ui} (Implicit-feedback data)
   - e.g.
     The number of times that user u purchased item i.
     The spent time of user u on webpage i.

   - No actions are set to $r_{ui} = 0$

2. Collaborative Filtering



Previous Work

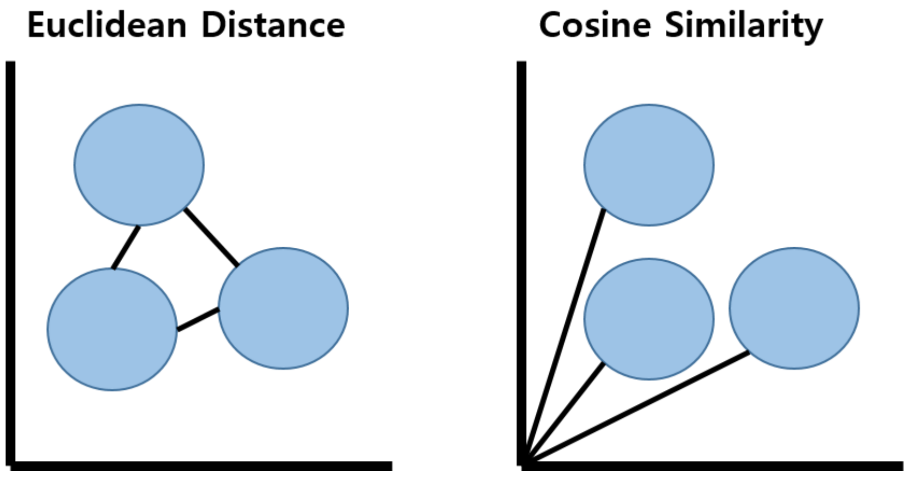
3. Neighborhood model (CF)

- User-oriented method 와 item-oriented method가 있음.
- 일반적으로 item-oriented method 의 accuracy와 scalability가 더 좋음.

- Neighborhood item의 similarity 를 측정 한 후, 가장 유사한 k개 item의
  r_{uj} $(observed)$를 통하여 r_{ui} $(unobserved)$ 값을 예측.

$$\hat{r}_{ui} = \frac{\sum_{j \in S^k(i;u)} s_{ij} r_{uj}}{\sum_{j \in S^k(i;u)} s_{ij}} \qquad (1)$$

r_{uj} : $Observed\ value$
r_{ui}: $Unobserved\ value$
s_{ij} : $Similarity\ between\ item\ i\ and\ j$
$S^k(i;u)$: Similarity set of top k items rated by u



Previous Work

4. Latent factor model (CF)

- Observed value r_{ui} 를 설명하는 latent factor를 찾는 것을 목적으로 함.
- 일반적으로 accuracy와 scalability 측면에서 neighborhood model 보다 좋음.
- 대표적으로 Singular Value Decomposition (SVD) method가 있음.
- $User\ factor\ vector$: $x_u \in \Re^f$, $Item$ factor $vector$: $y_i \in \Re^f$ f: $factor\ dimension$
- 두 벡터의 inner product를 통해 예측 : $\hat{r}_{ui} = x_u^T y_i$

Loss function: $$\min_{x_\star, y_\star} \sum_{r_{u,i} \text{ is known}} (r_{ui} - x_u^T y_i)^2 + \underbrace{\lambda(\|x_u\|^2 + \|y_i\|^2)}_{\text{Regularization term for avoid overfitting}} \quad (2)$$ Using SGD

Our Model

1. Latent factor model for implicit dataset

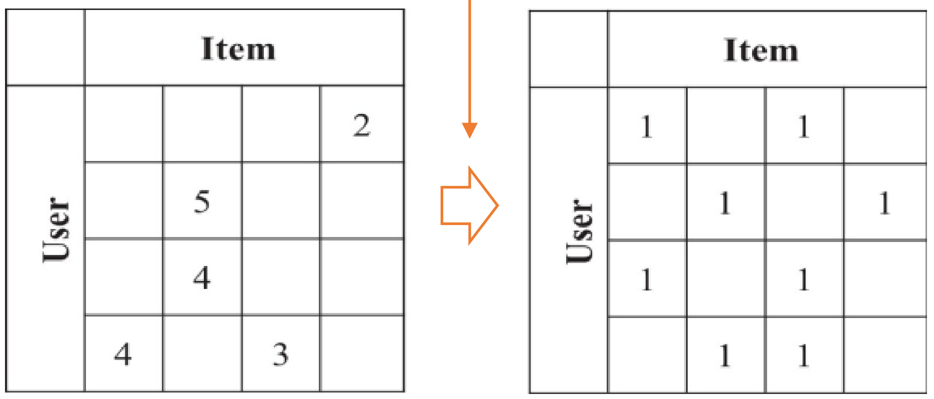
- r_{ui} 대신 p_{ui} (preference) 와 c_{ui} (confidence) 정의.

  - Preference

$$p_{ui} = \begin{cases} 1 & r_{ui} > 0 \\ 0 & r_{ui} = 0 \end{cases}$$

  - Confidence

$$c_{ui} = 1 + \alpha r_{ui} \qquad \alpha = 40$$



(a) ratings setting

(b) one-class setting

One class collaborative filtering (OCCF)

Our Model

1. Latent factor model for implicit dataset

- r_{ui} 대신 p_{ui} (preference) 와 c_{ui} (confidence) 정의.
  - Preference

    $$p_{ui} = \begin{cases} 1 & r_{ui} > 0 \\ 0 & r_{ui} = 0 \end{cases}$$

  - Confidence

    $$c_{ui} = 1 + \alpha r_{ui} \qquad \alpha = 40$$

- 두 벡터의 inner product를 통해 예측 : $p_{ui} = x_u^T y_i$
- Our goal : p_{ui} 를 결정하는 벡터 $x_u \in \Re^f$, $y_i \in \Re^f$ 를 최적화

Loss function:

$$\min_{x_\star, y_\star} \sum_{u,i} \underline{c_{ui}} (\underline{p_{ui}} - x_u^T y_i)^2 + \lambda \left(\sum_u \|x_u\|^2 + \sum_i \|y_i\|^2 \right) \tag{3}$$

Our Model

1. Latent factor model for implicit dataset

- Computational Complexity Issue
  - Explicit Loss function: $$\min_{x_\star, y_\star} \sum_{r_{u,i} \text{ is known}} (r_{ui} - x_u^T y_i)^2 + \lambda(\|x_u\|^2 + \|y_i\|^2) \qquad (2)$$

↓

Implicit model 에서는 모든 user-item pairs에 대한 optimization이 필요

  - Implicit Loss function: $$\min_{x_\star, y_\star} \sum_{u,i} c_{ui}(p_{ui} - x_u^T y_i)^2 + \lambda\left(\sum_u \|x_u\|^2 + \sum_i \|y_i\|^2\right) \qquad (3)$$

Sparse matrix 에서 user-item pairs의 개수는 수십억 개에 달함.
→ 계산량이 많아 SGD와 같은 optimization technique 을 활용할 수 없음
→ 다른 적절한 optimization technique 방법 필요

Our Model

2. Alternative Least Square (ALS)

- Implicit Loss function: $$\min_{x_\star, y_\star} \sum_{u,i} c_{ui}(p_{ui} - \underline{x_u^T y_i})^2 + \lambda \left(\sum_u \|x_u\|^2 + \sum_i \|y_i\|^2 \right) \quad (3)$$

서로 다른 두 변수의 내적 → **non-convex problem**

- x, y 중 하나를 상수 취급 → quadratic form (convex)

$$\frac{\partial L(x_u)}{\partial x_u} = 0 \longrightarrow x_u = (Y^T C^u Y + \lambda I)^{-1} Y^T C^u p(u) \quad (4)$$

$$\frac{\partial L(y_i)}{\partial y_i} = 0 \longrightarrow y_i = (X^T C^i X + \lambda I)^{-1} X^T C^i p(i) \quad (5)$$

- x, y 를 번갈아 가면서 re-computing 하여 update
- 10번 정도 업데이트 한 후, $\hat{p}_{ui} = x_u^T y_i$ 가 가장 큰 K개의 item 추천

Our Model

3. ALS proof

$$\frac{\partial L(x_u)}{\partial x_u} = -2\sum_{u,i} C_{ui}(p_{ui} - x_u^T y_i)Y_i + 2\lambda x_u$$

$$0 = -2\sum_{u,i} C_{ui}(p_{ui} - x_u^T y_i)y_i + 2\lambda x_u$$

$$\sum_i C_{ui}(x_u^T y_i)y_i + \lambda x_u = \sum_i C_{ui}(p_{ui})y_i$$

$$\sum_i C_{ui}y_i(y_i^T x_u) + \lambda x_u = \sum_i C_{ui}(p_{ui})y_i$$

$$(\sum_i C_{ui}y_i y_i^T + \lambda I)x_u = \sum_i C_{ui}(p_{ui})y_i$$

$$(YC_uY^T + \lambda I)x_u = Y^T C_u p(u)$$

$$\therefore\ x_u = (Y^T C^u Y + \lambda I)^{-1} Y^T C^u p(u)$$

m: user number, n: item number

User matrix: $X_u \in \Re^{m\times f}$

Item matrix: $Y_i \in \Re^{n\times f}$

Confidence matrix $C_u \in \Re^{n\times n}$, $C_i \in \Re^{m\times m}$ (diagonal)

$$\sum_i C_{ui}y_i y_i^T = C_{u1}y_1y_1^T + C_{u2}y_2y_2^T + C_{u3}y_3y_3^T + \cdots$$

$$Y = [y_1 y_2 \ldots y_i] \quad C = \begin{bmatrix} c_{u1} & 0 & 0 & 0 \\ 0 & c_{u2} & 0 & 0 \\ 0 & 0 & \ldots & 0 \\ 0 & 0 & 0 & c_{ui} \end{bmatrix} \quad Y^T = \begin{bmatrix} y_1^T \\ y_2^T \\ \ldots \\ y_i^T \end{bmatrix}$$

$$Y \times C \times Y^T = \left[c_{u1}y_1y_1^T + c_{u2}y_2y_2^T + \ldots + c_{ui}y_iy_i^T\right]$$

Our Model

4. Computational bottleneck

Bottleneck!

$$x_u = (\underline{Y^T C^u Y} + \lambda I)^{-1} \underline{Y^T C^u p(u)} \quad , u = 1,2,\cdots,m \qquad (4)$$

$$Y^T C^u Y = Y^T Y + Y^T (C^u - I) Y$$

$C^u - I$ has only n_u nonzero elements $n_u \ll n$

- Reduce computation cost: $O(f^2 n) \longrightarrow O(f^2 n_u)$
- Computation cost for find inverse matrix using Gaussian elimination: $O(f^3)$
- Total computation cost per user: $O(f^2 n + f^3)$
- Total computation cost for all user: $\underline{O(f^2 N + f^3 m)}$ $N: overall\ number\ of\ nonzero\ observations$

⇩

x_u 를 한 번 업데이트 할 때 마다 드는 계산 비용

Our Model

5. Explaining recommendations

- 좋은 추천시스템은 왜 추천했는지에 대한 이유를 설명할 수 있어야 함.
- 기존의 Latent factor model (SVD)은 왜 추천하는지에 대한 이유를 잘 설명할 수 없음

- Our model : $$\hat{p}_{ui} = y_i^T x_u = y_i^T \underbrace{(Y^T C^u Y + \lambda I)^{-1}}_{= W^u \in \Re^{f \times f} \text{ weighing matrix of user } u} Y^T C^u p(u) \quad (6)$$

$$= \sum_{j : r_{uj} > 0} s_{ij}^u c_{uj} \quad (7) \qquad s_{ij}^u = y_i^T W^u y_j$$

Our Model

5. Explaining recommendations

- 좋은 추천시스템은 왜 추천했는지에 대한 이유를 설명할 수 있어야 함.
- 기존의 Latent factor model (SVD)은 왜 추천하는지에 대한 이유를 잘 설명할 수 없음

- Our model :

$$\hat{p}_{ui} = y_i^T x_u = y_i^T \underline{(Y^T C^u Y + \lambda I)^{-1}} Y^T C^u p(u) \quad (6)$$

$= W^u \in \Re^{f \times f}$ *weighing matrix of user u*

$$\hat{p}_{ui} = y_i^T W^u \underbrace{Y^T C^u p(u)} = \sum_j y_i^T W^u y_j c_{uj} p_{uj} = \sum_{j: r_{uj} > 0} \underbrace{y_i^T W^u y_j}_{s_{ij}^u} c_{uj} = \sum_{j: r_{uj} > 0} s_{ij}^u c_{uj}$$

$$\begin{pmatrix} - & y_1 & - \\ - & y_2 & - \\ & \vdots & \\ - & y_n & - \end{pmatrix} \begin{pmatrix} c_{u1} & & & \\ & c_{u2} & & \\ & & \ddots & \\ & & & c_{un} \end{pmatrix} \begin{pmatrix} p_{u1} \\ p_{u2} \\ \vdots \\ p_{un} \end{pmatrix}$$

$Y^T \in \mathbb{R}^{f \times n}$ $\quad C^u \in \mathbb{R}^{n \times n}$ $\quad p(u) \in \mathbb{R}^n = \begin{cases} 1 & \text{if } r > 0 \\ 0 & \text{if } r = 0 \end{cases}$

$$= \sum_{j: r_{uj} > 0} s_{ij}^u c_{uj} \quad (7) \qquad s_{ij}^u = y_i^T W^u y_j$$

Our Model

5. Explaining recommendations

- 좋은 추천시스템은 왜 추천했는지에 대한 이유를 설명할 수 있어야 함.
- 기존의 Latent factor model (SVD)은 왜 추천하는지에 대한 이유를 잘 설명할 수 없음

- Our model :

$$\underbrace{\hat{p}_{ui}}_{\text{유저 u의 item i에 대한 선호도}} = y_i^T x_u = y_i^T \underbrace{(Y^T C^u Y + \lambda I)^{-1}}_{= W^u \in \Re^{f \times f} \text{ weighing matrix of user } u} Y^T C^u p(u) \quad (6)$$

$$= \sum_{j: r_{uj} > 0} s_{ij}^u \underbrace{c_{uj}}_{\text{유저 u의 item j에 대한 신뢰도}} \quad (7) \qquad \underbrace{s_{ij}^u = y_i^T W^u y_j}_{\text{유저 u의 관점에서 item i,j에 대한 유사도}}$$

1. **Predicted preference 를 설명할 수 있음**
2. **User 맞춤형 similarity를 활용**

Experiment in paper

1. Data

Item: 17,000 TV programs

User: 300,000 set top boxes

r_{ui} : TV프로그램 시청 시간

32 million nonzero values

< TV show watching data >

- training data : 4주 간 데이터
- test data: 그 이후 1주일 간 데이터

데이터 전처리

1. Test set 에서 'easy' prediction 데이터 제거

2. Test set 에서 30분 이하로 시청한 데이터는 0으로 처리

3. r_{ui}의 범위가 크기 때문에 log scaling 필요

$$\rightarrow \quad c_{ui} = 1 + \alpha \log(1 + r_{ui}/\epsilon), \quad \epsilon = 10^{-8}$$

4. 특정 채널을 틀고 다른 일을 하러 가는 경우 등의 Momentum을 보정하기 위해서 down-weight

→ 같은 channel 의 t번째 show의 r_{ui}에 $\frac{e^{-(at-b)}}{1+e^{-(at-b)}}$ 만큼 weight

a=2, b=6 으로 설정

Experiment in paper

2. Evaluation method

(Remind)

1. No negative feedback

- 해당 아이템에 대한 부정적인 행동을 추론하기 어렵다.
- 예를 들어, 유저 A가 Item 1을 구매하지 않았을 때, 선호하지 않아서 사지 않은 것인지, 혹은 상품을 알지 못했기 때문에 구매를 하지 않은 것인지, 확실하지 않다.

⇩

Recall 기반 평가지표 필요

⇩

$\boldsymbol{rank_{ui}}$ **고안**

$$Precision = \frac{TP}{TP + FP}$$

FP 추론 어려움

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

Positive만 고려하면 됨

Experiment in paper

2. Evaluation method

- $\boldsymbol{rank_{ui}}$ **:** percentile-ranking of program i for user u.

ex)

	Item 1	Item 2	Item 3
User A	0.6	0.1	0.3

Recommend value

⇨

	Item 1	Item 2	Item 3
User A	0%	100%	50%

$\boldsymbol{rank_{ui}}$

- Test period 동안 시청 데이터의 Expected percentile ranking으로 평가

$$\overline{rank} = \frac{\sum_{u,i} r^t_{ui} rank_{ui}}{\sum_{u,i} r^t_{ui}} \qquad (8)$$

r^t_{ui} : Test set observation

- $\overline{rank}$가 작을수록(0에 가까울 수록) 모델성능이 좋음.
- 무작위 예측의 경우 $\overline{rank}$ = 50%

Experiment in paper

3. Result #1

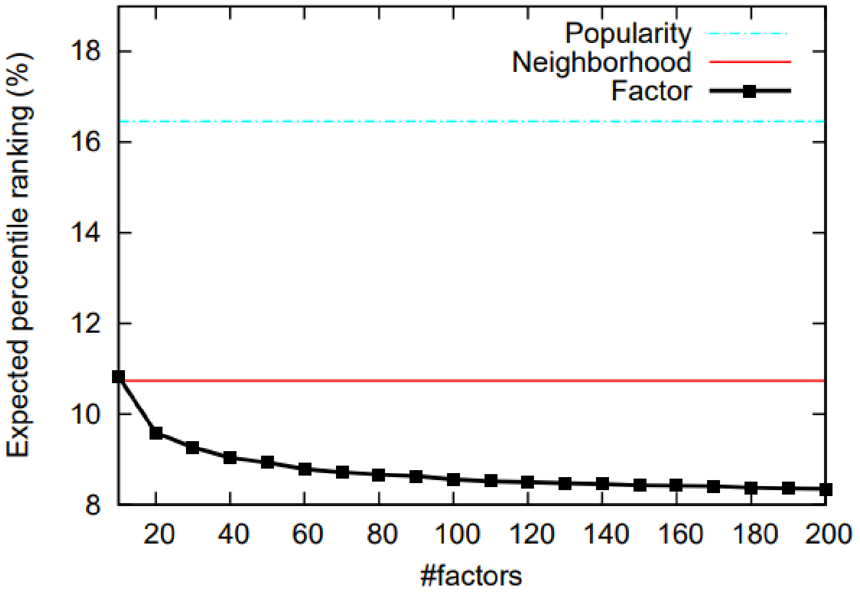


Figure 1. Comparing factor model with popularity ranking and neighborhood model.

- Fig1 shows that measured values of $\overline{rank}$ with different number of factors

- Popularity model: Sorting all items (TV shows) based on their popularity

- Neighborhood model:

$$\hat{r}_{ui} = \frac{\sum_{j \in S^k(i;u)} s_{ij} r_{uj}}{\sum_{j \in S^k(i;u)} s_{ij}}$$

factor model과 차이점

: similarity가 모든 user에 대해 같은 값을 가짐

Experiment in paper

3. Result #2

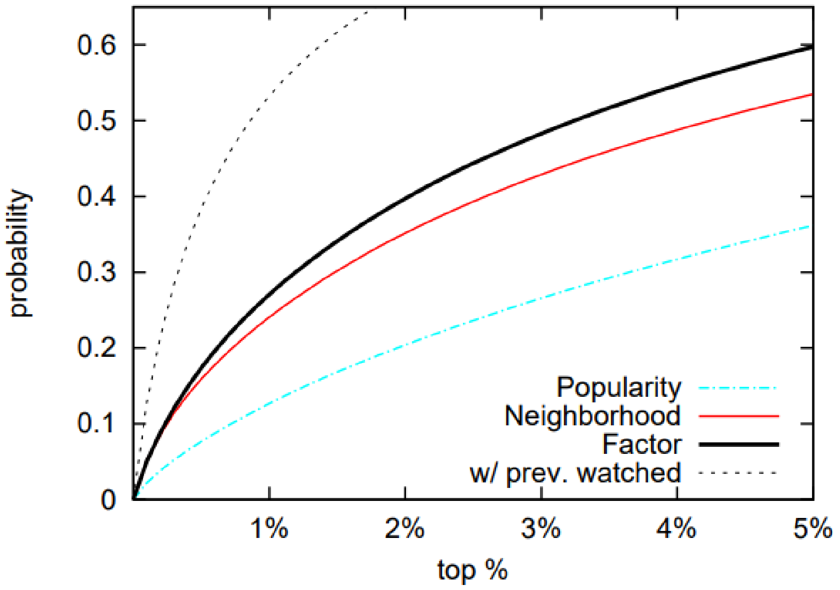


Figure 2. Cumulative distribution function of the probability that a show watched in the test set falls within top x% of recommended shows.

- 그래프는 $rank_{ui}$의 cumulative distribution 을 뜻함.
- x축은 rank top k% 의 아이템을 뜻함
- y축은 추천한 TV show를 유저가 실제로 시청한 비율을 뜻함
- ex) 상위 2% 추천 TV 프로그램을 실제로 시청할 확률 20%
- Quality of recommendation을 평가하는 방법
- Factor model이 가장 우수
- Dotted line은 train set에서 'easy' prediction (already watched TV program) 을 제외하지 않은 것

  → 성능이 훨씬 좋음

Experiment in paper

3. Result #3

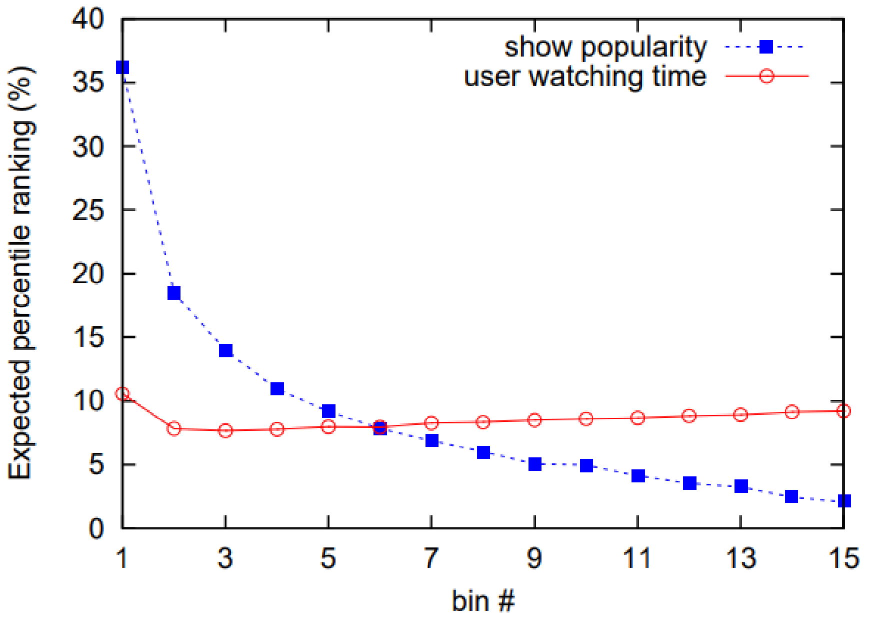


Figure 3. Analyzing the performance of the factor model by segregating users/shows based on different criteria.

Blue line : Item

- Item을 인기가 적은 순서대로 test set을 15개의 equal bins로 분류
  (1번째 bin: popularity 작음, 15번째 bin: popularity 큼)

- **인기가 많은 TV show 일 수록 performance 증가**
- **인기가 많은 TV show 를 추천할 수록 유저가 좋아할 확률이 큼**

Red line : User

- User를 TV 시청시간이 적은 순서대로 test set을 15개의 equal bins로 분류
  (1번째 bin: 시청시간 작음, 15번째 bin: 시청시간 많음)

- **유저의 시청시간과 관계 없이 performance 거의 일정**
- **이유: 하나의 set top box (TV) 로 여러 사람들이 보기 때문**

Experiment in paper

3. Result #4

Original

$$\min_{x_\star, y_\star} \sum_{u,i} \underline{c_{ui}}(\underline{p_{ui}} - x_u^T y_i)^2 + \lambda \left(\sum_u \|x_u\|^2 + \sum_i \|y_i\|^2 \right) \quad (3)$$

Model 1

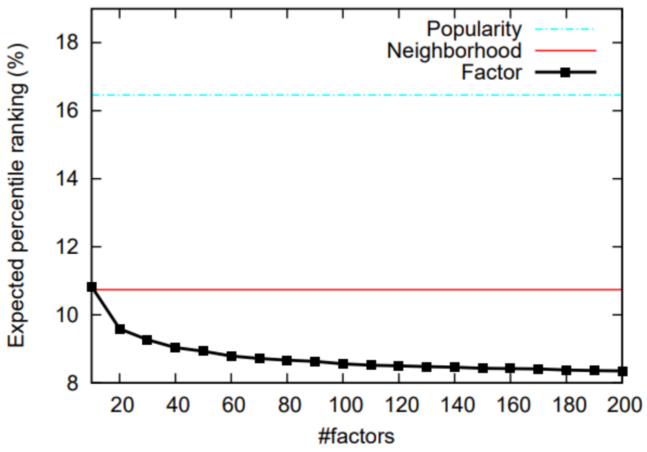
$$\min_{x_\star, y_\star} \sum_{u,i} (\underline{r_{ui}} - x_u^T y_i)^2 + \lambda_1 \left(\sum_u \|x_u\|^2 + \sum_i \|y_i\|^2 \right) \quad (9)$$

- Binary p_{ui} 를 사용하지 않고 r_{ui} 값 그대로 사용, confidence c_{ui} 사용하지 않음
- Regularization 이 없으면 popularity model 보다 성능이 안 좋음
- Regularization 을 진행해도 neighborhood model 보다 성능이 안 좋음

Model 2

$$\min_{x_\star, y_\star} \sum_{u,i} (\underline{p_{ui}} - x_u^T y_i)^2 + \lambda_2 \left(\sum_u \|x_u\|^2 + \sum_i \|y_i\|^2 \right) \quad (10)$$

- Binary p_{ui} 를 사용, confidence c_{ui} 사용하지 않음
- Model1과 neighborhood model 보다 성능이 좋음
- 하지만 여전히 Original model 보다는 성능이 안 좋음



Implement

1. Data

Music play data : last.fm 360K dataset

Item: 294,364 Artist

User: 358,868 music player

r_{ui} : 가수 음악 재생 수

18 million nonzero values

Item: 1000 Artist

User: 2,000 music player

r_{ui} : 가수 음악 재생 수

39,589 nonzero values

Preprocessing

- train : test = 8 : 2
- 계산 편의를 위해 데이터 축소
  - 다양한 artist를 재생한 user 상위 2000명
  - 재생 수가 많은 Artist (item) 상위 1000명
- Momentum 고려 x
- r_{ui} 범위 큼 : (0~400,000)
  → log scaling 필요
  → $c_{ui} = 1 + \alpha \log\left(1 + \frac{r_{ui}}{\epsilon}\right)$ 사용

Implement

2. Modeling

```python
class ImplicitCF:
    def __init__(self, dataframe):
        self.df = dataframe
        self.df['user'] = self.df['user'].astype("category")
        self.df['artist'] = self.df['artist'].astype("category")
        self.df.dropna(inplace=True)

        self.plays = coo_matrix((self.df['plays'].astype(float),
                                (self.df['artist'].cat.codes,
                                self.df['user'].cat.codes)))

    def _alternating_least_squares(self, Rui, factors, regularization, iterations):
        Cui = 40*(Rui/10**-8).log1p()

        artists, users = Cui.shape

        X = np.random.rand(artists, factors)
        Y = np.random.rand(users, factors)

        Ciu = Cui.T.tocsr()
        for iteration in range(iterations):
            self._least_squares(Cui, X, Y, regularization)
            self._least_squares(Ciu, Y, X, regularization)

        return X, Y
```

Data preprocessing

r_{ui} f λ 10

$C_{ui} = \alpha log(1+\frac{r_{ui}}{\varepsilon})$

$x_u = (Y^T C^u Y + \lambda I)^{-1} Y^T C^u p(u)$

$y_i = (X^T C^i X + \lambda I)^{-1} X^T C^i p(i)$

Alternating least square function

Implement

2. Modeling

```python
    def _least_squares(self, Cui, X, Y, regularization):
        artists, factors = X.shape
        YtY = Y.T.dot(Y)

        for u in range(artists):
            # accumulate YtCuY + regularization * I in A
            A = YtY + regularization * np.eye(factors)

            # accumulate YtCuPu in b
            b = np.zeros(factors)

            for i in Cui[u,:].indices:
                confidence = Cui[u,i]
                factor = Y[i]
                A += (confidence - 1) * np.outer(factor, factor)
                b += confidence * factor

            # Xu = (YtCuY + regularization * I)^-1 (YtCuPu)
            X[u] = np.linalg.solve(A, b)

    def factorize(self, factors, regularization, iterations):
        artist_factor, user_factor = self._alternating_least_squares(self.plays.tocsr(), factors, regularization, iterations)
        return artist_factor, user_factor
```

nonzero values

$$\underbrace{(\sum_i C_{ui} y_i y_i^T + \lambda I)}_{A} x_u = \underbrace{\sum_i C_{ui}(p_{ui}) y_i}_{b}$$

$$x_u = (Y^T C^u Y + \lambda I)^{-1} Y^T C^u p(u)$$

Optimization function

Output function

Implement

3. Evaluation function

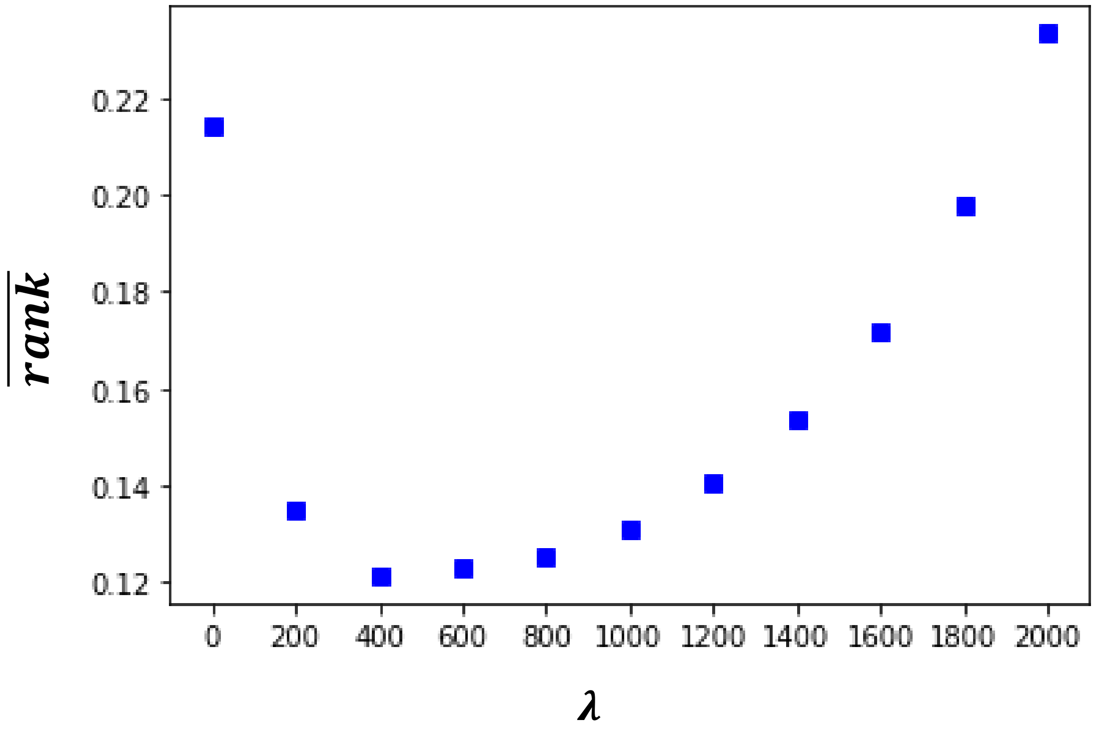
$$\overline{rank} = \frac{\sum_{u,i} r^t_{ui} rank_{ui}}{\sum_{u,i} r^t_{ui}}$$

```
regularize = 400
list_factors = []

for factors in [0, 20, 40, 60, 80, 100 ,120, 140, 160, 180, 200]:
    artist_factor, user_factor = cf_object.factorize(factors,regularize,10)
    cf_object.init_predict(artist_factor)

    array = user_factor.dot(artist_factor.T)
    order = array.argsort()
    ranks = order.argsort()/1998 # max(ranks) = 1998, min(ranks) = 0

    mean_rank = np.multiply(coo_matrix.todense(cf_object_test.plays.T), (1-ranks) ).sum()
    / coo_matrix.todense(cf_object_test.plays.T).sum()

    list_factors.append(mean_rank)
```

$rank_{ui}$

Implement

4. Result



Paper 기본 setting
($factor\ dim = 50,\ iteration = 10$)

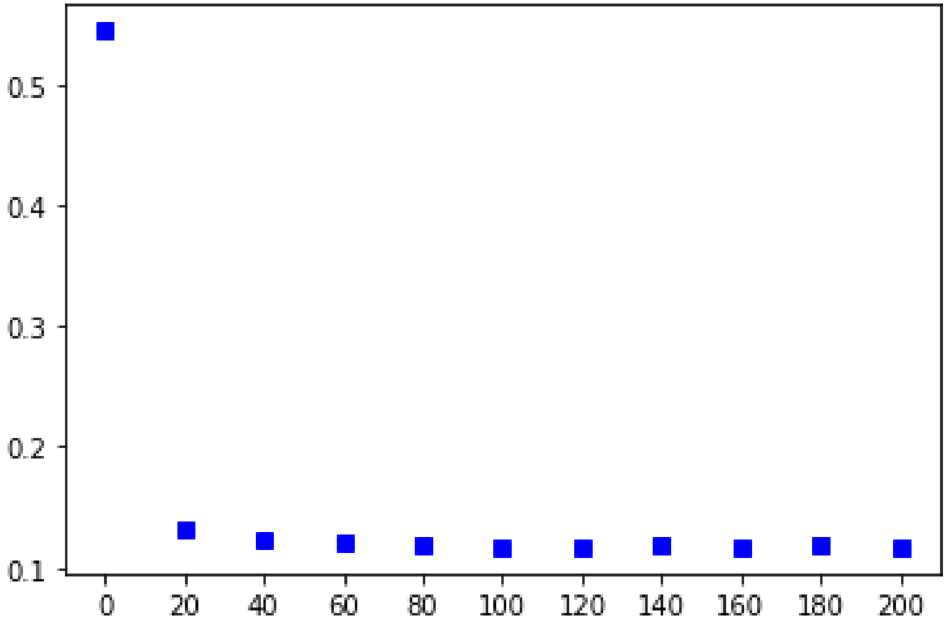
λ = [400, 600] 에서 최적값

λ가 작으면 약간의 overfitting 경향

λ가 커질수록 underfitting 경향

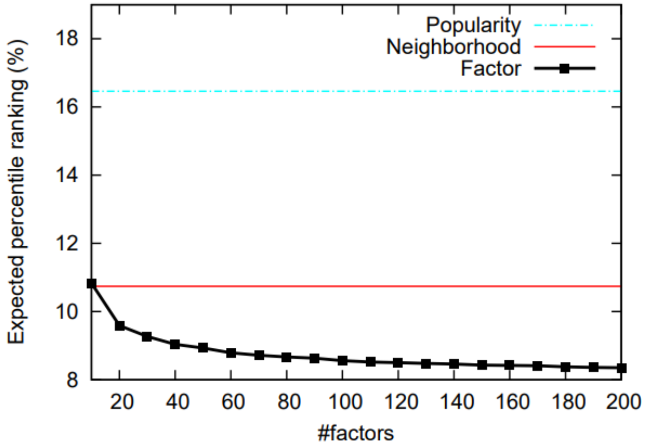
Implement

4. Result



$\overline{rank}$

f
$(latent\ factor\ dim)$



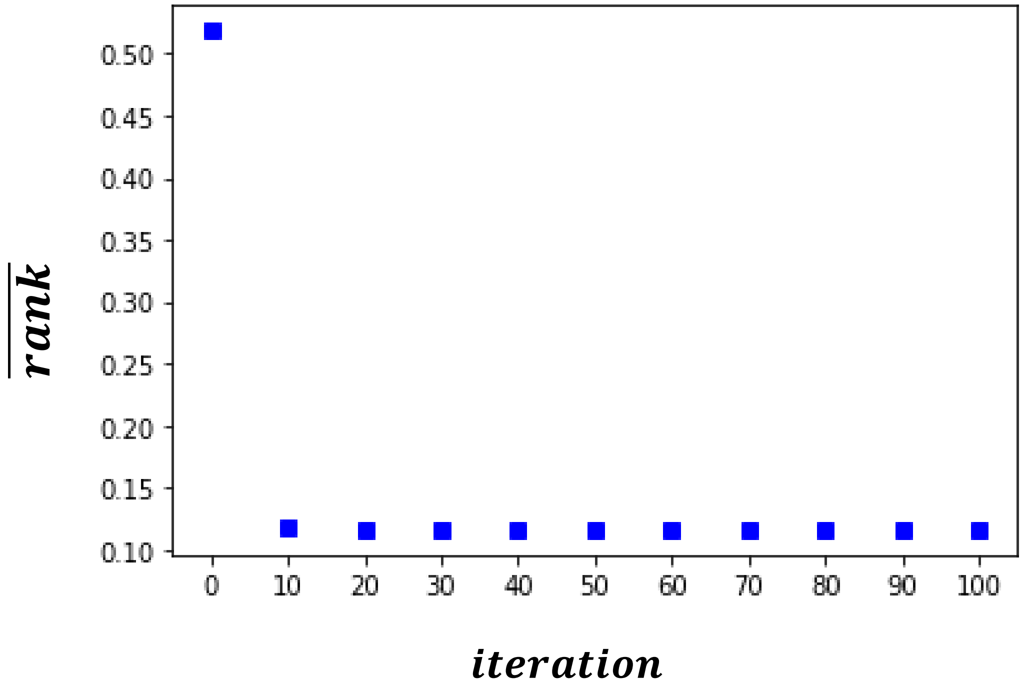
$(\lambda = 50,\ iteration = 10)$

$f = 0$ 일 때 $\overline{rank}$ = 0.5 (Random 모델과 같음)

f 가 커질수록 진동하면서 감소하는 경향

Implement

4. Result



$(\lambda = 50,\ factor\ dim = 100)$

$iteration$ = 0 일 때 $\overline{rank}$ = 0.5 (Random 모델과 같음)

$iteration$ 이 커질수록 진동하면서 감소하는 경향

Thank you