Lec 05. Collaborative Filtering for Implicit Feedback

Contents

---

1. Neighborhoo-based CF

One-class Collaborative Filtering

- Represent users' implicit feedback in a matrix form.
- Q : How to interpret missing values?
- A : Negative or positive-unlabeled

	Item1	Item2	Item3	Item4	Item5
User1	1	?	1	1	?
User2	?	?	1	?	1
User3	1	1	1	?	1
User4	?	1	?	1	?
User5	1	?	1	?	1

Recap : Three Key Questions

- Similarity measurement

- How to calcuate the **similarity** between users or items?
- Neighborhood selection
    - How to **select** similar users or items?
- Prediction function
    - How to **predict the rating** based on neighbors?

similarity measurement를 제외하고는 같다.

Measuring item Similarities

- It does not consider **mean-centered normalization.**
- A simple way to represent **missing values(?)** is to impute them into zeros, i.e., **negative feedback.**

	Item1	Item2	Item3	Item4	Item5
User1	1	0	1	1	0
User2	0	0	1	0	1
User3	1	1	1	0	1
User4	0	1	0	1	0
User5	1	0	1	0	1

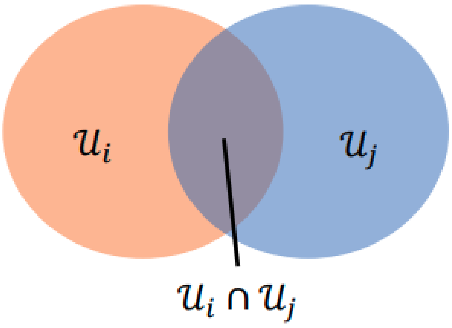
그다음에 여러 similarity 방법을 적용하여 유사도를 측정한다.

Jaccard Coefficient

- Each row and column is represented by a binary vector.
    - For simplicity, it is also represented by **a set of users or items.**
- The similarity between item i and item j is as follows.

$$sim(i,j) = \frac{|\mathcal{U}_i \cap \mathcal{U}_j|}{|\mathcal{U}_i \cup \mathcal{U}_j|}$$

$\mathcal{U}_i$: a set of users who click item i



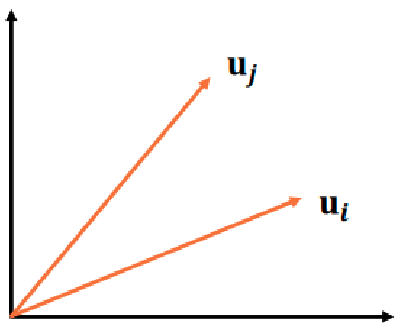
Cosine Similarity

$$sim(i,j) = \cos(\theta) = \frac{\mathbf{u}_i \cdot \mathbf{u}_j}{\|\mathbf{u}_i\|\|\mathbf{u}_j\|} = \frac{|\mathcal{U}_i \cap \mathcal{U}_j|}{\sqrt{c(i)}\sqrt{c(j)}}$$

$$c(x) = \begin{cases} \sum_{i \in \mathcal{I}} r_{xi} & if\ x \in \mathcal{U} \\ \sum_{u \in \mathcal{U}} r_{ux} & if\ x \in \mathcal{I} \end{cases}$$

$c(i)$: the number of ratings for item i

$c(u)$: the number of ratings for user u



Example : Cosine Similarity

➢ Which item is the most similar to item1?

	Item1	Item2	Item3	Item4	Item5
User1	1	0	1	1	0
User2	0	0	1	0	1
User3	1	1	1	0	1
User4	0	1	0	1	0
User5	1	0	1	0	1
		$\frac{1}{\sqrt{3}\sqrt{2}}$	$\frac{3}{\sqrt{3}\sqrt{4}}$	$\frac{1}{\sqrt{3}\sqrt{2}}$	$\frac{2}{\sqrt{3}\sqrt{3}}$

Inverse User Frequency

- Unlike the cosine similarity, **each user has different weights.**

$$sim(i,j) = \frac{|\mathcal{U}_i \cap \mathcal{U}_j|}{\sqrt{c(i)}\sqrt{c(j)}}$$

Each user's weight is equal.

⬇

$$sim(i,j) = \frac{\sum_{u \in \mathcal{U}_i \cap \mathcal{U}_j} \frac{1}{\log(1 + c(u))}}{\sqrt{c(i)}\sqrt{c(j)}}$$

As a user has more ratings,
the user's weight is less significant.

- It is a well-known method in information retrieval.

Example : Inverse User Frequency

Example: Inverse User Frequency

➢ **Which item is the most similar to item1?**

	Item1	Item2	Item3	Item4	Item5	Inverse user frequency
User1	1	0	1	1	0	$1/\log 4$
User2	0	1	0	1	1	$1/\log 4$
User3	1	1	1	0	1	$1/\log 5$
User4	0	1	1	0	1	$1/\log 4$
User5	1	0	0	1	0	$1/\log 3$

$$\frac{1}{\sqrt{3}\sqrt{3}} \quad \frac{2}{\sqrt{3}\sqrt{3}} \quad \frac{2}{\sqrt{3}\sqrt{3}} \quad \frac{\cancel{2}\,1}{\sqrt{3}\sqrt{3}}$$

$$\Rightarrow \frac{\frac{1}{\log \cancel{4}\,5}}{\sqrt{3}\sqrt{3}} \quad \frac{\frac{1}{\log 4}+\frac{1}{\log 5}}{\sqrt{3}\sqrt{3}} \quad \frac{\frac{1}{\log 4}+\frac{1}{\log 3}}{\sqrt{3}\sqrt{3}} \quad \frac{\frac{1}{\log 5}}{\sqrt{3}\sqrt{3}}$$

11

Unified Neighborhood-based Model

➢ **The similarity between item k and item j is as follows.**

$$sim(j,k) = \frac{\sum_{u \in \mathcal{U}_j \cap \mathcal{U}_k} \frac{1}{\log(1+c(u))}}{\sqrt{c(j)}\sqrt{c(k)}}$$

It uses the inverse user frequency.

➢ **The similarity between user u and user w is as follows.**

$$sim(u,w) = \frac{\sum_{i \in \mathcal{I}_u \cap \mathcal{I}_w} \frac{1}{\log(1+c(i))}}{\sqrt{c(u)}\sqrt{c(w)}}$$

It uses the inverse item frequency.

Item-based Prediction

➢ **The prediction rule is as follows.**

$$\hat{r}_{uj} = \sum_{k \in \mathcal{N}_j \cap \mathcal{I}_u} sim(j,k)$$

- $sim(j,k)$ is the similarity between two items j and k.
- $\mathcal{N}_j$ is a set of top-K nearest neighbors of item j.
- $\mathcal{I}_u$ is a set of items by user u.

➢ **Note that we may have $|\mathcal{N}_j \cap \mathcal{I}_u| < K$, but it is acceptable.**

User-based Prediction

➢ **The prediction rule is as follows.**

$$\hat{r}_{uj} = \sum_{w \in \mathcal{N}_u \cap \mathcal{U}_j} sim(u,w)$$

- $sim(u,w)$ is the similarity between two users u and w.
- $\mathcal{N}_u$ is a set of top-K nearest neighbors of user u.
- $\mathcal{U}_j$ is a set of users who click item j.

➢ **Note that we may have $|\mathcal{N}_u \cap \mathcal{U}_j| < K$, but it is acceptable.**

Prediction for the Unified Model

- **Q: How to combine user- and item-based scores?**
- **A: Consider # of ratings for the user and the item.**

Item-based prediction

$$\hat{r}_{uj} = \frac{1}{\sqrt{c(u)}} \sum_{k \in \mathcal{N}_j \cap \mathcal{I}_u} sim(j,k)$$

User-based prediction

$$\hat{r}_{uj} = \frac{1}{\sqrt{c(j)}} \sum_{w \in \mathcal{N}_u \cap \mathcal{U}_j} sim(u,w)$$

	Item1	Item2	Item3	Item4	Item5
User1	1	0	1	1	?
User2	0	1	0	0	1
User3	1	1	1	0	1
User4	0	1	1	1	0
User5	1	0	0	0	0

Prediction for the Unified Model

- For the prediction score of user u on item j,
- The item-based prediction is $$\hat{r}_{uj} = \frac{1}{\sqrt{c(u)}} \sum_{k \in \mathcal{N}_j \cap \mathcal{I}_u} sim(j,k)$$
- The user-based prediction is $$\hat{r}_{uj} = \frac{1}{\sqrt{c(j)}} \sum_{w \in \mathcal{N}_u \cap \mathcal{U}_j} sim(u,w)$$
- The final prediction rule is

$$\hat{r}_{uj} = \frac{1}{\sqrt{c(u)}} \sum_{k \in \mathcal{N}_j \cap \mathcal{I}_u} sim(j,k) + \frac{1}{\sqrt{c(j)}} \sum_{w \in \mathcal{N}_u \cap \mathcal{U}_j} sim(u,w)$$

$\hat{r_{uj}}$ 는 top-N recommendation 용도

---

2. Model-based CF

SVD with Implicit Feedback

- We are only interested in **correct item rankings**, **not caring about exact rating predictions.**
  - All missing values are filled in zeros.
- Simply, apply SVD for top-N recommendations

$$\mathbf{R}_{[m \times n]} = \mathbf{U}_{[m \times k]} \Sigma_{[k \times k]} \left(\mathbf{V}_{[n \times k]}\right)^{T}$$

R: binary matrix, where missing feedback is zero.

Recap : Objective Function for MF

- **Factorize a matrix $\mathbf{R}$ into two latent matrices $\mathbf{U}$ and $\mathbf{V}$.**
  - The matrix $\mathbf{R}$ is approximated as a product of $\mathbf{U}\mathbf{V}^{\mathbf{T}}$.
  - **For simplicity, we consider all missing values as zeros.**

$$\min_{\mathbf{U},\mathbf{V}} \sum_{u=1}^{m}\sum_{i=1}^{n}\left(r_{ui}-\mathbf{u}_u\mathbf{v}_i^{\mathbf{T}}\right)^2$$

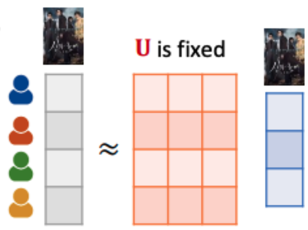
$\mathbf{u}_u$: u-th row of $\mathbf{U}$

$\mathbf{v}_\mathbf{i}$: i–th row of $\mathbf{V}$

Alternating Least Squares (ALS)

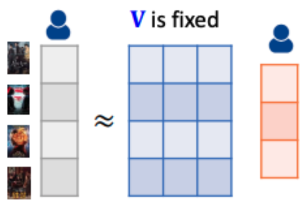
- **Minimize two loss functions alternatively.**
  - Keeping $\mathbf{U}$ fixed, solve for each of the n rows of $\mathbf{V}$.

$$\min_{\mathbf{v}_i}\frac{1}{2}\sum_{u=1}^{m}\left(r_{ui}-\mathbf{u}_u\mathbf{v}_i^{\mathbf{T}}\right)^2+\frac{\lambda}{2}\left(\|\mathbf{v}_i\|^2\right)$$



  - Keeping $\mathbf{V}$ fixed, we solve for each of the m rows of $\mathbf{U}$.

$$\min_{\mathbf{u}_u}\frac{1}{2}\sum_{i=1}^{n}\left(r_{ui}-\mathbf{v}_i\mathbf{u}_u^{\mathbf{T}}\right)^2+\frac{\lambda}{2}\left(\|\mathbf{u}_u\|^2\right)$$



- **It treats a least-squares regression problem.**

Training with ALS

- **Update two matrices $\mathbf{U}$ and $\mathbf{V}$ in parallel.**
  - **$\mathbf{U}$: latent user matrix ($m \times k$ matrix)**
    - Each user is represented by a latent vector ($1 \times k$ vector).
  - **$\mathbf{V}$: latent item matrix ($n \times k$ matrix)**
    - Each item is represented by a latent vector ($1 \times k$ vector).

Randomly initialize two matrices $\mathbf{U}$ and $\mathbf{V}$.

Repeat

For $u = 1, \ldots, m$ **do**

$$\mathbf{u}_u^{\mathbf{T}} = \left(\mathbf{V}^{\mathbf{T}}\mathbf{V} + \lambda\mathbf{I}\right)^{-1}\mathbf{V}^{\mathbf{T}}\mathbf{r}_u^{\mathbf{T}}$$

For $i = 1, \ldots, n$ **do**

$$\mathbf{v}_i^{\mathbf{T}} = \left(\mathbf{U}^{\mathbf{T}}\mathbf{U} + \lambda\mathbf{I}\right)^{-1}\mathbf{U}^{\mathbf{T}}\mathbf{r}_i$$

Until the stopping condition is satisfied

$\mathbf{r}_u$: u-th row of $\mathbf{R}$

$\mathbf{r}_i$: i–th column of $\mathbf{R}$

Missing as Negative Feedback

- All missing feedback is regarded as negative feedback.
- Then, adjust the weight for the confidence of missing values.

$$\min_{\mathbf{U},\mathbf{V}} \sum_{u=1}^{m} \sum_{i=1}^{n} y_{ui} \left(r_{ui} - \mathbf{u}_u \mathbf{v}_i^{\mathbf{T}} \right)^2$$

$$y_{ui} = \begin{cases} 1 & if\ r_{ui} \text{ exists} \\ 0 & \text{otherwise} \end{cases}$$

⬇

$$\min_{\mathbf{U},\mathbf{V}} \sum_{u=1}^{m} \sum_{i=1}^{n} w_{ui} \left(r_{ui} - \mathbf{u}_u \mathbf{v}_i^{\mathbf{T}} \right)^2$$

$$w_{ui} = \begin{cases} 1 & if\ r_{ui} \text{ exists} \\ \alpha & \text{otherwise} \end{cases}$$

$$\alpha \in [0, 1]$$

Weighted Matrix Factorization (WMF)

- **Factorize a matrix $\mathbf{R}$ into two latent matrices $\mathbf{U}$ and $\mathbf{V}$.**
  - The matrix $\mathbf{R}$ is approximated as a product of $\mathbf{U}\mathbf{V}^{\mathbf{T}}$.
  - **Note: we care about missing values.**

$$\min_{\mathbf{U},\mathbf{V}} \sum_{u=1}^{m} \sum_{i=1}^{n} w_{ui} \left(r_{ui} - \mathbf{u}_u \mathbf{v}_i^{\mathbf{T}} \right)^2$$

$$w_{ui} = \begin{cases} 1 & if\ r_{ui} \text{ exists} \\ \alpha & \text{otherwise} \end{cases}$$

$\mathbf{u}_u$: u-th row of $\mathbf{U}$ $\quad$ $\mathbf{v}_i$: i–th row of $\mathbf{V}$

1	1	α	...	1
α	α	1	...	1
1	1	α	...	α
⋮	⋮	⋮	⋮	⋮
1	α	α	...	1

$\mathbf{W}$ $\quad$ $\alpha \in [0, 1]$

1	1	0	...	1
0	0	1	...	1
1	1	0	...	0
⋮	⋮	⋮	⋮	⋮
1	0	0	...	1

$\mathbf{R}$

Weighting Scheme for Missing Values

- How to set the weight matrix W in WMF?
  - **Uniform** : use lower weights to negative samples.

- **User-oriented** : if a user has more positive examples, it is **more likely** that the **user does not like the other item.**
- **Item-oriented** : if an item has fewer positive examples, the missing feedback for this item is **negative with a higher probability.**

	Positive samples	Negative samples
Uniform	$w_{ui} = 1$	$w_{ui} = \alpha$
User-oriented	$w_{ui} = 1$	$w_{ui} \propto \sum_i r_{ui}$
Item-oriented	$w_{ui} = 1$	$w_{ui} \propto m - \sum_i r_{ui}$

$\alpha \in [0, 1]$ m: number of users

Implicit Matrix Factorization

- Approximate the binary preference matrix by minimizing the weighted RMSE.

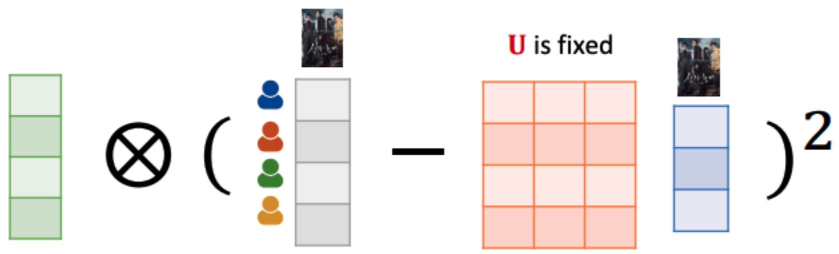
$$\min \frac{1}{2} \sum_{u=1}^{m} \sum_{i=1}^{n} c_{ui} \left(r_{ui} - \mathbf{u}_u \mathbf{v}_i^T \right)^2 + \frac{\lambda}{2} \left(\sum_{u=1}^{m} \|\mathbf{u}_u\|^2 + \sum_{i=1}^{n} \|\mathbf{v}_i\|^2 \right)$$

- $r_{ui} = 1$ if user u clicks item i, else 0.
- $c_{ui} = 1 + \alpha r_{ui}$
  - Positive feedback: $1 + \alpha r_{ui}$
  - Negative feedback: 1

Weighted ALS (wALS)

- **The confidence of predicting each entry is different.**

$$\min_{\mathbf{v}_i} \frac{1}{2}\sum_{u=1}^{m} c_{ui}\left(r_{ui} - \mathbf{u}_u\mathbf{v}_i^{\mathrm{T}}\right)^2 + \frac{\lambda}{2}\left(\|\mathbf{v}_i\|^2\right)$$



⊗: Element-wise product

Training with wALS

- **Update two matrices U and V in parallel.**
  - **U: user matrix ($m \times k$ matrix), V: item matrix ($n \times k$ matrix)**
  - **C: weight matrix corresponding to R**
    - $\tilde{\mathbf{C}}_u \in \mathbb{R}^{n\times n}$ is a diagonal matrix of $\mathbf{C}_{u*} \in \mathbb{R}^{1\times n}$
    - $\tilde{\mathbf{C}}_i \in \mathbb{R}^{m\times m}$ is a diagonal matrix of $\mathbf{C}_{*i} \in \mathbb{R}^{1\times m}$

$$\begin{bmatrix} c_{u1} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & c_{un} \end{bmatrix}$$

Randomly initialize two matrices U and V.

Repeat

For $u = 1, \ldots, m$ do

$$\mathbf{u}_u^{\mathrm{T}} = \left(\mathbf{V}^{\mathrm{T}}\tilde{\mathbf{C}}_u\mathbf{V} + \lambda\mathbf{I}\right)^{-1}\mathbf{V}^{\mathrm{T}}\tilde{\mathbf{C}}_u\mathbf{r}_u^{\mathrm{T}}$$

For $i = 1, \ldots, n$ do

$$\mathbf{v}_i^{\mathrm{T}} = \left(\mathbf{U}^{\mathrm{T}}\tilde{\mathbf{C}}_i\mathbf{U} + \lambda\mathbf{I}\right)^{-1}\mathbf{U}^{\mathrm{T}}\tilde{\mathbf{C}}_i\mathbf{r}_i$$

Until the stopping condition is satisfied

$\mathbf{r}_u$: u-th row of $\mathbf{R}$

$\mathbf{r}_i$: i-th column of $\mathbf{R}$

27

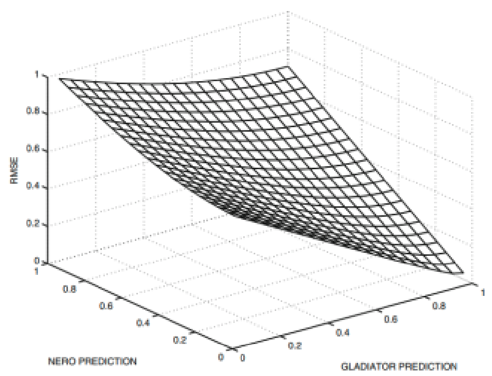
3. Bayesian Personalized Ranking

Limitation of Rating Prediction Models

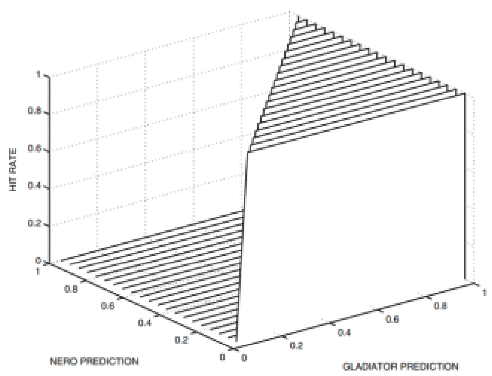
- Existing recommender models solve the recommendation problem as a **rating prediction problem.**
    - The **squared error of rating prediction is optimized.**
    - In practice, only **the top-N items are presented as a ranked list.**
- In many cases, **optimizing rating predictions many not provide the best recommendation lists.**
    - Example : all the low-ranked ratings are predicted very accurately, but high-ranked ratings incur significant errors.
- It arises because **the objective functions of prediction-based methods are not fully aligned with top-N recommendations.**

Ranking Objective Functions

- Minimize the objective function for **ranking evaluation** between R and UV^T.
- Challenge : the objective functions for ranking-based methods are **non-smooth.**
    - Tiny changes can **cause sudden jumps or drops.**
    - It is difficult to optimize with gradient descent techniques.



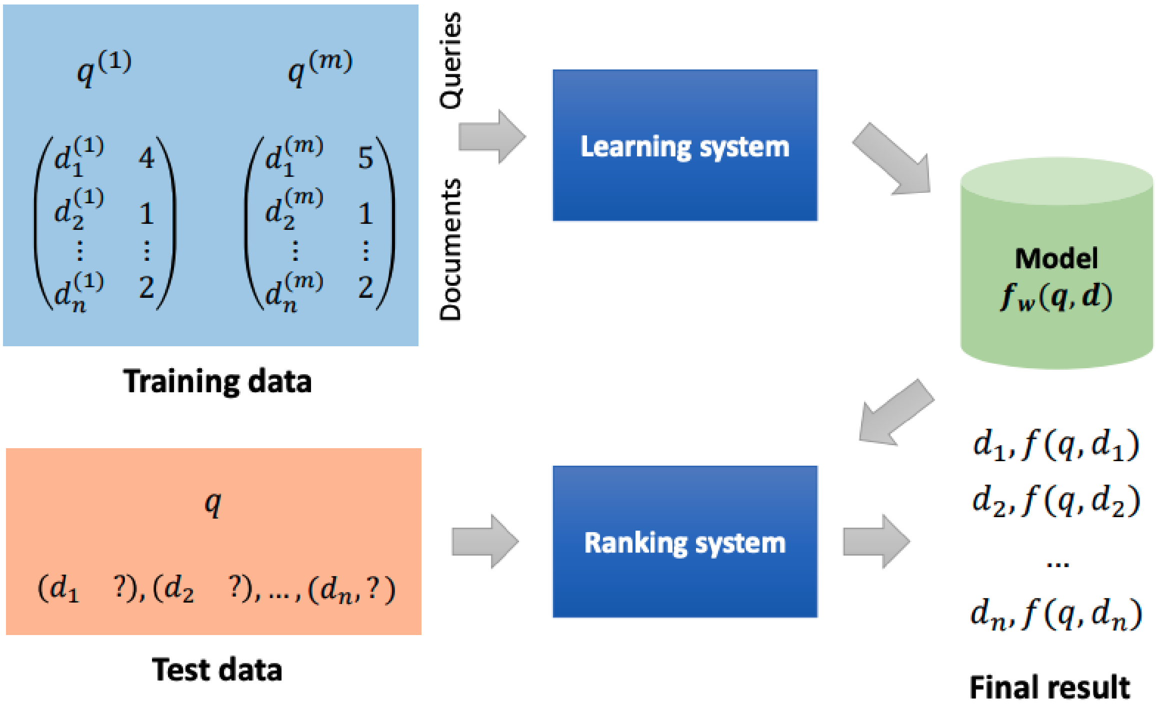
Smooth RMSE objective



Non-smooth hit rate

Learning to Rank

- Ranking is a key part of many IR problems, such as document retrieval, collaborative filtering, and online advertising.

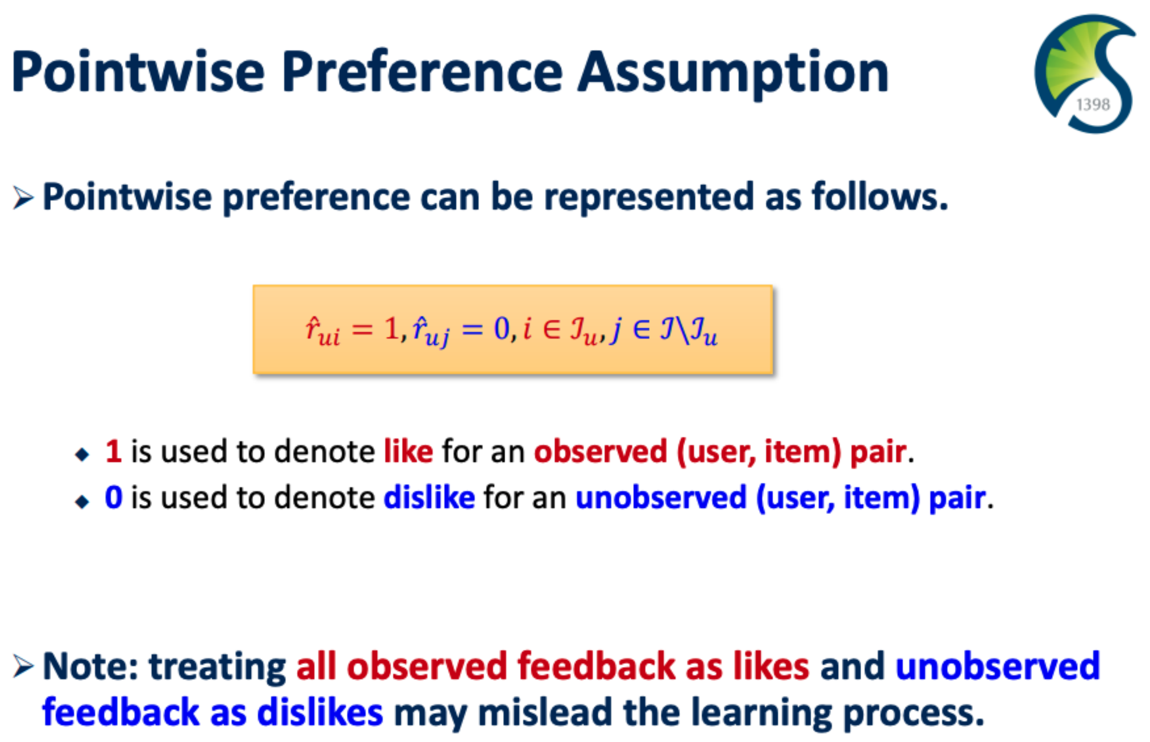


Pointwise Preference

- Alice give ratings for A = 5, B = 4, and C = 3.
- Predict the relevance score for a user-item pair.

(Alice, A)	5
(Alice, B)	4
(Alice, C)	3

(Bob, A)	?
(Carol, B)	?
(David, A)	?



Pairwise Preference

- Alice give ratings for **A = 5**, **B = 4**, and **C = 3**.
- Predict a **relative order** for a triplet of a user and two items.
    - It creates a binary classification problem and minimizes the number of pairwise inversions in the training data.

(Alice, A, B)	$+1$
(Alice, A, C)	$+1$
(Alice, B, A)	-1
(Alice, B, C)	$+1$
(Alice, C, A)	-1
(Alice, C, B)	-1

➡

(Bob, A, B)	?
(Carol, B, A)	?
(David, A, C)	?

$3P_2$ = 6가지

Pairwise Preference Assumption

- Pairwise preferences relax the assumption of pointwise preferences.

$$\hat{r}_{ui} > \hat{r}_{uj} = 0, i \in \mathcal{I}_u, j \in \mathcal{I} \backslash \mathcal{I}_u$$

- The relationship $\hat{r}_{ui} > \hat{r}_{uj}$ means that a user u is likely to prefer an item $i \in \mathcal{I}_u$ to an item $j \in \mathcal{I} \backslash \mathcal{I}_u$.

- Empirically, pairwise preferences helps achieve better results than pointwise preferences.

Prediction Rule

- How to predict the rating of user u in item i and item j?

$$\hat{r}_{ui} = \mathbf{u}_u \mathbf{v}_i^{\mathbf{T}} + b_i$$

$$\hat{r}_{uj} = \mathbf{u}_u \mathbf{v}_j^{\mathbf{T}} + b_j$$

We consider the difference between two predicted ratings.

- Question : Why not use b_u and μ?

  중복되는 값(서로 같은 값)이므로

Likelihood of Pairwise Preferences

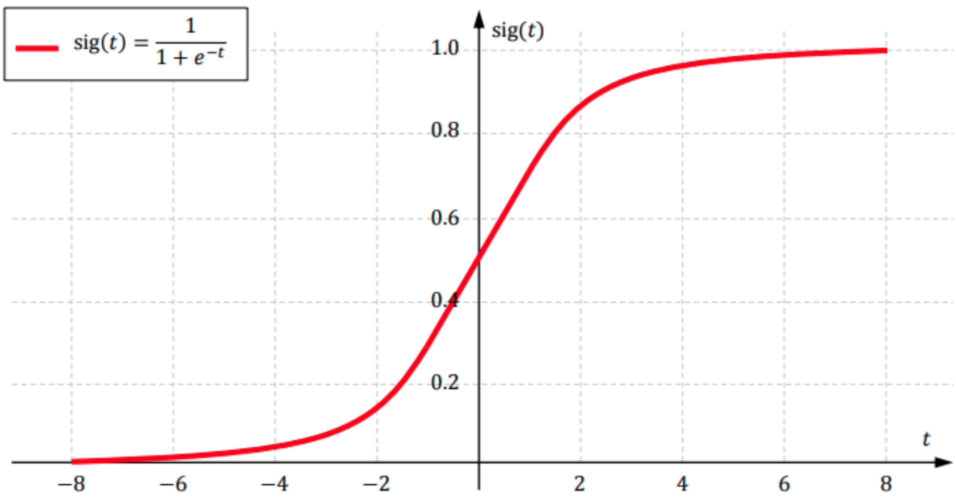
- The Bernoulli distribution of binary random variable $\delta((u,i) \succ (u,j))$ is defined as follows.

$$LPP_u = \prod_{i,j \in \mathcal{I}} P(\hat{r}_{ui} > \hat{r}_{uj})^{\delta((u,i)\succ(u,j))} \left(1 - P(\hat{r}_{ui} > \hat{r}_{uj})\right)^{(1-\delta((u,i)\succ(u,j)))}$$

$$LPP_u = \prod_{(u,i)\succ(u,j)} P(\hat{r}_{ui} > \hat{r}_{uj}) \prod_{(u,i)\preccurlyeq(u,j)} \left(1 - P(\hat{r}_{ui} > \hat{r}_{uj})\right)$$

- $(u,i) \succ (u,j)$ means that user u prefers item i to item j.

- To approximate the probability $P(\hat{r}_{ui} > \hat{r}_{uj})$, we use the sigmoid function $\sigma(\hat{r}_{uij})$ such that $\hat{r}_{uij} = \hat{r}_{ui} - \hat{r}_{uj}$.



$$LPP_u = \prod_{(u,i)\succ(u,j)} p(\hat{r}_{ui} > \hat{r}_{uj}) \times \prod_{(u,i)\preccurlyeq(u,j)} \left(1 - p(\hat{r}_{ui} > \hat{r}_{uj})\right)$$

$$\Big\downarrow \quad p(\hat{r}_{ui} > \hat{r}_{uj}) = \sigma(\hat{r}_{uij})$$

$$\ln LPP_u = \ln \prod_{(u,i)\succ(u,j)} \sigma(\hat{r}_{uij}) + \ln \prod_{(u,i)\preccurlyeq(u,j)} \left(1 - \sigma(\hat{r}_{uij})\right)$$

$$\Big\downarrow \quad \sigma(\hat{r}_{uij}) = \sigma(-\hat{r}_{uji})$$

$$\ln LPP_u = \ln \prod_{(u,i)\succ(u,j)} \sigma(\hat{r}_{uij}) + \ln \prod_{(u,i)\succ(u,j)} \left(1 - \sigma(-\hat{r}_{uji})\right)$$

$$\ln LPP_u = \ln \prod_{(u,i)\succ(u,j)} \sigma(\hat{r}_{uij}) + \ln \prod_{(u,i)\succ(u,j)} \left(1 - \sigma(-\hat{r}_{uji})\right)$$

$$\Big\downarrow \quad 1 - \sigma(-\hat{r}_{uji}) \approx \sigma(\hat{r}_{uij})$$

$$\ln LPP_u = \ln \prod_{(u,i)\succ(u,j)} \sigma(\hat{r}_{uij}) + \ln \prod_{(u,i)\succ(u,j)} \sigma(\hat{r}_{uij})$$

$$\Big\downarrow$$

$$\ln LPP_u = 2 \sum_{(u,i)\succ(u,j)} \ln \sigma(\hat{r}_{uij}) = 2 \sum_{i\in\mathcal{I}_u, j\in\mathcal{I}\setminus\mathcal{I}_u} \ln \sigma(\hat{r}_{uij})$$

Objective Function for BPR

- **Minimize the following function.**

$$\min_{\mathbf{U},\mathbf{V},\mathbf{b}} - \sum_{u \in \mathcal{U}} \sum_{i \in \mathcal{I}_u} \sum_{j \in \mathcal{I} \setminus \mathcal{I}_u} \ln \sigma(\hat{r}_{uij}) + \frac{\lambda}{2} \left(\sum_{u=1}^{m} \|\mathbf{u}_u\|^2 + \sum_{i=1}^{n} \|\mathbf{v}_i\|^2 + \sum_{i=1}^{n} \|b_i\|^2 \right)$$

$$\hat{r}_{uij} = \hat{r}_{ui} - \hat{r}_{uj} = (\mathbf{u}_u \mathbf{v}_i^{\mathbf{T}} + b_i) - (\mathbf{u}_u \mathbf{v}_j^{\mathbf{T}} + b_j)$$

- **Because negative items are too large, we randomly choose negative items.**

Training BPR with GD

- **Use the partial derivative to derive update rules.**

$$\hat{r}_{uij} = (\mathbf{u}_u \mathbf{v}_i^{\mathbf{T}} + b_i) - (\mathbf{u}_u \mathbf{v}_j^{\mathbf{T}} + b_j)$$

$$\nabla \mathbf{u}_u = -\sigma(-\hat{r}_{uij})(\mathbf{v}_i - \mathbf{v}_j) + \lambda \mathbf{u}_u$$

$$\nabla \mathbf{v}_i = -\sigma(-\hat{r}_{uij})\mathbf{u}_u + \lambda \mathbf{v}_i$$

$$\nabla \mathbf{v}_j = -\sigma(-\hat{r}_{uij})\mathbf{u}_u + \lambda \mathbf{v}_j$$

$$\nabla b_i = -\sigma(-\hat{r}_{uij}) + \lambda b_i$$

$$\nabla b_i = -\sigma(-\hat{r}_{uij})(-1) + \lambda b_i$$

Discussion

- The performance of BPR depends on sampling methods.
    - Random sampling

- How to choose negative samples?
    - Choose high-variance samples stored in memory, which achieves efficient sampling of true negatives with hugh quality.