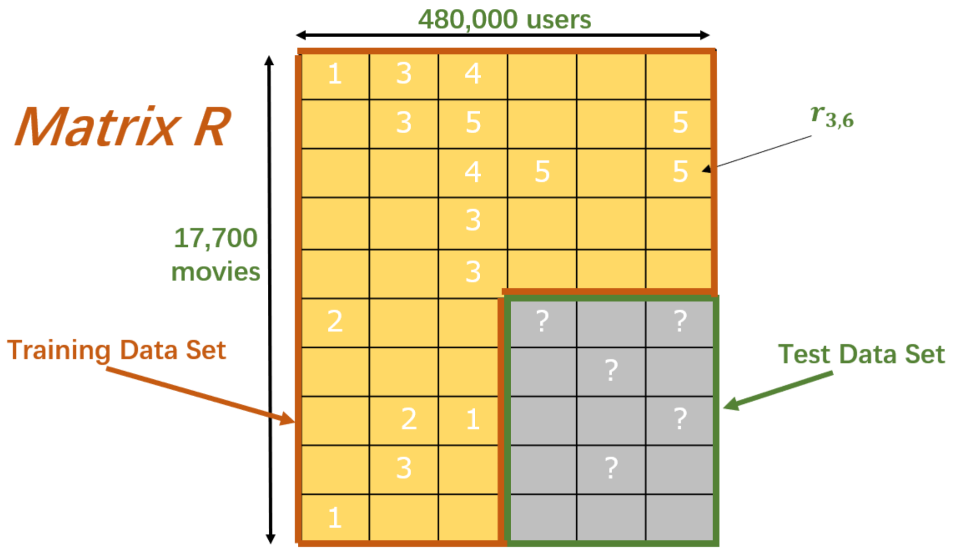
Recommender System

The Netflix Prize



- **Training data**
  - 100 million ratings, 480,000 users, 17,770 movies
  - 6 years of data: 2000-2005
- **Test data**
  - Last few ratings of each user (2.8 million)
  - **Evaluation criterion:** Root Mean Square Error (RMSE)

$$\text{RMSE} = \frac{1}{|R|}\sqrt{\sum_{(i,x)\in R}(\hat{r}_{xi} - r_{xi})^2}$$

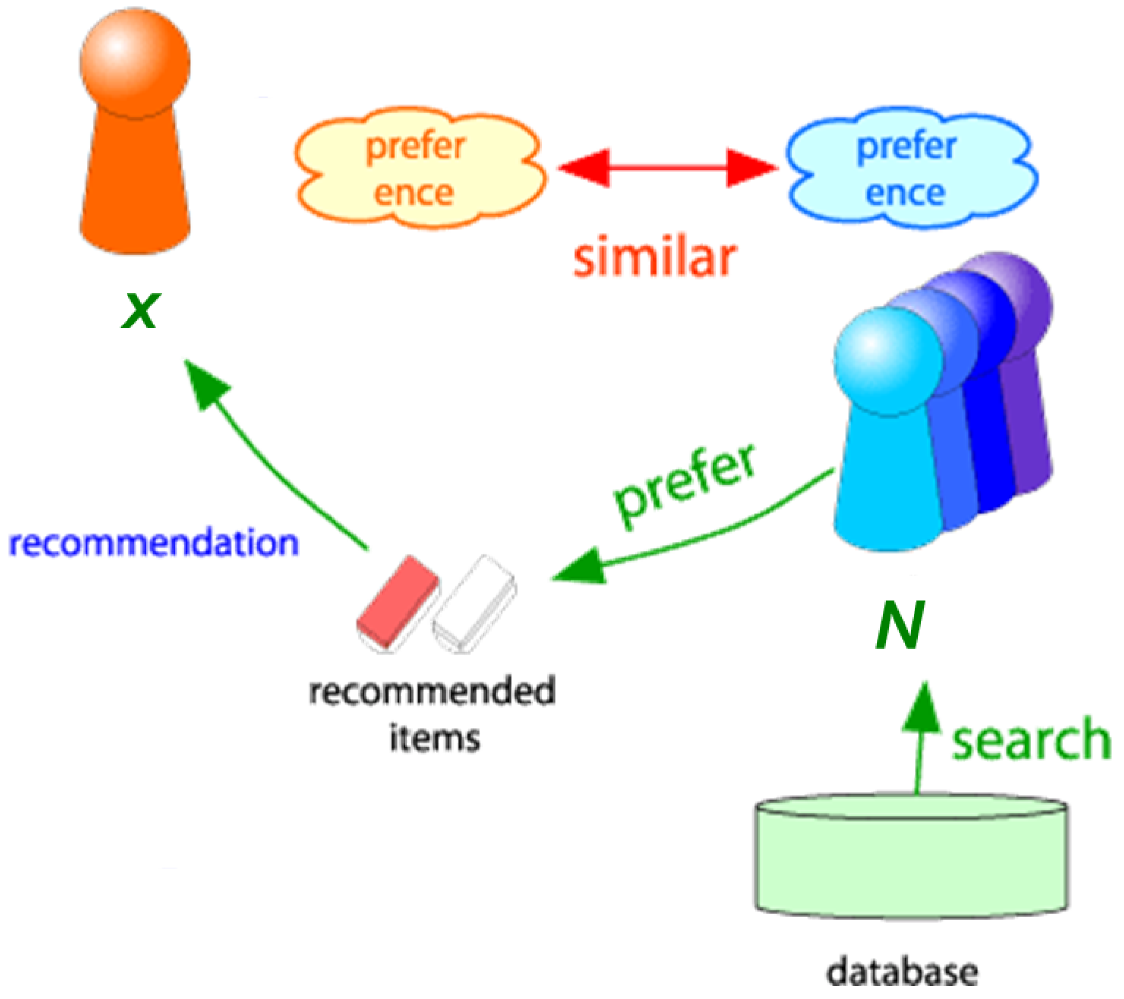
Predicted rating

True rating of user $\boldsymbol{x}$ on item $\boldsymbol{i}$

  - **Netflix's system RMSE: 0.9514**
- **Competition**
  - 2,700+ teams
  - **$1 million** prize for 10% improvement over Netflix
  - Winner improve RMSE to 0.8563

Collaborative Filtering

- Consider user $\boldsymbol{x}$
- Find set $\boldsymbol{N}$ of other users whose ratings are **"similar"** to $\boldsymbol{x}$'s ratings
- Estimate $\boldsymbol{x}$'s ratings based on ratings of users in $\boldsymbol{N}$



Finding "Similar" Users

- Let $\boldsymbol{r_x}$ be the vector of user $\boldsymbol{x}$'s ratings

$\boldsymbol{r_x}$ = [*, _, _, *, ***]
$\boldsymbol{r_y}$ = [*, _, **, **, _]

- **Jaccard similarity measure**
  - Problem: Ignores the value of the rating

$\boldsymbol{r_x}, \boldsymbol{r_y}$ as sets:
$\boldsymbol{r_x}$ = {1, 4, 5}
$\boldsymbol{r_y}$ = {1, 3, 4}

- **Cosine similarity measure**
  - $sim(\boldsymbol{x}, \boldsymbol{y}) = \cos(r_x, r_y) = \frac{r_x \cdot r_y}{||r_x|| \cdot ||r_y||}$
  - **Problem:** Treats missing ratings as "negative"

$\boldsymbol{r_x}, \boldsymbol{r_y}$ as points:
$\boldsymbol{r_x}$ = {1, 0, 0, 1, 3}
$\boldsymbol{r_y}$ = {1, 0, 2, 2, 0}

Similarity Metric

	HP1	HP2	HP3	TW	SW1	SW2	SW3
A	4			5	1		
B	5	5	4				
C				2	4	5	
D		3					3

Is sim(A,B)>sim(A,C) true?

- **Intuitively we want: sim(A, B) > sim(A, C)**
- **Jaccard similarity:** 1/5 < 2/4
- **Cosine similarity:** 0.386 > 0.322
    - Considers missing ratings as “negative”
    - **Solution: subtract the (row) mean**

	HP1	HP2	HP3	TW	SW1	SW2	SW3
A	$2/3$			$5/3$	$-7/3$		
B	$1/3$	$1/3$	$-2/3$				
C				$-5/3$	$1/3$	$4/3$	
D		0					0

sim A,B vs. A,C:
0.092 **>** -0.559

Finding "Similar" Users

- Let $\boldsymbol{r_x}$ be the vector of user $\boldsymbol{x}$'s ratings

$\boldsymbol{r_x}, \boldsymbol{r_y}$ as points:
$\boldsymbol{r_x}$ = {1, 0, 0, 1, 3}
$\boldsymbol{r_y}$ = {1, 0, 2, 2, 0}

- **Pearson correlation coefficient**
  - $\boldsymbol{S_{xy}}$ = items rated by both users $\boldsymbol{x}$ and $\boldsymbol{y}$

$\overline{\boldsymbol{r_x}}, \overline{\boldsymbol{r_y}}$… avg. rating of $\boldsymbol{x}, \boldsymbol{y}$

$$\boldsymbol{sim(x,y)} = \frac{\sum_{s \in S_{xy}} (\boldsymbol{r_{xs}} - \overline{\boldsymbol{r_x}})(\boldsymbol{r_{ys}} - \overline{\boldsymbol{r_y}})}{\sqrt{\sum_{s \in S_{xy}} (\boldsymbol{r_{xs}} - \overline{\boldsymbol{r_x}})^2} \sqrt{\sum_{s \in S_{xy}} (\boldsymbol{r_{ys}} - \overline{\boldsymbol{r_y}})^2}}$$

Rating Predictions

- **From similarity metric to recommendations:**
- Let $\boldsymbol{r_x}$ be the vector of user $\boldsymbol{x}$'s ratings
- Let $\boldsymbol{N}$ be the set of $\boldsymbol{k}$ users most similar to $\boldsymbol{x}$ who have rated item $\boldsymbol{i}$
- **Prediction for item $\boldsymbol{s}$ of user $\boldsymbol{x}$:**
  - $r_{xi} = \frac{1}{k} \sum_{y \in N} r_{yi}$
  - $r_{xi} = \frac{\sum_{y \in N} s_{xy} \cdot r_{yi}}{\sum_{y \in N} s_{xy}}$

Shorthand:
$\boldsymbol{s_{xy} = sim(x, y)}$

Item-Item Collaborative Filtering

- **Another view: Item-item**
  - For item $\boldsymbol{i}$, find other similar items
  - Estimate rating for item $\boldsymbol{i}$ based on ratings for similar items
  - Can use same similarity metrics and prediction functions as in user-user model

$$r_{xi} = \frac{\sum_{j \in N(i;x)} s_{ij} \cdot r_{xj}}{\sum_{j \in N(i;x)} s_{ij}}$$

$\boldsymbol{s_{ij}}$… similarity of items $\boldsymbol{i}$ and $\boldsymbol{j}$
$\boldsymbol{r_{xj}}$…rating of user $\boldsymbol{x}$ on item $\boldsymbol{j}$
$\boldsymbol{N(i;x)}$… set items rated by $\boldsymbol{x}$ similar to $\boldsymbol{i}$

Item-Item CF (|N|=2)

users

movies	1	2	3	4	5	6	7	8	9	10	11	12
1	1		3			5			5		4	
2			5	4			4			2	1	3
3	2	4		1	2		3		4	3	5	
4		2	4		5			4			2	
5			4	3	4	2					2	5
6	1		3		3			2			4	

☐ - unknown rating   ■ - rating between 1 to 5

Item-Item CF (|N|=2)

users

movies	1	2	3	4	**5**	6	7	8	9	10	11	12
1	1		3		?	5			5		4	
2			5	4			4			2	1	3
3	2	4		1	2		3		4	3	5	
4		2	4		5			4			2	
5			4	3	4	2					2	5
6	1		3		3			2			4	

- estimate rating of movie **1** by user **5**

Item-Item CF (|N|=2)

users

movies	1	2	3	4	**5**	6	7	8	9	10	11	12	sim(1,m)
1	1		3		?	5			5		4		1.00
2			5	4			4			2	1	3	-0.18
3	2	4		1	2		3		4	3	5		0.41
4		2	4		5			4			2		-0.10
5			4	3	4	2					2	5	-0.31
6	1		3		3			2			4		0.59

Neighbor selection:
Identify movies similar to movie **1**, **rated by user 5**

Here we use Pearson correlation as similarity:

1) Subtract mean rating m_i from each movie i

$m_1 = (1+3+5+5+4)/5 = 3.6$

row 1: [-2.6, 0, -0.6, 0, 0, 1.4, 0, 0, 1.4, 0, 0.4, 0]

2) Compute cosine similarities between rows

Item-Item CF (|N|=2)

users

movies	1	2	3	4	5	6	7	8	9	10	11	12	sim(1,m)
1	1		3		?	5			5		4		1.00
2			5	4			4			2	1	3	-0.18
3	2	4		1	2		3		4	3	5		0.41
4		2	4		5			4			2		-0.10
5			4	3	4	2					2	5	-0.31
6	1		3		3			2			4		0.59

Compute similarity weights:

$s_{1,3}=0.41$, $s_{1,6}=0.59$

Item-Item CF (|N|=2)

users

movies	1	2	3	4	5	6	7	8	9	10	11	12
1	1		3		**2.6**	5			5		4	
2			5	4			4			2	1	3
3	2	4		1	2		3		4	3	5	
4		2	4		5			4			2	
5			4	3	4	2					2	5
6	1		3		3			2			4	

Predict by taking weighted average:

$r_{1.5}$ = (0.41*2 + 0.59*3) / (0.41+0.59) = 2.6

$$r_{ix} = \frac{\sum_{j \in N(i;x)} s_{ij} \cdot r_{jx}}{\sum s_{ij}}$$

CF: Common Practice

- Define **similarity** s_{ij} of items i and j
- Select k nearest neighbors $N(i; x)$
  - Items most similar to i, that were rated by x
- Estimate rating r_{xi} as the weighted average:

$$r_{xi} = b_{xi} + \frac{\sum_{j \in N(i;x)} s_{ij} \cdot (r_{xj} - b_{xj})}{\sum_{j \in N(i;x)} s_{ij}}$$

Before:

$$r_{xi} = \frac{\sum_{j \in N(i;x)} s_{ij} r_{xj}}{\sum_{j \in N(i;x)} s_{ij}}$$

baseline estimate for r_{xi}

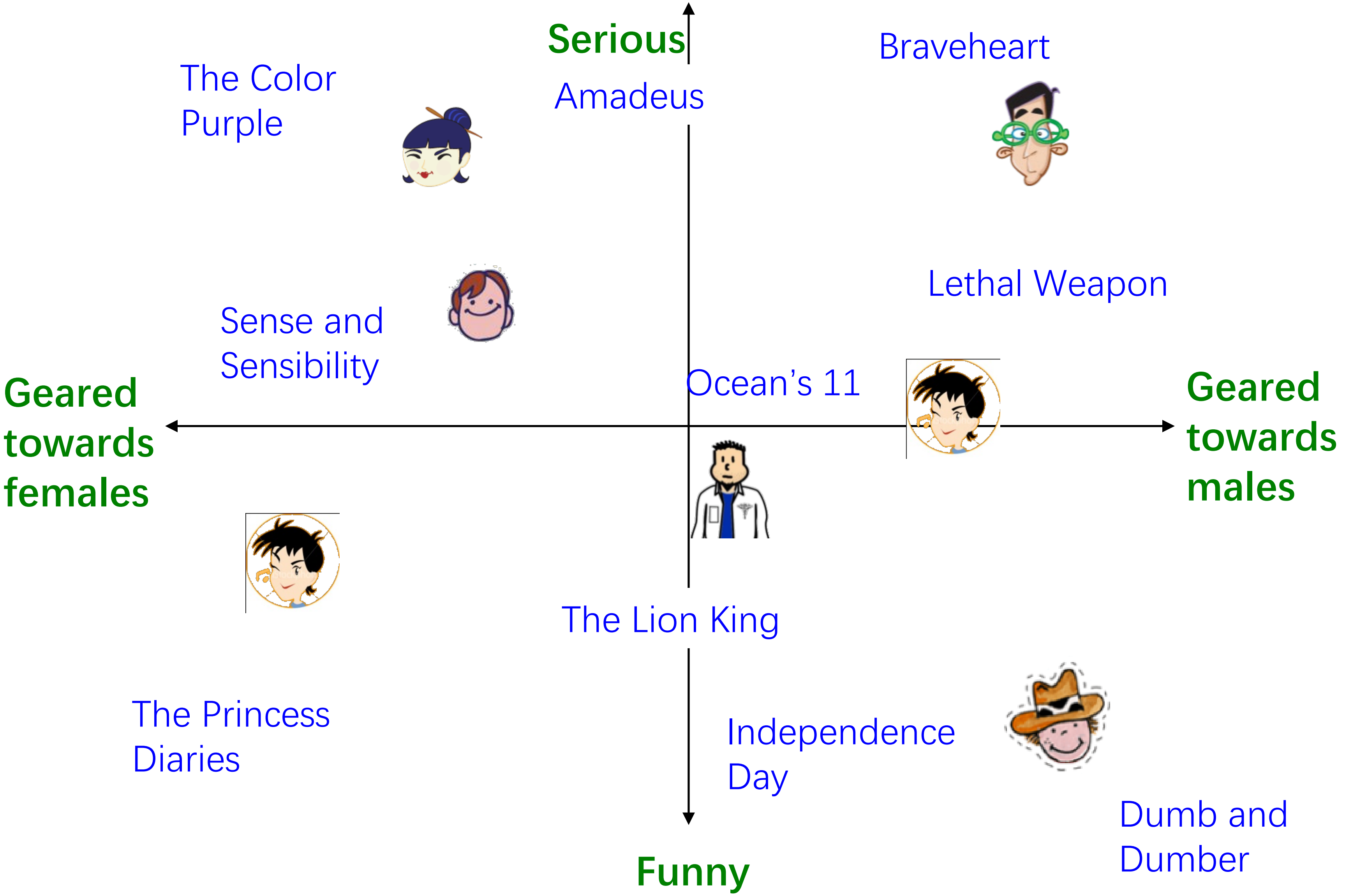
$$b_{xi} = \mu + b_x + b_i$$

μ = overall mean movie rating
b_x = rating deviation of user x
= *(avg. rating of user* x*)* $- \mu$
b_i = rating deviation of movie i

Pros/Cons of Collaborative Filtering

- **+ Works for any kind of item**
  - No feature selection needed
- **- Sparsity:**
  - If there is not enough users in the system, it is hard to find a match
  - If the user/ratings matrix is sparse, it is hard to find users that have rated the same items
  - Cannot recommend an item that has not been previously rated
- **- Popularity bias:**
  - Cannot recommend items to someone with unique taste
  - Tends to recommend popular items

Latent Factor Models (SVD)



Latent Factor Models

- "SVD" on Netflix data: $\boldsymbol{R} \approx \boldsymbol{Q} \cdot \boldsymbol{P}^T$

SVD: $A = U\Sigma V^T$

R (rows: items; columns: users)

1		3			5			5		4	
		5	4			4			2	1	3
2	4		1	2		3		4	3	5	
	2	4		5			4			2	
		4	3	4	2					2	5
1		3		3			2			4	

$\approx$

Q (rows: items; columns: factors)

.1	-.4	.2
-.5	.6	.5
-.2	.3	.5
1.1	2.1	.3
-.7	2.1	-2
-1	.7	.3

$\boldsymbol{P}^T$ (rows: factors; columns: users)

1.1	-.2	.3	.5	-2	-.5	.8	-.4	.3	1.4	2.4	-.9
-.8	.7	.5	1.4	.3	-1	1.4	2.9	-.7	1.2	-.1	1.3
2.1	-.4	.6	1.7	2.4	.9	-.3	.4	.8	.7	-.6	.1

- **Notice** $\boldsymbol{R}$ has missing entries

Latent Factor Models

users / items

1		3			5			5		4	
		5	4			4			2	1	3
2	4		1	2		3		4	3	5	
	2	4		5			4			2	
		4	3	4	2					2	5
1		3		3			2			4	

≈

factors / items

.1	-.4	.2
-.5	.6	.5
-.2	.3	.5
1.1	2.1	.3
-.7	2.1	-2
-1	.7	.3

Q

users / factors

1.1	-.2	.3	.5	-2	-.5	.8	-.4	.3	1.4	2.4	-.9
-.8	.7	.5	1.4	.3	-1	1.4	2.9	-.7	1.2	-.1	1.3
2.1	-.4	.6	1.7	2.4	.9	-.3	.4	.8	.7	-.6	.1

P^T

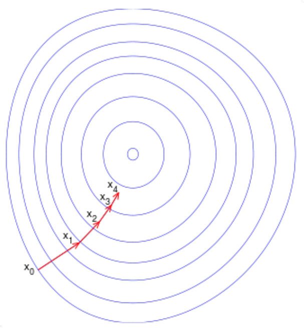
- SVD isn't defined when entries are missing! Use specialized methods to find P, Q:
  - $\min_{P,Q} \sum_{(i,x)\in \mathrm{R}} (r_{xi} - q_i \cdot p_x)^2$

$\hat{r}_{xi} = q_i \cdot p_x$

- **Add regularizer to avoid overfitting:**

$$\min_{P,Q} \sum_{training} (r_{xi} - q_i p_x)^2 + \left[\lambda_1 \sum_x \|p_x\|^2 + \lambda_2 \sum_i \|q_i\|^2 \right]$$

- **Stochastic Gradient decent to solve**



Ratings as Products of Factors

- **How to estimate the missing rating of user x for item i?**

$$\hat{r}_{xi} = q_i \cdot p_x = \sum_f q_{if} \cdot p_{xf}$$

q_i = row i of Q
p_x = column x of P^T

items

1		3			5			5		4	
		5	4	?		4			2	1	3
2	4		1	2		3		4	3	5	
	2	4		5			4			2	
		4	3	4	2					2	5
1		3		3			2			4	

≈

factors

items

.1	-.4	.2
-.5	.6	.5
-.2	.3	.5
1.1	2.1	.3
-.7	2.1	-2
-1	.7	.3

Q

•

users

factors

1.1	-.2	.3	.5	-2	-.5	.8	-.4	.3	1.4	2.4	-.9
-.8	.7	.5	1.4	.3	-1	1.4	2.9	-.7	1.2	-.1	1.3
2.1	-.4	.6	1.7	2.4	.9	-.3	.4	.8	.7	-.6	.1

P^T

Ratings as Products of Factors

Compared to collaboration filtering, latent factor models are space efficient and extensible, achieving better performance

- **How to estimate the missing rating of user x for item i?**

$$\hat{r}_{xi} = q_i \cdot p_x = \sum_f q_{if} \cdot p_{xf}$$

q_i = row i of Q
p_x = column x of P^T

items

1		3			5			5		4	
		5	4	2.4		4			2	1	3
2	4		1	2		3		4	3	5	
	2	4		5			4			2	
		4	3	4	2					2	5
1		3		3			2			4	

$\approx$

f factors

items

.1	-.4	.2
-.5	.6	.5
-.2	.3	.5
1.1	2.1	.3
-.7	2.1	-2
-1	.7	.3

Q

$\bullet$

users

f factors

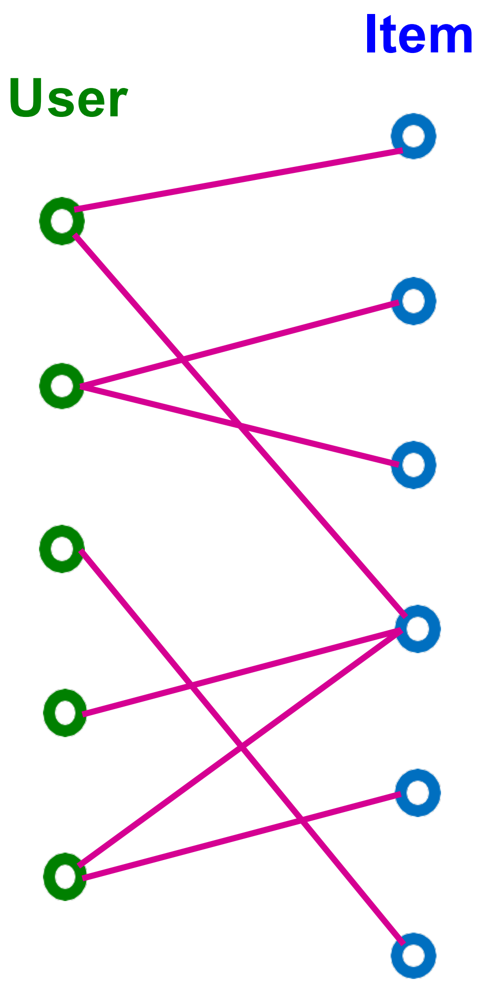
1.1	-.2	.3	.5	-2	-.5	.8	-.4	.3	1.4	2.4	-.9
-.8	.7	.5	1.4	.3	-1	1.4	2.9	-.7	1.2	-.1	1.3
2.1	-.4	.6	1.7	2.4	.9	-.3	.4	.8	.7	-.6	.1

P^T

GNN for Recommender System

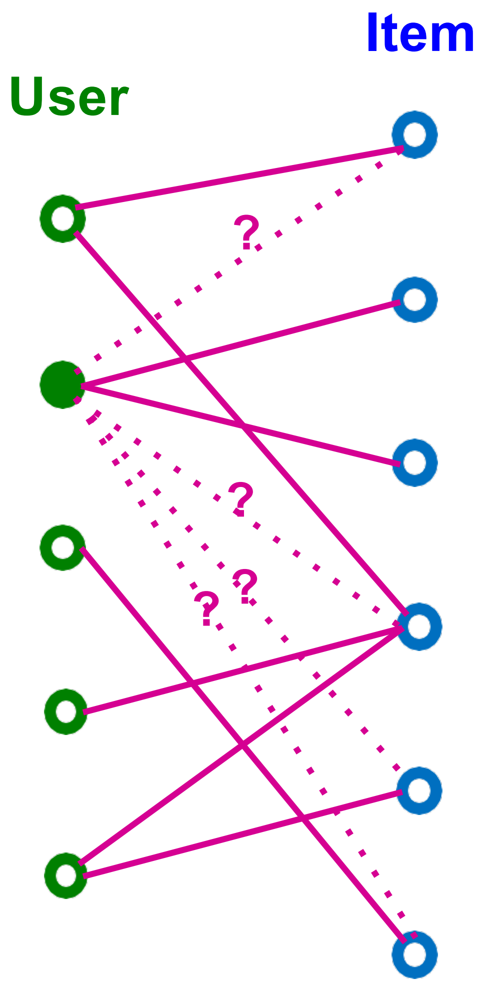
Recommender System as a Graph

- Recommender system can be naturally modeled as a **bipartite graph**
  - A graph with two node types: **users** and **items**.
  - **Edges** connect users and items
    - Indicates user-item interaction (e.g., click, purchase, review etc.)
    - Often associated with timestamp (timing of the interaction).



Recommendation Task

- **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.
  - For $u \in \boldsymbol{U}$, $v \in \boldsymbol{V}$, we need to get a **score** $f(u, v)$.

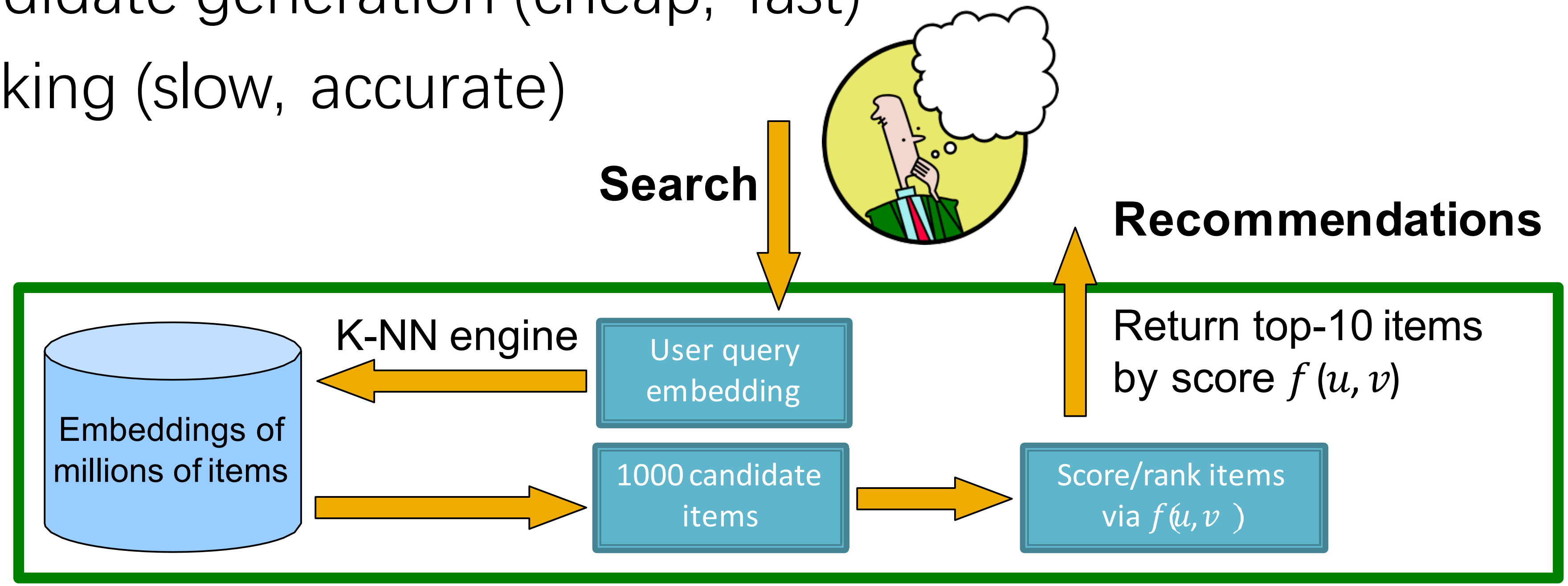


Modern Recommender System

- **Problem:** Cannot evaluate $f(u, v)$ for every user u – item v pair.

  Example $f(u, v)$: $f(u, v) = z_u \cdot z_v$

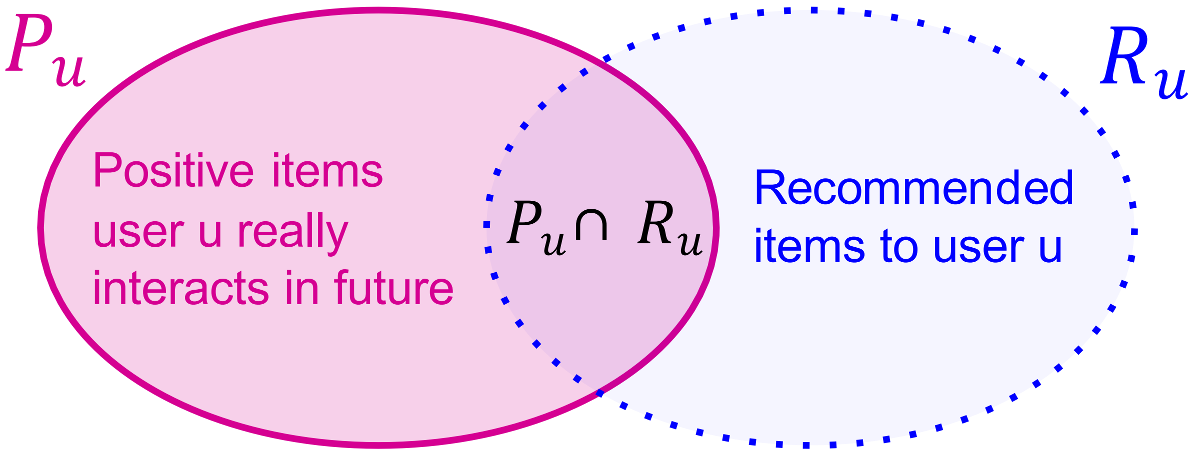
- **Solution:** 2-stage process:
  - Candidate generation (cheap, fast)
  - Ranking (slow, accurate)



Top-K Recommendation

- For each user, we recommend K items.
  - K is typically in the order of 10—100.
- The goal is to include as many **positive items(user interacts in future)** as possible in the top-K recommended items.
- **Evaluation metric**: Recall@K **for user $\boldsymbol{u}$ is**

$$|\boldsymbol{P}_{\boldsymbol{u}} \cap \boldsymbol{R}_{\boldsymbol{u}}| / |\boldsymbol{P}_{\boldsymbol{u}}|.$$



Training Objective

- Embedding-based models have:
  - An encoder to generate user embeddings $\boldsymbol{u}$
  - An encoder to generate item embeddings $\boldsymbol{v}$
  - Score function $f(\cdot,\cdot)$
- Objective: optimize the recall@K
- Two **surrogate loss functions**:
  - Binary loss
  - Bayesian Personalized Ranking (BPR) loss

Binary Loss in Recommender System

- Define **positive**/**negative** **edges**
  - A set of **positive edges** $\boldsymbol{E}$
  - A set of **negative edges** $\boldsymbol{E}_{\mathbf{neg}} = \{(u, v) \mid (u, v) \notin E, u \in \boldsymbol{U}, v \in \boldsymbol{V}\}$
- **Binary loss**: Binary classification of **positive**/**negative** edges using $\sigma(f_\theta(\boldsymbol{u}, \boldsymbol{v}))$:

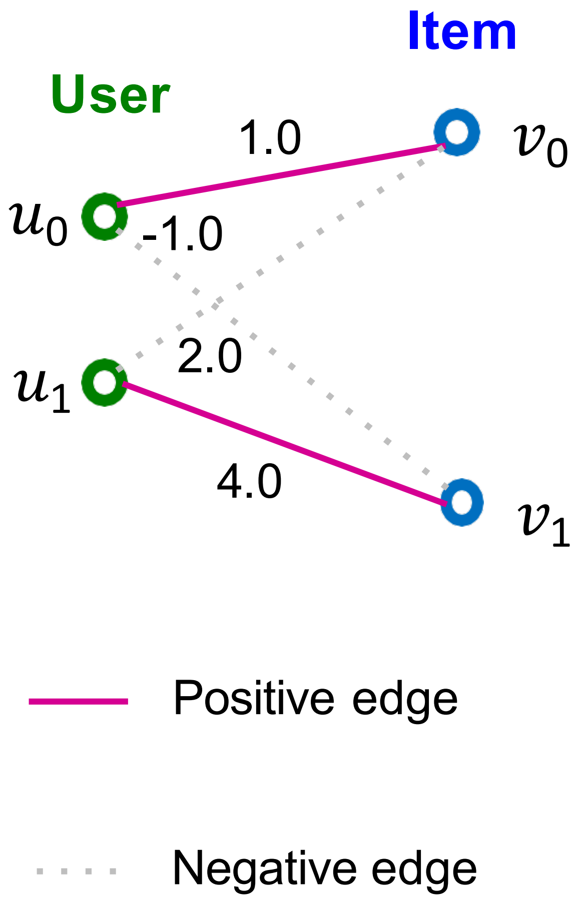
$$-\frac{1}{|\boldsymbol{E}|} \sum_{(u,v) \in E} \log(\sigma(f_\theta(\boldsymbol{u}, \boldsymbol{v}))) - \frac{1}{|E_{\text{neg}}|} \sum_{(u,v) \in E_{\text{neg}}} \log(1 - \sigma(f_\theta(\boldsymbol{u}, \boldsymbol{v})))$$

$\sigma(\cdot)$ is sigmoid function.

Issue with Binary Loss

- **In the binary loss, the scores of all positive edges are pushed higher than those of all negative edges.**
- **The recall metric is inherently personalized**

- **Let's consider the simplest case:**
  - Two users, two items
  - Metric: Recall@1.
  - A model assigns the score for every user-item pair.
- Training **Recall@1 is 1.0**, because v_0 (resp. v_1) is correctly recommended to u_0.
- However, **the binary loss would still penalize the model prediction** because the negative (u_1, v_0) edge gets the higher score than the positive edge (u_0, v_0).



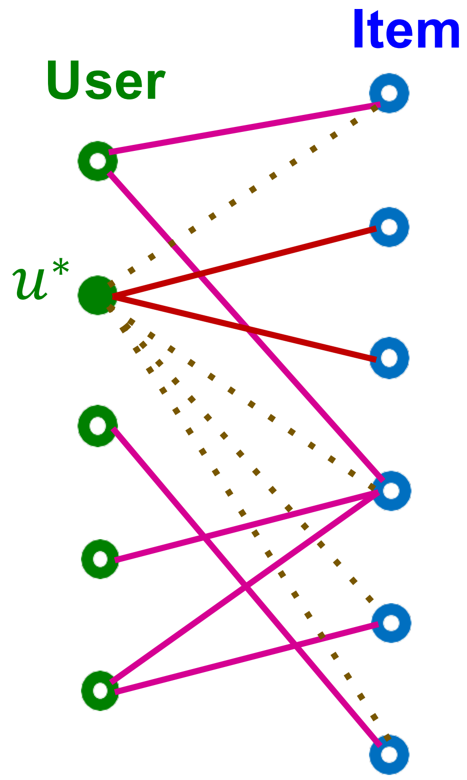
Loss Function: BPR Loss

- **Bayesian Personalized Ranking (BPR) loss** is a personalized surrogate loss that aligns better with the recall@K metric.
- For each user $u^* \in \boldsymbol{U}$, define the **rooted positive/negative edges** as
  - Positive edges rooted at u^*

    $$E\,(u^*) \equiv \{\,(u^*, v\,)\mid(u^*, v\,) \in E\,\}$$

  - Negative edges rooted at u^*

    $$E_{neg}\,(u\,*) \equiv \{(u\,*, v)\,|(u", v) \in E_{neg}\}$$



Rendle, S., Freudenthaler, C., Gantner, Z., & Schmidt-Thieme, L. (2009). Bpr: bayesian personalized ranking from implicit feedback. Proceedings of the twenty-fifth conference on uncertainty in artificial intelligence

Loss Function: BPR Loss (2)

- **Training objective: For each user u^***, we want the scores of rooted positive edges $\boldsymbol{E}(u^*)$ to be higher than those of rooted negative edges $\boldsymbol{E}_{\mathbf{neg}}(u^*)$.

- **BPR Loss for user $\boldsymbol{u}^*$:**

$$\mathrm{Loss}(u^*) = \frac{1}{|\boldsymbol{E}(u^*)| \cdot |\boldsymbol{E}_{\mathbf{neg}}(u^*)|} \sum_{(u^*, v_{\mathrm{pos}}) \in \boldsymbol{E}(u^*)} \sum_{(u^*, v_{\mathrm{neg}}) \in \boldsymbol{E}_{\mathbf{neg}}(u^*)} -\log\left(\sigma\left(\overbrace{f_\theta(\boldsymbol{u}^*, \boldsymbol{v}_{\mathrm{pos}}) - f_\theta(\boldsymbol{u}^*, \boldsymbol{v}_{\mathrm{neg}})}^{\text{Positive is relatively higher than negative}}\right)\right)$$

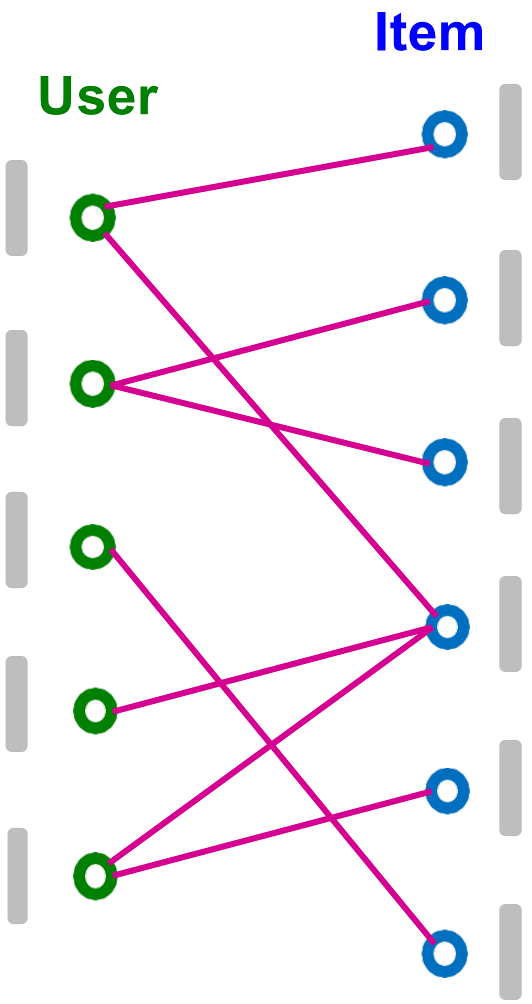
- Final BPR Loss:

$$\frac{1}{|\boldsymbol{U}|} \sum_{u^* \in \boldsymbol{U}} \mathrm{Loss}(u^*)$$

Neural Graph Collaborative Filtering

Conventional Collaborative Filtering

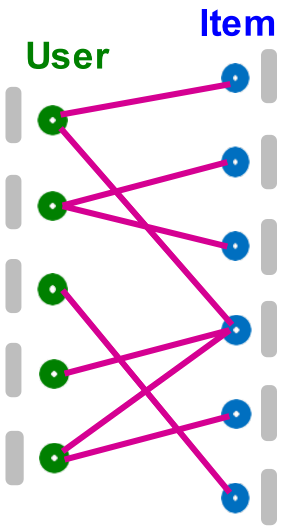
- Conventional collaborative filtering model is based on **shallow encoders**:
  - No user/item features.
  - Use shallow encoders for users and items
  - Only first order structures are captured
- GNNs are natural approach to improve!



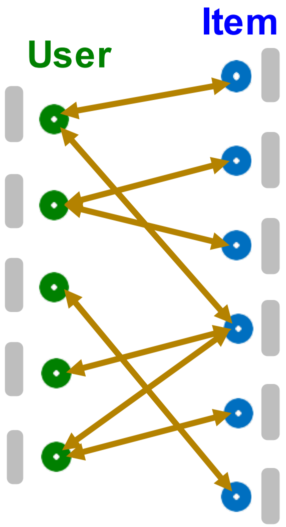
Learnable shallow user/item embeddings

NGCF: Overview

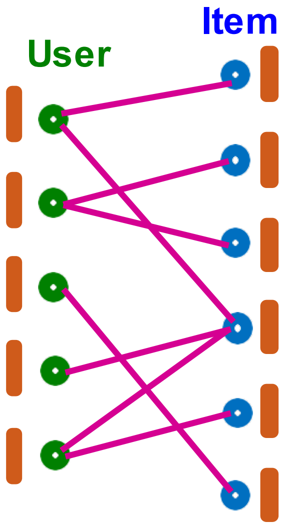
- **Neural Graph Collaborative Filtering (NGCF)** *explicitly* incorporates high-order graph structure when generating user/item embeddings.
- **Key idea**: Use a GNN to generate graph-aware user/item embeddings.



Initial shallow embeddings
(not graph-aware)



Use a GNN to propagate embeddings

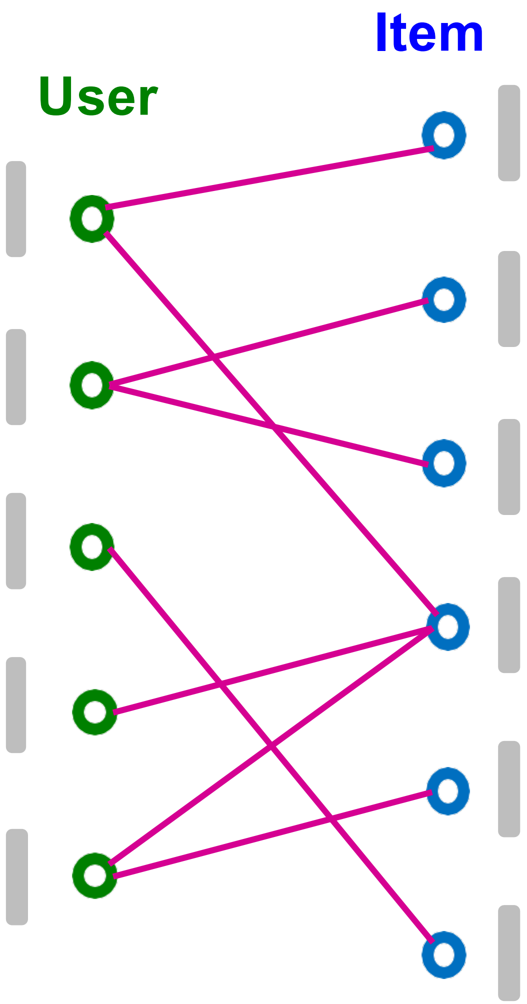


NGCF's graph-aware embeddings

Wang, X., He, X., Wang, M., Feng, F., & Chua, T. S. (2019, July). Neural graph collaborative filtering. In Proc. of *SIGIR 2019.*

Initial Node Embeddings

- Set the shallow learnable embeddings as the initial node features:
  - For every user $u \in U$, set $h_u^{(0)}$ as the user's shallow embedding.
  - For every item $v \in V$, set $h^{(0)}$ as the item's shallow embedding.
- **Two kinds of learnable params are jointly learned:**
  - Shallow user/item embeddings
  - GNN's parameters



Learnable shallow user/item embeddings

Neighbor Aggregation

- Iteratively update node embeddings using neighboring embeddings.

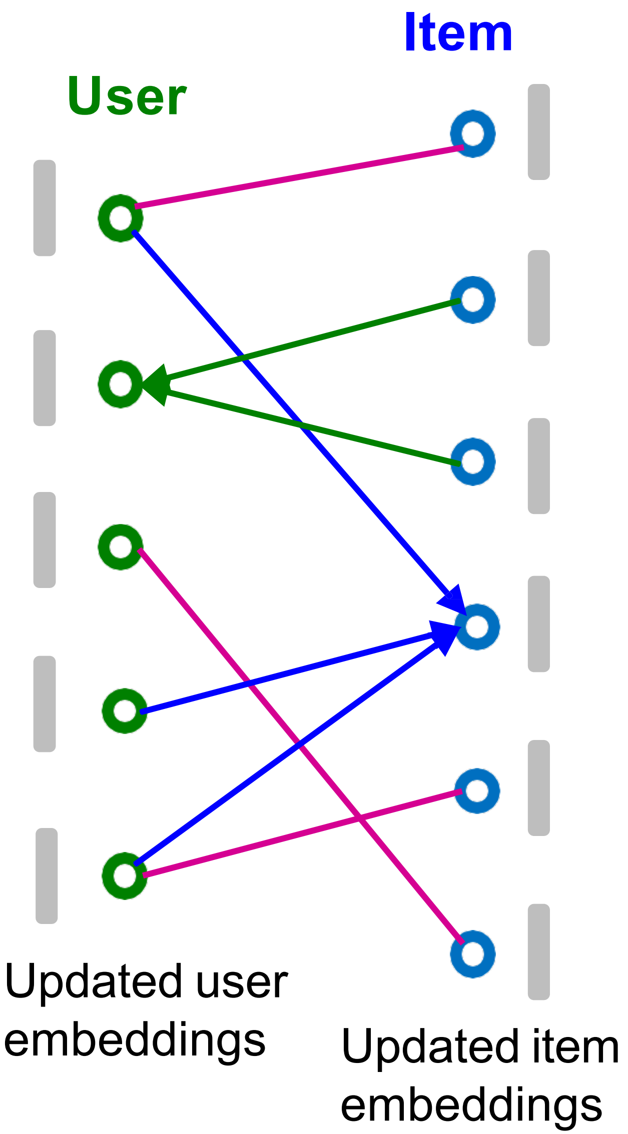
$$\boldsymbol{h}_v^{(k+1)} = \text{COMBINE}\left(\boldsymbol{h}_v^{(k)}, \text{AGGR}\left(\left\{\boldsymbol{h}_u^{(k)}\right\}_{u \in N(v)}\right)\right)$$

$$\boldsymbol{h}_u^{(k+1)} = \text{COMBINE}\left(\boldsymbol{h}_u^{(k)}, \text{AGGR}\left(\left\{\boldsymbol{h}_v^{(k)}\right\}_{v \in N(u)}\right)\right)$$

High-order graph structure is captured through iterative neighbor aggregation.

Different architecture choices are possible for AGGR and COMBINE.

- AGGR$(\cdot)$ can be MEAN $(\cdot)$
- COMBINE$(\boldsymbol{x}, \boldsymbol{y})$ can be ReLU (Linear(Concat$(\boldsymbol{x}, \boldsymbol{y})$))



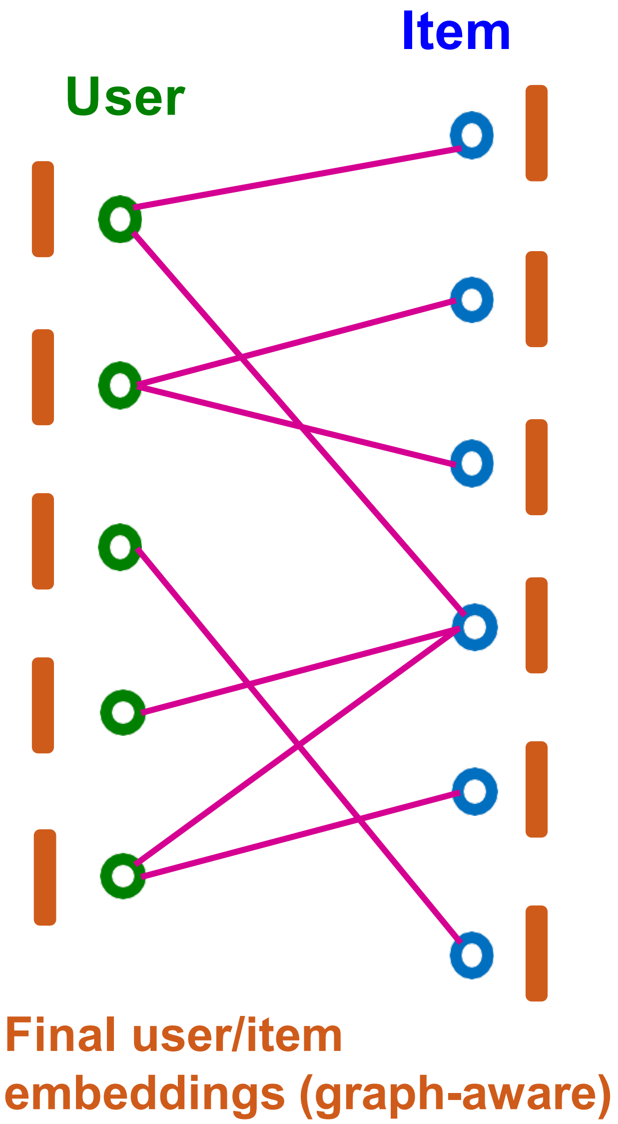
Final Embeddings and Score Function

- After K rounds of neighbor aggregation, we get the **final user/item embeddings** $h_\mu^{(K)}, h_v^{(K)}$
- For all $u \in \boldsymbol{U}$, $v \in \boldsymbol{V}$, we set

$$\mu \leftarrow h_\mu^{(0)} || \cdots || h_\mu^{(K)}$$

- Score function is the inner product

$$\text{score}(u, v) = \boldsymbol{u}^T \boldsymbol{v}$$



LightGCN

LightGCN

- **Can we simplify the GNN used in NGCF (e.g., remove its learnable parameters)?**
  - **Answer**: Yes!
  - **Bonus**: Simplification improves the recommendation performance!
  - Simplification of GCN by removing non-linearity

He, Xiangnan et al. "Lightgcn: Simplifying and powering graph convolution network for recommendation." In *Proceedings of SIGIR 2020*

Simplifying GCN

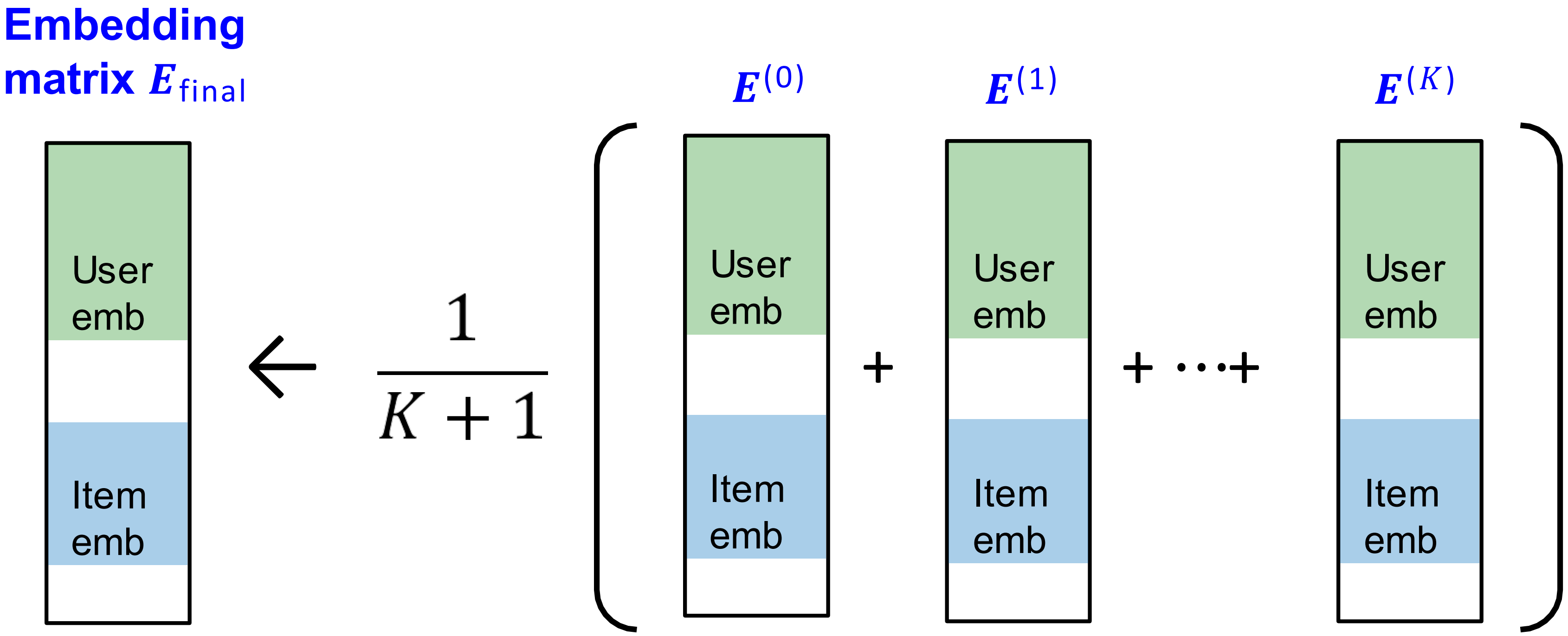
- Removing ReLU significantly simplifies GCN!

$$\boldsymbol{E}^{(K)} = \boxed{\widetilde{\boldsymbol{A}}^{K}\ \boldsymbol{E}}\ \boldsymbol{W} \qquad \boldsymbol{W} \equiv \boldsymbol{W}^{(0)} \dots \boldsymbol{W}^{(K-1)}$$

- **Algorithm**: Apply $\boldsymbol{E} \leftarrow \tilde{A}E$ for K times.
  - Each matrix multiplication diffuses the current embeddings to their one-hop neighbors.
  - **Note:** $\widetilde{\boldsymbol{A}}^{\boldsymbol{K}}$ is dense and never gets materialized. Instead, the above iterative matrix-vector product is used to compute
    - $\widetilde{\boldsymbol{A}}^{\boldsymbol{K}}\boldsymbol{E}$

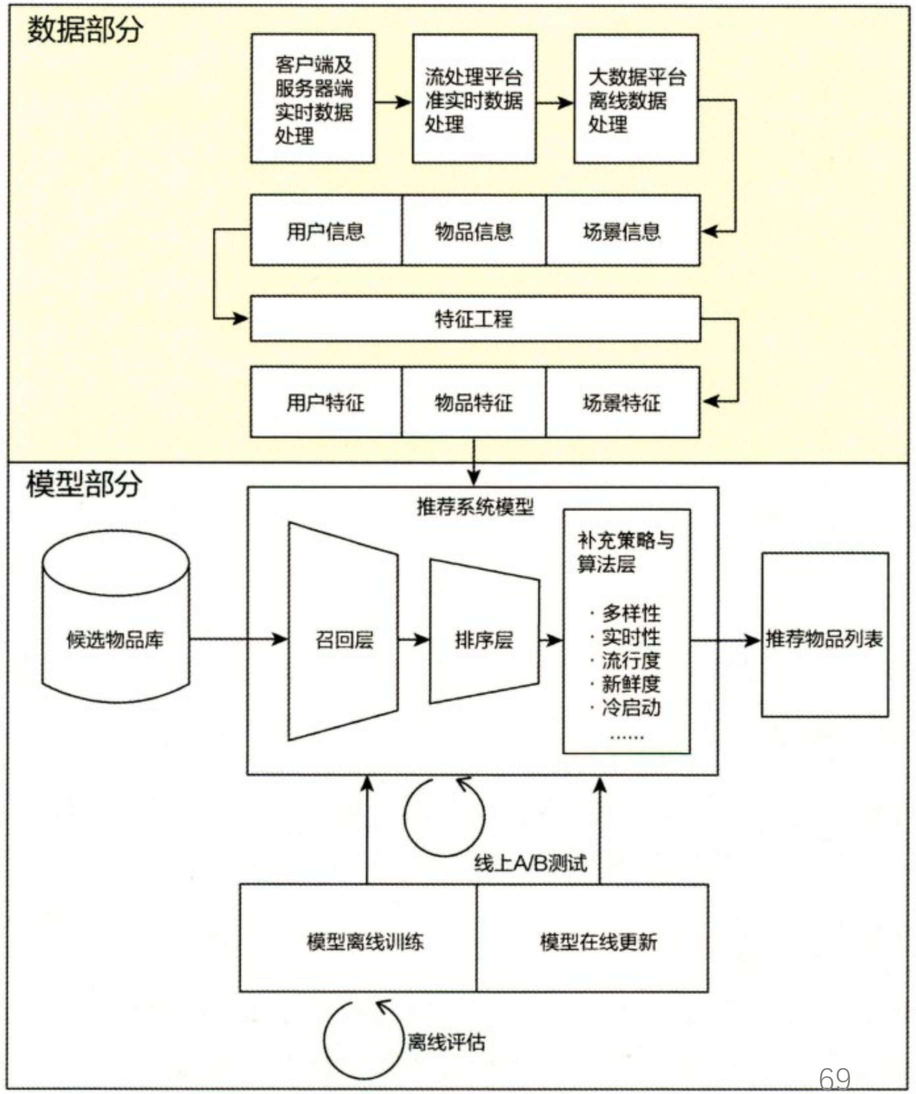
LightGCN

- Average the embedding matrices at different scales.



A Typical Modern Recommender System

- Data Processing
  - Streaming
  - Offline batch: MapReduce, Spark
  - Embedding of user, item, context
    - Machine learning models applied
- Model
  - Recall: e.g. LSH
  - Sort



Summary

- SVD
  - Dimension reduction
  - Find the latent factor

- Recommender system
  - Latent factor recommendation
  - GNN based approach