Recommender Systems

Mohsen Moghaddam, Ph.D.

Gary C. Butler Family Associate Professor
H. Milton Stewart School of Industrial and Systems Engineering
George W. Woodruff School of Mechanical Engineering
Georgia Institute of Technology

SAIL
Symbiotic & Augmented Intelligence Laboratory

Spring 2026

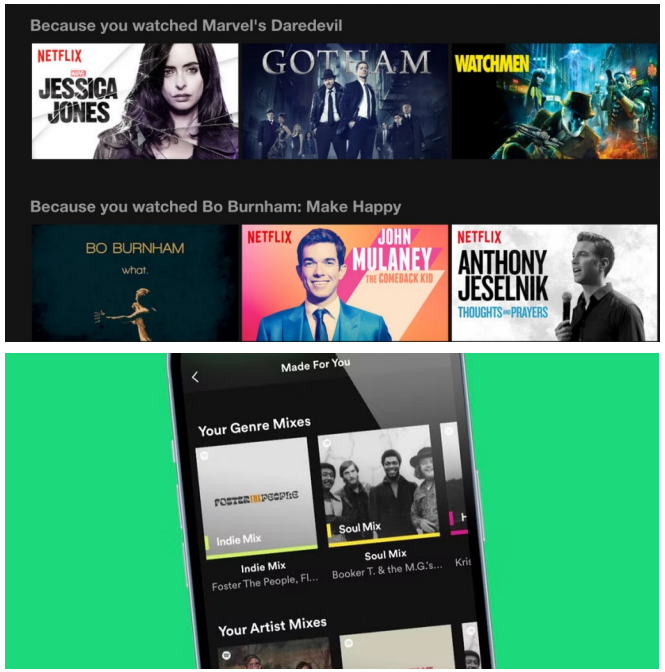
Learning Outcomes

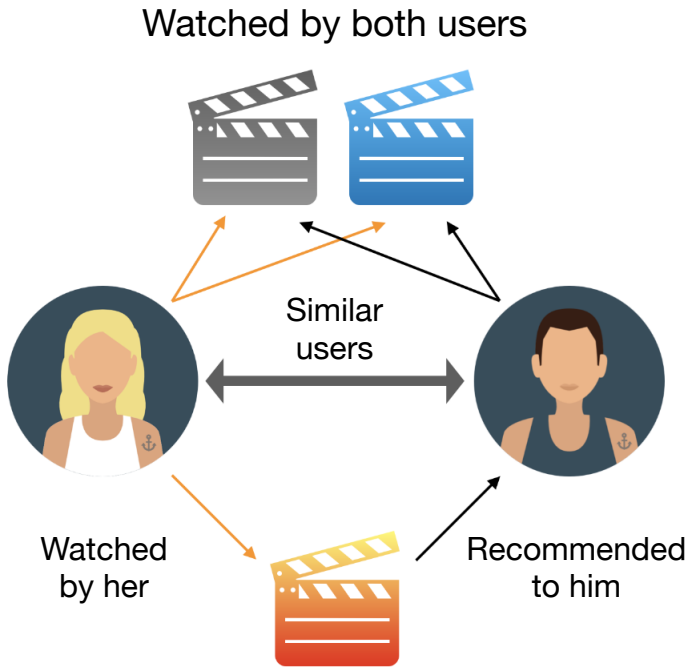
- Explain why user–item interaction data is sparse and motivates structured recommendation models
- Describe recommender systems as predicting missing preferences from observed user–item interactions
- Explain the low-rank assumption and interpret latent factors in recommender systems
- Formulate matrix factorization models as low-rank approximations of the user–item matrix
- Apply least-squares optimization to learn user and item embeddings from observed data
- Use learned models to predict unseen ratings and generate recommendations
- Interpret collaborative filtering as exploiting similarity among users or items
- Distinguish between user-based and item-based collaborative filtering methods
- Recognize common real-world applications of recommender systems

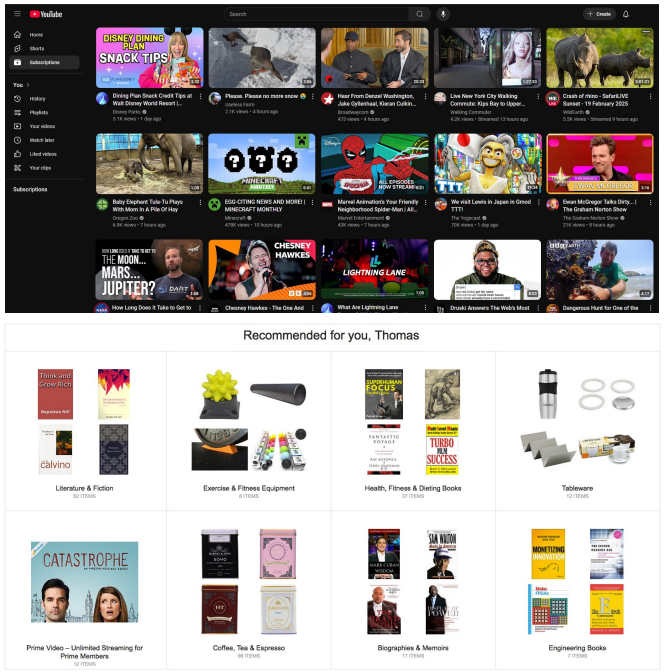
Motivation & Problem Setup

Recommender Systems

Goal: Learn how low-rank models and collaborative filtering enable personalized recommendations by **predicting user preferences** from sparse interaction data





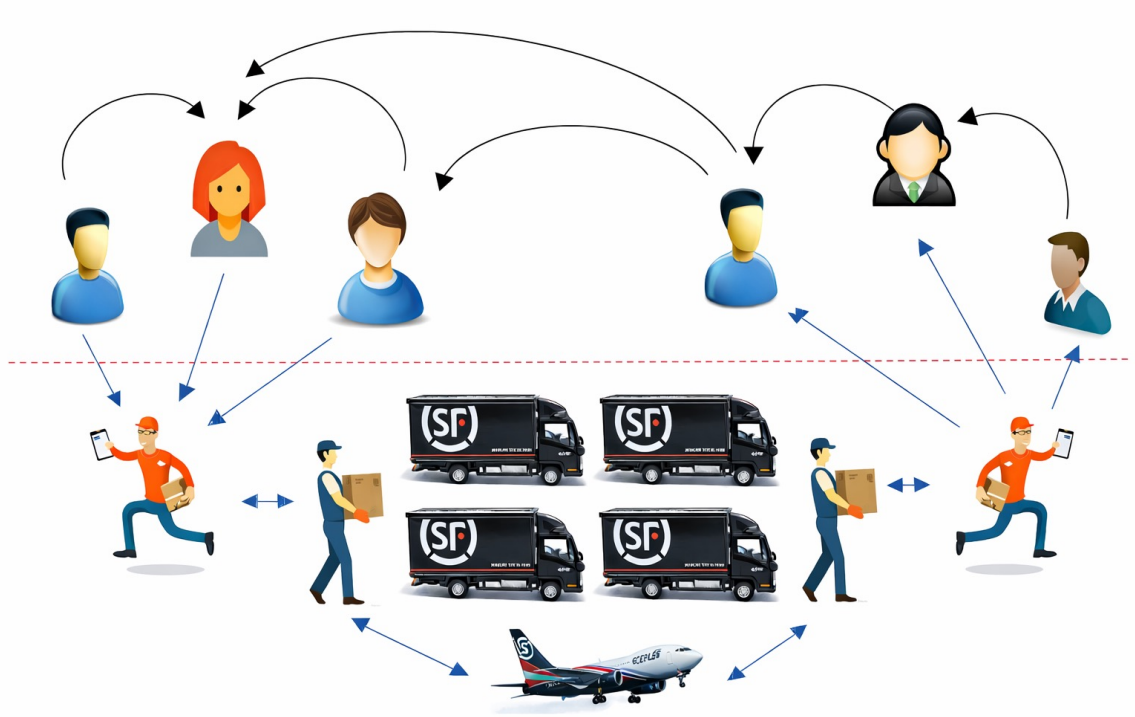


What assumption justifies recommending items liked by one user to another?

Example: Logistic Network Recommendation

Core Task: Given past data on user–item interactions, **predict** which items a user will like next

- **Network:** users, items, and interactions form a large bipartite graph
- **Inputs:** user–item interactions (ratings, clicks, purchases)
- **Output:** personalized ranking or rating prediction (not a single label)
- **Key challenge:** data is large, sparse, and noisy

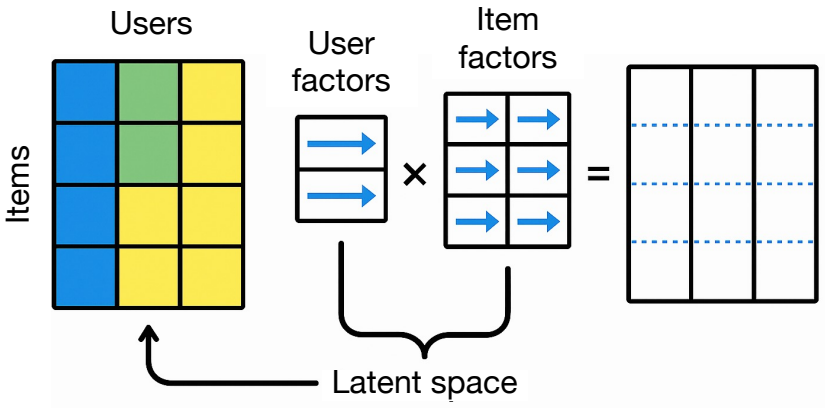


If we already know what a user liked before, why is prediction still hard?

Common Recommender Systems

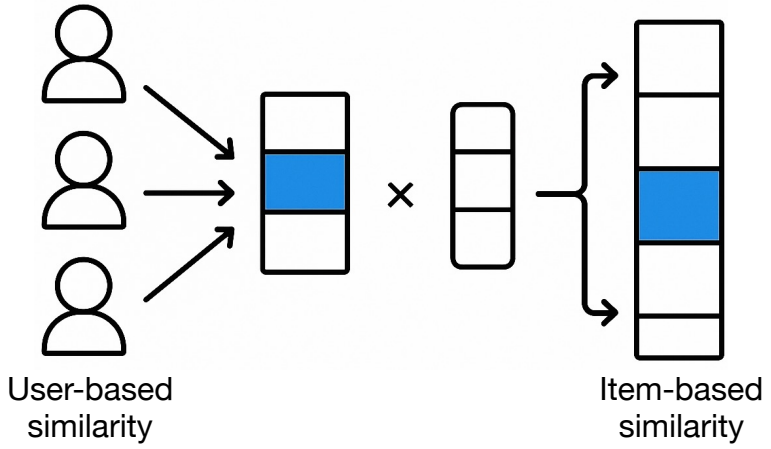
Low-Rank Models: Users and items are vectors in a latent space where interactions predict preferences

- **Matrix factorization:** Approximates the user–item matrix as the product of user and item latent factor matrices
- **Matrix completion:** Predicts missing user–item interactions by assuming the full interaction matrix has low rank



Collaborative Filtering: Recommends items using patterns in user–item interactions

- **User-based similarity:** Recommends items liked by users with similar interaction histories
- **Item-based similarity:** Recommends items similar to those a user has previously interacted with



Low-Rank Models

Why Low-Rank Models?

User–item data is high-dimensional, sparse, and noisy—but user preferences are driven by **few underlying factors**, not thousands of items

Example: Movie Preferences

- Movies blend a small number of genres (Action, Comedy, Drama, Sci-Fi, etc.)
- Users have preferences over few genres—they don't have thousands of unique tastes
- Many users and movies are similar in **latent ways**—preferences lie in a low-dimensional space

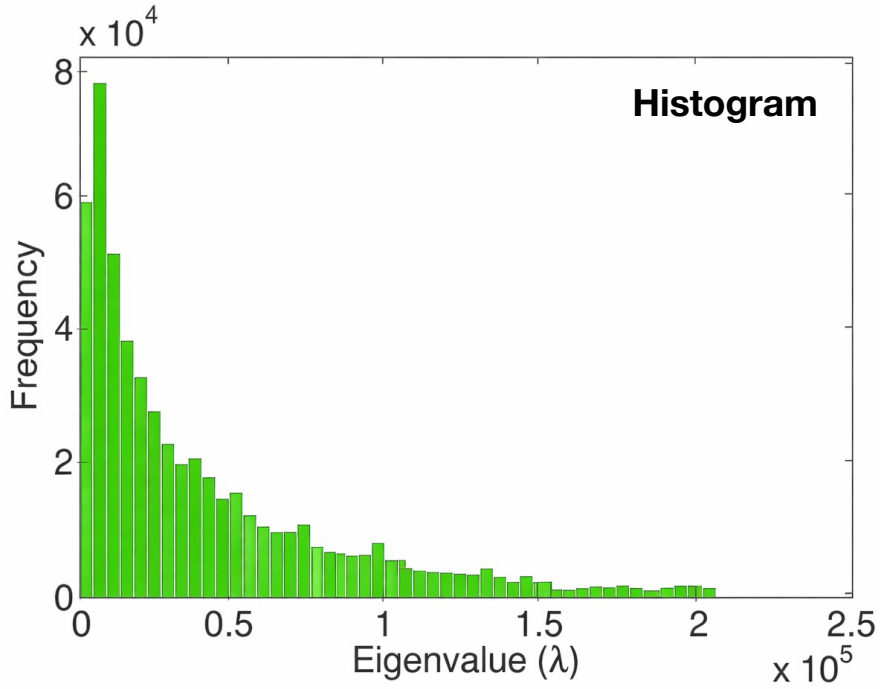
How can we identify the dominant patterns in user–item interactions?

Netflix Prize Competition (2006–2009)

- Predict missing entries in a large, incomplete, highly sparse **user–movie rating matrix** by improving on Netflix's Cinematch by at least 10% in RMSE ($1M prize)
- **Dataset:** Over 100 million ratings from about 480,000 users for 17,770 movies
- **BellKor's Pragmatic Chaos** won with just over a 10% improvement over Cinematch

Low-Rank Approximation

A small number of **latent factors** explain user preferences (e.g., genre affinity, popularity, taste intensity)



What happens if we keep too many latent factors?

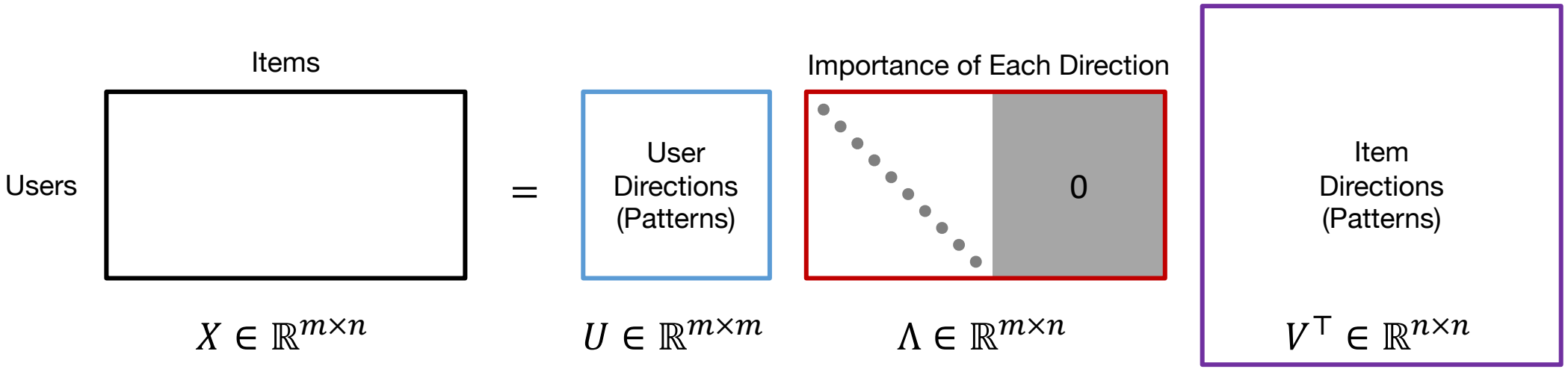
Singular Value Decomposition (SVD)

Low-rank models rely on the idea that data lies near a **low-dimensional subspace**

Given a user–item matrix $X \in \mathbb{R}^{m \times n}$, by SVD:

$$X = U \Lambda V^{\top}$$

- U: left singular vectors
- V: right singular vectors
- Λ: singular values



Singular Value Decomposition (SVD)

Low-rank models rely on the idea that data lies near a **low-dimensional subspace**

Given a user–item matrix $X \in \mathbb{R}^{m \times n}$, by SVD:

$$X = U \Lambda V^{\top}$$

- $U \in \mathbb{R}^{m \times m}$: user directions (user patterns)
- $V \in \mathbb{R}^{n \times n}$: item directions (item patterns)
- $\Lambda \in \mathbb{R}^{m \times n}$: importance of each direction

- User preferences depend on only a **few hidden factors** (e.g., taste, popularity)
- So, the matrix is approximately **low rank**:

$$r \ll \min\{m, n\}$$

Truncating SVD gives the best low-rank approx. in least-squares sense

Truncated SVD

Keeping only the top r, we get

$$X_r = U_r \Lambda_r V_r^\top \begin{cases} U_r \in \mathbb{R}^{m \times r} \\ V_r \in \mathbb{R}^{n \times r} \\ \Lambda_r \in \mathbb{R}^{r \times r} \end{cases} \qquad r \ll \min\{m, n\}$$

- This is exactly what we want: denoise, compress, capture dominant patterns
- But, to use SVD, we must fill missing entries → inject fake data/assumptions ✗

Factorization:

$$\Lambda_r = \Lambda_r^{1/2} \Lambda_r^{1/2}$$

$$\tilde{U} = U_r \Lambda_r^{1/2} \qquad \tilde{V} = V_r \Lambda_r^{1/2}$$

$$X_r = \tilde{U} \tilde{V}^\top$$

Matrix Factorization

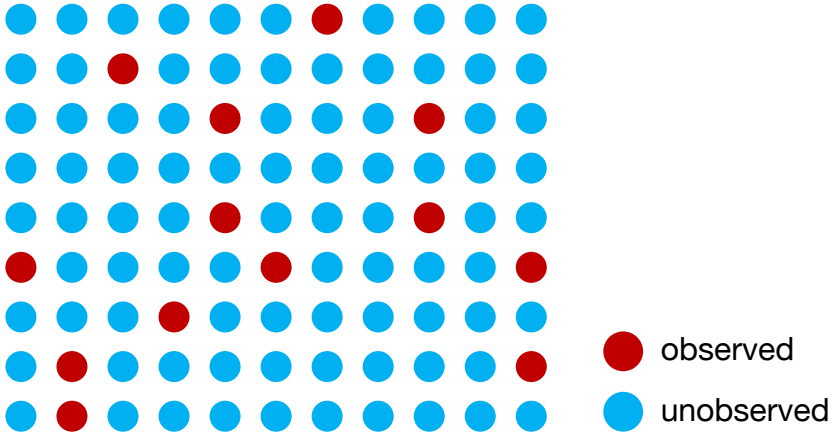
Given a **partially-observed** user–item matrix $X \in \mathbb{R}^{m \times n}$, factorize $X \approx UV^\top$, where

- $U \in \mathbb{R}^{m \times r}$: user factors (embeddings), where user i is vector $u_i \in \mathbb{R}^r$ $(\forall\, i = 1, \ldots, m)$
- $V \in \mathbb{R}^{n \times r}$: item factors (embeddings), where item j is vector $v_j \in \mathbb{R}^r$ $(\forall\, j = 1, \ldots, n)$

Prediction rule:

$$\hat{x}_{ij} = u_i^\top v_j$$

e.g., $i = 2, j = 3$



Users \ Items	1	2	3	4	5
1			4		
2	5			3	
3	1		2		2
4		2	5		
5	4			3	2

$\approx$

U (rows 1–5) × V (columns 1–5):

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

Optimization Problem ($r = 1$ Example)

If $r = 1$, then $u_i \in \mathbb{R}$ and $v_j \in \mathbb{R}$ (scalars), and $\hat{x}_{ij} = u_i v_j$, so we fit observed entries:

$$\min_{u_i, v_i} \sum_{(i,j) \in \Omega} \left(x_{ij} - u_i v_j\right)^2$$

the set of observed entries

- This is least squares
- Missing entries are ignored (**why?**)
- Nonconvex jointly, but convex in one variable at a time

Users \ Items	1	2	3	4	5
1			4		
2	5			3	
3	1		2		2
4		2	5		
5	4			3	2

$\fallingdotseq$

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

Alternating Least Squares

Update u_i with fixed v:

$$\frac{\partial}{\partial u_i} \sum_{j:(i,j)\in\Omega} \left(x_{ij} - u_i v_j\right)^2 = \sum_{j:(i,j)\in\Omega} 2\left(x_{ij} - u_i v_j\right)\left(-v_j\right) = 0 \quad \Rightarrow \quad u_i = \frac{\sum_{j:(i,j)\in\Omega} v_j x_{ij}}{\sum_{j:(i,j)\in\Omega} v_j^2}$$

Update v_j with fixed u:

$$\frac{\partial}{\partial v_j} \sum_{j:(i,j)\in\Omega} \left(x_{ij} - u_i v_j\right)^2 = \sum_{j:(i,j)\in\Omega} 2\left(x_{ij} - u_i v_j\right)\left(-u_i\right) = 0 \quad \Rightarrow \quad v_j = \frac{\sum_{i:(i,j)\in\Omega} u_i x_{ij}}{\sum_{i:(i,j)\in\Omega} u_i^2}$$

Each step reduces to independent 1D least-squares updates for users & items

Fitting

Step 1: Start with initial guesses (random, small constant, or heuristic-based) for user factors $(u_1, \ldots, u_m)$ and item factors $(v_1, \ldots, v_n)$

Step 2: If v_j are fixed, then for each user i, we solve:

$$\min_{u_i} \sum_{j:(i,j)\in\Omega} \left(x_{ij} - u_i v_j\right)^2 \quad \xrightarrow{\text{1D least-squares}} \quad u_i = \frac{\sum_{j:(i,j)\in\Omega} v_j x_{ij}}{\sum_{j:(i,j)\in\Omega} v_j^2}$$

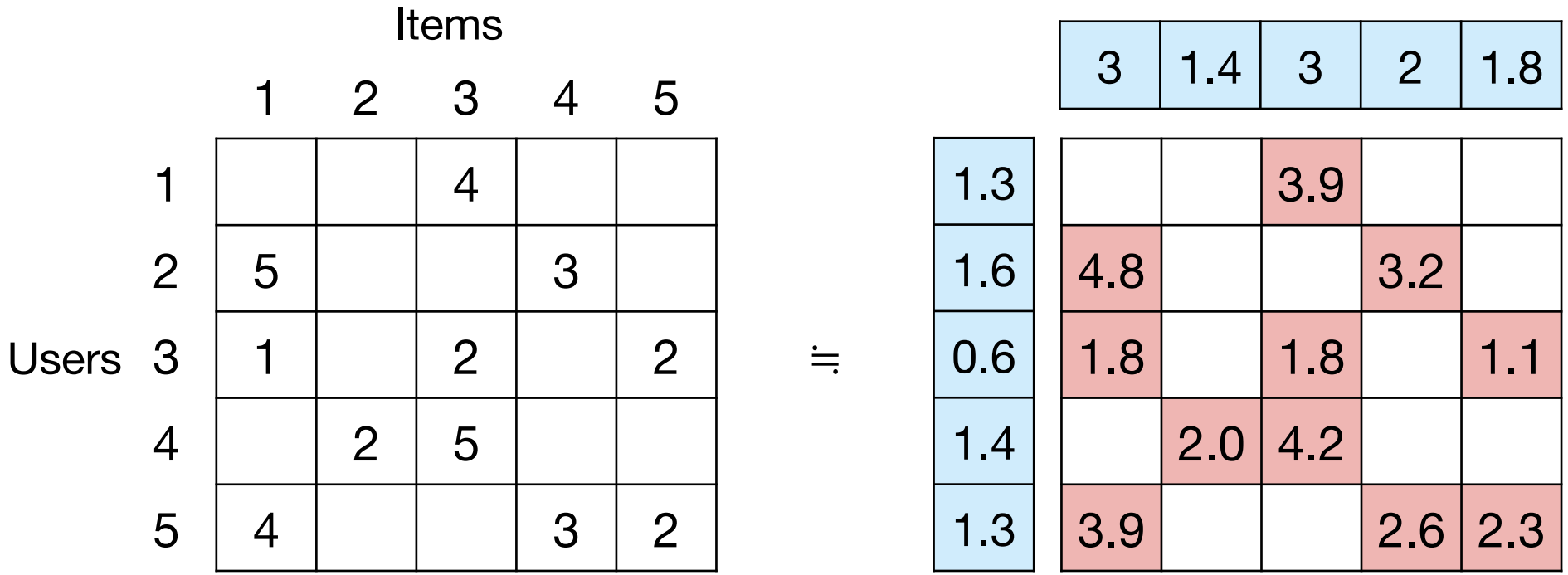
Step 3: Similarly, fixing u_i, for each item j:

$$\min_{v_j} \sum_{i:(i,j)\in\Omega} \left(x_{ij} - u_i v_j\right)^2 \quad \xrightarrow{\text{1D least-squares}} \quad v_j = \frac{\sum_{i:(i,j)\in\Omega} u_i x_{ij}}{\sum_{i:(i,j)\in\Omega} u_i^2}$$

Step 4: Repeat Steps 2 and 3 until changes are small (**coordinate descent**/**ALS**)

Fitting (cont.)

- **Predictions only appear after fitting latent factors**
- During fitting, we are not predicting missing entries—we are shaping the latent space to best explain observed data (missing entries are ignored, not zeroed)



Prediction and Generalization

We can now predict unseen ratings for any $(i, j) \notin \Omega$, $X_{ij} = u_i v_j$

- Large rank $r \rightarrow$ more expressive model, fits data better (too large overfits)
- Small rank $r \rightarrow$ better generalization, less noise

		Items				
		1	2	3	4	5
Users	1			4		
	2	5			3	
	3	1		2		2
	4		2	5		
	5	4			3	2

$\fallingdotseq$

	3	1.4	3	2	1.8
1.3	3.9	1.8	3.9	2.6	2.3
1.6	4.8	2.2	4.8	3.2	2.9
0.6	1.8	0.8	1.8	1.2	1.1
1.4	4.2	2.0	4.2	2.8	2.5
1.3	3.9	1.8	3.9	2.6	2.3

Regularization

Without additional constraints, the model can produce very large factor values, overfit the observed ratings, or generate extreme or unrealistic predictions

Regularization discourages large embeddings:

$$\min_{U,V} \sum_{(i,j)\in\Omega} \left(x_{ij} - u_i^\top v_j\right)^2 + \lambda(\|U\|_F^2 + \|V\|_F^2)$$

Frobenius norm:

$$\|U\|_F^2 = \sum_{i,k} U_{i,k}^2$$

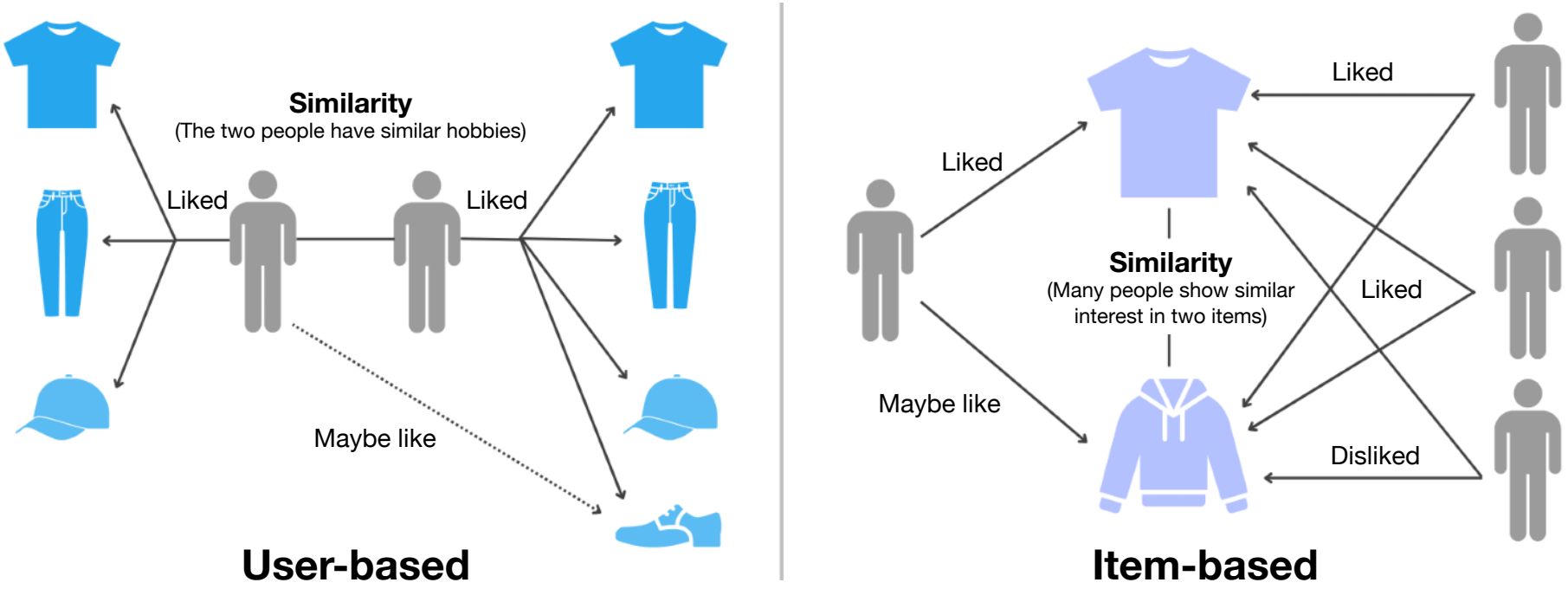
$$\|V\|_F^2 = \sum_{j,k} V_{j,k}^2$$

- Small/smooth user and item embeddings
- No fitting of noise in sparse data,
- Fewer extreme predictions,
- Improved generalization to unseen entries.

Collaborative Filtering

Collaborative Filtering via Similarity

Recommend items using patterns of **similar** users or items measured based on **cosine similarity** or **Pearson correlation**



What information do we lose when we drop latent factors?

Similarity Functions

Cosine similarity: Given two vectors $x, y \in \mathbb{R}^n$:

$$\text{sim}_{\cos}(x, y) = \frac{x^\top y}{\|x\|\|y\|} = \cos(\theta)$$ → θ: angle between x and y

→ Measures angle, not magnitude—use when absolute rating scale matters less or data is sparse

Pearson correlation:

$$\text{sim}_{\text{Pearson}}(x, y) = \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_i (x_i - \bar{x})^2}\sqrt{\sum_i (y_i - \bar{y})^2}} = \text{sim}_{\cos}(x - \bar{x}, y - \bar{y})$$

→ Measures **agreement relative to each user's mean** and corrects for users who always rate high or low (extreme raters)—use when rating bias matters

Similarity Functions: Example

Items co-rated by Users 1 and 2: $\{1, 2, 4\}$

$$x = [5, 4, 1],\ y = [4, 5, 1]$$

		Items			
		1	2	3	4
Users	1	5	4		1
	2	4	5		1
	3	1		5	4

$$\text{sim}_{\cos}(x, y) = \frac{x^\top y}{\|x\|\|y\|} = \frac{5 \cdot 4 + 4 \cdot 5 + 1 \cdot 1}{\sqrt{42} \cdot \sqrt{42}} \approx 0.976$$

$$\text{sim}_{\text{Pearson}}(x, y) = \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_i (x_i - \bar{x})^2}\sqrt{\sum_i (y_i - \bar{y})^2}} \approx 0.88$$

$$\bar{x} = \bar{y} = 3.33$$

Cosine compares direction; Pearson compares co-variation around the mean

User-Based Collaborative Filtering

Weighted average of **neighbors**' opinions:

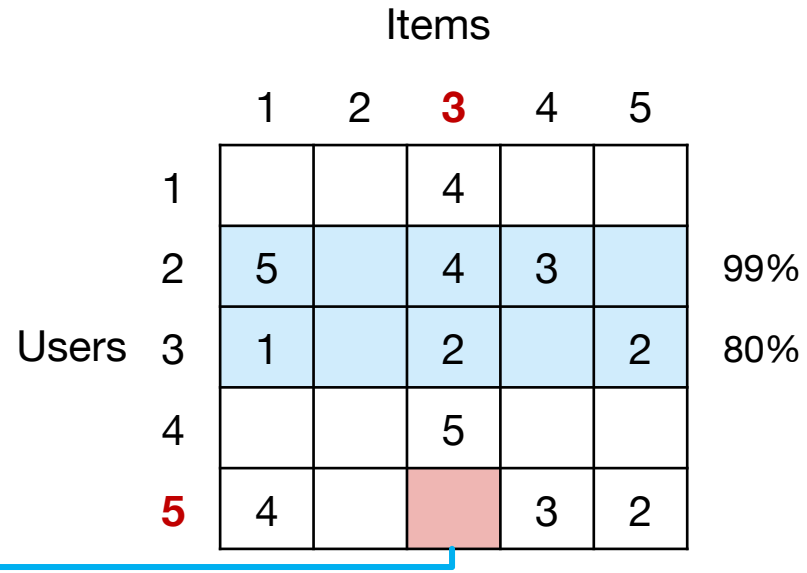
Step 1: Identify a set S_u of users **similar to** the target user u

Step 2: Collect their ratings x_{vi} for item i

Step 3: Compute a similarity-weighted average using $\text{sim}(u, v)$

$$\hat{x}_{ui} = \frac{1}{\sum_{v \in S_u} \text{sim}(u,v)} \sum_{v \in S_u} \text{sim}(u,v) x_{vi}$$

$$\frac{0.99 \times 4 + 0.8 \times 2}{0.99 + 0.8} = 3.11$$



Item-Based Collaborative Filtering

Weighted average of **similar items**' ratings (more stable):

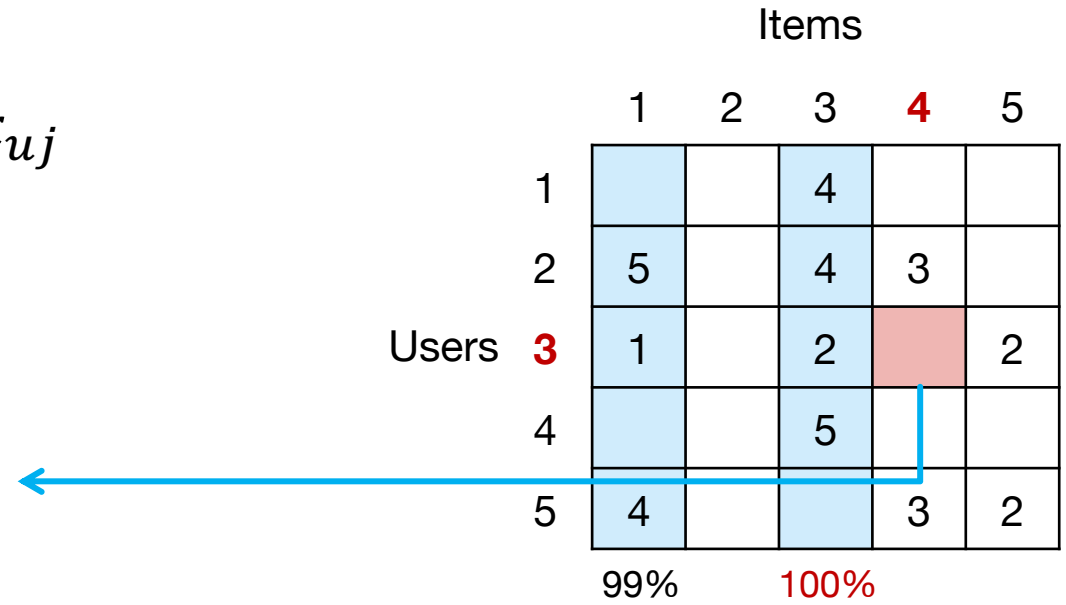
Step 1: Identify a set of items S_i **similar to** the target item i

Step 2: Collect the target user's ratings x_{uj} on similar items

Step 3: Predict by similarity-weighted averaging using $\text{sim}(i,j)$

$$\hat{x}_{ui} = \frac{1}{\sum_{j \in S_i} \text{sim}(i,j)} \sum_{j \in S_i} \text{sim}(i,j) x_{uj}$$

$$\frac{0.99 \times 1 + 1 \times 2}{0.99 + 1} = 1.50$$



Key Takeaways

What We Learned This Week

- Recommender systems predict missing user–item preferences from sparse interaction data to produce personalized rankings
- Low-rank structure motivates matrix factorization, learning user/item embeddings to infer unseen ratings via inner products
- Fitting uses observed entries only (e.g., least squares / SGD), and prediction fills in missing entries from learned factors
- User-based collaborative filtering recommends using similarity-weighted ratings from similar users
- Item-based collaborative filtering recommends using similarity-weighted ratings on similar items (often more stable/scalable)
- Similarity choice (cosine / Pearson) and neighborhood size strongly affect recommendation quality