Recommendation System: Basic Methods

Prof. Srijan Kumar

RecSys Drives The World

- Recommendation systems are ubiquitous
  - **E-commerce:** Products to buyers
  - **Job applications:** Jobs to applicants, Applicants to recruiters
  - **Dating:** Partners
  - **Services:** Doctors, lawyers, restaurants
  - **Social networks:** News that you read, posts that you see
- **Most of what you do is powered or influenced by some recommender system!**

Types of Recommendations

- **Editorial and hand curated**
  - List of favorites
  - Lists of “essential” items
- **Simple aggregates**
  - Top 10, Most Popular, Recent Uploads
- **Personalized to individual users**
  - Amazon, Netflix, …
  - **Main focus of this class**

General Setting

- **Input:**
  - U = Set of users
  - I = Set of items
  - R = Set of observed ratings between subset of users and subset of items
- **Output:**
  - Predict ratings for any (user, item) pair

	Avatar	LOTR	Matrix	Pirates
Alice	1		5	
Bob		3		1
Carol				1
David			4	

Recommendations: Key Challenges

- **(1) Gathering “known” ratings for matrix**
  - How to collect the data in the utility matrix

- **(2) Extrapolate unknown ratings from the known ones**
  - Mainly interested in high unknown ratings
    - We are not interested in knowing what you don’t like but what you like

- **(3) Evaluating extrapolation methods**
  - How to measure success/performance of recommendation methods

Challenge 1: Gathering Ratings

- **Explicit**
  - Ask people to rate items
  - Doesn't work well in practice – people can't be bothered
- **Implicit**
  - Learn ratings from user actions
    - E.g., purchase implies high rating
  - What about low ratings?

Challenge 2: Extrapolating Utilities

- **Key problem:** Utility matrix ***U*** is **sparse**
  - Most people have not rated most items
  - **Cold start:**
    - New items have no ratings
    - New users have no history

- **Four approaches:**
  - **1)** Content-based
  - **2)** Collaborative
  - **3)** Latent factor based
  - **4)** Deep learning based

Our focus

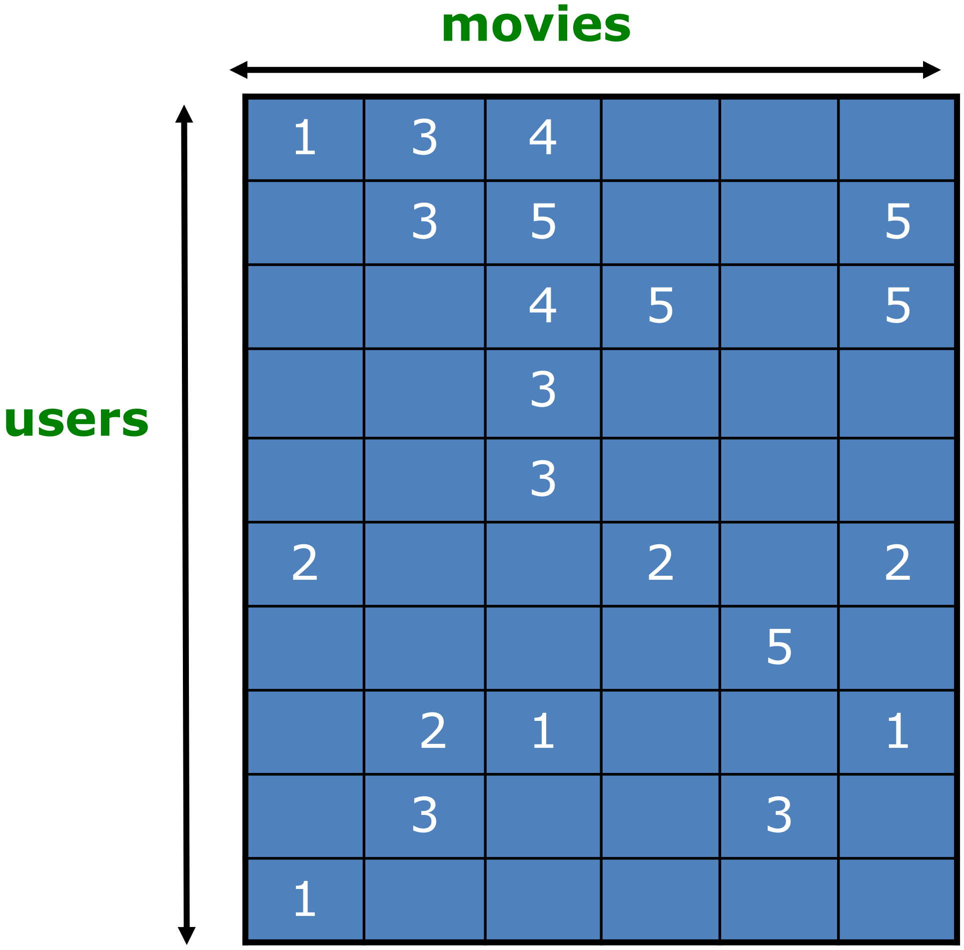
Challenge 3: Evaluations

- Metrics to evaluate recommendations and ranking lists
- **Online setting:** Purchase rate, Click-through rate
- **Offline setting:** which metrics to use?

- **How do we evaluate recommender systems?**
  - Need evaluation setting and metric

Evaluation Setting

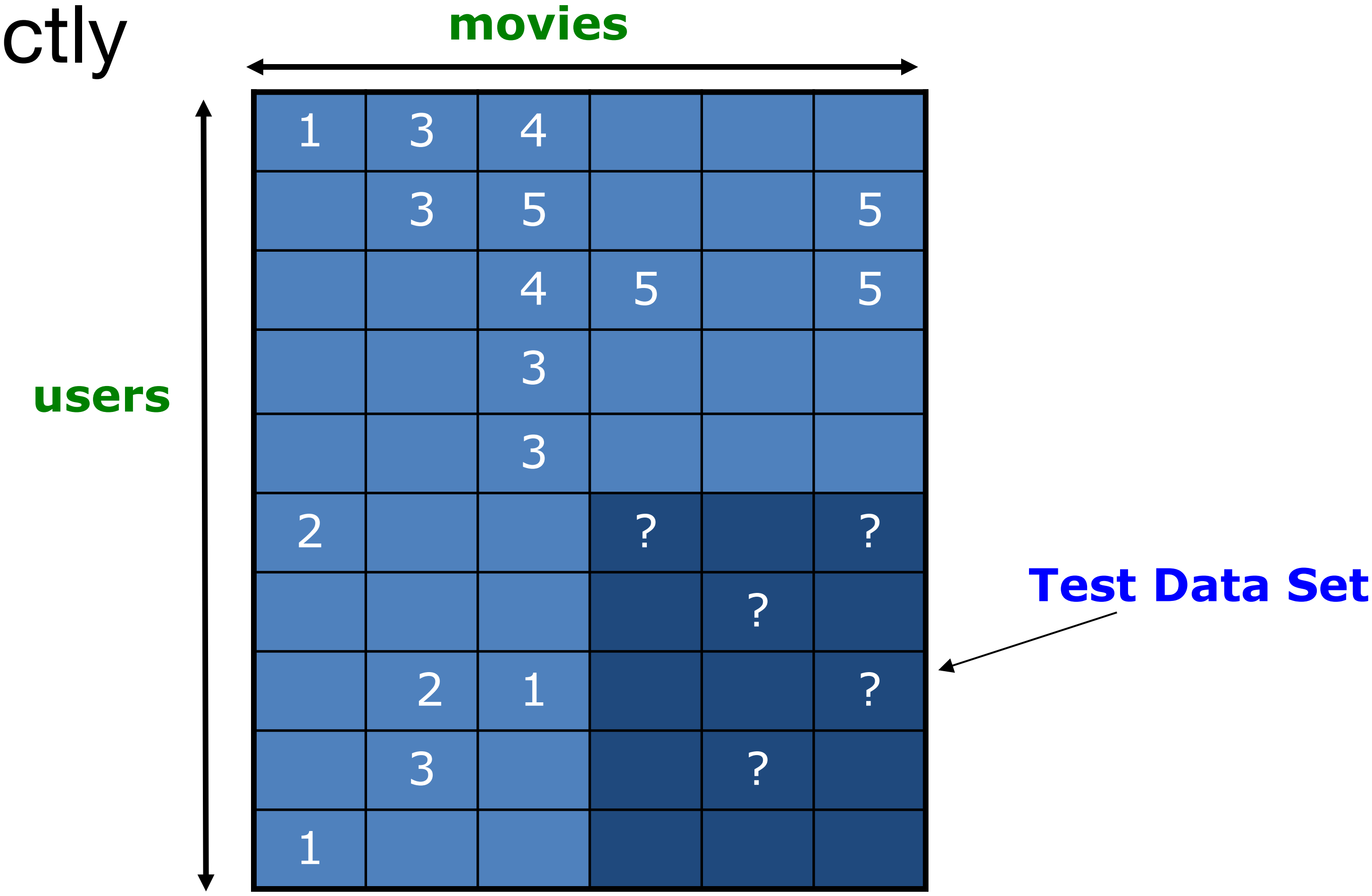
- Input: rating matrix



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

Evaluation Setting

- Hide some known ratings and try to predict it correctly



movies

users

Test Data Set

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

Evaluation Metrics

- **Compare predictions with known ratings**
  - **Root-mean-square error (RMSE)**
    - $\sqrt{\sum_{xi}\left(r_{xi} - r_{xi}^{*}\right)^{2}}$ where $\boldsymbol{r_{xi}}$ is predicted, $\boldsymbol{r_{xi}^{*}}$ is the true rating of ***x*** on ***i***
    - Lower is better
- **Another approach: 0/1 model**
  - **Mean Reciprocal Rank (MRR)**
    - The 1/rank of the true item in the ranked list of items
  - **Recall@10**
    - Fraction of times the true item is in the top 10 in the ranked list of items

Problems with Evaluation Measures

- **Narrow focus on accuracy sometimes misses the point**
  - Prediction Diversity
  - Prediction Context
  - Order of predictions
- **In practice, we care only to predict high ratings:**
  - RMSE might penalize a method that does well for high ratings and badly for others

RecSys: Approach Styles

1. **Content-based:** which items have properties similar to the items you already like?
2. **Collaborative filtering:** what do similar users like?
3. **Latent factor based:** learn latent representation of all users' behavior
4. **Deep learning based:** combine everything together in a unified, deep framework

Recommendation System Methods

- Content-based Models
- Collaborative Filtering
- Latent Factor Models
- Deep Learning Method: NCF

Reference material: http://mmds.org/

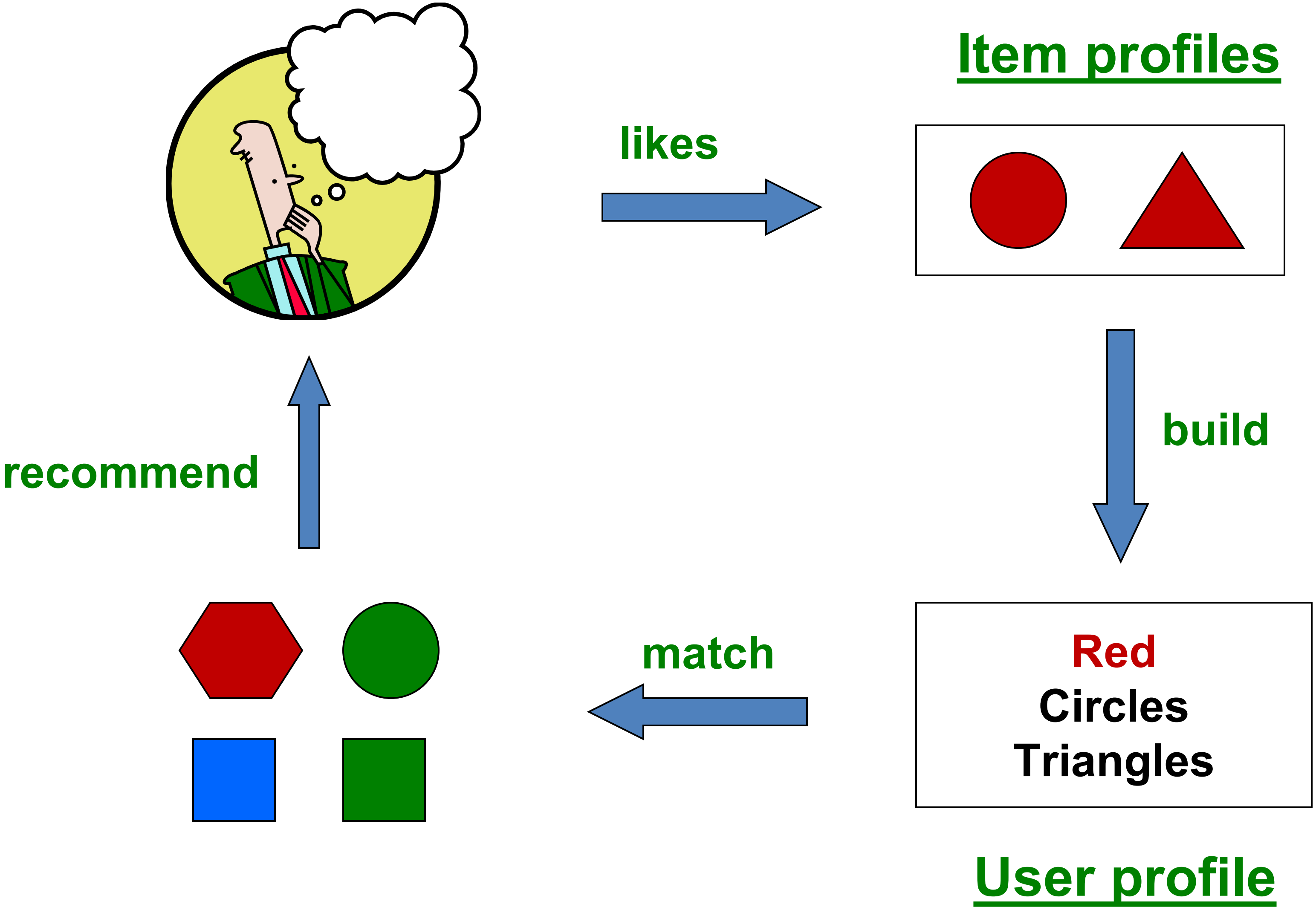
Content-based Recommendations

- **Main idea:** Recommend items to customer ***x*** similar to previous items rated highly by ***x***

Example:

- **Movie recommendations**
  - – Recommend movies with same actor(s), director, genre, …
- **Websites, blogs, news**
  - – Recommend other sites with “similar” content

Workflow



Item profiles

likes

build

Red
Circles
Triangles

User profile

match

recommend

Item Profiles

- For each item, create an **item profile** i

- **Profile is a set (vector) of features**
  - **Movies:** author, title, actor, director,…
  - **Text Embeddings**

User Profiles and Prediction

- **User profile possibilities:**
  - Weighted average of rated item profiles

- **Prediction heuristic:**
  - Given user profile ***x*** and item profile ***i***, estimate $u(\boldsymbol{x}, \boldsymbol{i}) = \cos(\boldsymbol{x}, \boldsymbol{i}) = \frac{\boldsymbol{x} \cdot \boldsymbol{i}}{||\boldsymbol{x}|| \cdot ||\boldsymbol{i}||}$

Pros: Content-based Approach

- **No need for data on other users**
  - No cold-start or sparsity problems
- **Able to recommend to users with unique tastes**
- **Able to recommend new & unpopular items**
  - No first-rater problem
- **Able to provide explanations**
  - Can provide explanations of recommended items by listing content-features that caused an item to be recommended

Cons: Content-based Approach

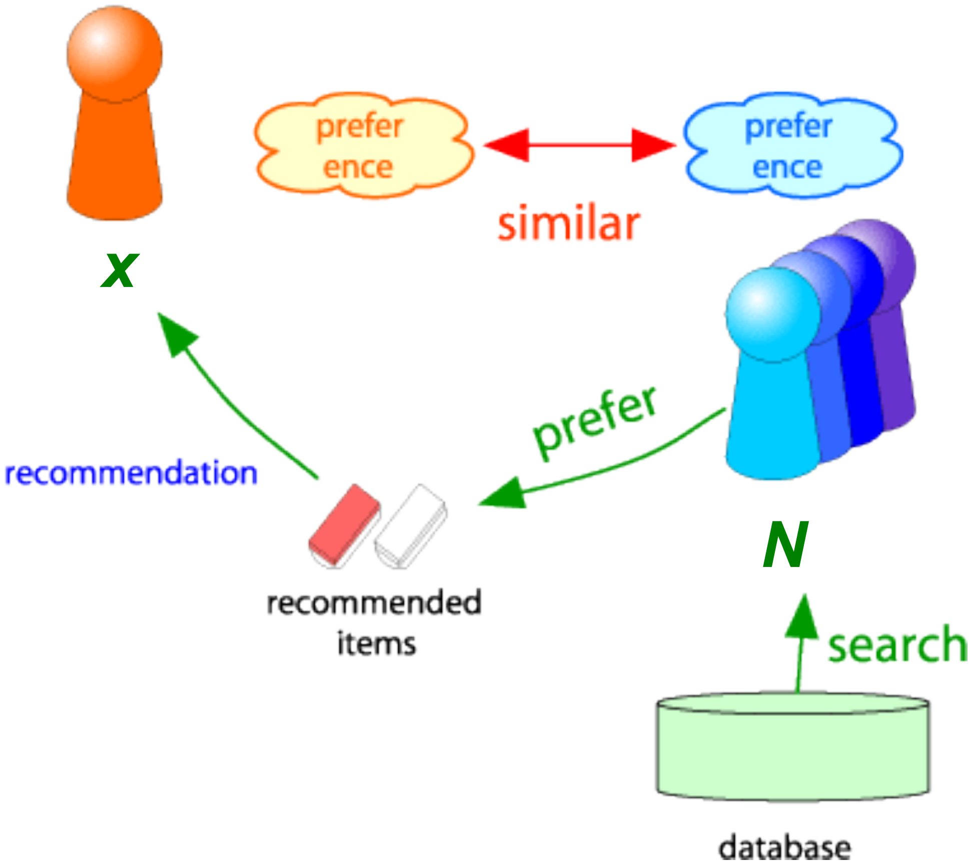
- **Finding the appropriate features is hard**
  - E.g., images, movies, music
- **Recommendations for new users**
  - How to build a user profile?
- **Overspecialization**
  - Never recommends items outside user’s content profile
  - People might have multiple interests
  - **Unable to exploit quality judgments of other users**

Recommendation Systems

- ✓ Content-based
- ☞ Collaborative Filtering
- Latent Factor Models
- Deep Learning Model: NCF

Collaborative Filtering

- **Harnessing quality judgments of other users**
- Consider user x
- Find set ***N*** of other users whose ratings are "**similar**" to ***x***'s ratings
- Estimate ***x***'s ratings based on ratings of users in ***N***



Finding “Similar” Users

- How to find similarity between users?
- r_x = vector of user ***x***’s ratings; r_y = ***y***’s ratings

r_x = [*, _, _, *, ***]
r_y = [*, _, **, **, _]

- **Metric 1: Jaccard similarity measure**
  - Jaccard = (Number of common items rated by both) / (Number of items rated by either)
  - **Problem:** Ignores the value of the rating

Finding “Similar” Users

- **Metric 2: Cosine similarity measure**
  - Treats r_x and r_y as vectors
  - sim(***x***, ***y***) = cos($\boldsymbol{r_x}$, $\boldsymbol{r_y}$) = $\frac{r_x \cdot r_y}{||r_x|| \cdot ||r_y||}$
  - **Problem:** Treats missing ratings as “negative”

r_x, r_y as vectors:

$\boldsymbol{r_x}$ = {1, 0, 0, 1, 3}

$\boldsymbol{r_y}$ = {1, 0, 2, 2, 0}

Similarity Metric: Solution

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

- Intuitively we want: sim(A, B) > sim(A, C)
- But, Jaccard similarity: 1/5 < 2/4
- Cosine similarity: 0.386 > 0.322
  - – Problem: Considers missing ratings as "negative"
  - – Solution: Normalize by subtracting the row mean. Then calculate the cosine similarity.
  - – Note: Cosine similarity is correlation when the data is centered at 0

Similarity Metric: Solution

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

sim(A,B) vs sim(A,C):
0.386 > 0.322

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 sim(A,C):
0.092 > -0.559

Rating Predictions

From similarity metric to recommendations:

- Let $\boldsymbol{r_x}$ be the vector of user ***x***’s ratings
- Let ***N*** be the set of ***k*** users most similar to ***x*** who have rated item ***i***
- **Prediction for item *s* of *user 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:
$s_{xy} = sim(x, y)$

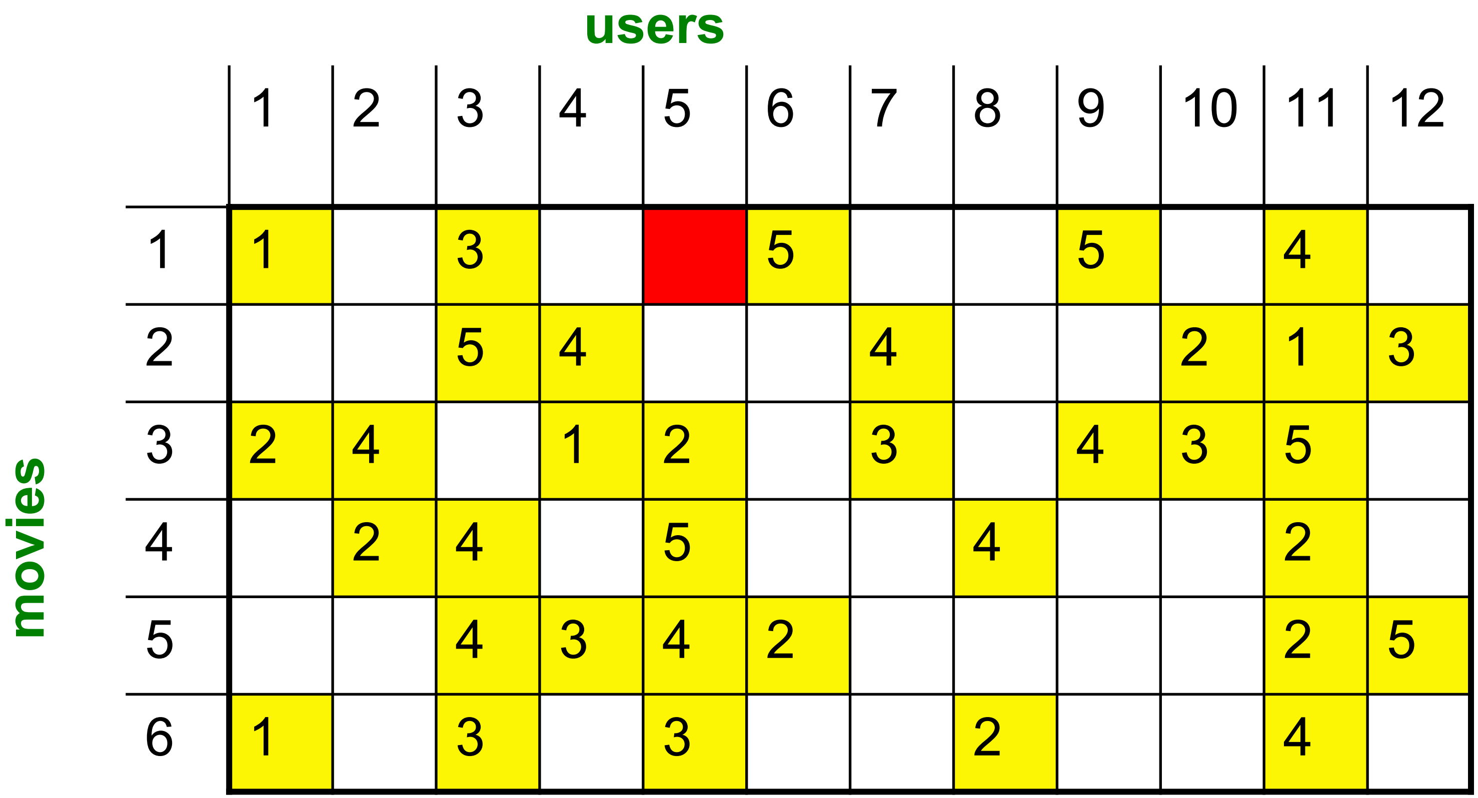
Item-Item Collaborative Filtering

- **So far: User-user collaborative filtering**
- **Another view: Item-item**
  - For item ***i***, find other similar items
  - Estimate rating for item ***i*** based on ratings for similar items
  - Can use same similarity metrics and prediction functions as in user-user model

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

s_{ij} = similarity of items ***i*** and ***j***
r_{xj} = rating of user ***u*** on item ***j***
N(i;x)= set items rated by ***x*** similar to ***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	

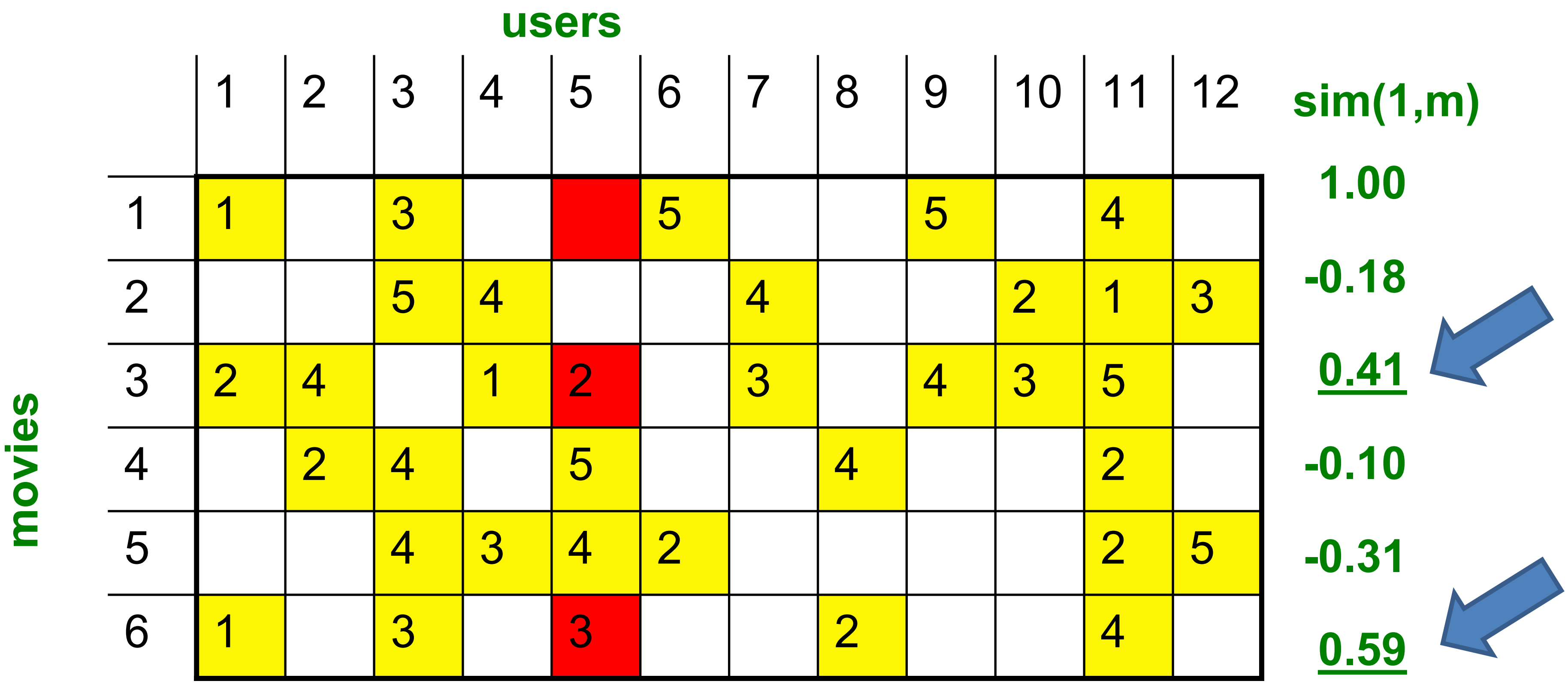
☐ unknown rating

🟨 rating between 1 to 5

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

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

Item-Item CF (|N|=2)



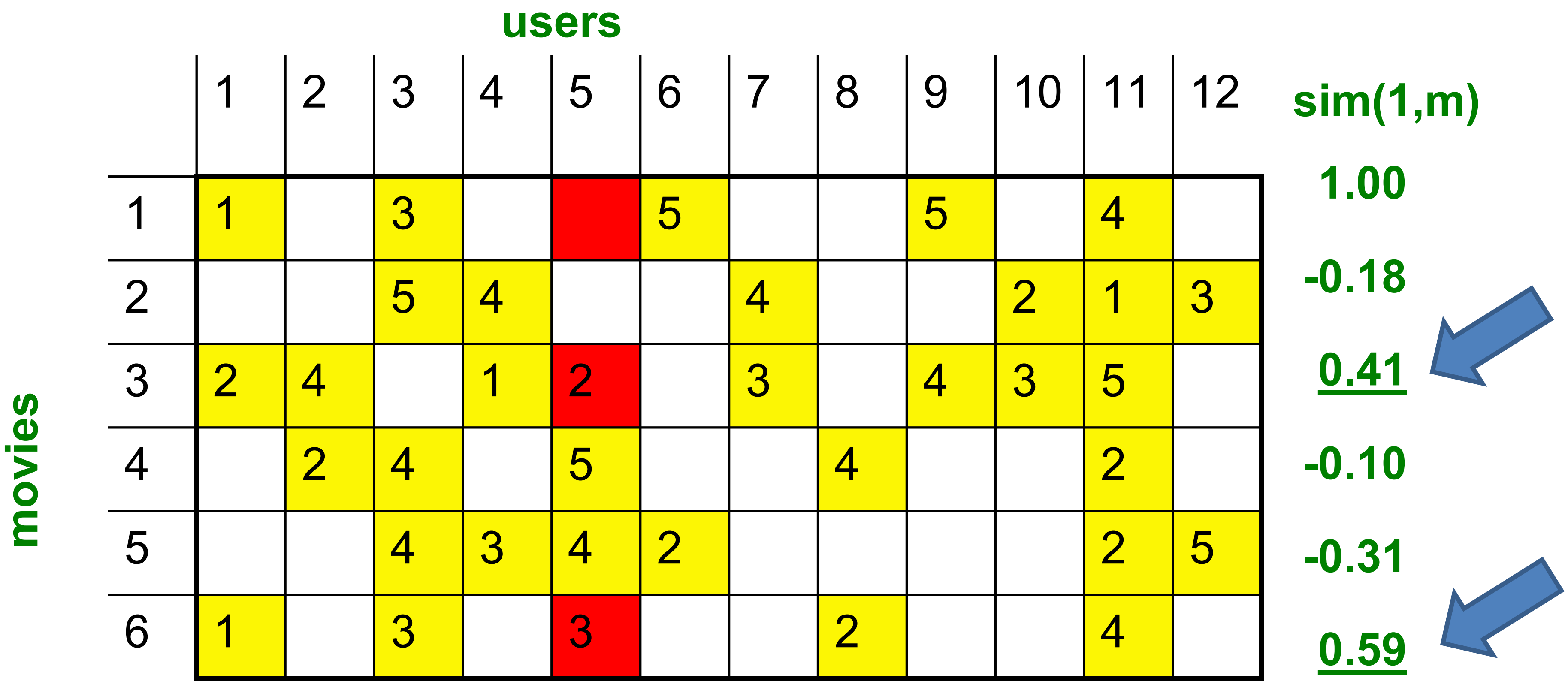
	users												
movies	1	2	3	4	5	6	7	8	9	10	11	12	sim(1,m)
1	1		3		🟥	5			5		4		1.00
2			5	4			4			2	1	3	-0.18
3	2	4		1	2		3		4	3	5		0.41
4		2	4		5			4			2		-0.10
5			4	3	4	2					2	5	-0.31
6	1		3		3			2			4		0.59

Use correlation as similarity: Subtract mean rating m_i from each movie *i*. Mean $m_1 = (1+3+5+5+4)/5 = \mathbf{3.6}$

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

Then, 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**

Predict by taking weighted average:

$r_{1,5}$ = (0.41*2 + 0.59*3) / (0.41+0.59) = 2.6

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

Item-Item vs. User-User

- In practice, it has been observed that **item-item often works better than user-user**
- **Why?** Items are simpler, users have multiple tastes

Pros of Collaborative Filtering

- **Works for any kind of item**
  - No feature selection needed
- **Leverages other users' behavior**
  - Incorporates community actions

Cons of Collaborative Filtering

- **Cold Start:**
  - Need enough users in the system to find a match
- **Sparsity:**
  - The user/ratings matrix is sparse
  - Hard to find users that have rated the same items
- **First rater:**
  - Cannot recommend an item that has not been previously rated. What about new items, esoteric items?
- **Popularity bias:**
  - Cannot recommend items to someone with unique taste
  - Tends to recommend popular items

Recommendation Systems

- ✓ Content-based
- ✓ Collaborative Filtering
- ☞ Latent Factor Models
- Deep Learning Models

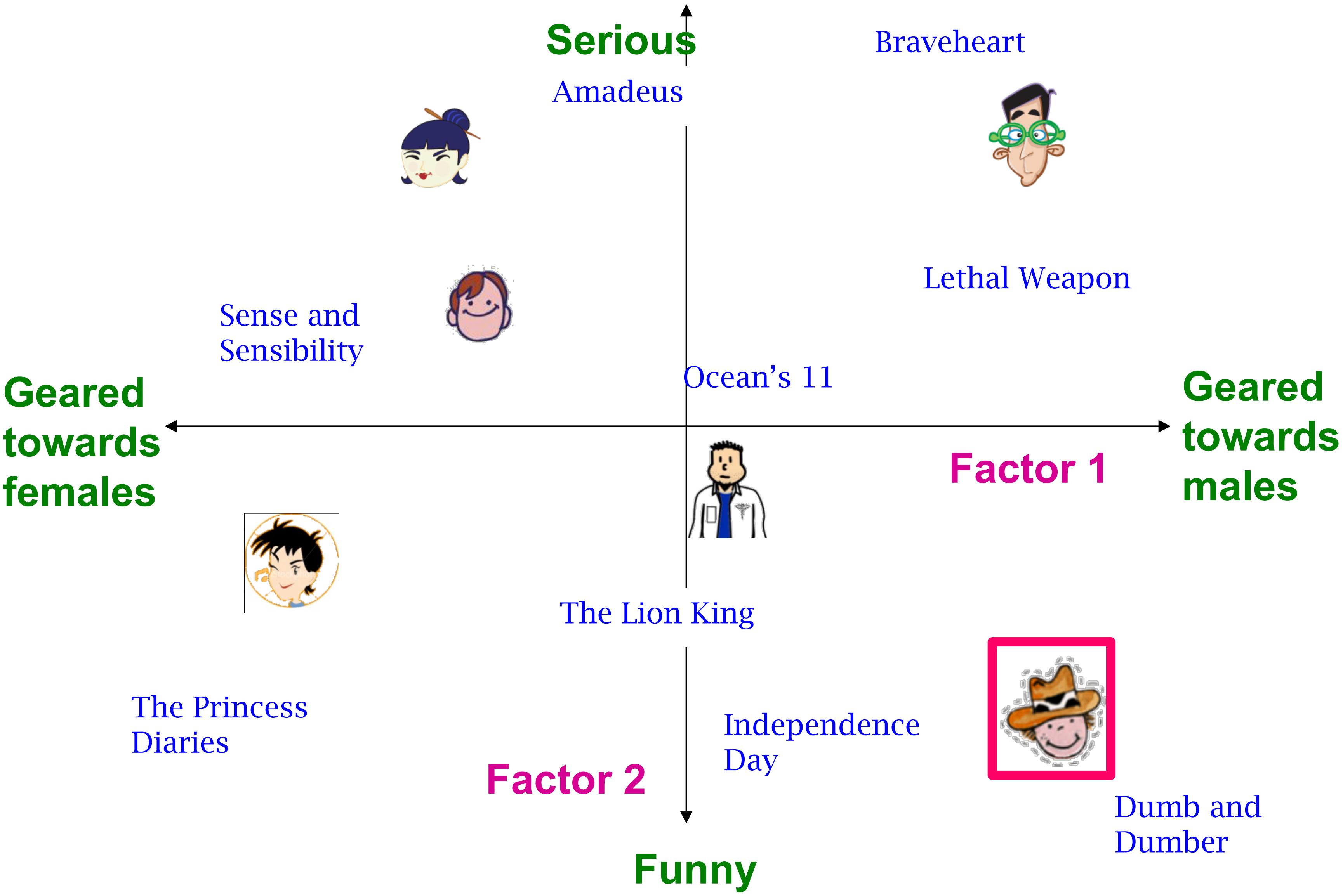
Latent Factor Models

- These models learn **latent factors** to represent users and items from the rating matrix
  - Latent factors are not directly observable
  - These are derived from the data
- **Recall:** Network embeddings
- **Methods:**
  - Singular value decomposition (SVD)
  - Principal Component Analysis (PCA)
  - Eigendecompositon

Latent Factors: Example

- **Embedding axes** are a type of latent factors
- **In a user-movie rating matrix:**
- **Movie latent factors** can represent axes:
  - Comedy vs drama
  - Degree of action
  - Appropriateness to children
- **User latent factors** will measure a user's affinity towards corresponding movie factors

Latent Factors: Example



Serious

Braveheart

Amadeus

Lethal Weapon

Sense and Sensibility

Ocean's 11

Geared towards females

Geared towards males

Factor 1

The Lion King

The Princess Diaries

Independence Day

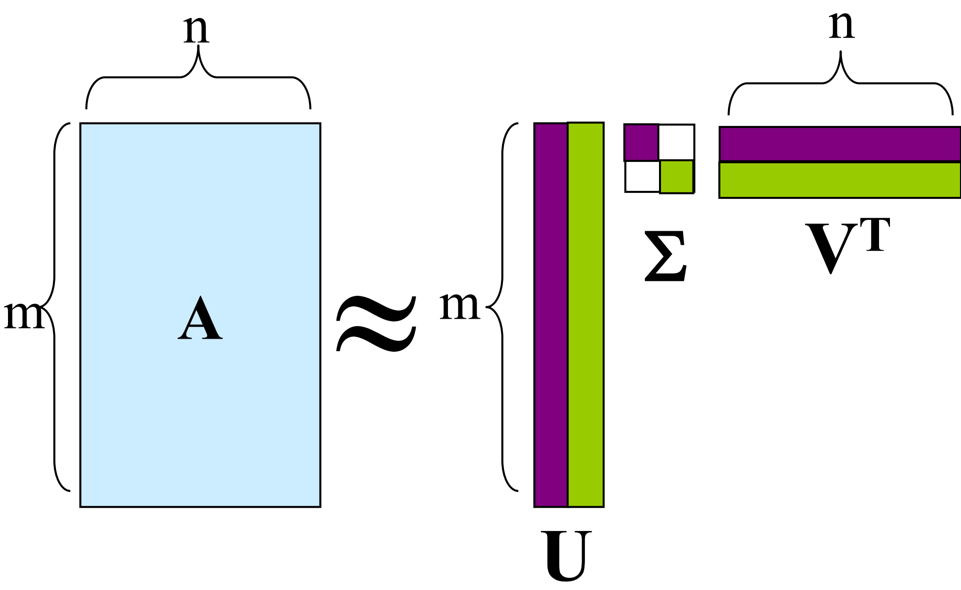
Dumb and Dumber

Factor 2

Funny

SVD

- **SVD:** SVD decomposes an input matrix into multiple factor matrices
  - $A = U \Sigma V^T$
  - Where,
  - **A**: Input data matrix
  - **U**: Left singular vecs
  - **V**: Right singular vecs
  - Σ: Singular values



SVD

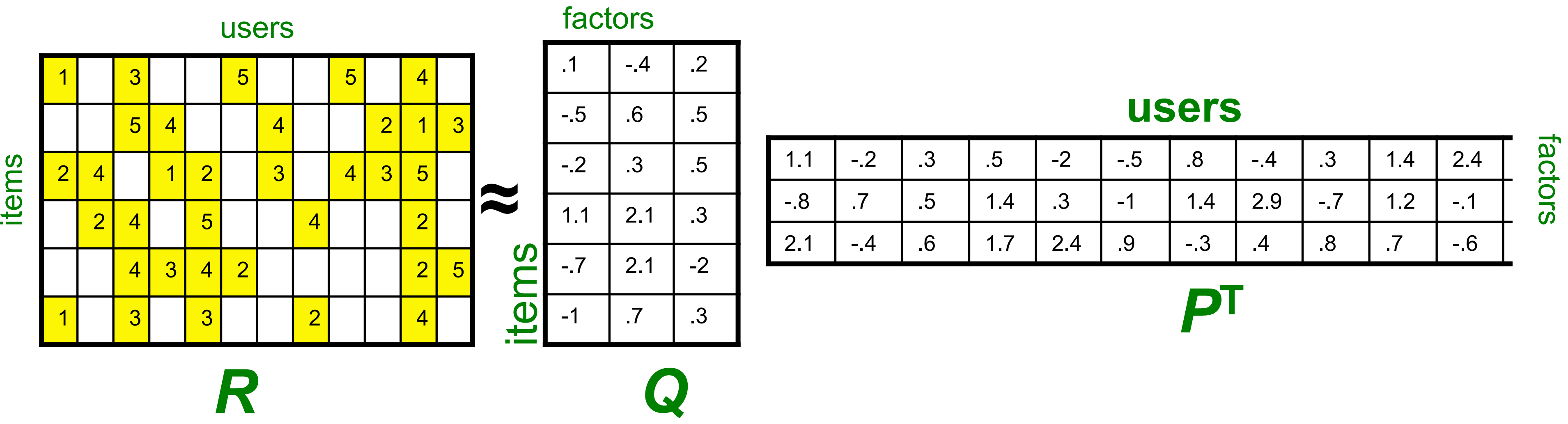
- SVD gives **minimum reconstruction error** (Sum of Squared Errors):

$$\min_{U,\mathrm{V},\Sigma} \sum_{ij\in A} \left(A_{ij} - [U\Sigma V^{\mathrm{T}}]_{ij}\right)^2$$

- **SSE** and **RMSE** are monotonically related:
  - $RMSE = \frac{1}{c}\sqrt{SSE}$ ➔ SVD is minimizing RMSE
- **Complication:** The sum in SVD error term is over all entries. But our ***R*** has missing entries.
  - **Solution:** no-rating in interpreted as zero-rating.

SVD on Rating Matrix

- "SVD" on rating data: $\mathbf{R} \approx \mathbf{Q} \cdot \boldsymbol{P}^T$
- Each row of Q represents an item
- Each column of P represents a user



R

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	

Q

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

P^T

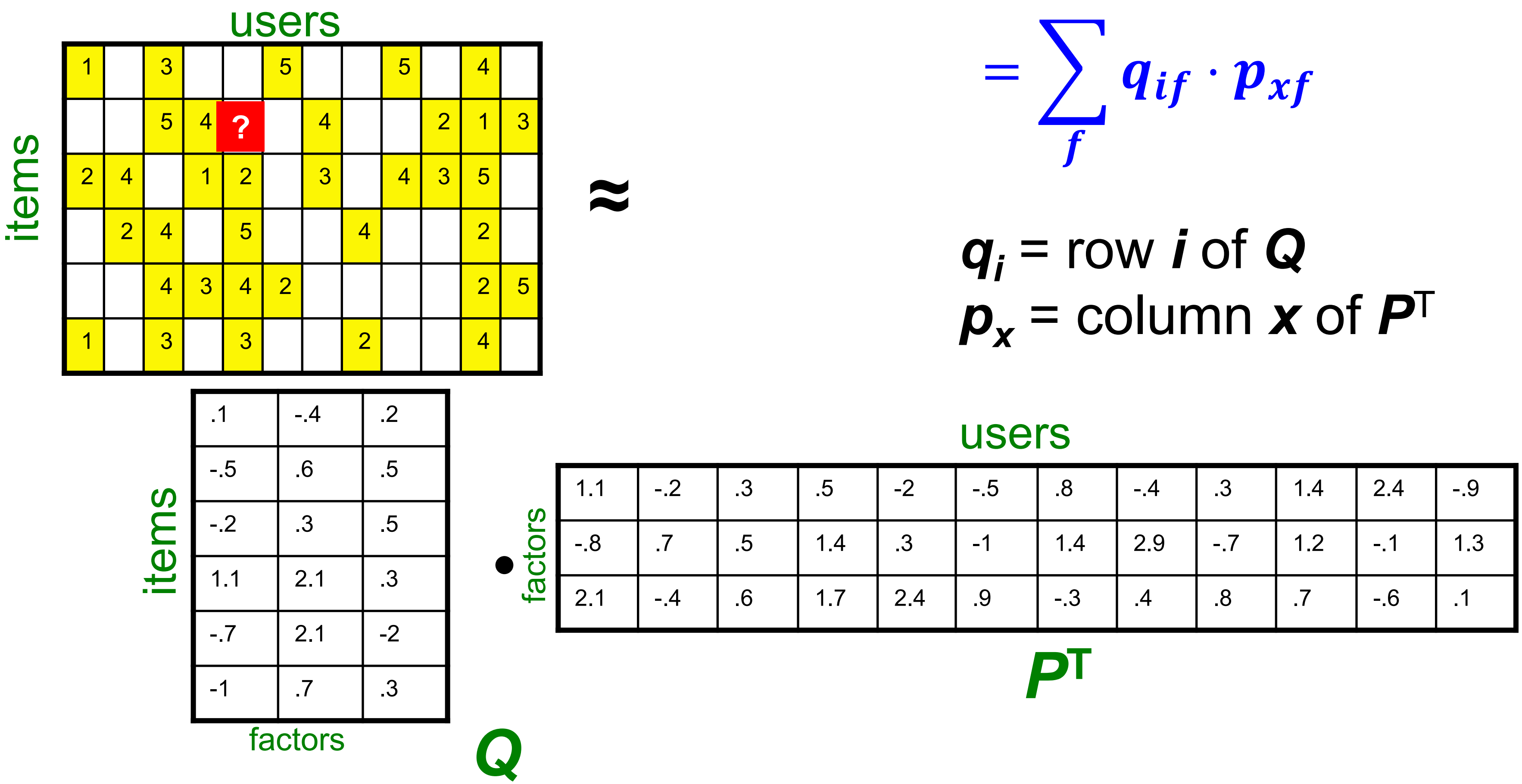
1.1	-.2	.3	.5	-2	-.5	.8	-.4	.3	1.4	2.4
-.8	.7	.5	1.4	.3	-1	1.4	2.9	-.7	1.2	-.1
2.1	-.4	.6	1.7	2.4	.9	-.3	.4	.8	.7	-.6

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



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	

≈

items

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

factors

Q

•

users

factors

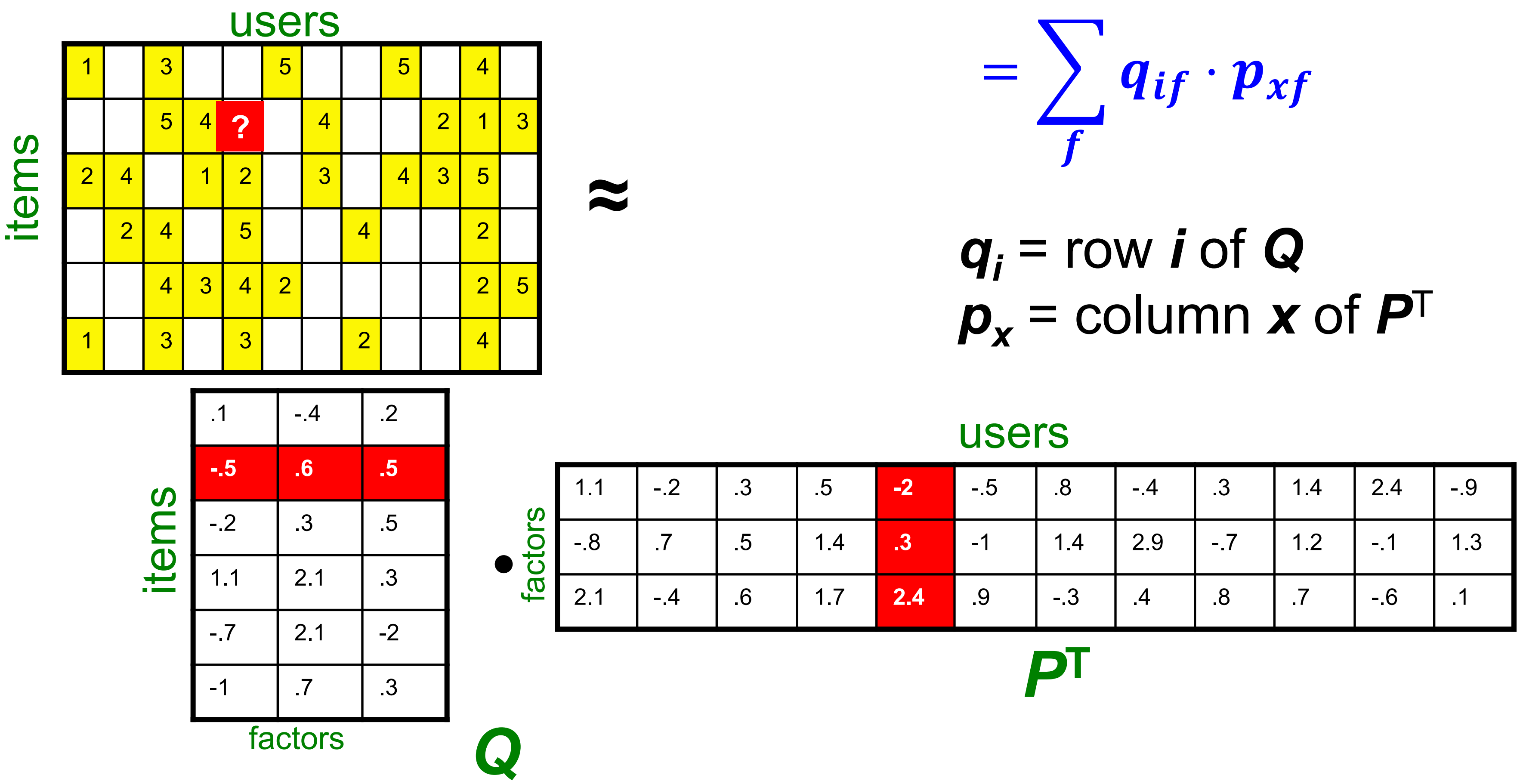
1.1	-.2	.3	.5	-2	-.5	.8	-.4	.3	1.4	2.4	-.9
-.8	.7	.5	1.4	.3	-1	1.4	2.9	-.7	1.2	-.1	1.3
2.1	-.4	.6	1.7	2.4	.9	-.3	.4	.8	.7	-.6	.1

P^T

Ratings as Products of Factors

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

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



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	

≈

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

•

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

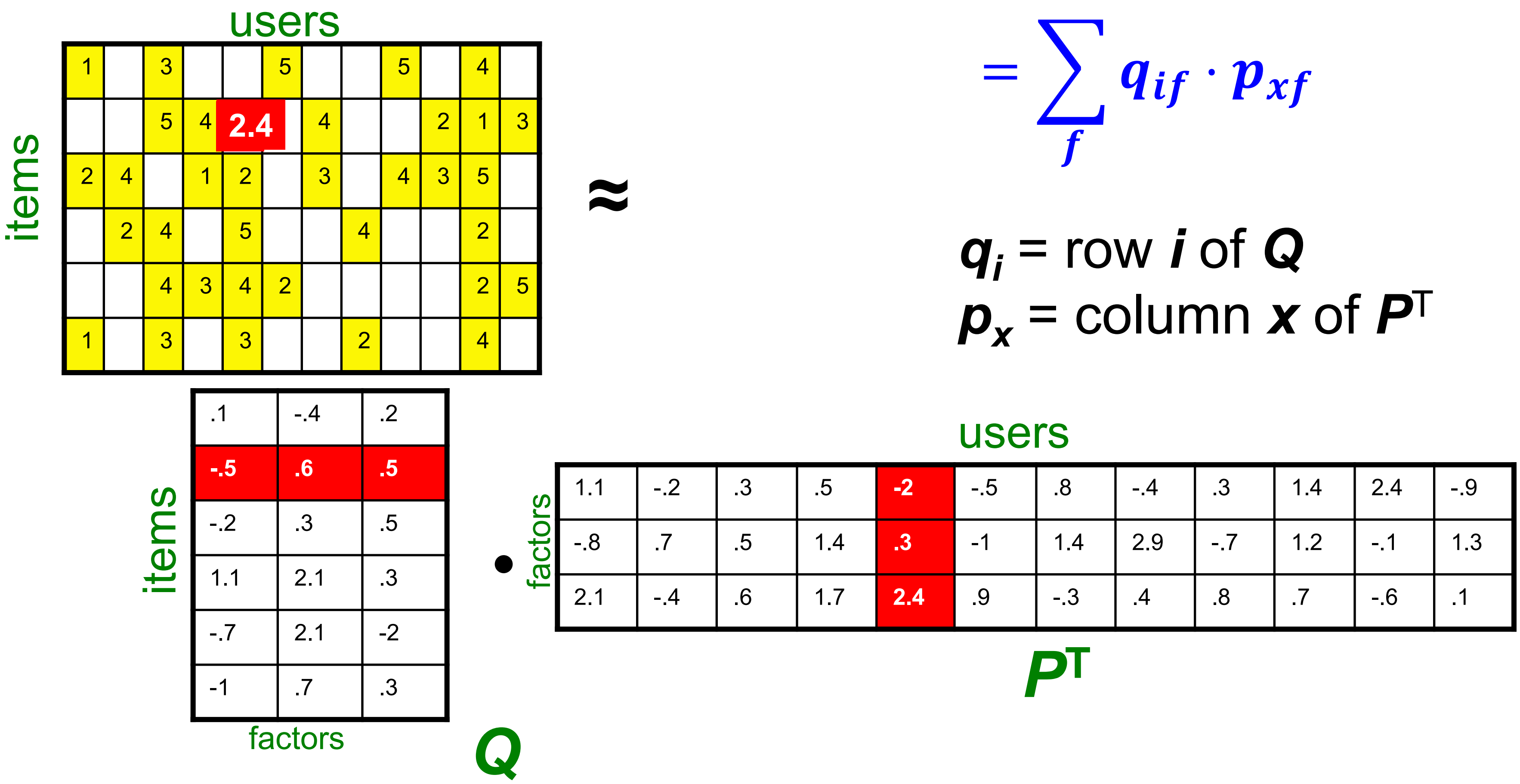
q_i = row ***i*** of ***Q***

p_x = column ***x*** of P^T

Ratings as Products of Factors

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

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



users

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	

≈

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

items

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

factors

Q

•

users

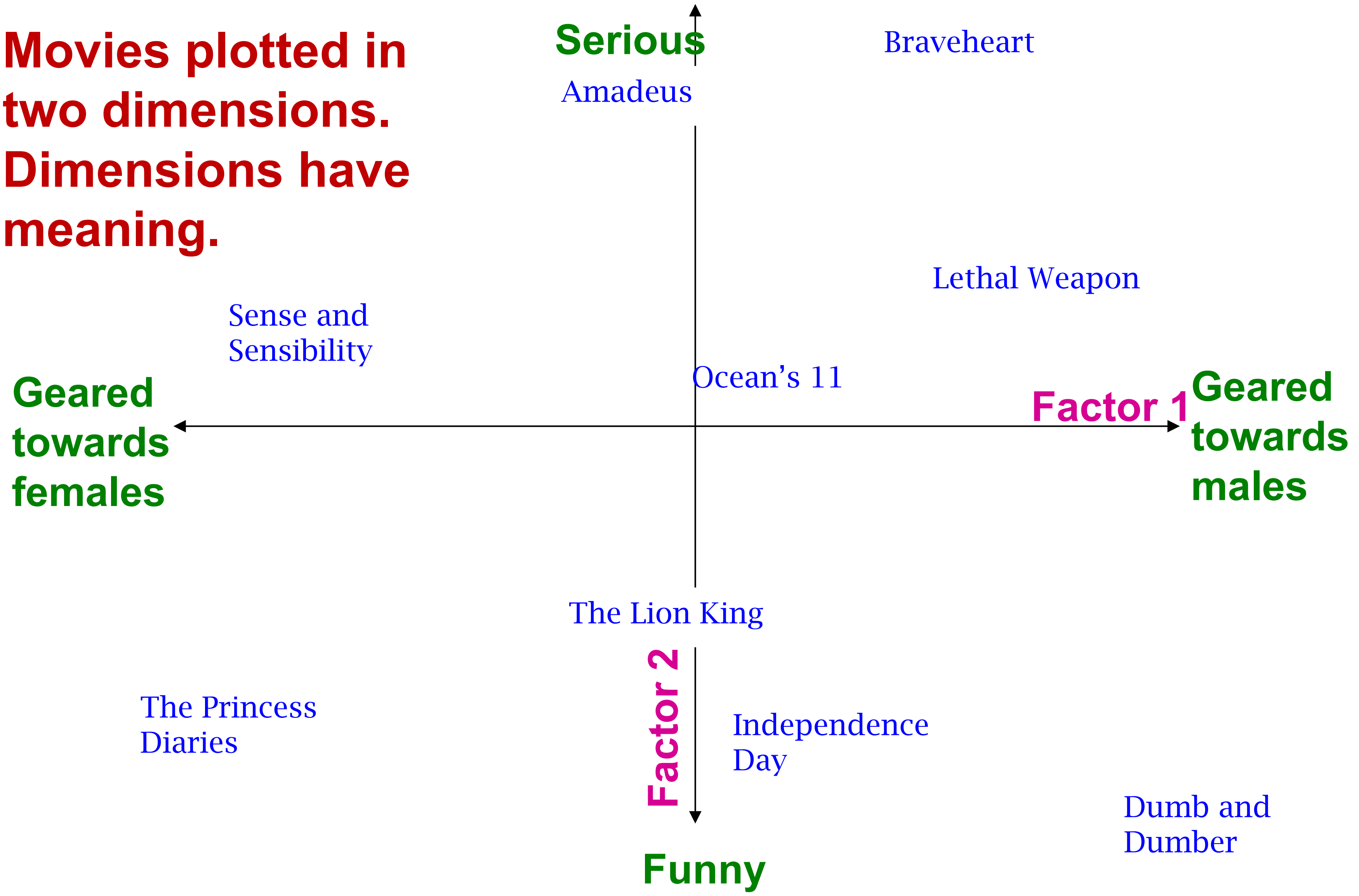
factors

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

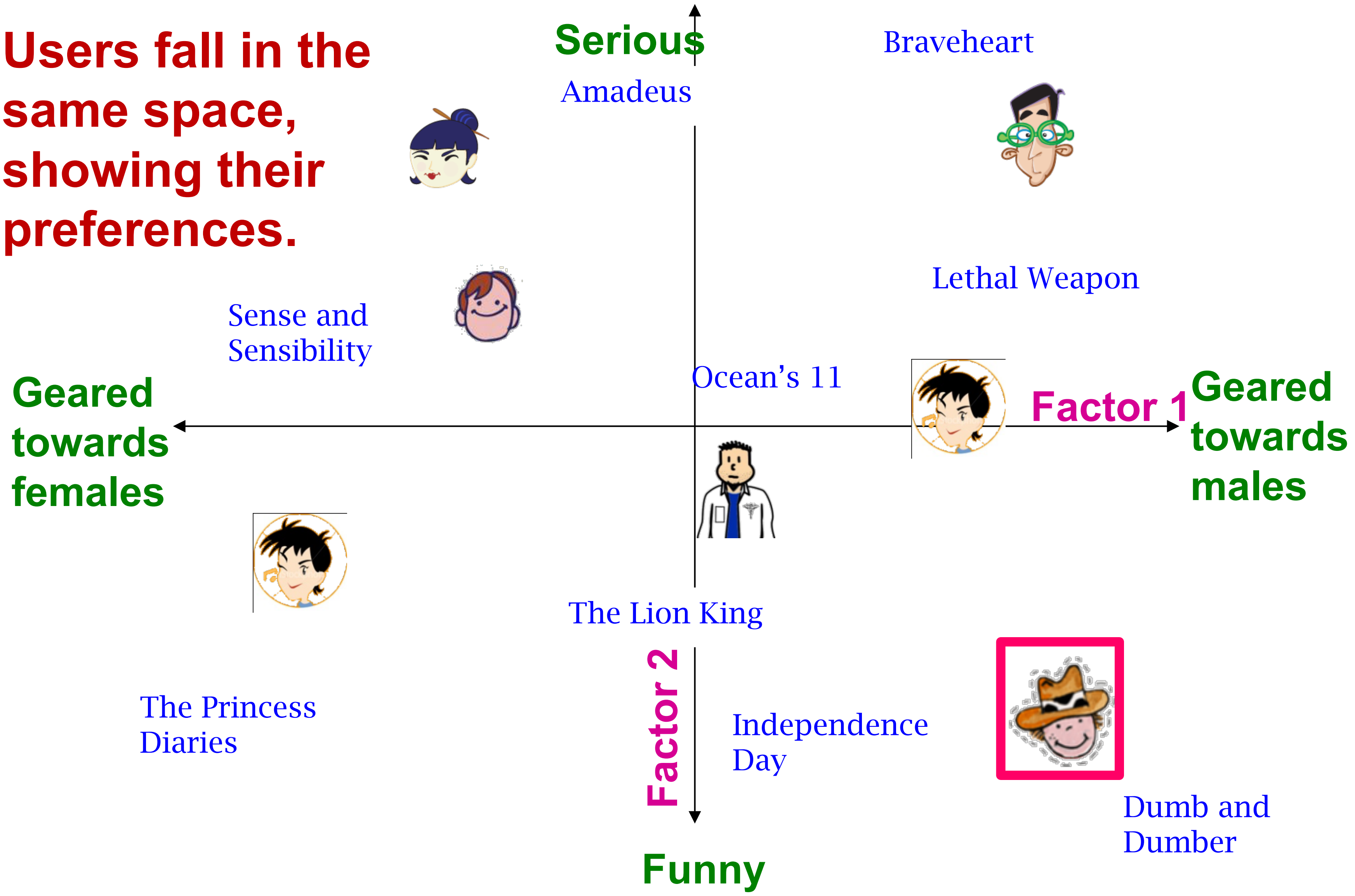
Latent Factor Models: Example

Movies plotted in two dimensions. Dimensions have meaning.



Latent Factor Models

Users fall in the same space, showing their preferences.



Serious

Amadeus

Braveheart

Lethal Weapon

Sense and Sensibility

Ocean’s 11

Geared towards females

Factor 1

Geared towards males

The Lion King

The Princess Diaries

Factor 2

Independence Day

Dumb and Dumber

Funny

Recommendation Systems

- ✓ Content-based
- ✓ Collaborative Filtering
- ✓ Latent Factor Models
- ☞ Deep Learning Models: NCF

- **Reference Paper:** **Neural Collaborative Filtering.** He Xiangnan, Liao Lizi, Zhang Hanwang, Nie Liqiang, Hu Xia, Tat-Seng Chua. WWW 2017

Matrix Factorization

- MF uses an **inner product as the interaction function**
  - – Latent factors are **independent** with each other
- **Limitations:** The simple choice of **inner product** function can limit the expressiveness of a MF model.
- **Potential solution:** increase the number of factors. However,
  - – This increases the complexity of the model
  - – Leads to overfitting

Improving Matrix Factorization

- **Key question:** How can we improve matrix factorization?
- **Answer:** Learn the relation between factors from the data, rather than fixing it to be the simple, fixed inner product
  - Does not increase the complexity
  - Does not lead to overfitting
- **One solution:** Neural Collaborative Filtering

Neural Collaborative Filtering

- **Neural Collaborative Filtering (NCF)** is a deep learning version of the traditional recommender system
- **Learns the interaction function** with a deep neural network
  - Non-linear functions, e.g., multi-layer perceptrons, to learn the interaction function
  - Models well when **latent factors are not independent** with each other, especially true in large real datasets

50

Neural Collaborative Filtering

- Neural extensions of traditional recommender system
- **Input:** rating matrix, user profile and item features (optional)
  - If user/item features are unavailable, we can use one-hot vectors
- **Output:** User and item embeddings, prediction scores
- Traditional matrix factorization is a special case of NCF

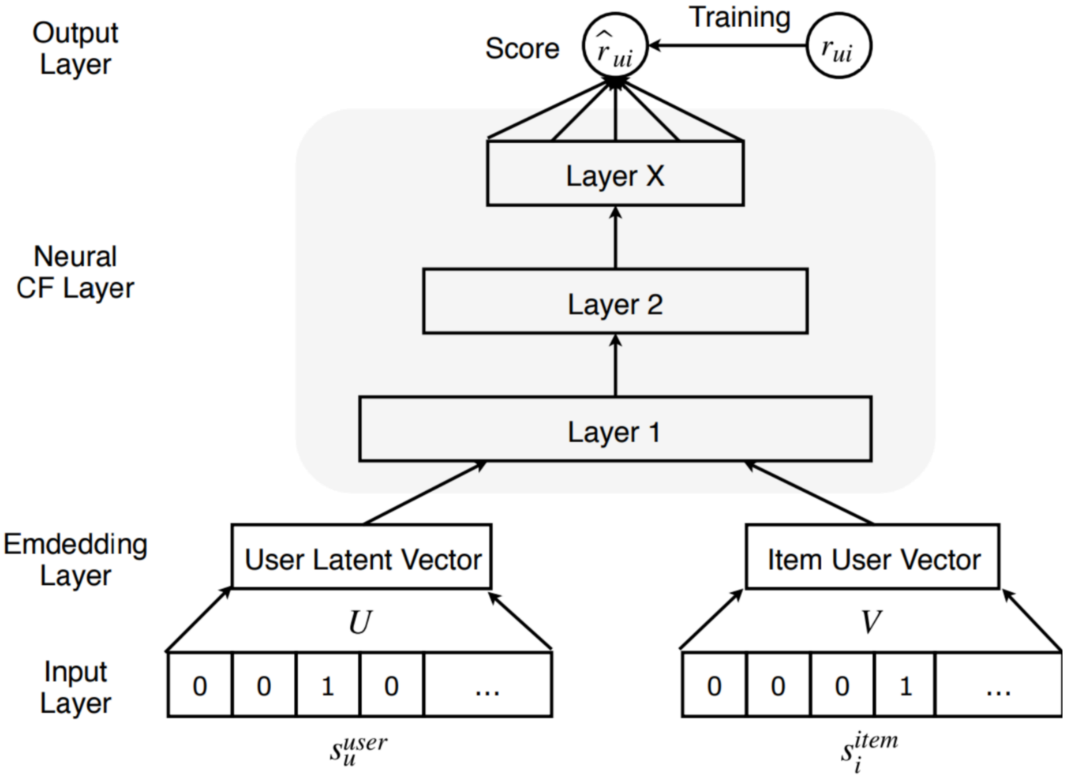
NCF Setup

- User feature vector: s_u^{user}
- Item feature vector: s_i^{item}
- User embedding matrix: U
- Item embedding matrix: I
- Neural network: f
- Neural network parameters: Θ
- Predicted rating:

$$\hat{r}_{ui} = f(U^T \cdot s_u^{user}, V^T \cdot s_i^{item} | U, V, \theta)$$

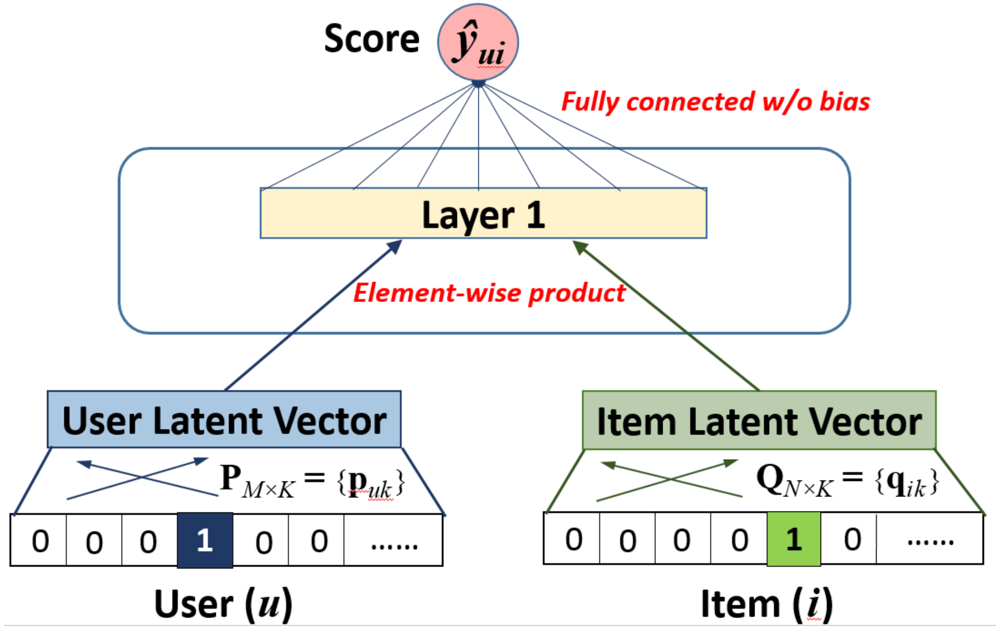
NCF Model Architecture

- **Multiple layers of fully connected layers** form the Neural CF layer.
- **Output is a rating score** $\hat{r}_{ui}$
- **Real rating score** is r_{ui}



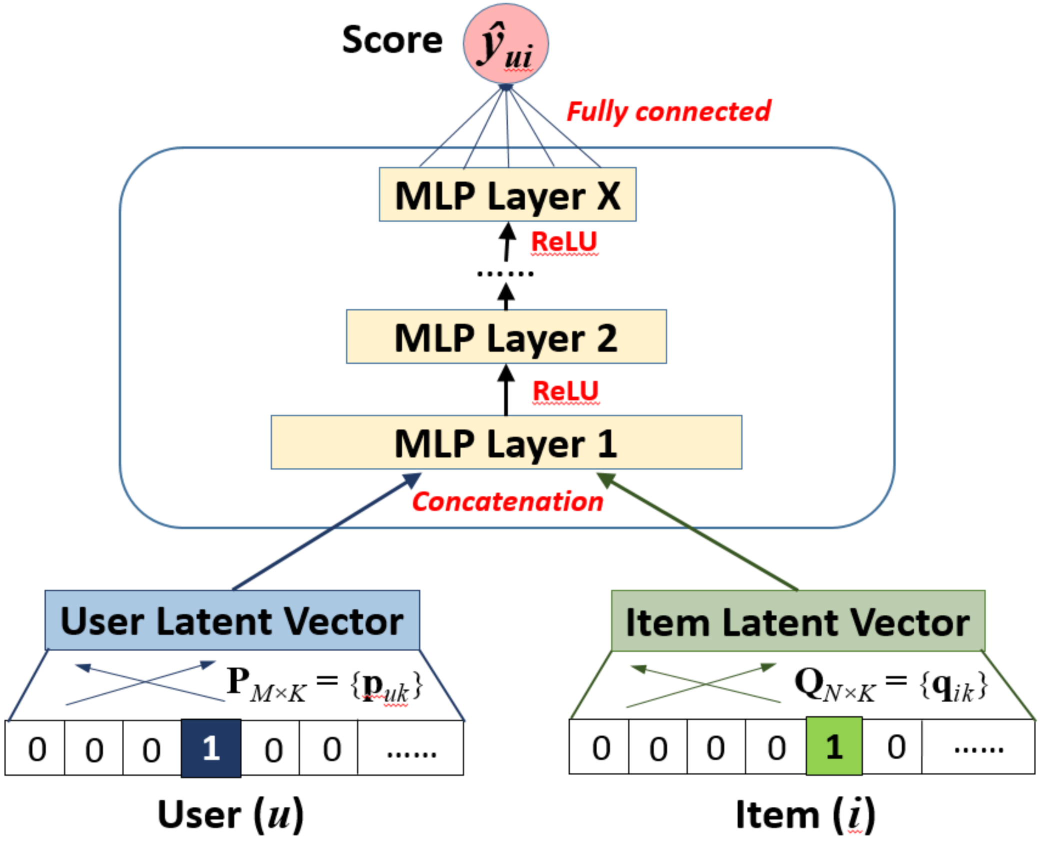
1-Layer NCF

- **Layer 1** an element-wise product
- **Output Layer** as a fully connected layer without bias



Multi-Layer NCF

- Each layer is a **multi-layer perceptron**, with non-linearity on the top
- Final score is used to **calculate the loss and train** the layers



NCF model: Loss function

- **Train** on the difference between predicted rating and the real rating
- **Use negative sampling** to reduce the negative data points
- **Loss = cross-entropy loss**

$$\mathcal{L} = -\sum_{(u,i)\in\mathcal{O}\cup\mathcal{O}^-} r_{ui}\log\hat{r}_{ui} + (1-r_{ui})\log(1-\hat{r}_{ui})$$

Experimental Setup

- **Two public datasets:** MovieLens, Pinterest
  - Transform MovieLens ratings to 0/1 implicit case

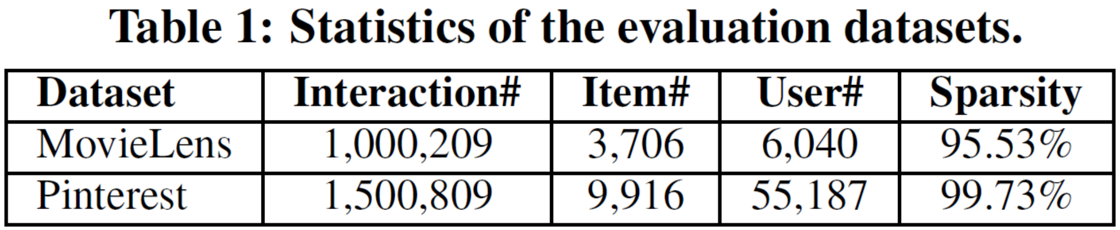


Table 1: Statistics of the evaluation datasets.

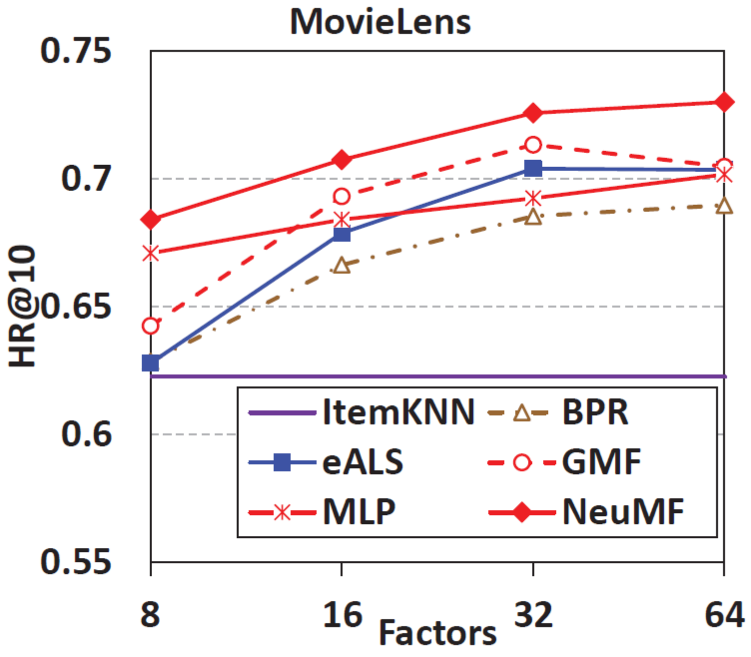
Dataset	Interaction#	Item#	User#	Sparsity
MovieLens	1,000,209	3,706	6,040	95.53%
Pinterest	1,500,809	9,916	55,187	99.73%

- **Evaluation protocols:**
  - **Leave-one-out setting:** hold-out the latest rating of each user as the test
  - **Top-k evaluation:** create a ranked list of items
  - **Evaluation metrics:**
    - **Hit Ratio:** does the correct item appear in top 10

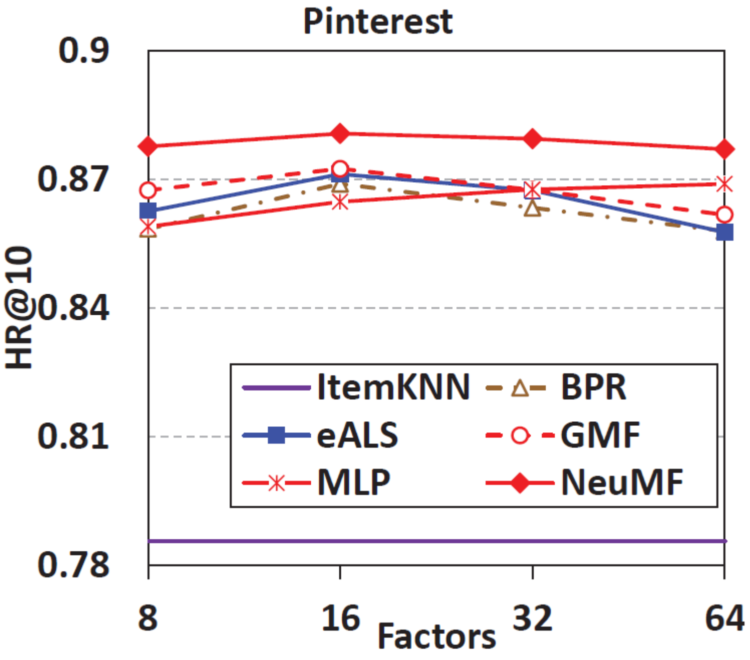
Baselines

- **Item Popularity**
  - Items are ranked by their popularity
- **ItemKNN** [Sarwar et al, WWW'01]
  - The standard item-based CF method
- **BPR** [Rendle et al, UAI'09]
  - Bayesian Personalized Ranking optimizes MF model with a pairwise ranking loss
- **eALS** [He et al, SIGIR'16]
  - The state-of-the-art CF method for implicit data. It optimizes MF model with a varying-weighted regression loss.

Performance vs. Embedding Size



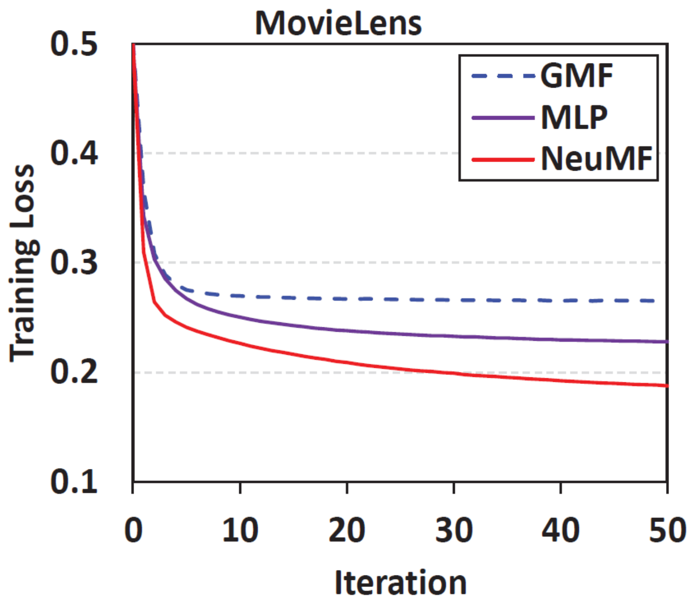
(a) MovieLens — HR@10



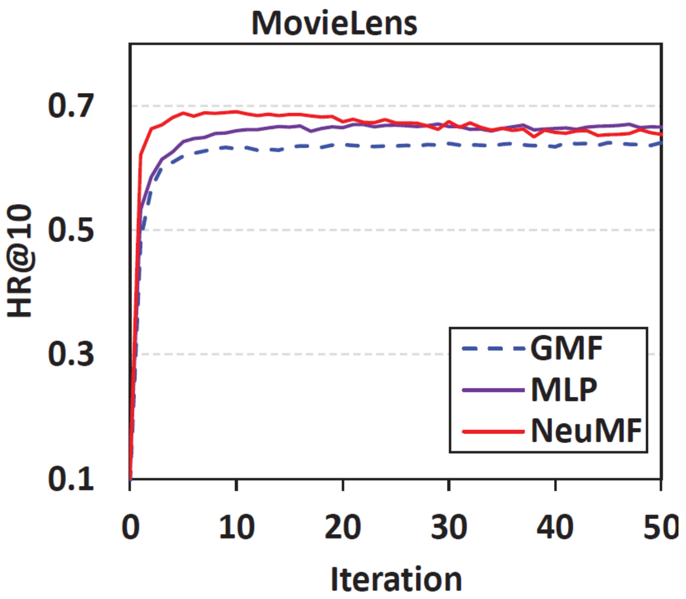
(c) Pinterest — HR@10

- **NeuMF > eALS and BPR** (5% improvement)
- **NeuMF > MLP** (MLP has lower training loss but higher test loss)

Convergence Behavior



(a) Training Loss



(b) HR@10

- Most effective updates in the **first 10 iterations**
- More iterations make **NeuMF overfit**
- **Trade-off** between representation ability and generalization ability of a model.

Is Deeper Helpful?

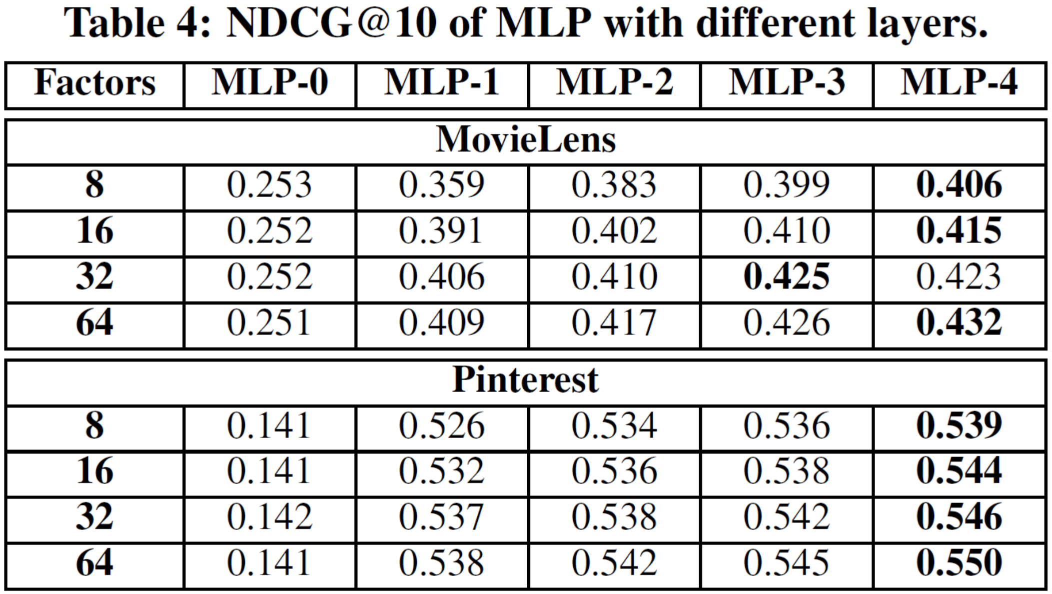


Table 4: NDCG@10 of MLP with different layers.

Factors	**MLP-0**	**MLP-1**	**MLP-2**	**MLP-3**	**MLP-4**
MovieLens					
8	0.253	0.359	0.383	0.399	**0.406**
16	0.252	0.391	0.402	0.410	**0.415**
32	0.252	0.406	0.410	**0.425**	0.423
64	0.251	0.409	0.417	0.426	**0.432**
Pinterest					
8	0.141	0.526	0.534	0.536	**0.539**
16	0.141	0.532	0.536	0.538	**0.544**
32	0.142	0.537	0.538	0.542	**0.546**
64	0.141	0.538	0.542	0.545	**0.550**

- Same number of factors, but more nonlinear layers improves the performance.
- Linear layers degrades the performance
- Improvement diminishes for more layers

NCF: Shortcomings

- Architecture is **limited**
- NCF **does not model the temporal behavior** of users or items
  - Recall: users and items exhibit temporal bias
  - NCF has the same input for user
- **Non-inductive:** new users and new items, on which training was not done, can not be processed