CSCE 670 - Information Storage and Retrieval

Week 6: Fundamentals of Recommender Systems

Yu Zhang

yuzhang@tamu.edu

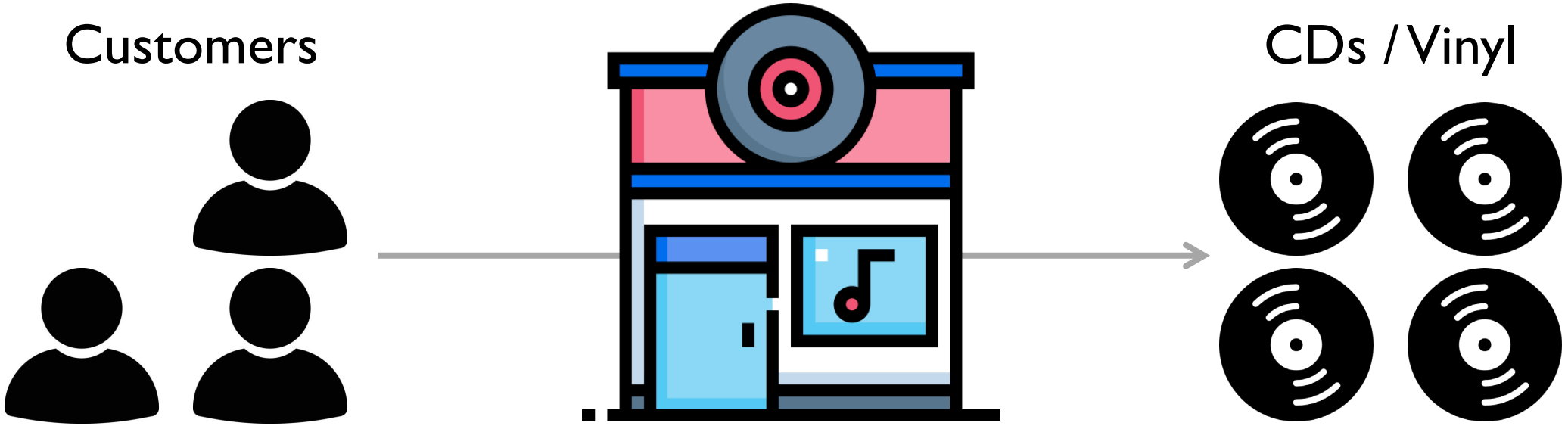
Course Website: https://yuzhang-teaching.github.io/CSCE670-S26.html

Adapted from the slides by Prof. Jure Leskovec (Stanford)

Recap: Phase 1

- ✅ Phase 1: Search Engines
  - basics, Boolean and ranked retrieval, link analysis, evaluation, learning to rank (ML + ranking), …
- Phase 2: Recommender Systems
  - basics, non-personalized recommendation, collaborative filtering, matrix factorization, implicit recommendation, …
- Phase 3: From Foundations to Modern Methods
  - embedding learning, Transformer, "small" language models, … (for search and recommendation)
- Phase 4: Large Language Models (!!)

Our Capabilities So Far

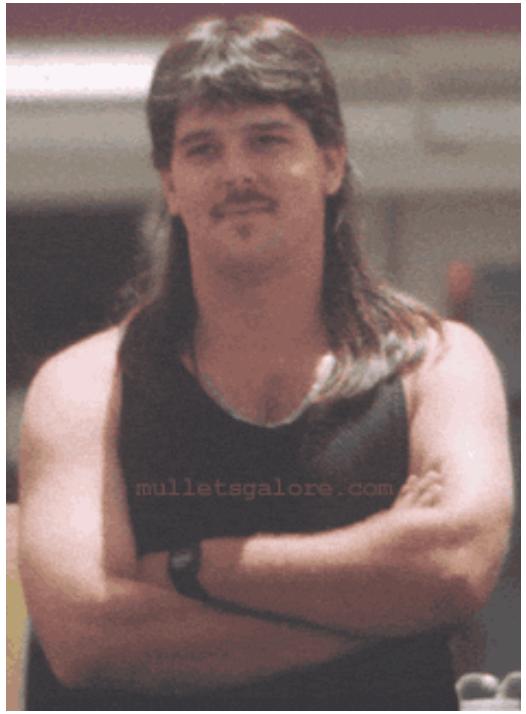


- Given a query, find *relevant* CDs
  - Exact matching (Boolean, phrase, proximity, wildcard)
  - Ranked retrieval (TF-IDF, BM25, learning to rank)
  - Link analysis (PageRank, HITS)

Phase 2

- ✅ Phase 1: Search Engines
  - basics, Boolean and ranked retrieval, link analysis, evaluation, learning to rank (ML + ranking), …
- Phase 2: Recommender Systems
  - basics, non-personalized recommendation, collaborative filtering, matrix factorization, implicit recommendation, …
- Phase 3: From Foundations to Modern Methods
  - embedding learning, Transformer, "small" language models, … (for search and recommendation)
- Phase 4: Large Language Models (!!)

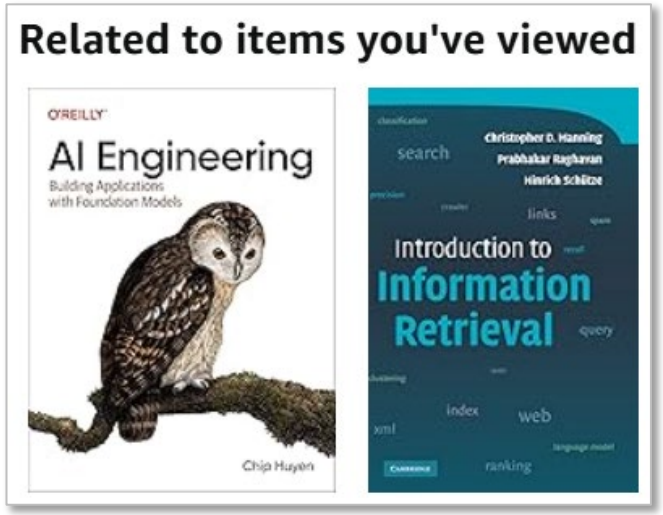
Example of Recommender Systems



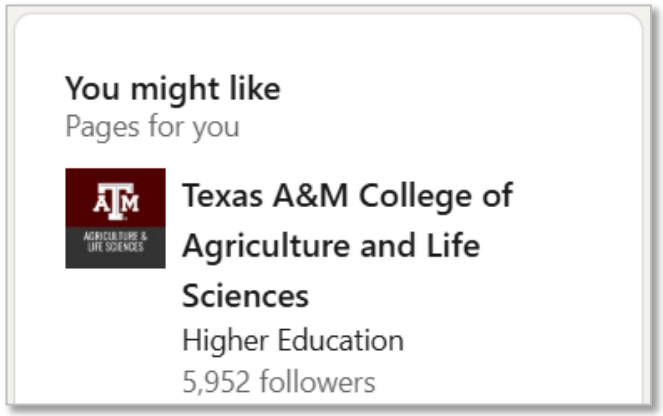
- Customer X
  - Buys Metallica CD
  - Buys Megadeth CD

- Customer Y
  - Does search on Metallica
  - Recommender systems should suggest Megadeth from data collected about Customer X

Amazon



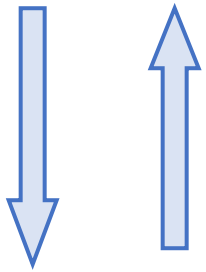
LinkedIn



Example of Recommender Systems

User

Search



Recommendation

Items: products,
websites, blogs, news, …

Examples:

amazon.com

PANDORA

StumbleUpon

del.icio.us

NETFLIX

movielens
helping you find the *right* movies

last·fm
the social music revolution

Google News

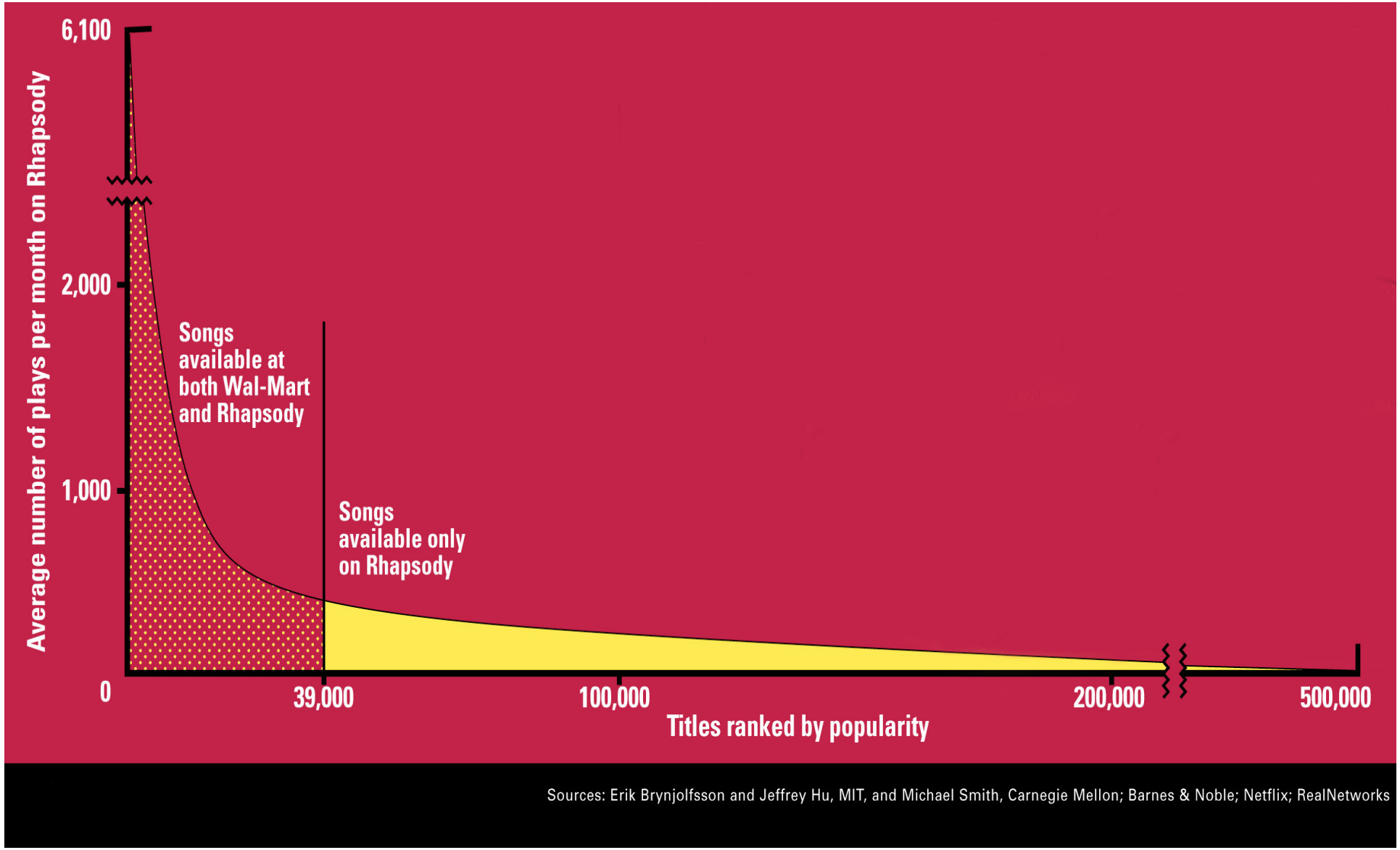
YouTube

XBOX LIVE

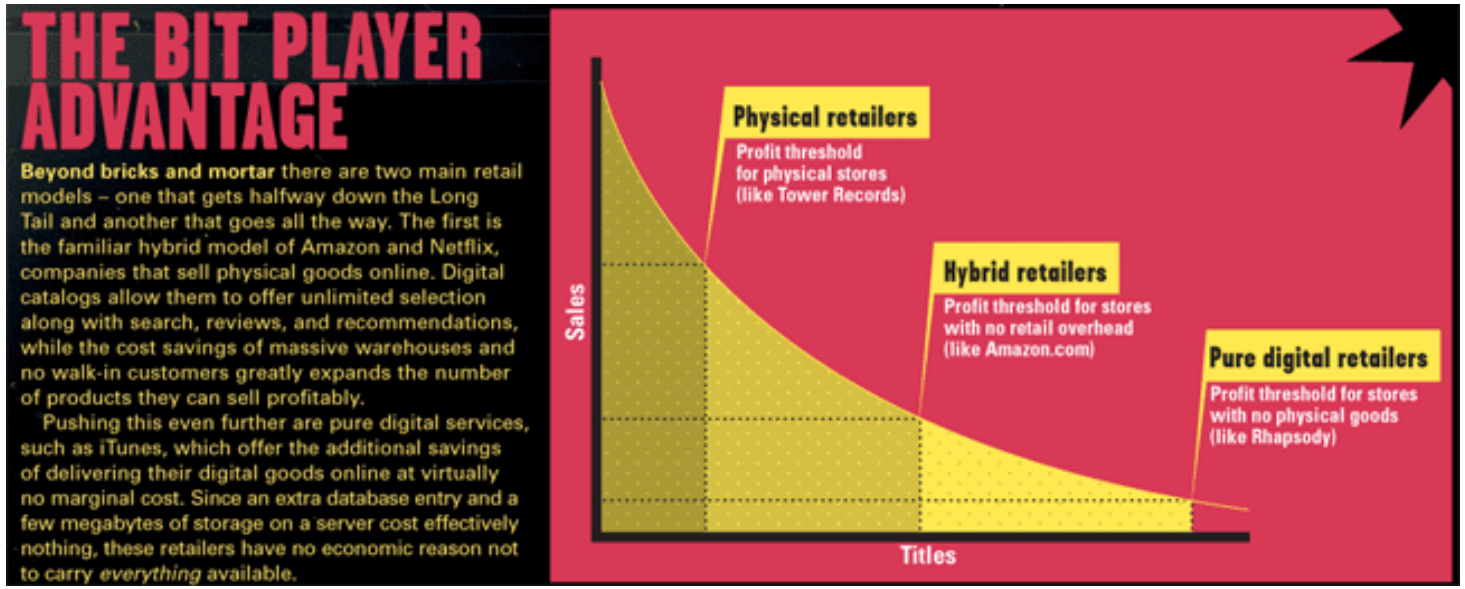
Why do we need recommendation?

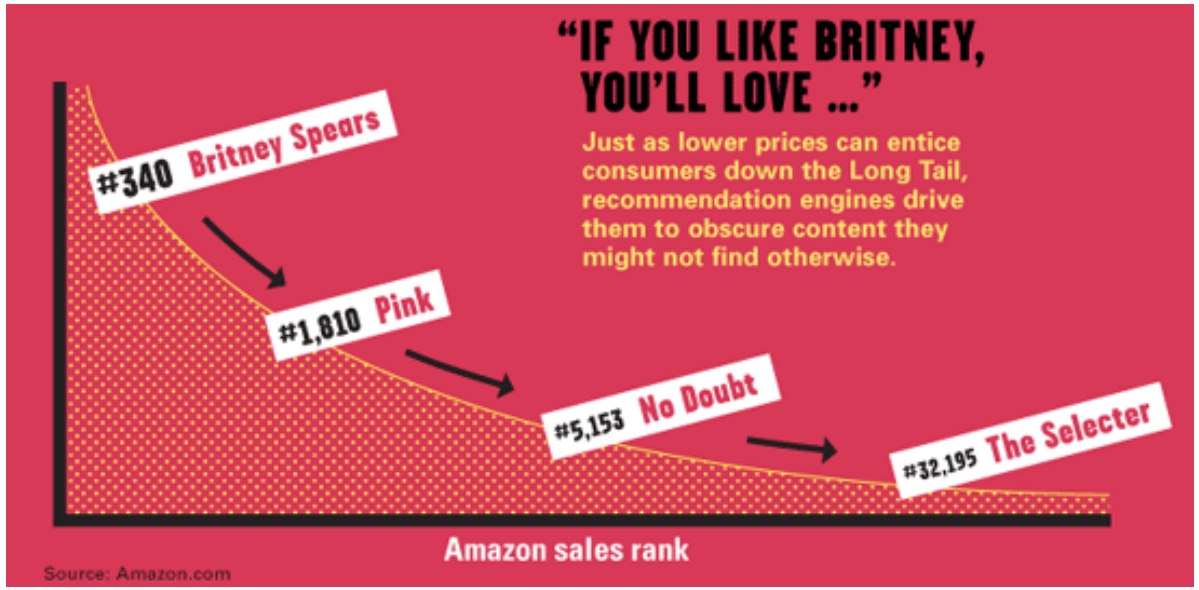
- Shelf space is a scarce commodity for traditional retailers
  - Also: TV networks, movie theaters,…
- Web enables near-zero-cost dissemination of information about products
  - From scarcity to abundance
- More choice necessitates better filters
  - How *Into Thin Air* (published in 1997) made *Touching the Void* (published in 1988) a bestseller
  - *Touching the Void* did not become a bestseller until a similar bestseller book appears 9 years later.
  - Amazon's recommendation engine

The Long Tail



The Long Tail





Types of Recommendations

- Editorial and hand-curated (not personalized)
  - "*Store Manager's Pick*"
  - Promoted items
- Simple aggregates (not personalized)
  - Most liked/clicked this month/week/day
  - Most recent
- Personalized approaches
  - Content-Based (this week)
  - Collaborative Filtering (this week)
  - Matrix Factorization (this week and next week)
  - Bayesian Personalized Ranking (next week)

Formal Setup

- $\mathcal{X}$: A set of users
- $\mathcal{S}$: A set of items

- Utility function $u: \mathcal{X} \times \mathcal{S} \rightarrow \mathcal{R}$
  - $\mathcal{R}$ = set of ratings, which is a totally ordered set
    - E.g., 1-5 stars
    - E.g., real number in [0,1]

Utility Matrix U

	Avatar	LOTR	Matrix	Pirates
Alice	1		0.2	
Bob		0.5		0.3
Carol	0.2		1	
David				0.4

Key Problems

- Problem 1: Gathering "known" ratings for matrix
  - How to collect the data in the utility matrix?
  - Evaluating the quality of an item solely based on its average rating?
- Problem 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
- Problem 3: Evaluating extrapolation methods
  - How to measure success/performance of recommendation methods?

Problem 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?

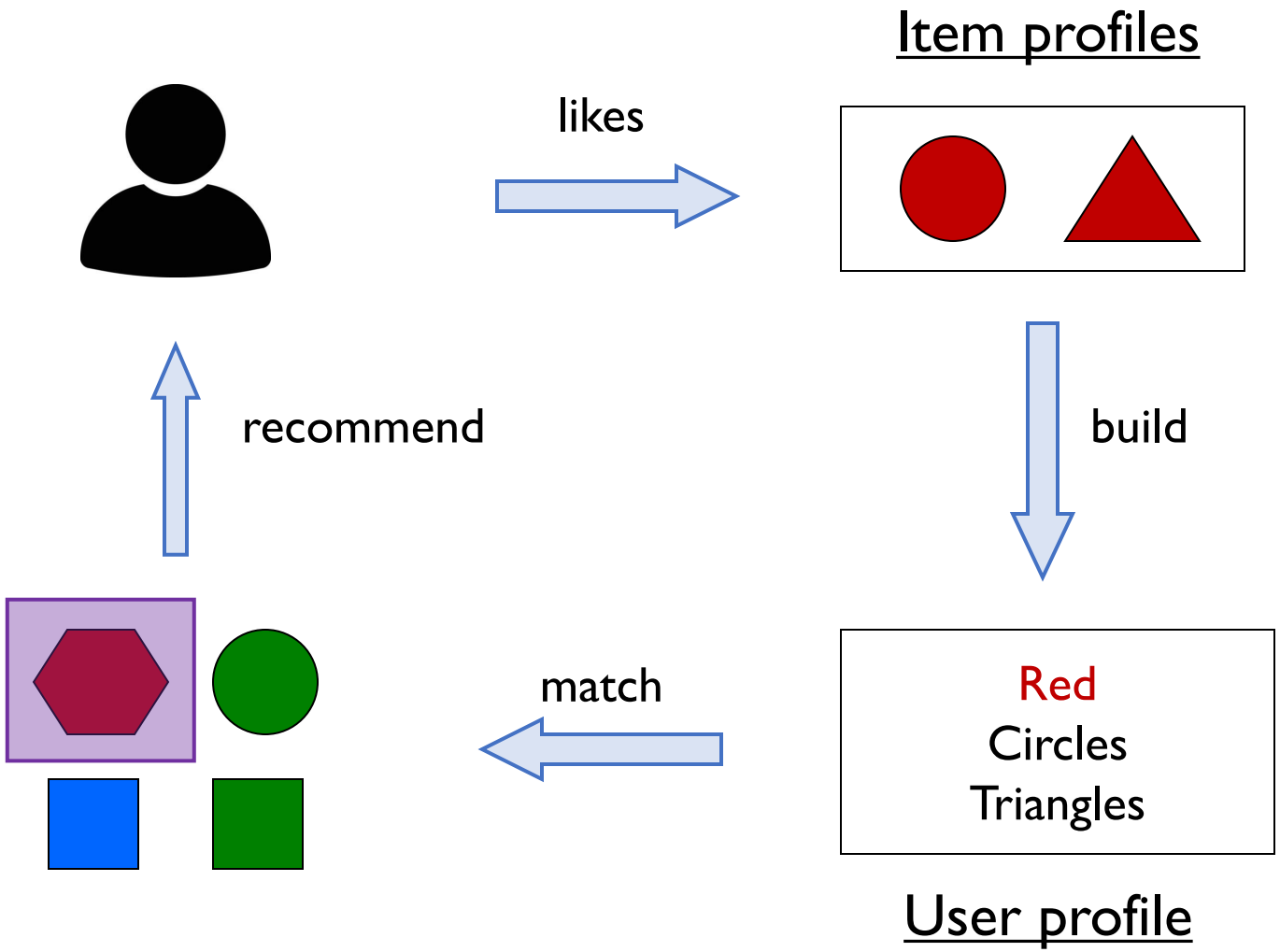
Problem 2: Extrapolating Utilities

- 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
- Solutions to be introduced today:
  - Content-Based Approach
  - Collaborative Filtering

Content-Based Approach
(Calculating User-Item Similarity)

Content-Based Recommender Systems

- Idea: Recommend items to user 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



Profiles

- Item Profile $\boldsymbol{i}$
  - A set (vector) of features
  - Movies: author, title, actor, director, …
  - Text: Set of "important" words in document (e.g., based on TF-IDF)
- User Profile $\boldsymbol{x}$
  - Weighted average of rated item profiles
- Prediction
  - Given $\boldsymbol{x}$ and $\boldsymbol{i}$, estimate $u(\boldsymbol{x}, \boldsymbol{i}) = \cos(\boldsymbol{x}, \boldsymbol{i}) = \frac{\boldsymbol{x}^T \boldsymbol{i}}{\|\boldsymbol{x}\| \cdot \|\boldsymbol{i}\|}$

Example

- Rating scale: $\mathcal{R} = \{1, 0, -1\}$
- User x has rated 3 songs
  - Song 1: "*sunny day*", rating: 1
  - Song 2: "*cloudy day*", rating: 0
  - Song 3: "*rainy day*", rating: -1
- Let's simplify the model and use Boolean representation (rather than TF-IDF)
- Item profiles

	sunny	*cloudy*	*rainy*	*day*
Song 1	1	0	0	1
Song 2	0	1	0	1
Song 3	0	0	1	1

Example

- User x has rated 3 songs
  - Song 1:"*sunny day*", rating: 1
  - Song 2:"*cloudy day*", rating: 0
  - Song 3:"*rainy day*", rating: -1
- Item profiles

	sunny	*cloudy*	*rainy*	*day*
Song 1	1	0	0	1
Song 2	0	1	0	1
Song 3	0	0	1	1

- User profile: User $x = 1 \times$ Song 1 $+ 0 \times$ Song 2 $+ (-1) \times$ Song 3

	sunny	*cloudy*	*rainy*	*day*
User x	1	0	-1	0

Example

- User profile: User $x = 1 \times$ Song 1 $+ 0 \times$ Song 2 $+ (-1) \times$ Song 3

	sunny	*cloudy*	*rainy*	*day*
User x	1	0	-1	0

- New Song: "*sunny cloudy*"
- Will User x like it?

- Profile of New Song

	sunny	*cloudy*	*rainy*	*day*
User x	1	1	0	0

- $u(\text{User } x, \text{New Song}) = \cos\left(\begin{bmatrix}1\\0\\-1\\0\end{bmatrix}, \begin{bmatrix}1\\1\\0\\0\end{bmatrix}\right) = \frac{1}{\sqrt{2}\times\sqrt{2}} = \frac{1}{2}$

Content-Based Recommender Systems: Pros

- + 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

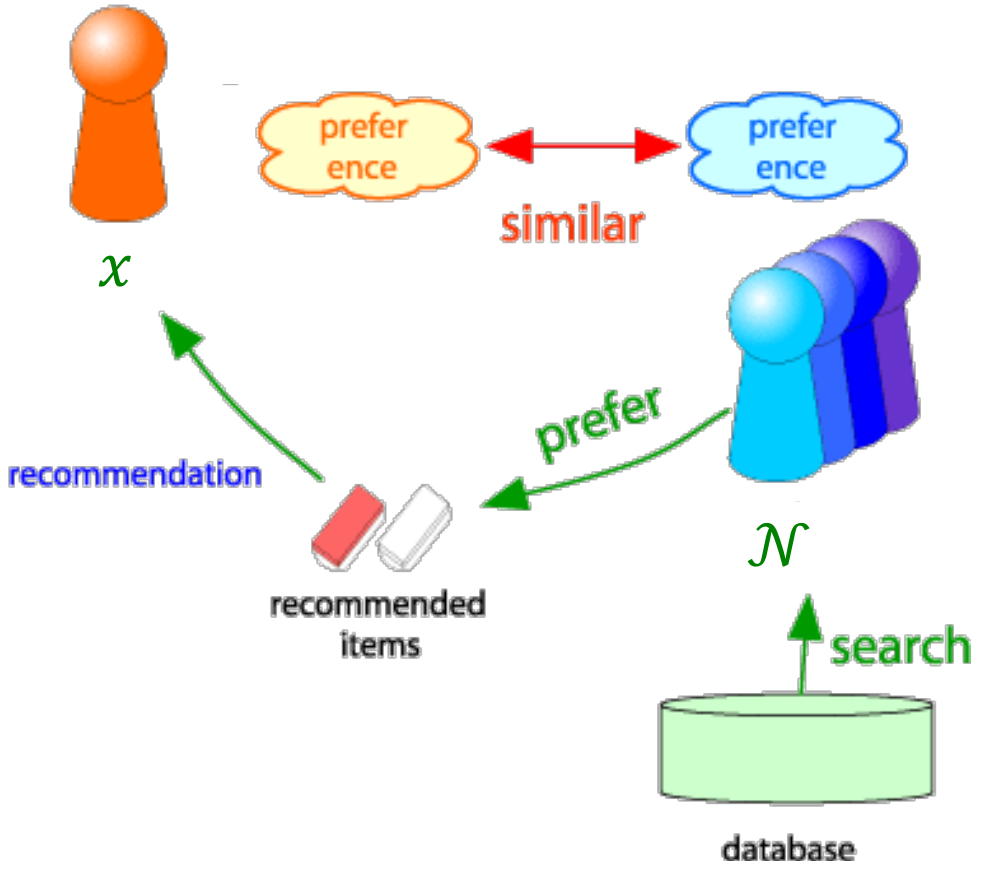
Content-Based Recommender Systems: Cons

- – 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

Collaborative Filtering
(Harnessing Quality Judgments of Other Users)

Collaborative Filtering

- Consider user x
- Step 1: Find a set $\mathcal{N}$ of other users whose ratings are "similar" to x's ratings
- Step 2: Estimate x's ratings based on ratings of users in $\mathcal{N}$



Step 1: Finding Similar Users

- How to define User-User similarity?
- Example:

	Item 1	Item 2	Item 3	Item 4	Item 5
User x	*			*	***
User y	*		**	**	

- (Bad) Solution 1: Jaccard Similarity (between two sets)

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

x: $\{1,4,5\}$, y: $\{1,3,4\}$, $J(x,y) = \frac{1}{2}$

Problem: Ignores the value of the rating

Step 1: Finding Similar Users

- How to define User-User similarity?
- Example:

	Item 1	Item 2	Item 3	Item 4	Item 5
User x	*			*	***
User y	*		**	**	

- (Bad) Solution 2: Cosine Similarity (between two vectors)
  - x: $[1, 0, 0, 1, 3]^T$
  - y: $[1, 0, 2, 2, 0]^T$
  - Problem: Treats missing ratings as "negative"

Step 1: Finding Similar Users

- How to define User-User similarity?
- Example:

	Item 1	Item 2	Item 3	Item 4	Item 5
User x	*			*	***
User y	*		**	**	

- Solution 3: Pearson Correlation Coefficient
  - Consider S_{xy} = items rated by both users x and y

$$sim(x,y) = \frac{\sum_{i \in S_{xy}} (U_{xi} - \bar{U}_x)(U_{yi} - \bar{U}_y)}{\sqrt{\sum_{i \in S_{xy}} (U_{xi} - \bar{U}_x)^2} \sqrt{\sum_{i \in S_{xy}} (U_{yi} - \bar{U}_y)^2}}$$

$\bar{U}_x, \bar{U}_y$: average rating given by x and y

How to understand the Pearson Correlation Coefficient?

$$sim(x,y) = \frac{\sum_{i \in S_{xy}} (U_{xi} - \bar{U}_x)(U_{yi} - \bar{U}_y)}{\sqrt{\sum_{i \in S_{xy}} (U_{xi} - \bar{U}_x)^2} \sqrt{\sum_{i \in S_{xy}} (U_{yi} - \bar{U}_y)^2}}$$

- Original Table

	Item 1	Item 2	Item 3	Item 4	Item 5
User x	*			*	***
User y	*		**	**	

- Step 1: Subtract the (row) mean

	Item 1	Item 2	Item 3	Item 4	Item 5
User x	1-5/3=-2/3			1-5/3=-2/3	3-5/3=4/3
User y	1-5/3=-2/3		2-5/3=1/3	2-5/3=1/3	

How to understand the Pearson Correlation Coefficient?

$$sim(x,y) = \frac{\sum_{i \in S_{xy}} (U_{xi} - \bar{U}_x)(U_{yi} - \bar{U}_y)}{\sqrt{\sum_{i \in S_{xy}} (U_{xi} - \bar{U}_x)^2} \sqrt{\sum_{i \in S_{xy}} (U_{yi} - \bar{U}_y)^2}}$$

- Original Table

	Item 1	Item 2	Item 3	Item 4	Item 5
User x	*			*	***
User y	*		**	**	

- Step 2: Only keep the column rated by both x and y

	Item 1	Item 4
User x	1-5/3=-2/3	1-5/3=-2/3
User y	1-5/3=-2/3	2-5/3=1/3

- Step 3: Calculate the Cosine Similarity
  - Pearson is equivalent to Cosine after some data normalization steps!

Step 2: Rating Prediction

- Let $\mathcal{N}$ be the set of k users that are most similar to x who have rated item i
- Prediction for item i of user x:
  - Simple average:

$$U_{xi} = \frac{1}{k} \sum_{y \in \mathcal{N}} U_{yi}$$

  - Weighted by User-User similarity:

$$U_{xi} = \frac{\sum_{y \in \mathcal{N}} sim(x, y) \cdot U_{yi}}{\sum_{y \in \mathcal{N}} sim(x, y)}$$

  - Many other tricks possible…

- So far, User-User Collaborative Filtering
  - Using User-User similarity to predict User-Item similarity

How about Item-Item Collaborative Filtering?

- Using Item-Item similarity to predict User-Item similarity

- Step 1: For item i, find other similar items
- Step 2: Estimate rating for item i based on ratings for similar items

- Can use same similarity metrics and prediction functions as in user-user model
  - E.g., Weighted by Item-Item similarity:

$$U_{xi} = \frac{\sum_{j \in \mathcal{N}} sim(i,j) \cdot U_{xj}}{\sum_{j \in \mathcal{N}} sim(i,j)}$$

$sim(i,j)$: Pearson Correlation Coefficient between item i and item j

$\mathcal{N}$: the set of items rated by x that are similar to i

Example ($|\mathcal{N}| = 2$)

users

CDs

	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

Example ($|\mathcal{N}| = 2$)

users

CDs

	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 CD 1 by user 5

Example ($|\mathcal{N}| = 2$)

users

CDs

	1	2	3	4	**5**	6	7	8	9	10	11	12	$sim(1,\cdot)$
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 that are similar to CD 1, rated by user 5

Example ($|\mathcal{N}| = 2$)

users

CDs

	1	2	3	4	5	6	7	8	9	10	11	12	$sim(1,\cdot)$
1	1		3		**2.6**	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:

$$U_{51} = (0.41 \times 2 + 0.59 \times 3) \,/\, (0.41 + 0.59) = 2.6$$

Collaborative Filtering: Common Practice

- So far,

$$U_{xi} = \frac{\sum_{j \in \mathcal{N}} sim(i,j) \cdot U_{xj}}{\sum_{j \in \mathcal{N}} sim(i,j)}$$

- In practice,

$$U_{xi} = b_{xi} + \frac{\sum_{j \in \mathcal{N}} sim(i,j) \cdot (U_{xj} - b_{xj})}{\sum_{j \in \mathcal{N}} sim(i,j)}$$

  - b_{xi}: baseline estimate for U_{xi} $\quad (b_{xi} = \mu + b_x + b_i)$
  - μ: overall mean CD rating
  - b_x: rating deviation of user x, which is the (avg. rating given by user x) – μ
  - b_i: rating deviation of item i, which is the (avg. rating given to item i) – μ

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
  - It is impossible for a piece of music to be both 60's rock and 1700's baroque.
  - There are individuals who like both 60's rock and 1700's baroque, and who buy examples of both types of music.

	Avatar	LOTR	Matrix	Pirates
Alice	1		0.8	
Bob		0.5		0.3
Carol	0.9		1	0.8
David			1	0.4

Collaborative Filtering: Pros and Cons

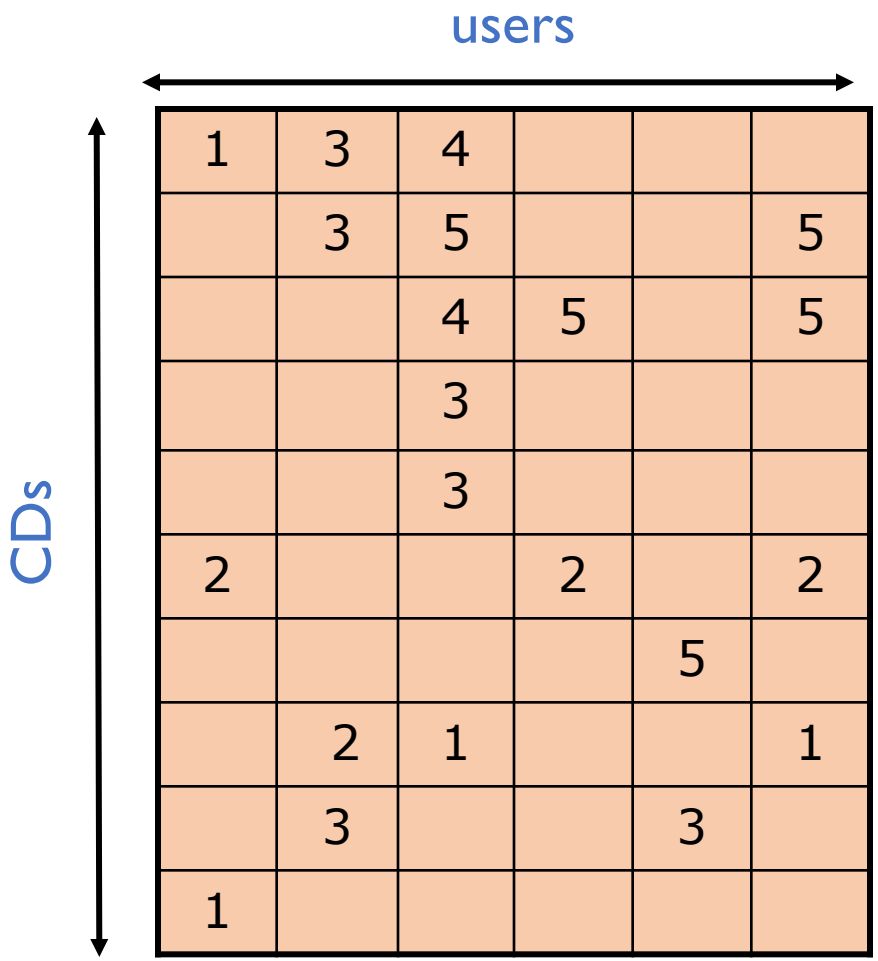
- + Works for any kind of item
  - No feature selection (e.g., text information) needed
- - Cold start
  - Need enough users in the system to find a match
- - Sparsity
  - The user/ratings matrix is sparse
  - Hard to find users that have rated the same items
- - First rater
  - Cannot recommend an item that has not been previously rated
  - New items & esoteric items
- - Popularity bias
  - Cannot recommend items to someone with unique taste
  - Tends to recommend popular items

Hybrid Methods

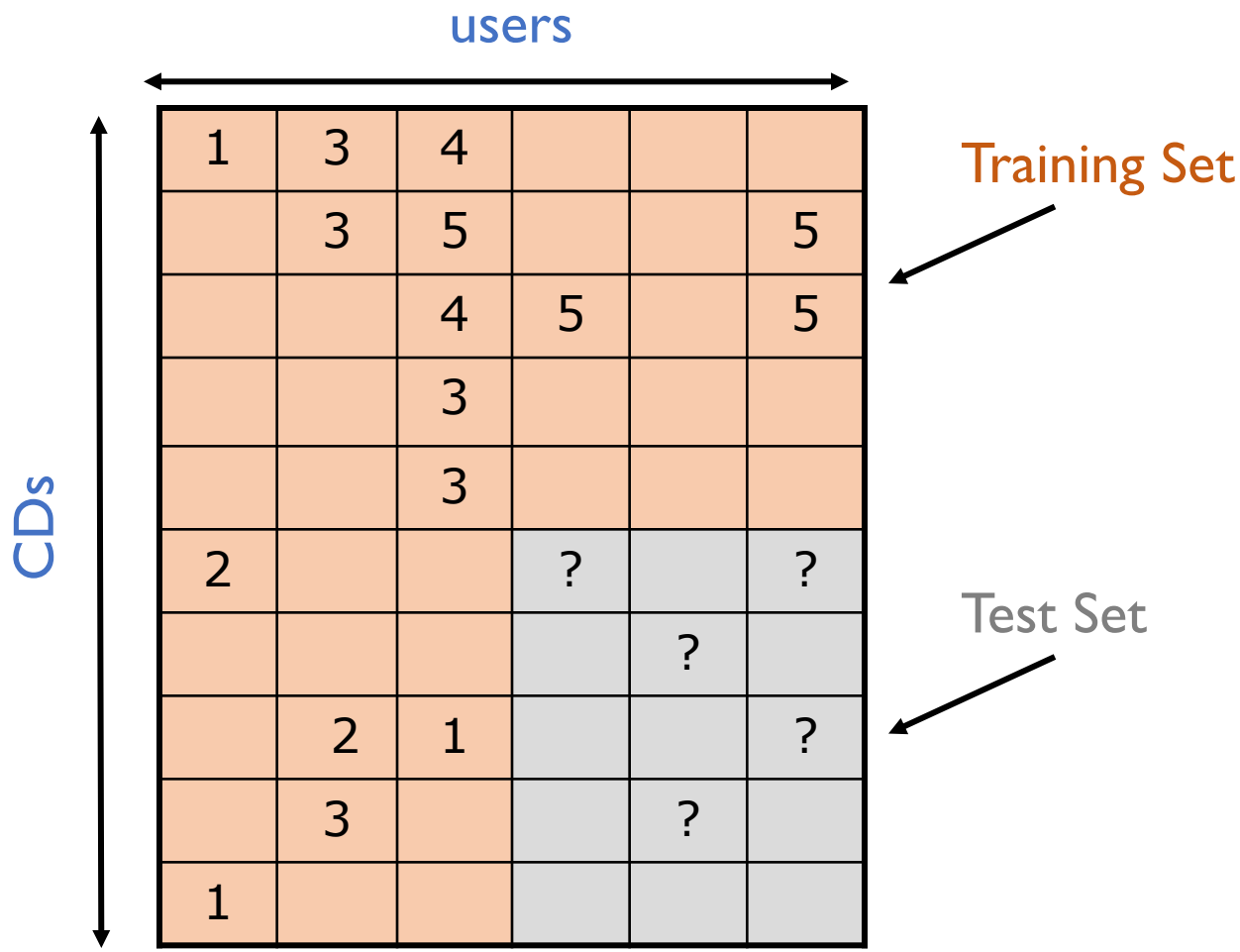
- Implement two or more different recommenders (e.g., content-based and collaborative filtering) and combine predictions
  - Perhaps using a linear model
  - "*Learning to Recommend*"
- Add content-based approaches to collaborative filtering
  - Building item profiles to deal with the new item problem
  - Building demographics to deal with the new user problem

Evaluation of Recommender Systems

Utility Matrix



Training vs. Testing



Evaluation Metrics

- Compare predictions with *known* ratings. (There are many predictions whose ground truth is *unknown*.)
  - Root-mean-square error (RMSE)
    - $\sqrt{\frac{1}{M}\sum_{x,i}\left(U_{xi} - U_{xi}^{*}\right)^2}$ where U_{xi} is predicted, and U_{xi}^{*} is the true rating of x on i; M is the number of testing samples
  - Precision@10
    - $\frac{\text{Among the top 10 items with known ratings, how many are relevant (e.g., ≥4 stars)}}{10}$
  - nDCG@10
  - Recall@10
    - $\frac{\text{Among the top 10 items with known ratings, how many are relevant (e.g., ≥4 stars)}}{\text{Among ALL items with known ratings, how many are relevant (e.g., ≥4 stars)}}$

Additional Comments

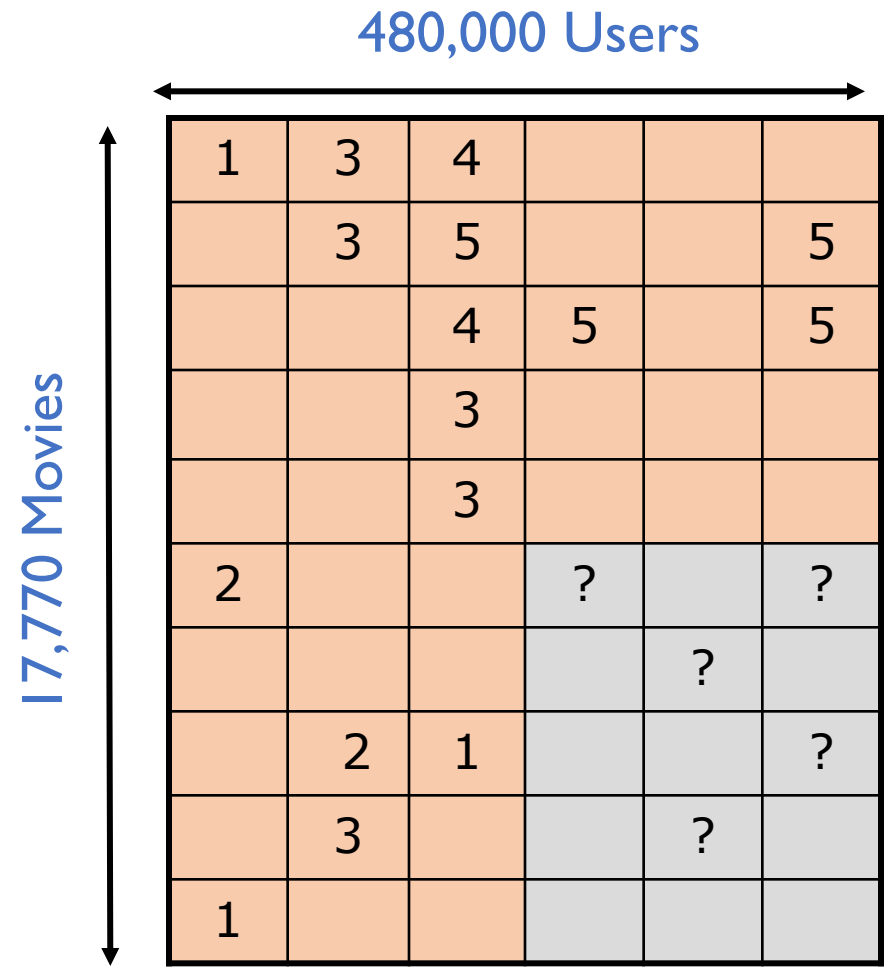
- Problem with RMSE: In practice, our main interest is in accurately predicting high ratings.
  - It is far more important to predict whether you would give 5 stars or 4 stars to a CD you might like than to predict whether you would give 2 stars or 1 star to one you dislike.
  - However, RMSE may penalize methods that perform well on high ratings but poorly on others.

- Tip: Leverage ALL the data
  - Don't try to reduce data size in an effort to make fancy algorithms work
  - Simple methods on large data do best
  - More data beats better algorithms: http://anand.typepad.com/datawocky/2008/03/more-data-usual.html

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: RMSE
    - $\sqrt{\frac{1}{M}\sum_{x,i}\left(U_{xi} - U_{xi}^{*}\right)^2}$ where U_{xi} is predicted, and U_{xi}^{*} is the true rating of x on i; M is the number of testing samples
    - Netflix's system RMSE: 0.9514
- Competition
  - 2,700+ teams
  - $1 million prize for 10% improvement on Netflix

The Netflix Prize



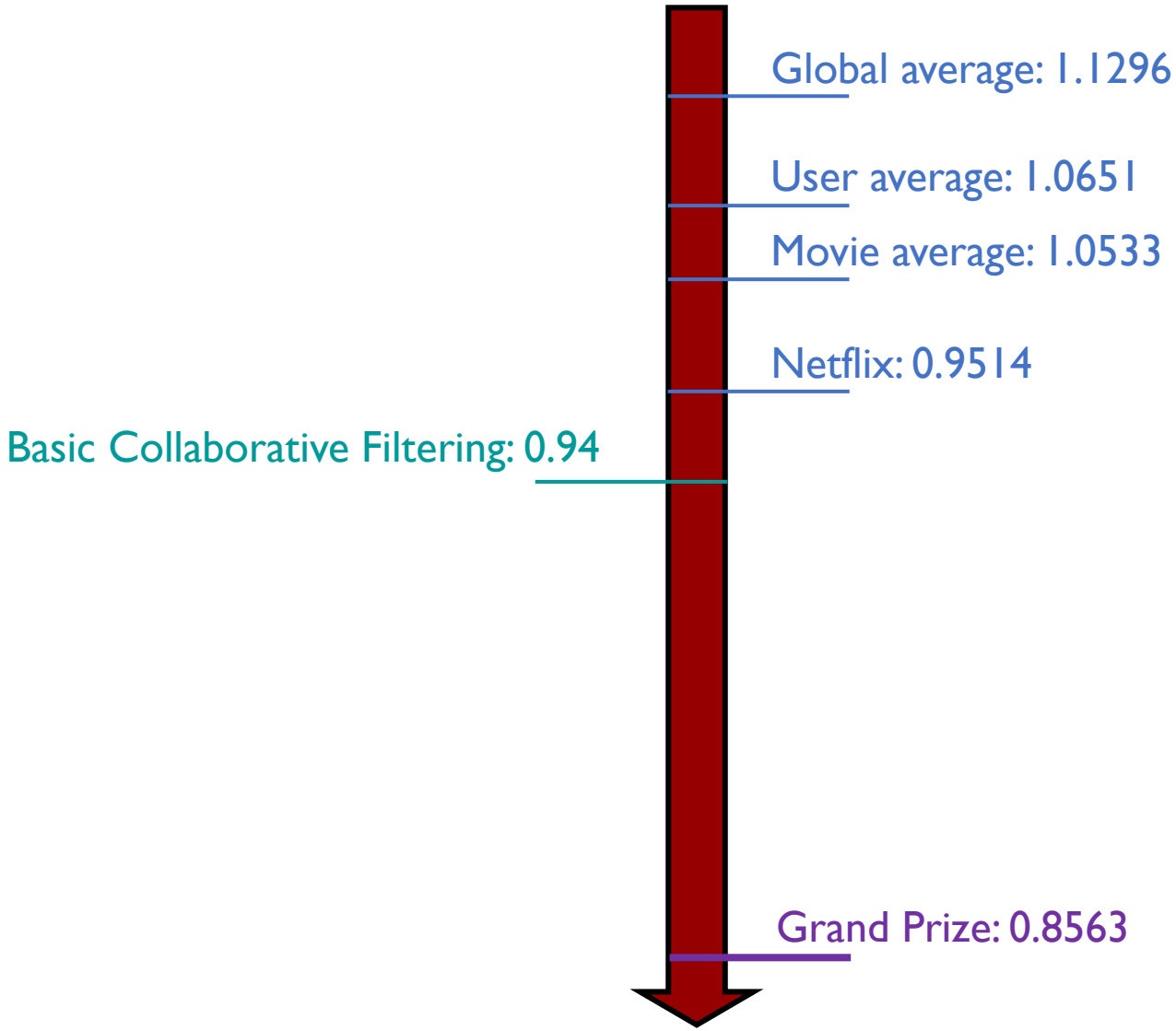
Training Set

Test Set

RMSE:

$$\sqrt{\frac{1}{M}\sum_{x,i}\left(U_{xi}-U_{xi}^{*}\right)^{2}}$$

Performance of Various Models



Recap: Modeling Deviations

- Basic Collaborative Filtering:

$$U_{xi} = \frac{\sum_{j \in \mathcal{N}} sim(i,j) \cdot U_{xj}}{\sum_{j \in \mathcal{N}} sim(i,j)}$$

- In practice,

$$U_{xi} = b_{xi} + \frac{\sum_{j \in \mathcal{N}} sim(i,j) \cdot (U_{xj} - b_{xj})}{\sum_{j \in \mathcal{N}} sim(i,j)}$$

  - b_{xi}: baseline estimate for U_{xi} $\quad (b_{xi} = \mu + b_x + b_i)$
  - μ: overall mean movie rating
  - b_x: rating deviation of user x, which is the (avg. rating given by user x) $- \mu$
  - b_i: rating deviation of item i, which is the (avg. rating given to item i) $- \mu$

One Step Further: Learning the Weight

$$U_{xi} = b_{xi} + \sum_{j \in \mathcal{N}} w_{ij} \, (U_{xj} - b_{xj})$$

- w_{ij} is learned from training data
  - We allow $\sum_{j \in \mathcal{N}} w_{ij} \neq 1$.
- w_{ij} models the interaction between pairs of movies.
  - It does not depend on user x.
- What is the objective?
  - RMSE! $\sqrt{\frac{1}{M} \sum_{x,i} \left(U_{xi} - U_{xi}^{*} \right)^2}$
  - Or equivalently: $\sum_{x,i} (U_{xi} - U_{xi}^{*})^2$

Recommendations via Optimization

$$J(w) = \sum_{x,i} (U_{xi} - U^*_{xi})^2 = \sum_{x,i} \left(\left[b_{xi} + \sum_{j \in \mathcal{N}} w_{ij} \left(U_{xj} - b_{xj} \right) \right] - U^*_{xi} \right)^2$$

- How to find the values of w_{ij}?
  - Gradient descent!

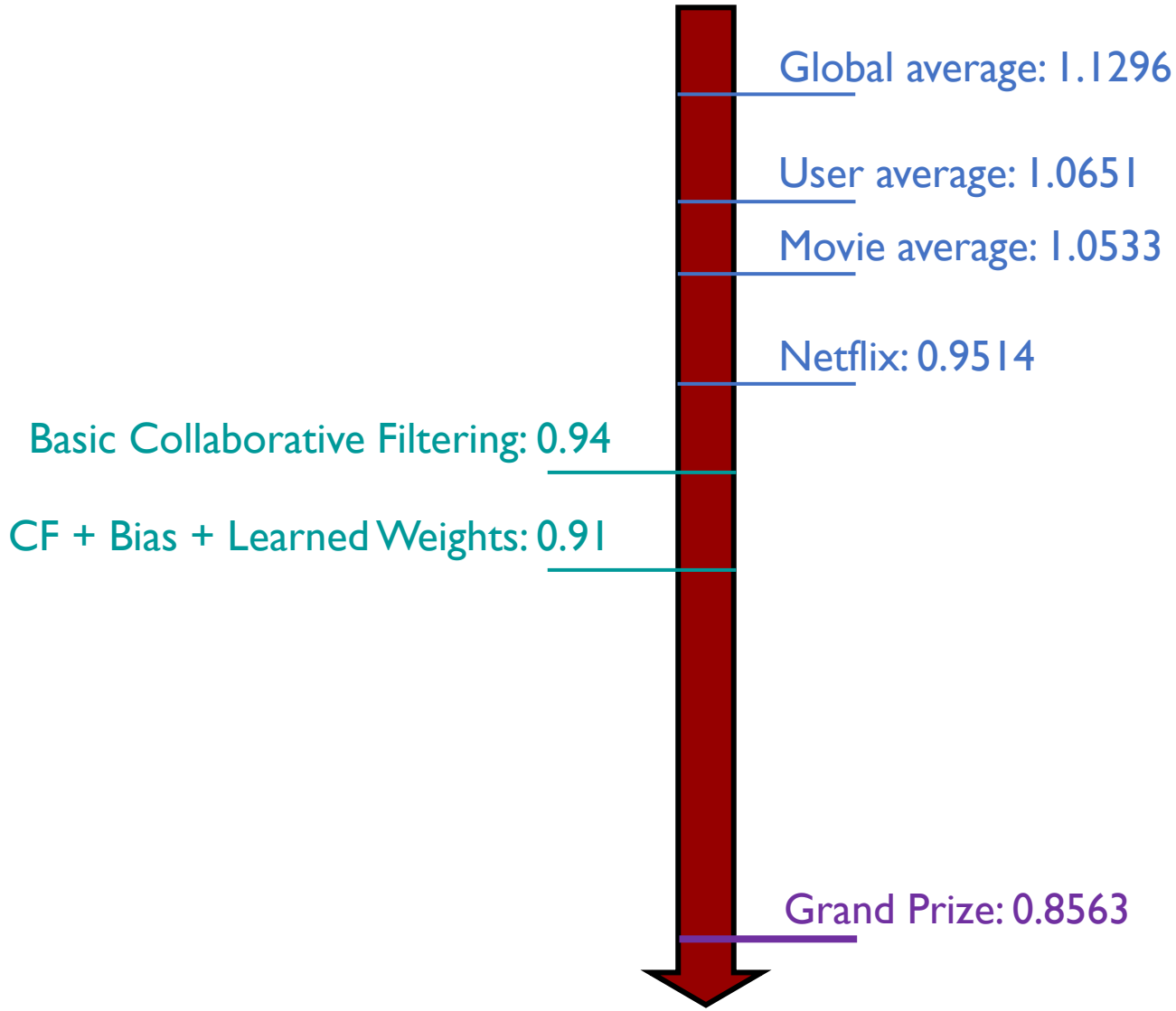
$$\frac{\partial J}{\partial w_{ij}} = 2 \sum_{x,i} \left(\left[b_{xi} + \sum_{j \in \mathcal{N}} w_{ij} \left(U_{xj} - b_{xj} \right) \right] - U^*_{xi} \right) \left(U_{xj} - b_{xj} \right)$$

(for $j \in \mathcal{N}$)

$$\frac{\partial J}{\partial w_{ij}} = 0$$

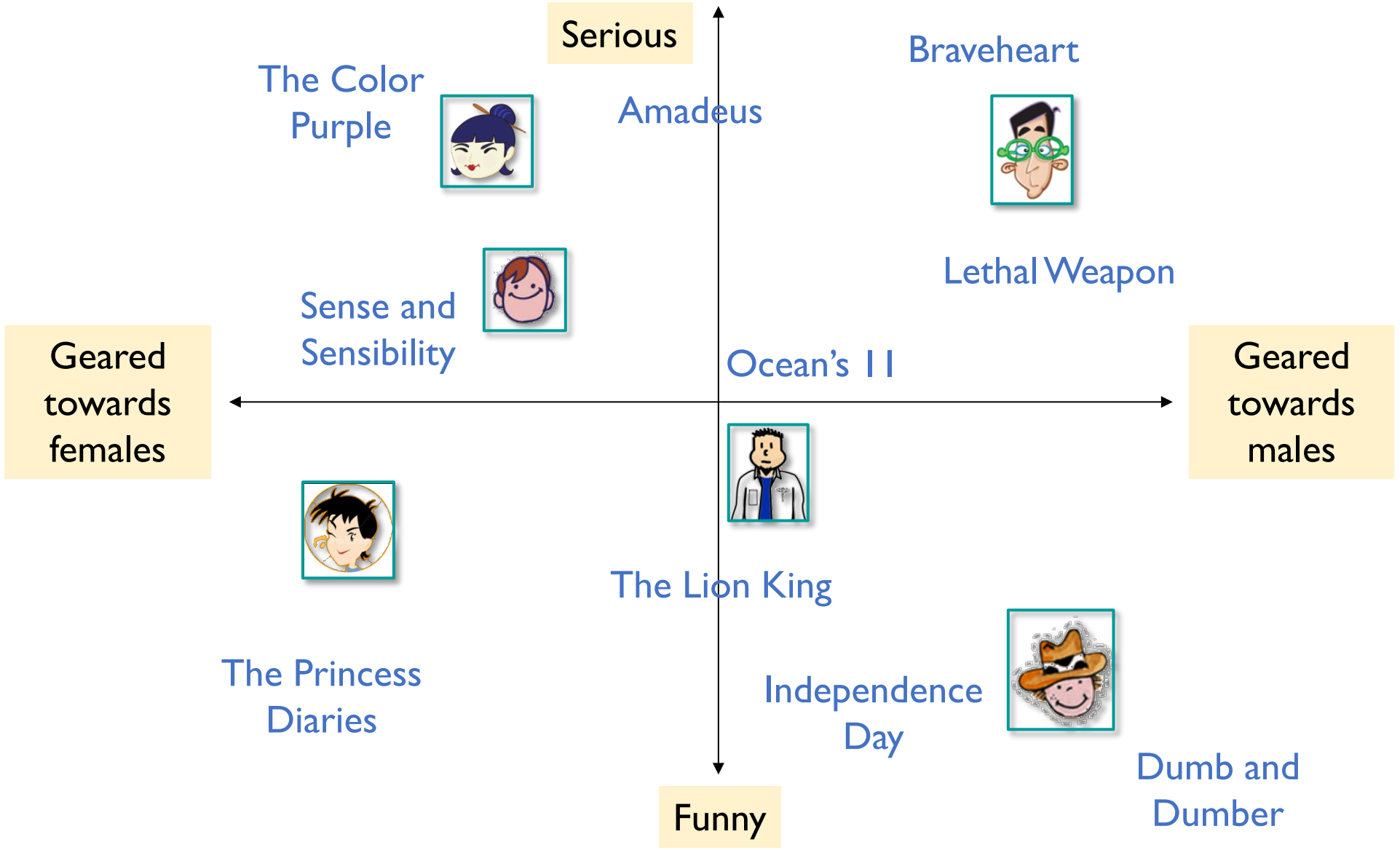
(for $j \notin \mathcal{N}$)

Performance of Various Models



Matrix Factorization
(Latent-Factor Models)

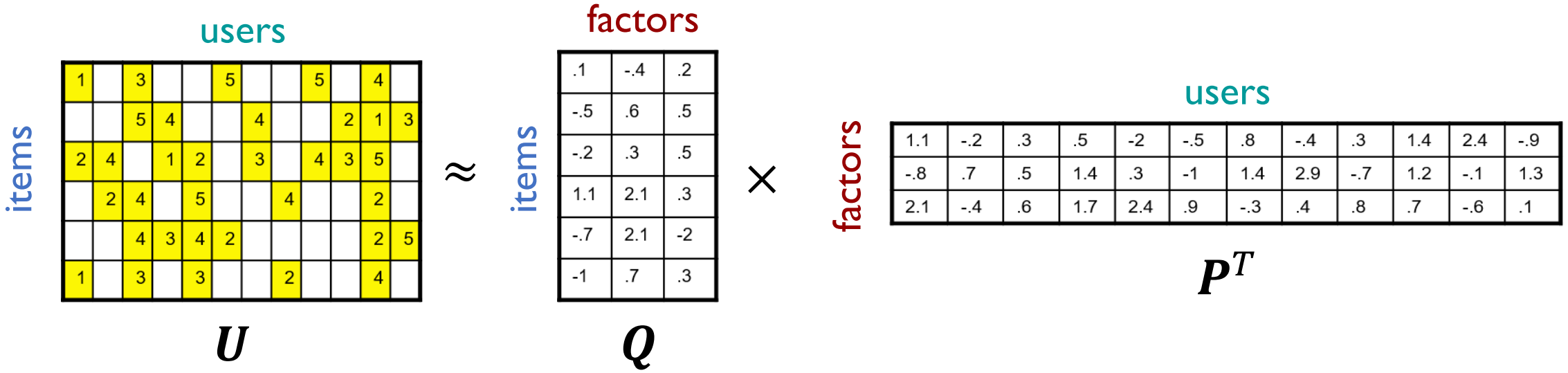
There are certain latent factors that influence users' ratings.



Latent-Factor Models

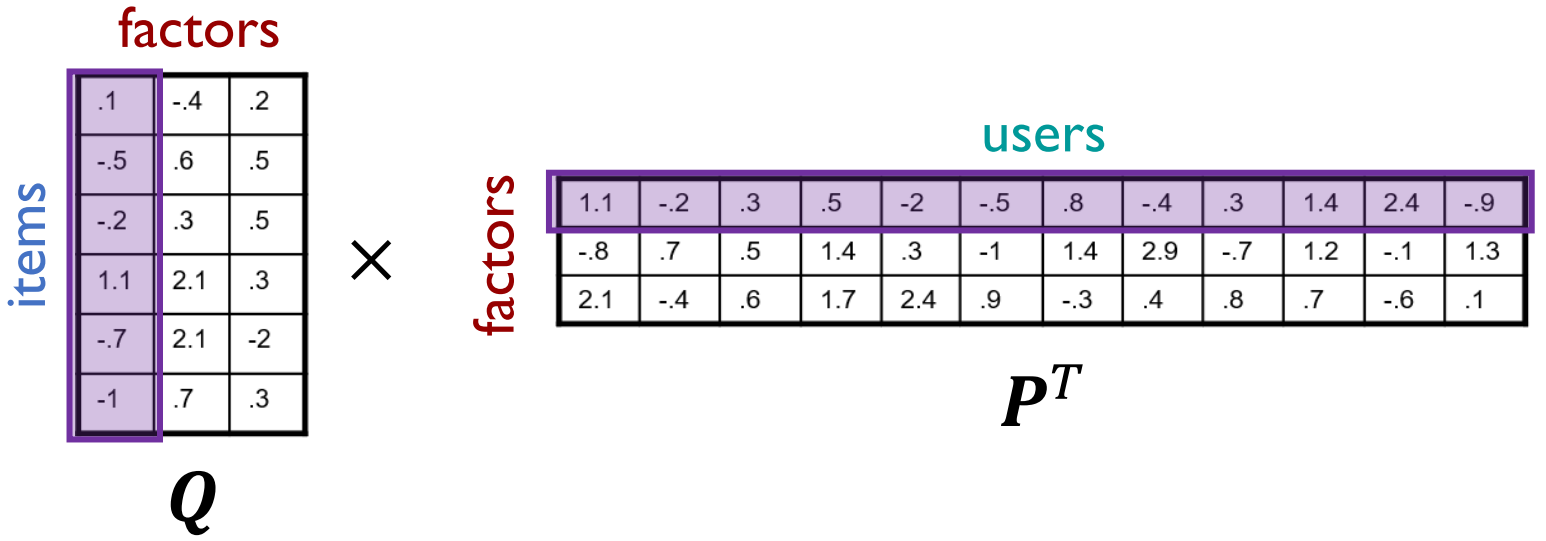
- $\boldsymbol{U} \approx \boldsymbol{Q}\boldsymbol{P}^T$

The number of factors is small.
In other words, $\boldsymbol{Q}$ and $\boldsymbol{P}$ are "thin".



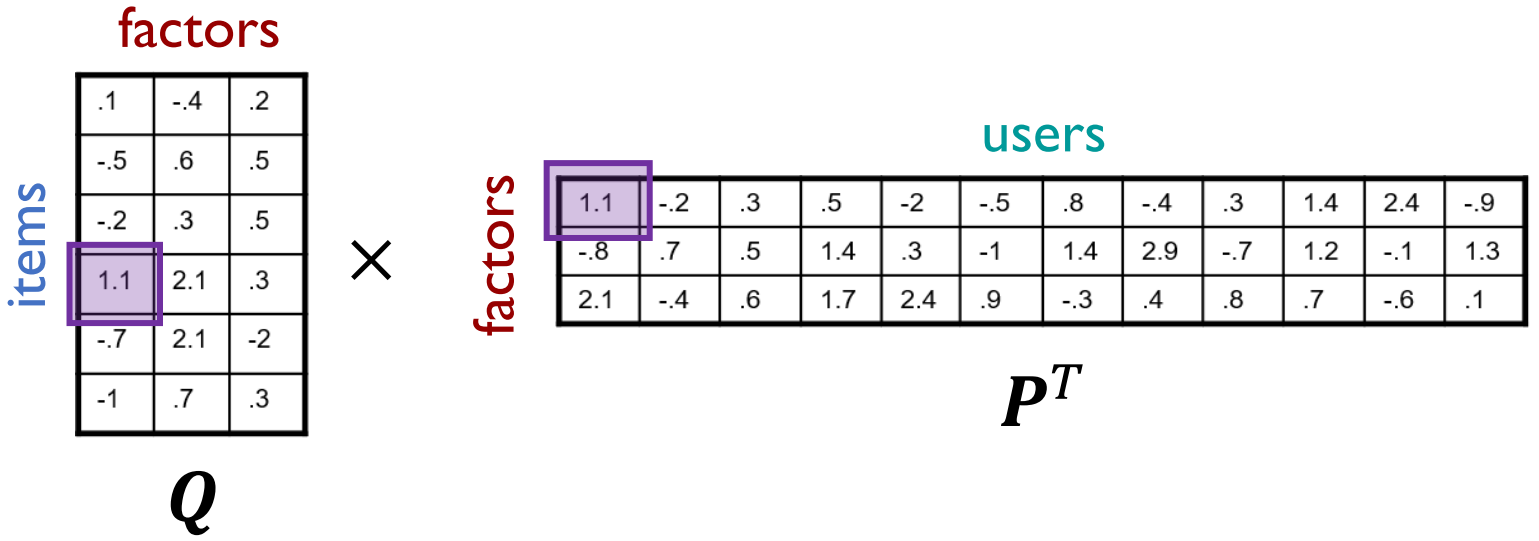
- For now, let's assume this is mathematically doable.
  - $\boldsymbol{U}$ has missing entries but let's first ignore that!
  - Basically, we will want the reconstruction error to be small on known ratings and we don't care about the values on the missing ones.

How to interpret Q and P?



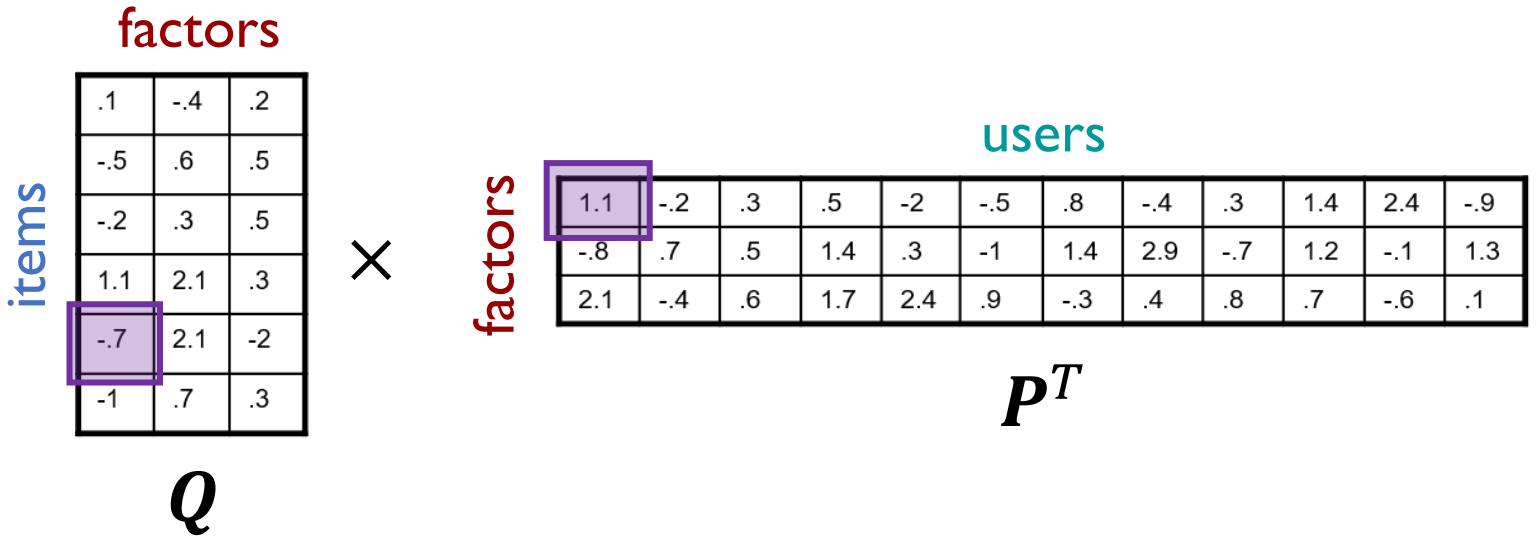
- Let's assume that the first factor is *the level of seriousness*.

How to interpret Q and P?



- Let's assume that the first factor is *the level of seriousness*.
  - The seriousness of User 1 is 1.1
  - The seriousness of Movie 4 is 1.1
  - So, User 1 may like Movie 4

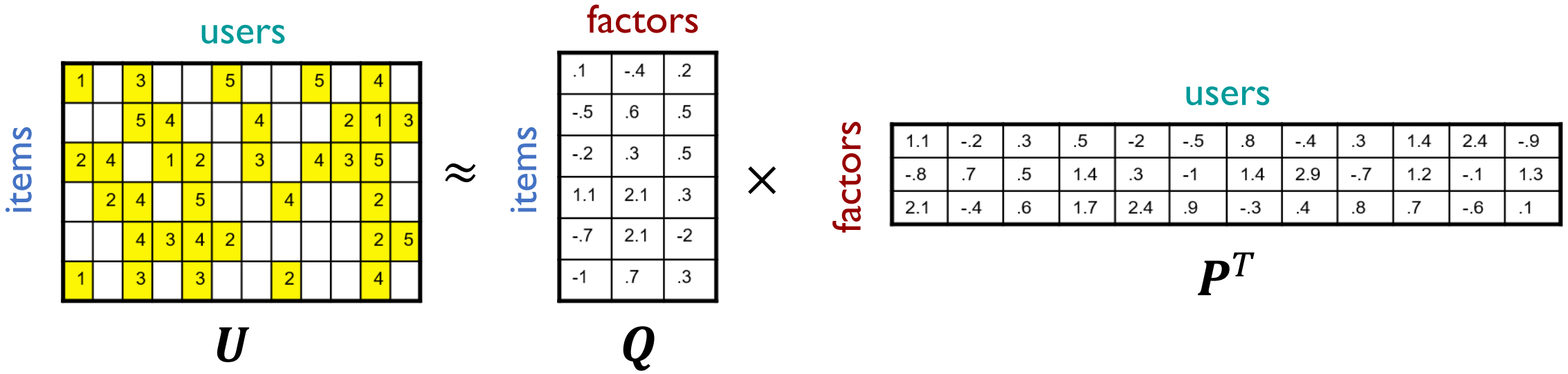
How to interpret Q and P?



- Let's assume that the first factor is *the level of seriousness*.
  - The seriousness of User 1 is 1.1
  - The seriousness of Movie 5 is -0.7
  - So, User 1 may NOT like Movie 5

Of course, we need to consider all the factors.

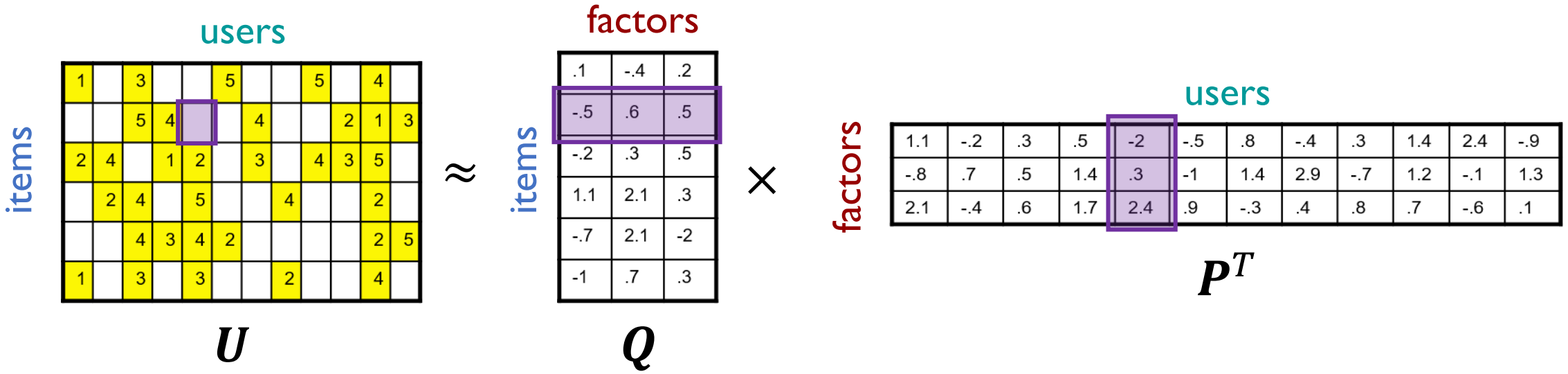
- Ratings as "sum of products"



$$U_{xi} = \sum_{\phi\text{: all factors}} Q_{i\phi} \cdot P_{x\phi}$$

Estimating the Missing Rating

- Ratings as "sum of products"



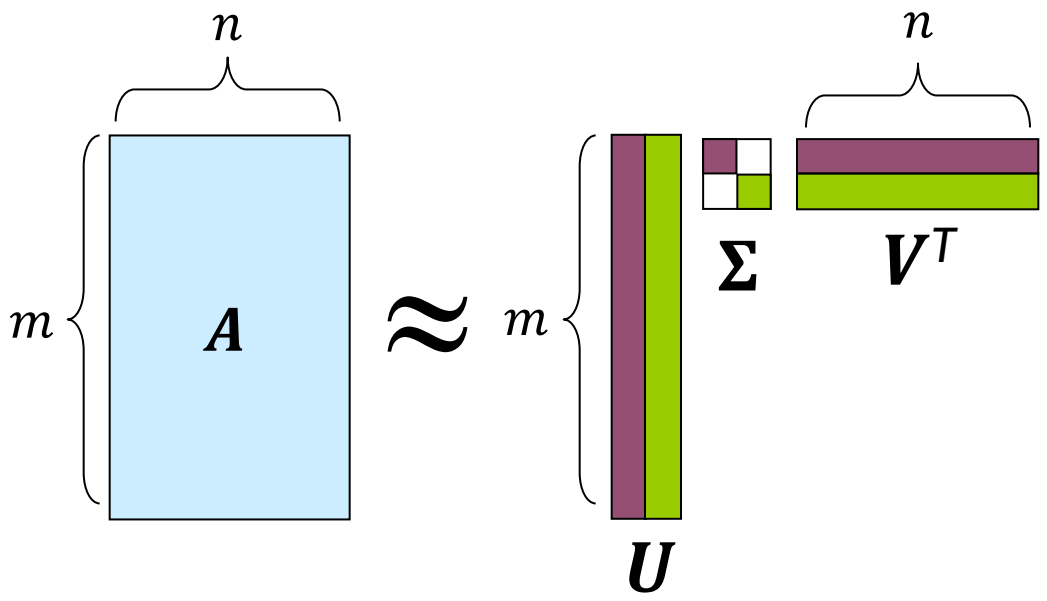
$$U_{xi} = \sum_{\phi\text{: all factors}} Q_{i\phi} \cdot P_{x\phi} = (-0.5) \times (-2) + 0.6 \times 0.3 + 0.5 \times 2.4 = 2.38$$

How to find Q and P?

Singular Value Decomposition (SVD)

$$\boldsymbol{A} \approx \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$$

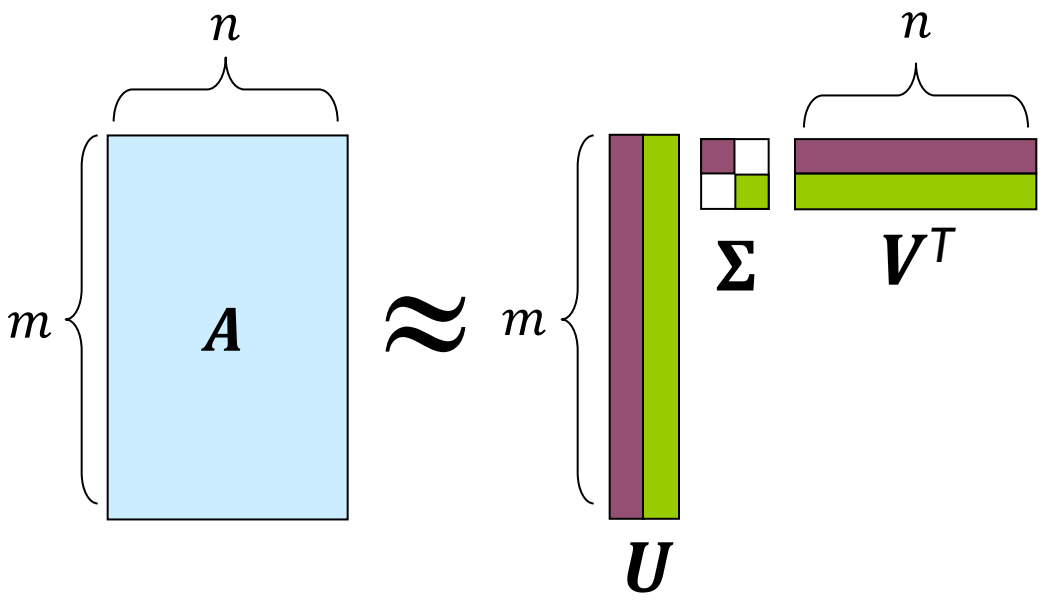
- Input matrix: $\boldsymbol{A}$
- Step 1: Compute $\boldsymbol{A}^T\boldsymbol{A}$
- Step 2: Find the eigenvalues of eigenvectors of $\boldsymbol{A}^T\boldsymbol{A}$
  - Eigenvalues $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n \geq 0$
  - Eigenvectors $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_n$
- Step 3: Consider the largest k eigenvalues and their corresponding eigenvectors only. (The choice of k depends on how closely you wish to approximate)
  - $\boldsymbol{V} = [\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_k]$



Singular Value Decomposition (SVD)

$$\boldsymbol{A} \approx \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$$

- Step 2: Find the eigenvalues of eigenvectors of $\boldsymbol{A}^T\boldsymbol{A}$
  - Eigenvalues $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n \geq 0$
  - Eigenvectors $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_n$
- Step 3: Consider the largest k eigenvalues and their corresponding eigenvectors only.
  - $\boldsymbol{V} = [\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_k]$
  - $\boldsymbol{\Sigma} = \text{diag}\{\sqrt{\lambda_1}, \sqrt{\lambda_2}, \ldots, \sqrt{\lambda_k}\}$
- Step 4: $\boldsymbol{U} = \boldsymbol{A}\boldsymbol{V}\boldsymbol{\Sigma}^{-1}$
  - Or you can do Steps 1-3 again for $\boldsymbol{A}\boldsymbol{A}^T$ (rather than $\boldsymbol{A}^T\boldsymbol{A}$) to get $\boldsymbol{U}$



SVD is good, but …

- SVD gives the minimum reconstruction error if we know all entries in $\boldsymbol{A}$.

$$\min_{\boldsymbol{U},\boldsymbol{\Sigma},\boldsymbol{V}} \sum_{i,j} \left(A_{ij} - [\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T]_{ij}\right)^2$$

  - Exactly our objective!
  - Using SVD for our matrix factorization task?

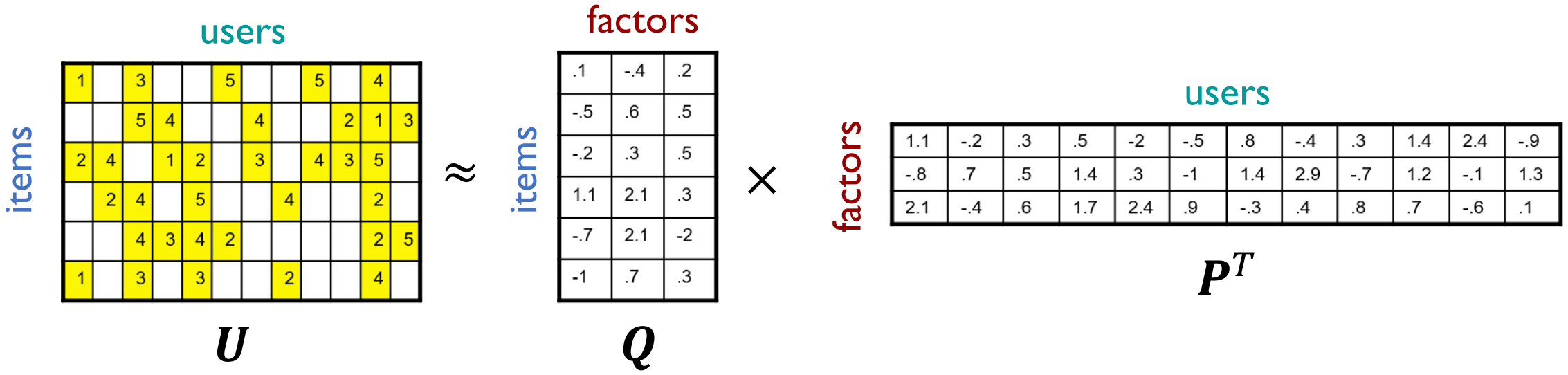
Latent Factor Model	User-Item Matrix: $\boldsymbol{U}$	User-Factor Matrix: $\boldsymbol{Q}$	Factor-Item Matrix: $\boldsymbol{P}^T$
SVD	Input Matrix: $\boldsymbol{A}$	$\boldsymbol{U}$	$\boldsymbol{\Sigma}\boldsymbol{V}^T$

- BUT, our user-item matrix $\boldsymbol{U}$ has missing values!
  - How to interpret missing values? (as 0? a bad idea)
  - Does the property of minimum reconstruction error still hold if there are missing values? (we don't know)

Factorizing a Matrix with Missing Values

$$\min_{\boldsymbol{Q},\boldsymbol{P}} \sum_{(x,i)\text{ known}} (U_{xi} - [\boldsymbol{Q}\boldsymbol{P}^T]_{xi})^2 = \sum_{(x,i)\text{ known}} (U_{xi} - \boldsymbol{q}_i \boldsymbol{p}_x^T)^2$$

- $\boldsymbol{q}_i$ (item vector): the row corresponding to item i in $\boldsymbol{Q}$
- $\boldsymbol{p}_x^T$ (user vector): the column corresponding to user x in $\boldsymbol{P}^T$



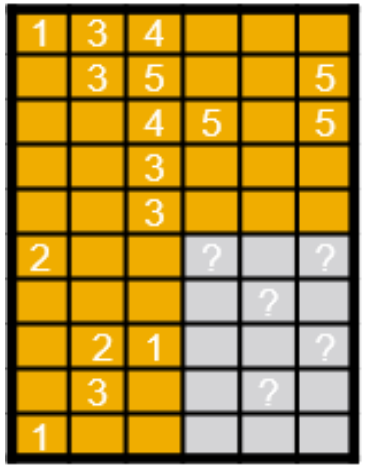
Overfitting

$$\min_{\boldsymbol{Q},\boldsymbol{P}} \sum_{(x,i)\ \text{known}} (U_{xi} - \boldsymbol{q}_i \boldsymbol{p}_x^T)^2$$

- $\boldsymbol{q}_i$ (item vector): the row corresponding to item i in $\boldsymbol{Q}$
- $\boldsymbol{p}_x^T$ (user vector): the column corresponding to user x in $\boldsymbol{P}^T$

- No closed form solution.
- All item vectors and user vectors are parameters to be learned!

- Overfitting: With too much freedom (too many free parameters) the model starts fitting noise in the training data, thus not generalizing well to unseen test data.



Regularization

- Model parameters can be "complicated" where there are sufficient training data
- Model parameters should be "simple" where training data are scarce

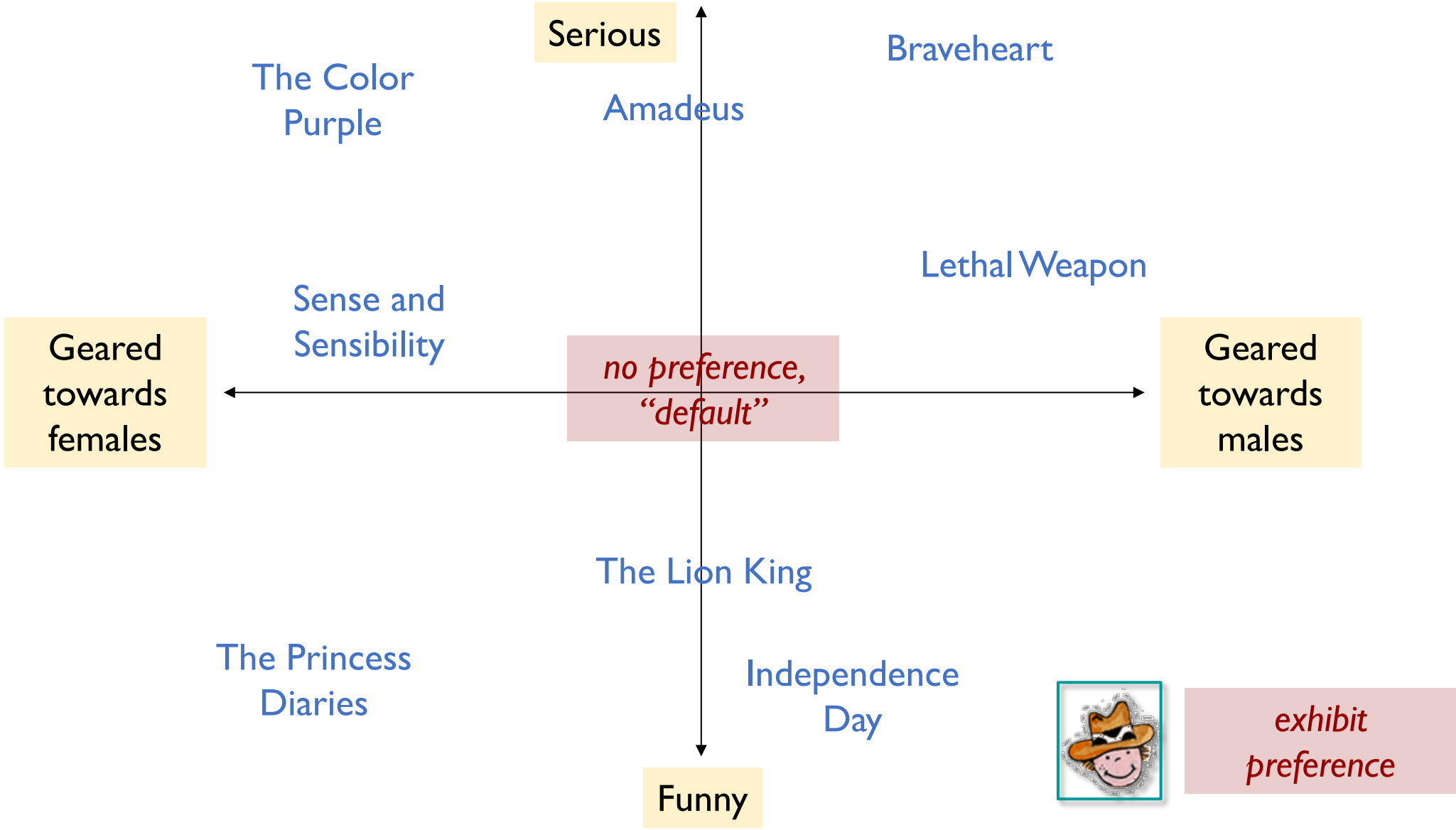
$$\min_{\boldsymbol{Q},\boldsymbol{P}} \sum_{(x,i)\ \text{known}} (U_{xi} - \boldsymbol{q}_i \boldsymbol{p}_x^T)^2 + \left[c_1 \sum_x \|\boldsymbol{p}_x\|^2 + c_2 \sum_i \|\boldsymbol{q}_i\|^2 \right]$$

Original Objective | Regularization Term

(c_1 and c_2 are hyperparameters)

- How to understand the Regularization Term?

The Effect of Regularization



The Effect of Regularization

- If the user has rated hundreds of movies, it stays about where the data places it.

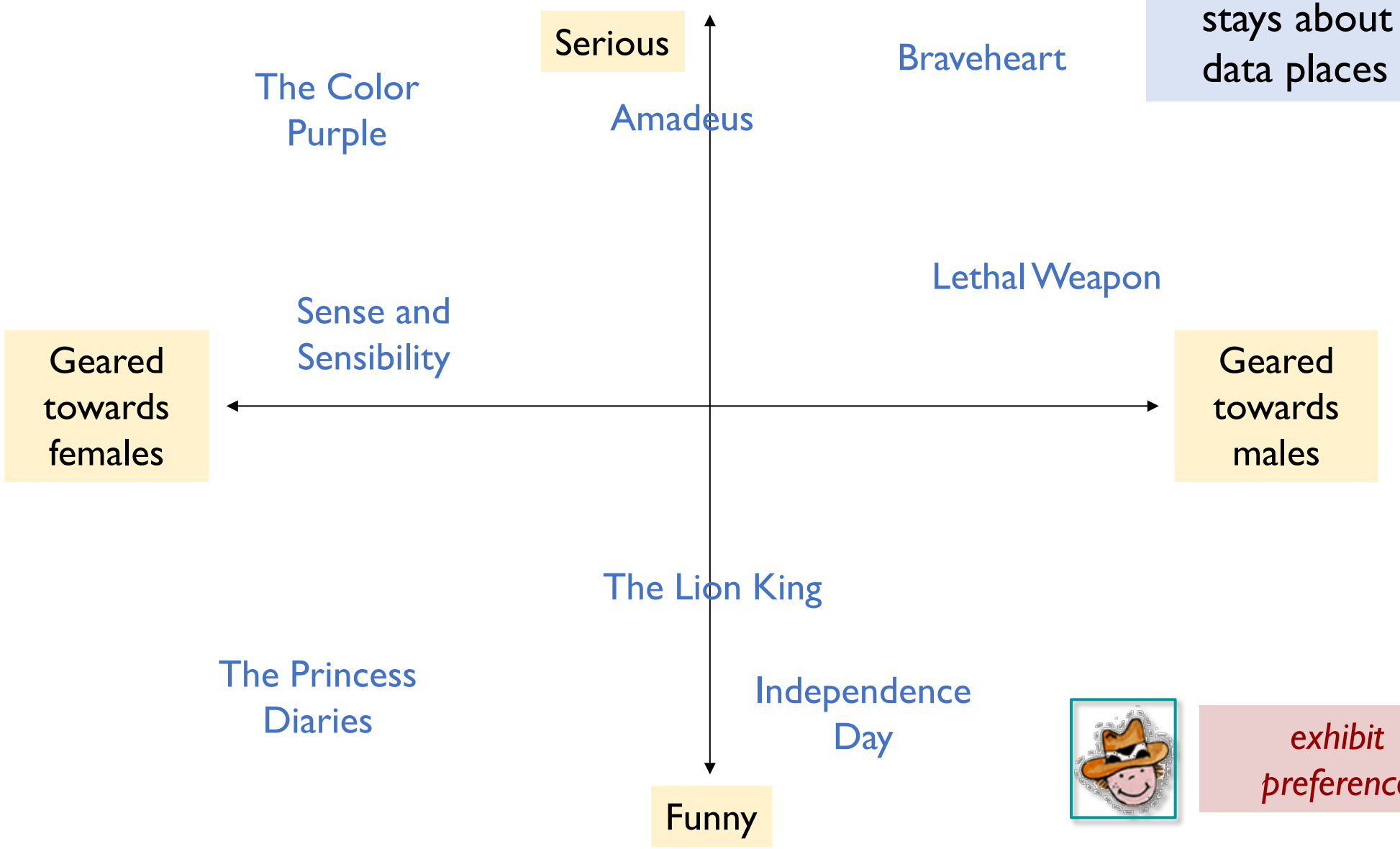
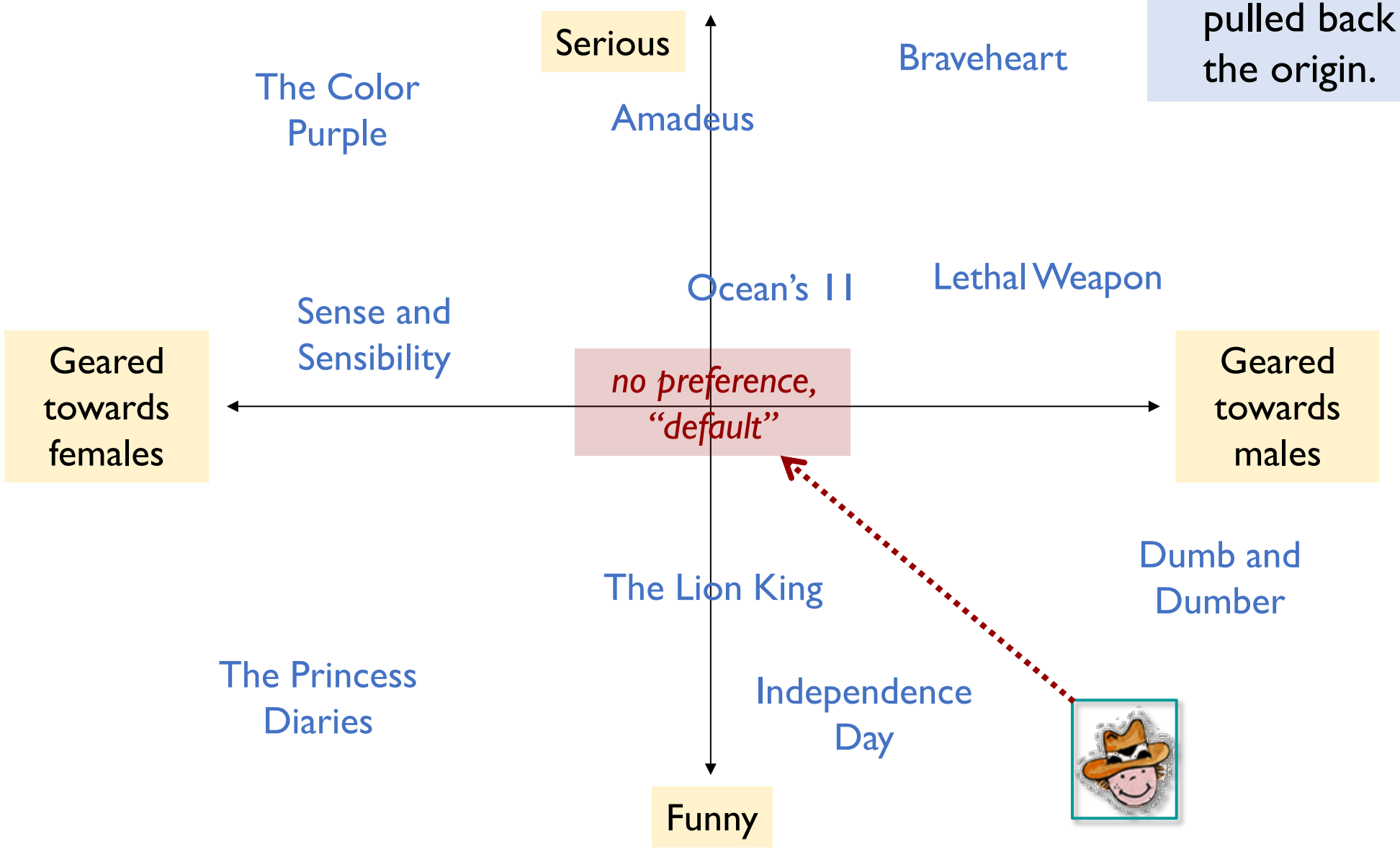


exhibit preference

The Effect of Regularization

- If the user has rated only a handful, it is pulled back towards the origin.

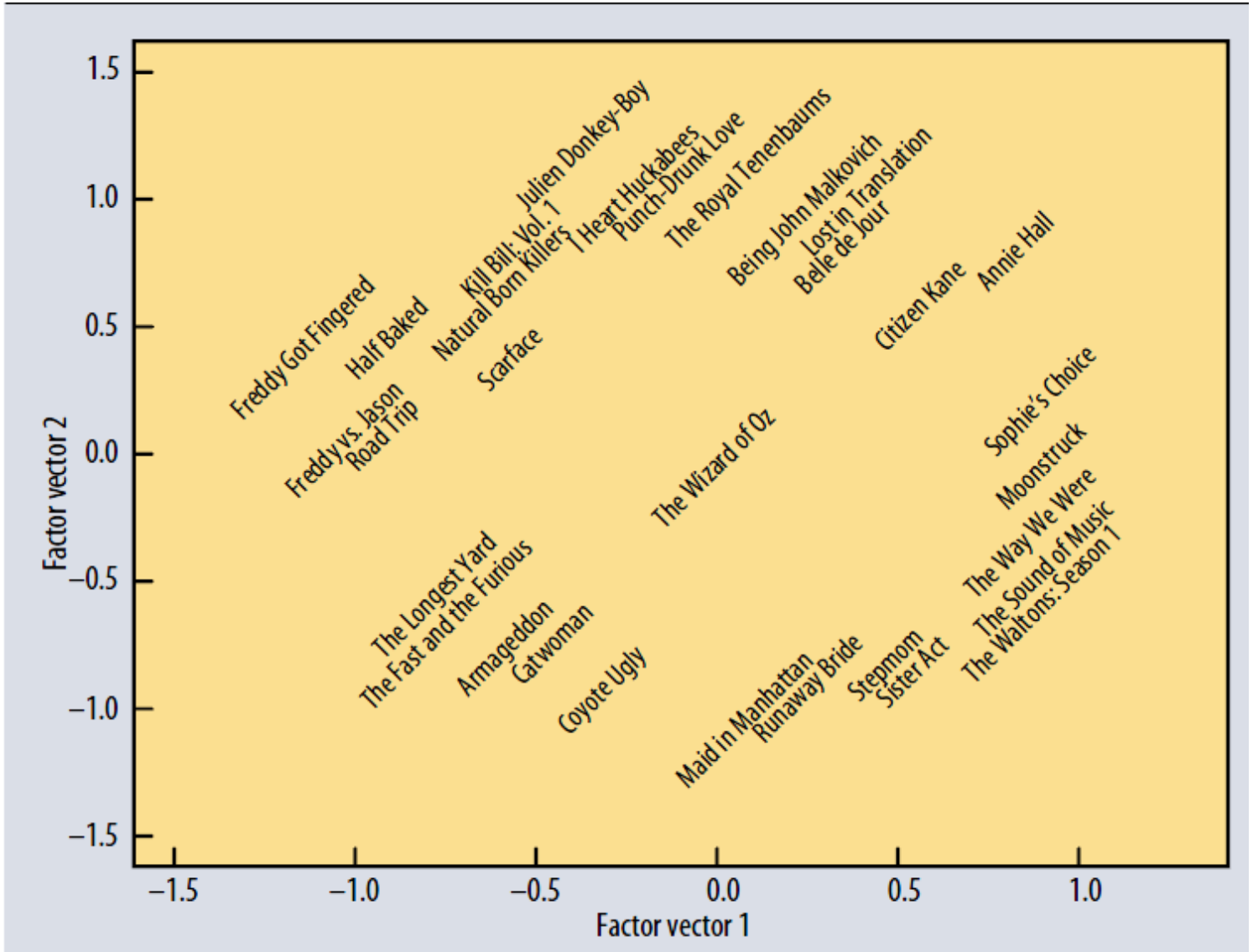


Gradient Descent

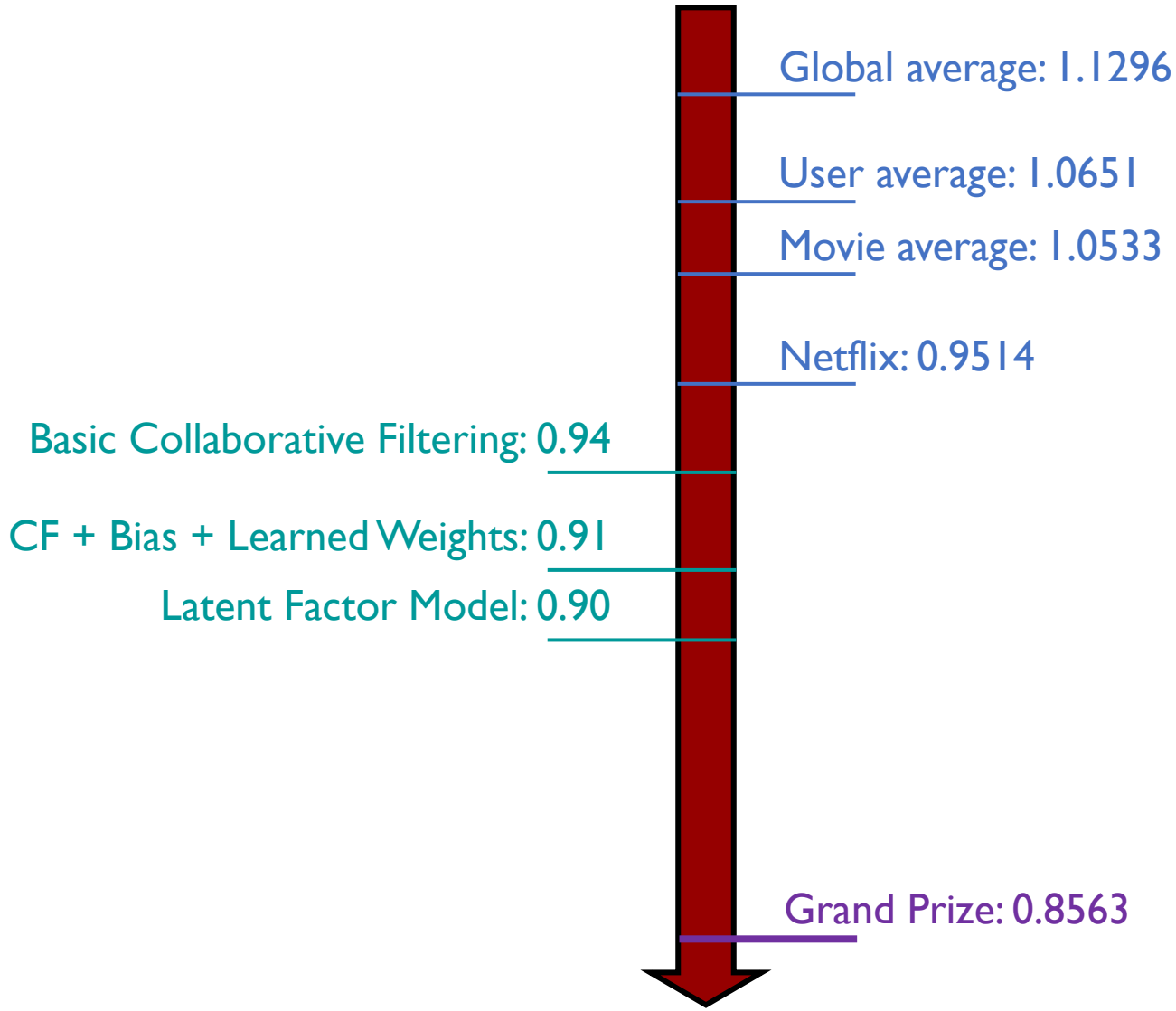
$$\min_{\boldsymbol{Q},\boldsymbol{P}} J = \sum_{(x,i)\text{ known}} (U_{xi} - \boldsymbol{q}_i \boldsymbol{p}_x^T)^2 + \left[c_1 \sum_x \|\boldsymbol{p}_x\|^2 + c_2 \sum_i \|\boldsymbol{q}_i\|^2 \right]$$

- Step 1: Initialize $\boldsymbol{Q}$ and $\boldsymbol{P}$ using SVD (pretend missing ratings are 0)
- Step 2: Gradient descent
  - $P_{x\phi} = P_{x\phi} - \eta \frac{\partial J}{\partial P_{x\phi}}$
    - $\frac{\partial J}{\partial P_{x\phi}} = \sum_{(x,i)\text{ known}} \left(-2(U_{xi} - \boldsymbol{q}_i \boldsymbol{p}_x^T) Q_{i\phi} + 2c_1 P_{x\phi} \right)$
  - $Q_{i\phi} = Q_{i\phi} - \eta \frac{\partial J}{\partial Q_{i\phi}}$
    - $\frac{\partial J}{\partial Q_{i\phi}} = \sum_{(x,i)\text{ known}} \left(-2(U_{xi} - \boldsymbol{q}_i \boldsymbol{p}_x^T) P_{x\phi} + 2c_2 Q_{i\phi} \right)$

Learned Item Vectors in the Latent Factor Space



Performance of Various Models



Extending Latent Factor Models to Include Bias

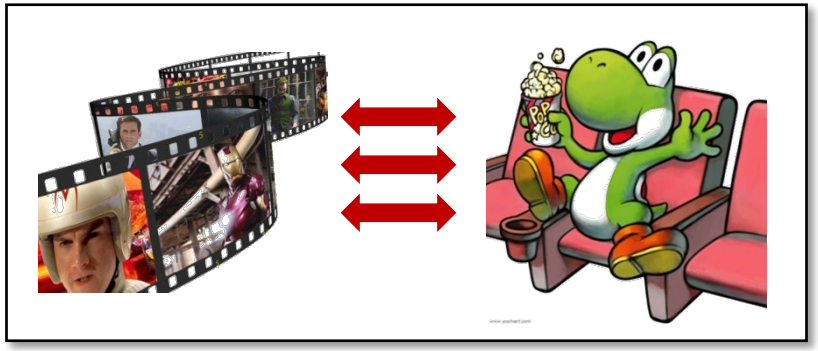
Bias, Again

- Basic Latent Factor Model: $U_{xi} = \boldsymbol{q}_i \boldsymbol{p}_x^T$
- Latent Factor Model with Bias: $U_{xi} = \mu + b_x + b_i + \boldsymbol{q}_i \boldsymbol{p}_x^T$
  - μ: overall mean movie rating
  - b_x: rating deviation of user x
  - b_i: rating deviation of item i

user bias

movie bias

user-movie interaction



Bias, Again

- Latent Factor Model with Bias:

$$U_{xi} = \mu + b_x + b_i + \boldsymbol{q}_i \boldsymbol{p}_x^T$$

  - μ: overall mean movie rating
    - E.g., $\mu = 2.7$
  - b_x: rating deviation of user x (*to be learned*)
    - E.g., Bob is a critical reviewer. Based on the training data, his rating will be 0.7 star lower than the mean $\Rightarrow b_x = -0.7$.
  - b_i: rating deviation of item i (*to be learned*)
    - E.g., Star Wars will get a mean rating of 0.5 higher than the average $\Rightarrow b_i = 0.5$
  - $\boldsymbol{q}_i$ and $\boldsymbol{p}_x$: vector of user x and item i in the latent factor space (*to be learned*)
    - E.g., based on the genre, Bob likes Star Wars $\Rightarrow \boldsymbol{q}_i \boldsymbol{p}_x^T = 0.3$
  - $U_{xi} = 2.7 - 0.7 + 0.5 + 0.3 = 2.8$

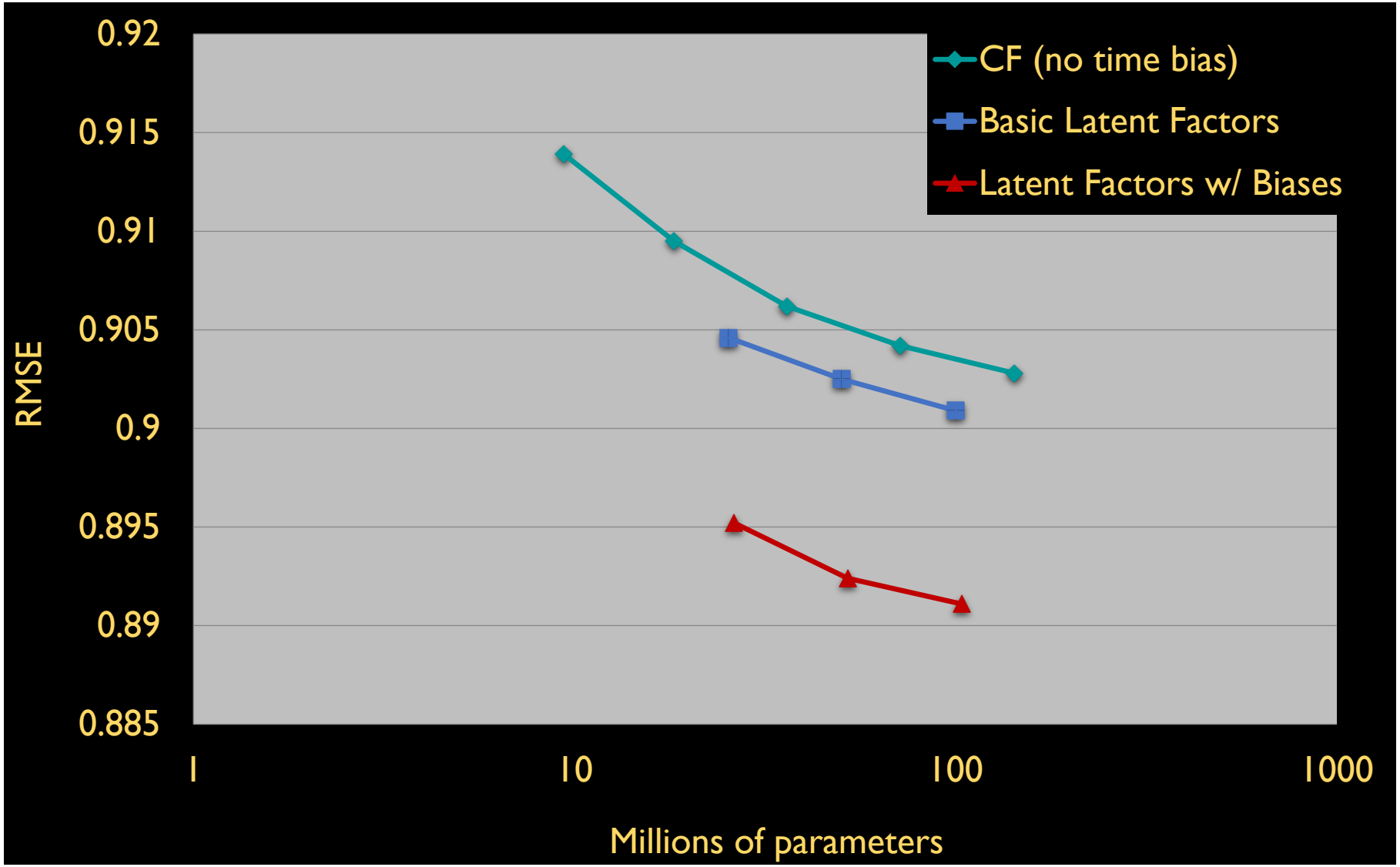
Fitting the New Model

$$\min_{\boldsymbol{Q},\boldsymbol{P},b_x,b_i} J = \sum_{(x,i)\text{ known}} (U_{xi} - (\mu + b_x + b_i + \boldsymbol{q}_i \boldsymbol{p}_x^T))^2$$
$$+ \left[c_1 \sum_x \|\boldsymbol{p}_x\|^2 + c_2 \sum_i \|\boldsymbol{q}_i\|^2 + c_3 \sum_x \|b_x\|^2 + c_4 \sum_i \|b_i\|^2 \right]$$

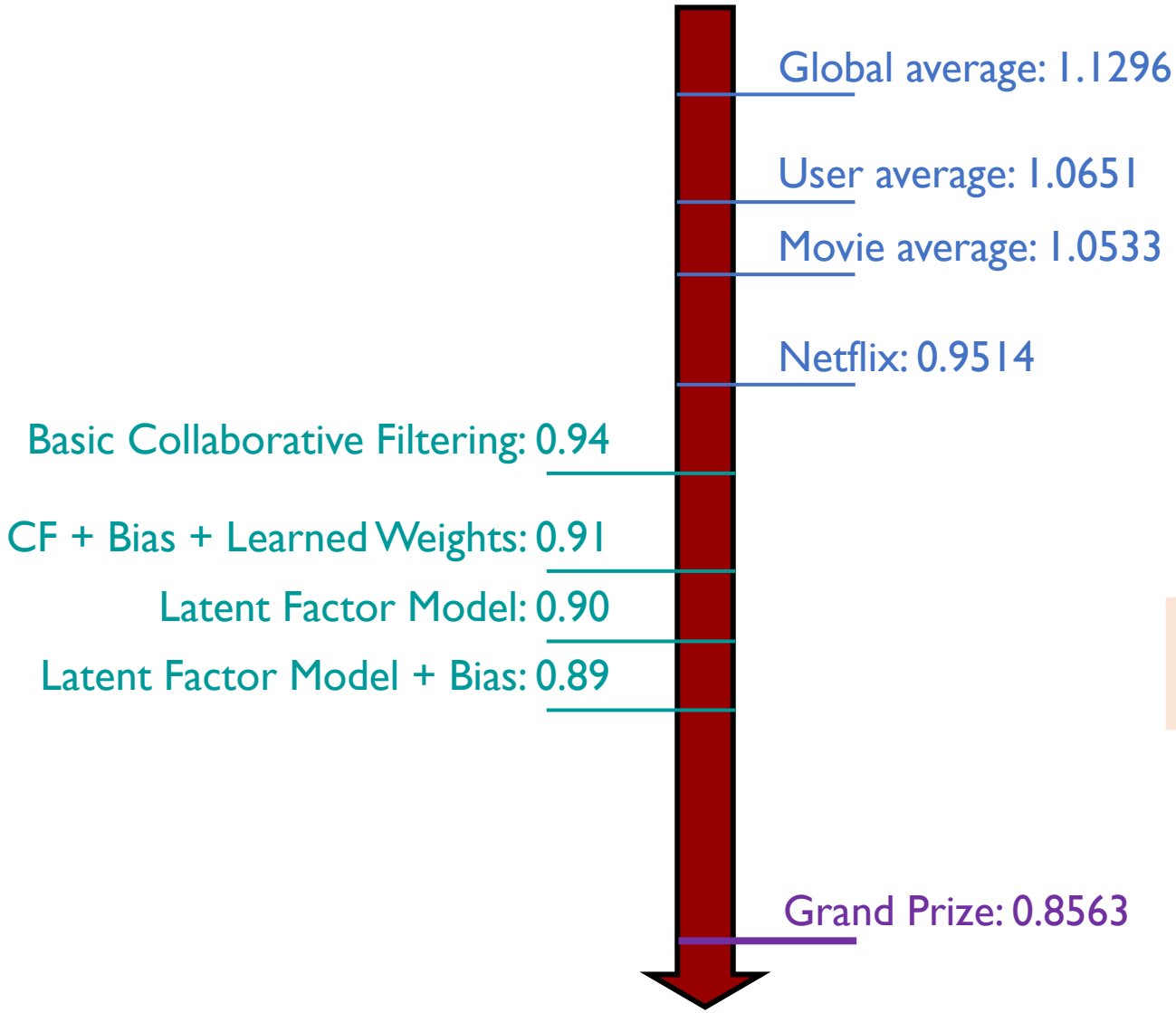
- Both biases b_x, b_i as well as interactions $\boldsymbol{q}_i, \boldsymbol{p}_x$ are treated as parameters to be learned via gradient descent
  - $P_{x\phi} = P_{x\phi} - \eta \frac{\partial J}{\partial P_{x\phi}}, \quad Q_{i\phi} = Q_{i\phi} - \eta \frac{\partial J}{\partial Q_{i\phi}}$
  - $b_x = b_x - \eta \frac{\partial J}{\partial b_x}, \quad b_i = b_i - \eta \frac{\partial J}{\partial b_i}$

Performance of Various Models

- Which hyperparameter determines the number of parameters?
  - Number of factors



Performance of Various Models



What were the next steps?
Next week!

Thank You!

Course Website: https://yuzhang-teaching.github.io/CSCE670-S26.html