CS5228 LECTURE 8: RECOMMENDATION SYSTEMS

Bryan Hooi

School of Computing

National University of Singapore

Slide Credit: Stanford CS246 – Mining Massive Datasets, Jure Leskovec (some slides originally from: Yehuda Koren, Robert Bell, Padhraic Smyth)
https://xkcd.com/1098/

REVIEW: LOGISTIC REGRESSION

Features

Number of '$'	Number of '!'
5	2

$x = \begin{pmatrix} 5 \\ 2 \end{pmatrix}$

(Predicted probability of the email being spam)

? $\hat{y}$

Parameters:

Weights $w = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$

1	2

Bias $b = -5$

-5

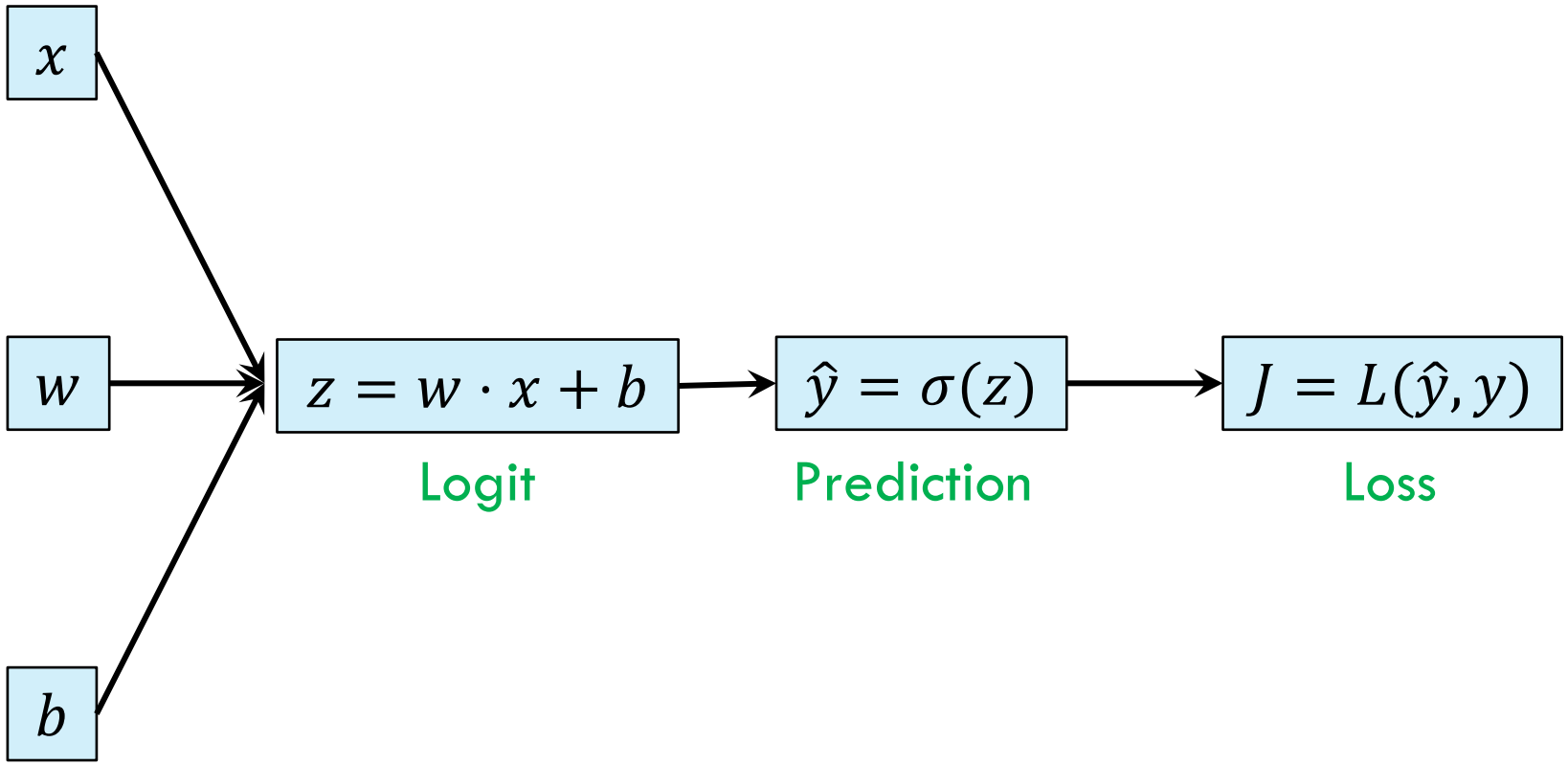
Prediction:

Sigmoid function

Dot product

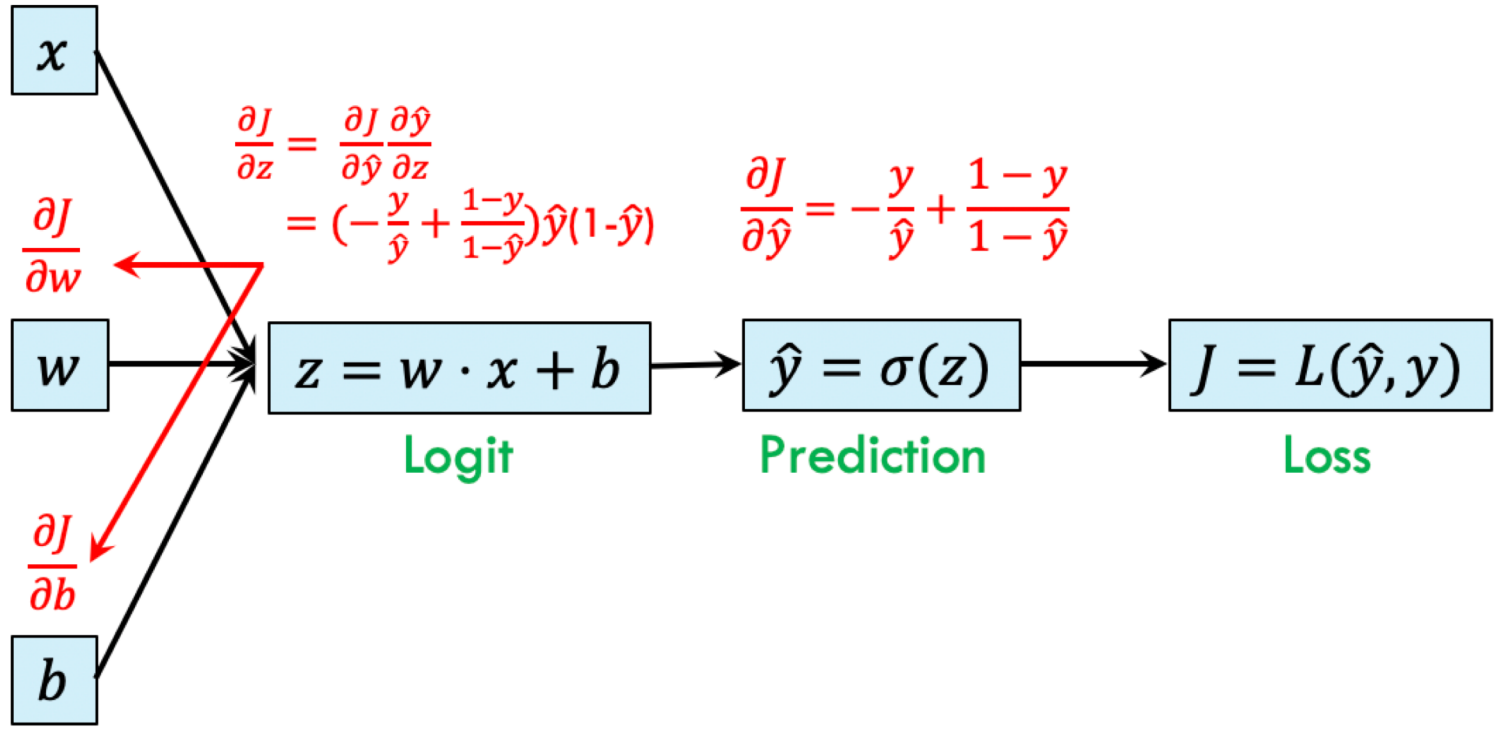
$$\hat{y} = \sigma(x \cdot w + b) = \sigma\left(\begin{pmatrix} 5 \\ 2 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 2 \end{pmatrix} - 5\right) = \sigma(9 - 5) = \frac{1}{1 + e^{-4}} = 0.982$$

REVIEW: LOGISTIC REGRESSION AS A COMPUTATIONAL GRAPH



REVIEW: BACKWARD PASS

Gradient of loss w.r.t. parameters: used for gradient descent



REVIEW: GRADIENT DESCENT

We want to find w, b to minimize $J(w, b)$

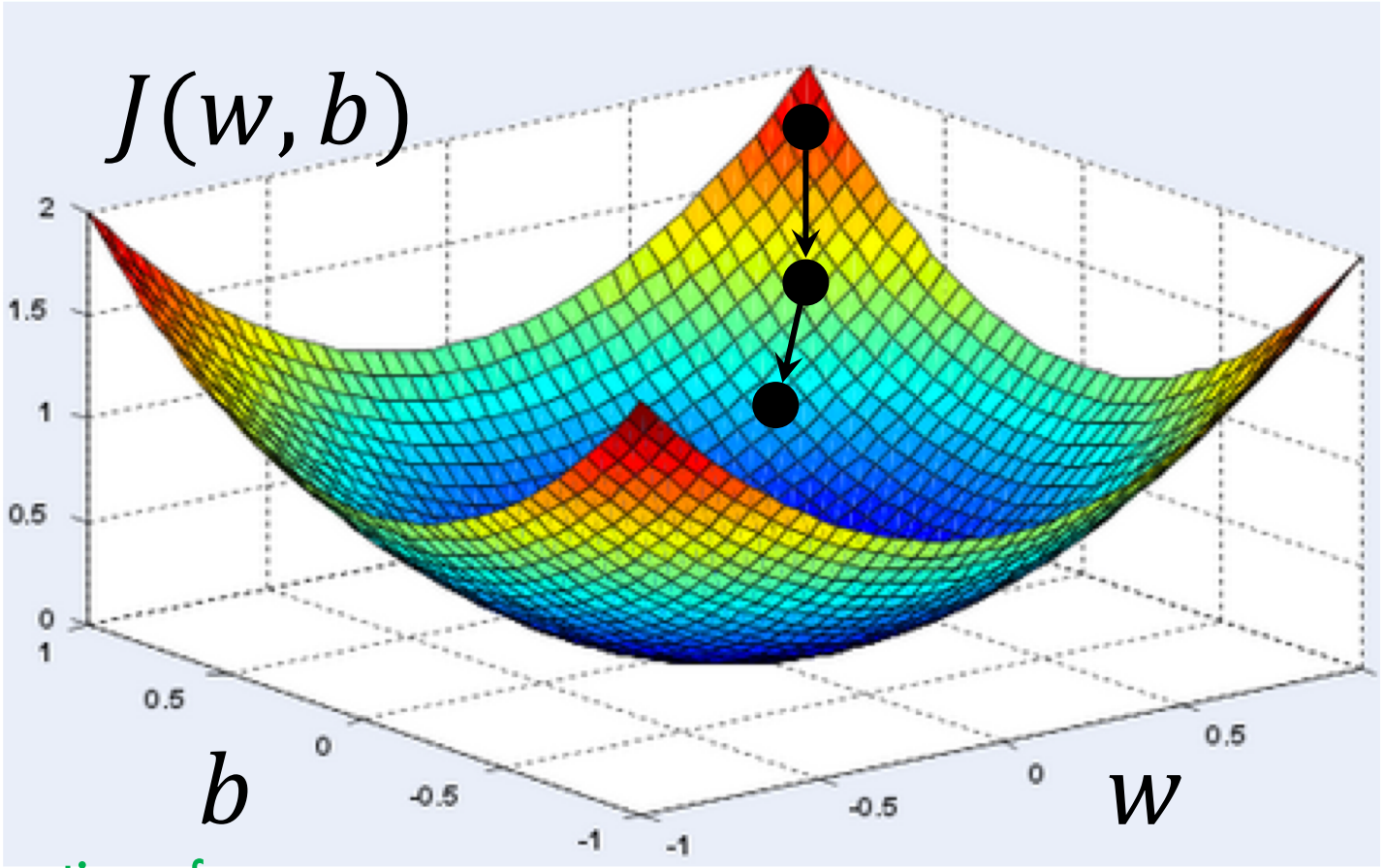
Start at an arbitrary point, then move in the negative gradient direction

$$w \leftarrow w - \alpha \frac{\delta J(w,b)}{\delta w}$$

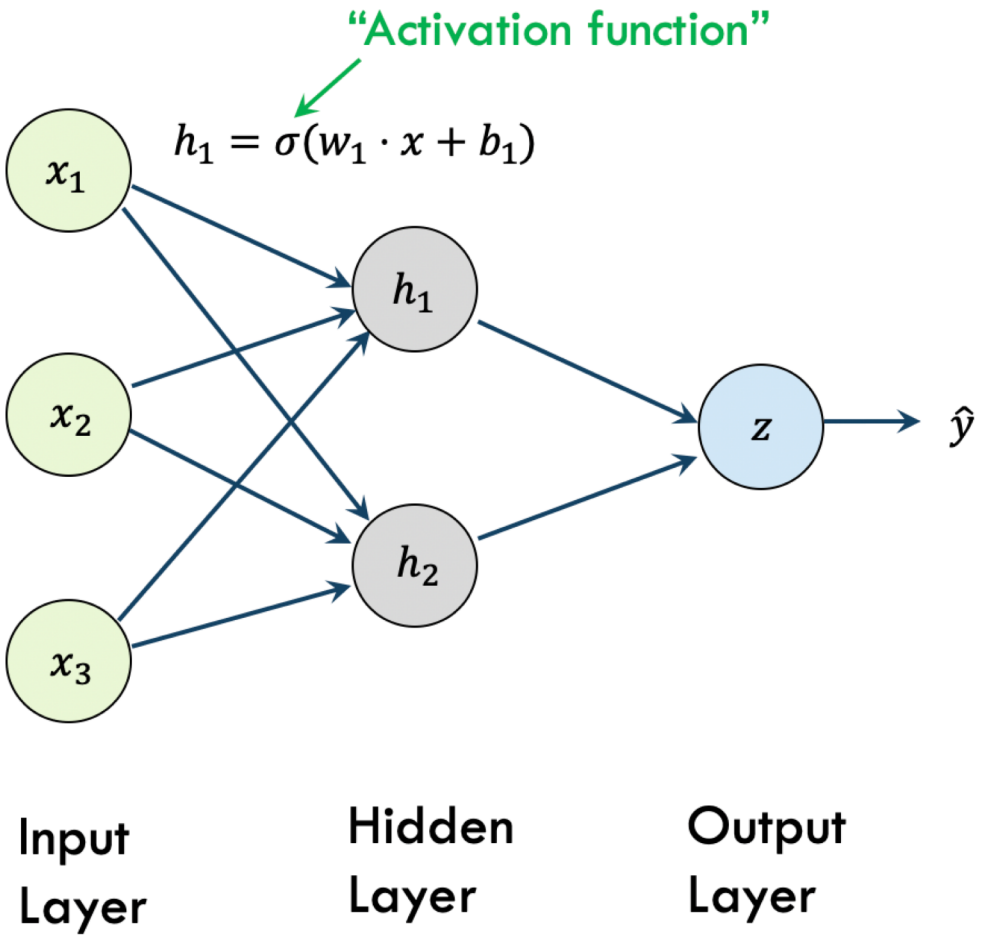
$$b \leftarrow b - \alpha \frac{\delta J(w,b)}{\delta b}$$

"Learning Rate"

"Slope": direction of steepest decrease of J



REVIEW: NEURAL NETWORKS



Each neuron computes a linear function of its inputs, and (optionally) applies a nonlinear activation function

OUR EVERYDAY DECISIONS ARE INCREASINGLY DRIVEN BY RECOMMENDATIONS

RECOMMENDATION ENGINE FOR ACADEMIC RESEARCH PAPERS

Arxiv Sanity Preserver
Built in spare time by @karpathy to accelerate research.
Serving last 125845 papers from cs.[CV|CL|LG|AI|NE]/stat.ML

most recent | top recent | top hype | friends | discussions | recommended | library

Only show v1 | Last day | Last 3 days | Last week | Last month | Last year | All time

Recommended papers: (based on SVM trained on tfidf of papers in your library, refreshed every day or so)

Semi-supervised Anomaly Detection on Attributed Graphs
Atsutoshi Kumagai, Tomoharu Iwata, Yasuhiro Fujiwara
2/27/2020 stat.ML | cs.LG
10 pages

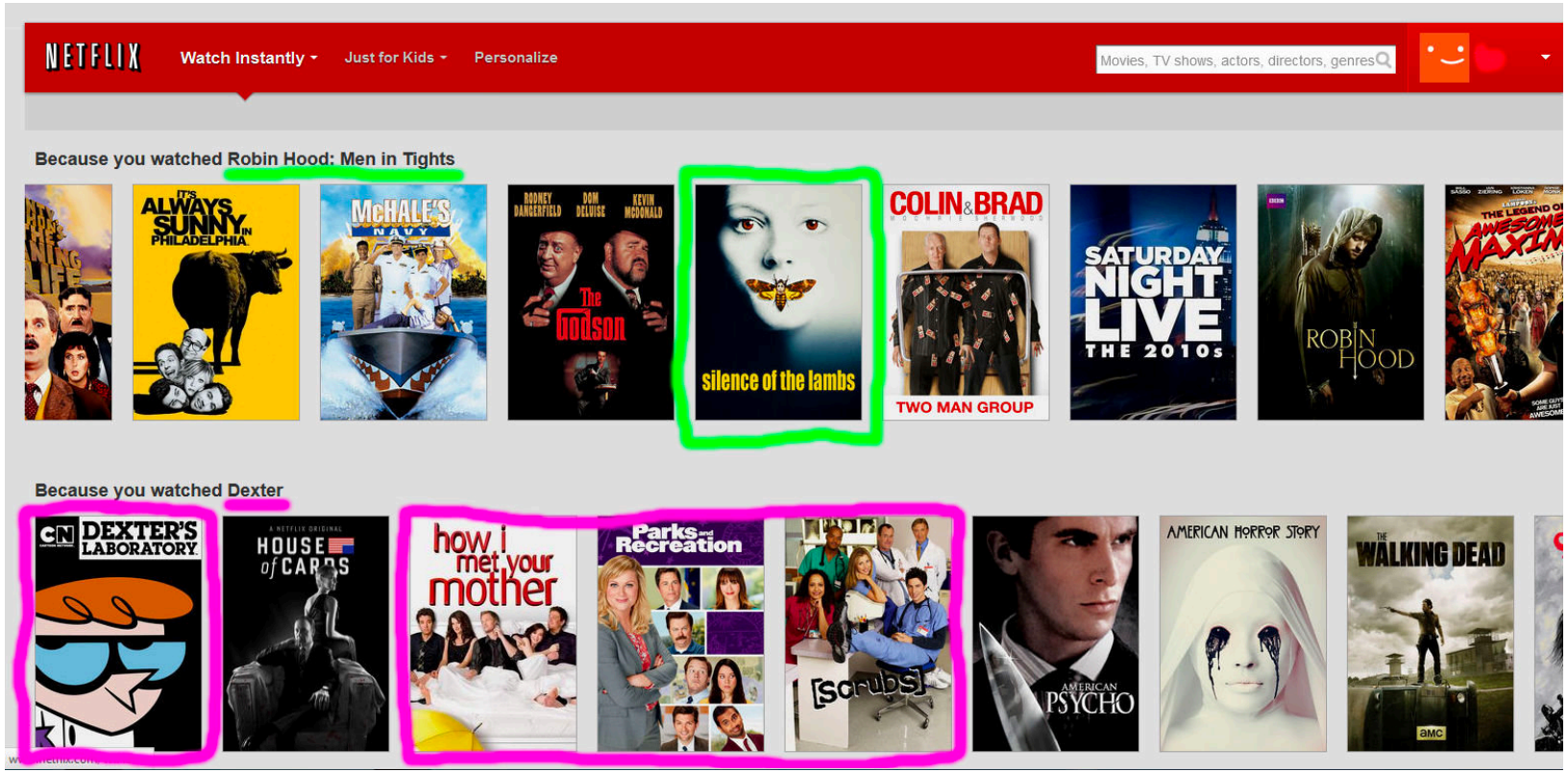
2002.12011v1 pdf
show similar | discuss

We propose a simple yet effective method for detecting anomalous instances on an attribute graph with label information of a small number of instances. Although with standard anomaly detection methods it is usually assumed that instances are independent and identically distributed, in many real-world

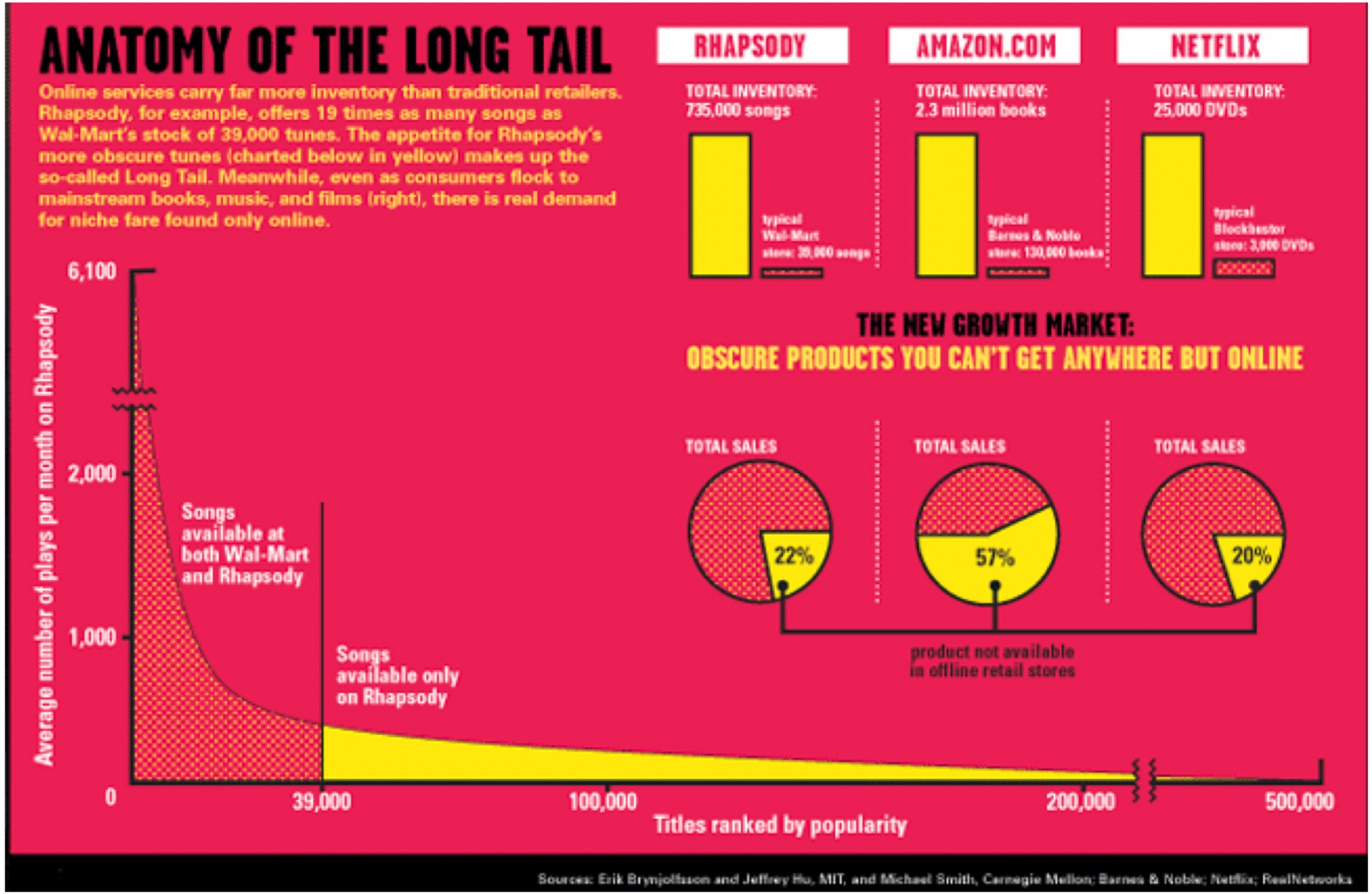
FROM SCARCITY TO OVER-ABUNDANCE OF CHOICE

The Web enables near-frictionless access to an over-abundance of choices, necessitating algorithms to recommend them to us

Netflix: about 80% of content watched is through its recommendation system[1]



Source: https://www.wired.co.uk/article/how-do-netflixs-algorithms-work-machine-learning-helps-to-predict-what-viewers-will-like
https://www.theguardian.com/film/filmblog/2015/jul/10/netflix-weirdest-recommendations-inbetweeners



Source: http://www.wired.com/wired/archive/12.10/tail.html

PROBLEM FORMULATION

U: set of users

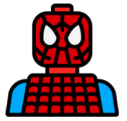
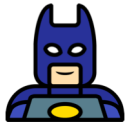
P: set of products

R: set of ratings, for some pairs of users and items

Goal: predict ratings for a set of "test" pairs of users and items

Products **P**

Users **U**

5		?
1		2
	4	3

EXPLICIT VS IMPLICIT FEEDBACK

Explicit: directly use rating values

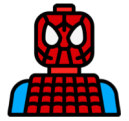
- E.g. 1 to 5 stars, like/dislike

Implicit: use data which is a proxy for user interest:

- E.g. whether a user watched a movie, click history, time spent on a webpage etc.
- Usually more abundant and easier to collect

Products **P**

Users **U**


5		?
1		2
	4	3

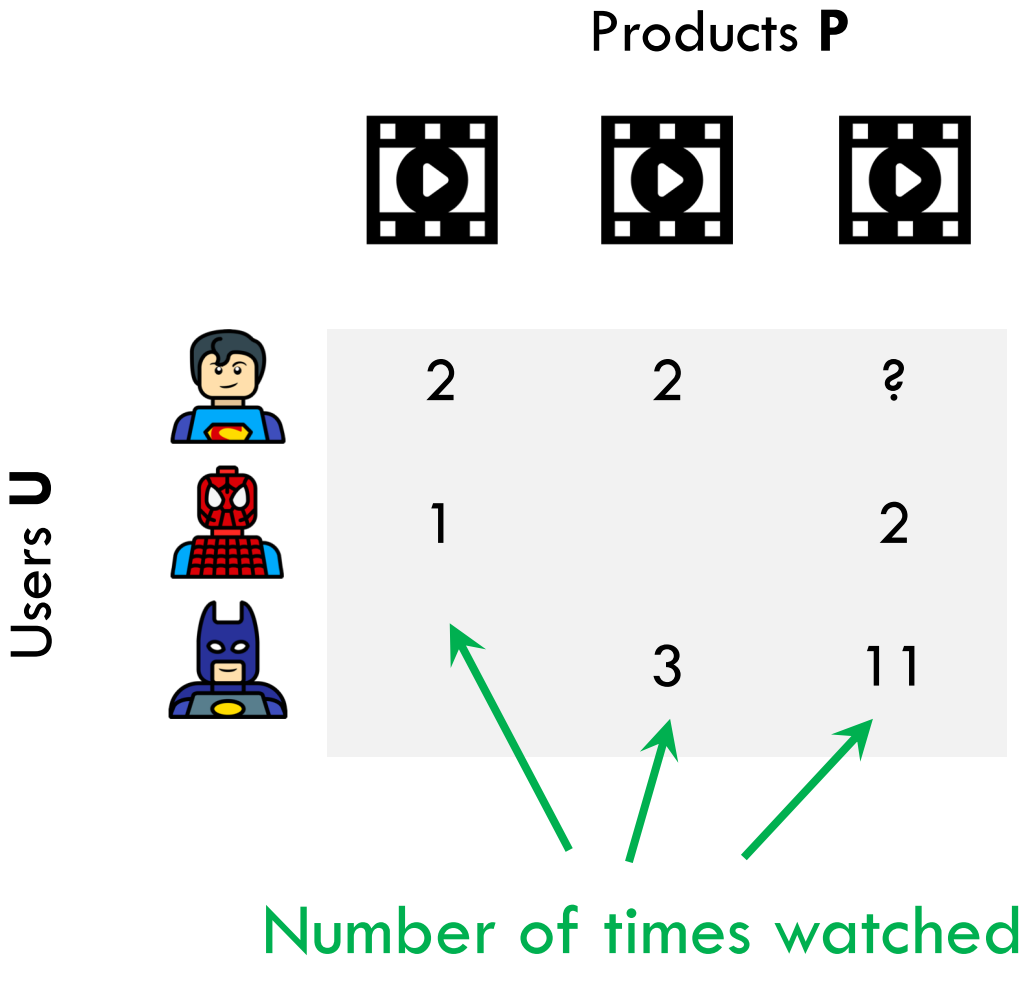
EXPLICIT VS IMPLICIT FEEDBACK

Explicit: directly use rating values

- E.g. 1 to 5 stars, like/dislike

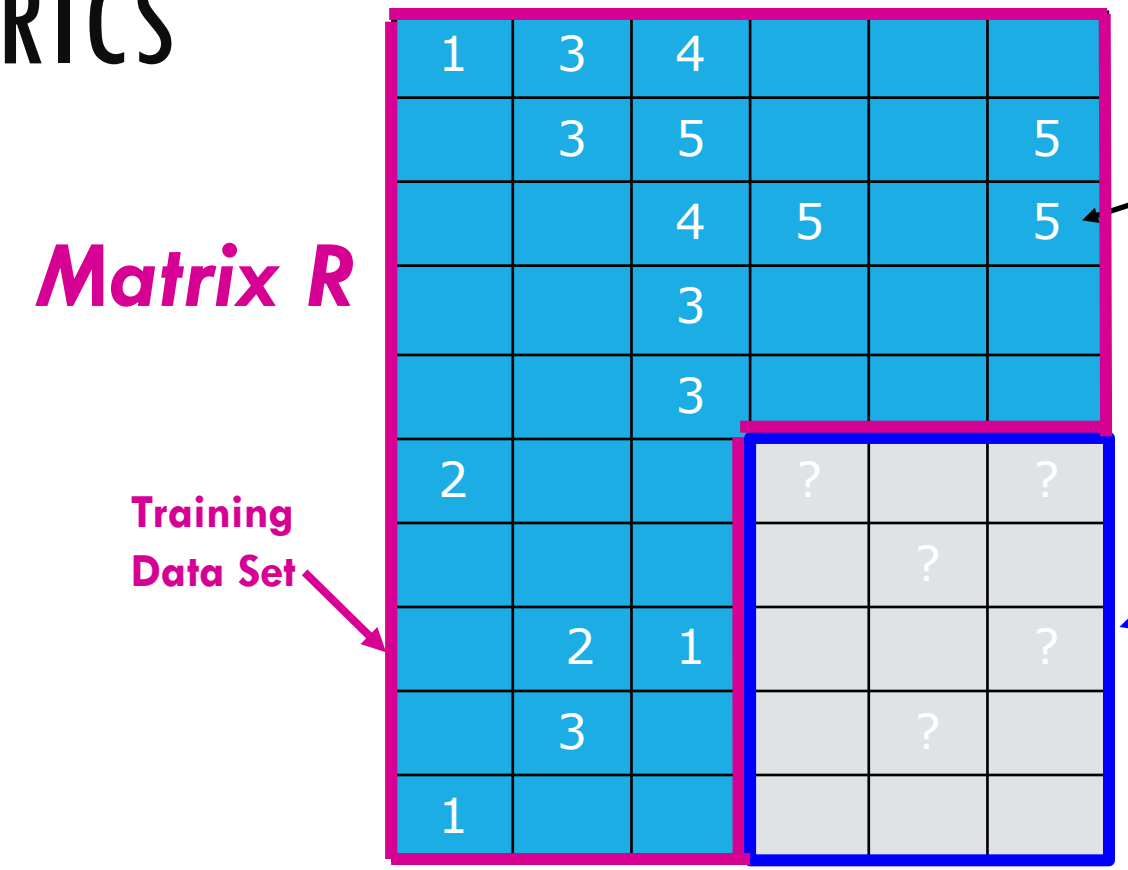
Implicit: use data which is a proxy for user interest:

- E.g. whether a user watched a movie, click history, time spent on a webpage etc.
- Usually more abundant and easier to collect



EVALUATION METRICS

Root Mean Square Error (RMSE) on the observed ratings is one of the most common metrics, and was used for the $1,000,000 Netflix prize competition (2009)



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

Predicted rating

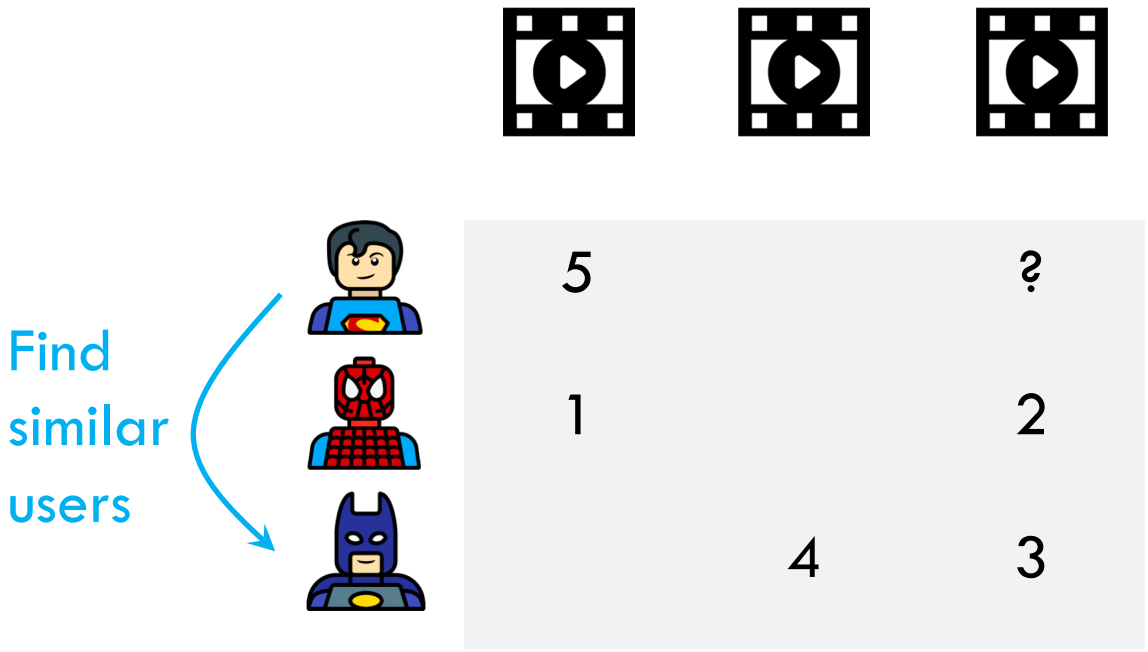
True rating of user ***x*** on item ***i***

RECOMMENDATION OVERVIEW

1. Introduction
2. Similarity-Based Methods
3. Latent Factor Models

COLLABORATIVE FILTERING

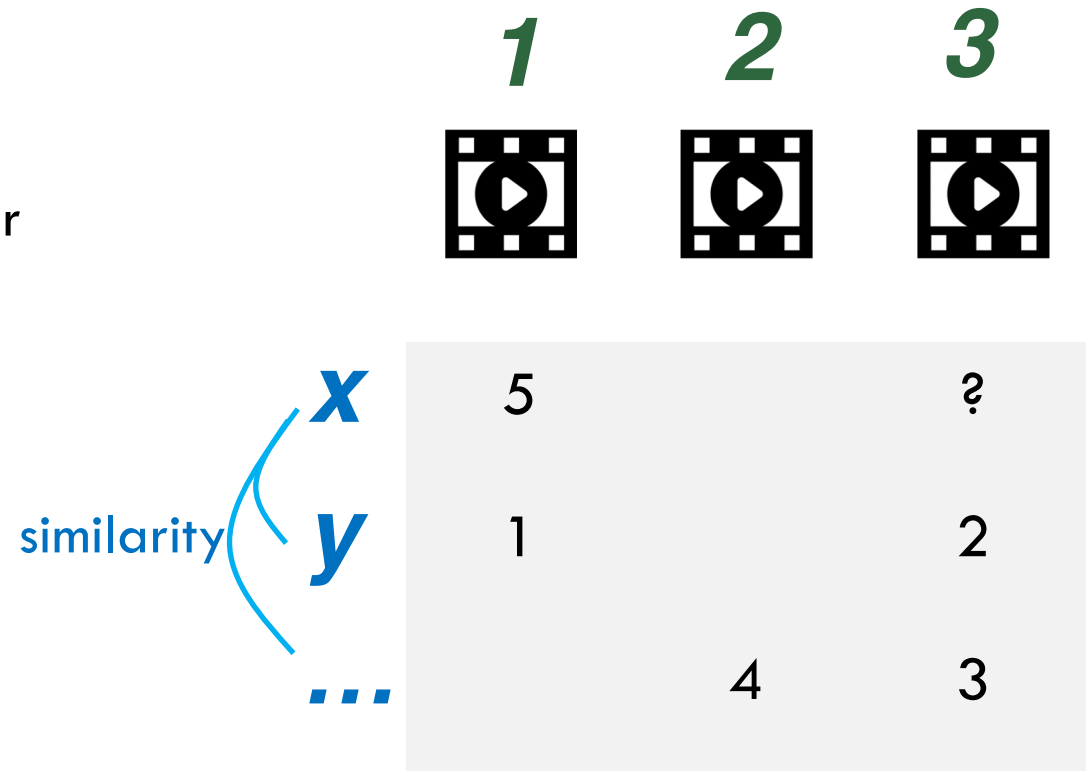
Collaborative filtering models are based on **similarity:** estimate a user's preferences based on the preferences of similar users



SIMILARITY MEASURES

Jaccard similarity: between sets of items rated by each user

- E.g. users x and y have sets {1} and {1, 3}, so their Jaccard similarity is (intersection) / (union) = ½
- Ignores rating values



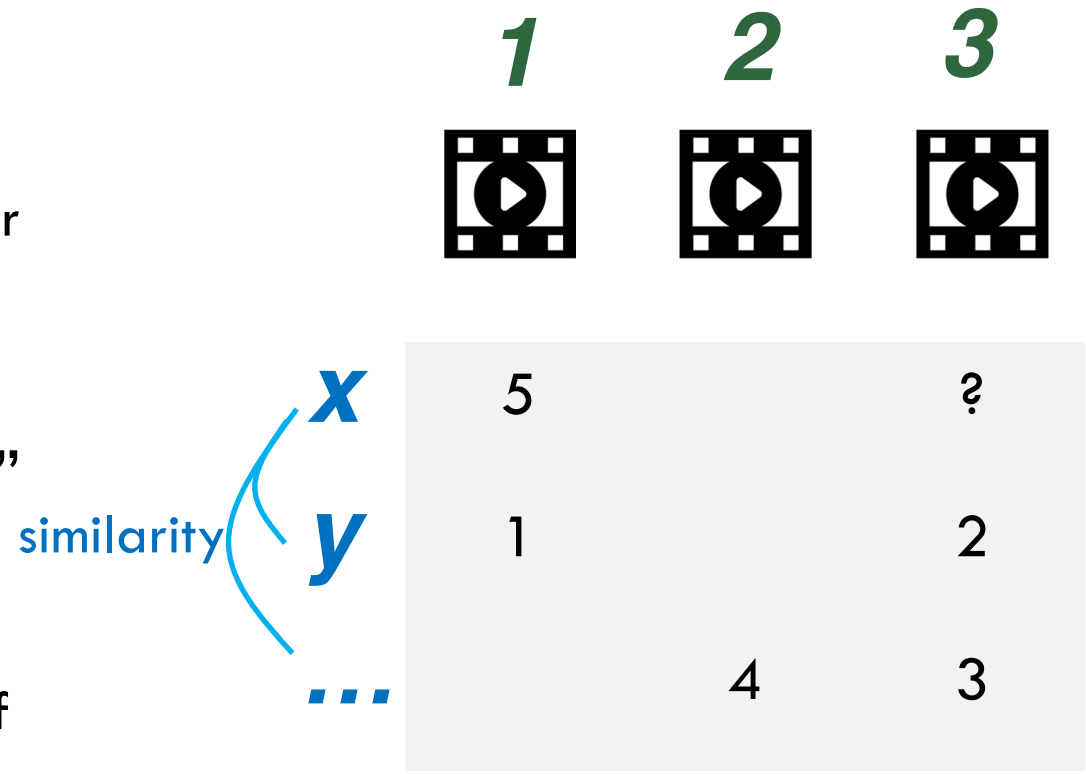
SIMILARITY MEASURES

Jaccard similarity: between sets of items rated by each user

- E.g. users x and y have sets {1} and {1, 3}, so their Jaccard similarity is (intersection) / (union) = ½
- Ignores rating values

Cosine similarity: between "users as points"

- E.g. $r_x = (5, 0, 0)$, $r_y = (1, 0, 2)$
- Cosine similarity = $r_x \cdot r_y / (\|r_x\| \|r_y\|) = 0.577$
- Measures similarity by comparing the directions of the vectors r_x and r_y



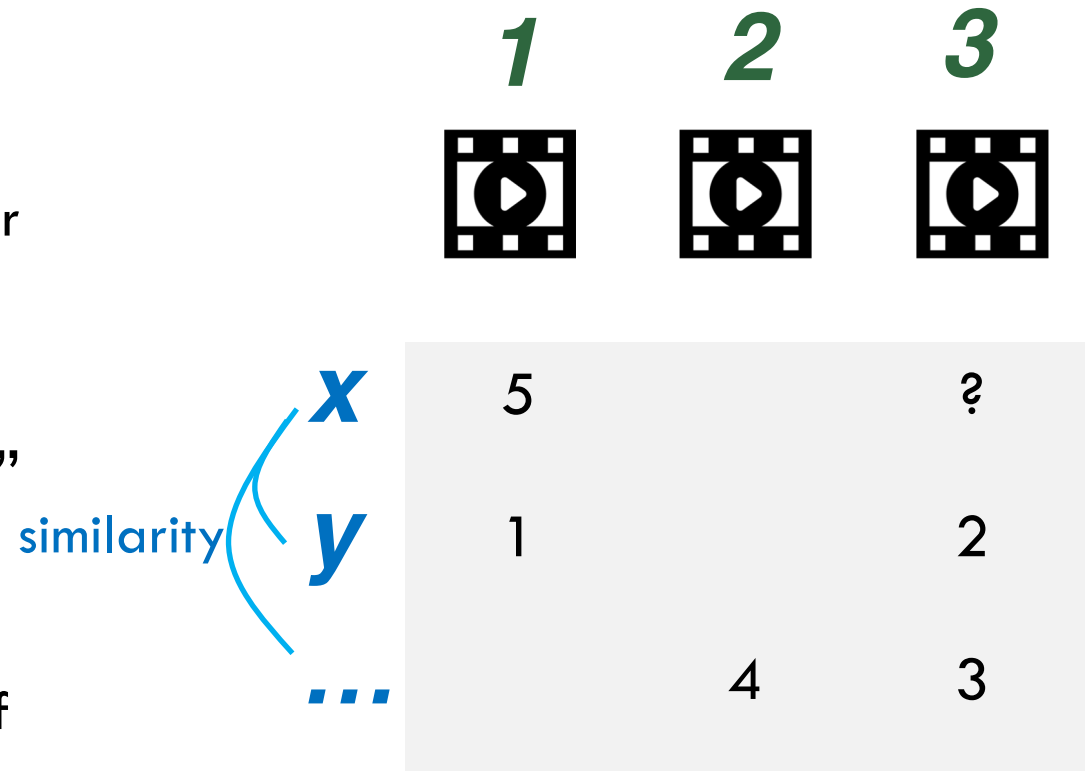
RATING-BASED SIMILARITY MEASURES

Jaccard similarity: between sets of items rated by each user

- E.g. users x and y have sets {1} and {1, 3}, so their Jaccard similarity is (intersection) / (union) = ½
- Ignores rating values

Cosine similarity: between "users as points"

- E.g. $r_x = (5, 0, 0)$, $r_y = (1, 0, 2)$
- Cosine similarity = $r_x \cdot r_y / (\|r_x\| \|r_y\|) = 0.577$
- Measures similarity by comparing the directions of the vectors r_x and r_y



SIMILARITY-BASED RECOMMENDATION

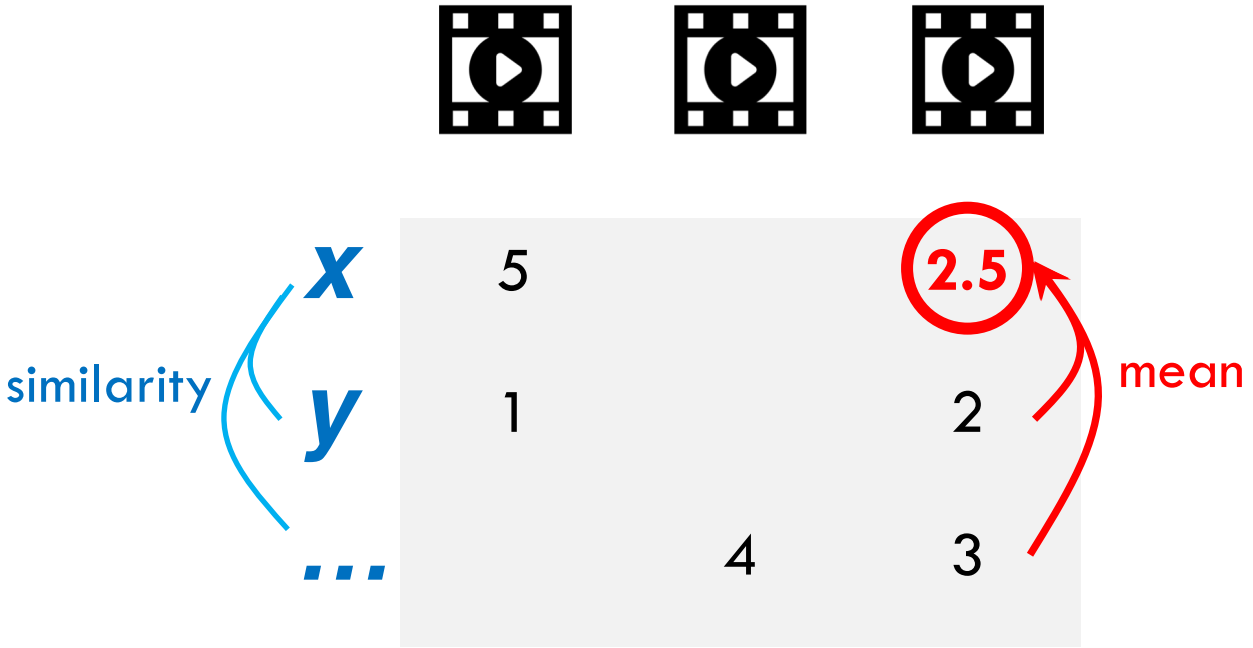
To predict ***x***'s rating for product ***i:***

Let ***N*** be the set of ***k*** most similar users to ***x*** who have rated ***i***

Then we can predict based on:

- **Mean:** predict the mean of the ratings among ***N***, for product ***z***

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



SIMILARITY-BASED RECOMMENDATION

To predict ***x***'s rating for product ***i:***

Let ***N*** be the set of ***k*** most similar users to ***x*** who have rated ***i***

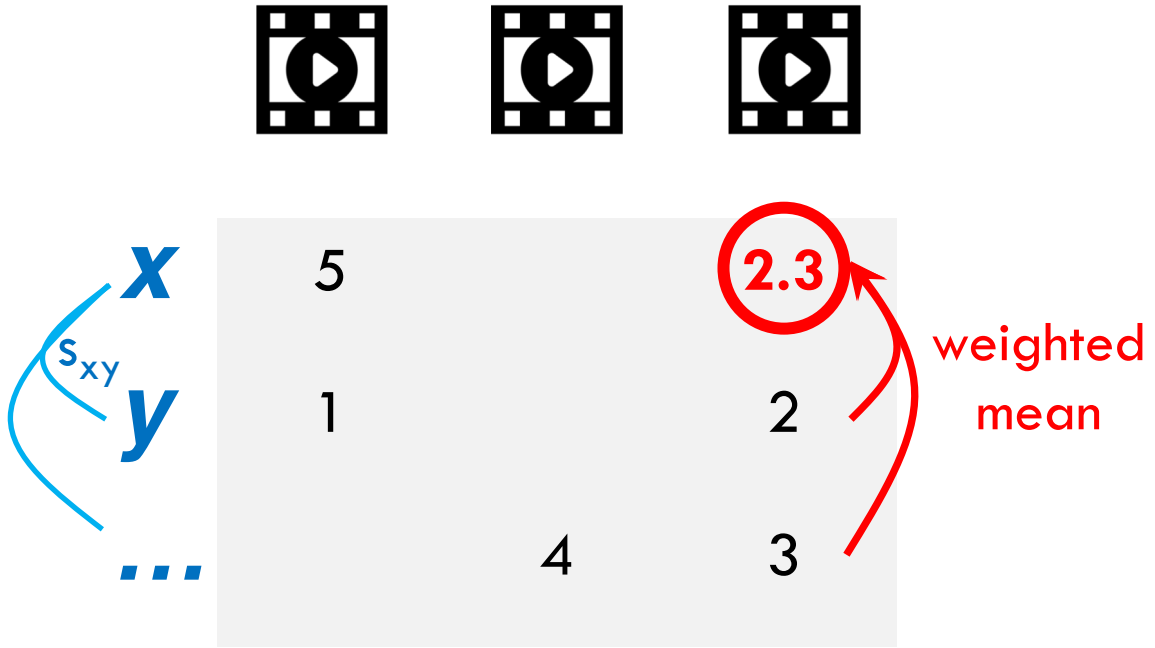
Then we can predict based on:

- **Mean:** predict the mean of the ratings among ***N***, for product ***z***

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

- **Weighted Mean:** predict the weighted mean, weighted by similarity

$$r_{xi} = \frac{\sum_{y \in N} s_{xy} \cdot r_{yi}}{\sum_{y \in N} s_{xy}}$$



ITEM-ITEM SIMILARITY-BASED RECOMMENDATION

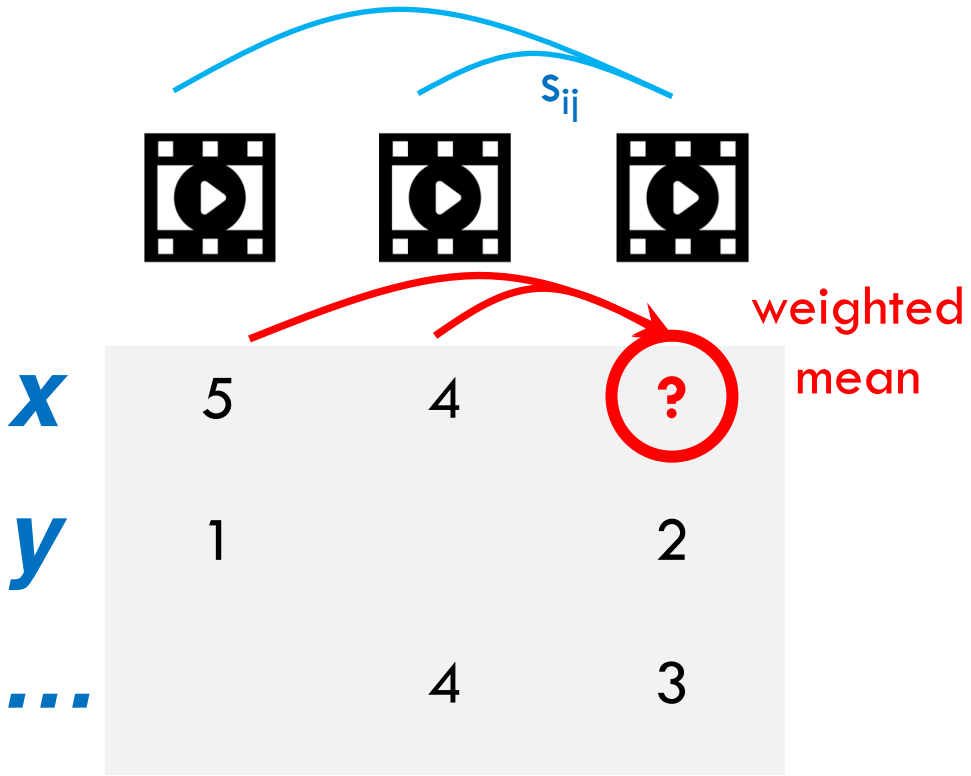
Instead of considering **user-user** similarity, we can consider **item-item** similarity

Same approach as before, except swapping the role of users and items

Similarity between items *i* and *j*

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

Similar items to *i* which are rated by x



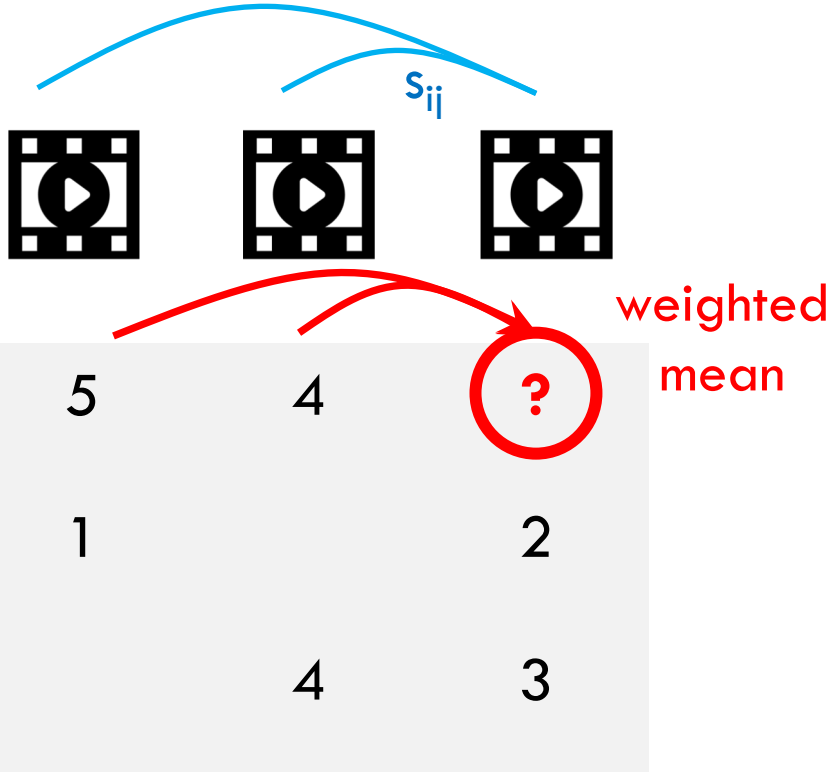
QUIZ: ITEM-ITEM RECOMMENDATION

Similarity between items *i* and *j*

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

Similar items to *i* which are rated by x

Q: Using Jaccard distance for item-item similarity, what is the predicted value (at the "?" position)

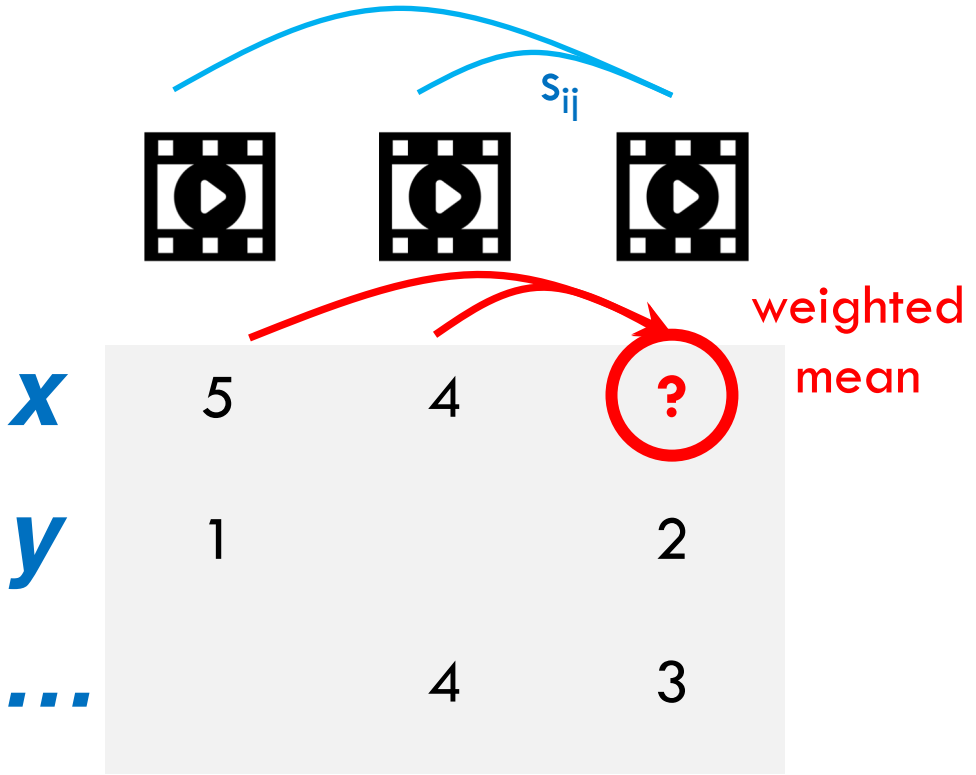


QUIZ: ITEM-ITEM RECOMMENDATION

Similarity between items *i* and *j*

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

Similar items to *i* which are rated by x



Q: Using Jaccard distance for item-item similarity, what is the predicted value (at the "?" position)
A: the Jaccard similarities between column 3 and the other 2 columns are both 1/3. Thus, the resulting weights are equal and the weighted mean is **4.5**.

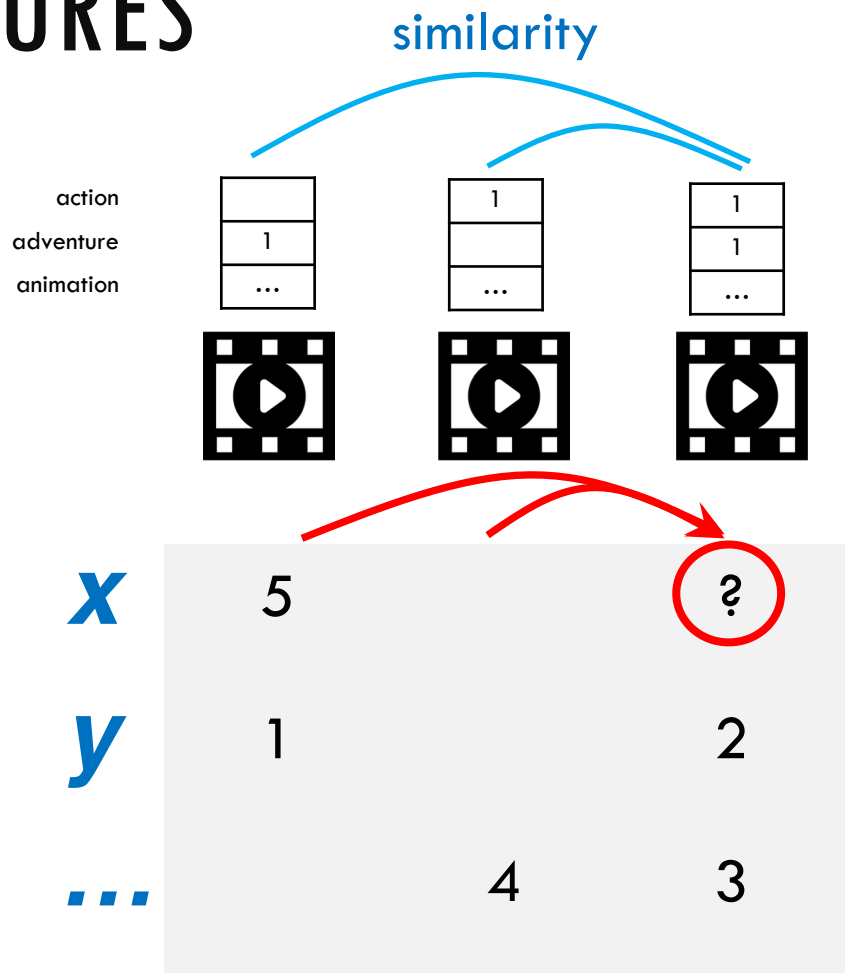
CONTENT-BASED SIMILARITY MEASURES

Previously, we measured item-item similarity based on measures like Jaccard distance, which consider the sets of users who watched an item

An alternate choice is to use **content-based similarity:** if we have some features to represent each item, we can measure similarity between these feature vectors

Given such a similarity measure, we can recommend based on top-k most similar items in the same way as before

Using such features can be helpful in addressing the **cold-start problem** (i.e. the difficulty in recommending a new movie, which has no history of being watched)



COMMON PRACTICE: COLLABORATIVE FILTERING WITH BASELINE ESTIMATES

Idea: some users tend to give higher ratings; i.e. they have a high **baseline**

Similarly, some products tend to receive higher ratings (e.g. they are better)

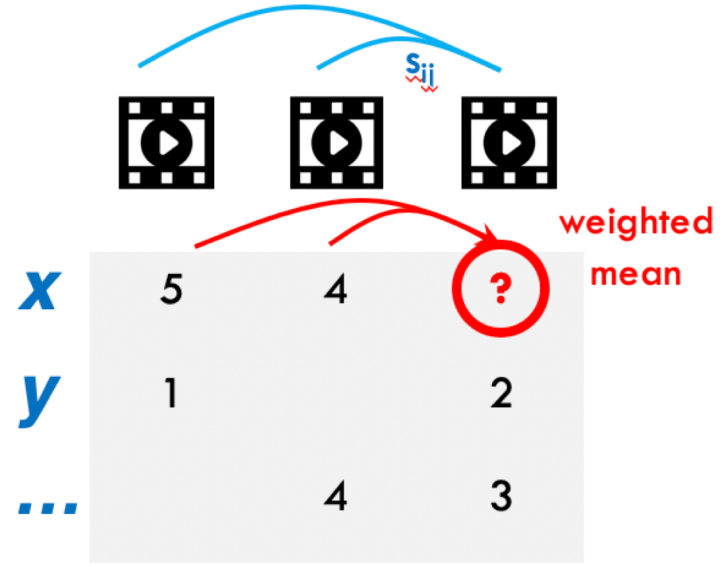
Define:

- **Overall mean rating** μ: mean over all ratings
- **Rating deviation of user *x*:** (Average rating given by ***x***) - μ
- **Rating deviation of movie *i*:** (Average rating received by ***i***) - μ
- **Baseline estimate (for user *x* and movie *i*):**

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

Overall mean

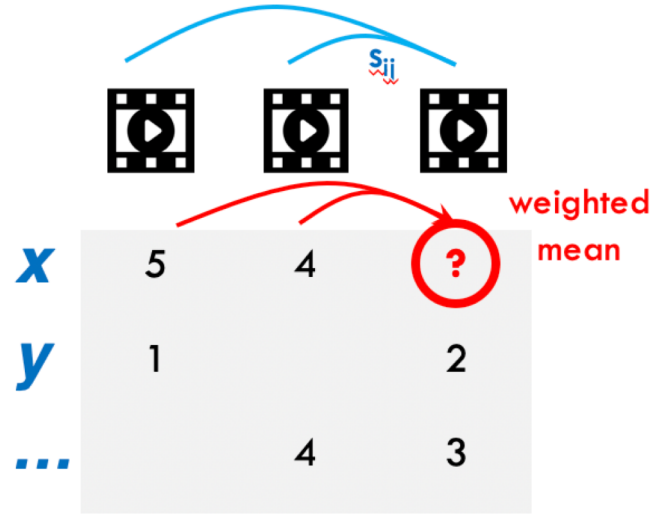
Rating deviation of user and movie



COMMON PRACTICE: COLLABORATIVE FILTERING WITH BASELINE ESTIMATES

Previous item-item similarity approach:

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



Instead, we first subtract the baseline ratings, then add them before predicting:

Subtract baseline rating

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

Add baseline rating

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

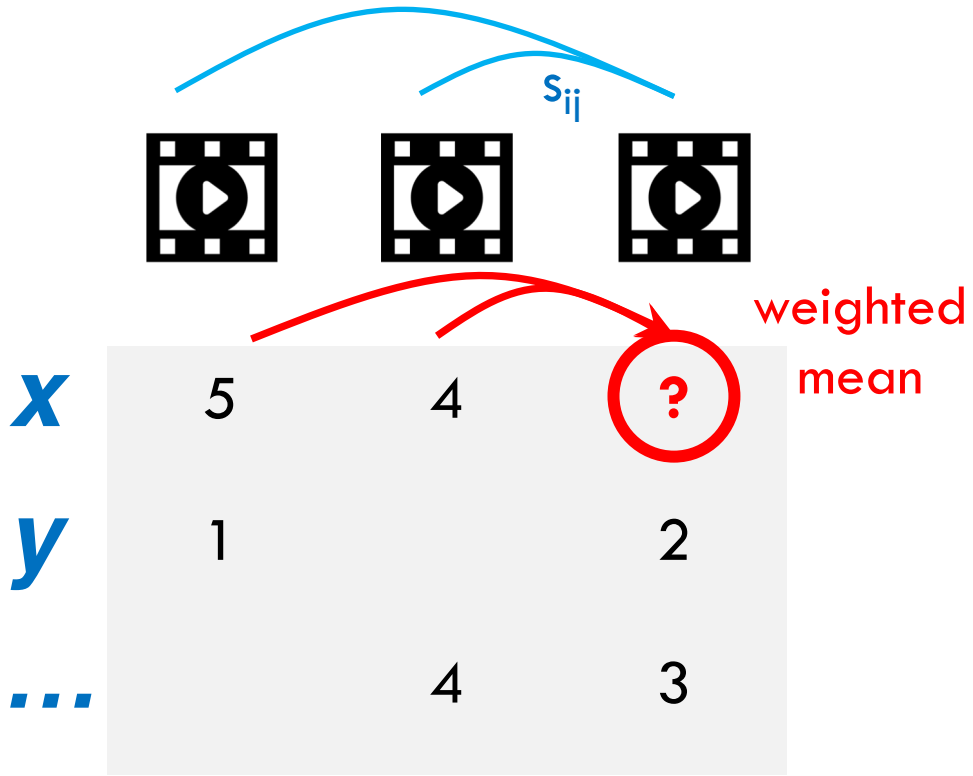
Overall mean

Rating deviation of user and movie

ITEM-ITEM VS USER-USER

In practice, it has been observed that item-item similarity performs better than user-user

Why? Items (e.g. movies) are simple, but users can be complex. Thus, looking at similar items provides a more accurate predictor.



PROS AND CONS OF COLLABORATIVE FILTERING

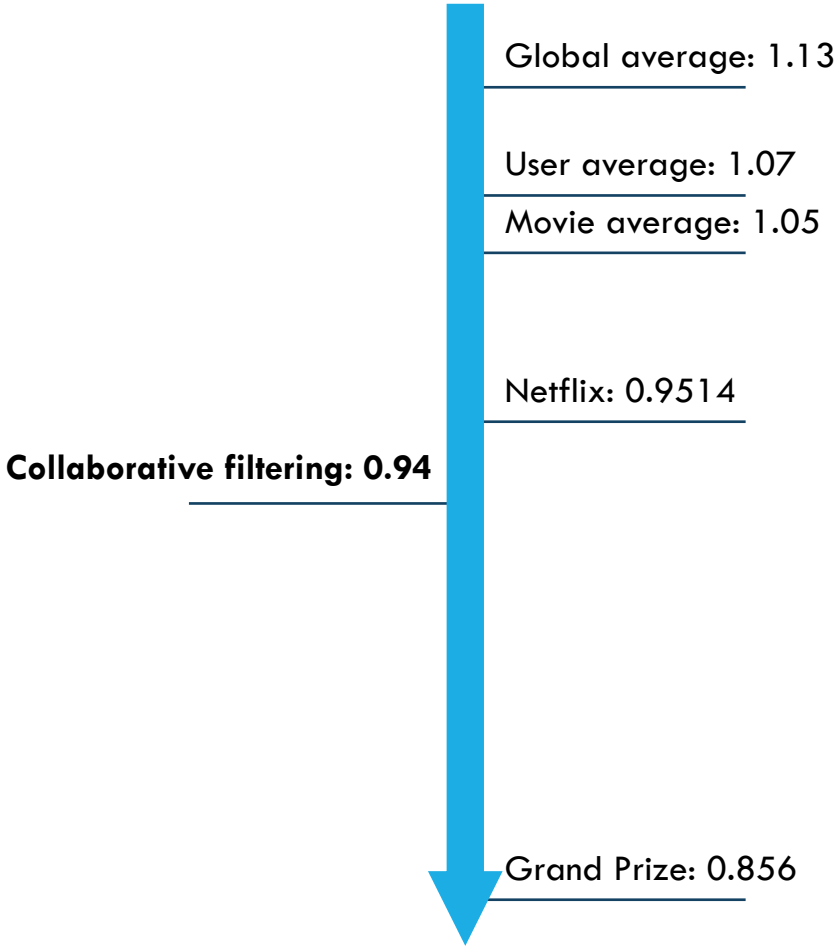
Pros

- **No features required**: as compared to feature-based prediction approaches

Cons

- **Sparsity:** if the rating matrix is sparse, it is hard to find other items rated by the same user
- **Cold start**: cannot recommend for users or items with no ratings
- **Sensitive to k** (no. of similar items)**:** higher k = need more items before we can recommend. Lower k = noisy predictions.
- **Only considers "local effects",** i.e. it always predicts based on similar items. In contrast, our next approach considers "global effects", i.e. considering all items.

PERFORMANCE COMPARISON (NETFLIX CHALLENGE)



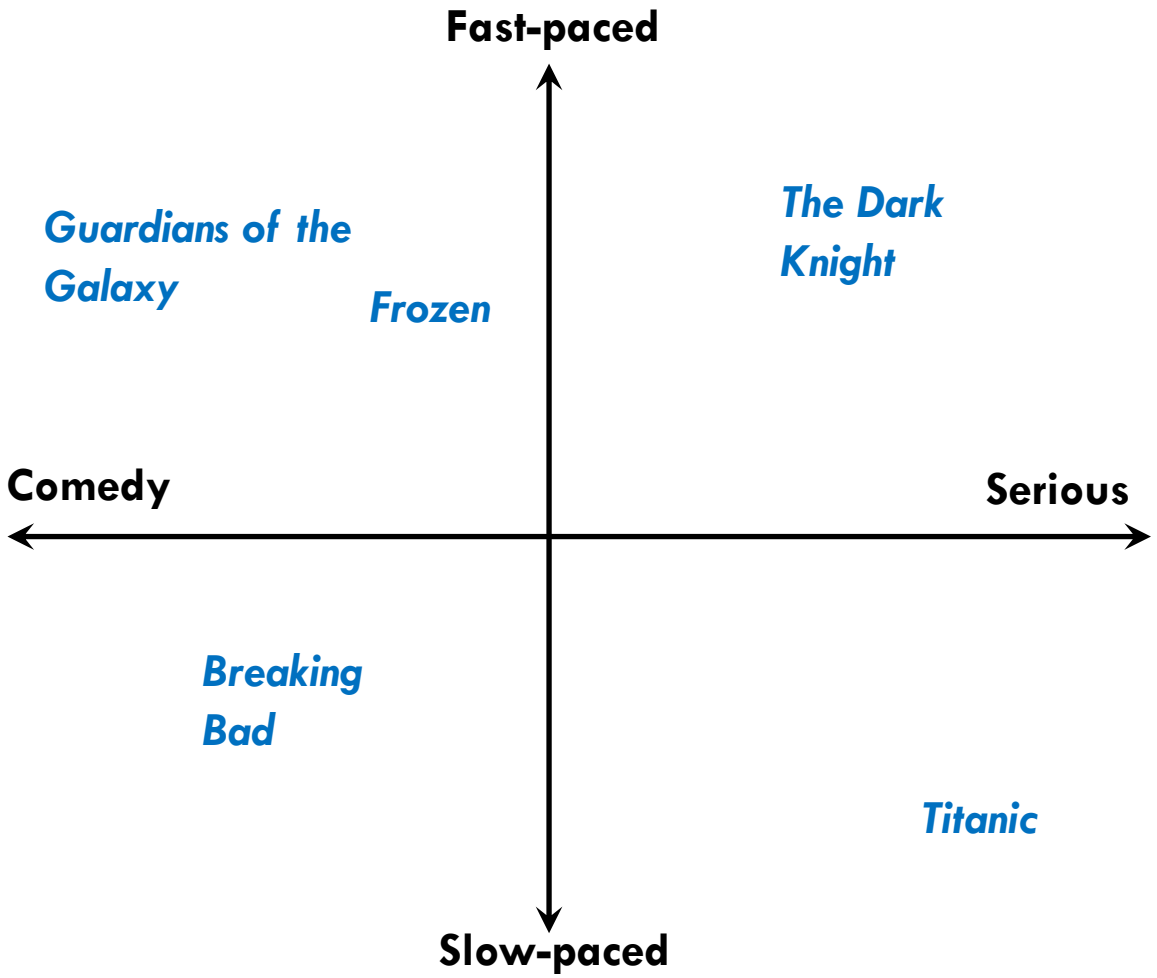
RECOMMENDATION OVERVIEW

1. Introduction
2. Similarity-Based Methods
3. Latent Factor Models

LATENT FACTOR MODELS: INTRODUCTION

Based on the idea that there are underlying "**factors**" that describe items (movies)

- E.g. some movies are serious; others are funny



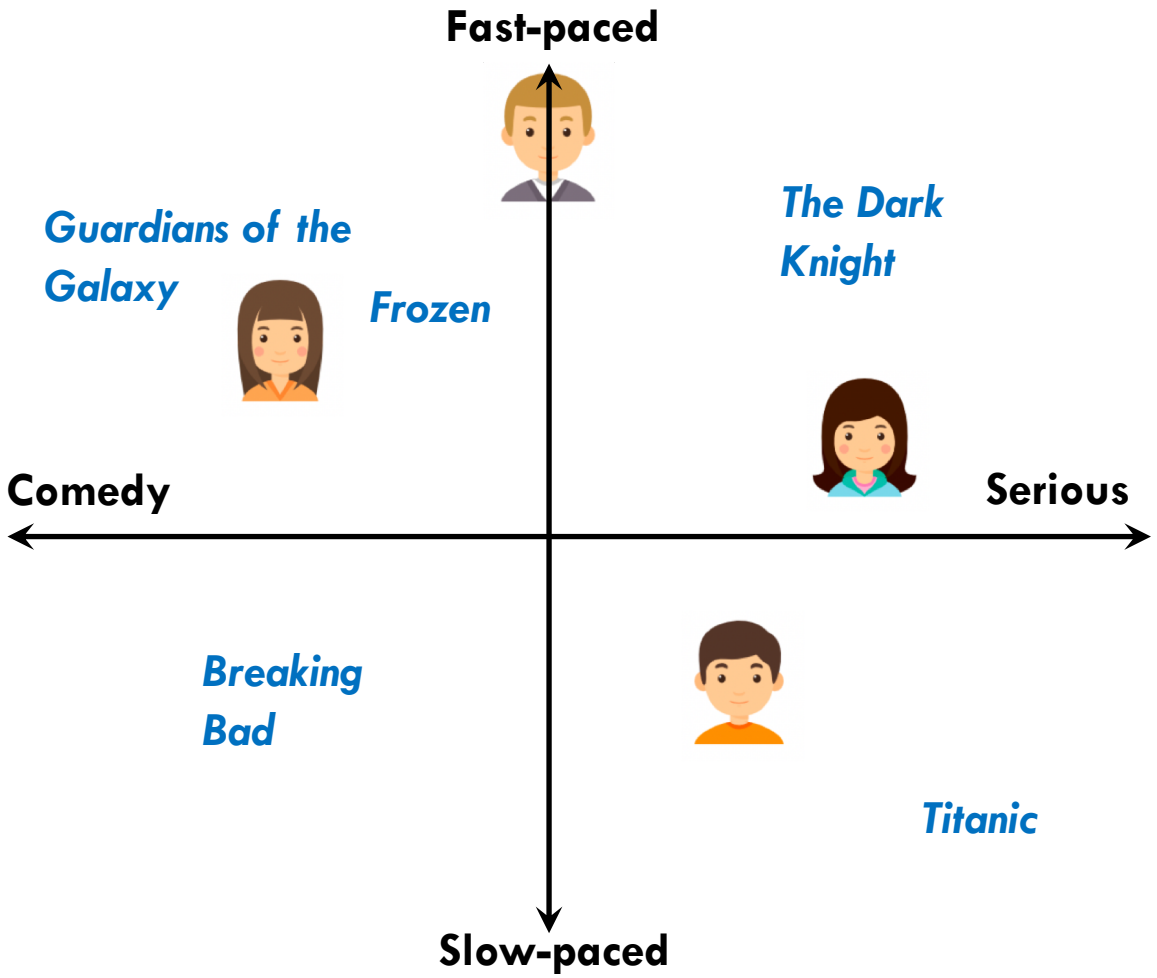
LATENT FACTOR MODELS: INTRODUCTION

Based on the idea that there are underlying "**factors**" that describe items (movies)

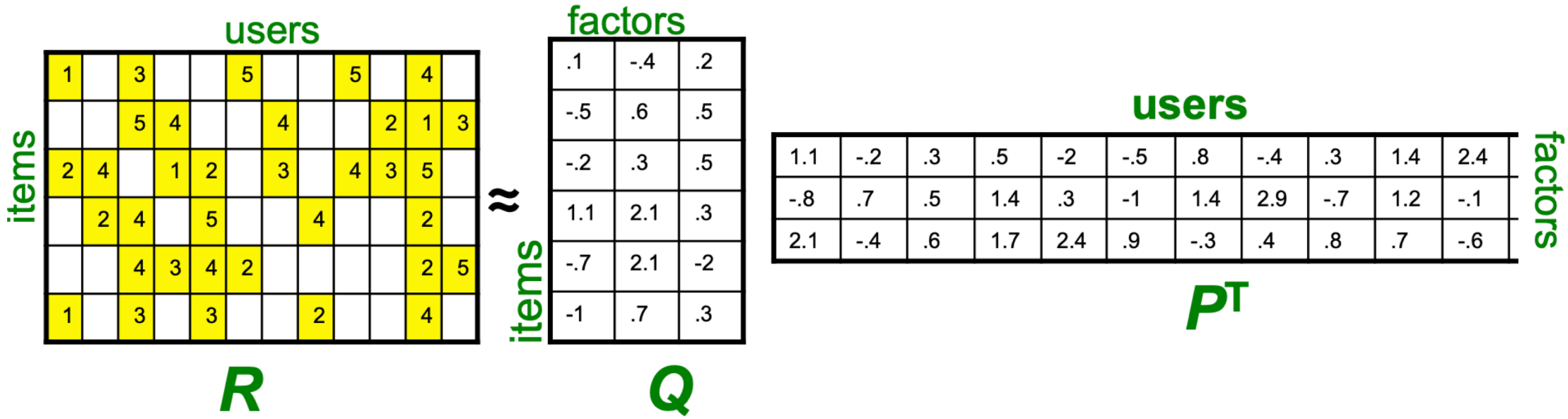
- E.g. some movies are serious; others are funny

These factors then affect how users rate them:

- E.g. some users like serious movies; others prefer funny movies



MATRIX FACTORIZATION MODEL



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	

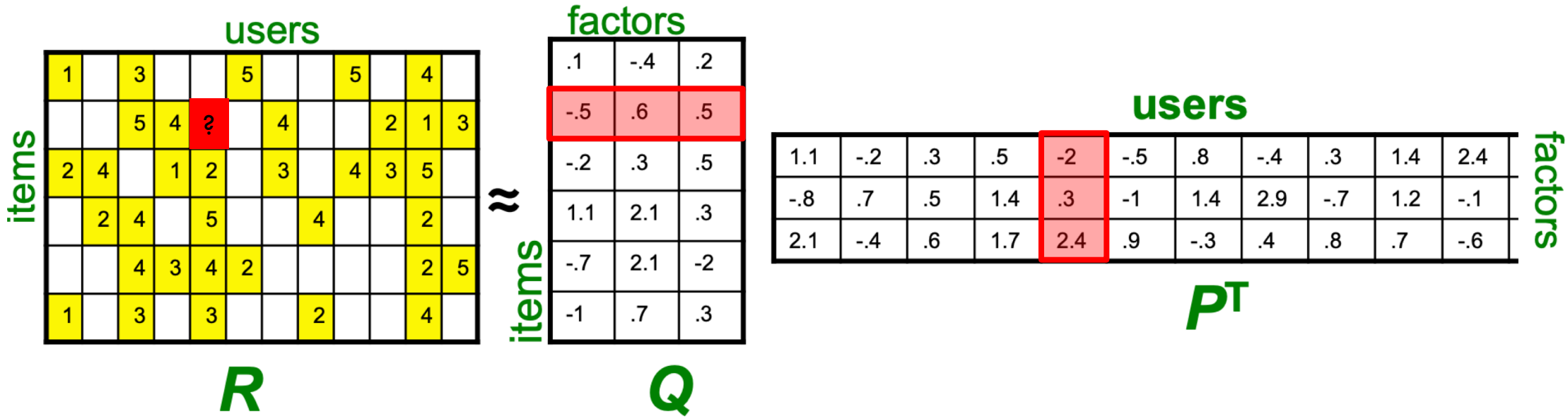
.1	-.4	.2
-.5	.6	.5
-.2	.3	.5
1.1	2.1	.3
-.7	2.1	-2
-1	.7	.3

1.1	-.2	.3	.5	-2	-.5	.8	-.4	.3	1.4	2.4	
-.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	

Approximate the rating matrix as a product of two matrices: $R = Q \cdot P^T$

This approximation only needs to be accurate on the known ratings, ignoring the missing parts

PREDICTING A TEST RATING



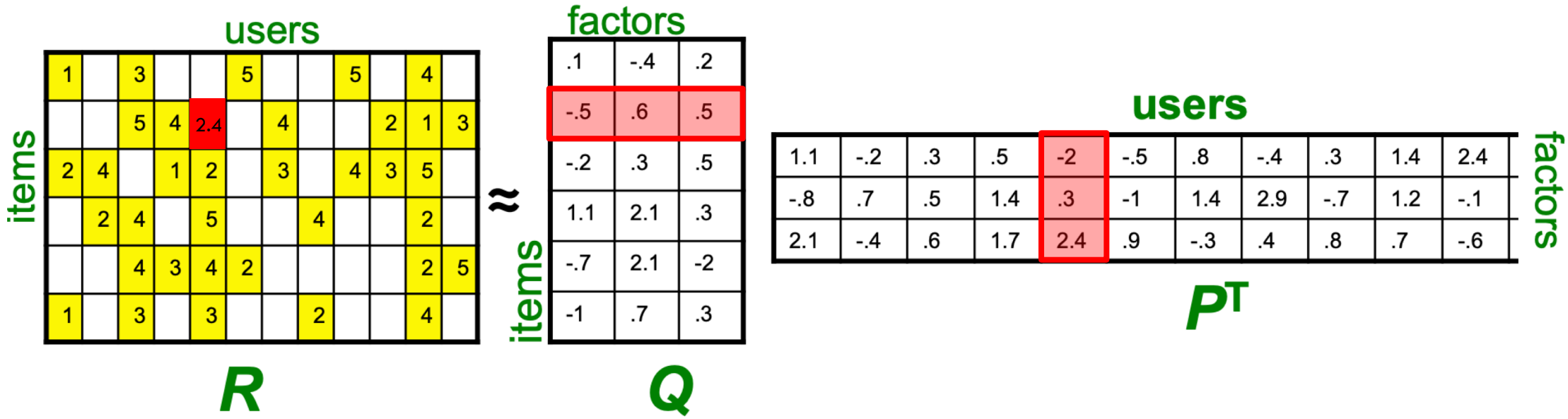
Prediction for user x and item i:

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

Row i of Q

Column x of p

PREDICTING A TEST RATING



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	

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

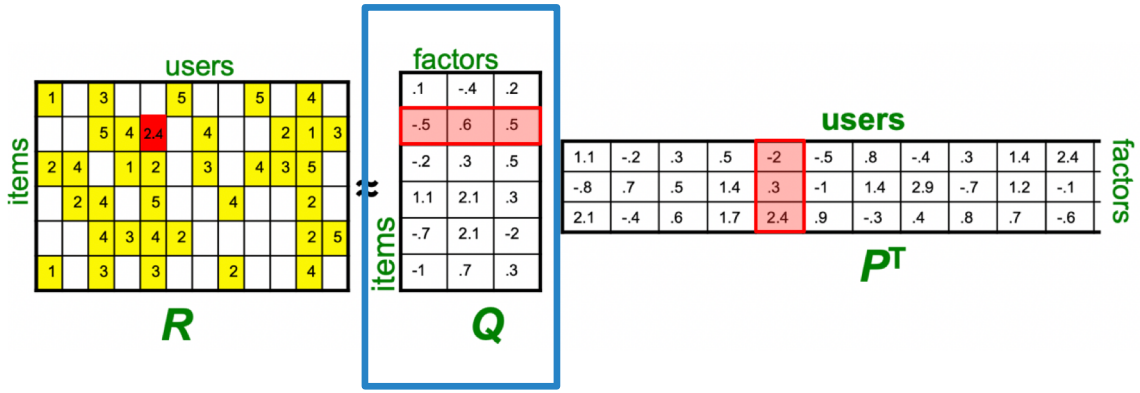
1.1	-.2	.3	.5	-2	-.5	.8	-.4	.3	1.4	2.4
-.8	.7	.5	1.4	.3	-1	1.4	2.9	-.7	1.2	-.1
2.1	-.4	.6	1.7	2.4	.9	-.3	.4	.8	.7	-.6

Prediction for user x and item i:

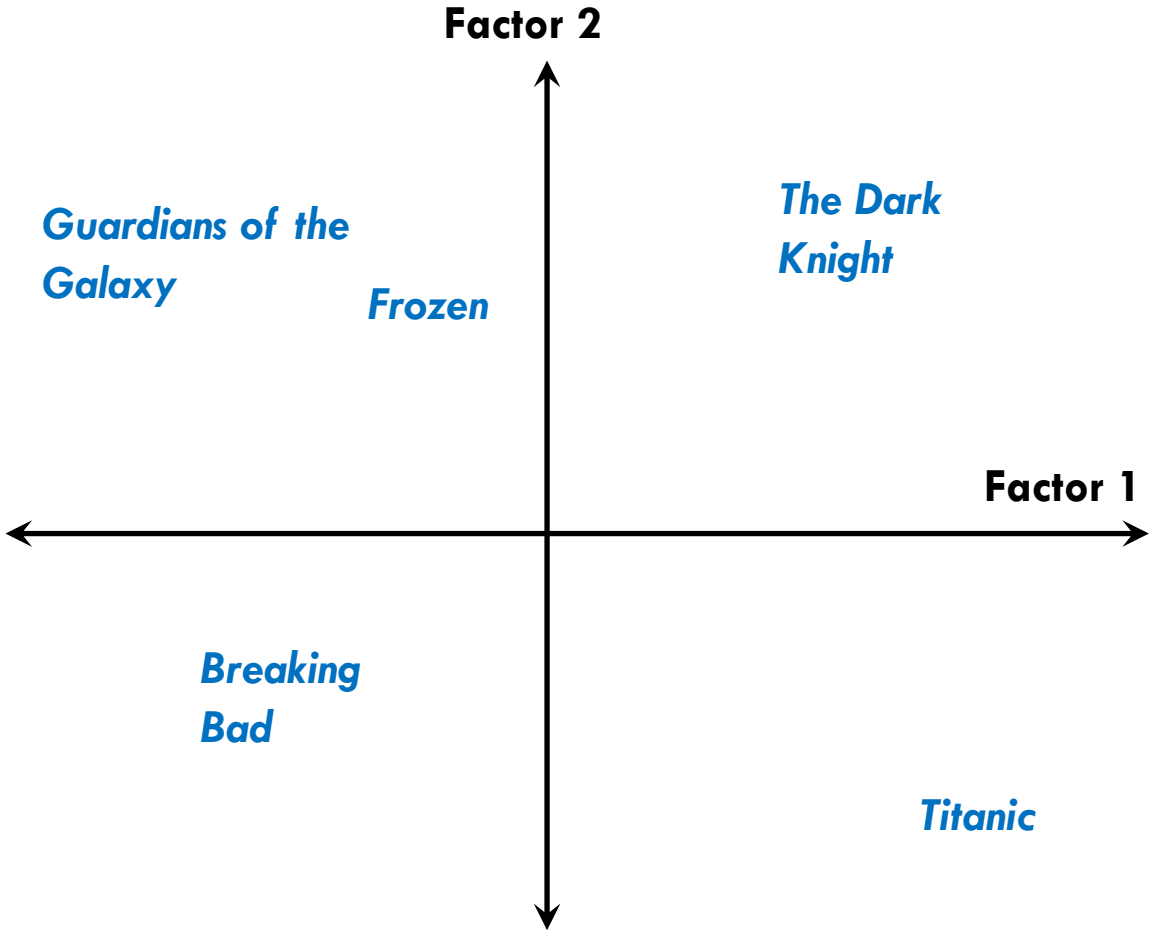
$$\hat{r}_{xi} = q_i \cdot p_x$$

Row i of Q Column x of p

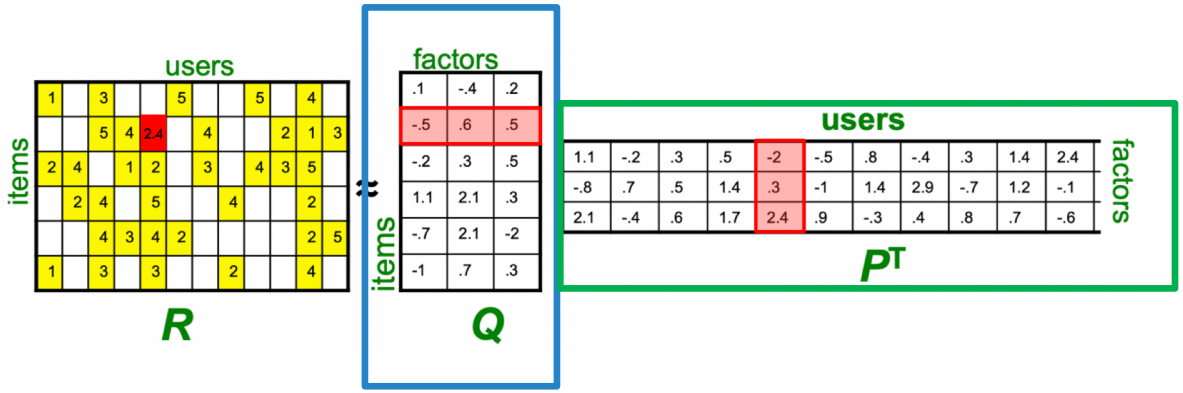
INTERPRETATION USING LATENT FACTORS

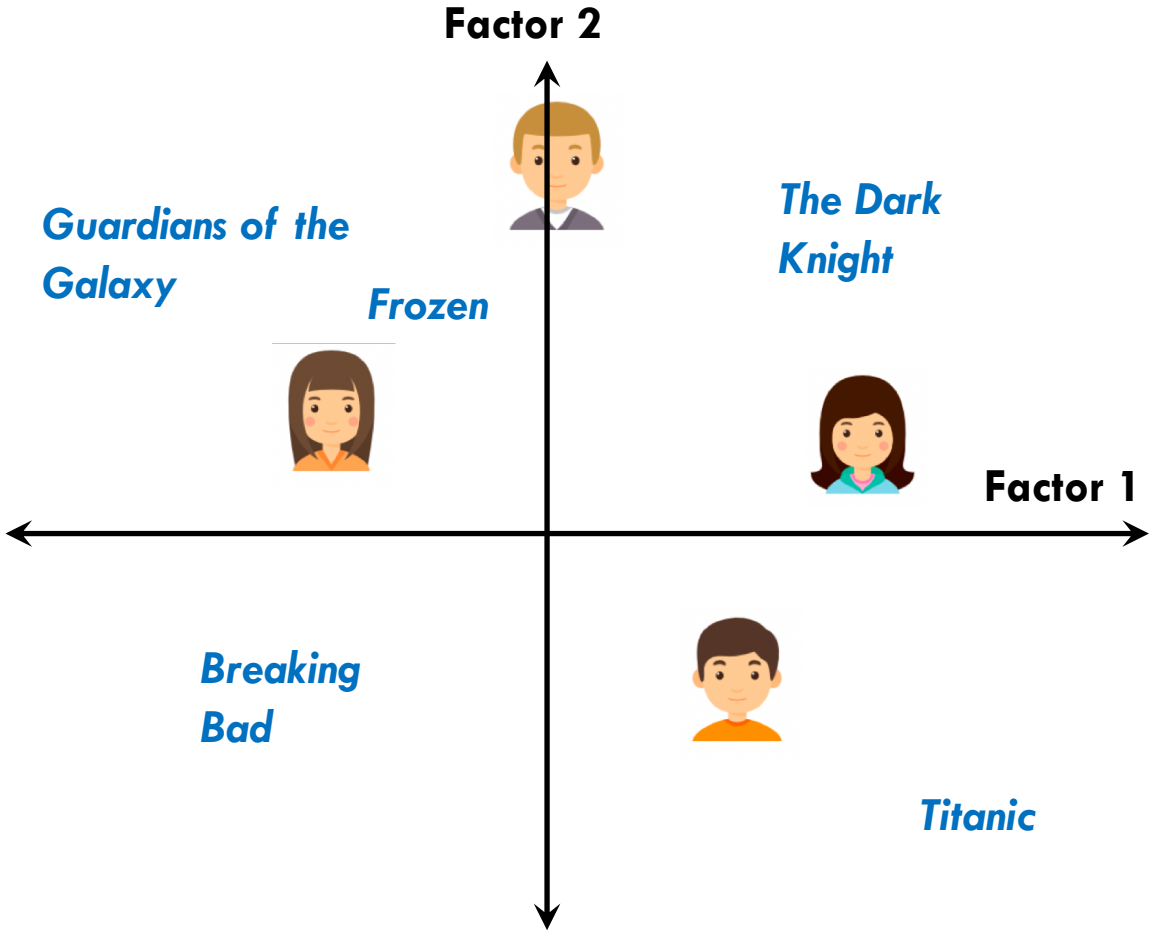


Item factors: each item ***i*** is a represented by its "latent factors", given in the vector q_i



INTERPRETATION USING LATENT FACTORS

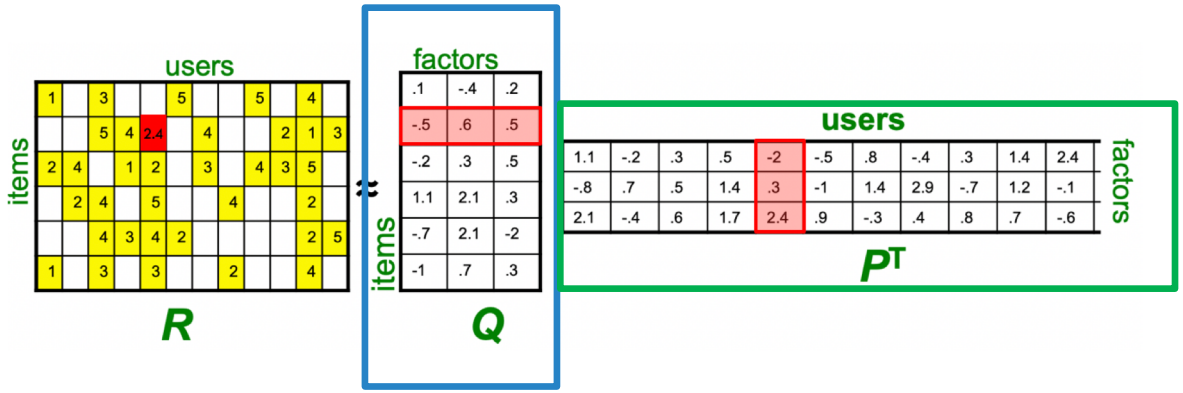




Item factors: item ***i*** is a represented by its "latent factors" vector q_i

User factors: user ***x*** is a represented by its "latent factors" vector p_x

INTERPRETATION USING LATENT FACTORS

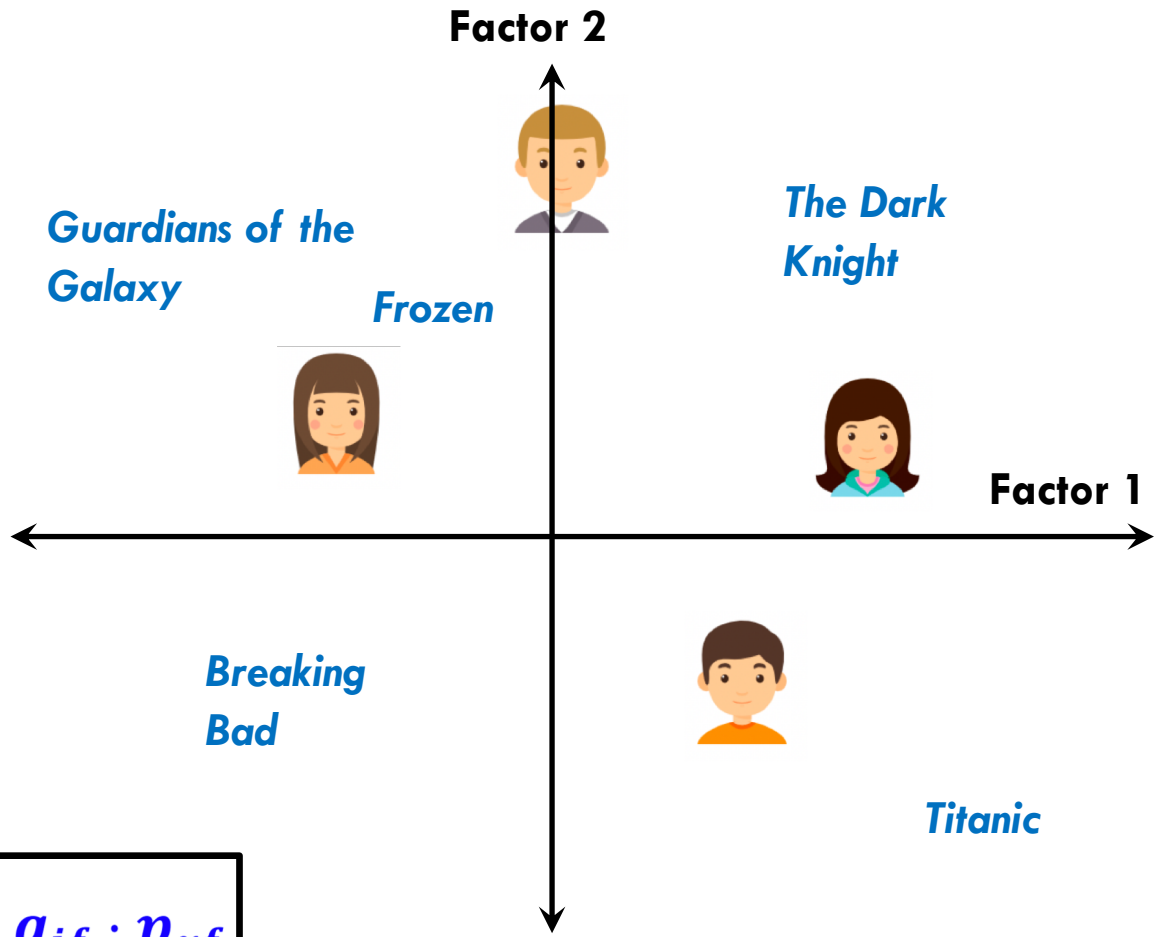


Item factors: item ***i*** is a represented by its "latent factors" vector q_i

User factors: user ***x*** is a represented by its "latent factors" vector p_x

Predicted rating is the dot product between latent factors:

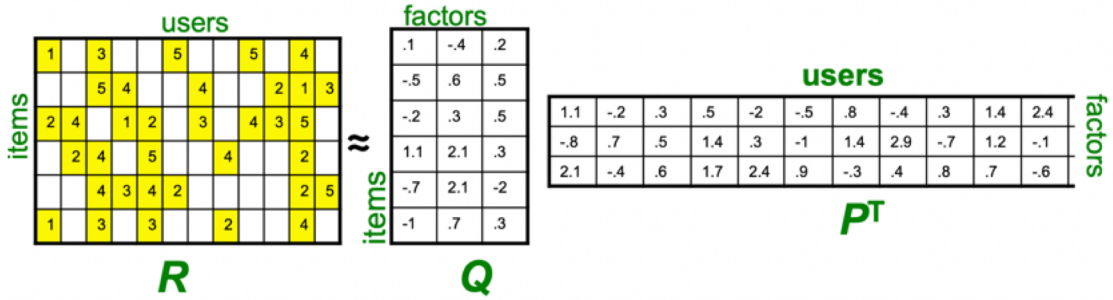
$$\hat{r}_{xi} = q_i \cdot p_x = \sum_f q_{if} \cdot p_{xf}$$



LOSS FUNCTION FOR MATRIX FACTORIZATION MODEL

A common loss function is **Mean Square Error** (MSE). It is evaluated only over the seen ratings, not the missing positions:

$$\sum_{(i,x)\in R}(r_{xi} - q_i \cdot p_x)^2$$



Matrix factorization fits the latent factor matrices Q and P by minimizing this loss function:

$$\min_{P,Q} \sum_{(i,x)\in R}(r_{xi} - q_i \cdot p_x)^2$$

FITTING P AND Q USING GRADIENT DESCENT

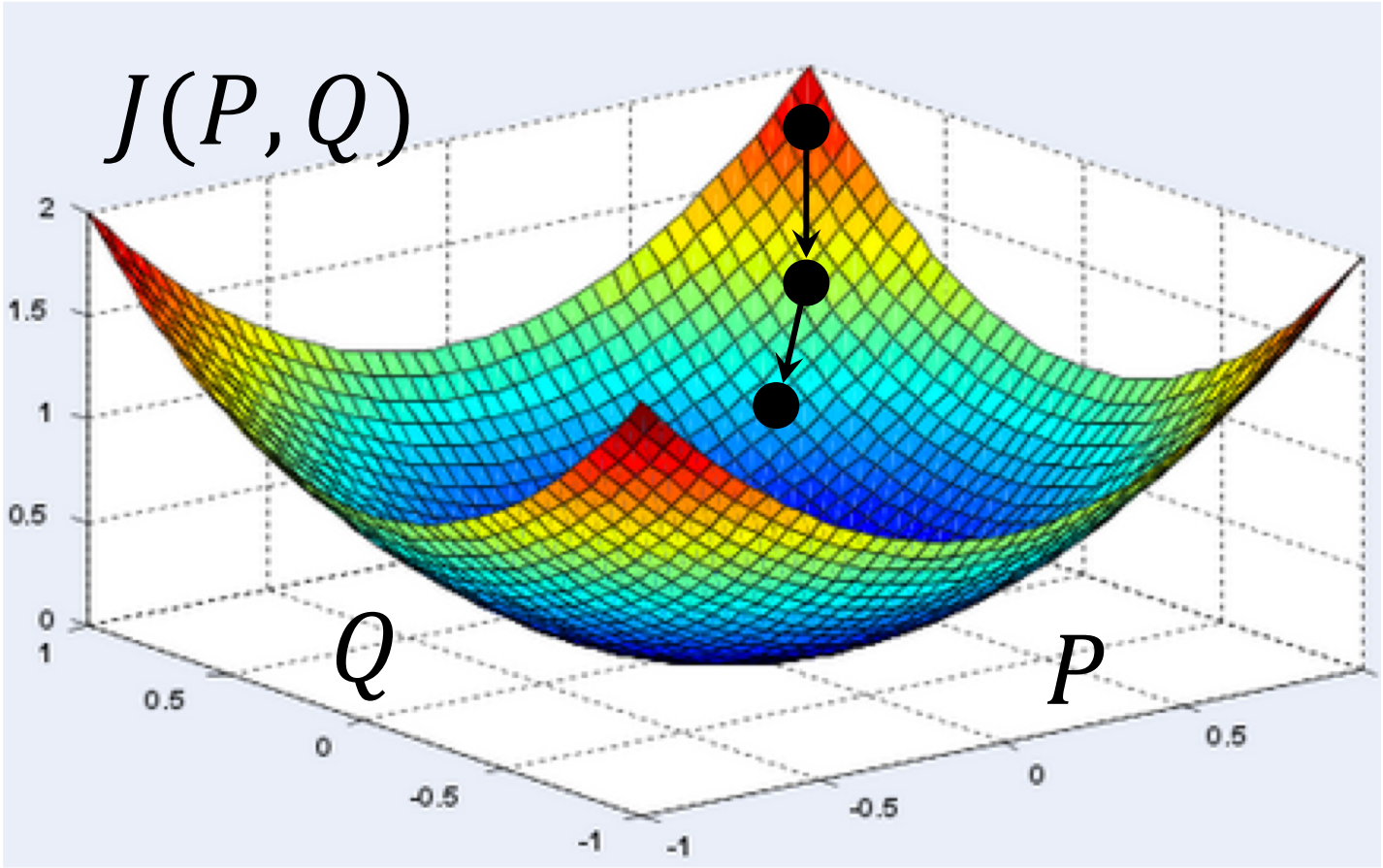
Like in previous lectures, we can do this using gradient descent:

$$P \leftarrow P - \alpha \frac{\delta J(P,Q)}{\delta P}$$

$$Q \leftarrow Q - \alpha \frac{\delta J(P,Q)}{\delta Q}$$

Learning Rate

"Slope": direction of steepest decrease of J



REGULARIZATION

Recall that an overly flexible model (e.g. a model with too many parameters) can lead to **overfitting**

For matrix factorization, a high number of factors can lead to overfitting

We can prevent overfitting using regularization:

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

λ_1, λ_2 … user set regularization parameters

NONNEGATIVE MATRIX FACTORIZATION

Nonnegative Matrix Factorization (NMF) uses the exact same objective, but adds the constraints that all the entries or P and Q must be nonnegative.

Optional

NONNEGATIVE MATRIX FACTORIZATION

Nonnegative Matrix Factorization (NMF) uses the exact same objective, but adds the constraints that all the entries or P and Q must be nonnegative.

This can be done using **projected gradient descent**: this is like regular gradient descent, but has a "projection" function *Proj* which replaces all negative values in P and Q by 0:

$$\mathrm{P} \leftarrow Proj(P - \alpha \frac{\delta J(P,Q)}{\delta P})$$

$$Q \leftarrow Proj(Q - \alpha \frac{\delta J(P,Q)}{\delta Q})$$

Lin CJ. Projected gradient methods for nonnegative matrix factorization. Neural computation. 2007 Oct;19(10):2756-79.

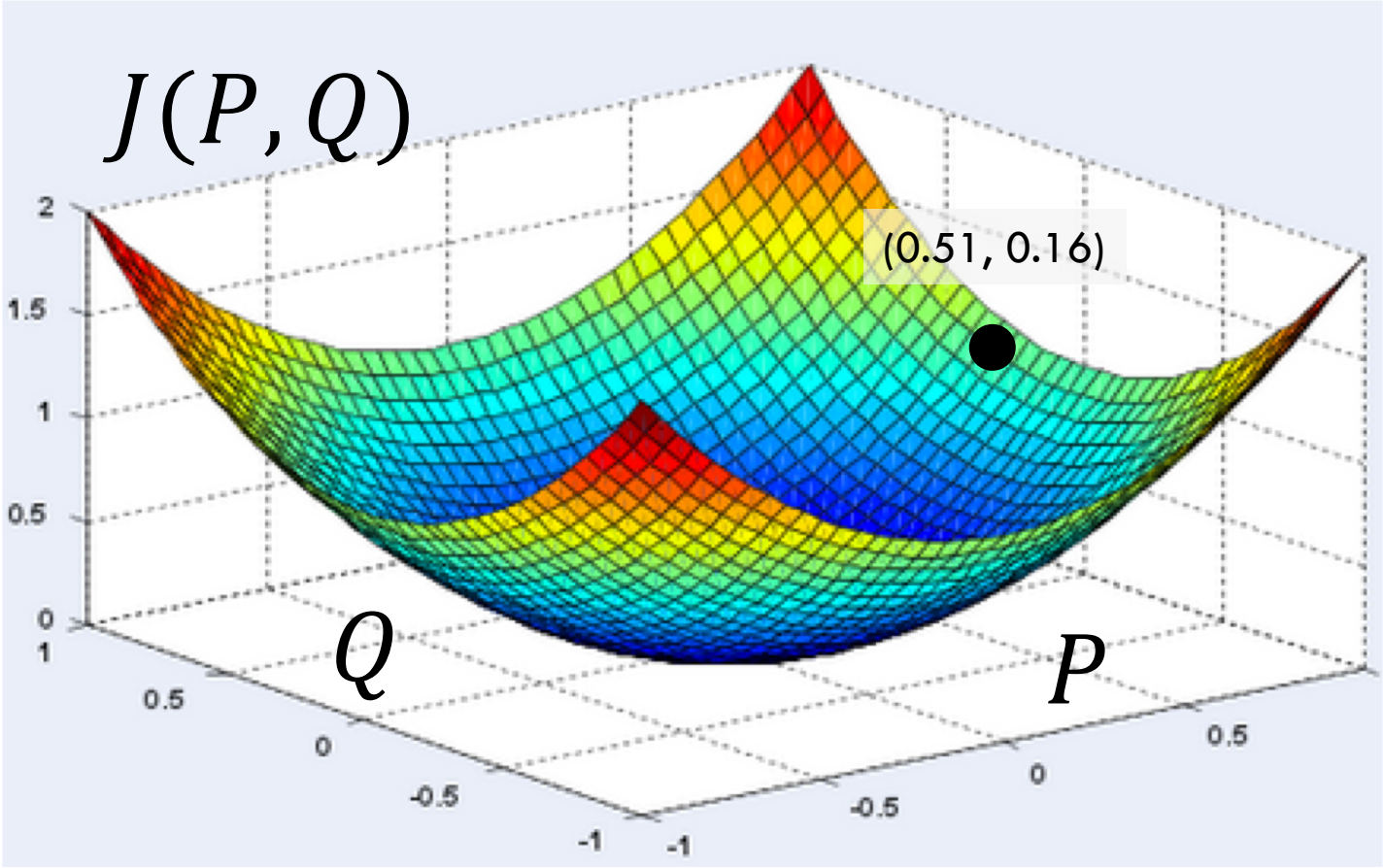
Optional

FITTING NONNEGATIVE MATRIX FACTORIZATION

Projected Gradient Descent:

$$\mathrm{P} \leftarrow Proj(P - \alpha \frac{\delta J(P,Q)}{\delta P})$$

$$Q \leftarrow Proj(Q - \alpha \frac{\delta J(P,Q)}{\delta Q})$$



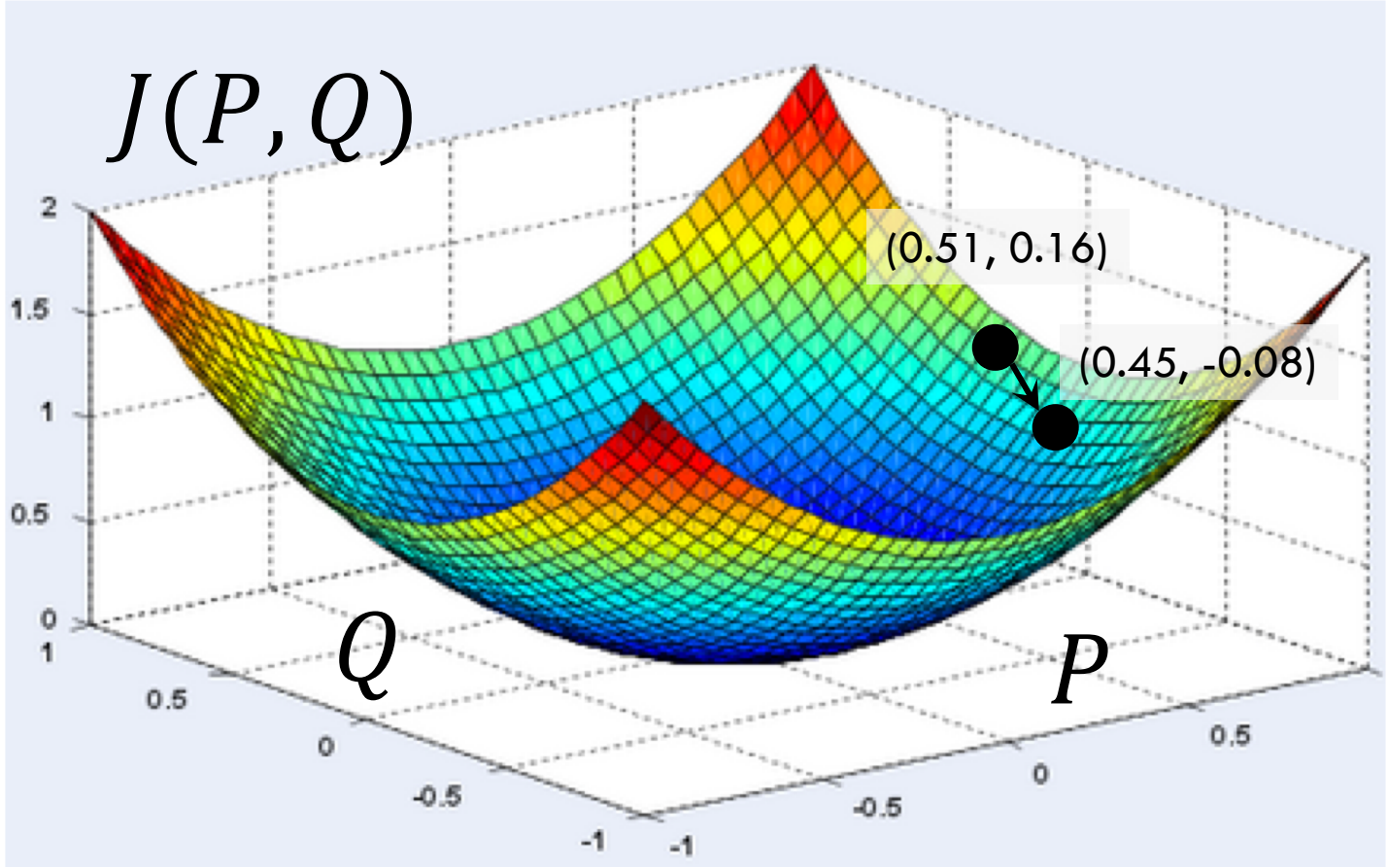
Optional

FITTING NONNEGATIVE MATRIX FACTORIZATION

Projected Gradient Descent:

$$P \leftarrow Proj(P - \alpha \frac{\delta J(P,Q)}{\delta P})$$

$$Q \leftarrow Proj(Q - \alpha \frac{\delta J(P,Q)}{\delta Q})$$



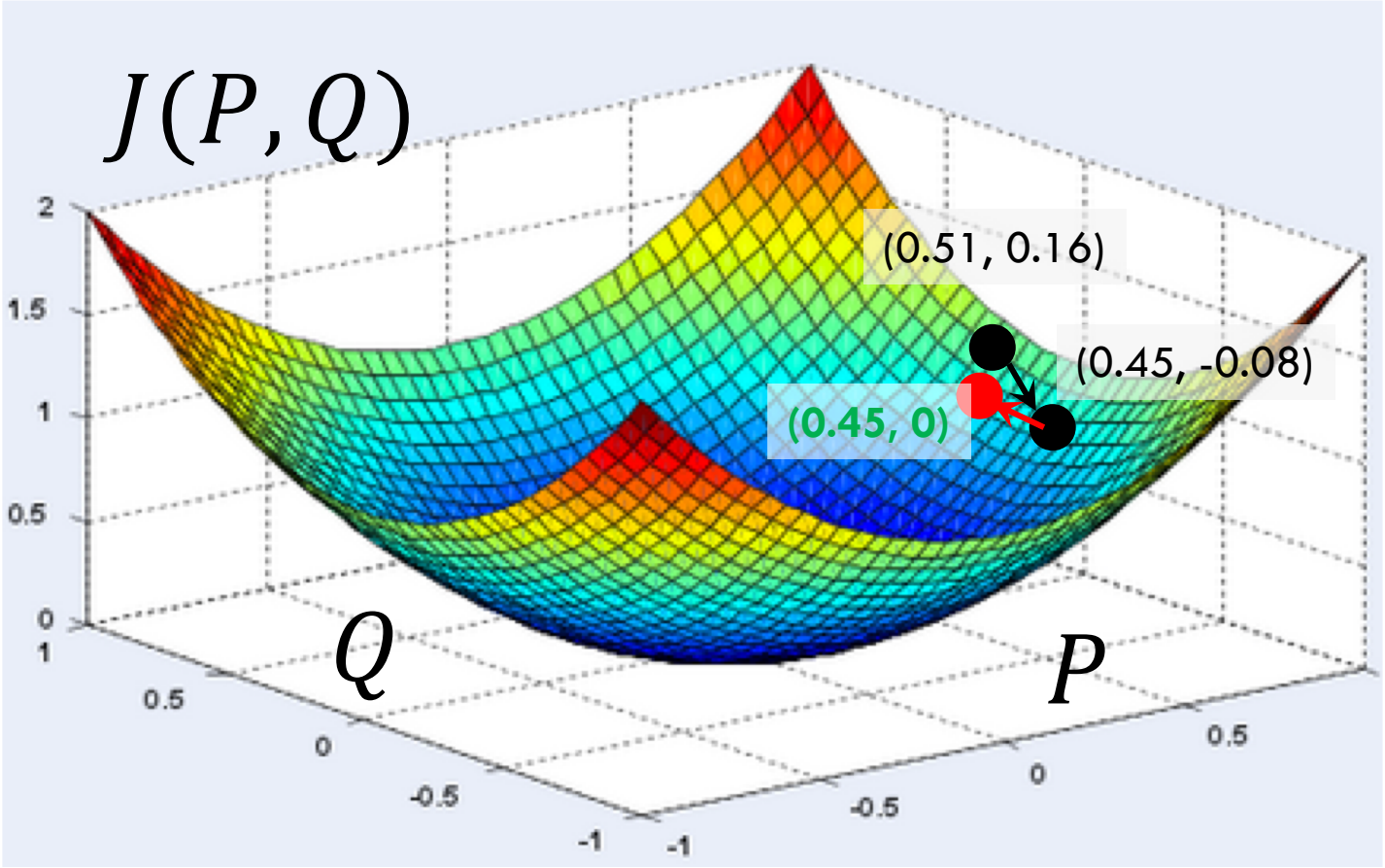
Optional

FITTING NONNEGATIVE MATRIX FACTORIZATION

Projected Gradient Descent:

$$P \leftarrow Proj(P - \alpha \frac{\delta J(P,Q)}{\delta P})$$

$$Q \leftarrow Proj(Q - \alpha \frac{\delta J(P,Q)}{\delta Q})$$

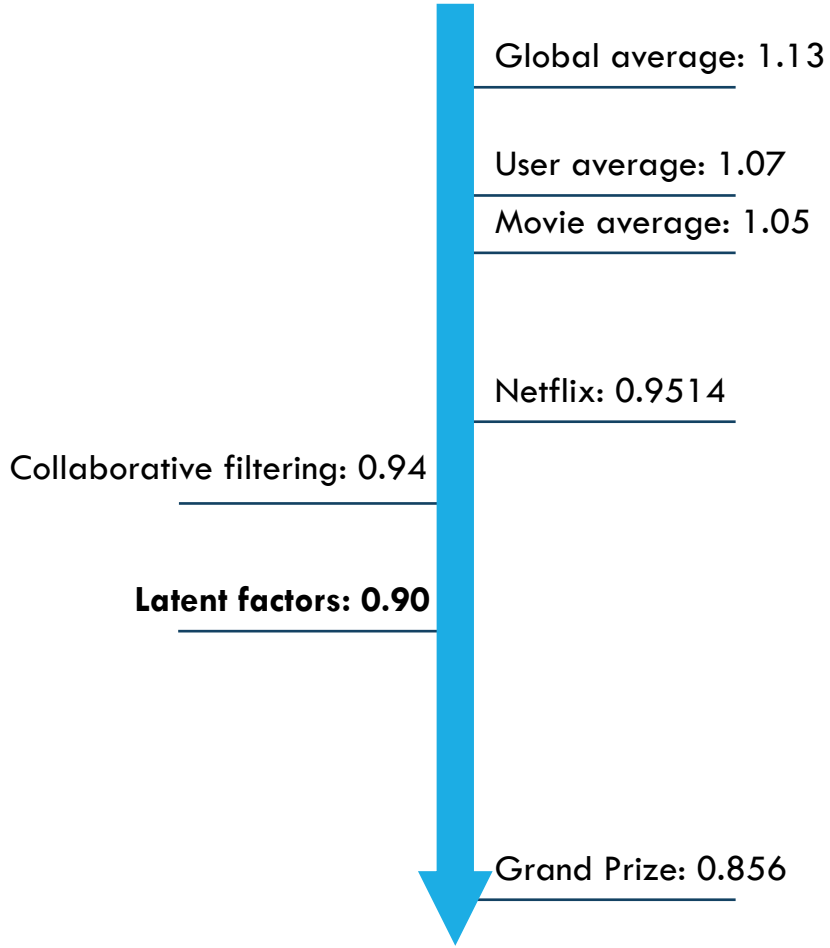


INTERPRETABILITY OF NONNEGATIVE MATRIX FACTORIZATION

Interpretability: like matrix factorization, P and Q can be interpreted as "factors", but now the factors are always nonnegative. This can make them easier to understand in many cases.

Sparsity: NMF tends to produce sparse P and Q, i.e. containing exact zeroes. This can be also provide benefits for interpretability.

PERFORMANCE COMPARISON (NETFLIX CHALLENGE)



LATENT FACTOR MODEL WITH BIASES

Recall our earlier discussion: some users tend to give higher ratings; similarly, some movies tend to receive higher ratings

To capture this, we add **biases** for each user and movie to the model:

$$r_{xi} = \mu + b_x + b_i + q_i \cdot p_x$$

- μ: Overall mean rating
- b_x: Bias for user *x*
- b_i: Bias for movie *i*
- $q_i \cdot p_x$: User-Movie interaction

LATENT FACTOR MODEL WITH BIASES

We can fit this model similarly to before using gradient descent, but now including the bias terms:

$$\min_{Q,P} \sum_{(x,i)\in R} \left(r_{xi} - (\mu + b_x + b_i + q_i\, p_x)\right)^2$$

User-movie interaction

LATENT FACTOR MODEL WITH BIASES

We can fit this model similarly to before using gradient descent, but now including the bias terms:

$$\min_{Q,P} \sum_{(x,i)\in R} \left(r_{xi} - (\mu + b_x + b_i + q_i\, p_x)\right)^2$$

User-movie interaction

Q: Will the optimal solution for the latent factors be affected by any of the following preprocessing steps?

1. Centering the rows (i.e. shifting each row to make its mean 0)
2. Centering the columns (i.e. shifting each column to make its mean 0)
3. Centering both the rows and columns

LATENT FACTOR MODEL WITH BIASES

We can fit this model similarly to before using gradient descent, but now including the bias terms:

$$\min_{Q,P} \sum_{(x,i)\in R} \left(r_{xi} - (\mu + b_x + b_i + q_i\, p_x)\right)^2$$

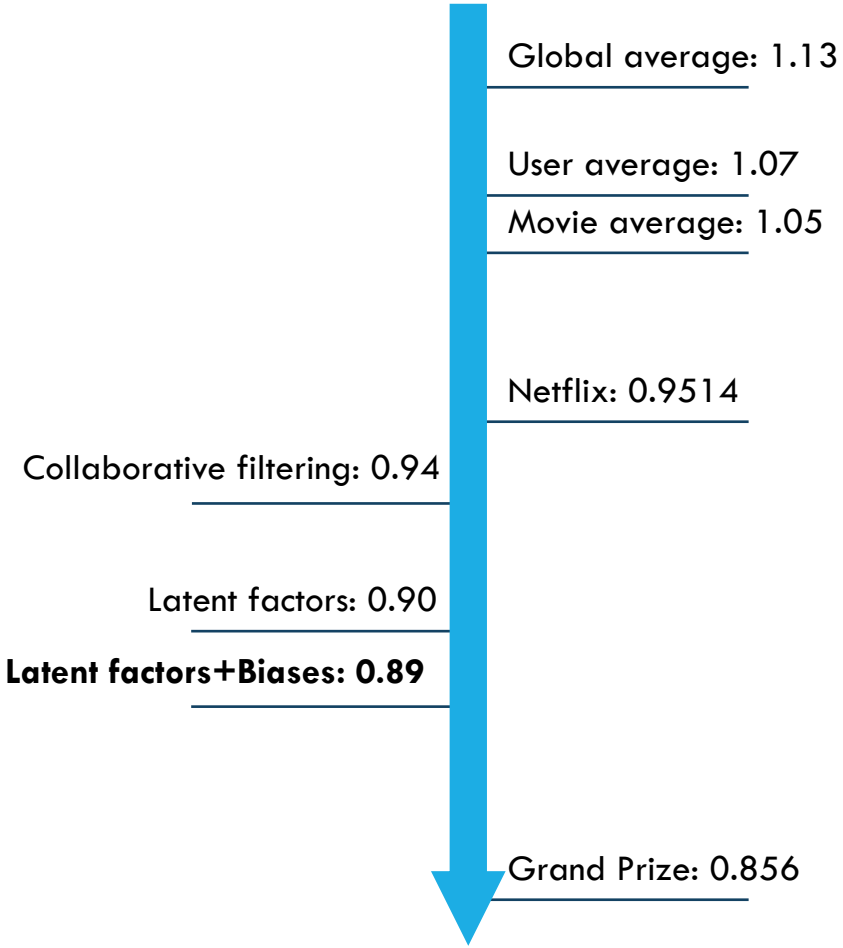
User-movie interaction

Q: Will the optimal solution for the latent factors be affected by any of the following preprocessing steps?

1. Centering the rows (i.e. shifting each row to make its mean 0)
2. Centering the columns (i.e. shifting each column to make its mean 0)
3. Centering both the rows and columns

A: No to all. Any shifts to a row or column are "absorbed" by the bias terms, and have no effect on the latent factors.

PERFORMANCE COMPARISON (NETFLIX CHALLENGE)



TEMPORALLY VARYING BIAS MODEL

A user or movie's baseline ('bias') level may change over time

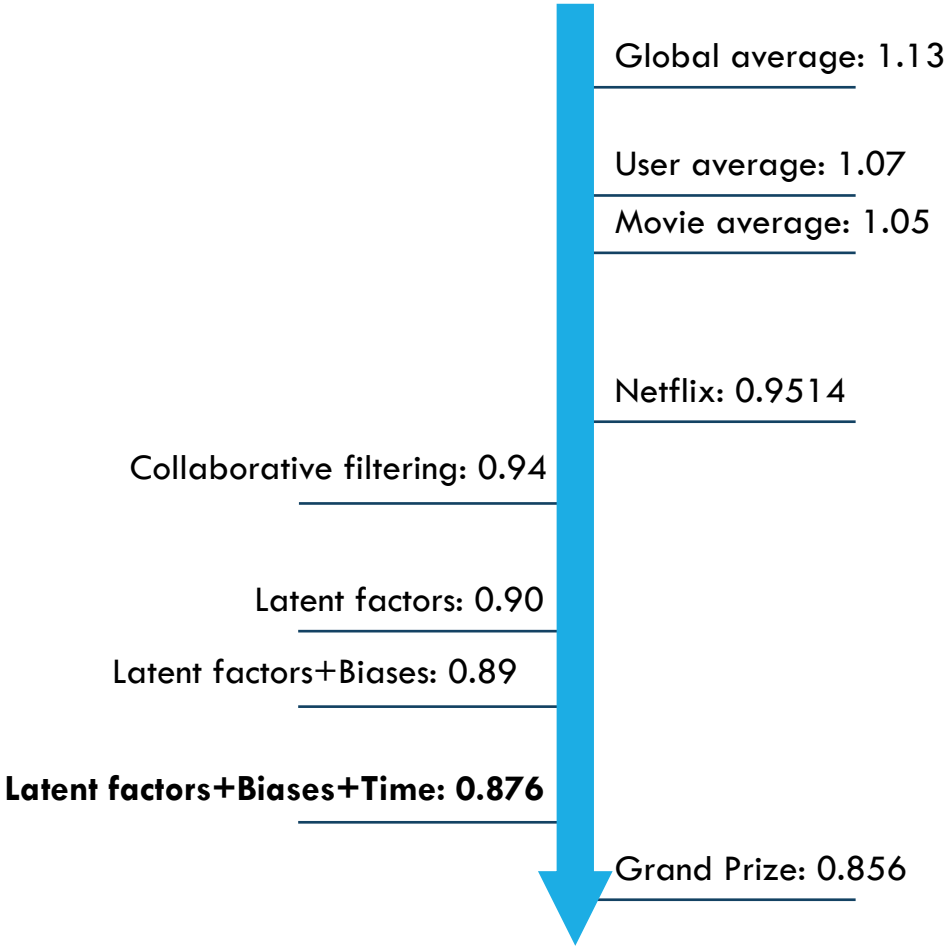
- E.g. early ratings for a product are unfavorable, but gradually improve as the company fixes the issues with the product
- Models which allow change over time are more flexible and often more accurate in practice

Original model: $r_{xi} = \mu + b_x + b_i + q_i \cdot p_x$

Add time dependence to biases: $r_{xi} = \mu + b_x(t) + b_i(t) + q_i \cdot p_x$

For example, let time be in granularity of weeks, i.e. t=1 is week 1, t=2 is week 2, etc. Then instead of fitting a single bias level b_x for user x, we are fitting a separate bias level for this user for each week, i.e. $b_x(1), b_x(2), \ldots$

PERFORMANCE COMPARISON (NETFLIX CHALLENGE)



PRACTICAL CONSIDERATIONS: EVALUATION

The methods we have discussed so far aim at improving RMSE.

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

However, is this too narrow, and does it actually provide benefits for practical applications?

Goodhart's Law [1]: "When a measure becomes a target, it ceases to be a good measure."

Charles Goodhart

Strathern, Marilyn (1997). "'Improving ratings': audit in the British University system". European Review.

DRAWBACKS OF NARROW FOCUS ON RMSE

Ignores prediction diversity

RMSE does not focus on the top scoring items, but the top items are key for recommendation

RMSE only sums over the *present* (user, item) pairs, not missing ones.

- It only aims to predict scores accurately, but not which items will be rated by a user. In practice, users do not rate products randomly [1] (i.e. "Not Missing At Random"), and the information about which products a user rated is often important to exploit.
- This requires the use of implicit rather than explicit feedback.

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

[1] Steck, Harald. "Training and testing of recommender systems on data missing not at random." *KDD 2010*.

PRACTICAL CONSIDERATIONS: EVALUATION

Reassuringly, [1] finds that small improvements (e.g. 1%) in RMSE can still translate to significant improvements in the quality of the top K movies.

[2] provides a somewhat contrasting view: [2] finds that implicit feedback approaches perform better at finding relevant items than the best Netflix Challenge models (see paper for details).

Main conclusion: in general, make sure your choice of metric is as closely aligned as possible with the practical goals of the application

[1] Yehuda Koren (2007-12-18). "How useful is a lower RMSE?".
[2] Steck, Harald. "Training and testing of recommender systems on data missing not at random." *KDD 2010*.

Optional

COMBINING DEEP LEARNING AND CF: NEURAL CF

Standard matrix factorization predicts a rating r_{xi} as the **dot product** of the factor vectors q_i and p_x

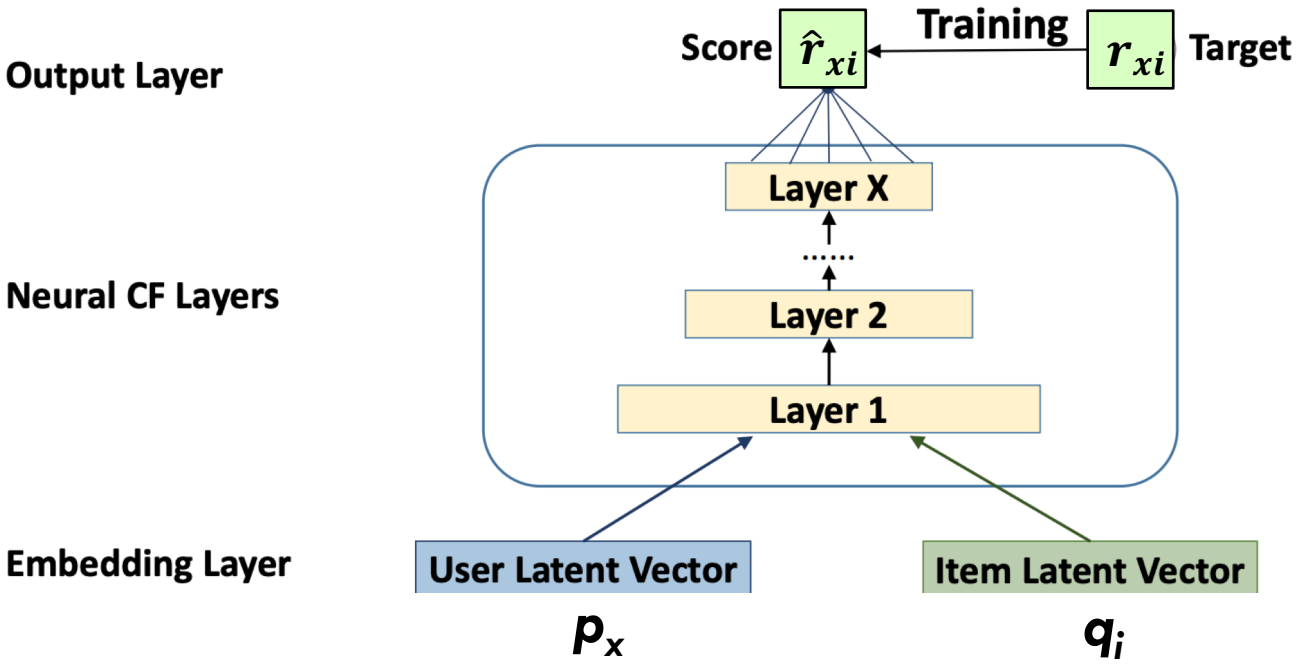
Thus, its predictions are **linear** in the factor vectors

To make this model more flexible and nonlinear, **Neural CF** incorporates deep learning models with multiple nonlinear layers.

He X, Liao L, Zhang H, Nie L, Hu X, Chua TS. Neural collaborative filtering. WWW 2017 Apr 3 (pp. 173-182).

Optional

COMBINING DEEP LEARNING AND CF: NEURAL CF

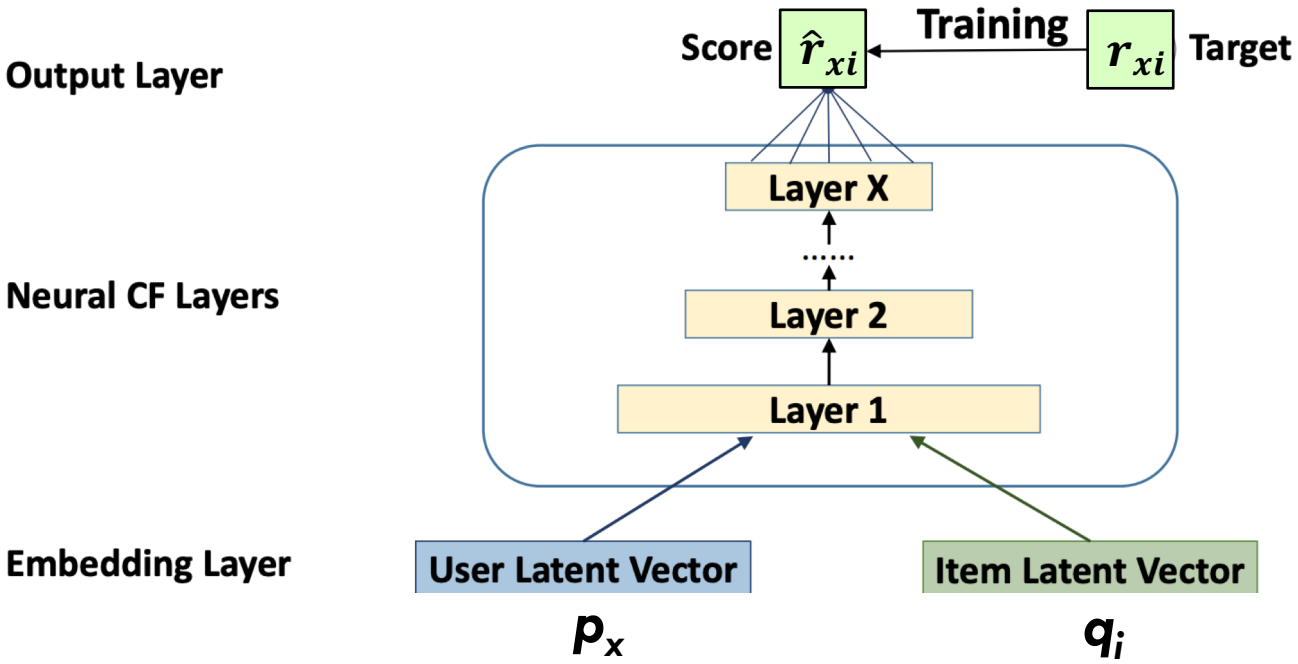


Instead of taking the dot product of p_x and q_i, here we use them as input into a series of nonlinear deep learning layers

Like regular CF, the output is our model's prediction for the rating score.

Optional

COMBINING DEEP LEARNING AND CF: NEURAL CF



The exact design of the inner Neural CF layers is flexible, and different designs are considered in the paper.

E.g. "Generalized Matrix Factorization": $\hat{r}_{xi} = \sigma(\boldsymbol{w}^T(\boldsymbol{p}_x \odot \boldsymbol{q}_i))$

Activation — Weight vector — Elementwise product