Datamining – CSE2525

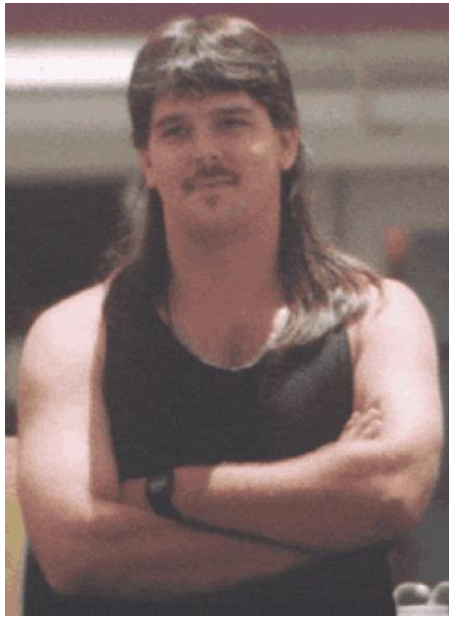
Recommender systems + challenge

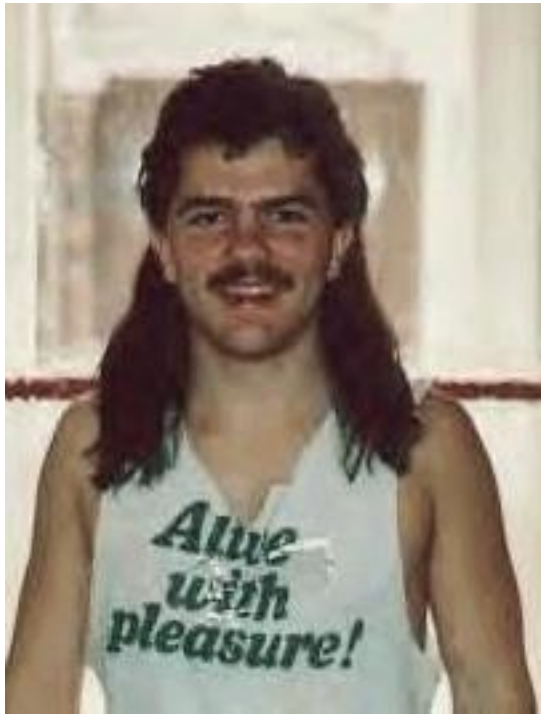
December 4, 2020

The plan

- Recommender systems
  - Content-based recommendation
  - Collaborative filtering
  - Nearest neighbor
  - Biases
  - *Latent factors*
- Challenge
  - Introduction
  - Expectations
  - Logistics (Brightspace)

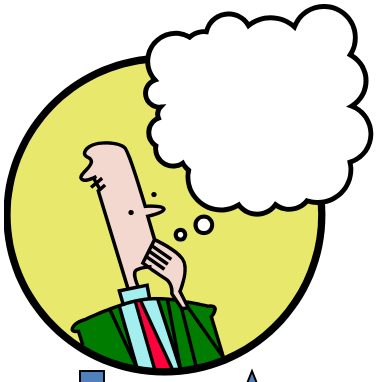
Example: Recommender Systems





- **Customer X**
  - Buys Metallica CD
  - Buys Iron Maiden CD
- **Customer Y**
  - Does search on Metallica
  - Recommender system suggests Iron Maiden from data collected about customer **X**

Recommendations



Search

Recommendations

Items

Products, web sites, blogs, news items, …

Examples:

amazon.com

Spotify

NETFLIX

cool blue

bol.com

last.fm

STEAM

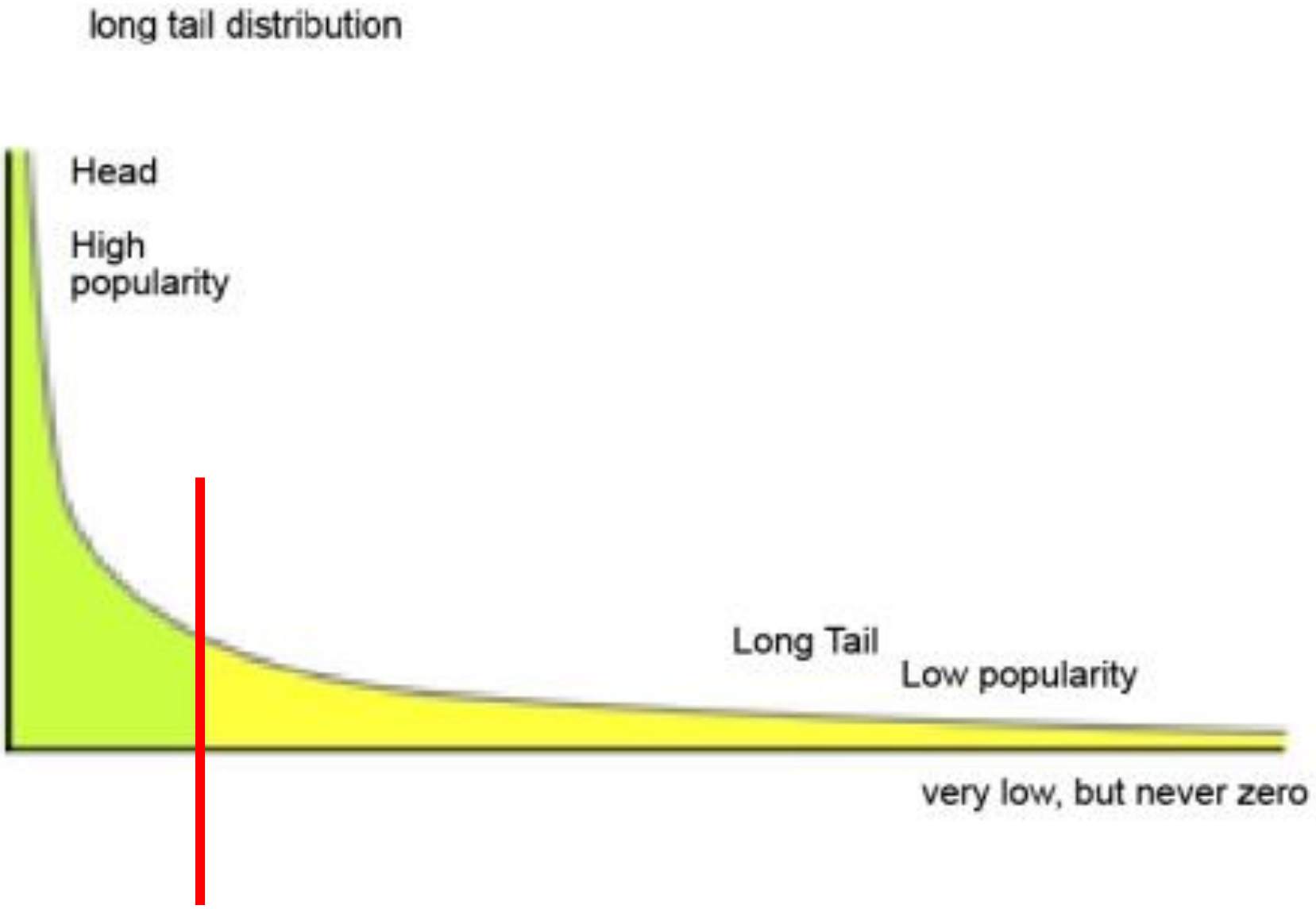
YouTube

XBOX LIVE

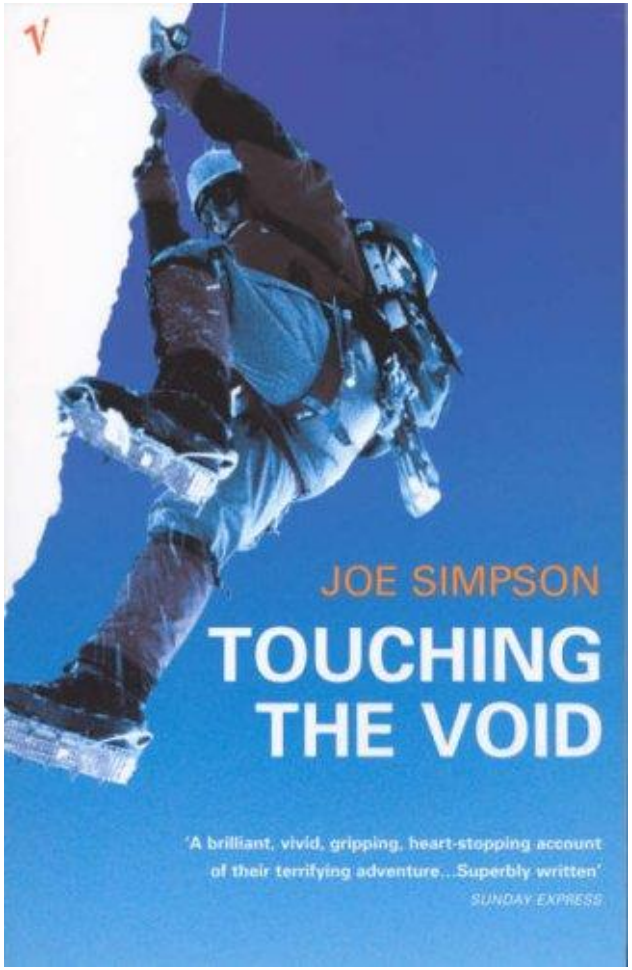
From Scarcity to Abundance

- **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**
  - Recommendation engines
  - From forgotten literature to best-seller in 10 years

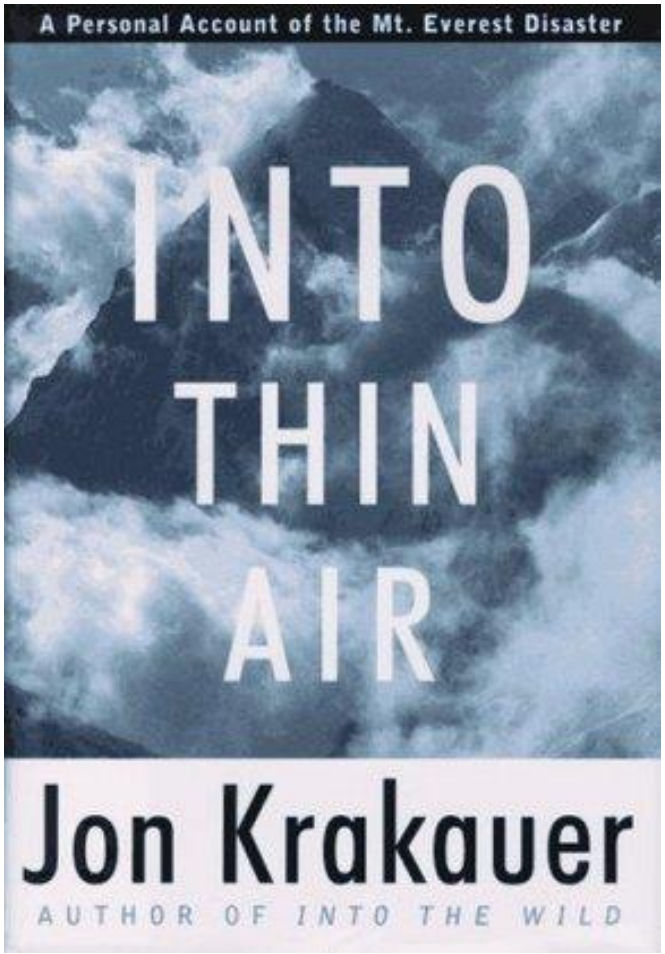
From Scarcity to Abundance



Riding the long tail to success



1988



1997

Types of Recommendations

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

Formal Model

- X = set of **Customers**
- S = set of **Items**

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

Utility Matrix

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

Key Problems

- **(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

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?

(2) Extrapolating Utilities

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

- **Three approaches to recommender systems:**
  - Content-based: examine properties of items
  - Collaborative filtering: item-item or user-user similarity
    - Nearest neighbor query
    - Latent factor models
  - Hybrids: 1 + 2 + demography + knowledge based

CONTENT-BASED RECOMMENDER SYSTEMS

Content-based Recommendations

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

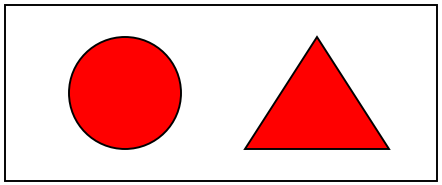
Example:

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

Plan of Action

likes

Item profiles

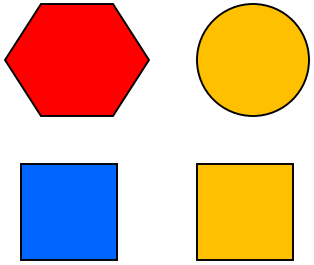


build

Red
Circles
Triangles

User profile

match



recommend

Item Profiles

- For each item, create an **item profile**
- **Profile is a set (vector) of features**
  - **Movies:** author, title, actor, director,…
  - **Text:** Set of "important" words in document
- **How to pick important features?**
  - Usual heuristic from text mining is **TF-IDF** (Term frequency * Inverse Doc Frequency)
    - **Term … Feature**
    - **Document … Item**

User Profiles and Prediction

- **User profile possibilities:**
  - Weighted average of rated item profiles
  - **Variation:** weight by difference from average rating for item
  - …
- **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

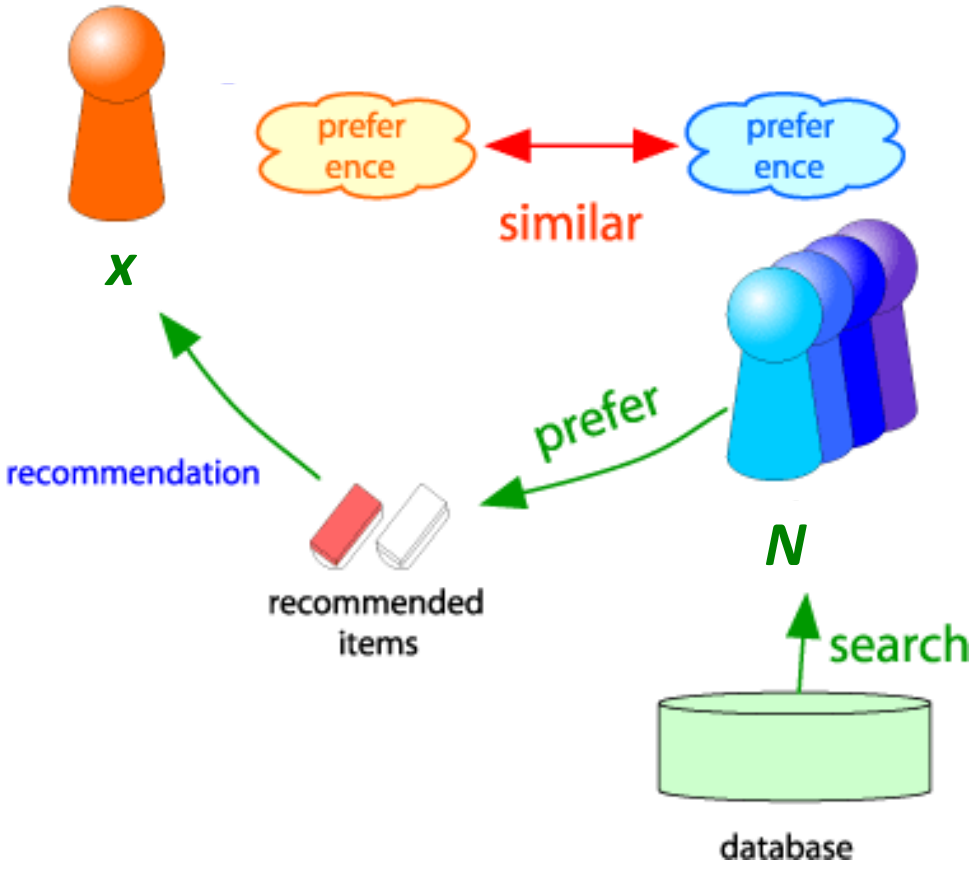
Cons: Content-based Approach

- **–: 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**

COLLABORATIVE FILTERING: NEAREST NEIGHBOR SYSTEM

Collaborative Filtering

- 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

- Let r_x be the vector of user x’s ratings

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

Similarity measures (1)

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

r_x, r_y as sets:

$r_x = \{1, 4, 5\}$

$r_y = \{1, 3, 4\}$

Similarity measures (2)

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

r_x, r_y as points/vectors

$\boldsymbol{r_x} = \{2, 0, 0, 1, 3\}$

$\boldsymbol{r_y} = \{1, 0, 2, 3, 0\}$

Similarity measures (3)

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

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

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

r_x, r_y … avg. rating of **x**, **y**

Similarity Metric

	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*)**
- **Jaccard similarity:** 1/5 < 2/4
- **Cosine similarity:** 0.386 > 0.322
  - Considers missing ratings as "negative"

Similarity Metric

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

- **Cosine similarity:** 0.386 **>** 0.322
  - Considers missing ratings as "negative"
  - **Solution: subtract the (row) mean**

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

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

Notice cosine sim. is correlation when data is centered at 0

Rating Predictions

From similarity metric to recommendations:

- Let 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*:**
  - $r_{xi} = \frac{1}{k} \sum_{y \in N} r_{yi}$
  - $r_{xi} = \frac{\sum_{y \in N} s_{xy} \cdot r_{yi}}{\sum_{y \in N} s_{xy}}$
  - Other options?
- **Many other tricks possible…**

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

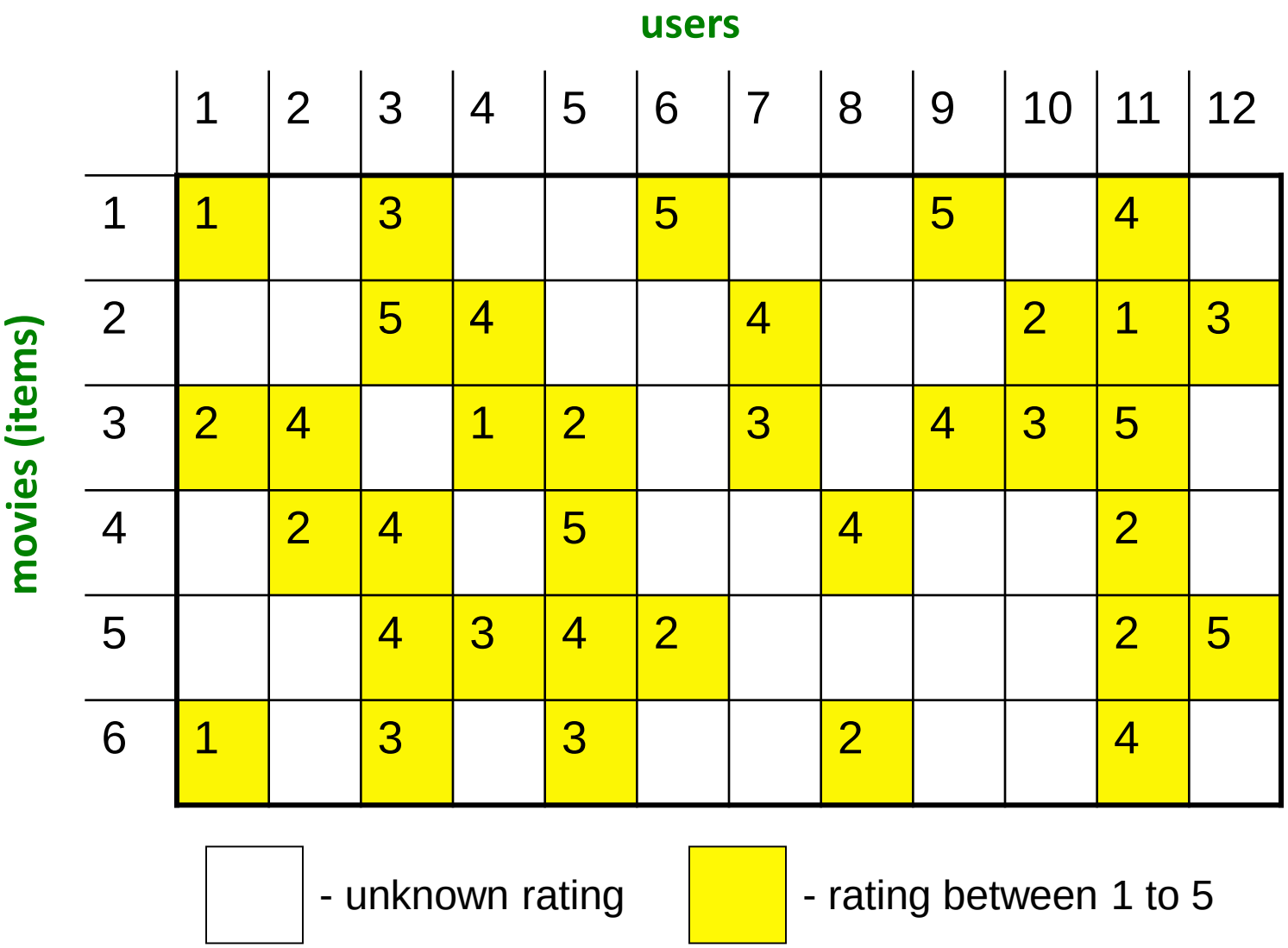
Item-Item Collaborative Filtering

- **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

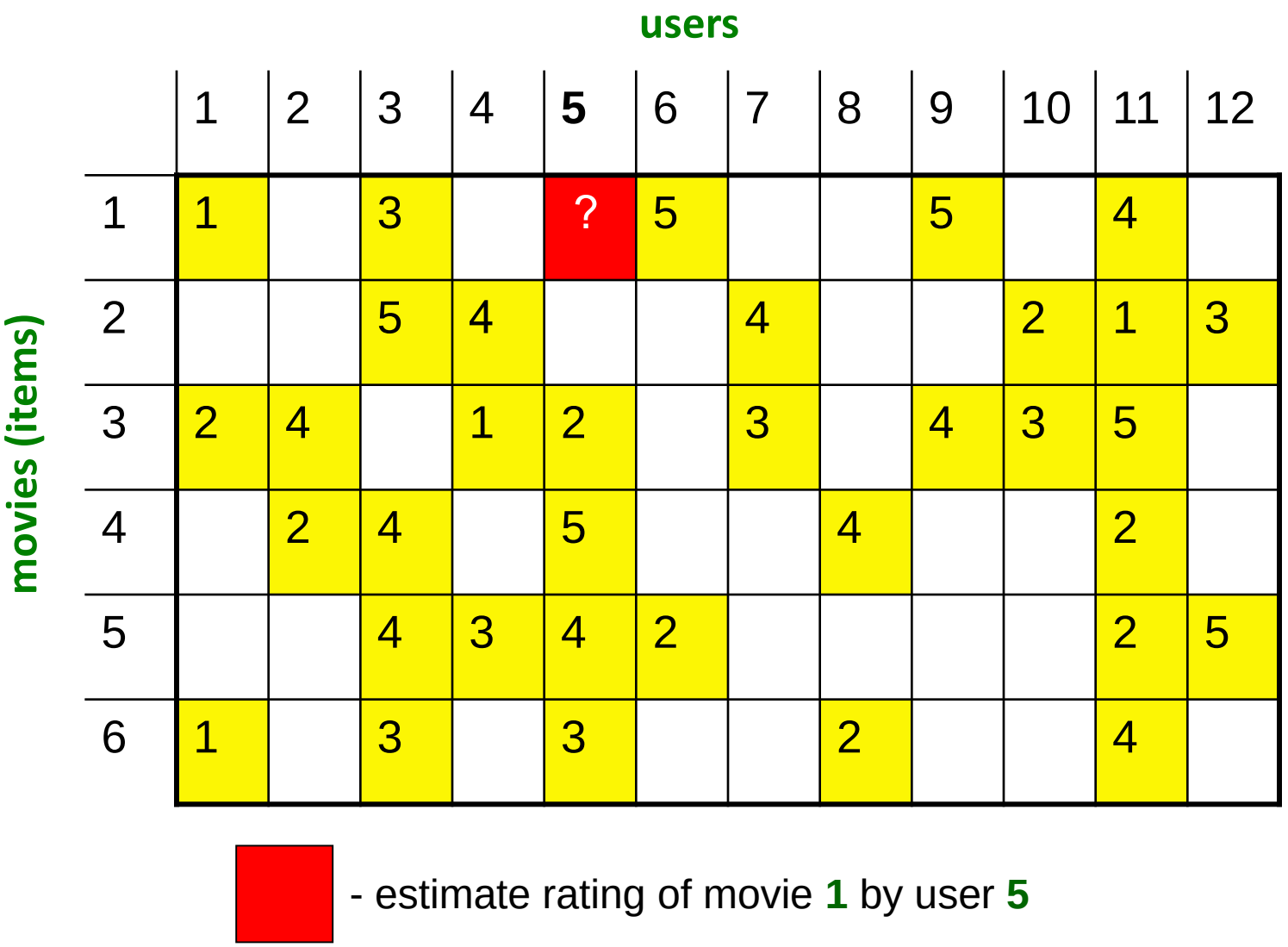
$$r_{xi} = \frac{\sum_{j \in N(i;x)} s_{ij} \cdot r_{xj}}{\sum_{j \in N(i;x)} s_{ij}}$$

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)



Item-Item CF (|N|=2)



	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	

Recipe

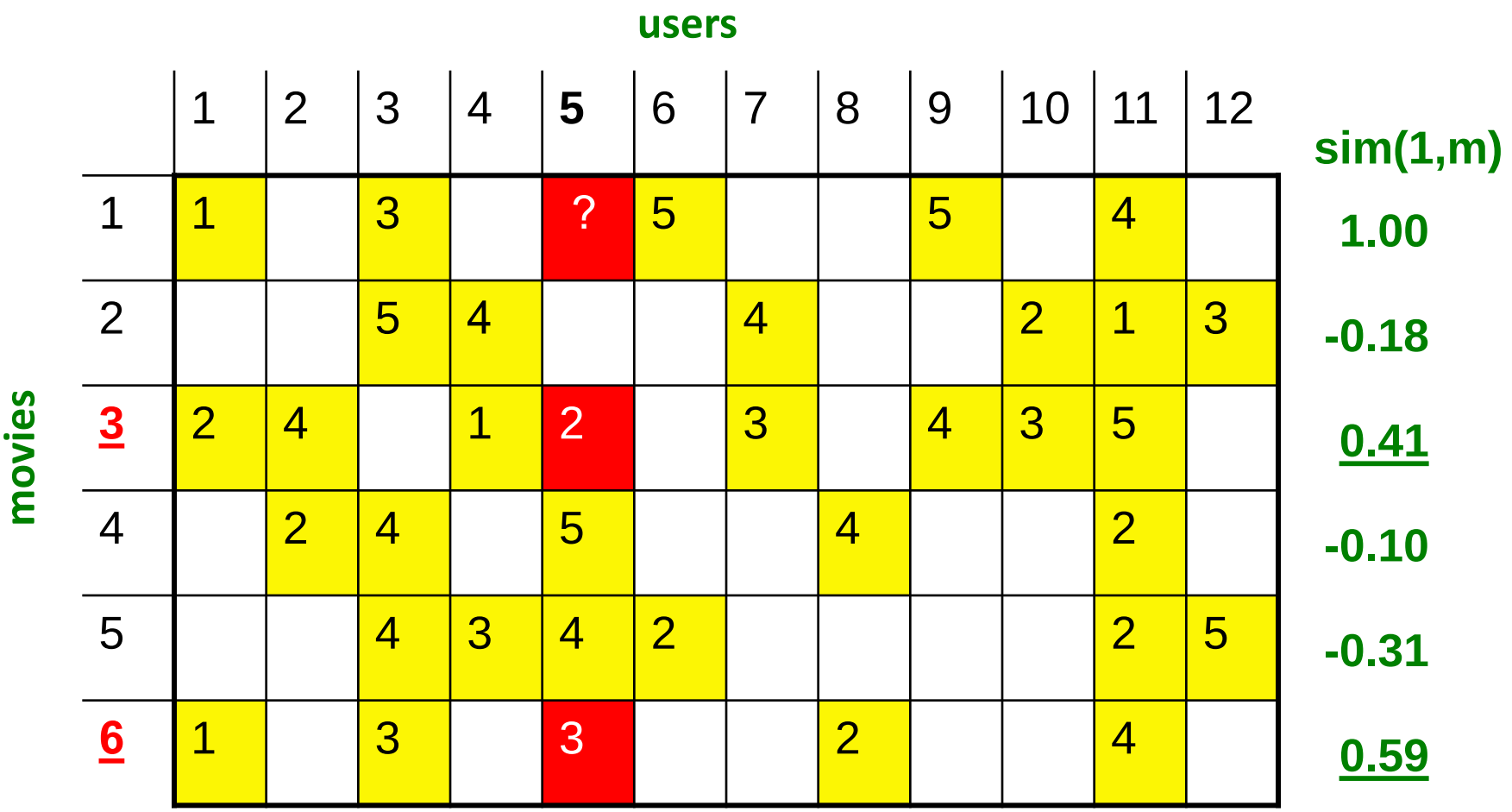
- Use Pearson correlation to calculate similarities

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

- Identify closest 2 neighbors (movies)
- Predict based on neighbors

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

Item-Item CF (|N|=2)



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

Here we use Pearson correlation as similarity:

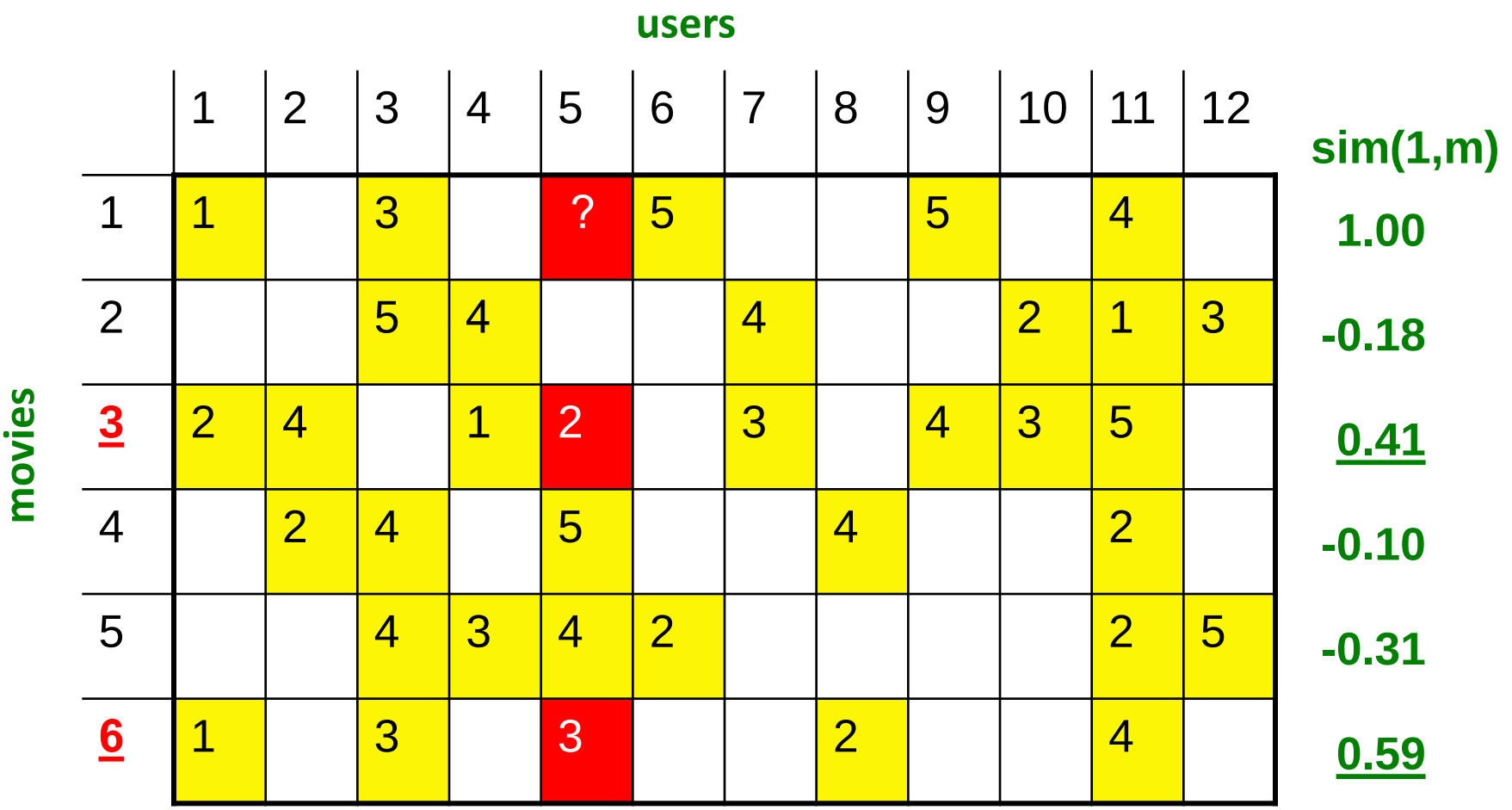
1) Subtract mean rating m_i from each movie ***i***

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

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

2) Compute cosine similarities between rows

Item-Item CF (|N|=2)



Compute similarity weights:

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

Item-Item CF (|N|=2)

users

movies

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

Predict by taking weighted average:

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

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

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/Cons of Collaborative Filtering

- **+ Works for any kind of item**
  - No feature selection 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

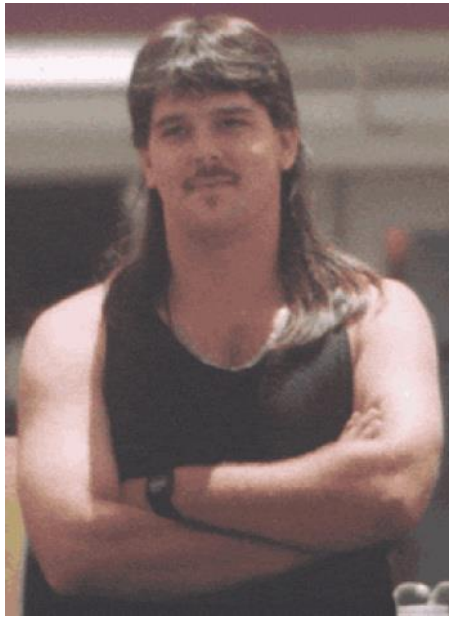
Hybrid Methods

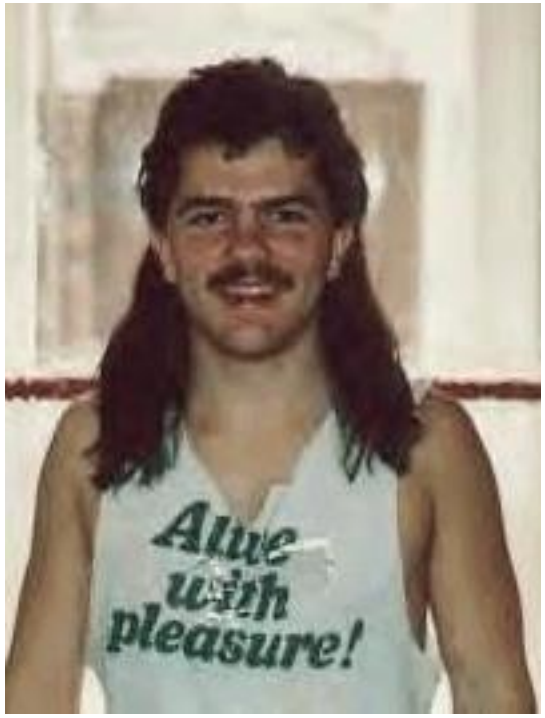
- **Implement two or more different recommenders and combine predictions**
  - Perhaps using a linear model
- **Add content-based methods to collaborative filtering**
  - Item profiles for new item problem
  - Demographics to deal with new user problem

Collaborative Filtering: Complexity

- Expensive step is finding ***k*** most similar customers: **O(|X|)**
- **Too expensive to do at runtime**
  - Could pre-compute
- Naïve pre-computation takes time **O(k ·|X|)**
    - X … set of customers
- **We already know how to do this!**
  - Near-neighbor search in high dimensions (**LSH**)
  - Clustering (another lecture)
  - Dimensionality reduction (another lecture)

Example: Recommender Systems

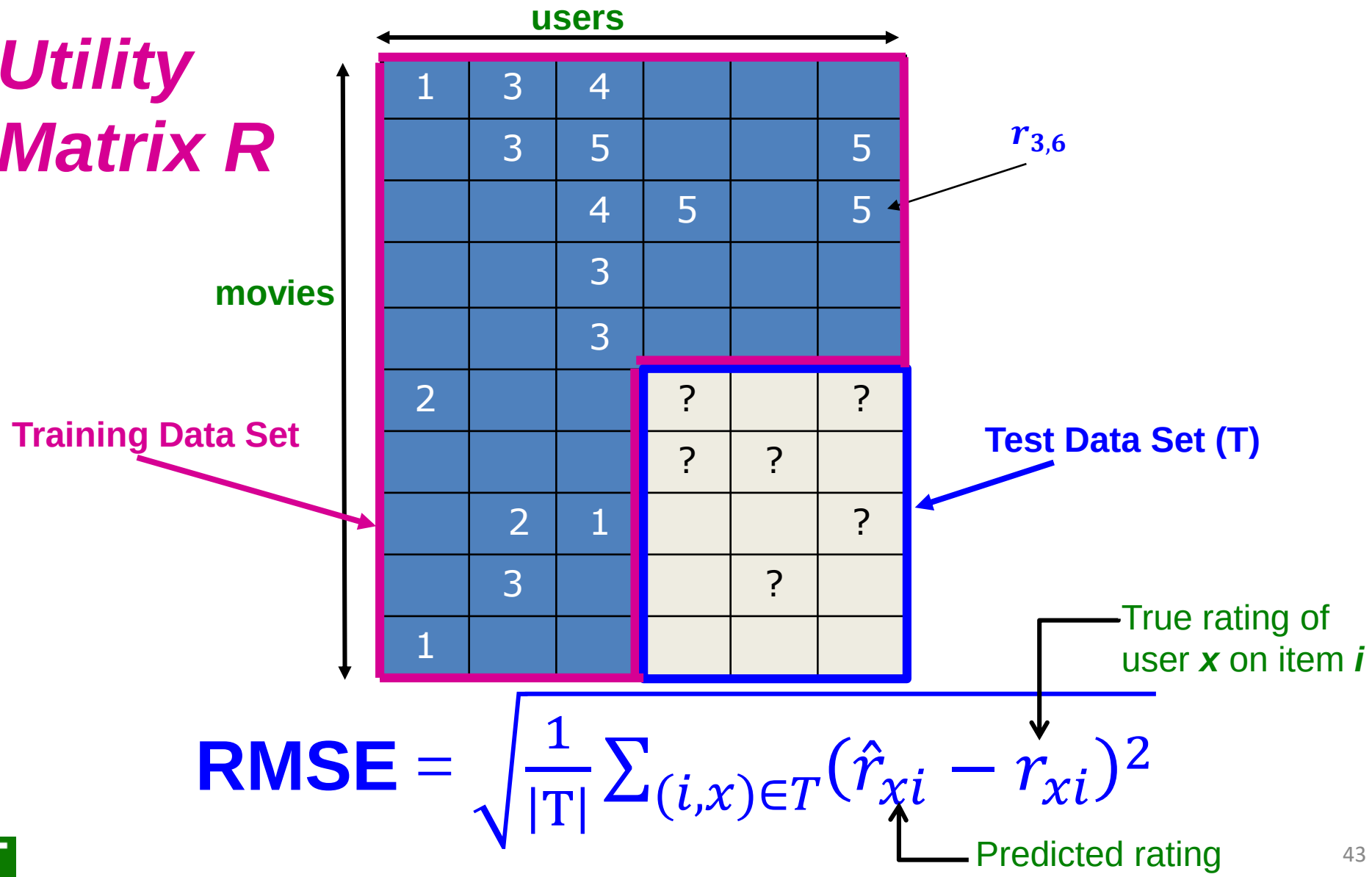




- **Customer X**
  - Buys Metallica CD
  - Buys Iron Maiden CD
- **Customer Y**
  - Does search on Metallica
  - Recommender system suggests Iron Maiden from data collected about customer **X**

EVALUATING RECOMMENDER SYSTEMS

Evaluating systems



Evaluating Predictions

- **Compare predictions with known ratings**
  - **Root-mean-square error** (RMSE)
    - $= \sqrt{\frac{1}{|R|}\sum_{(i,x)\in R}(\hat{r}_{xi} - r_{xi})^2}$
    - where $\boldsymbol{r_{xi}}$ is true rating, $\hat{r}_{\boldsymbol{xi}}$ is the predicted rating of ***x*** on ***i***
  - **Precision at top 10**:
    - % of those in top 10
  - **Rank Correlation**:
    - Spearman's *correlation* between system's and user's complete rankings
- **Another approach: 0/1 model**
  - **Coverage:**
    - Number of items/users for which system can make predictions
  - **Precision:**
    - Accuracy of predictions
  - **Receiver operating characteristic** (ROC)
    - Tradeoff curve between false positives and false negatives

Problems with Error Measures

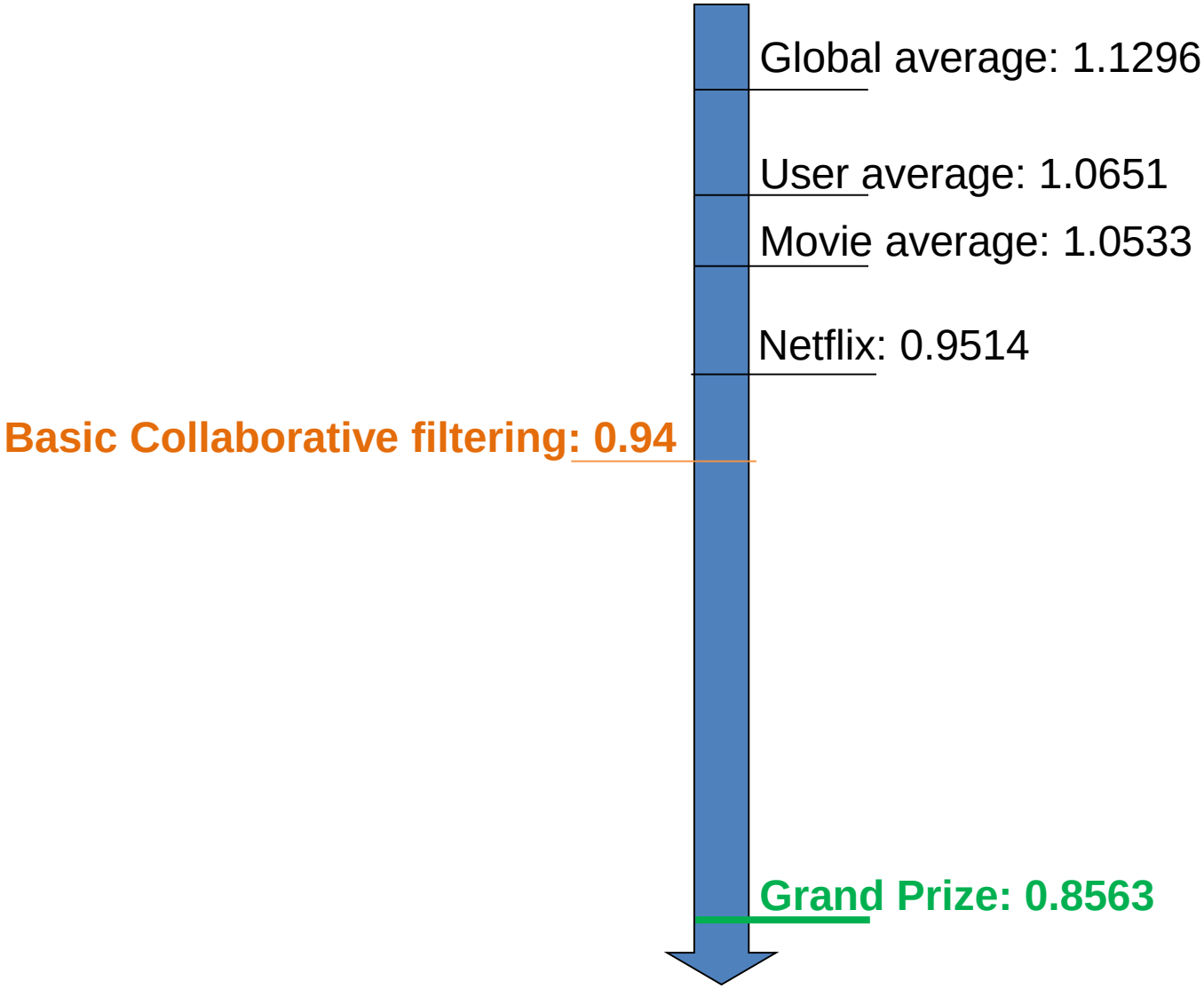
- **Narrow focus on accuracy sometimes misses the point**
  - Prediction Diversity
  - Prediction Context
- **In practice, we care only to predict high ratings:**
  - RMSE might penalize a method that does well for high ratings and badly for others

MODELING (GLOBAL) BIASES

The  Prize

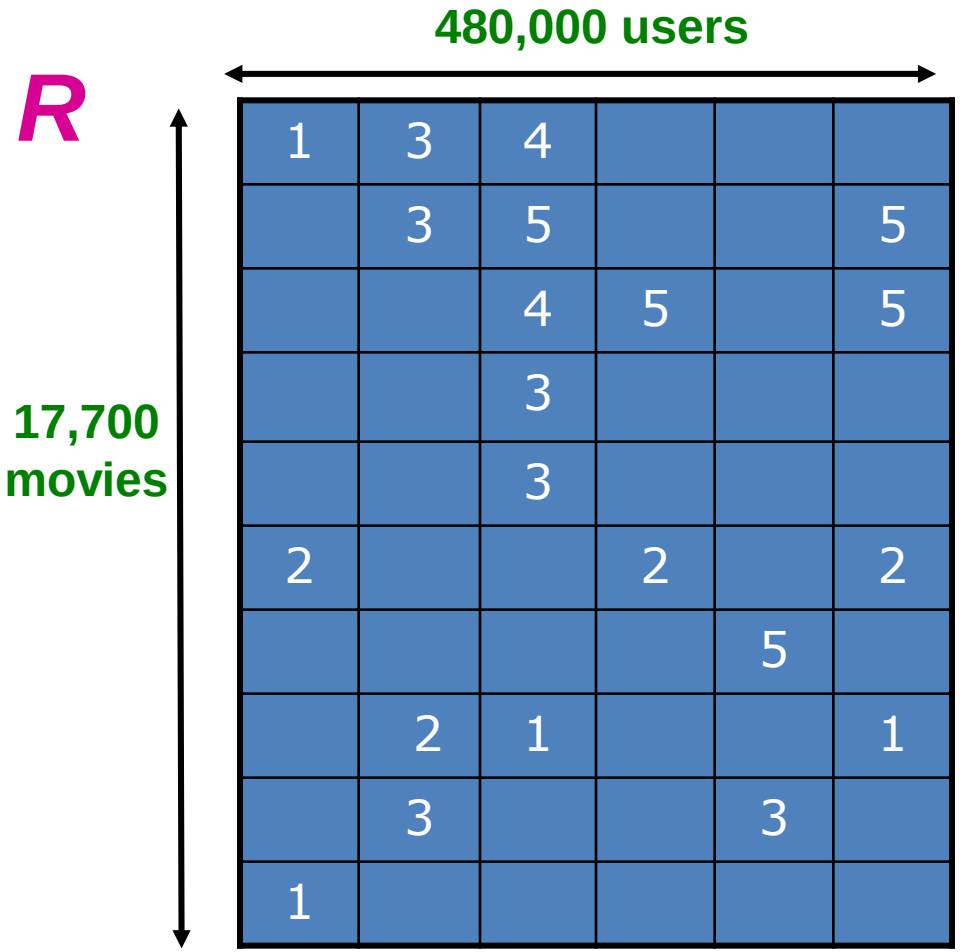
- **Training data**
  - 100 million ratings, 480,000 users, 17,770 movies
  - 6 years of data: 2000-2005
  - 5000 ratings/movie
- **Test data**
  - Last few ratings of each user (2.8 million)
  - **Evaluation criterion:** Root Mean Square Error (RMSE) $=$ $\sqrt{\frac{1}{|R|}\sum_{(i,x)\in R}(\hat{r}_{xi}-r_{xi})^2}$
  - **Netflix's system RMSE: 0.9514**
- **Competition**
  - 2,700+ teams
  - **$1 million** prize for 10% improvement on Netflix

Performance of Various Methods



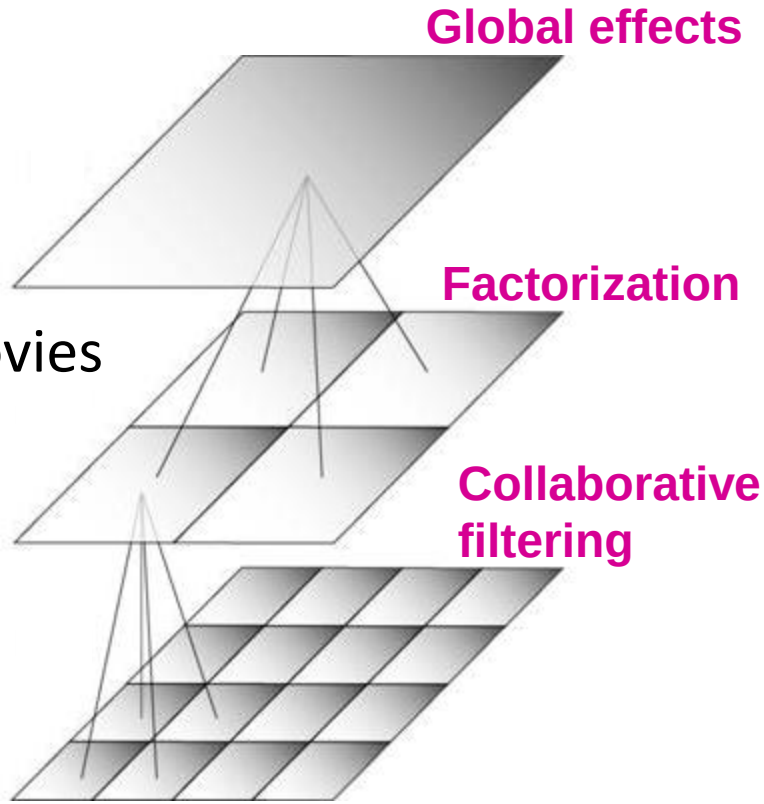
The Netflix Utility Matrix *R*

Matrix R

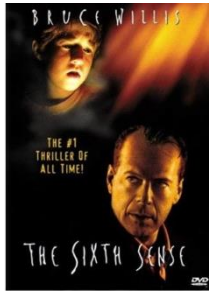


Winner of Netflix Challenge

- **Multi-scale modeling of the data:** Combine top level, "regional" modeling of the data, with a refined, local view:
  - **Global:**
    - Overall deviations of users/movies
  - **Factorization:**
    - Addressing "regional" effects
  - **Collaborative filtering: NN**
    - Extract local patterns



Modeling Local & Global Effects



- **Global: biases**
  - Mean movie rating: **3.7 stars**
  - *The Sixth Sense* is **0.5** stars above avg.
  - Joe rates **0.2** stars below avg.

    ⇒ Baseline estimation:

    ***Joe* will rate *The Sixth Sense* 4 stars**
- **Local neighborhood (CF/NN):**
  - *Joe* didn't like related movie *Signs*
  - **⇒ Final estimate:**

    ***Joe* will rate *The Sixth Sense* 3.8 stars**

CF: Modeling Local & Global Effects

- **In practice we get better estimates if we model deviations:**

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

Before:

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

baseline estimate for r_{xi}

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

μ = overall mean rating
b_x = rating deviation of user x
= (*avg. rating of* user x) – μ
b_i = (*avg. rating of* movie i) – μ

Solution? Problems?

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

baseline estimate for r_{xi}

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

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

Problems/Issues:

1) Similarity measures are "arbitrary"

2) Pairwise similarities neglect interdependencies among users

3) Taking a weighted average can be restricting

Solution: Instead of s_{ij} use w_{ij} that we estimate directly from data

Idea: Interpolation Weights w_{ij}

- Use a **weighted sum** rather than **weighted avg.**:

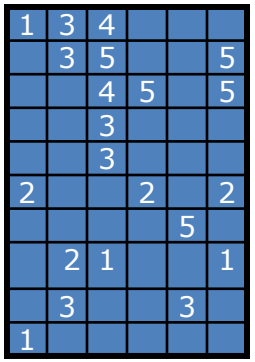
$$\widehat{r_{xi}} = b_{xi} + \sum_{j \in N(i;x)} w_{ij}\left(r_{xj} - b_{xj}\right)$$

- **A few notes:**
  - $\boldsymbol{N(i;x)}$ … set of movies rated by user ***x*** that are similar to movie ***i***
  - $\boldsymbol{w_{ij}}$ is the interpolation weight (some real number)
    - We allow: $\boldsymbol{\sum_{j \in N(i,x)} w_{ij} \neq 1}$
  - $\boldsymbol{w_{ij}}$ models interaction between pairs of movies (it does not depend on user ***x***)

Idea: Interpolation Weights w_{ij}

- $\widehat{r_{xi}} = b_{xi} + \sum_{j \in N(i,x)} w_{ij}\left(r_{xj} - b_{xj}\right)$
- **How to set w_{ij}?**
  - Remember, error metric is: $\sqrt{\frac{1}{|R|}\sum_{(i,x)\in R}(\hat{r}_{xi} - r_{xi})^2}$ or equivalently **SSE:** $\sum_{(i,x)\in R}(\hat{r}_{xi} - r_{xi})^2$
  - Find w_{ij} that minimize **SSE** on **training data!**
    - Models relationships between item ***i*** and its neighbors ***j***
  - w_{ij} can be **learned/estimated** based on ***x*** and all other users that rated ***i***

Recommendations via Optimization



- **Goal:** Make good recommendations
  - Quantify goodness using **RMSE:**
    Lower RMSE ⇒ better recommendations
  - Want to make good recommendations on items that user has not yet seen. Can't really do this!
  - **Let's set build a system such that it works well on known (user, item) ratings**
    And **hope** the system will also predict well the **unknown ratings**

Recommendations via Optimization

- **Idea: Let's set values *w* such that they work well on known (user, item) ratings**
- **How to find such values *w*?**
- **Idea:** Define an objective function and solve the optimization problem

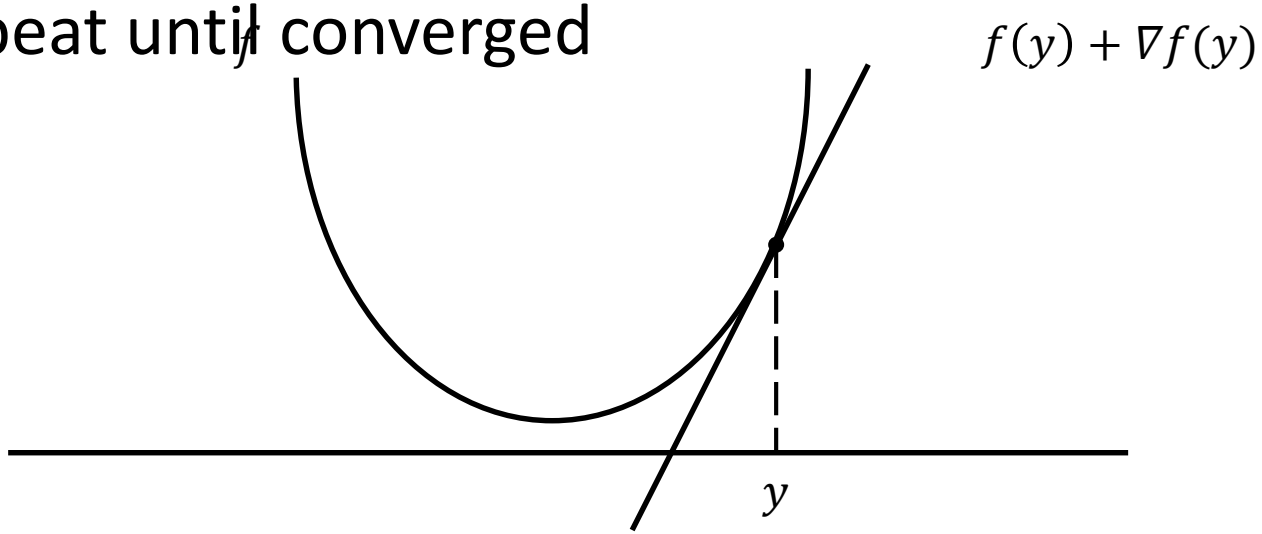
- Find $\boldsymbol{w_{ij}}$ that minimize **SSE** on **training data**!

$$SSE = \sum_{x,i} \left(\underbrace{\left[b_{xi} + \sum_{j \in N(i;x)} w_{ij}\left(r_{xj} - b_{xj}\right) \right]}_{\text{Predicted rating}} - \underbrace{r_{xi}}_{\text{True rating}} \right)^2$$

- Think of $\boldsymbol{w}$ as a matrix of numbers

Detour: Minimizing a function

- **A simple way to minimize a function $f(x)$:**
  - Compute the derivative ∇f
  - Start at some point y and evaluate $\nabla f(y)$
  - Make a step in the reverse direction of the gradient: $y = y - \nabla f(y)$
  - Repeat until converged



Interpolation Weights

- **We have the optimization problem, now what?**
- **Gradient decent:**
  - **Iterate until convergence:** $\boldsymbol{w \leftarrow w - \eta \nabla_w SSE}$ η (eta) learning rate
  - **where $\boldsymbol{\nabla_w SSE}$ is the gradient (derivative evaluated on data):**

$$SSE = \sum_{x,i} \left(\left[b_{xi} + \sum_{j \in N(i;x)} w_{ij}(r_{xj} - b_{xj}) \right] - r_{xi} \right)^2$$

$$\nabla_w SSE = \left[\frac{\partial SSE(w)}{\partial w_{ij}} \right] = 2 \sum_{x,i} \left(\left[b_{xi} + \sum_{k \in N(i;x)} w_{ik}(r_{xk} - b_{xk}) \right] - r_{xi} \right) (r_{xj} - b_{xj})$$

$$\textbf{for } \boldsymbol{j \in \{N(i;x), \forall i, \forall x\}}$$

$$\textbf{else } \frac{\partial SSE(w)}{\partial w_{ij}} = \mathbf{0}$$

  - **Note:** We fix movie $\boldsymbol{i}$, go over all $\boldsymbol{r_{xi}}$, for every movie $\boldsymbol{j \in N(i;x)}$, we compute $\frac{\partial SSE(w)}{\partial w_{ij}}$

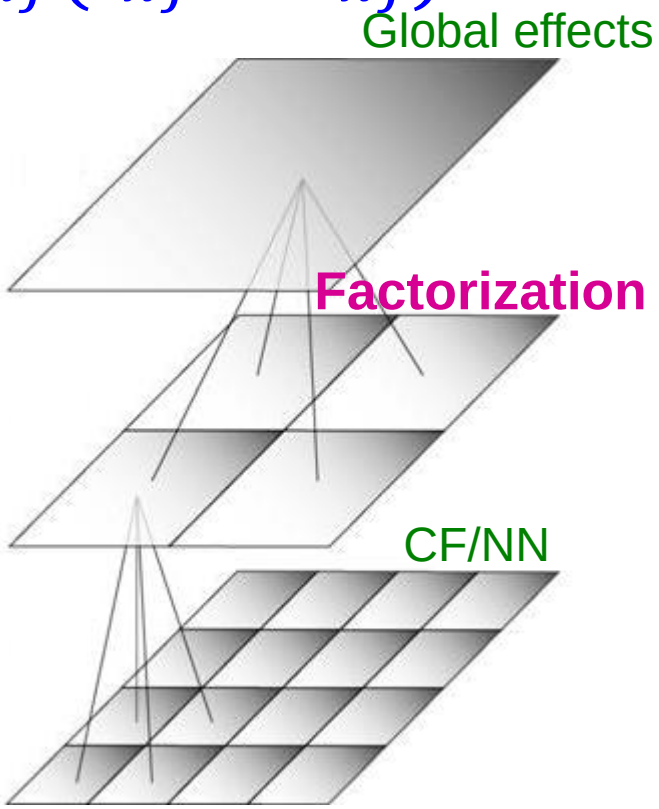
while $|w_{new} - w_{old}| > \varepsilon$:

$w_{old} = w_{new}$

$w_{new} = w_{old} - \eta \cdot \nabla w_{old}$

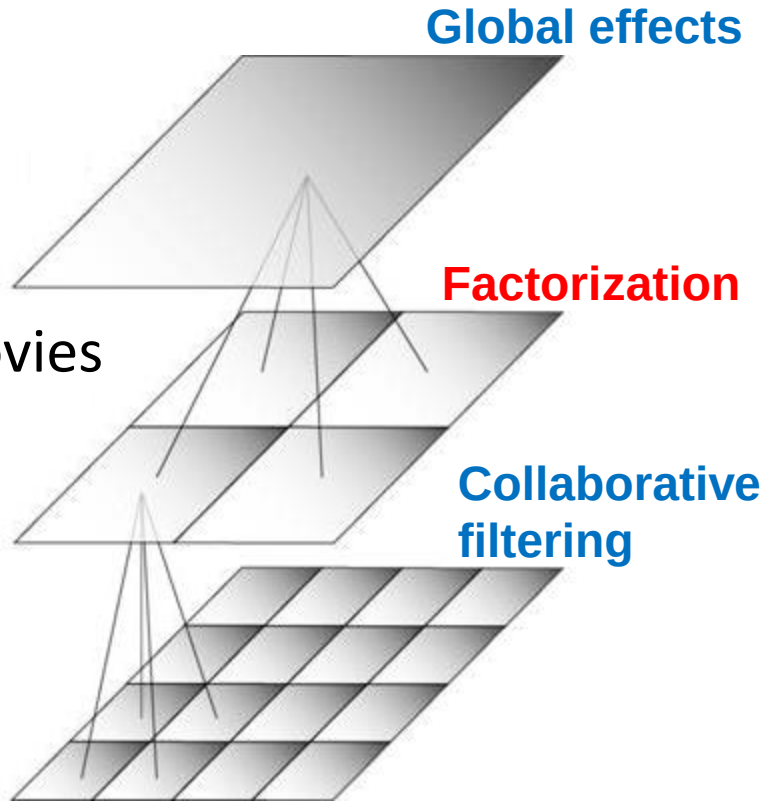
Interpolation Weights

- **So far:** $\widehat{r_{xi}} = b_{xi} + \sum_{j \in N(i;x)} w_{ij}\left(r_{xj} - b_{xj}\right)$
  - Weights $\boldsymbol{w_{ij}}$ derived based on their role; **no use of an arbitrary similarity measure** ($\boldsymbol{w_{ij} \neq s_{ij}}$)
  - Explicitly account for interrelationships among the neighboring movies
- **Next: Latent factor model**
  - Extract "regional" correlations



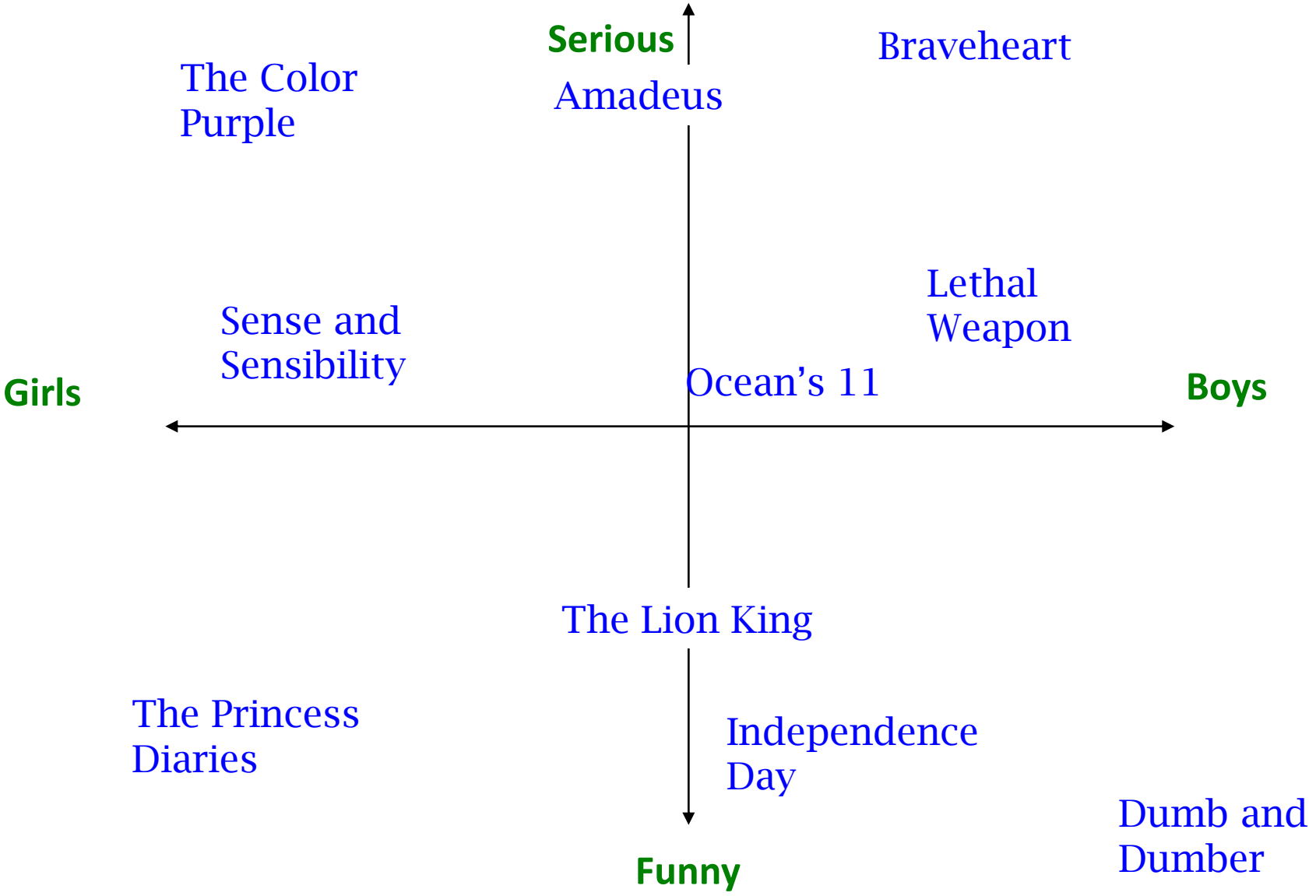
Winner of Netflix Challenge

- **Multi-scale modeling of the data:** Combine top level, "regional" modeling of the data, with a refined, local view:
  - **Global:**
    - Overall deviations of users/movies
  - **Factorization:**
    - Addressing "regional" effects
  - **Collaborative filtering: NN**
    - Extract local patterns

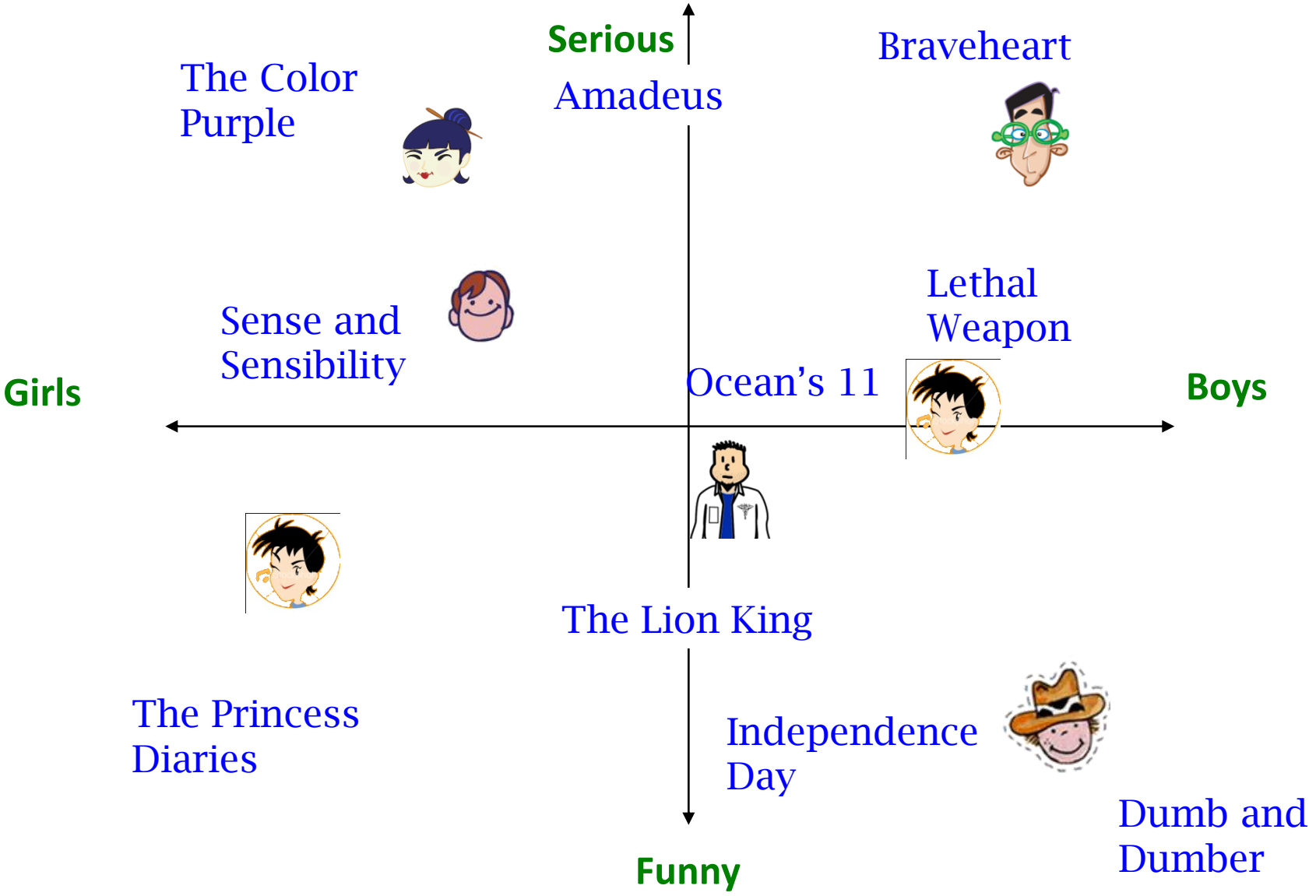


COLLABORATIVE FILTERING: LATENT FACTOR MODELS

Latent Factor Models

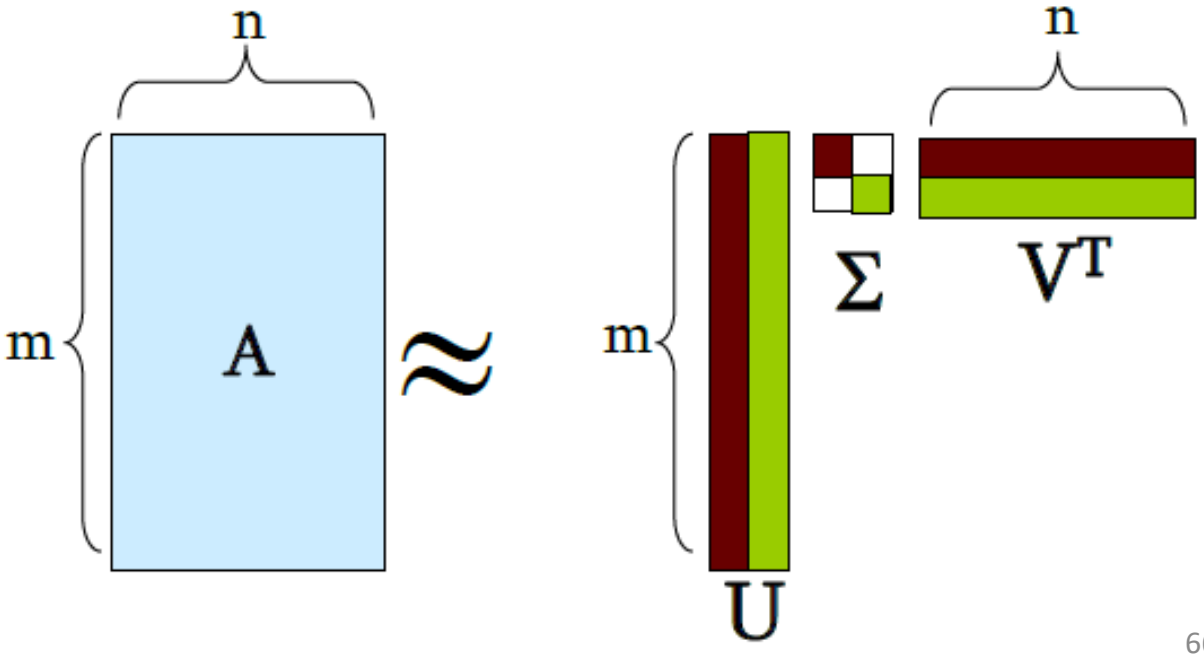


Latent Factor Models



Remember SVD?

- **A**: Input data matrix
- **U**: Left singular vectors (UU^T=I, mxm)
- **V**: Right singular vectors (VV^T=I, nxn)
- **Σ**: Singular values (on diagonal, >=0, mxn)
- A = $U\Sigma V^T$

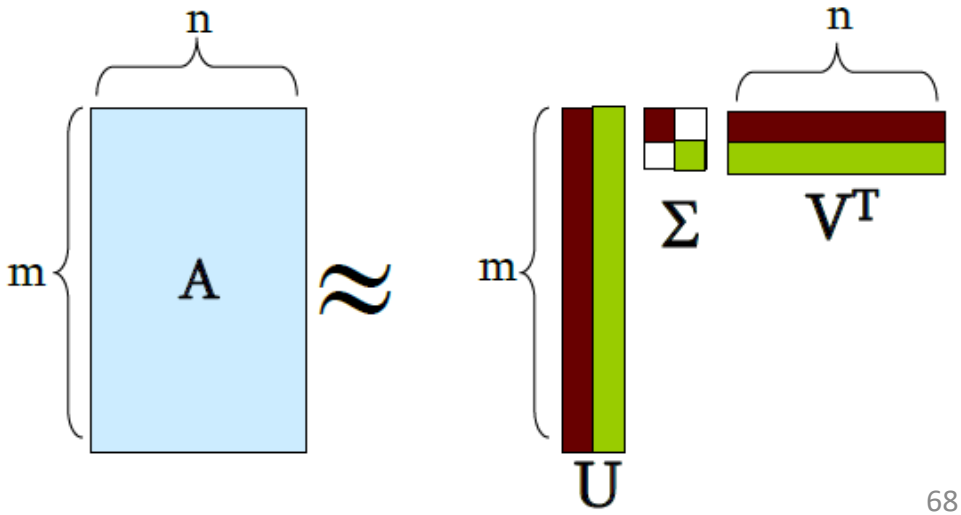


Our needs

- Multiply sigma with V and get $A=UV^T$ decomposition
- SVD will count errors also on missing entries (zeros)
- SVD insists U,V are orthonormal. We don't care
- Heuristic low-rank UV^Tdecomposition

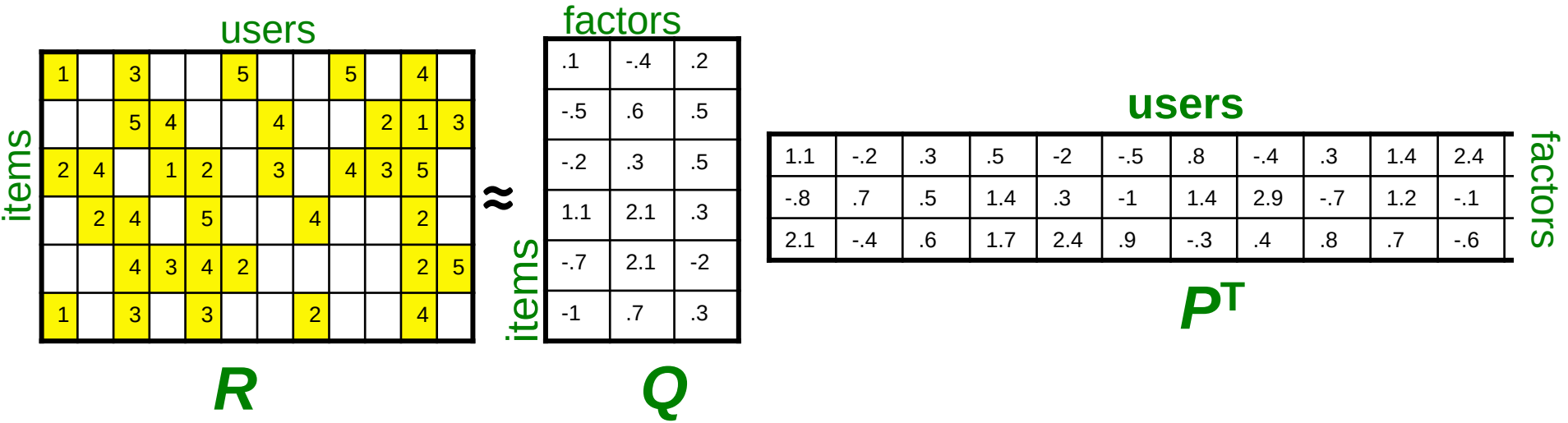
Modifications

- **Only use r highest singular values**
- **A**: Input data matrix
- **U**: Left singular vectors (m x r) (will be called Q)
- **V**: Right singular vectors (n x r) (will be called P)
- **Σ**: Singular values (rxr)
- **Σ V (**will be called P)
- A = $U\Sigma V^{\mathrm{T}}$



Economy-class low-rank heuristic SVD

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



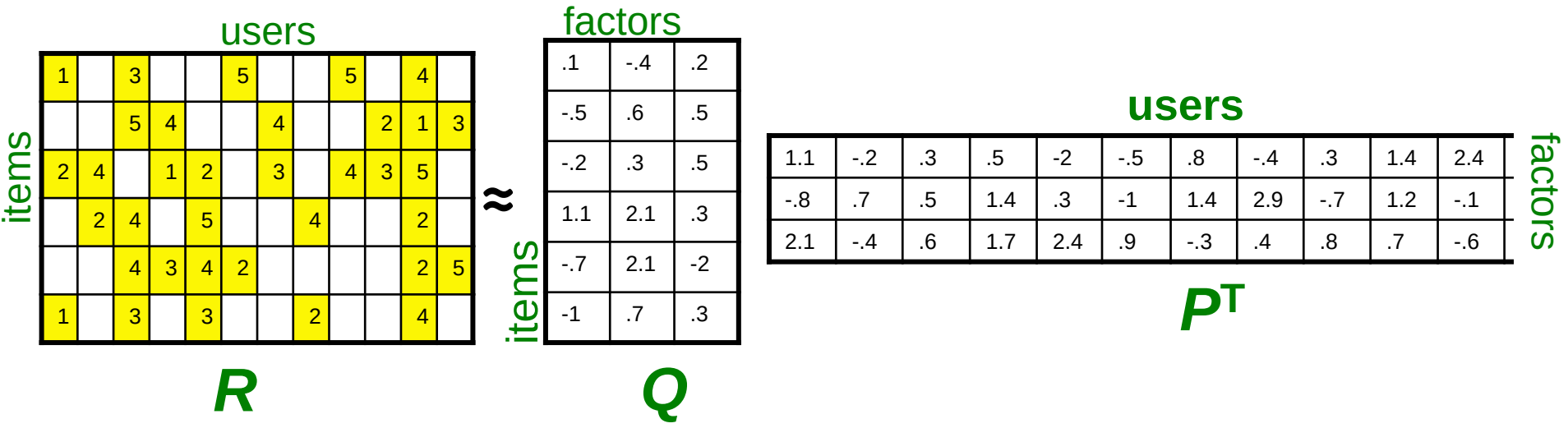
1		3			5			5		4	
		5	4			4			2	1	3
2	4		1	2		3		4	3	5	
	2	4		5			4			2	
		4	3	4	2					2	5
1		3		3			2			4	

.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

Item-User factorization

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



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

- For now let's assume we can approximate the rating matrix $\boldsymbol{R}$ as a product of "thin" $\boldsymbol{Q} \cdot \boldsymbol{P}^T$
  - $\boldsymbol{R}$ has missing entries but let's ignore that for now!
    - 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

Ratings as Products of Factors

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

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

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*?**

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

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*?**

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

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

users

items

1		3			5			5		4	
		5	4	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

items

f factors

Q

•

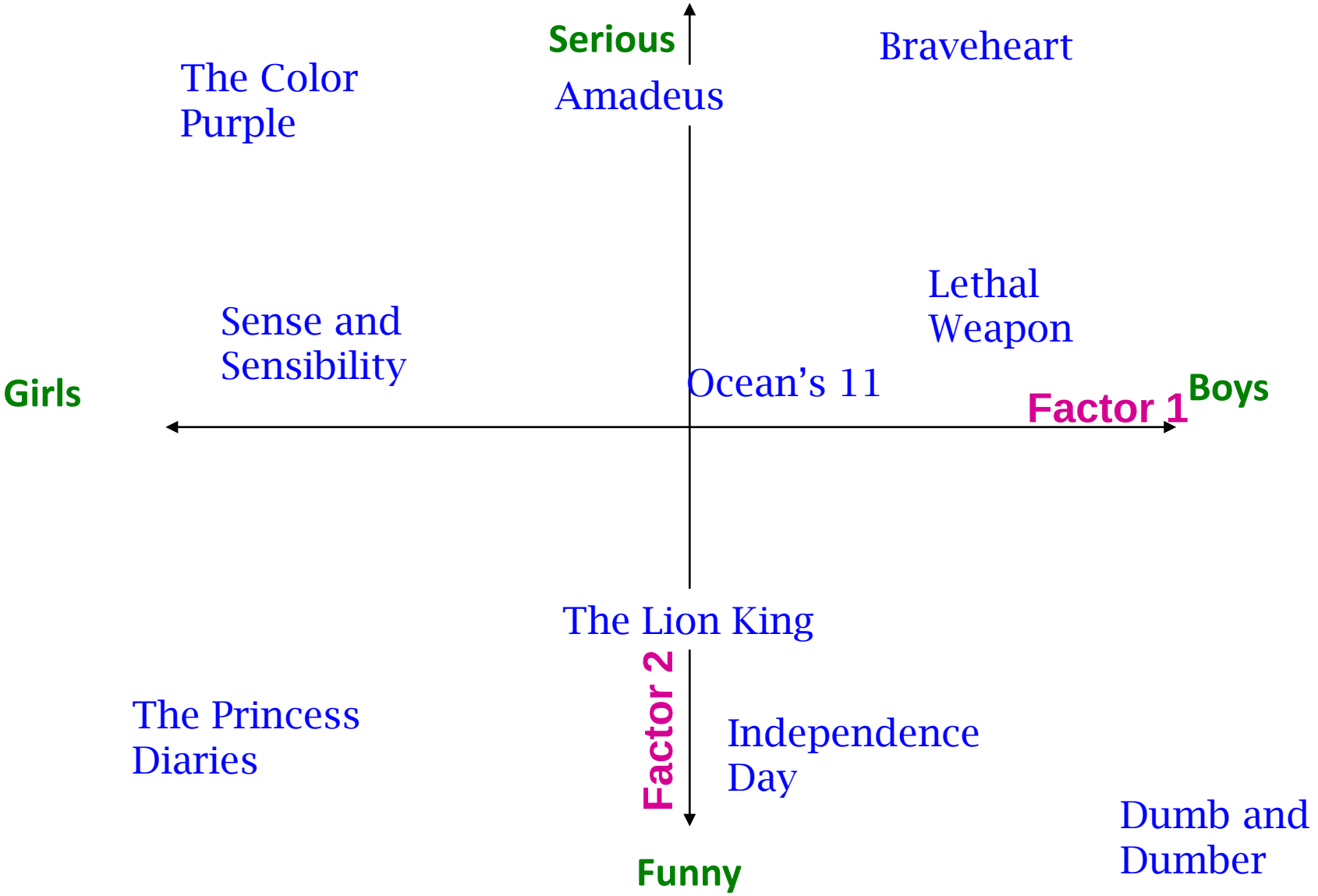
users

f factors

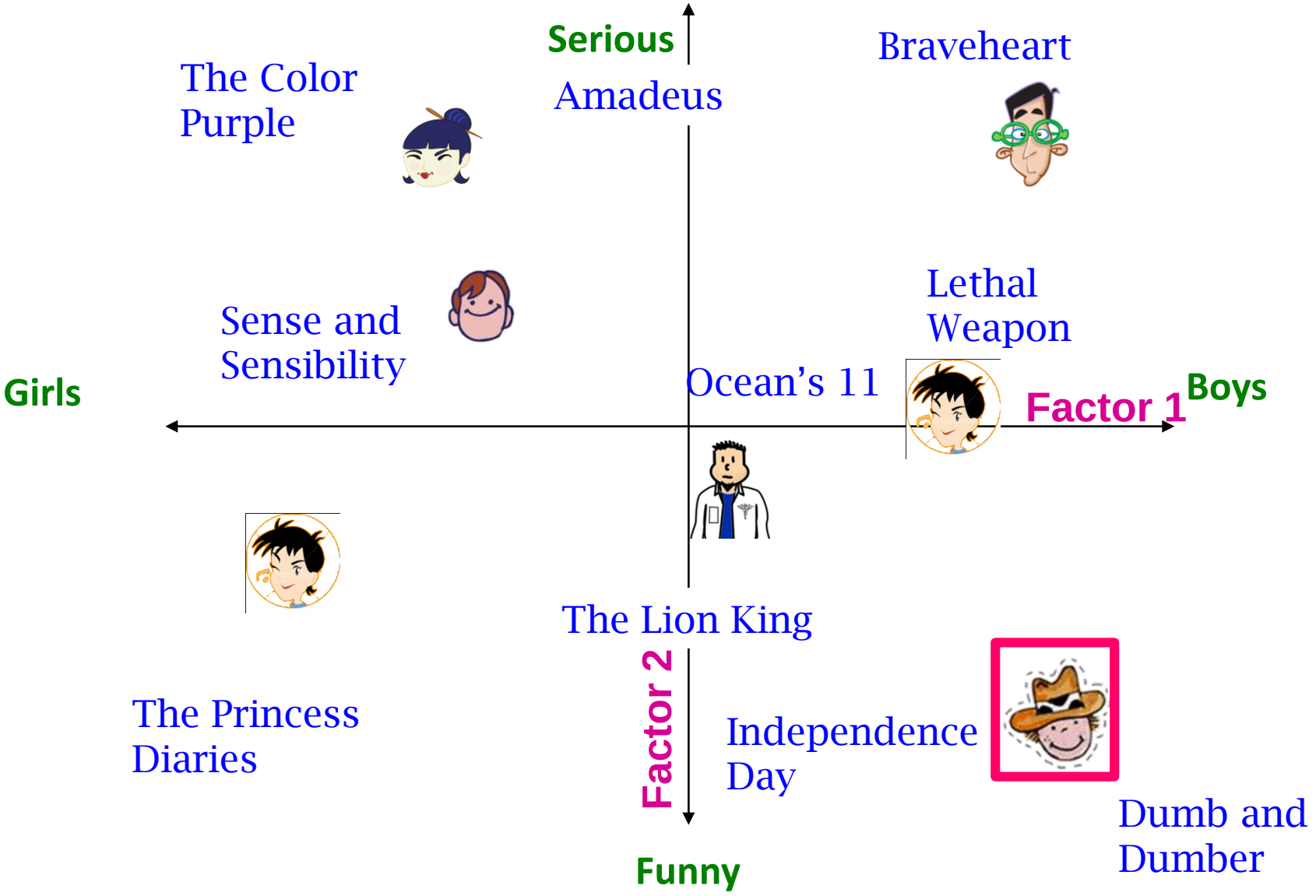
1.1	-.2	.3	.5	-2	-.5	.8	-.4	.3	1.4	2.4	-.9
-.8	.7	.5	1.4	.3	-1	1.4	2.9	-.7	1.2	-.1	1.3
2.1	-.4	.6	1.7	2.4	.9	-.3	.4	.8	.7	-.6	.1

P^T

Latent Factor Models



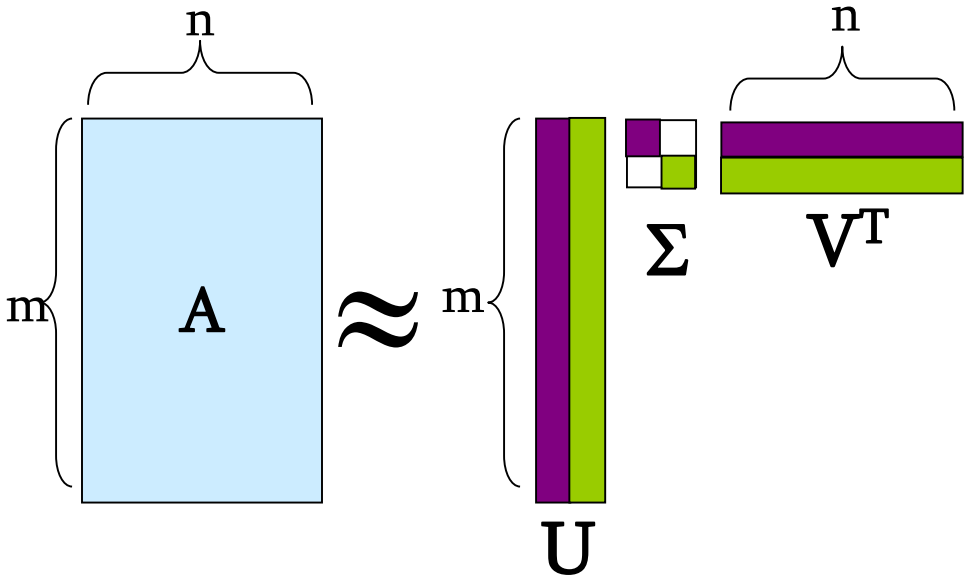
Latent Factor Models



FINDING THE LATENT FACTORS

Recap: SVD and LF

- **SVD**
  - **A**: Input data matrix
  - **U**: Left singular vecs
  - **V**: Right singular vecs
  - **Σ**: Singular values
  - A = $U\Sigma V^{\mathrm{T}}$



- So in our case: "Economy-SVD" on Netflix data:

  $R \approx Q \cdot P^T$

- ***A*** = ***R***, ***Q*** = ***U***, $\boldsymbol{P}^{\mathrm{T}} = \boldsymbol{\Sigma}\,\boldsymbol{V}^{\mathrm{T}}$

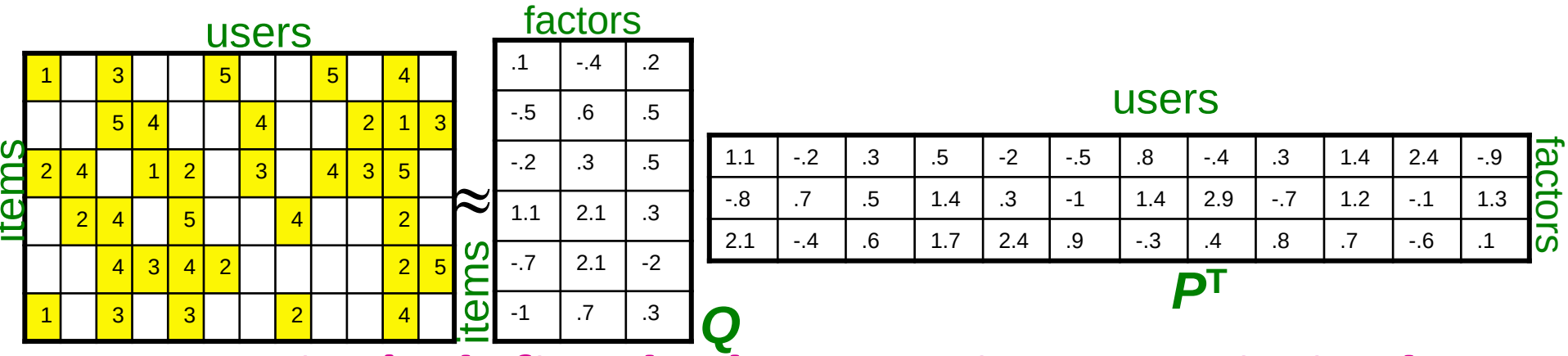
SVD: More good stuff

- **We already know that SVD gives minimum reconstruction error (Sum of Squared Errors):**

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

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

Latent Factor Models



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

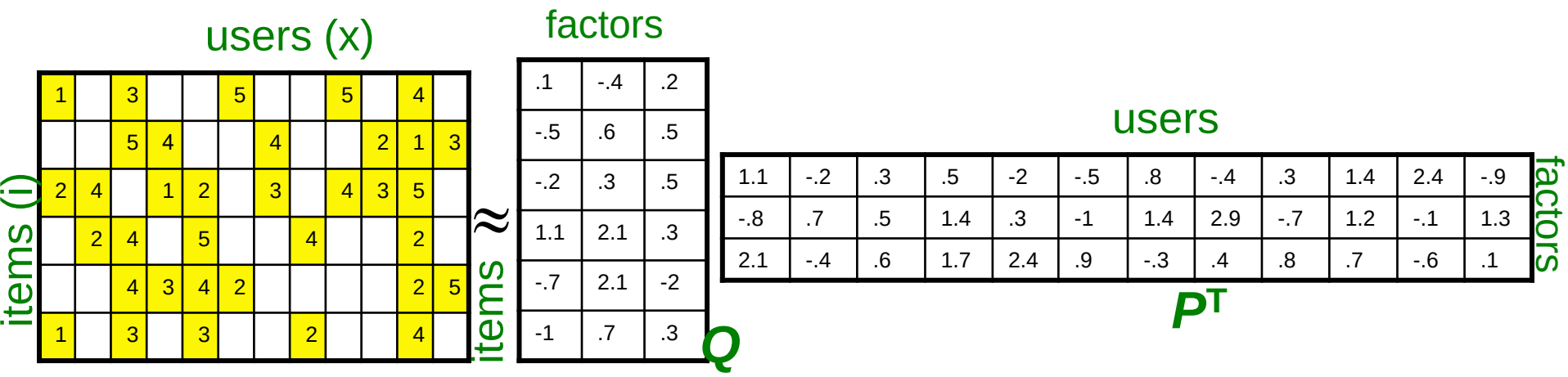
1.1	-.2	.3	.5	-2	-.5	.8	-.4	.3	1.4	2.4	-.9
-.8	.7	.5	1.4	.3	-1	1.4	2.9	-.7	1.2	-.1	1.3
2.1	-.4	.6	1.7	2.4	.9	-.3	.4	.8	.7	-.6	.1

- **SVD isn't defined when entries are missing!**
- **Use specialized methods to find *P*, *Q***
  - $\min_{P,Q} \sum_{(i,x)\in R} \left(r_{ix} - q_i \cdot p_x\right)^2$ $\quad \hat{r}_{ix} = q_i \cdot p_x$
  - **Note:**
    - We don't require cols of ***P, Q*** to be orthogonal/unit length
    - ***P, Q*** map users/movies to a latent factor space
    - *The most popular model among Netflix contestants*

Latent Factor Models

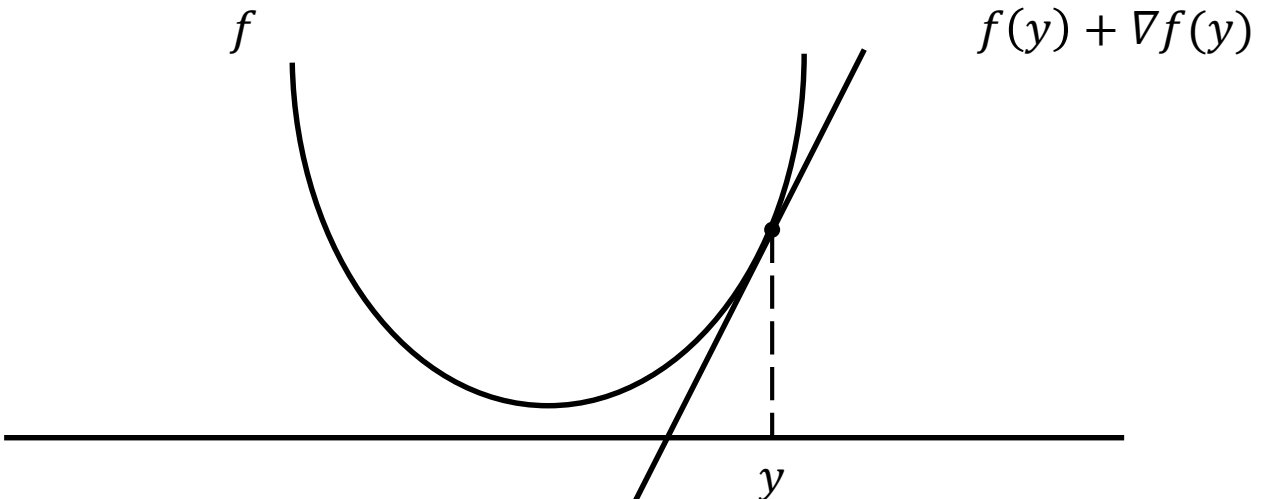
- **Our goal is to find P and Q such that:**

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

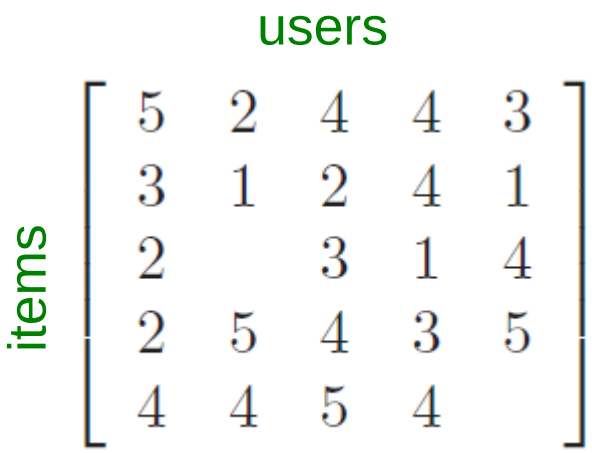


Gradient descent

- Compute the derivative $\boldsymbol{\nabla f}$
- Start at some point y and evaluate $\nabla f(y)$
- Make a step in the reverse direction of the gradient: $y = y - \nabla f(y)$
- Repeat until converged

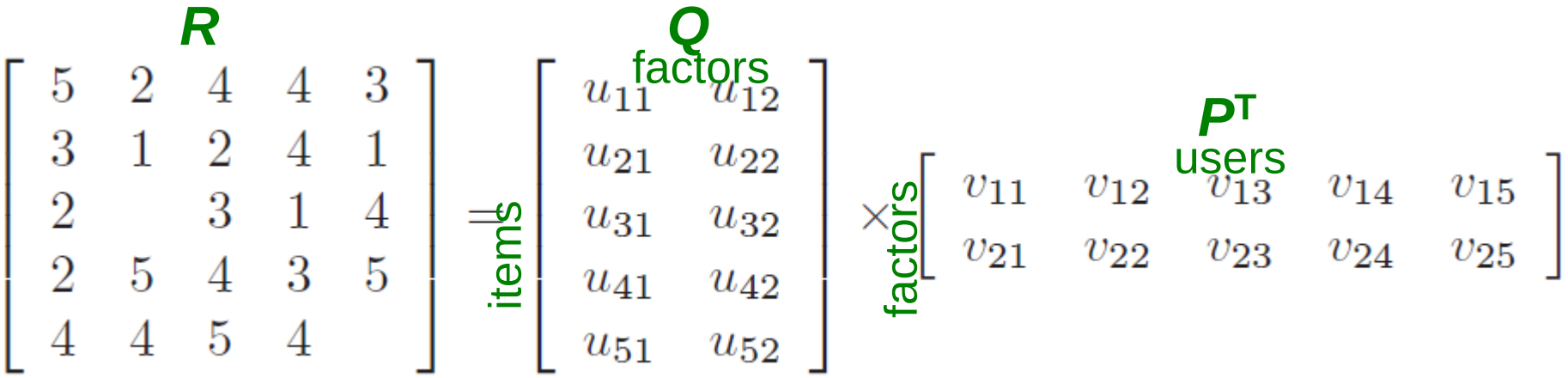


Example optimization



Goal

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



Random initialization

$$\begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 1 & 1 \\ 1 & 1 \\ 1 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} =$$

SSE

- $SSE = \sum_{(i,x)\in R}(r_{ix} - \hat{r}_{ix})^2$

$\boldsymbol{R}$

$$\begin{bmatrix} 5 & 2 & 4 & 4 & 3 \\ 3 & 1 & 2 & 4 & 1 \\ 2 & & 3 & 1 & 4 \\ 2 & 5 & 4 & 3 & 5 \\ 4 & 4 & 5 & 4 & \end{bmatrix}$$

$\hat{\boldsymbol{R}}$

$$\begin{bmatrix} 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 \end{bmatrix}$$

3	0	2	2	1
1	-1	0	2	-1
0		1	-1	2
0	3	2	1	3
2	2	3	2	

SSE=18+7+6+23+21 = 75

$$RMSE = \sqrt{{}^{75}/_{23}} = 1.806$$

Let's update one value

$$\begin{bmatrix} \mathbf{X} & 1 \\ 1 & 1 \\ 1 & 1 \\ 1 & 1 \\ 1 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 \end{bmatrix}$$

- What is the impact of this change on the SSE?

Incremental computation

$$\begin{bmatrix} x & 1 \\ 1 & 1 \\ 1 & 1 \\ 1 & 1 \\ 1 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} x+1 & x+1 & x+1 & x+1 & x+1 \\ 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 \end{bmatrix}$$

Only changes are in first row! $\begin{bmatrix} 5 & 2 & 4 & 4 & 3 \end{bmatrix}$

$$(5-(x+1))^2 + (2-(x+1))^2 + (4-(x+1))^2 + (4-(x+1))^2 + (3-(x+1))^2$$

Simplify

$$(4-x)^2 + (1-x)^2 + (3-x)^2 + (3-x)^2 + (2-x)^2$$

Gradient descent = take derivative and set to zero

$$-2 \times ((4-x) + (1-x) + (3-x) + (3-x) + (2-x)) = 0$$

$$-2 \times (13 - 5x) = 0 \qquad x = 2.6.$$

Update x in Q

$$\begin{bmatrix} 2.6 & 1 \\ 1 & 1 \\ 1 & 1 \\ 1 & 1 \\ 1 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} =$$

Next step

$$\begin{bmatrix} 2.6 & 1 \\ 1 & 1 \\ 1 & 1 \\ 1 & 1 \\ 1 & 1 \end{bmatrix} \times \begin{bmatrix} \mathbf{y} & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 3.6 & 3.6 & 3.6 & 3.6 & 3.6 \\ 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 \end{bmatrix}$$

$$\begin{bmatrix} 2.6 & 1 \\ 1 & 1 \\ 1 & 1 \\ 1 & 1 \\ 1 & 1 \end{bmatrix} \times \begin{bmatrix} y & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 2.6y+1 & 3.6 & 3.6 & 3.6 & 3.6 \\ y+1 & 2 & 2 & 2 & 2 \\ y+1 & 2 & 2 & 2 & 2 \\ y+1 & 2 & 2 & 2 & 2 \\ y+1 & 2 & 2 & 2 & 2 \end{bmatrix}$$

Exercise: Incremental update

- Change to SSE: first column

$$\begin{bmatrix} 2.6y+1 & 3.6 & 3.6 & 3.6 & 3.6 \\ y+1 & 2 & 2 & 2 & 2 \\ y+1 & 2 & 2 & 2 & 2 \\ y+1 & 2 & 2 & 2 & 2 \\ y+1 & 2 & 2 & 2 & 2 \end{bmatrix}$$

$$\big(5-(2.6y+1)\big)^2+\big(3-(y+1)\big)^2+\big(2-(y+1)\big)^2+\big(2-(y+1)\big)^2+\big(4-(y+1)\big)^2$$

- Simplify - derivative = 0 - calculate y

$$(4-2.6y)^2+(2-y)^2+(1-y)^2+(1-y)^2+(3-y)^2$$

$$-2\times\big(2.6(4-2.6y)+(2-y)+(1-y)+(1-y)+(3-y)\big)=0$$

$$y=17.4/10.76=1.617.$$

- Update y in P^T

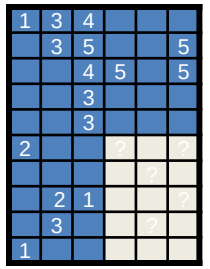
$$\begin{bmatrix} 2.6 & 1 \\ 1 & 1 \\ 1 & 1 \\ 1 & 1 \\ 1 & 1 \end{bmatrix}\times\begin{bmatrix} 1.617 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}=\begin{bmatrix} 5.204 & 3.6 & 3.6 & 3.6 & 3.6 \\ 2.617 & 2 & 2 & 2 & 2 \\ 2.617 & 2 & 2 & 2 & 2 \\ 2.617 & 2 & 2 & 2 & 2 \\ 2.617 & 2 & 2 & 2 & 2 \end{bmatrix}$$

Complete decomposition

- Preprocessing
  - Normalization by movie or by user
- Initialization
  - Random fixed number/average of non-blanks
  - Perturbation
- Update order
  - row-by-row/column-by-column
  - random permutation order
- Convergence
  - threshold per element update/per cycle update

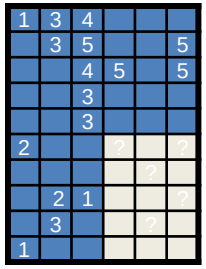
Back to Our Problem

- **Want to minimize SSE for unseen test data**
- **Idea: Minimize SSE on training data**
  - Want large ***k*** (# of factors) to capture all the signals
  - But, **SSE** on **test data** begins to rise for ***k*** > 2



- This is a classical example of **overfitting:**
  - With too much freedom (too many free parameters) the model starts fitting noise
    - That is it fits too well the training data and thus **not generalizing** well to unseen test data

Regularization



- **To solve overfitting we introduce regularization:**
  - Allow rich model where there are sufficient data
  - Shrink aggressively where data are scarce

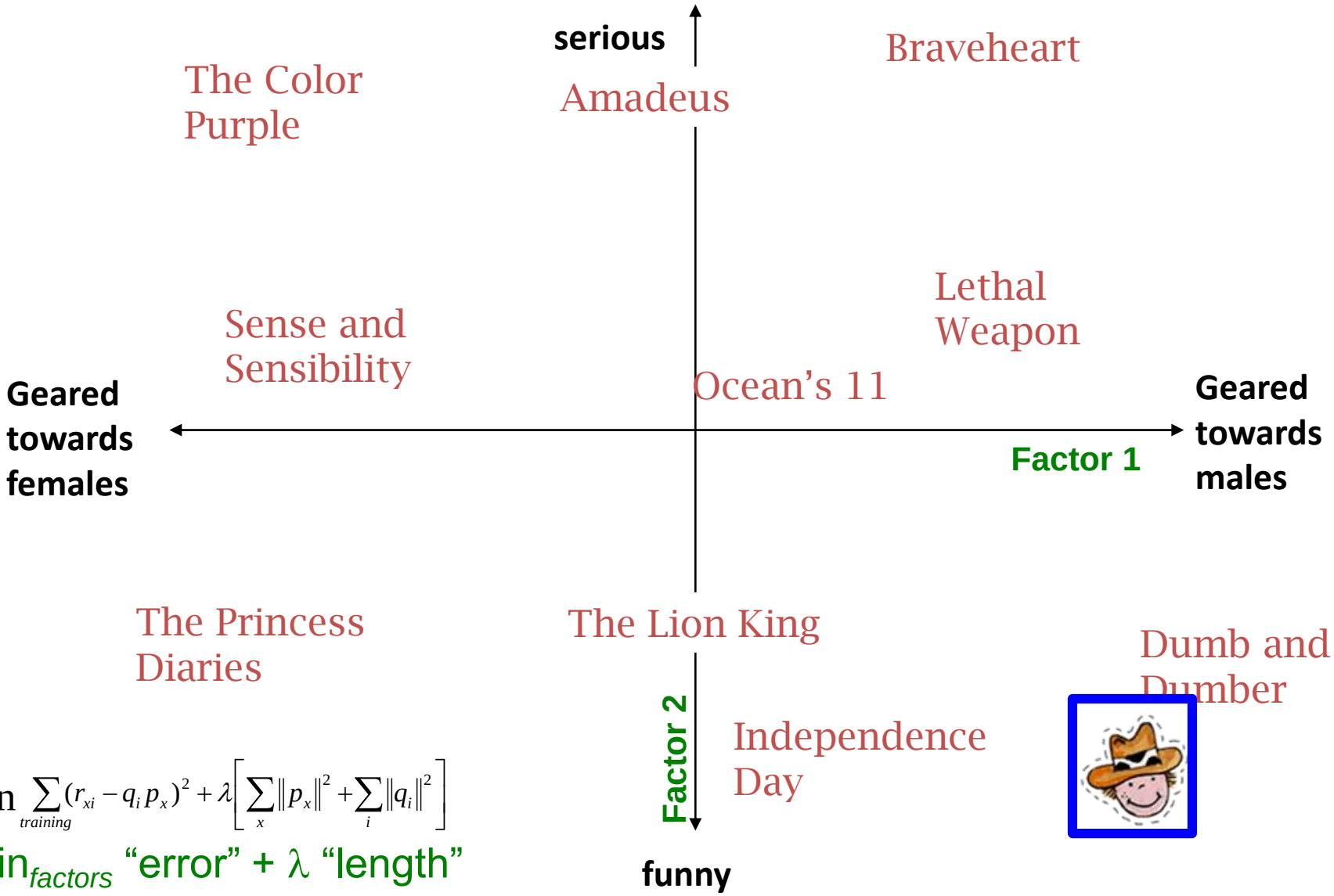
$$\min_{P,Q} \sum_{training} (r_{xi} - q_i p_x)^2 + \left[\lambda_1 \sum_x \|p_x\|^2 + \lambda_2 \sum_i \|q_i\|^2 \right]$$

"error" "length"

λ_1, λ_2 … user set regularization parameters

Note: We do not care about the "raw" value of the objective function, but we care about P,Q that achieve the minimum of the objective

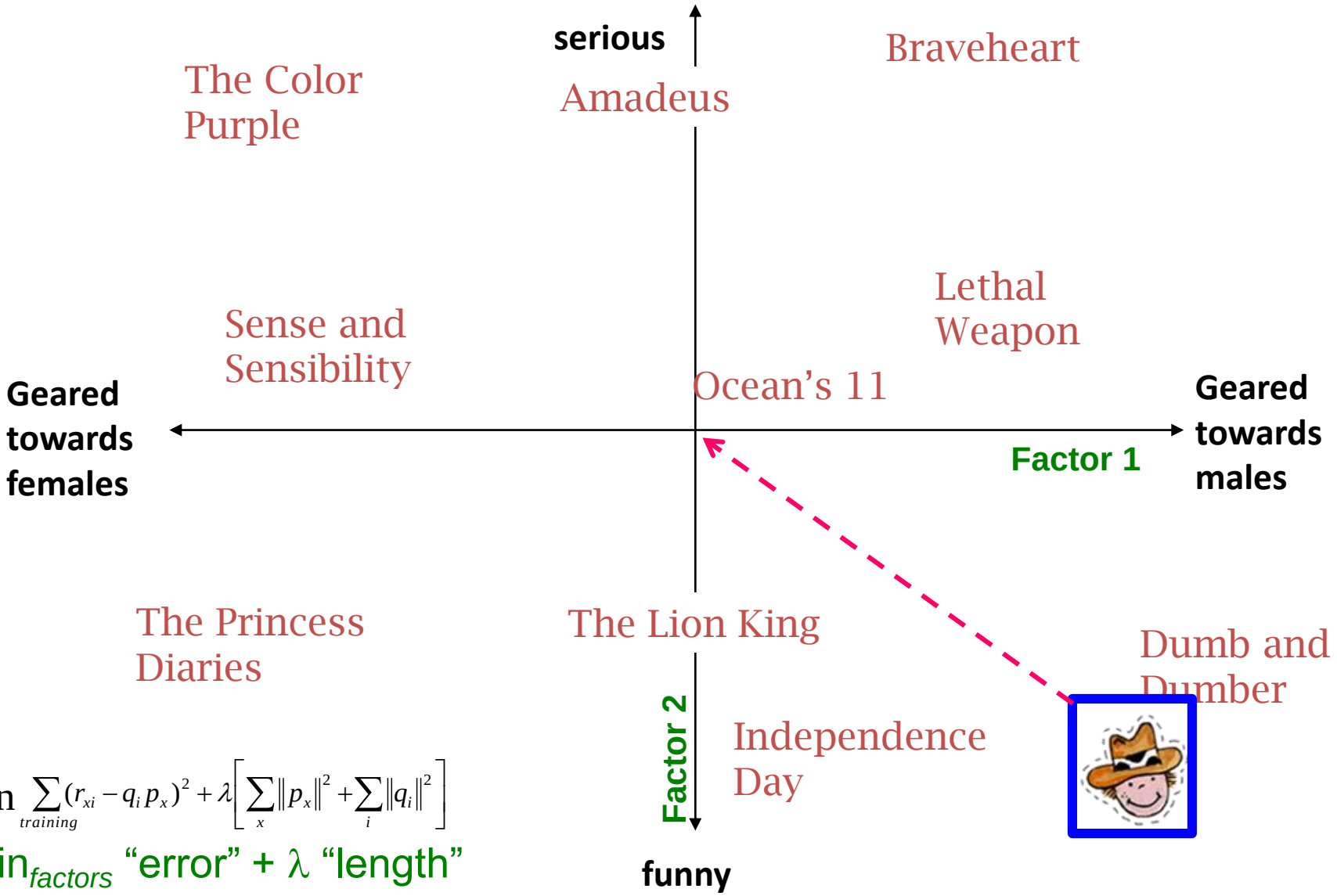
The Effect of Regularization



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

$\min_{factors}$ "error" + λ "length"

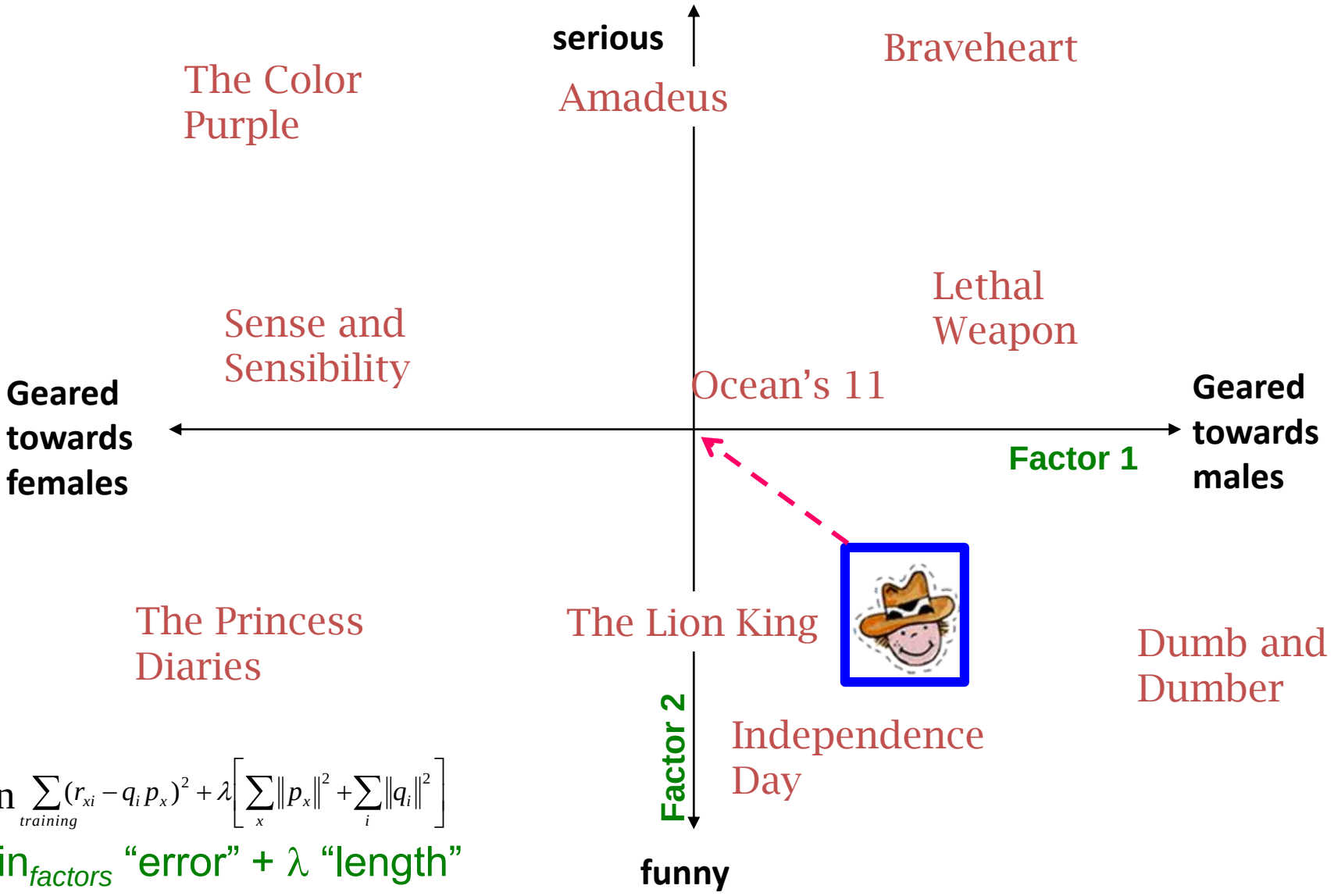
The Effect of Regularization



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

$\min_{factors}$ "error" + λ "length"

The Effect of Regularization



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

$\min_{factors}$ "error" + λ "length"

EXTENDING LATENT FACTOR MODEL TO INCLUDE BIASES

Baseline Predictor

- We have expectations on the rating by user ***x*** of movie ***i***, even without estimating ***x***'s attitude towards movies like ***i***

- Rating scale of user ***x***
- Values of other ratings user gave recently (day-specific mood, anchoring, multi-user accounts)

- (Recent) popularity of movie ***i***
- Selection bias; related to number of ratings user gave on the same day ("frequency")

Example: modeling biases

- **Global: biases**
  - Mean movie rating: **3.7 stars**
  - *The Sixth Sense* is **0.5** stars above avg.
  - Joe rates **0.2** stars below avg.

    ⇒ Baseline estimation:

    ***Joe* will rate *The Sixth Sense* 4 stars**
- **Local neighborhood (CF/NN):**
  - *Joe* didn't like related movie *Signs*
  - **⇒ Final estimate:**

    ***Joe* will rate *The Sixth Sense* 3.8 stars**

Modeling Biases and Interactions

Baseline predictor

- Separates users and movies
- Benefits from insights into user's behavior
- Among the main practical contributions of the competition

User-Movie interaction

- Characterizes the matching between users and movies
- Attracts most research in the field
- Benefits from algorithmic and mathematical innovations

- μ = overall mean rating
- b_x = bias of user x
- b_i = bias of movie i

Putting bias and LF together

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

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

- **Example:**
  - Mean rating: $\mu = 3.7$
  - You are a critical reviewer: your ratings are 1 star lower than the mean: $b_x = -1$
  - Star Wars gets a mean rating of *0.5* higher than average movie: $b_i = +0.5$
  - Baseline rating for you on Star Wars:
    $= 3.7 - 1 + 0.5 = 3.2$

Fitting the New Model

- **Solve:**

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

goodness of fit

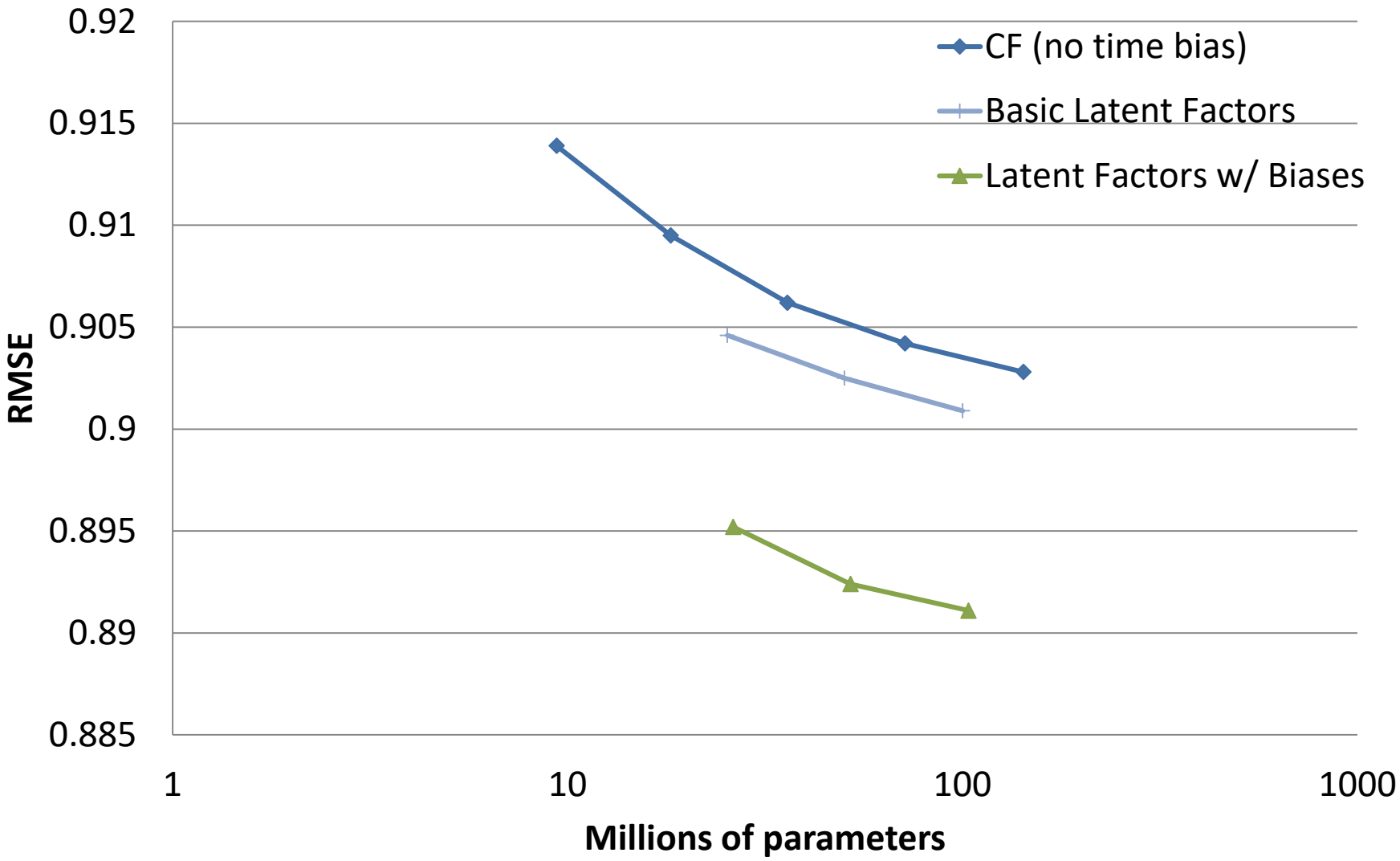
$$+ \left(\lambda_1 \sum_i \|q_i\|^2 + \lambda_2 \sum_x \|p_x\|^2 + \lambda_3 \sum_x \|b_x\|^2 + \lambda_4 \sum_i \|b_i\|^2 \right)$$

regularization

λ is selected via grid-search on a validation set

- **Gradient decent to find parameters**
  - **Note:** Both biases b_x, b_i as well as interactions q_i, p_x are treated as parameters (we estimate them)

Performance of Various Methods

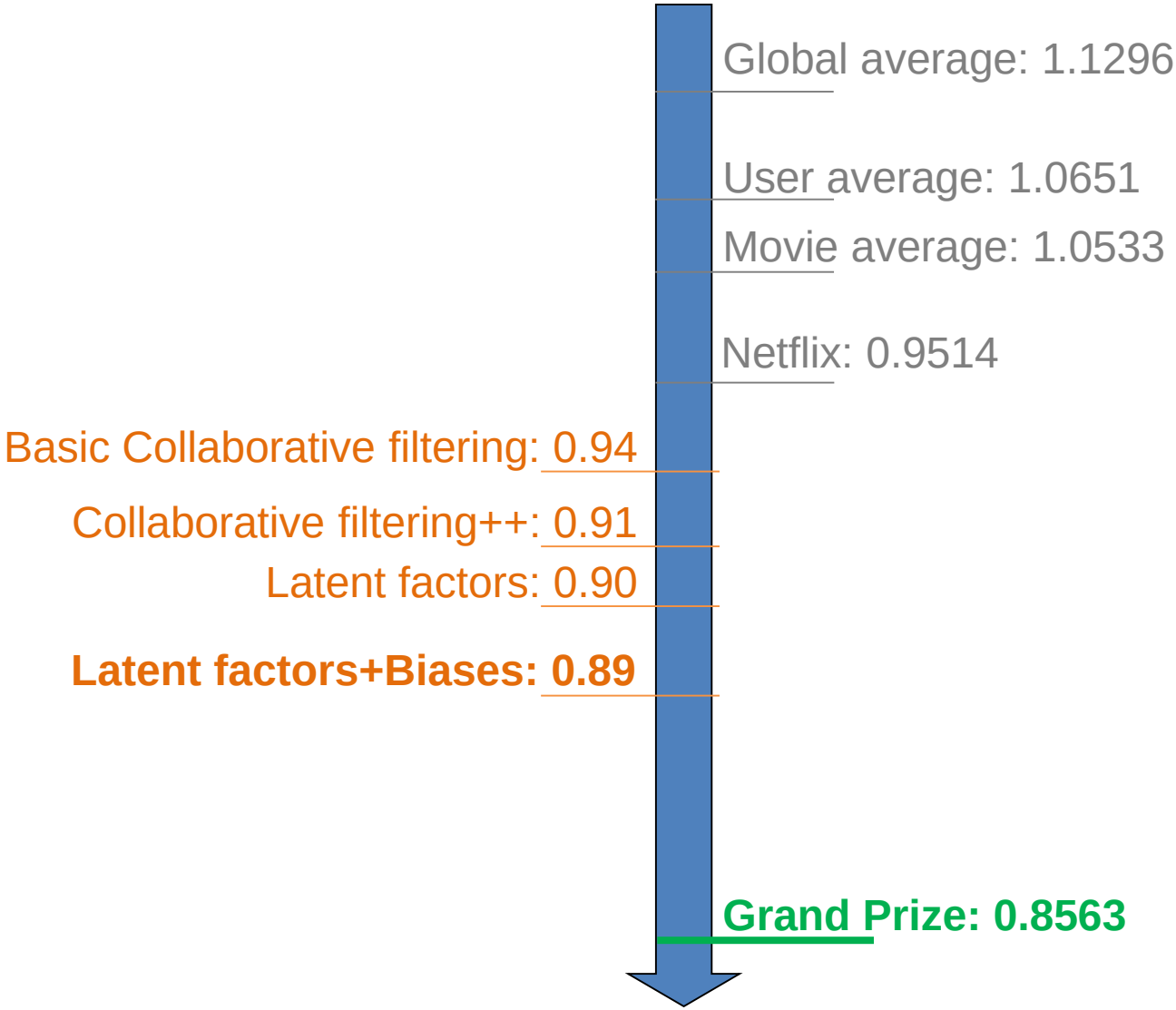


THE NETFLIX CHALLENGE: 2006-09

Awards

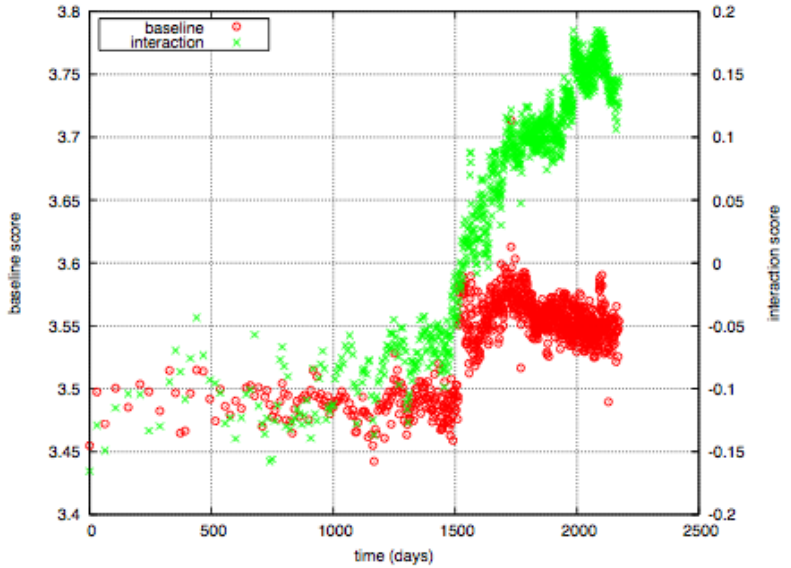
- 1M grand prize for 10 % improvement over Netflix' model
- Once per year 'progress prize' of $50,000
  - At least 1% improvement
  - First two were won by BellKor/BigChaos
- One submission per day
- 10% -> 30 day last call

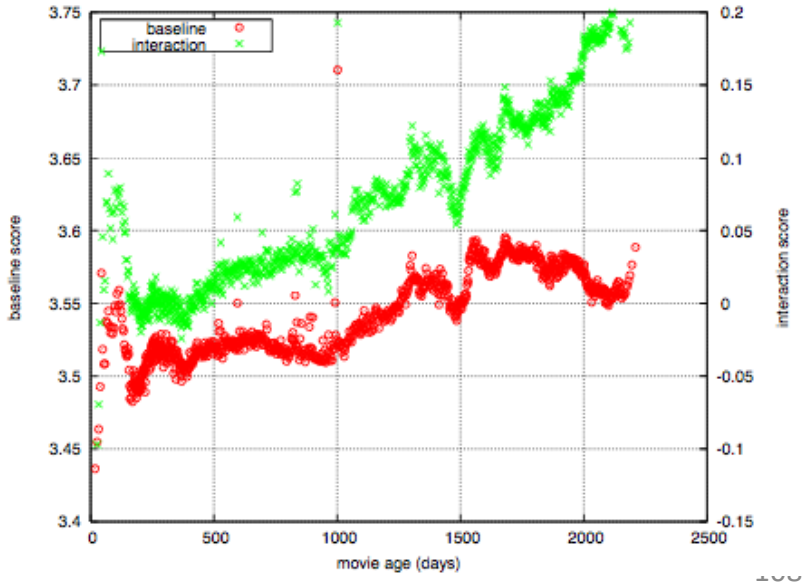
Performance of Various Methods



Temporal Biases Of Users

- **Sudden rise in the average movie rating** (early 2004)
  - Improvements in Netflix
  - GUI improvements
  - Meaning of rating changed
- **Movie age**
  - Users prefer new movies without any reasons
  - Older movies are just inherently better than newer ones





Y. Koren, Collaborative filtering with temporal dynamics, KDD '09

Temporal Biases & Factors

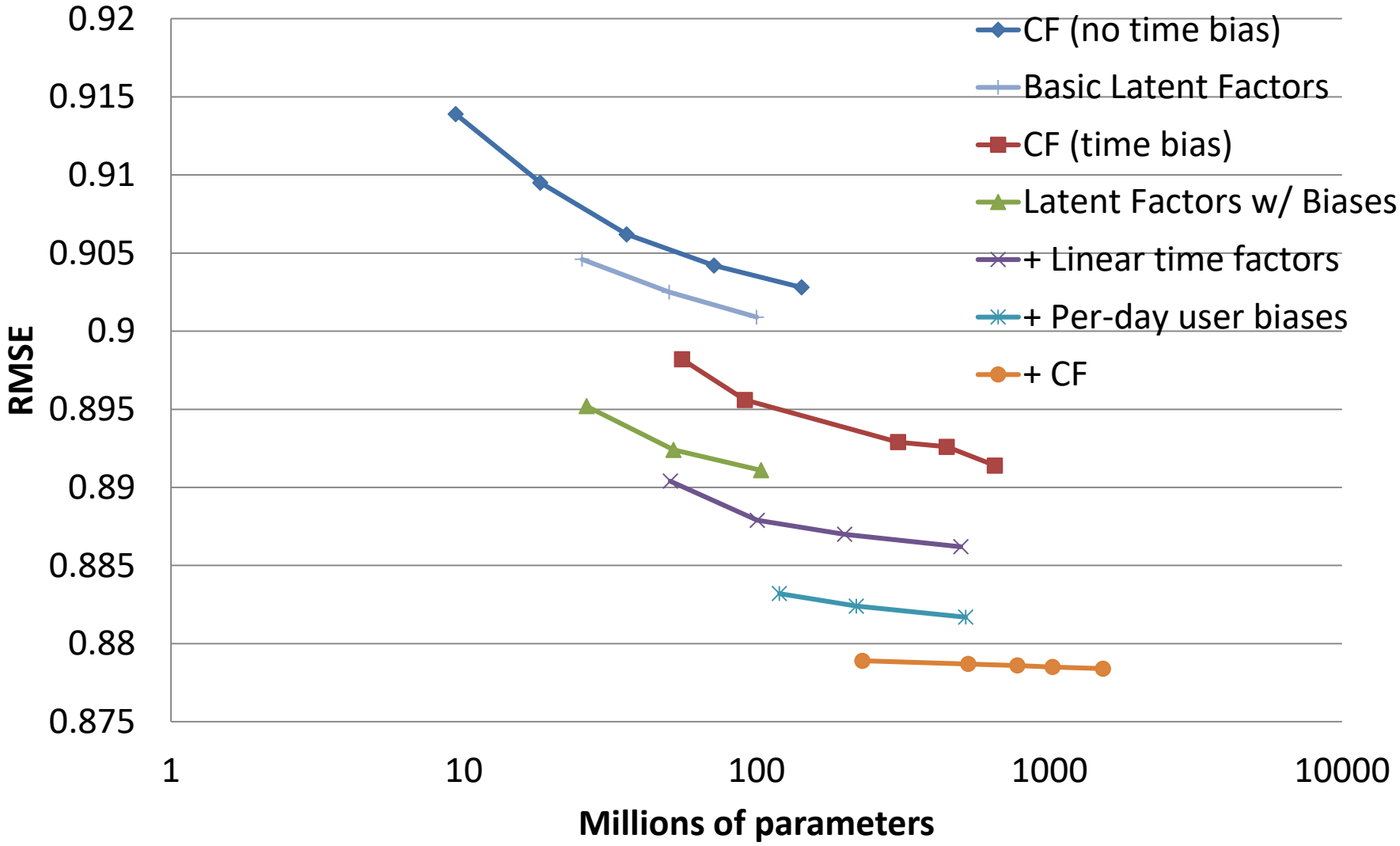
- **Original model:**
  $r_{xi} = \mu + b_x + b_i + q_i \cdot p_x$
- **Add time dependence to biases:**
  $r_{xi} = \mu + b_x(t) + b_i(t) + q_i \cdot p_x$
  - Make parameters b_x and b_i to depend on time
  - **(1)** Parameterize time-dependence by linear trends
    (2) Each bin corresponds to 10 consecutive weeks

$$b_i(t) = b_i + b_{i,\mathrm{Bin}(t)}$$

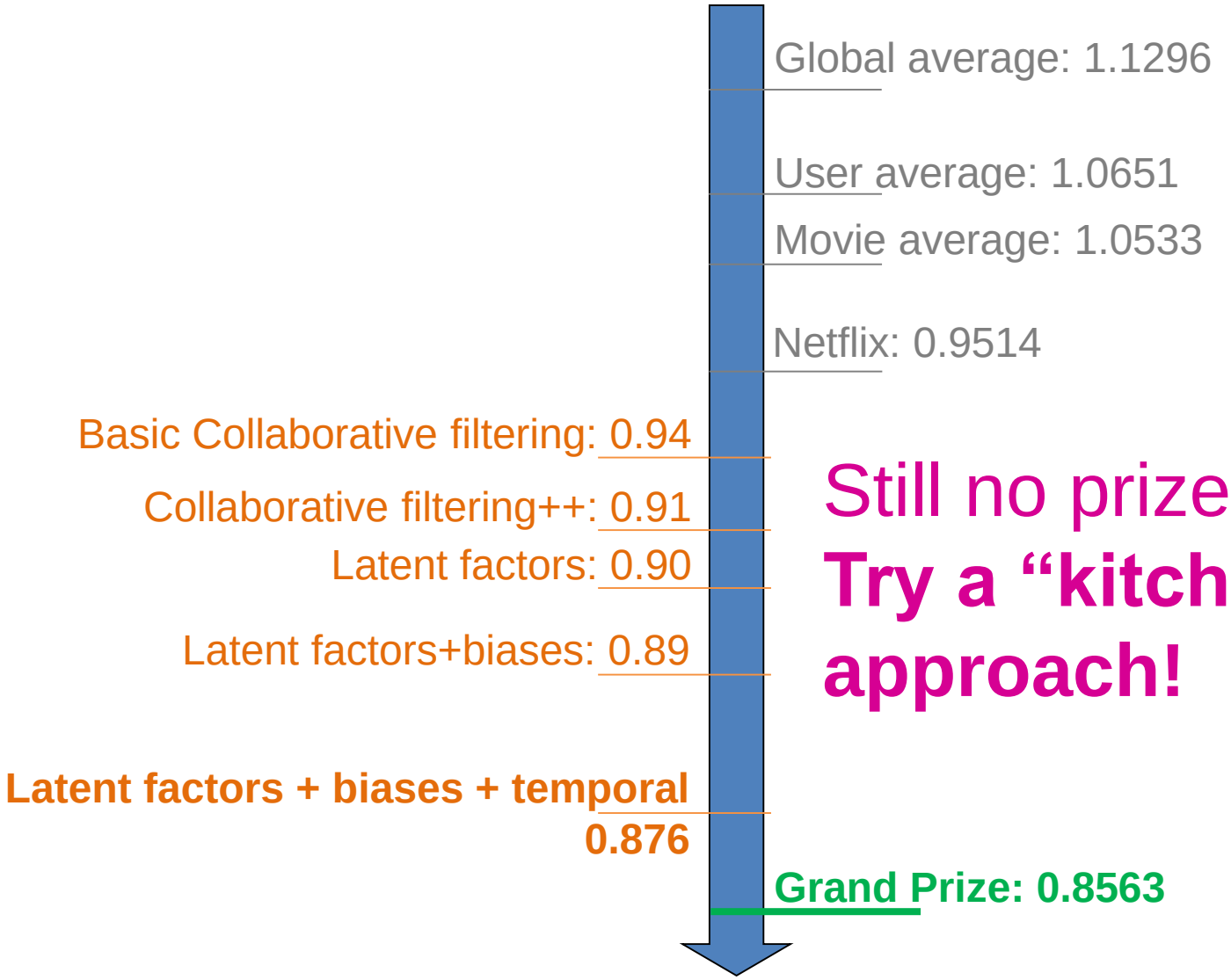
- **Add temporal dependence to factors**
  - $p_x(t)$… user preference vector on day t

Y. Koren, Collaborative filtering with temporal dynamics, KDD '09

Adding Temporal Effects



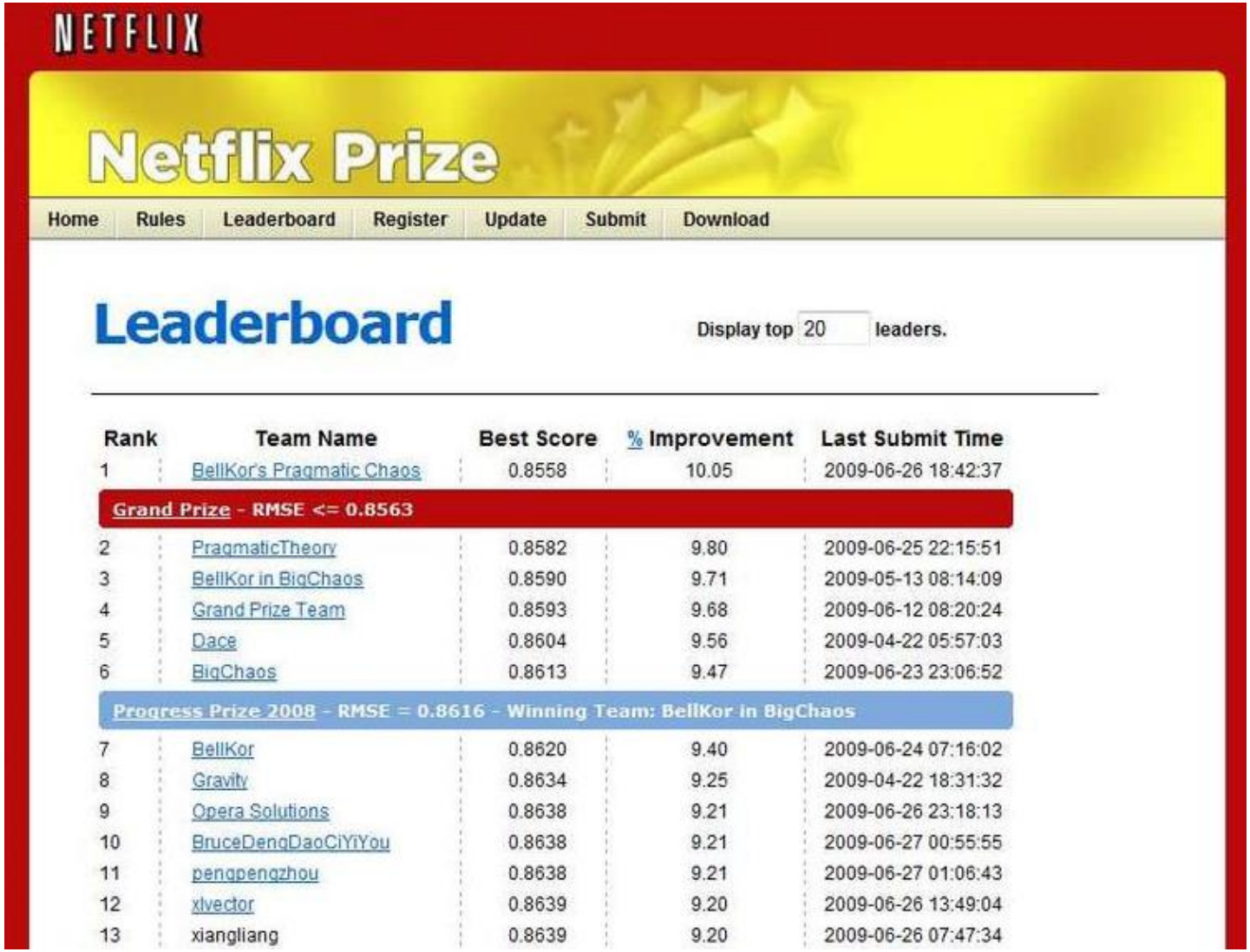
Performance of Various Methods



Still no prize! ☹

Try a "kitchen sink" approach!

Ranking on June 26th 2009



NETFLIX

Netflix Prize

Home | Rules | Leaderboard | Register | Update | Submit | Download

Leaderboard

Display top 20 leaders.

Rank	Team Name	Best Score	% Improvement	Last Submit Time
1	BellKor's Pragmatic Chaos	0.8558	10.05	2009-06-26 18:42:37
Grand Prize - RMSE <= 0.8563				
2	PragmaticTheory	0.8582	9.80	2009-06-25 22:15:51
3	BellKor in BigChaos	0.8590	9.71	2009-05-13 08:14:09
4	Grand Prize Team	0.8593	9.68	2009-06-12 08:20:24
5	Dace	0.8604	9.56	2009-04-22 05:57:03
6	BigChaos	0.8613	9.47	2009-06-23 23:06:52
Progress Prize 2008 - RMSE = 0.8616 - Winning Team: BellKor in BigChaos				
7	BellKor	0.8620	9.40	2009-06-24 07:16:02
8	Gravity	0.8634	9.25	2009-04-22 18:31:32
9	Opera Solutions	0.8638	9.21	2009-06-26 23:18:13
10	BruceDengDaoCiYiYou	0.8638	9.21	2009-06-27 00:55:55
11	pengpengzhou	0.8638	9.21	2009-06-27 01:06:43
12	xlvector	0.8639	9.20	2009-06-26 13:49:04
13	xiangliang	0.8639	9.20	2009-06-26 07:47:34

June 26th submission triggers 30-day "last call"

24 Hours from the Deadline

- **Submissions limited to 1 a day**
  - Only 1 final submission could be made in the last 24h
- **25 hours before deadline…**
  - **BellKor** team member in Austria notices (by chance) that **Ensemble** posts a score that is slightly better than BellKor's
- **Frantic last 24 hours for both teams**
  - Much computer time on final optimization
  - Carefully calibrated to end about an hour before deadline
- **Final submissions**
  - **BellKor** submits a little early (on purpose), 40 mins before deadline
  - **Ensemble** submits their final entry 20 mins later
  - **….and everyone waits….**

NETFLIX

Netflix Prize

COMPLETED

Home | Rules | Leaderboard | Update | Download

Leaderboard

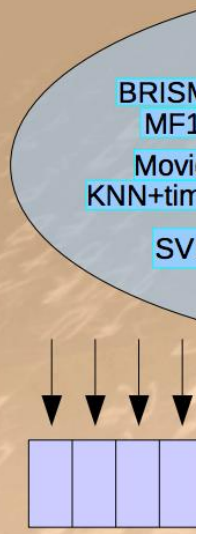
Showing Test Score. Click here to show quiz score

Display top 20 leaders.

Rank	Team Name	Best Test Score	% Improvement	Best Submit Time
Grand Prize - RMSE = 0.8567 - Winning Team: BellKor's Pragmatic Chaos				
1	BellKor's Pragmatic Chaos	0.8567	10.06	2009-07-26 18:18:28
2	The Ensemble	0.8567	10.06	2009-07-26 18:38:22
3	Grand Prize Team	[illegible]	[illegible]	[illegible]
4	Opera Solutions and Vandelay United	0.8588	9.84	2009-07-10 01:12:31
5	Vandelay Industries !	0.8591	9.81	2009-07-10 00:32:20
6	PragmaticTheory	0.8594	9.77	2009-06-24 12:06:56
7	BellKor in BigChaos	0.8601	9.70	2009-05-13 08:14:09
8	Dace	0.8612	9.59	2009-07-24 17:18:43
9	Feeds2	0.8622	9.48	2009-07-12 13:11:51
10	BigChaos	0.8623	9.47	2009-04-07 12:33:59
11	Opera Solutions	0.8623	9.47	2009-07-24 00:34:07
12	BellKor	0.8624	9.46	2009-07-26 17:19:11
Progress Prize 2008 - RMSE = 0.8627 - Winning Team: BellKor in BigChaos				
13	xiangliang	0.8642	9.27	2009-07-15 14:53:22
14	Gravity	0.8643	9.26	2009-04-22 18:31:32
15	Ces	0.8651	9.18	2009-06-21 19:24:53
16	Invisible Ideas	0.8653	9.15	2009-07-15 15:53:04
17	Just a guy in a garage	0.8662	9.06	2009-05-24 10:02:54
18	J Dennis Su	0.8666	9.02	2009-03-07 17:16:17
19	Craig Carmichael	0.8666	9.02	2009-07-25 16:00:54
20	acmehill	0.8668	9.00	2009-03-21 16:20:50
Progress Prize 2007 - RMSE = 0.8723 - Winning Team: KorBell				

Million $ Awarded Sept 21st 2009

BellKorPragmaticChaos



Normalization of global effects

- baseline ratings
- user effects + temporal
- movie effects + temporal
- temporal user-movie effects

Neighborhood models

- item-item/user-user
- non-independent neighbors/neighbor density

Implicit modelling

- merely the fact of a rating implies 'like'

Matrix factorization: symmetric/asymmetric/++

Regression

Restricted Boltzmann Machines

External information: news/twitter/...

Regularization: to avoid overfitting

Ensemble methods

~~Netflix Challenge 2~~

- Announced August 9th, 2009
- Cancelled March 12, 2010 due to privacy challenges
- Researchers could connect IMDB user names from overlaying anonymized Netflix data with IMDB ratings
  - The ratings of less-popular films, coupled with the dates they're rated, form a kind of movie-preference fingerprint that can be used to make matches

FOR DUMMIES®

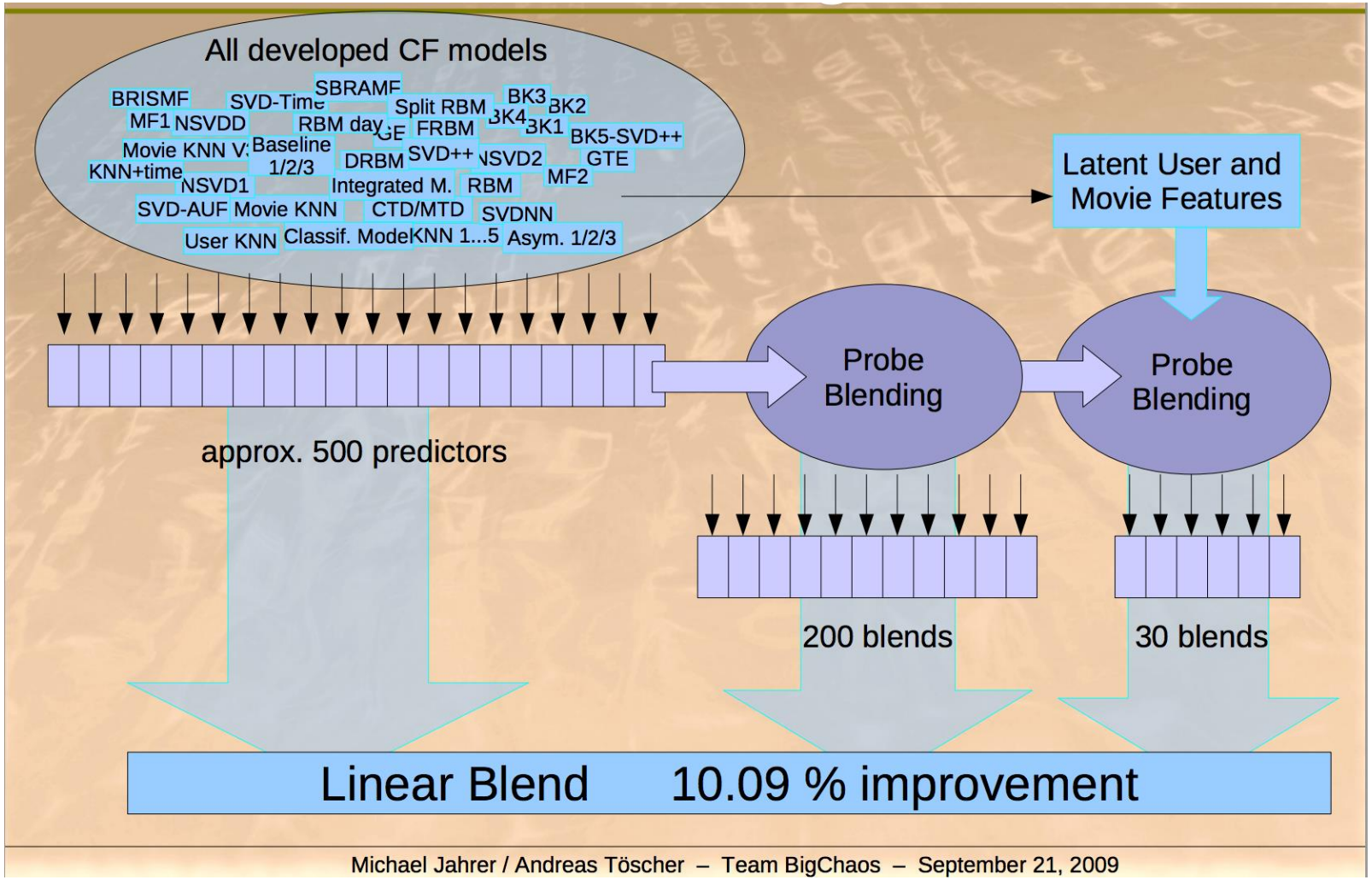
Challenge

- Movie recommendation system
- ~ 1M ratings
- Predict ratings of 90k movies
- Error is measured with RMSE
- Write report (?)
- Submit scores to rank on leaderboard (?)

Expectations

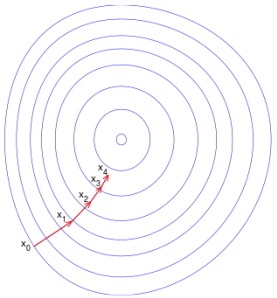
- Absolute minimum:
  - Implement CF, latent factors
  - Combine predictors and use global biases
- Further ideas:
  - local biases, compare item-item vs. user-user, interpolated weights, parameter optimization for LF, linear or learned model combiners, multiple LF/CF models, surprise us positively

Winning model of Netflix challenge



BONUS MATERIAL - NOT COVERED USEFUL FOR PROJECT

Stochastic Gradient Descent



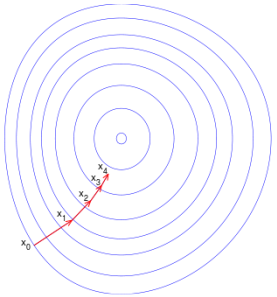
- **Want to find matrices *P* and *Q*:**

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

- **Gradient decent:**
  - Initialize ***P*** and ***Q*** (using SVD, pretend missing ratings are 0)
  - Do gradient descent:
    - $P \leftarrow P - \eta \cdot \nabla P$
    - $Q \leftarrow Q - \eta \cdot \nabla Q$
    - where ∇Q is gradient/derivative of matrix ***Q***:
      $\nabla Q = [\nabla q_{if}]$ and $\nabla q_{if} = \sum_{x,i} -2(r_{xi} - q_i p_x) p_{xf} + 2\lambda_2 q_{if}$
      - Here q_{if} is entry ***f*** of row q_i of matrix ***Q***
  - **Observation: Computing gradients is slow!**

How to compute gradient of a matrix?
Compute gradient of every element independently!

Stochastic Gradient Descent



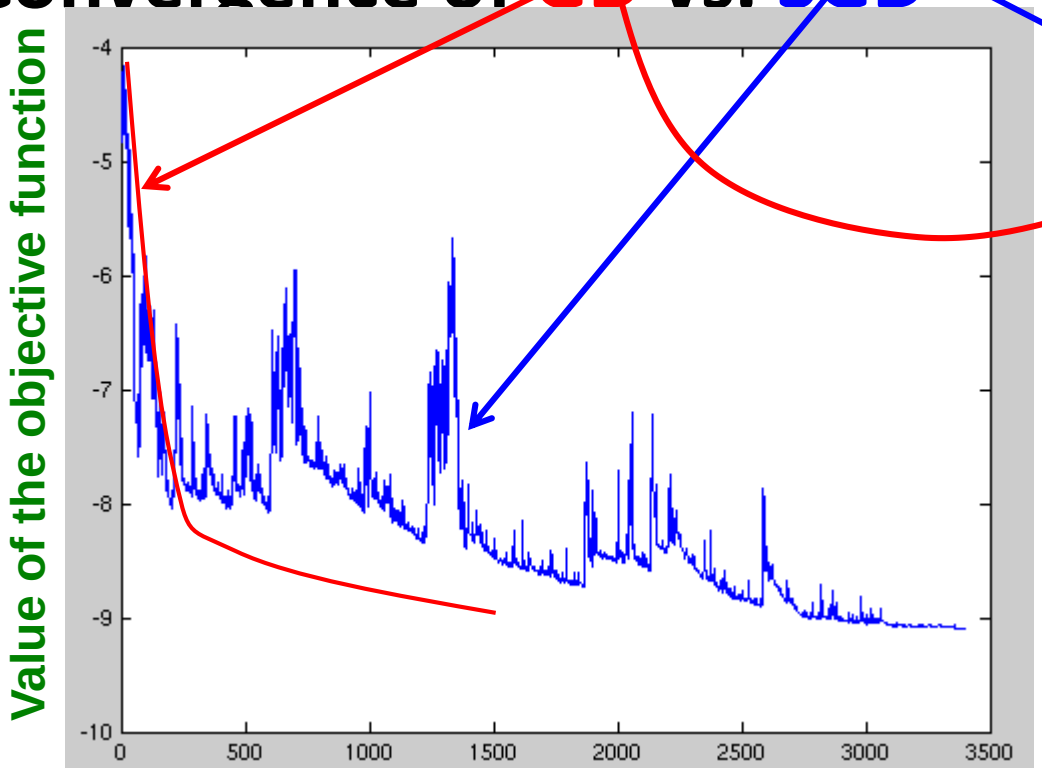
- **Gradient Descent (GD) vs. Stochastic GD**
  - **Observation:** $\nabla Q = [\nabla q_{if}]$ where

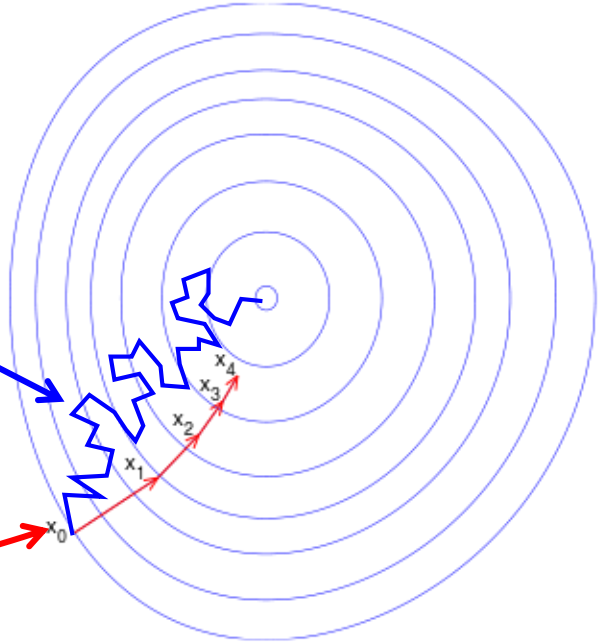
$$\nabla q_{if} = \sum_{x,i} -2\left(r_{xi} - q_{if}p_{xf}\right)p_{xf} + 2\lambda q_{if} = \sum_{x,i} \nabla \boldsymbol{Q}\,(\boldsymbol{r_{xi}})$$

    - Here $\boldsymbol{q_{if}}$ is entry $\boldsymbol{f}$ of row $\boldsymbol{q_i}$ of matrix $\boldsymbol{Q}$
  - $\boldsymbol{Q} = \boldsymbol{Q} - \eta\nabla\boldsymbol{Q} = \boldsymbol{Q} - \eta\left[\sum_{x,i} \nabla \boldsymbol{Q}\,(\boldsymbol{r_{xi}})\right]$
  - **Idea:** Instead of evaluating gradient over all ratings evaluate it for each individual rating and make a step
- **GD:** $\boldsymbol{Q} \leftarrow \boldsymbol{Q} - \eta\left[\sum_{\boldsymbol{r_{xi}}} \nabla \boldsymbol{Q}(\boldsymbol{r_{xi}})\right]$
- **SGD:** $\boldsymbol{Q} \leftarrow \boldsymbol{Q} - \mu\nabla \boldsymbol{Q}(\boldsymbol{r_{xi}})$
  - **Faster convergence!**
    - Need more steps but each step is computed much faster

SGD vs. GD

- **Convergence of GD vs. SGD**





GD improves the value of the objective function at every step.
SGD improves the value but in a "noisy" way.
GD takes fewer steps to converge but each step takes much longer to compute.
In practice, **SGD** is much faster!

Stochastic Gradient Descent

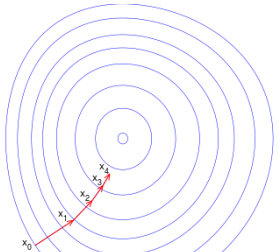
- **Stochastic gradient decent:**
  - Initialize ***P*** and ***Q*** (using SVD, pretend missing ratings are 0)
  - Then iterate over the ratings (multiple times if necessary) and update factors:

    For each r_{xi}:

    - $\varepsilon_{xi} = 2(r_{xi} - q_i \cdot p_x)$ (derivative of the "error")
    - $q_i \leftarrow q_i + \mu_1 (\varepsilon_{xi}\, p_x - \lambda_2\, q_i)$ (update equation)
    - $p_x \leftarrow p_x + \mu_2 (\varepsilon_{xi}\, q_i - \lambda_1\, p_x)$ (update equation)

- **2 for loops:**
  - For until convergence:
    - For each $\mathbf{r_{xi}}$
      - Compute gradient, do a "step"

μ … learning rate



Recap: Collaborative Filtering (CF)

- **Earliest and most popular collaborative filtering method**
- Derive unknown ratings from those of "**similar**" movies (item-item variant)
- Define **similarity measure s_{ij}** of items ***i*** and ***j***
- Select ***k***-nearest neighbors, compute the rating
  - ***N(i; x):*** items most similar to ***i*** that were rated by ***x***

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

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

Tip: Add Data

- **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
- **Add more data**
  - e.g., add IMDB data on genres
- **More data beats better algorithms**