Announcements

We will be holding group OH for each HW

- HW2 group OH was yesterday
- The recording and slides are uploaded to canvas and Ed
- We will announce the group OH for HW3/4

Note to other teachers and users of these slides: We would be delighted if you found our material useful for giving your own lectures. Feel free to use these slides verbatim, or to modify them to fit your own needs. If you make use of a significant portion of these slides in your own lecture, please include this message, or a link to our web site: http://www.mmds.org

Recommender Systems: Latent Factor Models

CS246: Mining Massive Datasets
Jure Leskovec, Stanford University
Mina Ghashami, Amazon

http://cs246.stanford.edu



The Netflix Prize

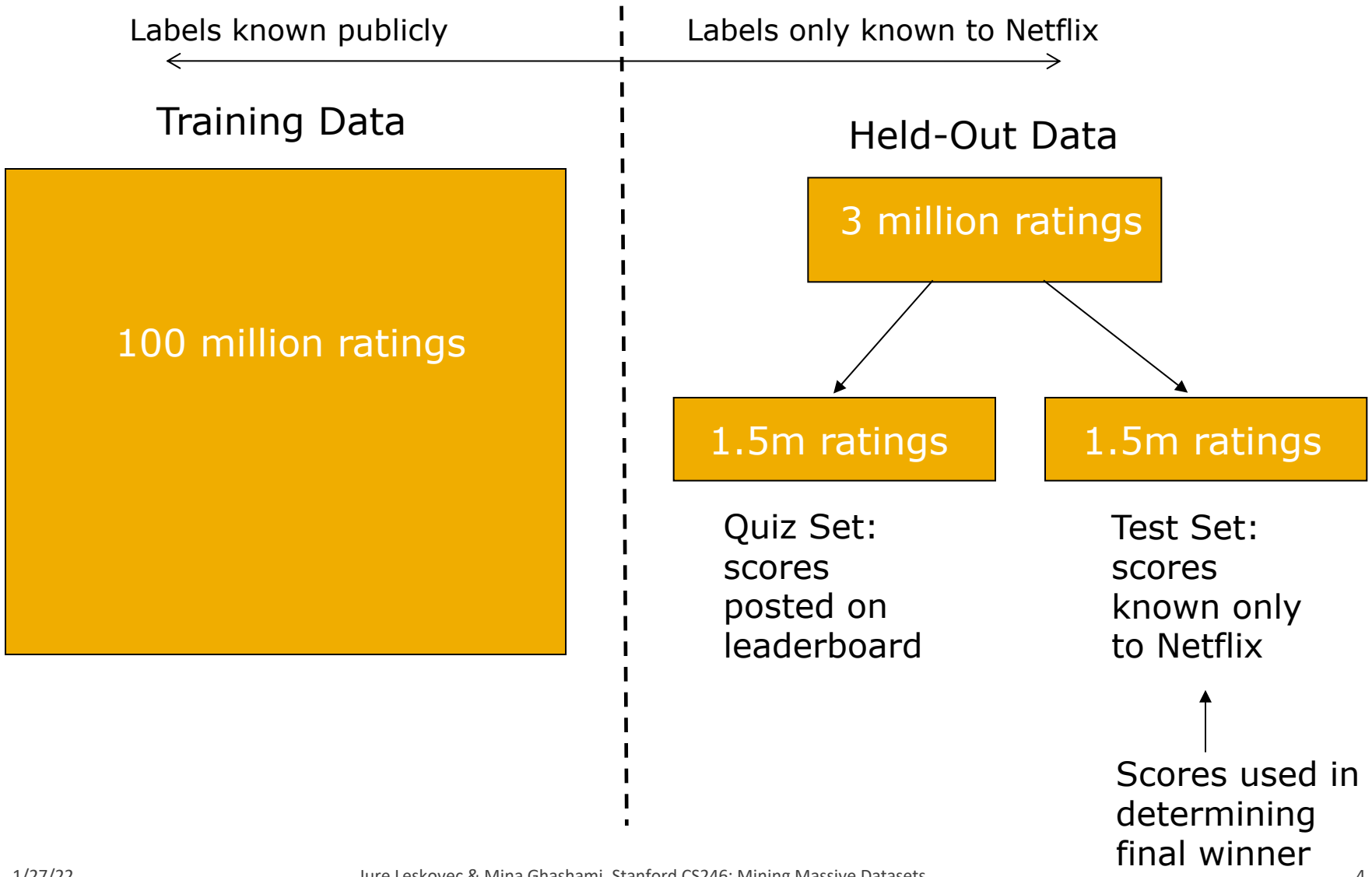
- **Training data**
  - 100 million ratings, 480,000 users, 17,770 movies
  - 6 years of data: 2000-2005
- **Test data**
  - Last few ratings of each user (2.8 million)
  - **Evaluation criterion:** Root Mean Square Error (RMSE)

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

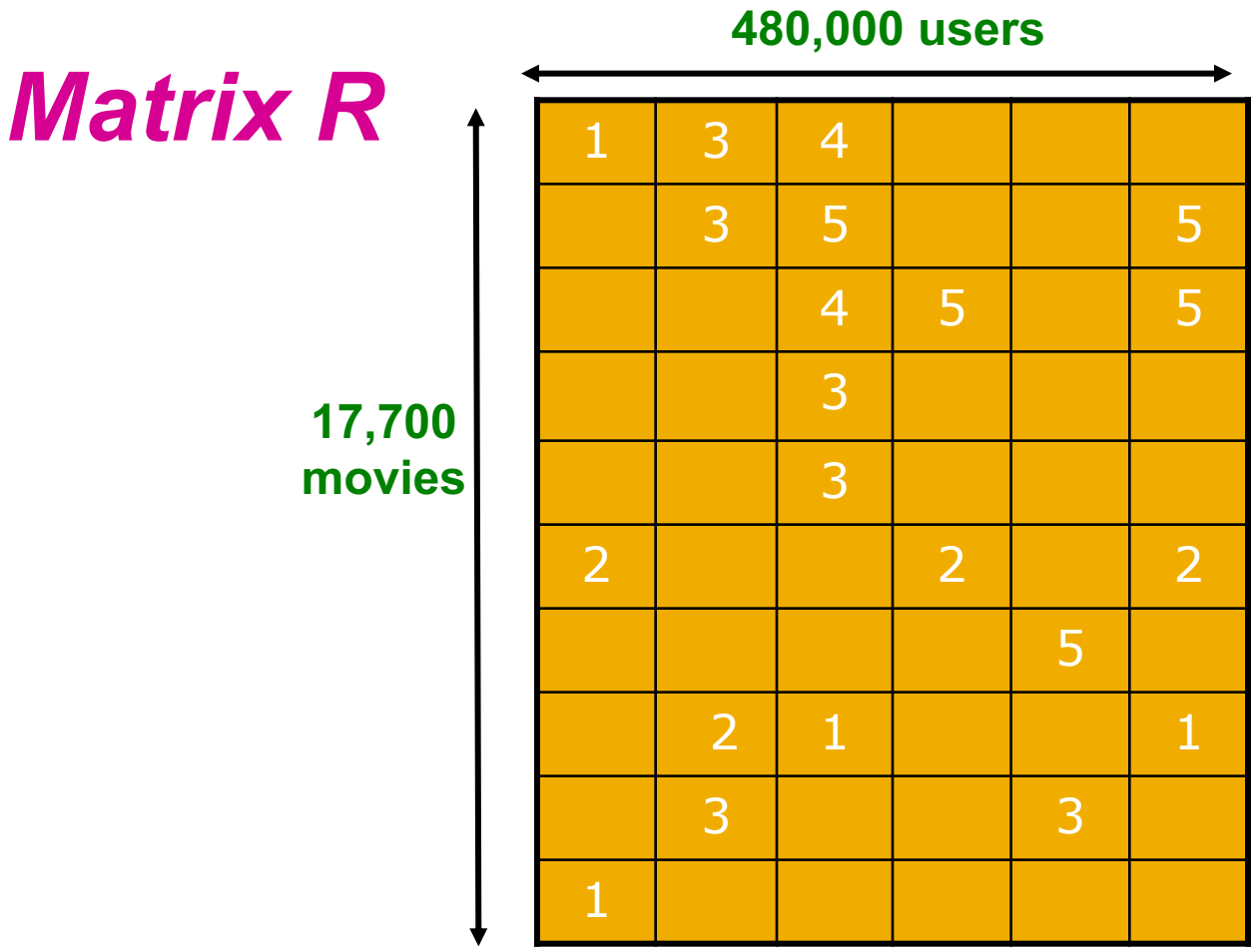
r_{xi}: true rating of user ***x*** on item ***i***

  - **Netflix's system RMSE: 0.9514**
- **Competition**
  - 2,700+ teams
  - **$1 million** prize for 10% improvement on Netflix

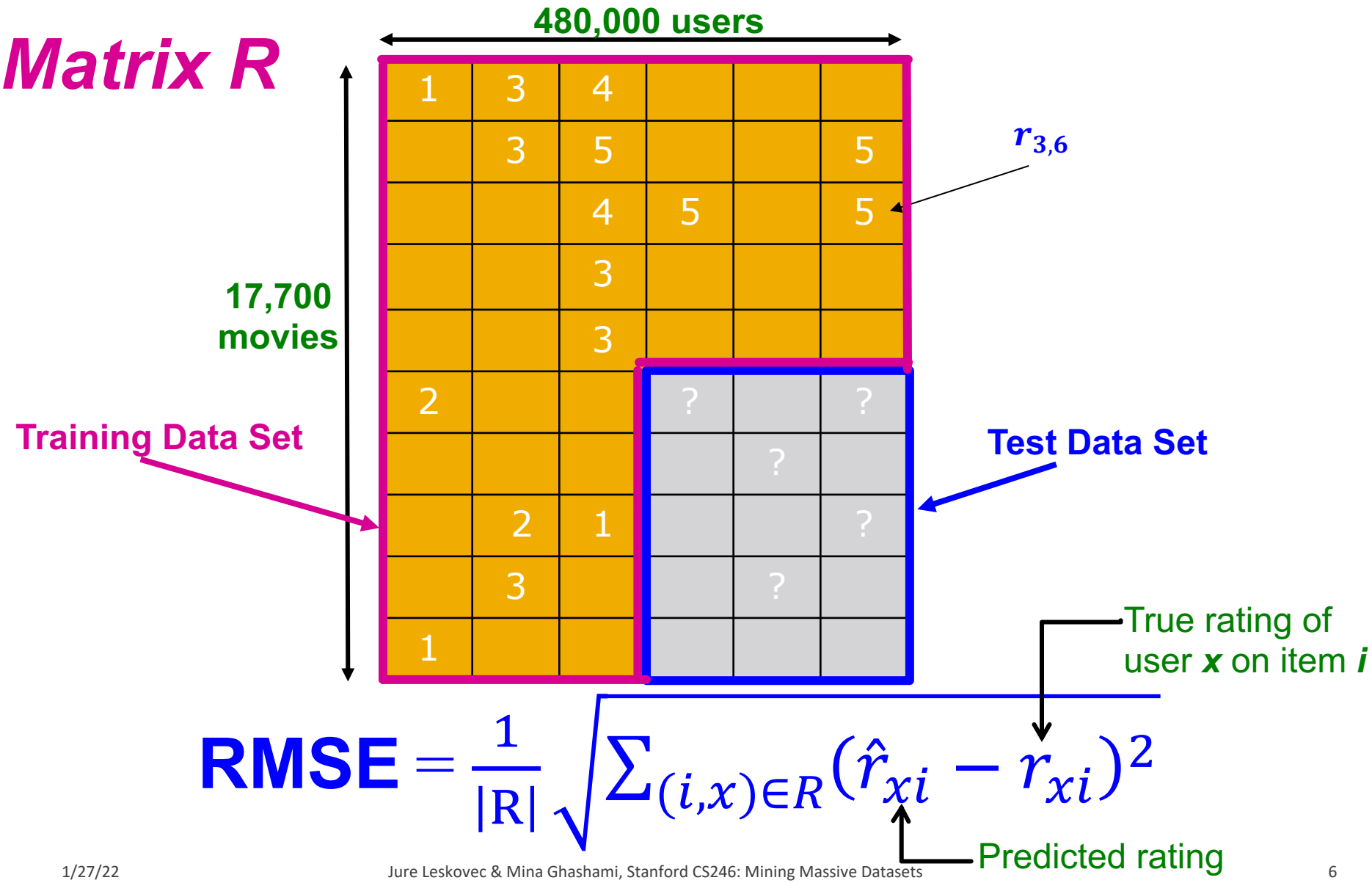
Competition Structure



The Netflix Utility Matrix *R*

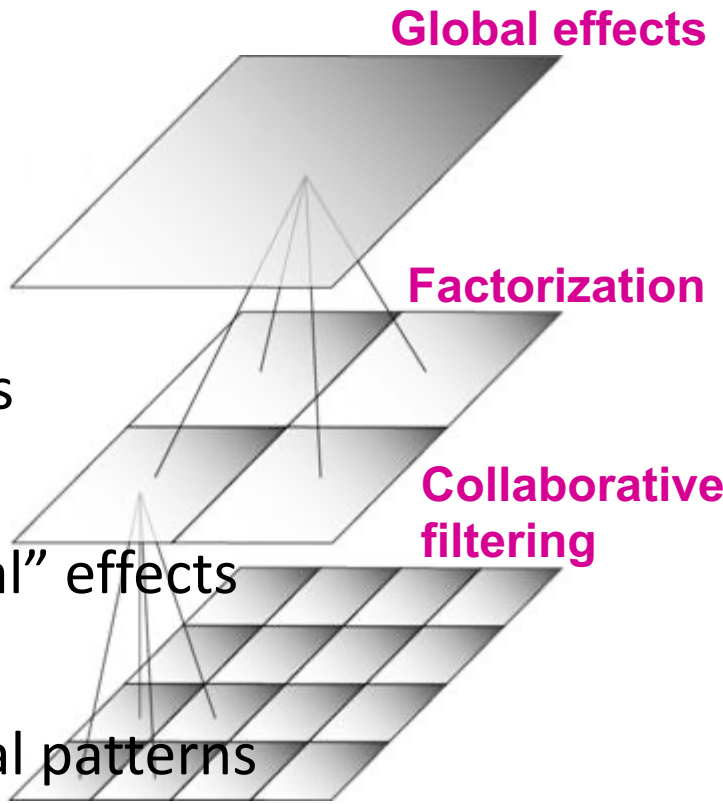


Utility Matrix *R*: Evaluation



BellKor Recommender System

- **The winner of the 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
  - **Regional:**
    - Factorization: Addressing "regional" effects
  - **Local:**
    - Collaborative filtering: Extract local patterns



Modeling Local & Global Effects

- **Global:**
  - Overall deviations of users/movies from average
    - Average movie rating: **3.7 stars**
    - *The Sixth Sense* is **0.5** stars above avg.
    - *Joe* rates **0.2** stars below avg.

      ⇒ Baseline estimation:

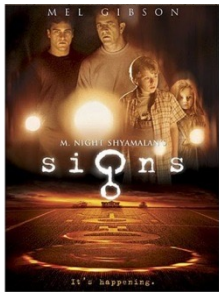
      ***Joe* will rate *The Sixth Sense* 4 stars**
    - **That is 4 = 3.7+0.5-0.2**





- **Local (CF/NN):**
  - *Joe* didn't like related/similar movie *Signs*
  - **⇒ Final estimate: based on CF**

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



Recap: Collaborative Filtering (CF)

- **Item-Item collaborative filtering method:**
  - Derive unknown ratings from "**similar**" movies
  - 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***

Recap: Collaborative Filtering (CF)

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

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

baseline estimate for r_{xi}

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

μ = overall avg. 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 $\boldsymbol{x}$ that are similar to movie $\boldsymbol{i}$
  - $\boldsymbol{w_{ij}}$ is the **interpolation weight** (some real number)
    - Note, 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 $\boldsymbol{x}$)

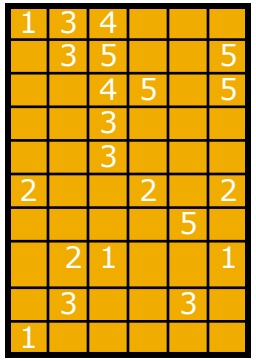
Idea: Interpolation Weights w_{ij}

- $\widehat{r_{xi}} = b_{xi} + \sum_{j \in N(i,x)} w_{ij}\left(r_{xj} - b_{xj}\right)$
- **How to set w_{ij}?**
  - Remember, error metric is: $\frac{1}{|R|}\sqrt{\sum_{(i,x)\in R}(\hat{r}_{xi} - r_{xi})^2}$

    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***

Why is this a good idea?

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 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

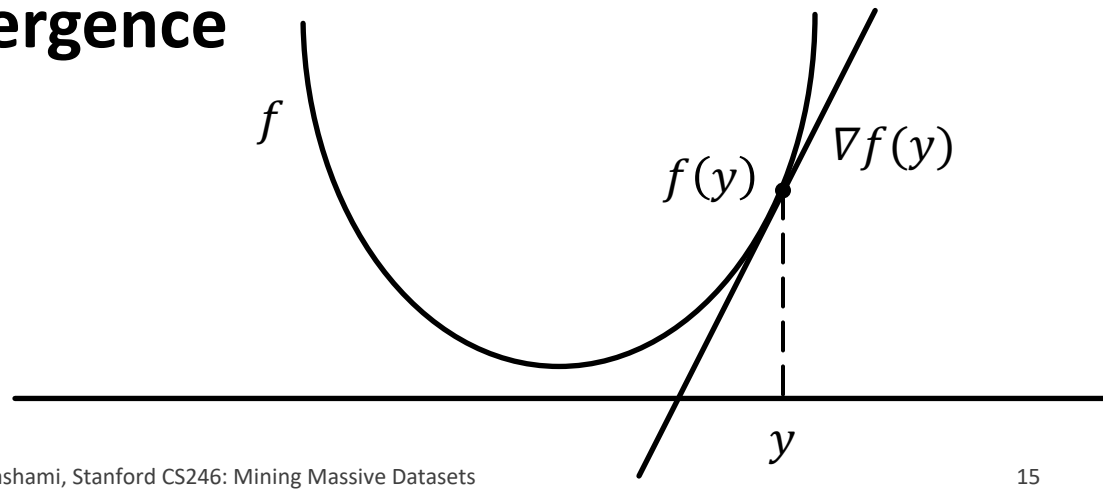
- Find w_{ij} that minimize **SSE** on **training data**!

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

- Think of w as a matrix of weights

Detour: Minimizing a function

- **A simple way to minimize a function $f(x)$: Gradient Descent:**
  - Compute the derivative $\nabla f(x)$
  - **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 convergence**



Interpolation Weights

- **The optimization problem is:**
  $$J(w) = \sum_{x,i \in R} \left(\left[b_{xi} + \sum_{j \in N(i;x)} w_{ij} \left(r_{xj} - b_{xj} \right) \right] - r_{xi} \right)^2$$
  We apply gradient descent:
  - **Iterate until convergence:** $\boldsymbol{w \leftarrow w - \eta \nabla_w J}$ η … learning rate

  where $\boldsymbol{\nabla_w J}$ is the gradient (derivative evaluated on data):

$$\nabla_w J = \left[\frac{\partial J(w)}{\partial w_{ij}} \right]$$

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

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

else $\boldsymbol{\frac{\partial J(w)}{\partial w_{ij}} = 0}$

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

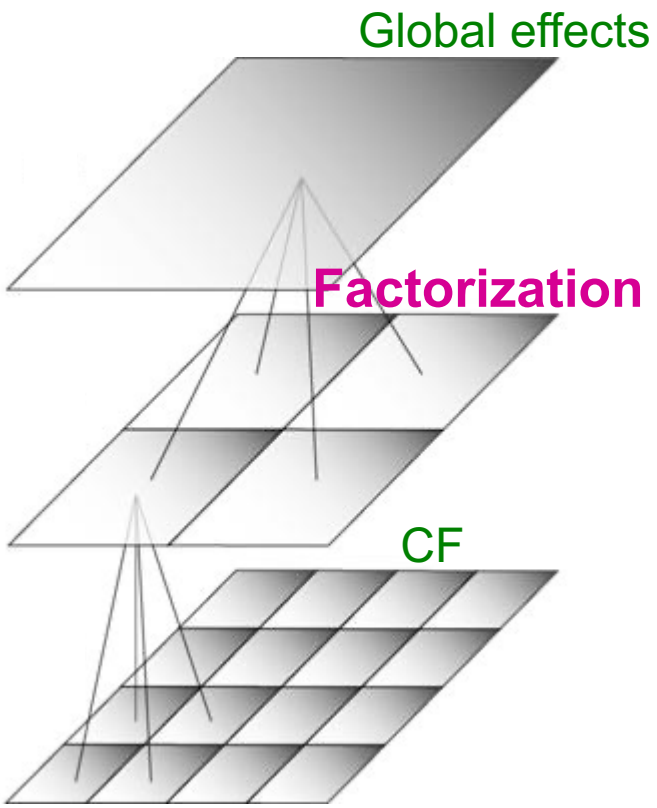
while $|w_{new} - w_{old}| > \varepsilon$:

$w_{old} = w_{new}$

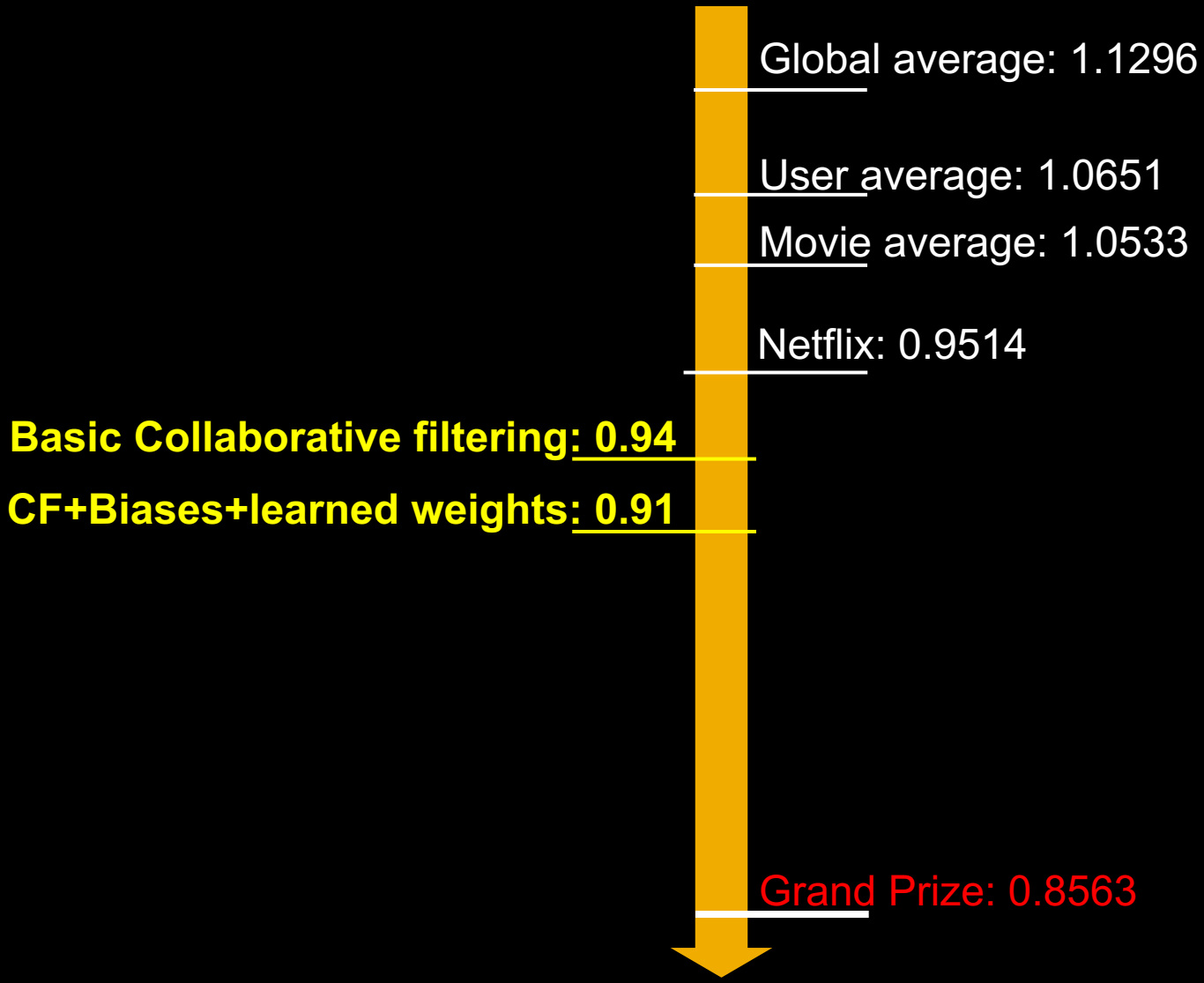
$w_{new} = w_{old} - \eta \cdot \nabla w_{old}$

Interpolation Weights

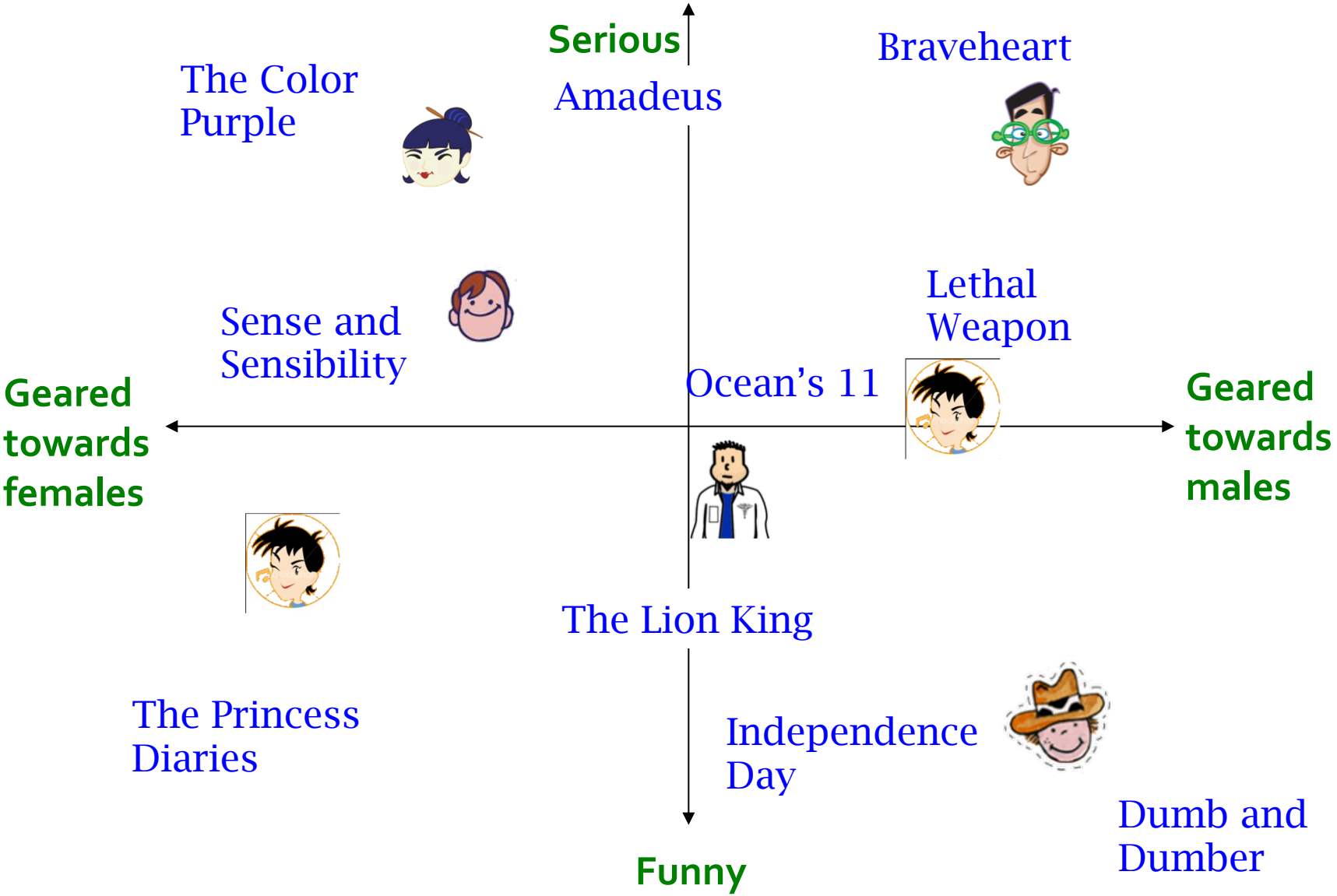
- **So far:** $\widehat{r_{xi}} = b_{xi} + \sum_{j \in N(i;x)} w_{ij}\left(r_{xj} - b_{xj}\right)$
  - Weights w_{ij} derived based on their roles; **no use of an arbitrary similarity measure** ($w_{ij} \neq s_{ij}$)
  - Explicitly account for interrelationships among the neighboring movies
- **Next: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



Performance of Various Methods



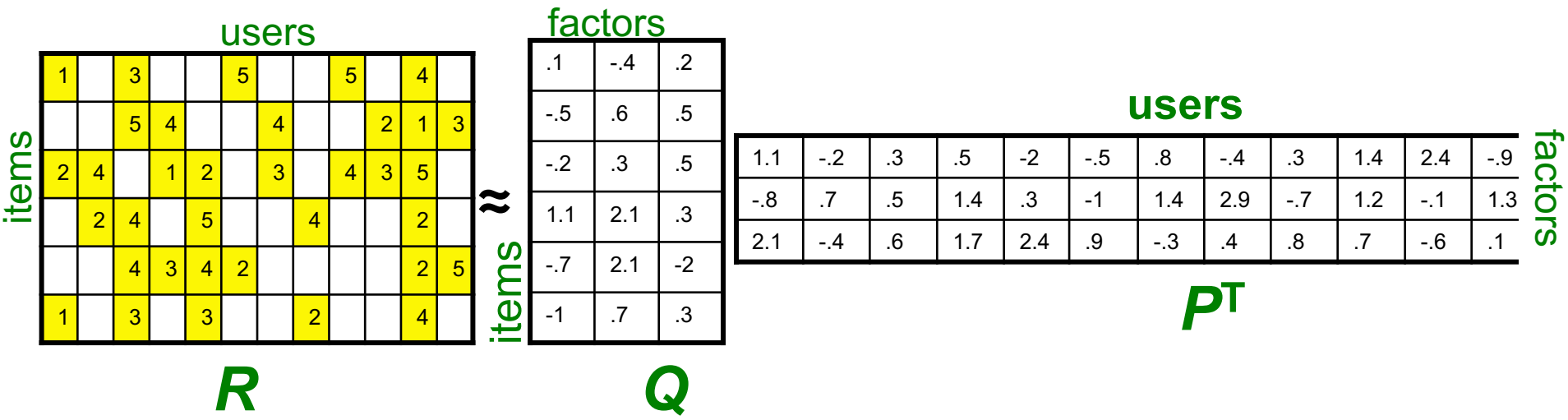
Latent Factor Models (e.g., SVD)



Latent Factor Models

SVD: $A = U \Sigma V^T$

- "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	-.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

- 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 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*?**

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

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

users

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

≈

items

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

factors

Q

•

users

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

P^T

Ratings as Products of Factors

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

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

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

users

items

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

≈

items

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

factors

Q

•

users

factors

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

P^T

Ratings as Products of Factors

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

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

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	

≈

items

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

f factors

Q

•

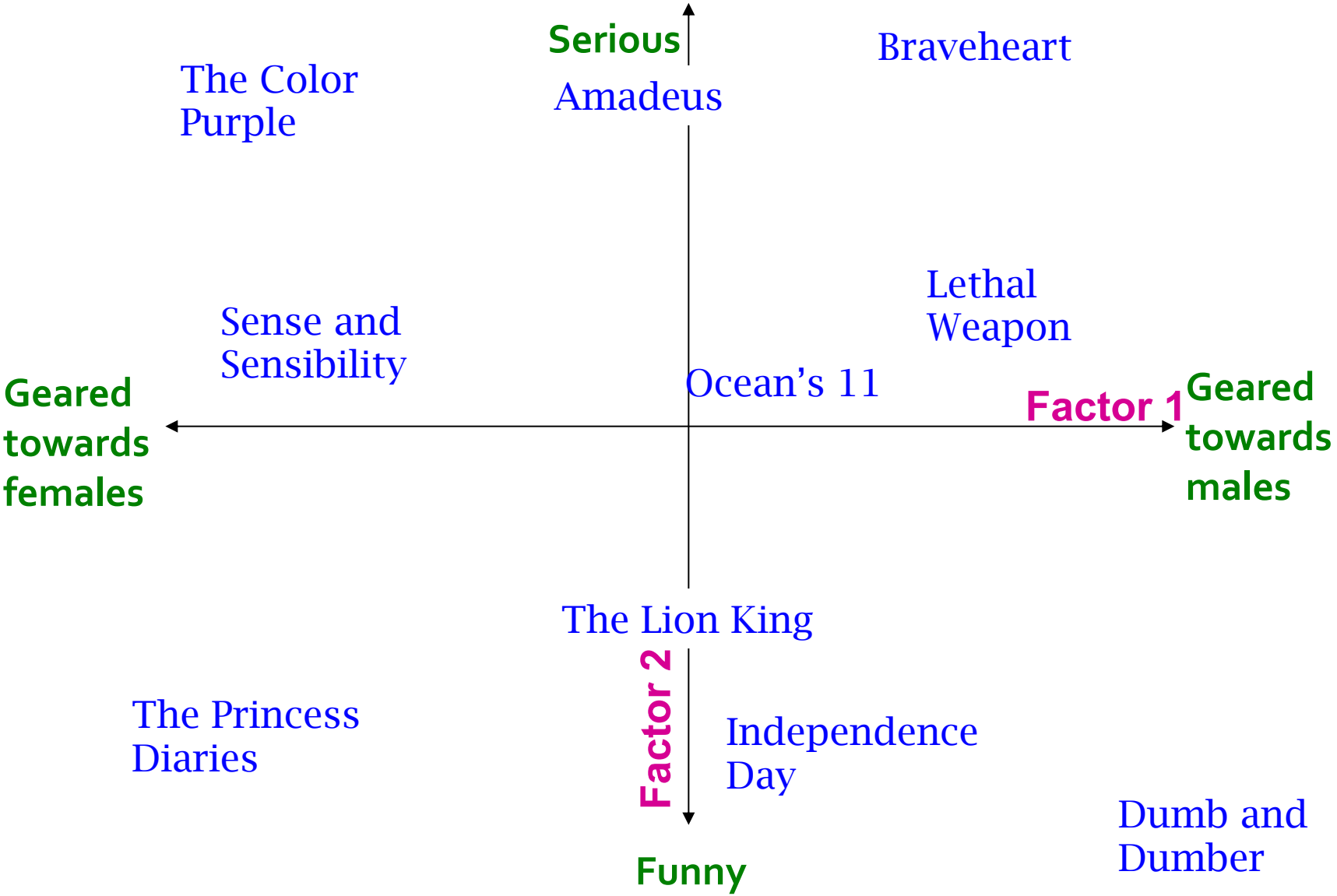
users

f factors

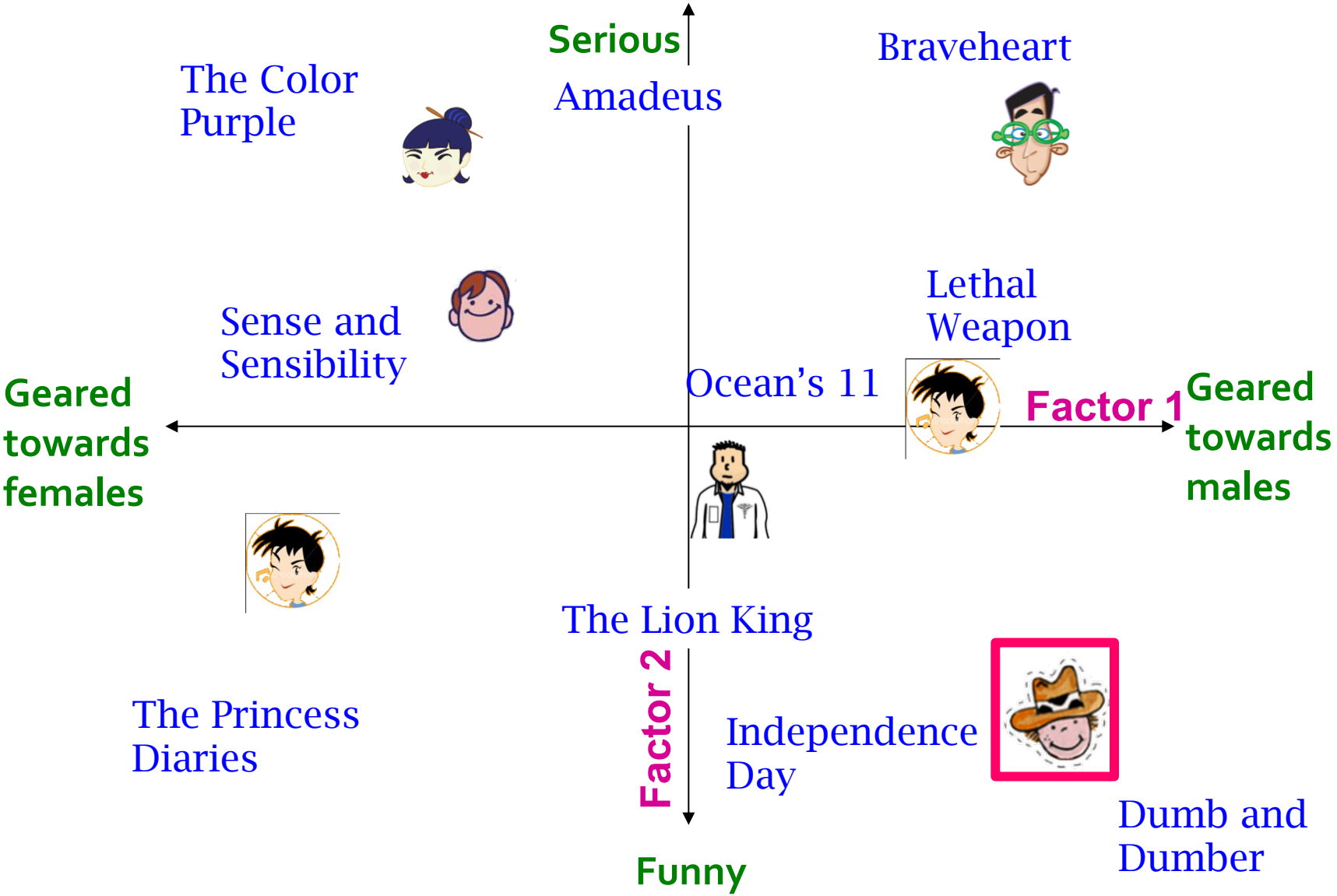
1.1	-.2	.3	.5	**-2**	-.5	.8	-.4	.3	1.4	2.4	-.9
-.8	.7	.5	1.4	**.3**	-1	1.4	2.9	-.7	1.2	-.1	1.3
2.1	-.4	.6	1.7	**2.4**	.9	-.3	.4	.8	.7	-.6	.1

P^T

Latent Factor Models

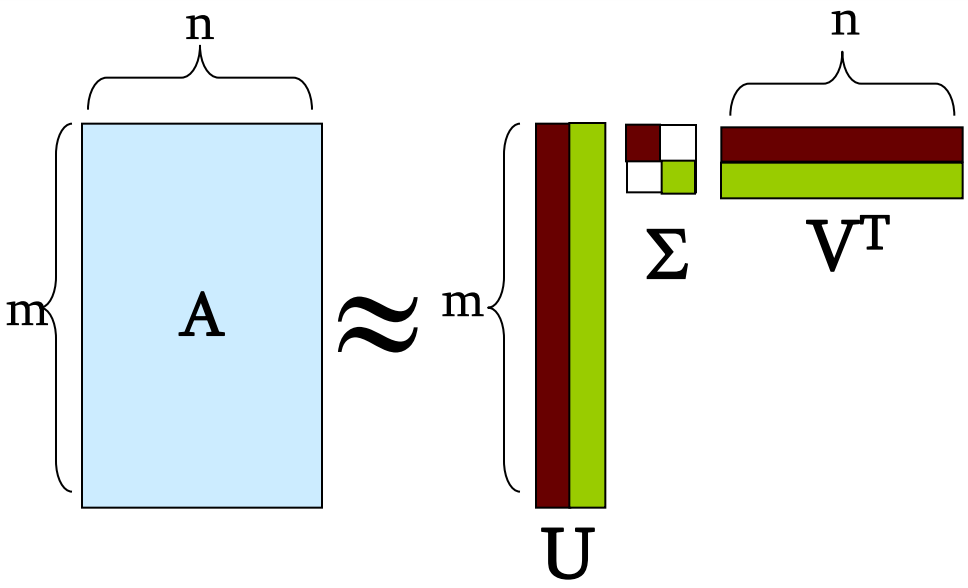


Latent Factor Models



Recap: SVD

- **Remember SVD:**
  - **A**: Input data matrix
  - **U**: Left singular vecs
  - **V**: Right singular vecs
  - **Σ**: Singular values



- **So in our case:**

  "SVD" on Netflix data: $R \approx Q \cdot P^T$

  $A = R,\ Q = U,\ P^T = \Sigma V^T$

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

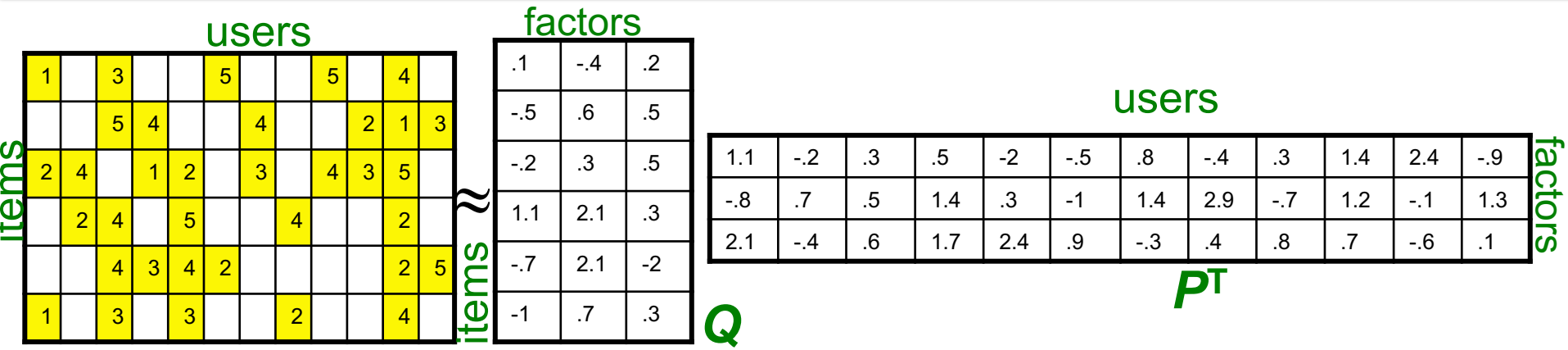
SVD: More good stuff

- **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



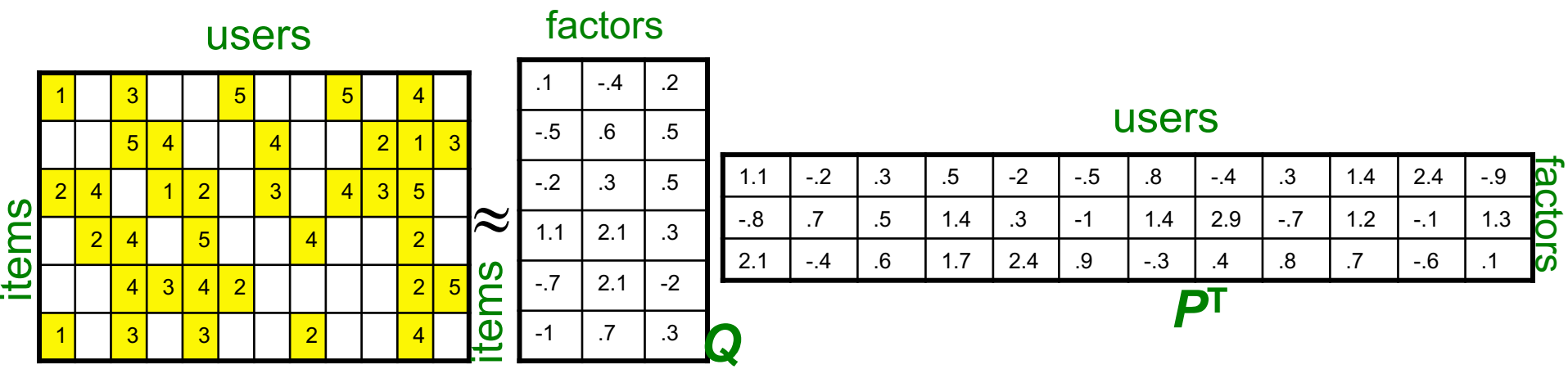
- **SVD isn't defined when entries are missing!**
- **Use specialized methods to find *P*, *Q***
  - $\min_{P,Q} \sum_{(i,x)\in R} \left(r_{xi} - q_i \cdot p_x\right)^2$ $\qquad \hat{r}_{xi} = q_i \cdot p_x$
  - **Note:**
    - We don't require cols of ***P, Q*** to be orthogonal/unit length
    - ***P,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

Finding the Latent Factors

Latent Factor Models

- **Objective function: find P and Q such that:**

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



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	

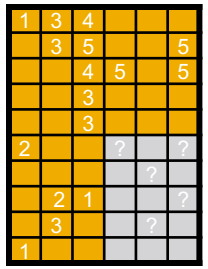
.1	-.4	.2
-.5	.6	.5
-.2	.3	.5
1.1	2.1	.3
-.7	2.1	-2
-1	.7	.3

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

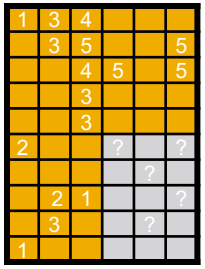
Back to Our Problem

- **Goal: 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, the model fits the training data too well and is thus **not generalizing** well to unseen test data



Dealing with Missing Entries



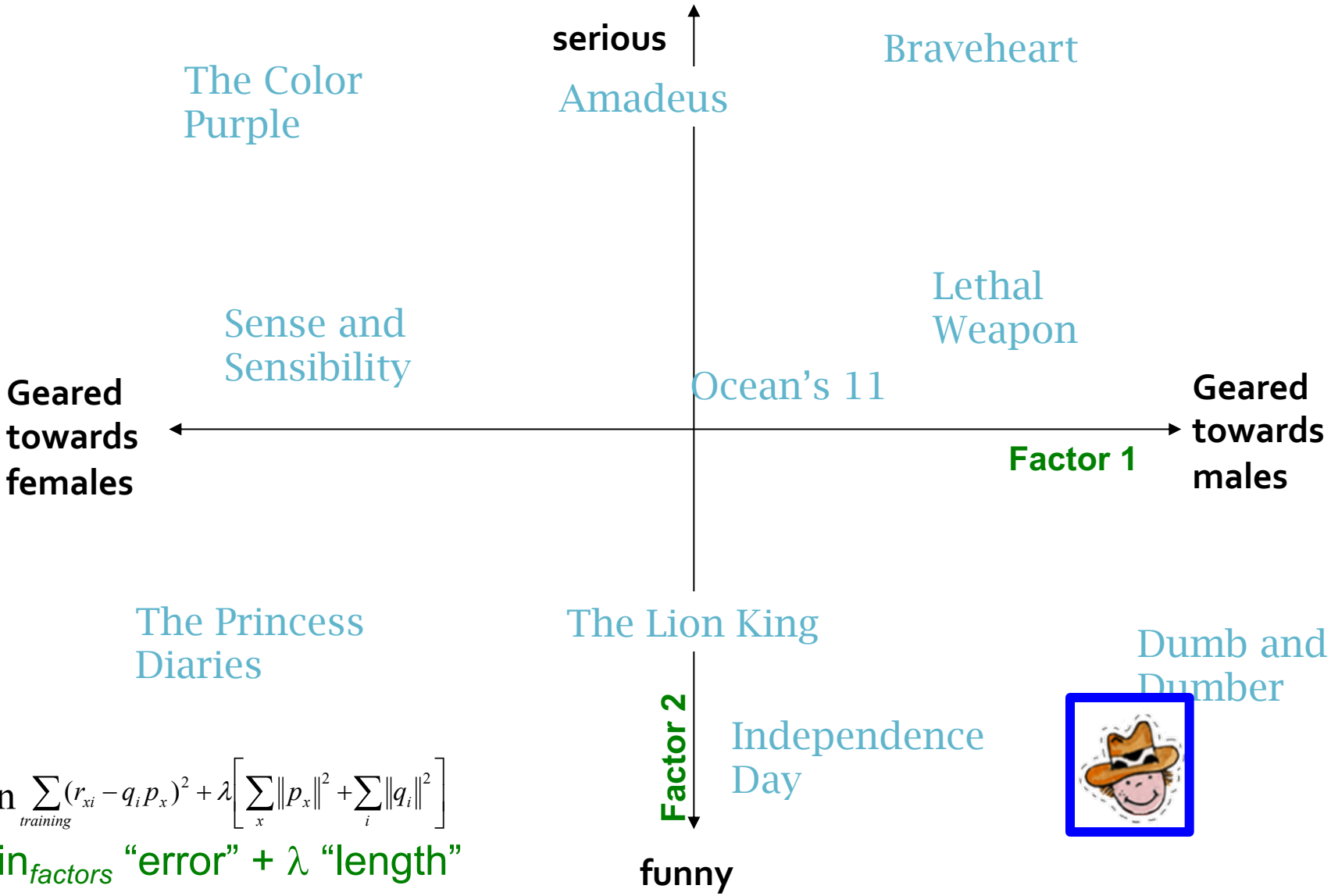
- **To prevent overfitting we introduce regularization:**
  - Allow rich model where there is sufficient data
  - Shrink aggressively where data is scarce

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

$\lambda_1, \lambda_2 \ldots$ hyperparameters

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

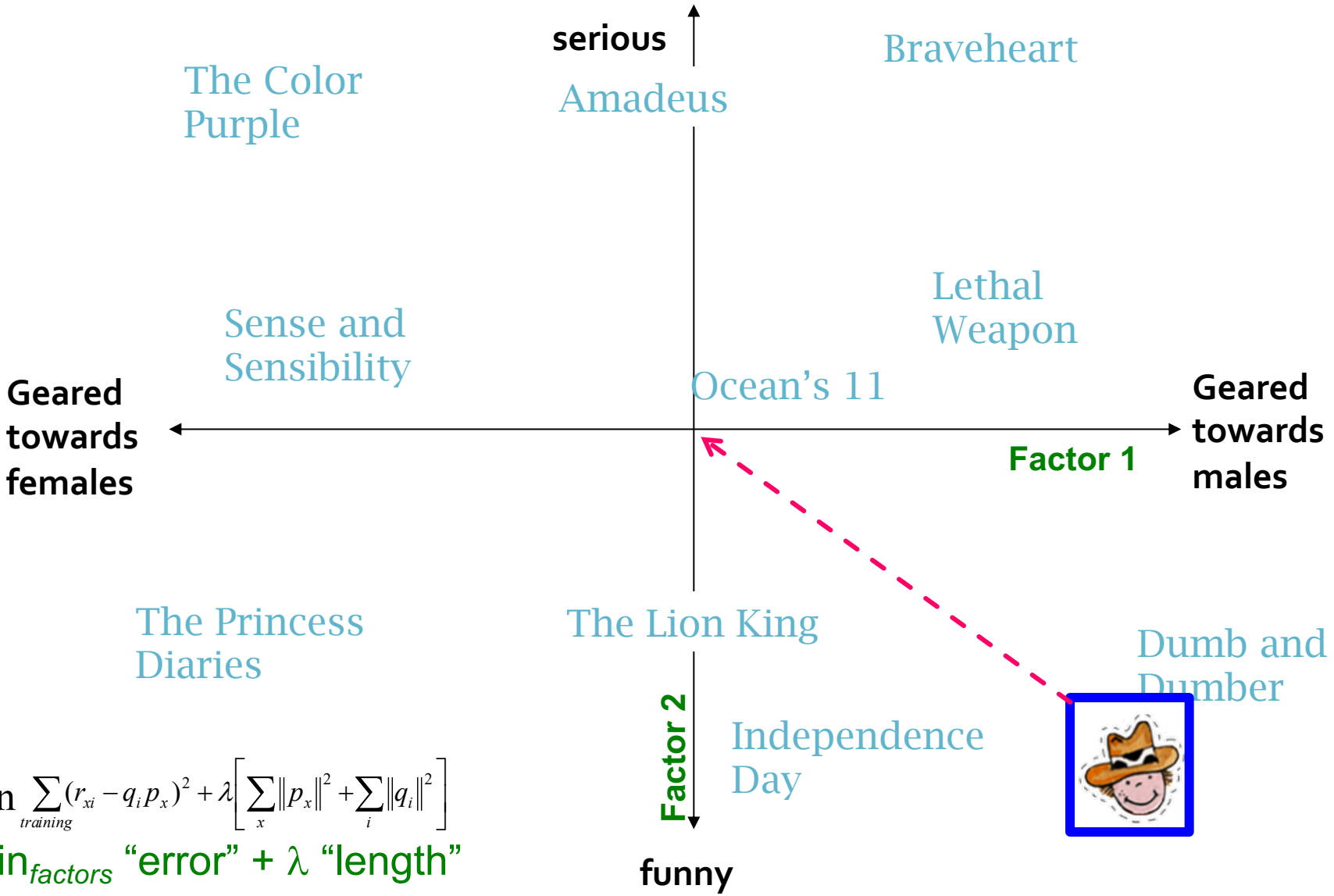
The Effect of Regularization



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

$\min_{factors}$ "error" + λ "length"

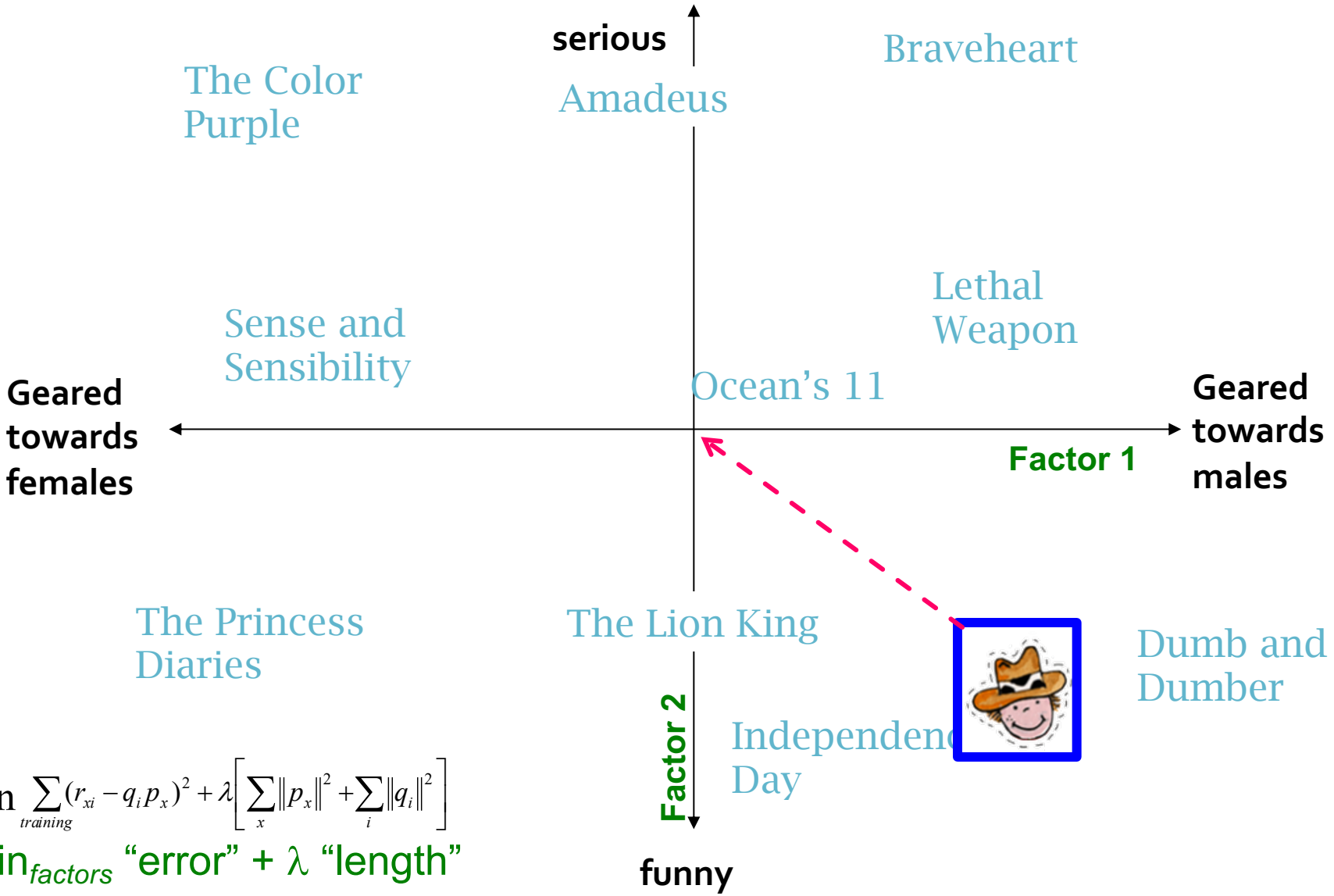
The Effect of Regularization



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

$\min_{factors}$ "error" + λ "length"

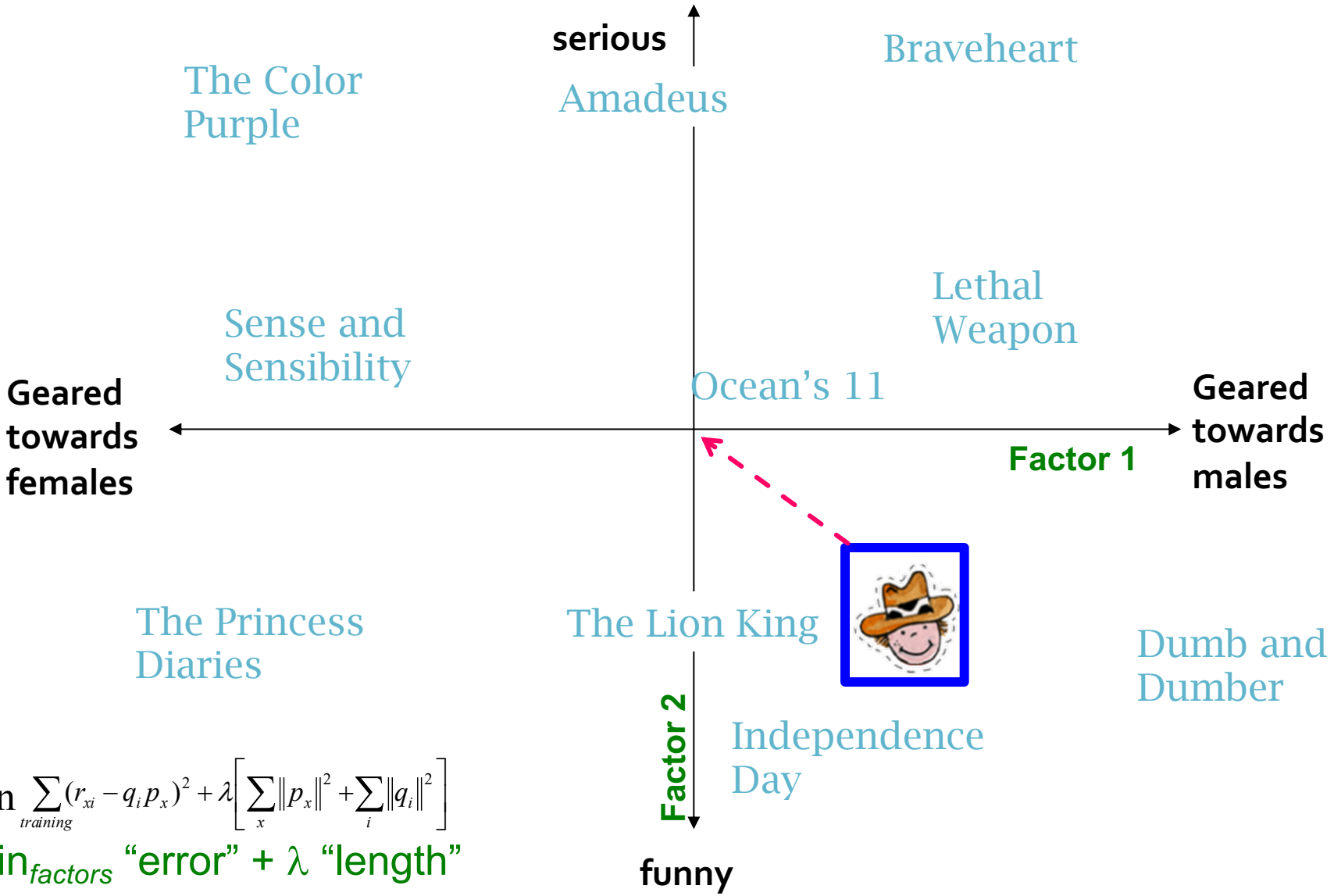
The Effect of Regularization



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

$\min_{factors}$ "error" + λ "length"

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"

How to solve new objective function?

- **Our objective function is:**

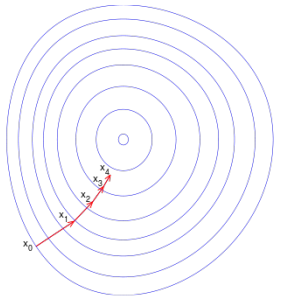
$$J(P,Q) = \sum_{training} (r_{xi} - q_i p_x)^2 + \left[\lambda_1 \sum_x \|p_x\|^2 + \lambda_2 \sum_i \|q_i\|^2\right]$$

- **Variables are:**

$$P = \begin{bmatrix} p_{11} & \cdots & p_{1k} \\ \vdots & \ddots & \vdots \\ p_{n1} & \cdots & p_{nk} \end{bmatrix}, \mathrm{Q} = \begin{bmatrix} q_{11} & \cdots & q_{1k} \\ \vdots & \ddots & \vdots \\ q_{m1} & \cdots & q_{mk} \end{bmatrix}$$

- **We use Gradient Descent to find optimal values of *P* and *Q***

Gradient Descent



- **Gradient descent:**
  - **Initialize *P* and *Q*** (using SVD, pretend missing ratings are 0)
  - **Do gradient descent on objective function *J(P,Q)***:
    - $P \leftarrow P - \eta \cdot \nabla_P J$
    - $Q \leftarrow Q - \eta \cdot \nabla_Q J$
  - Since ***P*** and ***Q*** are matrices, we perform the update step on every entry independently:

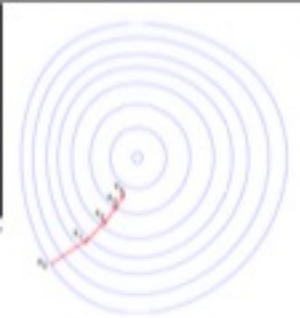
    Ex: for entry at row ***i***, column ***f*** of matrix ***Q***

$$q_{if} = q_{if} - \eta \nabla_{q_{if}} J$$

$$\nabla_{q_{if}} J = \sum_{x:(x,i) \in training} -2\left(r_{xi} - q_i p_x\right) p_{xf} + 2\lambda_2 q_{if}$$

  - **Observation: Computing gradients is slow!**

Stochastic Gradient Descent



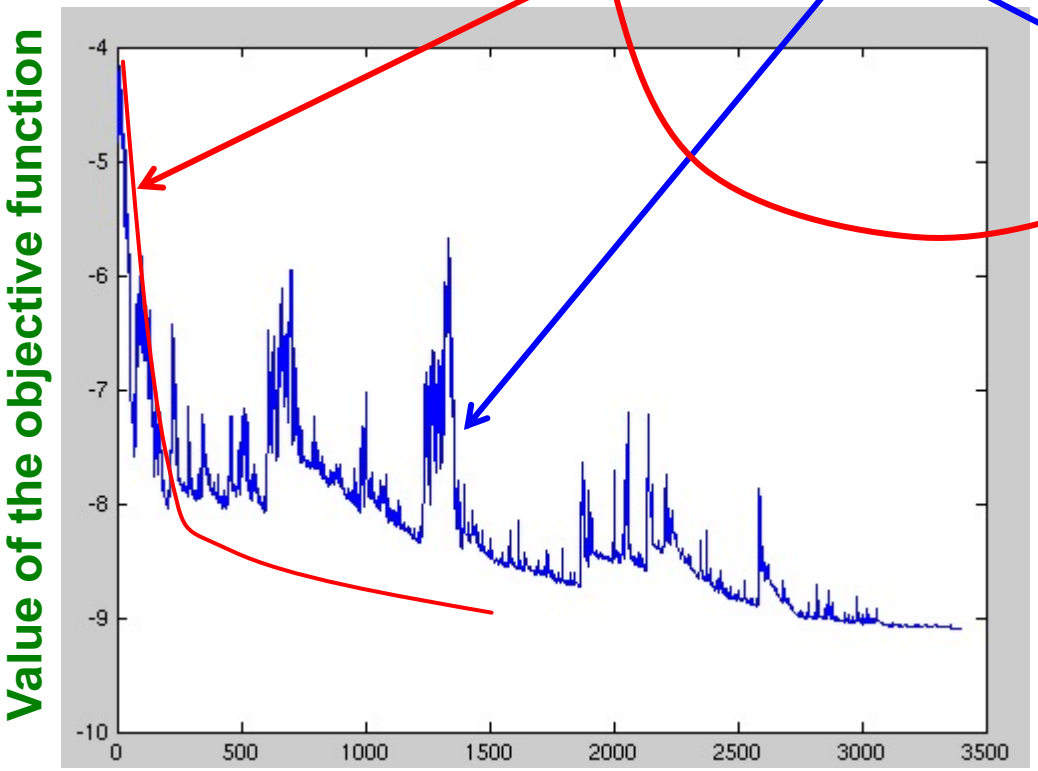
- **Gradient Descent (GD) vs. Stochastic GD**
  - **Observation:** $\nabla_Q J = [\nabla q_{if}]$ where

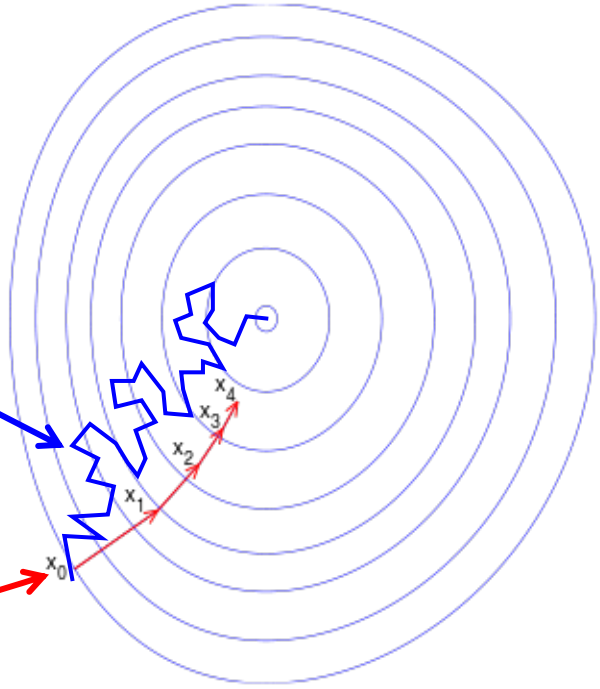
$$\nabla_{q_{if}} J = \sum_{x:(x,i)\in training} -2(r_{xi} - q_{if}p_{xf})p_{xf} + 2\lambda q_{if} = \sum_{x:(x,i)\in training} \nabla \boldsymbol{Q}(\boldsymbol{r_{xi}})$$

  - **Idea:** Instead of evaluating gradient over all ratings evaluate it on one 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

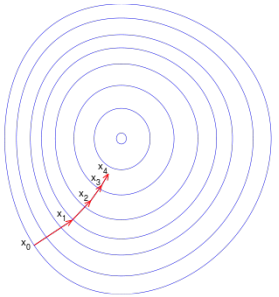
- **Convergence of GD vs. SGD**





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

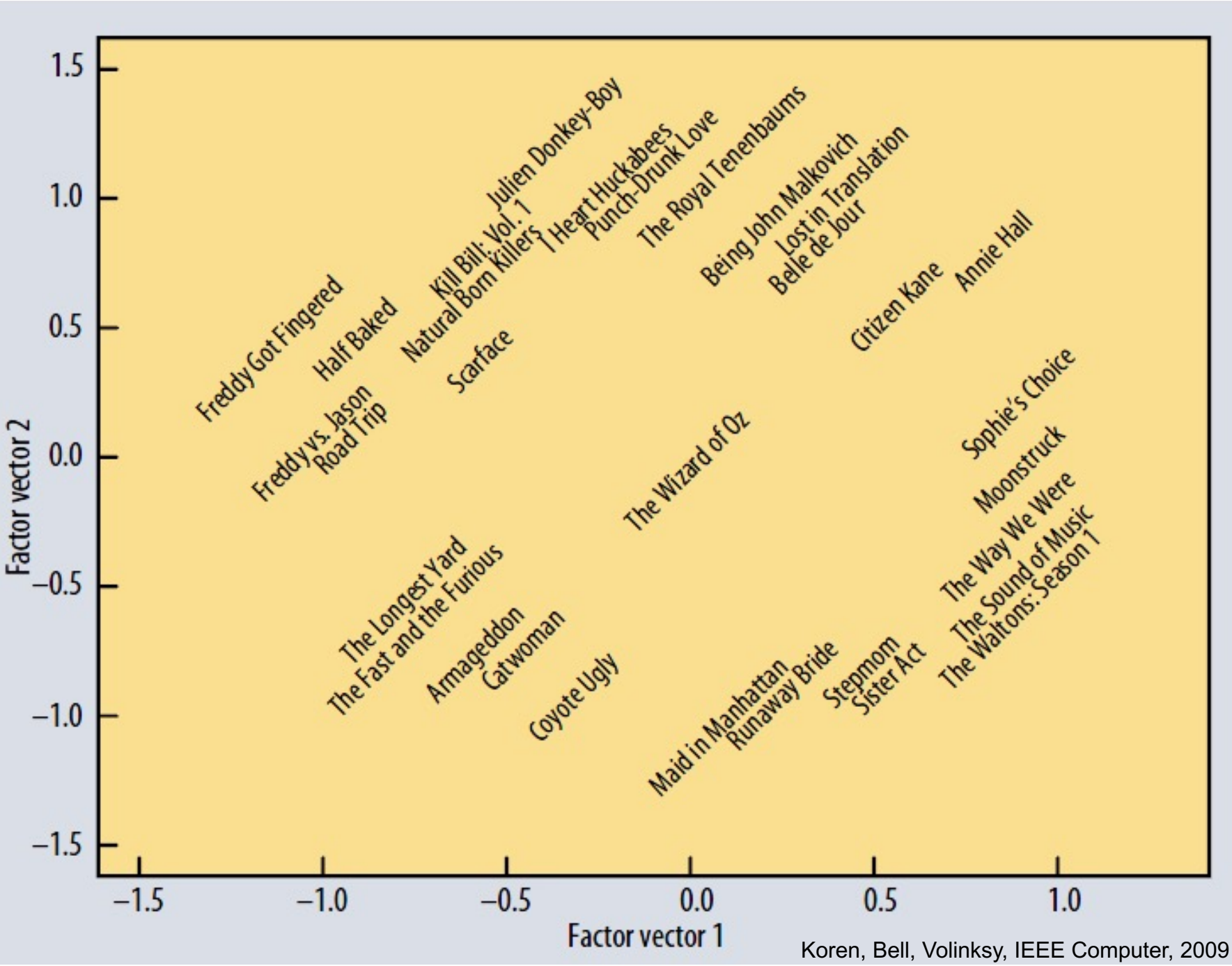


- **Stochastic gradient descent:**
  - Initialize ***P*** and ***Q*** (using SVD, pretend missing ratings are 0)
  - Then iterate over the ratings (multiple times if necessary) and update factors:

    For each r_{xi}:

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

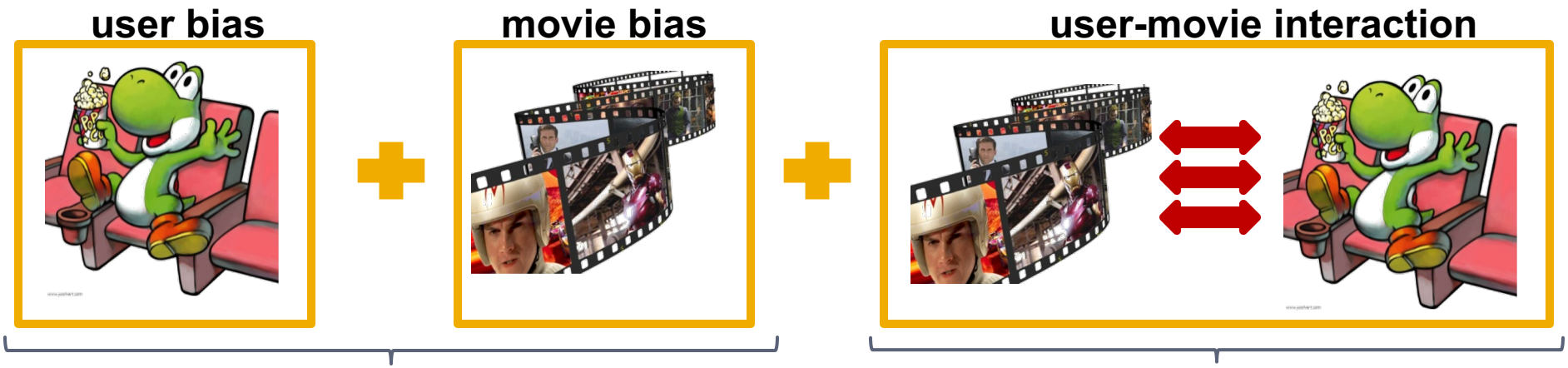
    μ … learning rate
- **Two For loops:**
  - For until convergence:
    - For each $\mathbf{r_{xi}}$
      - Compute gradient, do a "step" as above



Koren, Bell, Volinksy, IEEE Computer, 2009

Extending Latent Factor Model to Include Biases

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

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")

Putting It All 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 mean rating is 1 star lower than the mean: $b_x = -1$
  - Star Wars gets a mean rating of *0.5* higher than average movie: $b_i = +0.5$
  - Predicted rating for you on Star Wars: $= 3.7 - 1 + 0.5 = 3.2$
  - ***Final score*** $= 3.2 + q_i \cdot p_x$

Fitting the New Model

- **Solve:**

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

goodness of fit

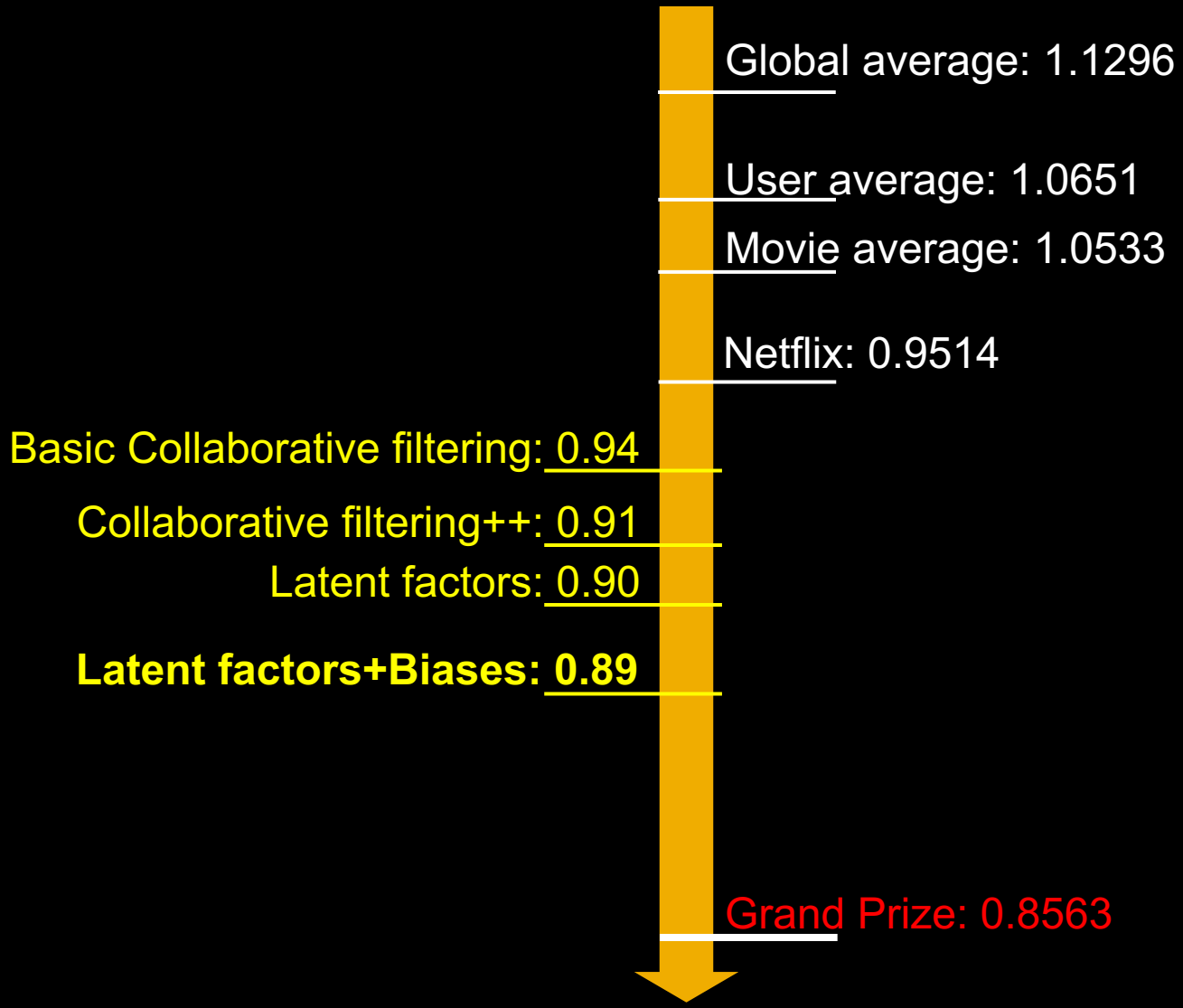
$$+\left(\lambda_1 \sum_i \|q_i\|^2 + \lambda_2 \sum_x \|p_x\|^2 + \lambda_3 \sum_x \|b_x\|^2 + \lambda_4 \sum_i \|b_i\|^2\right)$$

regularization

λ is selected via grid-search on a validation set

- **Stochastic gradient descent to find parameters**
  - **Note:** Both biases b_x, b_i as well as interactions q_i, p_x are treated as parameters (and we learn them)

Performance of Various Methods



The Netflix Challenge: 2006-09

Temporal Biases Of Users

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

Temporal Biases & Factors

- **Original model:**

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

- **Add time dependence to biases:**

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

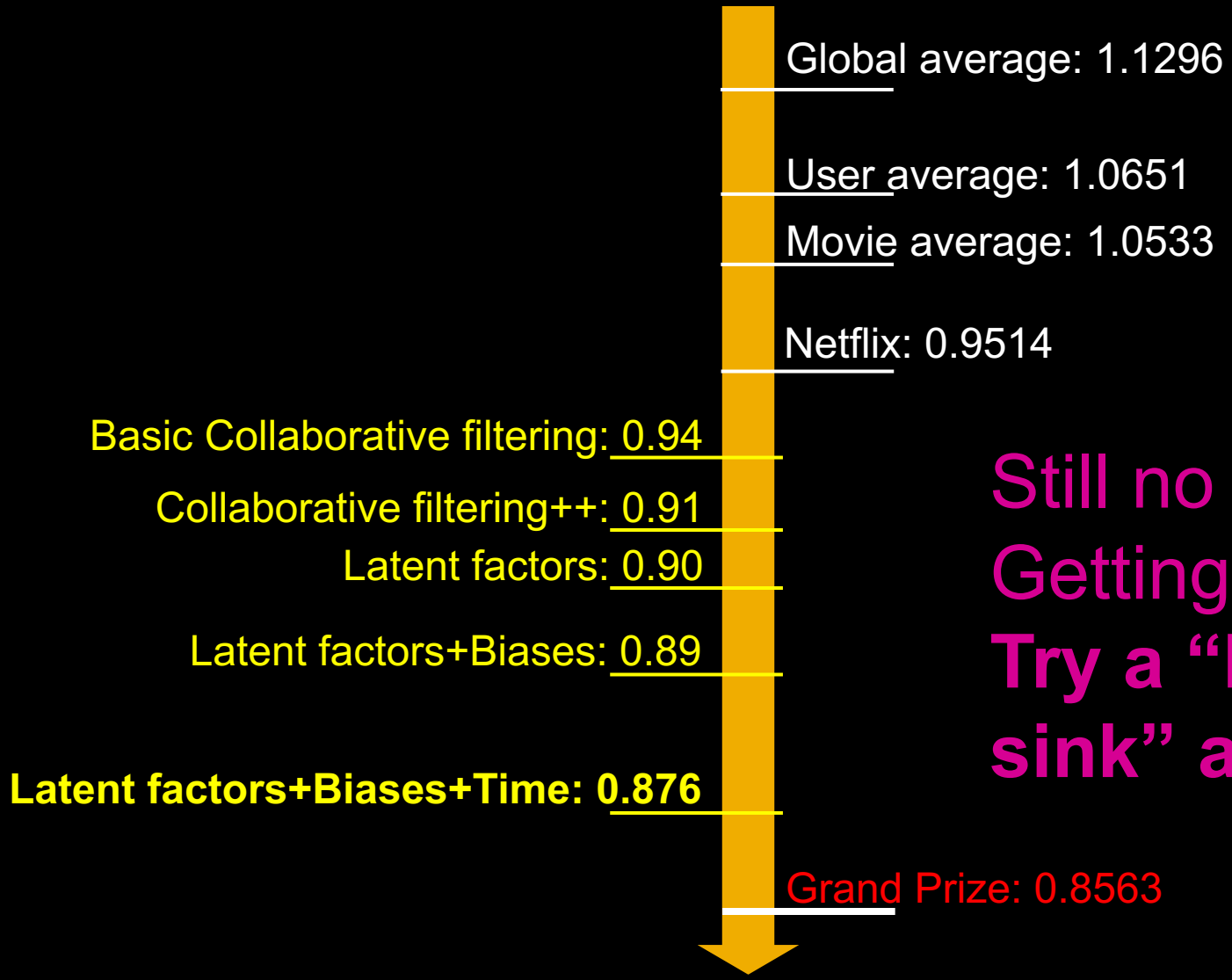
  - Make parameters b_x and b_i to depend on time
  - **(1)** Parameterize time-dependence by linear trends
    (2) Each bin corresponds to 10 consecutive weeks

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

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

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

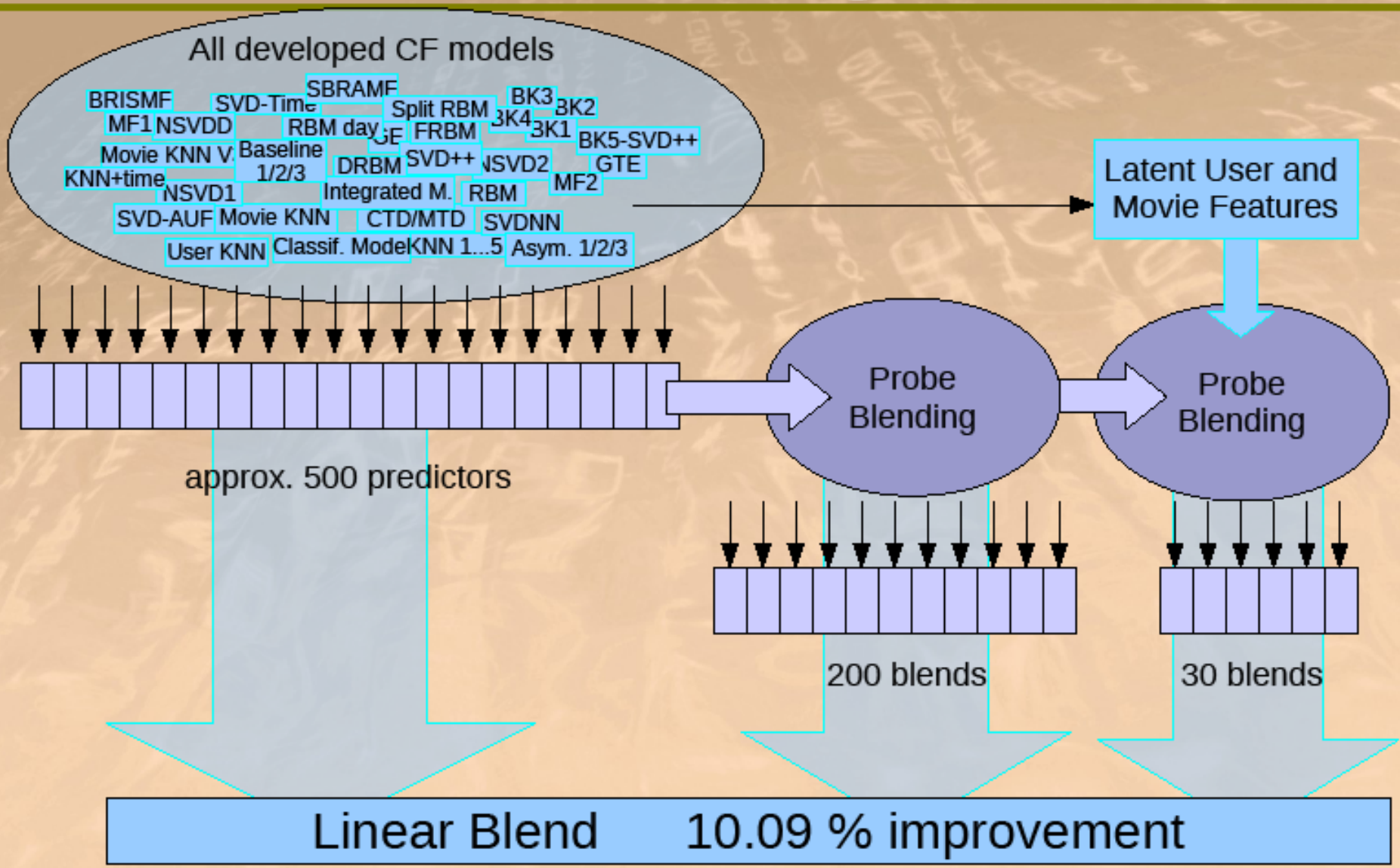
Performance of Various Methods



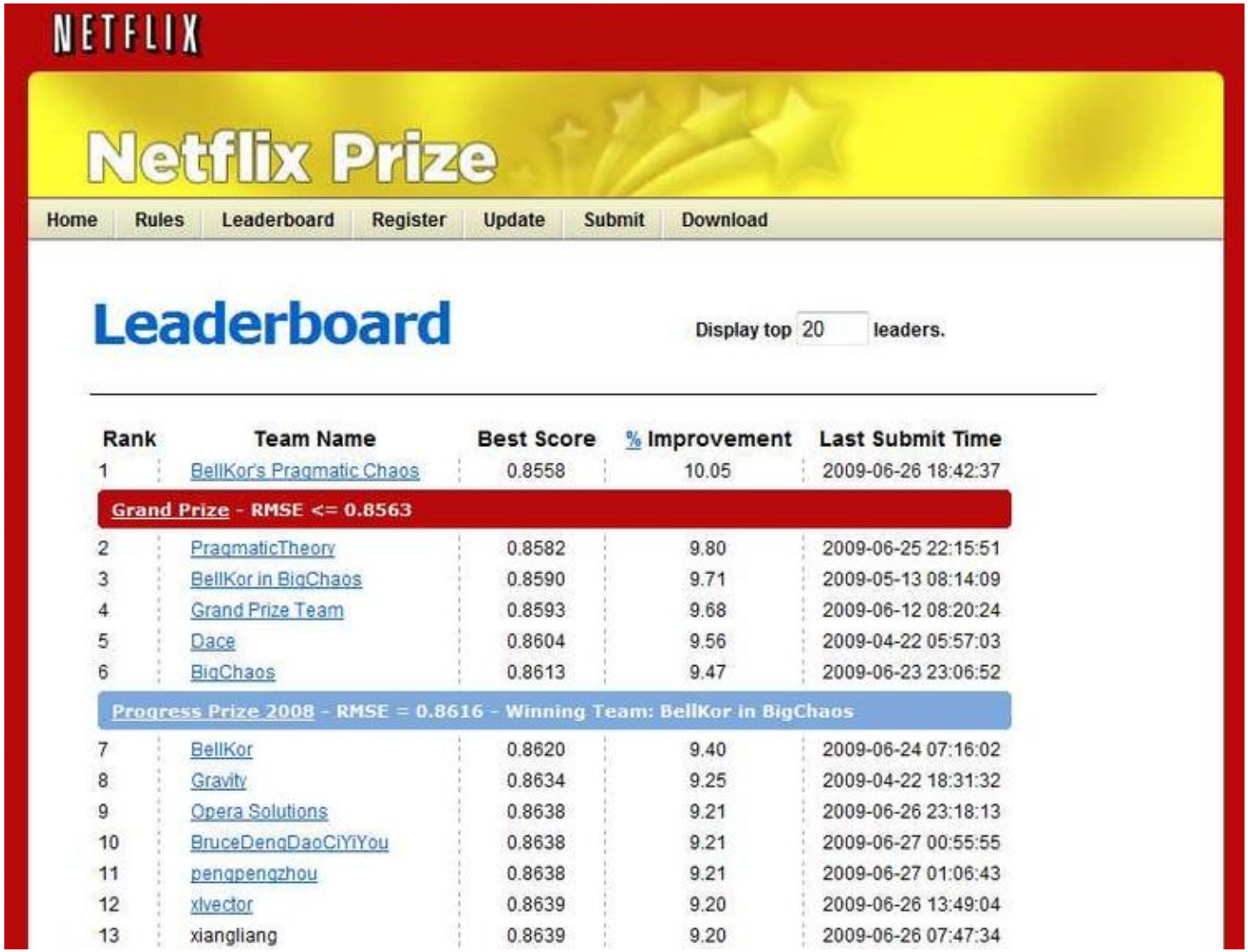
Still no prize! ☹
Getting desperate.
Try a "kitchen sink" approach!

The big picture

Solution of BellKor's Pragmatic Chaos



Standing on June 26th



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"

The Last 30 Days

- **Ensemble team formed**
  - Group of other teams on leaderboard forms a new team
  - Relies on combining their models
  - Quickly also get a qualifying score over 10%
- **BellKor**
  - Continue to get small improvements in their scores
  - Realize they are in direct competition with team **Ensemble**
- **Strategy**
  - Both teams carefully monitoring the leader board
  - Only sure way to check for improvement is to submit a set of predictions
    - This alerts the other team of your latest score

24 Hours from the Deadline

- **Submissions limited to 1 a day**
  - Only 1 final submission could be made in the last 24 hours
- **24 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	[illegible]			

Million $ Awarded Sept 21st



What's the moral of the story?

Submit early! ☺

Acknowledgments

- Some slides and plots borrowed from Yehuda Koren, Robert Bell and Padhraic Smyth
- **Further reading:**
  - Y. Koren, Collaborative filtering with temporal dynamics, KDD ’09
- https://web.archive.org/web/20141130213501/http://www2.research.att.com/~volinsky/netflix/bpc.html
- https://web.archive.org/web/20141227110702/http://www.the-ensemble.com/