CS 5683: Algorithms & Methods for Big Data Analytics

Recommender Systems: Latent Factor Models

Arunkumar Bagavathi

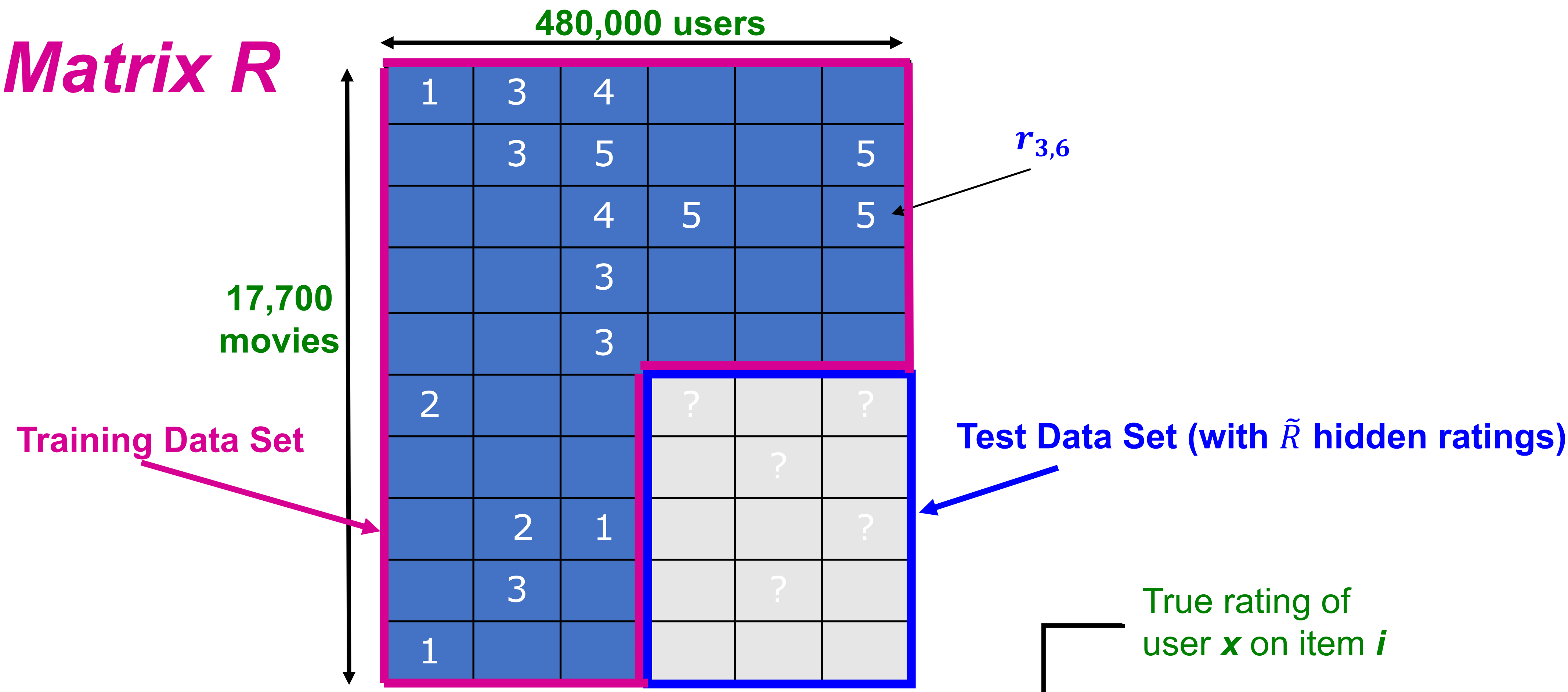
Department of Computer Science

Oklahoma State university

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) $= \frac{1}{|R|}\sqrt{\sum_{(i,x)\in R}(\hat{r}_{xi} - r_{xi})^2}$
  - **Netflix's system RMSE: 0.9514**
- **Competition**
  - 2,700+ teams
  - **$1 million** prize for 10% improvement on Netflix

Netflix Utility Matrix & Evaluation



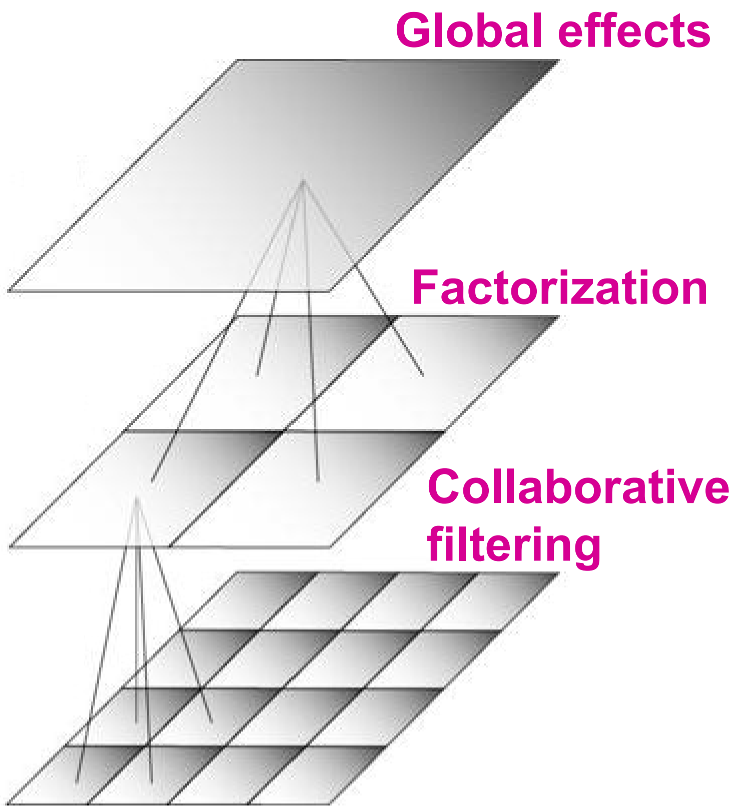
$$\textbf{RMSE} = \sqrt{\frac{1}{\tilde{R}} \sum_{(i,x)\in\tilde{R}} (\hat{r}_{xi} - r_{xi})^2}$$

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

Predicted rating

BellKor Recommender System

- **The winner of the Netflix Challenge!**
- **Multi-scale modeling of the data:**
  Combine a refined, top level, “regional” modeling data, with local view:
  - **Global:**
    - Overall deviations of users/movies
  - **Factorization:**
    - Addressing “regional” effects
  - **Collaborative filtering:**
    - Extract local patterns



Modeling Local & Global Effects

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

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

baseline estimate for r_{xi}

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

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:

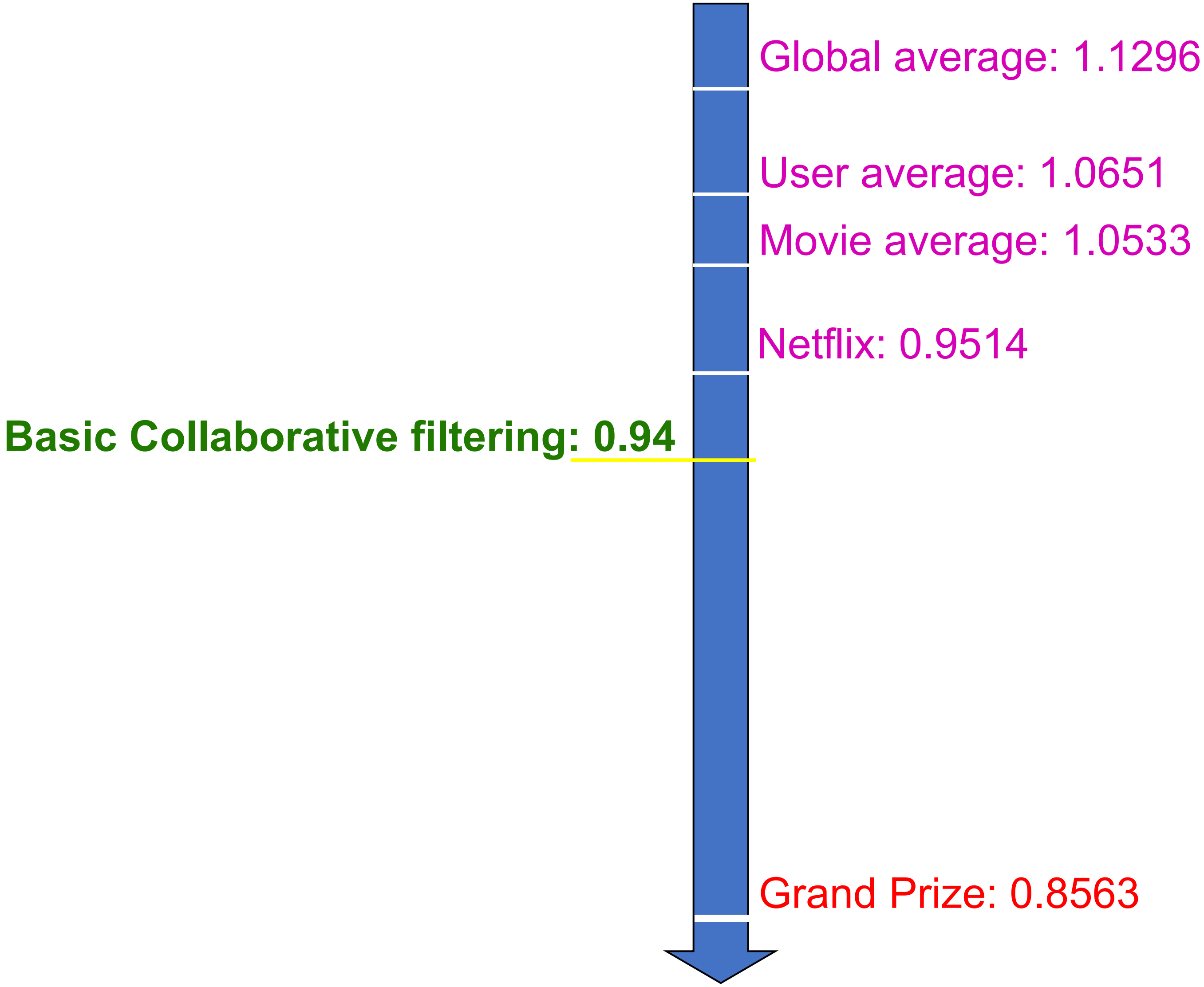
1) Similarity measures are "arbitrary"

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

Performance of Various Methods



Idea: Interpolation Weights w_{ij}

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

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

- **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: $\sum_{j \in N(i,x)} \boldsymbol{w_{ij}} \neq \mathbf{1}$
  - $\boldsymbol{w_{ij}}$ 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: $\sqrt{\frac{1}{\tilde{R}} \sum_{(i,x) \in \tilde{R}} (\hat{r}_{xi} - r_{xi})^2}$ or equivalently **SSE:** $\boldsymbol{\sum_{(i,x) \in R} (\hat{r}_{xi} - r_{xi})^2}$
  - 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 $\Rightarrow$ 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**

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

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

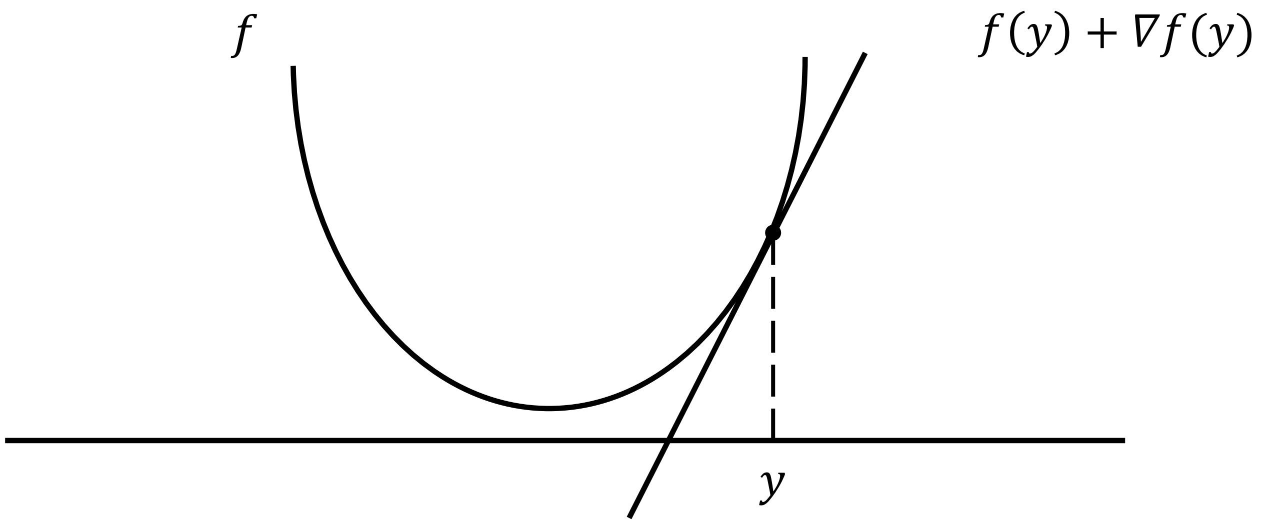
- Find $\boldsymbol{w_{ij}}$ that minimize **SSE** on **training data**!

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

- Think of ***w*** as a vector of numbers

Detour: Minimizing a Function

- **A simple way to minimize a function $f(x)$:**
  - Compute a 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

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

- **We have the optimization problem, now what?**
- **Gradient decent:**
  - **Iterate until convergence:** $\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \boldsymbol{\nabla_w J}$

η … learning rate

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

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

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

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

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

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

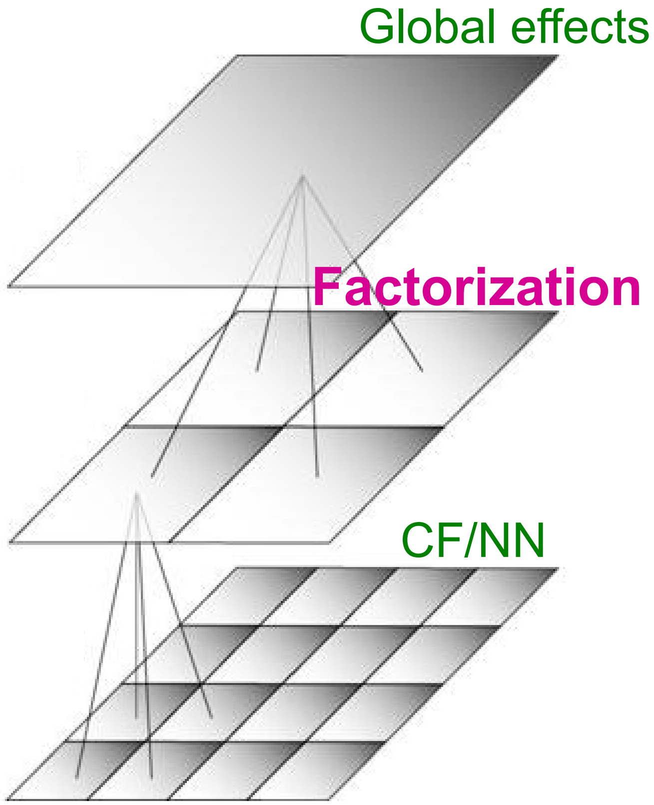
$w_{old} = w_{new}$

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

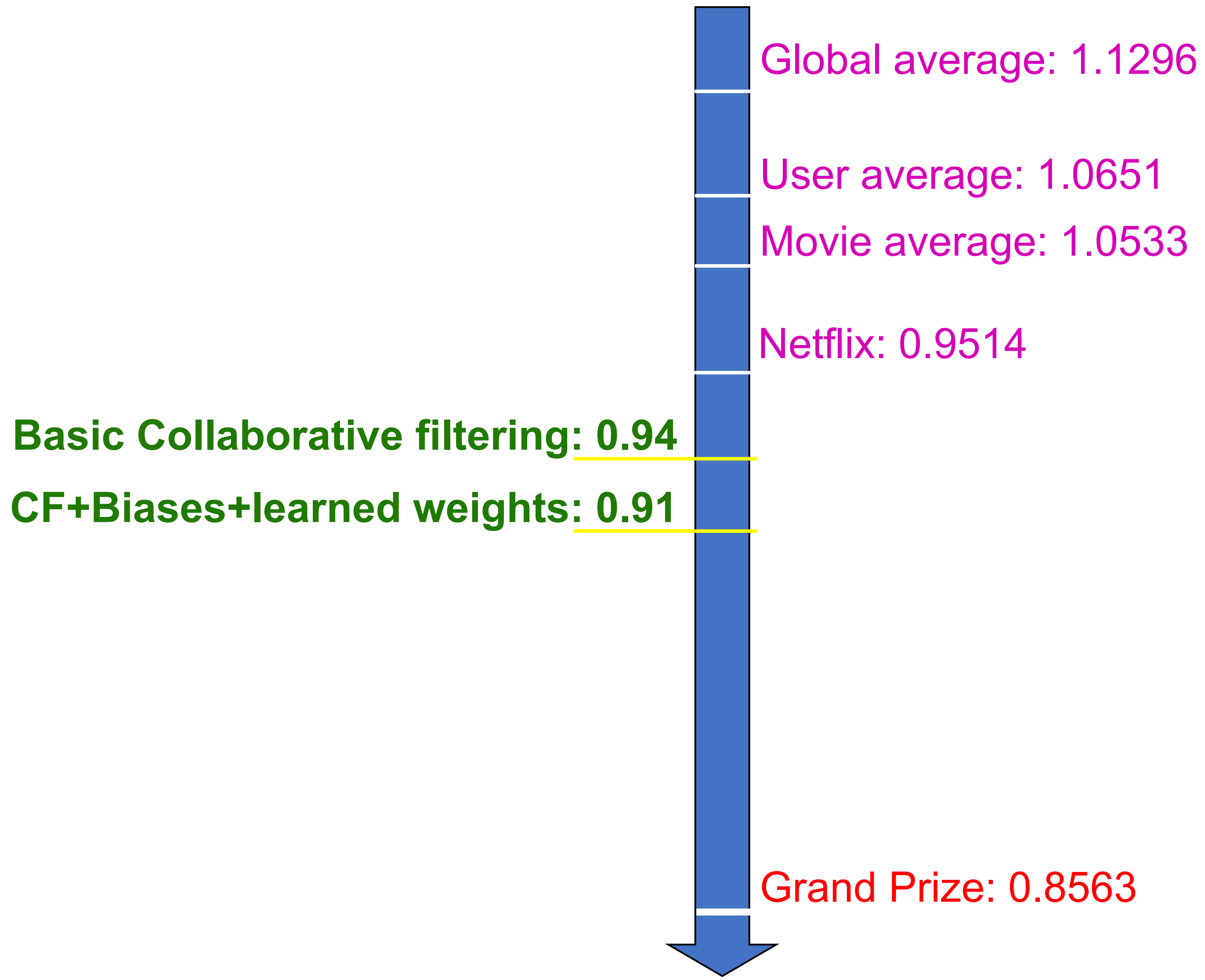
Interpolation Weights

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

- **Next: Latent factor model**
  - Extract "regional" correlations



Performance of Various Methods



Latent Factor Models

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

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

R (rows: items, columns: users)

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

Q (rows: items, columns: factors)

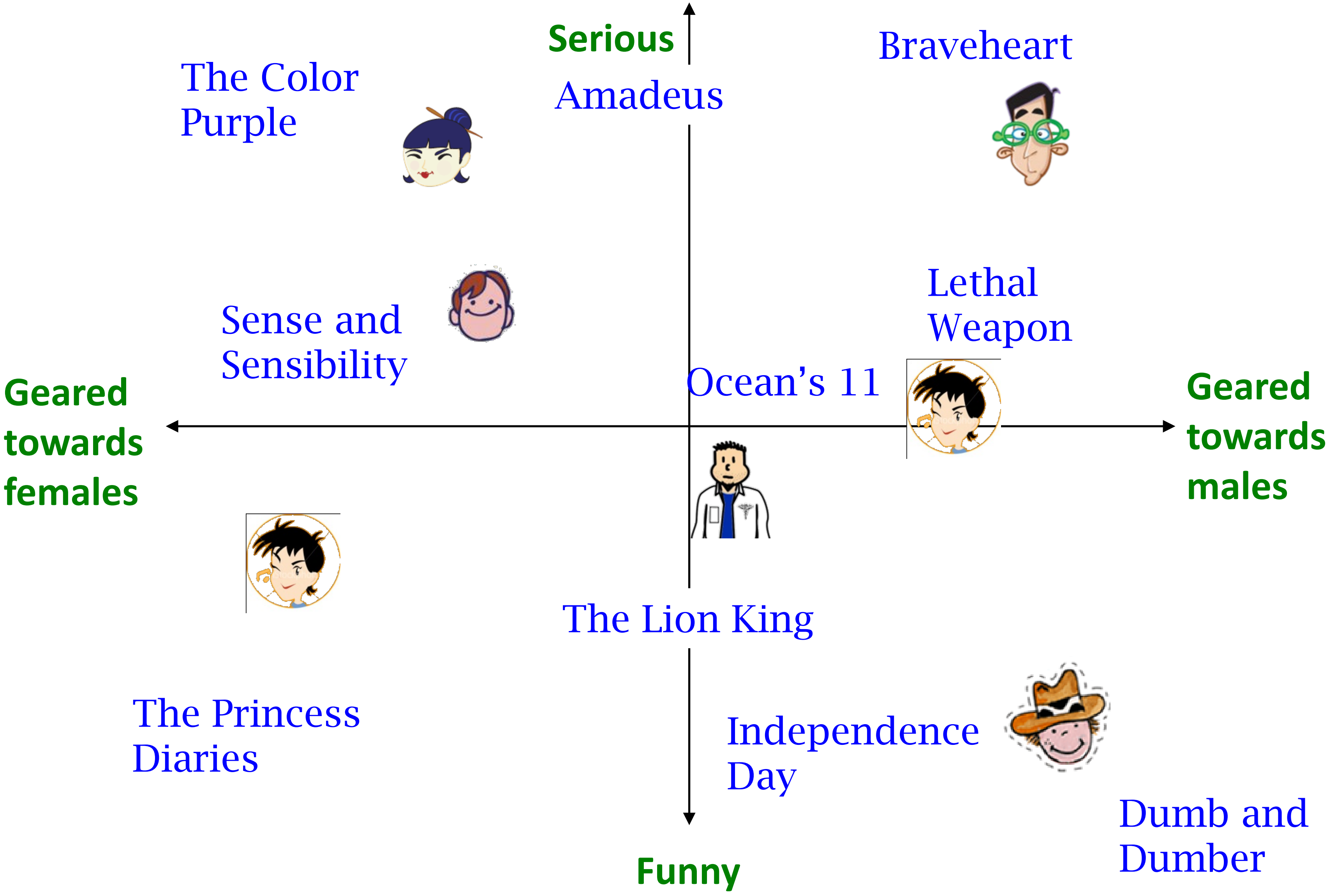
.1	-.4	.2
-.5	.6	.5
-.2	.3	.5
1.1	2.1	.3
-.7	2.1	-2
-1	.7	.3

$\boldsymbol{P}^T$ (rows: factors, columns: users)

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

Latent Factor Models (e.g., SVD)



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*?**

users (columns) × items (rows):

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

≈

Q (items × factors):

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

P^T (factors × users):

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

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

$$= \sum_f q_{if} \cdot p_{xf}$$

$\boldsymbol{q_i}$ = row $\boldsymbol{i}$ of $\boldsymbol{Q}$
$\boldsymbol{p_x}$ = column $\boldsymbol{x}$ of $\boldsymbol{P}^T$

Ratings as Products of Factors

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

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	

$\approx$

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

$$= \sum_f q_{if} \cdot p_{xf}$$

q_i = row i of Q

p_x = column x of P^T

items

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

factors

Q

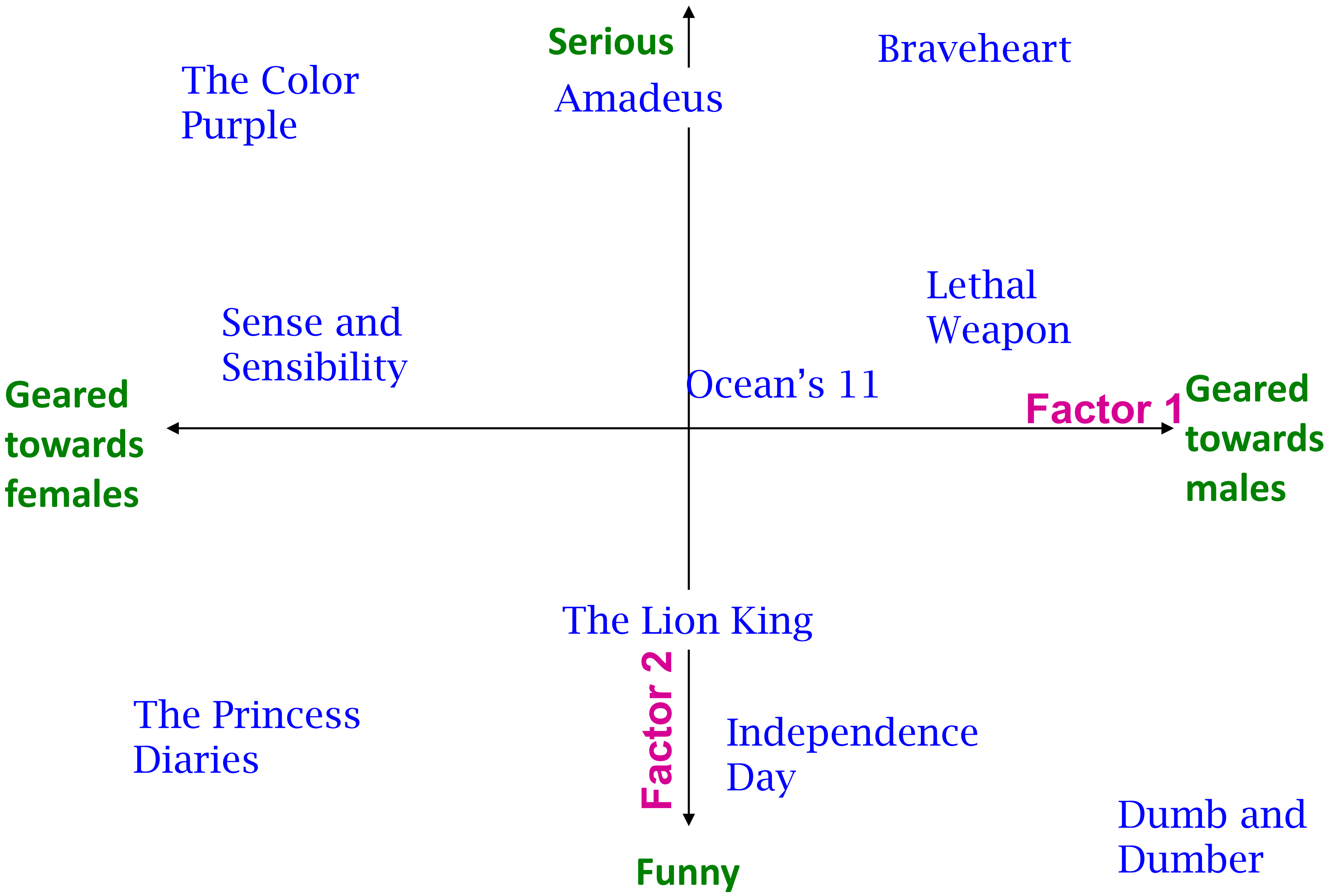
users

factors

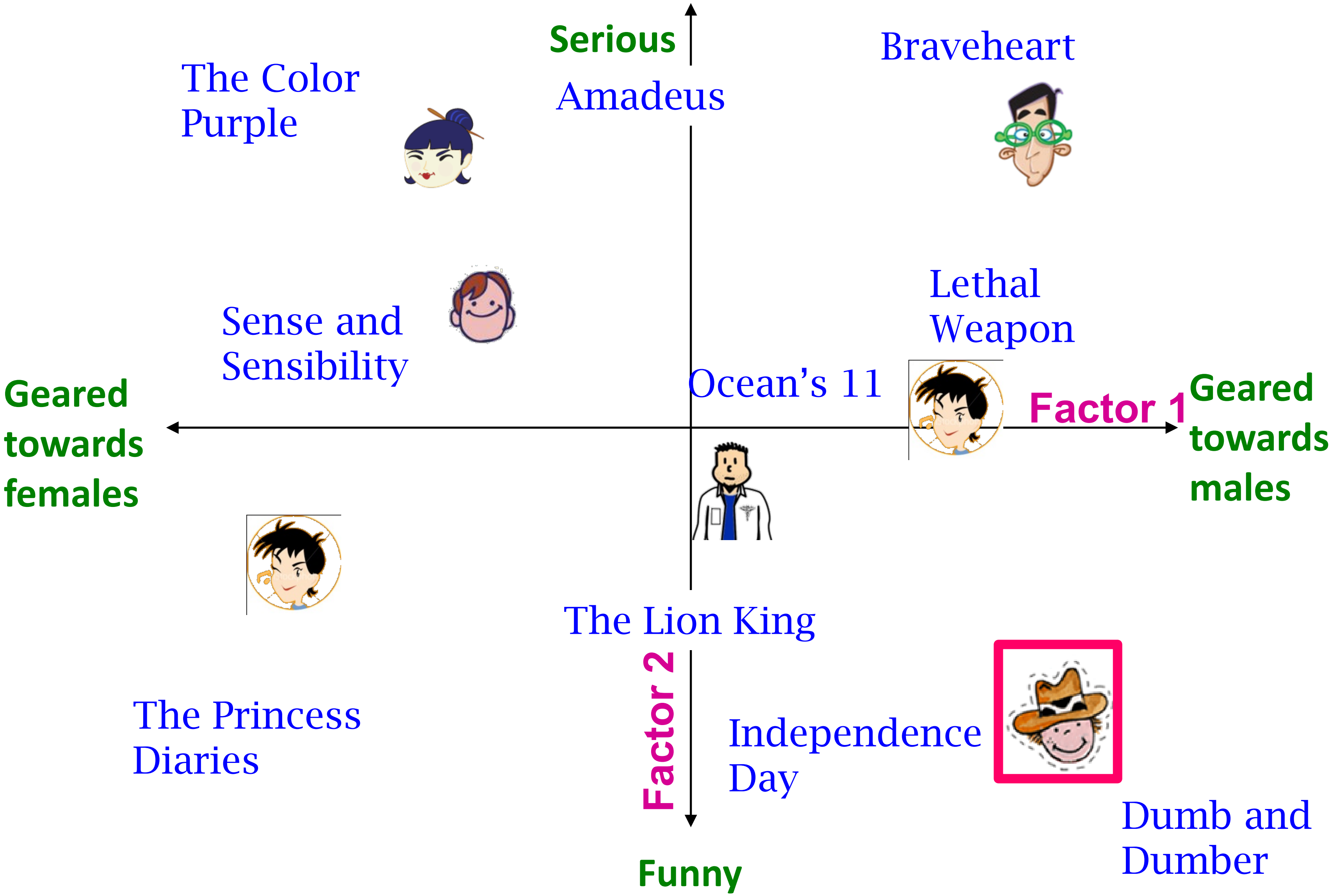
1.1	-.2	.3	.5	**-2**	-.5	.8	-.4	.3	1.4	2.4	-.9
-.8	.7	.5	1.4	**.3**	-1	1.4	2.9	-.7	1.2	-.1	1.3
2.1	-.4	.6	1.7	**2.4**	.9	-.3	.4	.8	.7	-.6	.1

P^T

Latent Factor Models

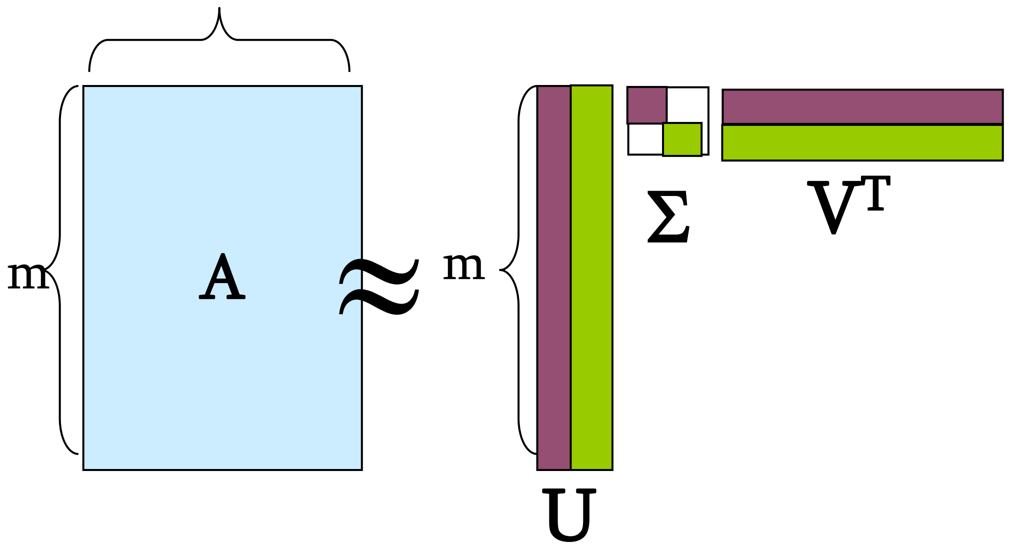


Latent Factor Models



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: $R \approx Q \cdot P^T$
  $\boldsymbol{A = R,\ \ Q = U,\ \ P^T = \Sigma\, V^T}$

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

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

SVD: Good Stuff

users (columns) × items (rows):

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

≈

Q (items × factors):

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

P^T (factors × users):

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

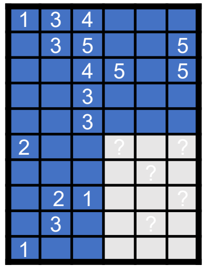
- **SVD isn't defined when entries are missing!**
- **Use specialized methods to find *P, Q***

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

- **Note:**
  - We don't require cols of *P, Q* to be orthogonal/unit length
  - *P, Q* map users/movies to a latent space
  - The most popular model among Netflix contestants

Back to Our Problem

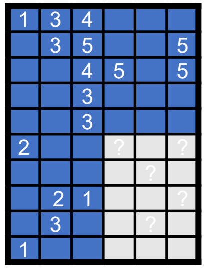
- **Want to minimize SSE for unseen test data**
- **Idea: Minimize SSE on training data**
  - Want large ***k*** (# of factors) to capture all the signals
  - But, **SSE** on test data begins to rise for ***k*** > 2



- 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

Dealing with Missing Entries

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

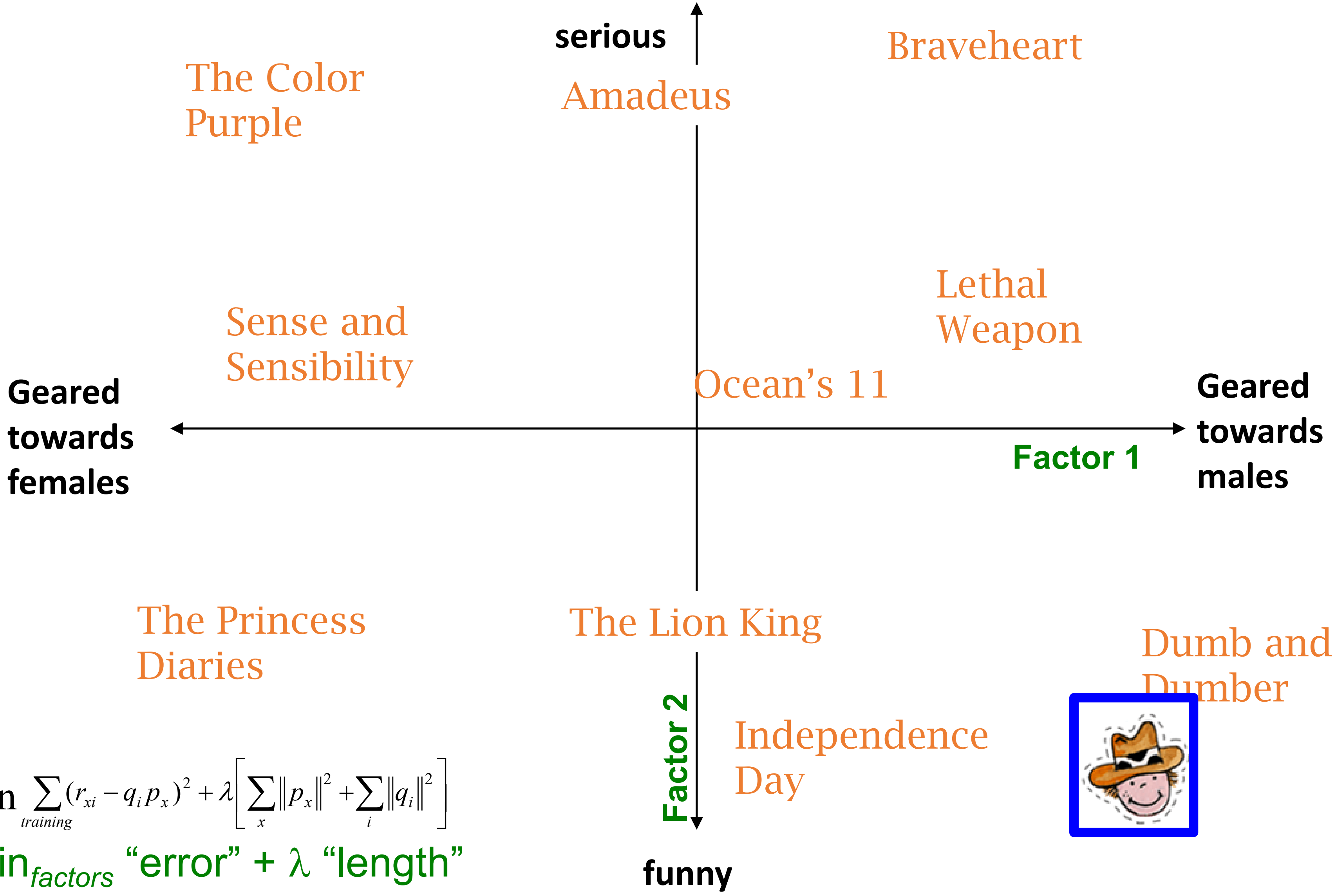


$$\min_{P,Q} \underbrace{\sum_{training} (r_{xi} - q_i p_x)^2}_{\text{“error”}} + \left[\underbrace{\lambda_1 \sum_x \|p_x\|^2 + \lambda_2 \sum_i \|q_i\|^2}_{\text{“length”}} \right]$$

λ_1, λ_2 … user set regularization parameters

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

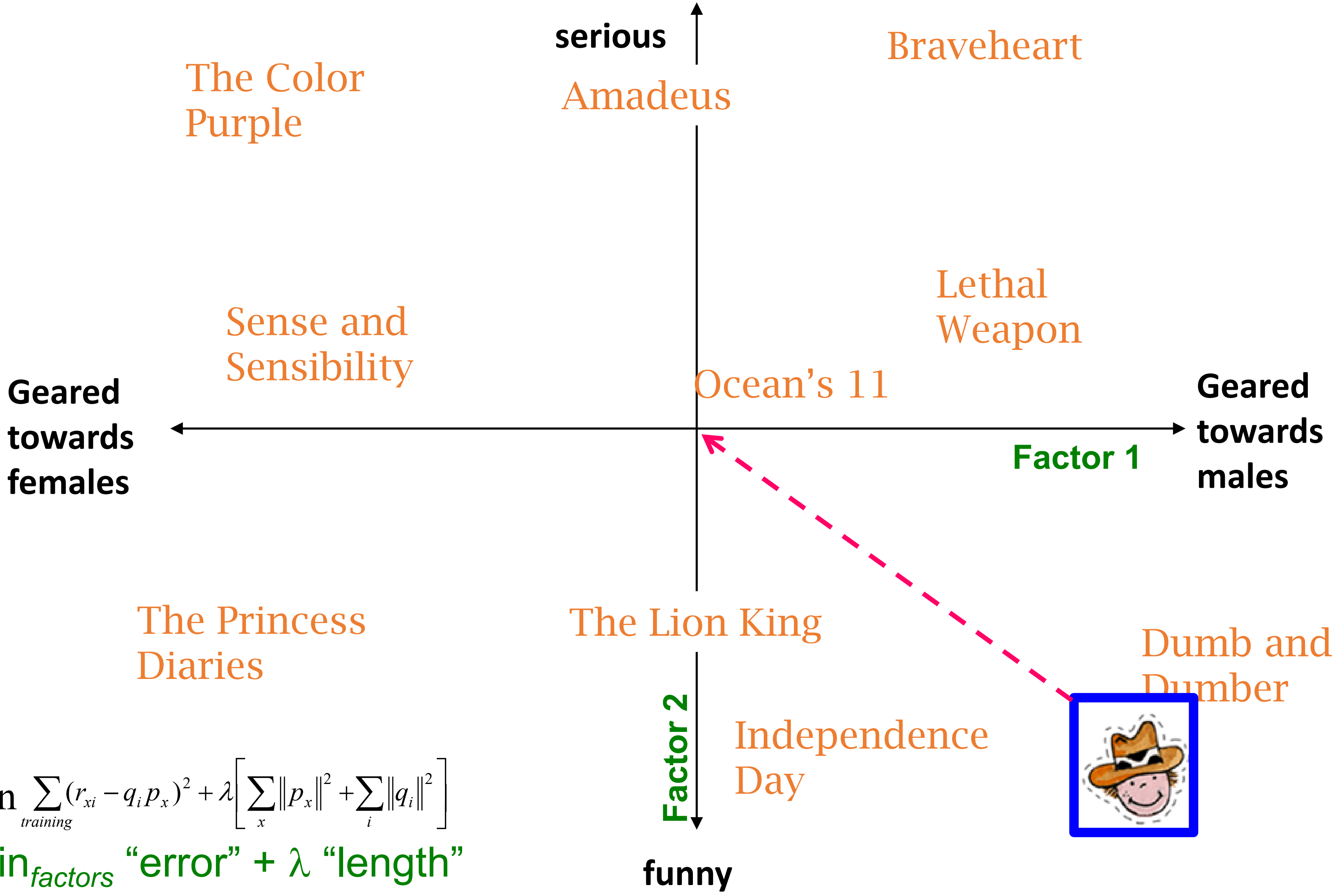
Effect of Regularization



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

$\min_{factors}$ "error" + λ "length"

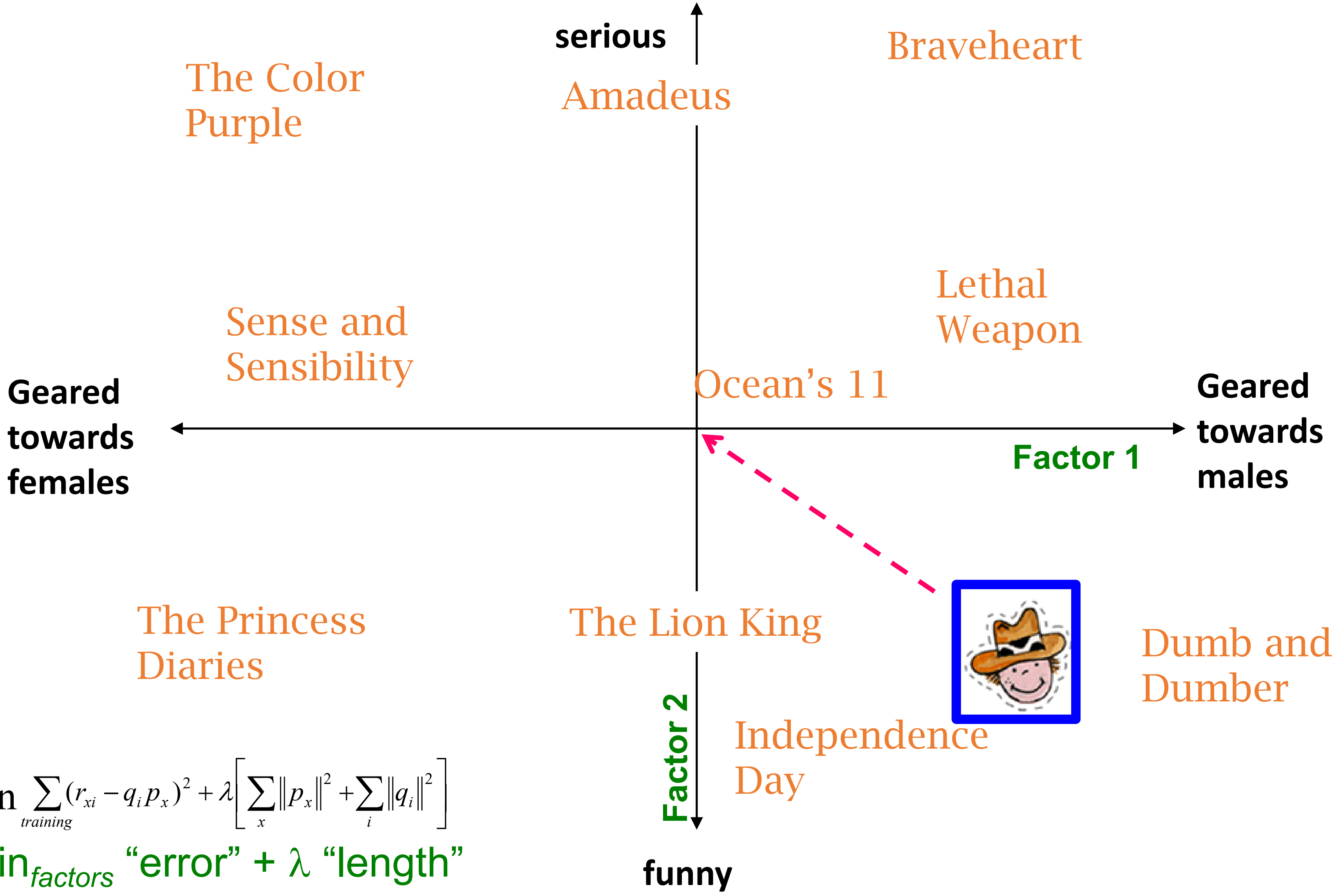
Effect of Regularization



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

$\min_{factors}$ “error” + λ “length”

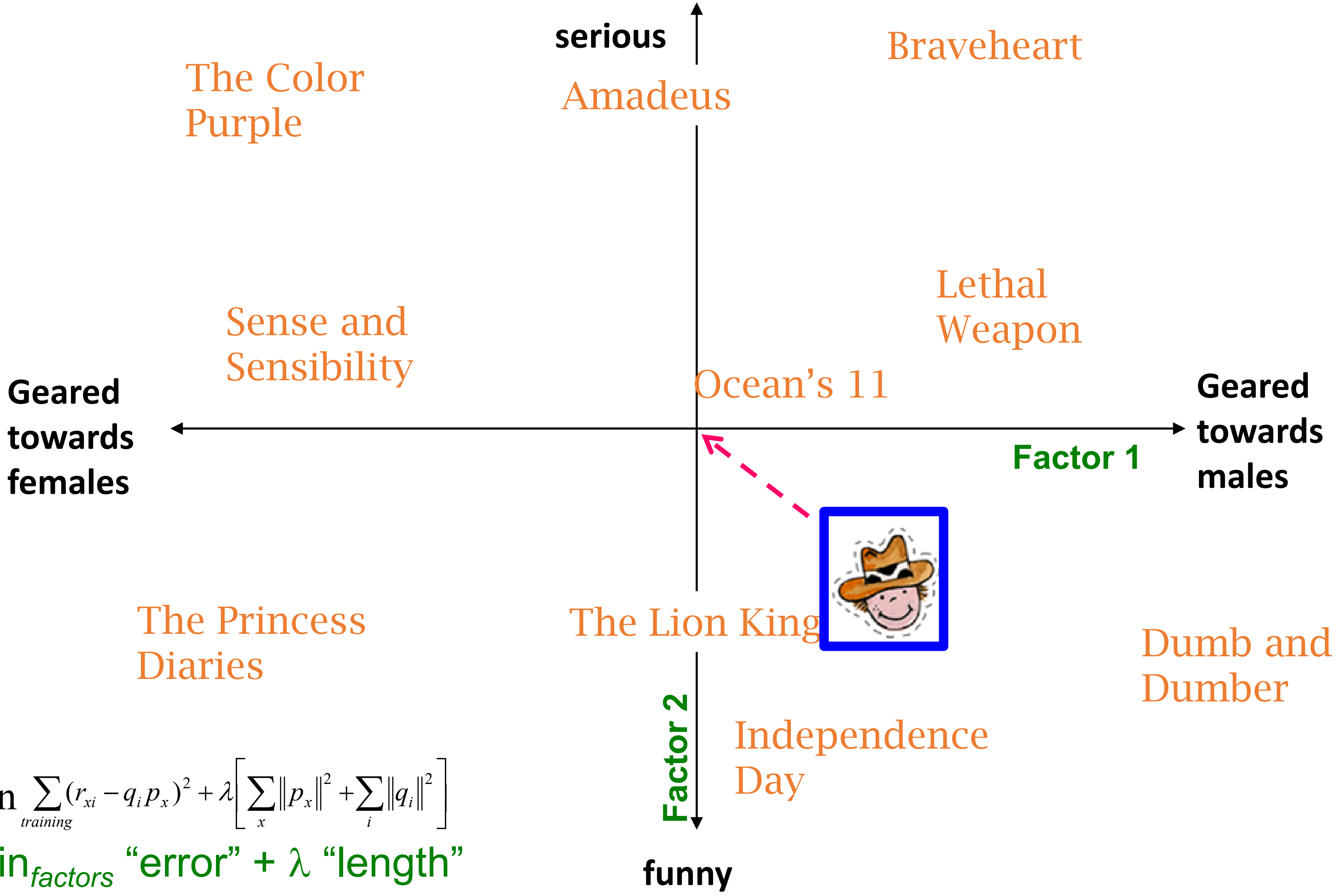
Effect of Regularization



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

$\min_{factors}$ "error" + λ "length"

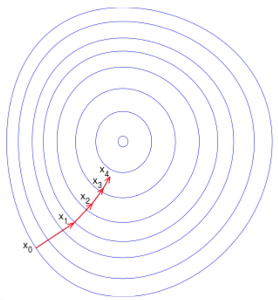
Effect of Regularization



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

$\min_{factors}$ "error" + λ "length"

Gradient Descent



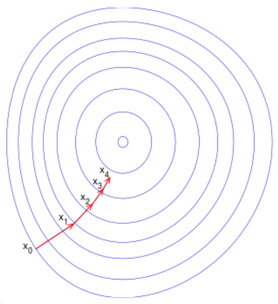
- **Want to find matrices *P* and *Q*:**

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

- **Gradient decent:**
  - Initialize ***P*** and ***Q*** (using SVD, pretend missing ratings are 0)
  - Do gradient descent:
    - $\boldsymbol{P} \leftarrow \boldsymbol{P} - \eta \cdot \nabla \mathbf{P}$
    - $\boldsymbol{Q} \leftarrow \boldsymbol{Q} - \eta \cdot \nabla \mathbf{Q}$
    - where $\nabla \boldsymbol{Q}$ 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 $\boldsymbol{q_{if}}$ is entry ***f*** of row $\boldsymbol{q_i}$ of matrix ***Q***
  - **Observation: Computing gradients is slow!**

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

Stochastic Gradient Descent



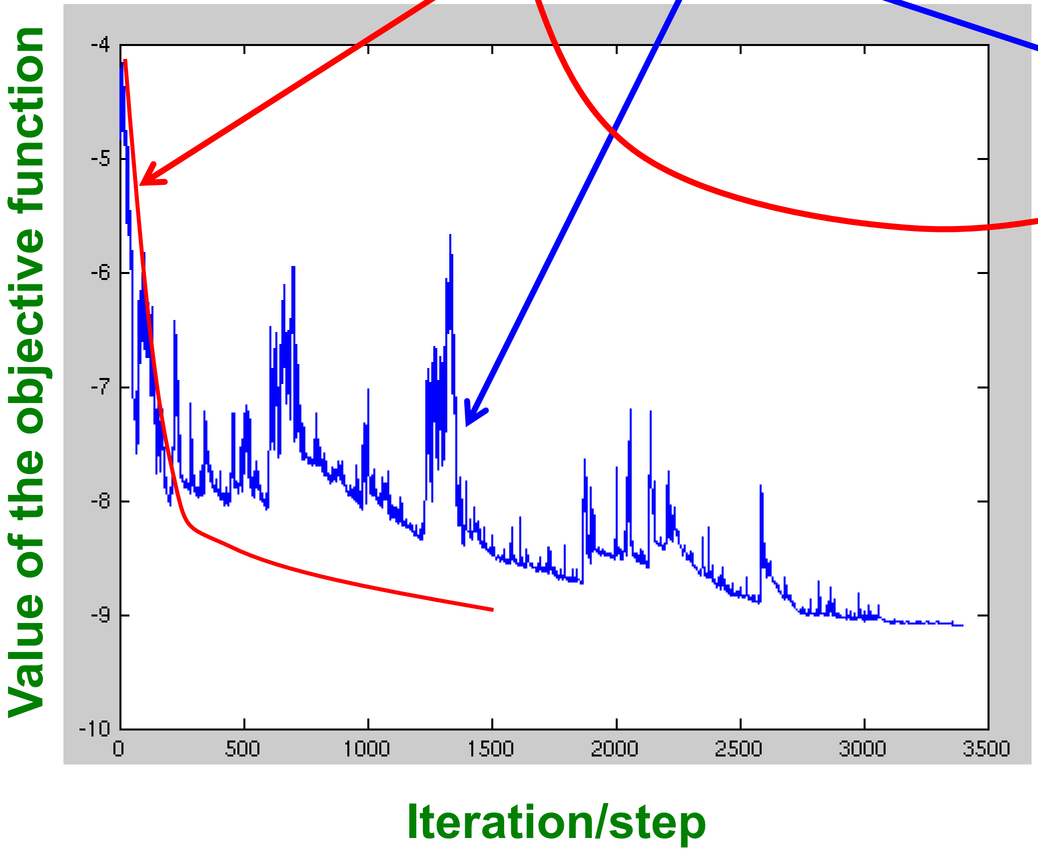
- **Gradient Descent (GD) vs. Stochastic GD**
  - **Observation:** $\nabla Q = [\nabla q_{if}]$ where

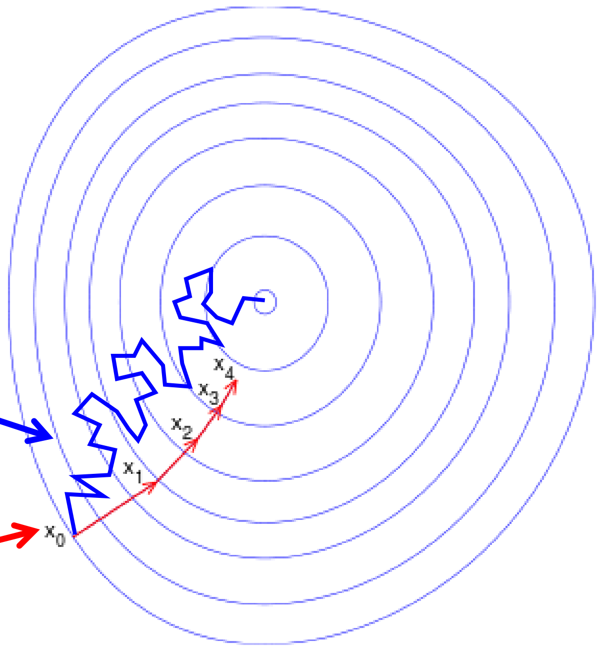
$$\nabla q_{if} = \sum_{x,i} -2(r_{xi} - q_{if}p_{xf})p_{xf} + 2\lambda q_{if} = \sum_{\boldsymbol{x},\boldsymbol{i}} \nabla \boldsymbol{Q}\,(\boldsymbol{r}_{\boldsymbol{xi}})$$

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

GD Vs. SGD

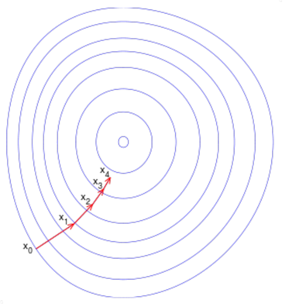
- **Convergence of GD vs. SGD**





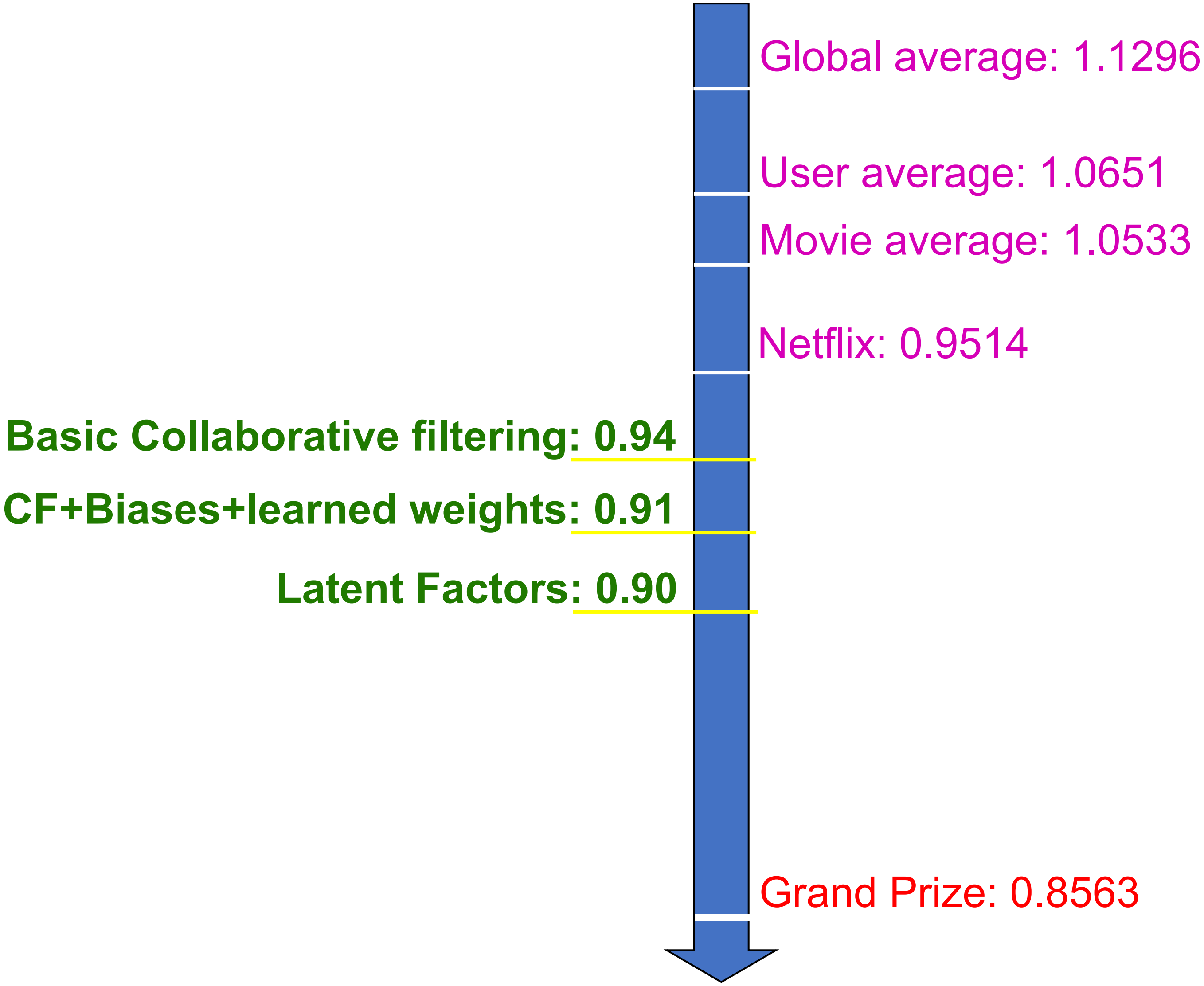
GD improves the value of the objective function at every step.
SGD improves the value but in a “noisy” way.
GD takes fewer steps to converge but each step takes much longer to compute.
In practice, **SGD** is much faster!

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"

Performance of Various Methods

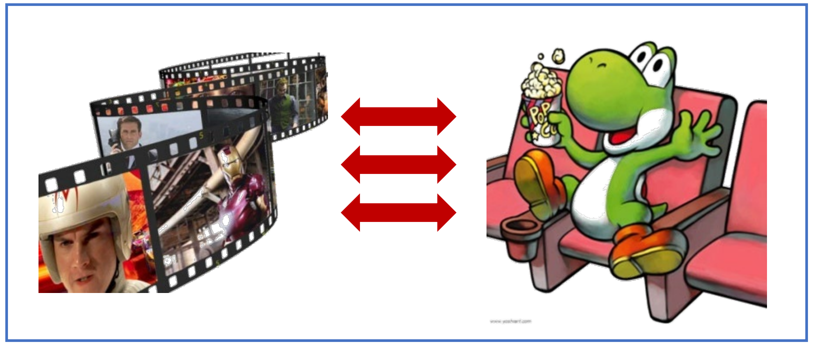


Modeling Biases and Interactions

user bias

movie bias

user-movie interaction



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

- $\boldsymbol{\mu}$ = overall mean rating
- $\boldsymbol{b_x}$ = bias of user $\boldsymbol{x}$
- $\boldsymbol{b_i}$ = bias of movie $\boldsymbol{i}$

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 All Together

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

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

- **Example:**
  - Mean rating: μ ***= 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: $b_i = +0.5$
  - Predicted rating for you on Star Wars:
    = 3.7 - 1 + 0.5 = 3.2

Fitting the New Model

- **Solve:**

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

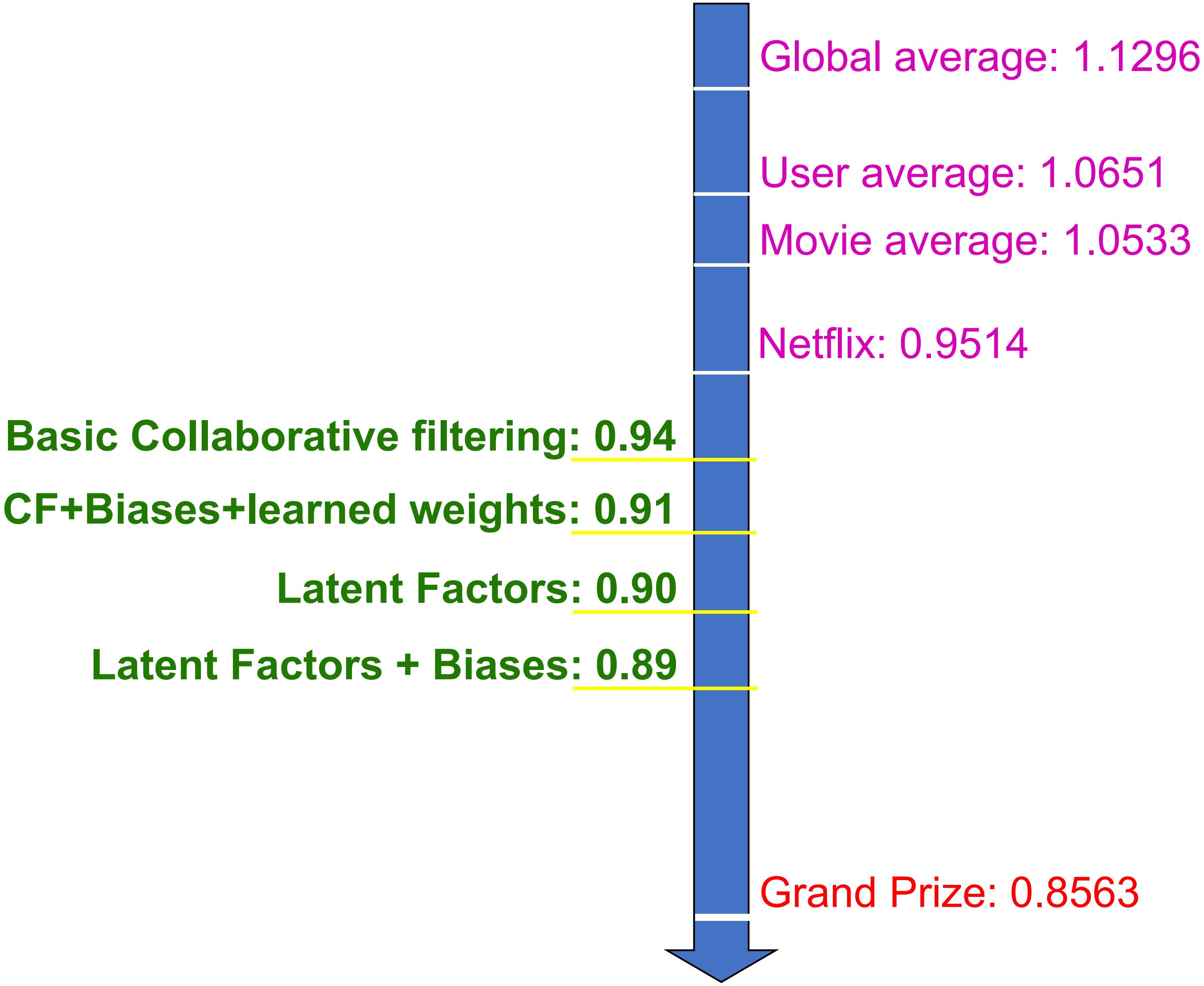
$$+ \left(\lambda_1 \sum_i \|q_i\|^2 + \lambda_2 \sum_x \|p_x\|^2 + \lambda_3 \sum_x \|b_x\|^2 + \lambda_4 \sum_i \|b_i\|^2 \right)$$

regularization

λ is selected via grid-search on a validation set

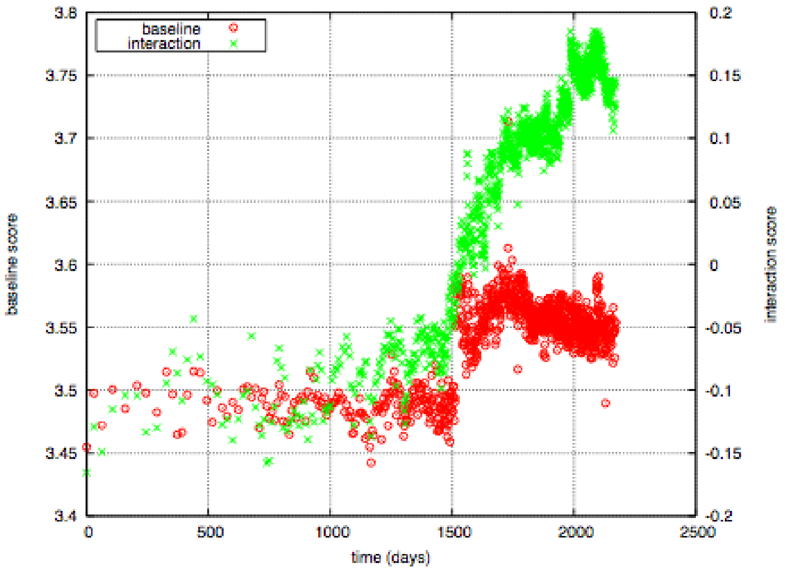
- **Stochastic gradient decent to find parameters**
  - **Note:** Both biases b_x, b_i as well as interactions q_i, p_x are treated as parameters (we estimate them)

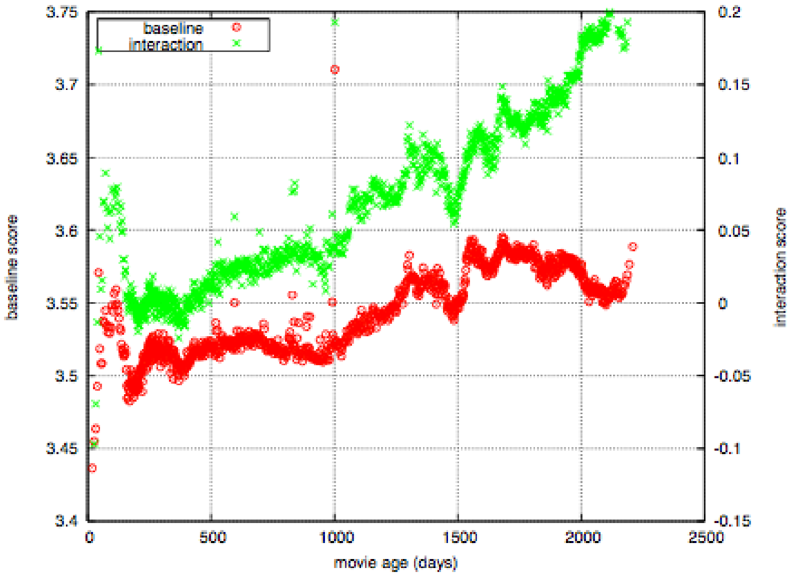
Performance of Various Methods



Temporal Biases of Users

- **Sudden rise in the average movie rating**
  - 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

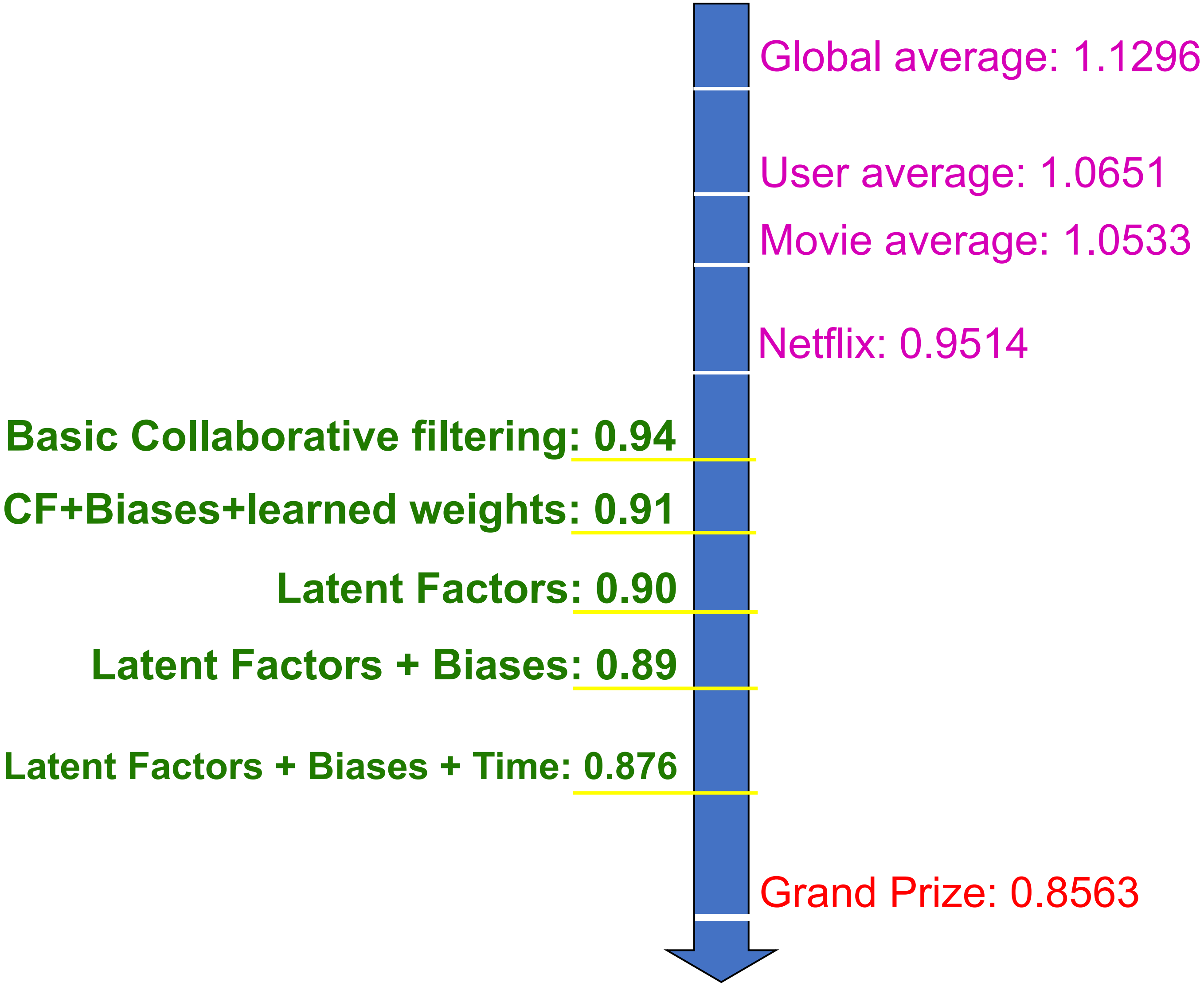
Temporal Biases & Factors

- **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

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

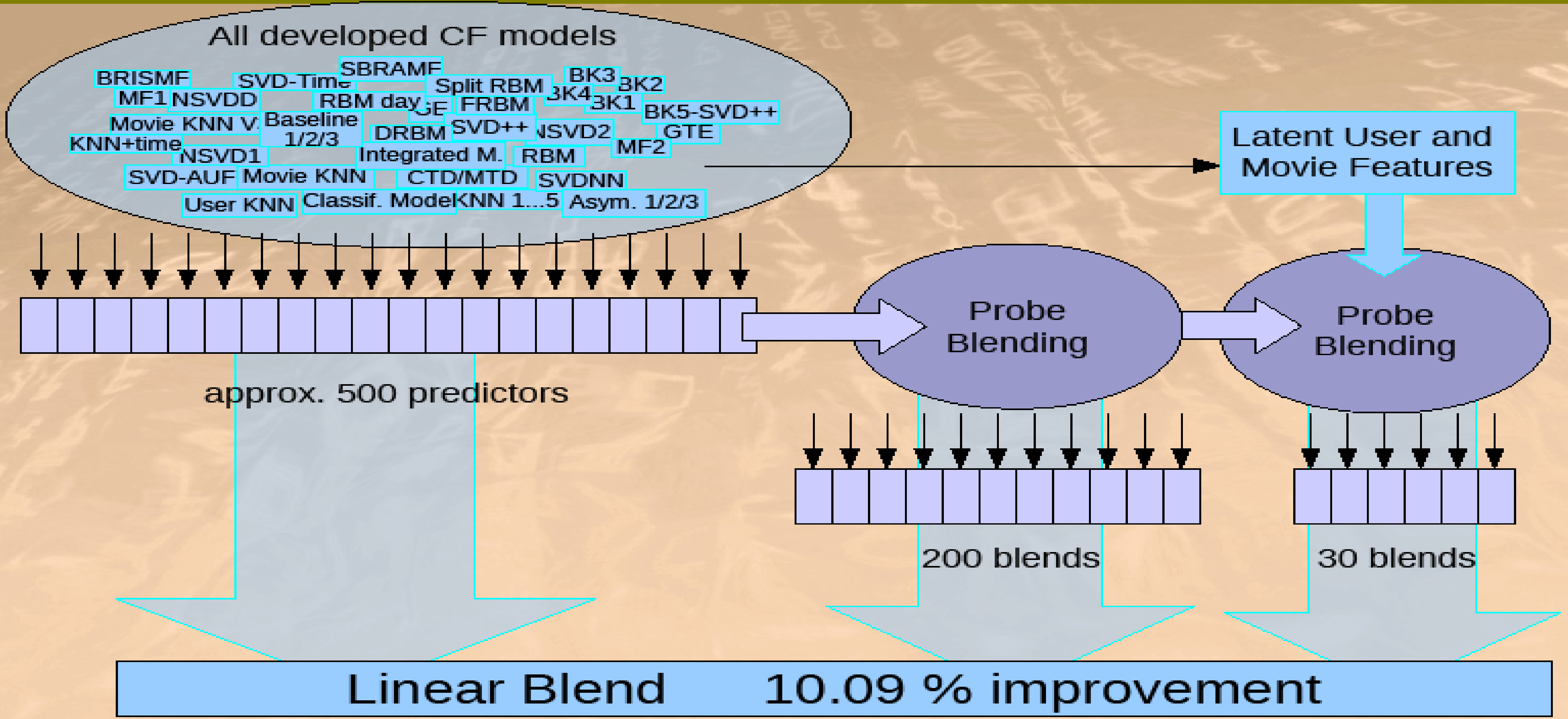
- **Add temporal dependence to factors**
  - $p_x(t)$… user preference vector on day t

Performance of Various Methods



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

Solution of BellKor's Pragmatic Chaos



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 that they are in direct competition with Ensemble

- **Strategy**
  - Both teams carefully monitoring the leaderboard
  - 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 24h
- **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	0.8582	[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				

Questions???

Acknowledgements

Most of this lecture slides are obtained from the Mining Massive Datasets course: http://www.mmds.org/

Further reading:

- Y. Koren, Collaborative filtering with temporal dynamics, KDD ’09
- http://www2.research.att.com/~volinsky/netflix/bpc.html
- http://www.the-ensemble.com/