Mining Massive Datasets

Lecture 6

Artur Andrzejak
http://pvs.ifi.uni-heidelberg.de

RUPRECHT-KARLS-
UNIVERSITÄT
HEIDELBERG

Note on Slides

A substantial part of these slides come (either verbatim or in a modified form) from the book
Mining of Massive Datasets
by Jure Leskovec, Anand Rajaraman, Jeff Ullman
(Stanford University).
For more information, see the website accompanying the book: http://www.mmds.org.

Recommender Systems

High dim. data	Graph data	Infinite data	Machine learning	Apps
Locality sensitive hashing	PageRank, SimRank	Filtering data streams	SVM	Recommender systems
Clustering	Community Detection	Web advertising	Decision Trees	Association Rules
Dimensionality reduction	Spam Detection	Queries on streams	Perceptron, kNN	Duplicate document detection

Programming in Spark & MapReduce

The Netflix Prize: Introduction

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

The Netflix Utility Matrix *R*

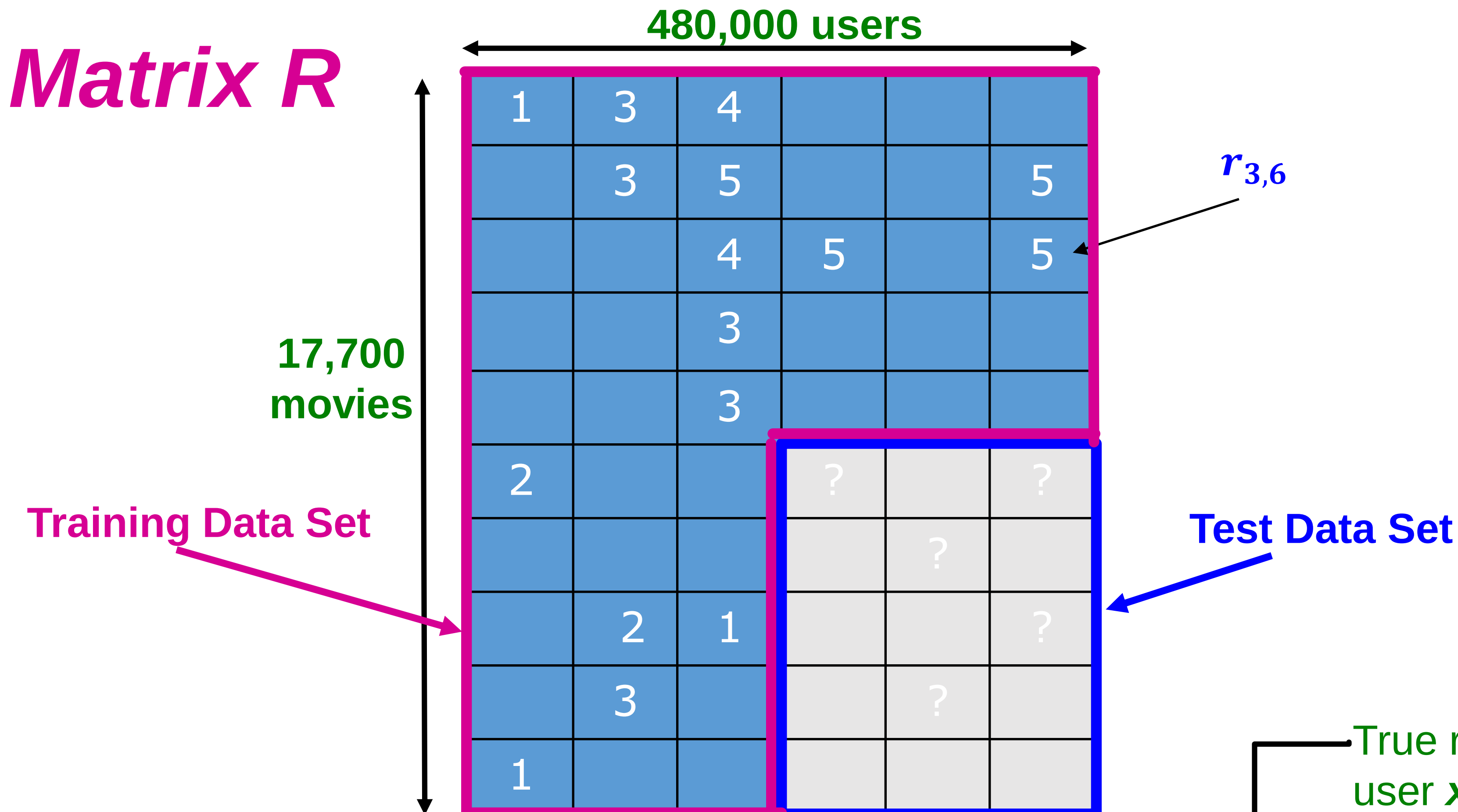
Matrix R

480,000 users

17,700 movies

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

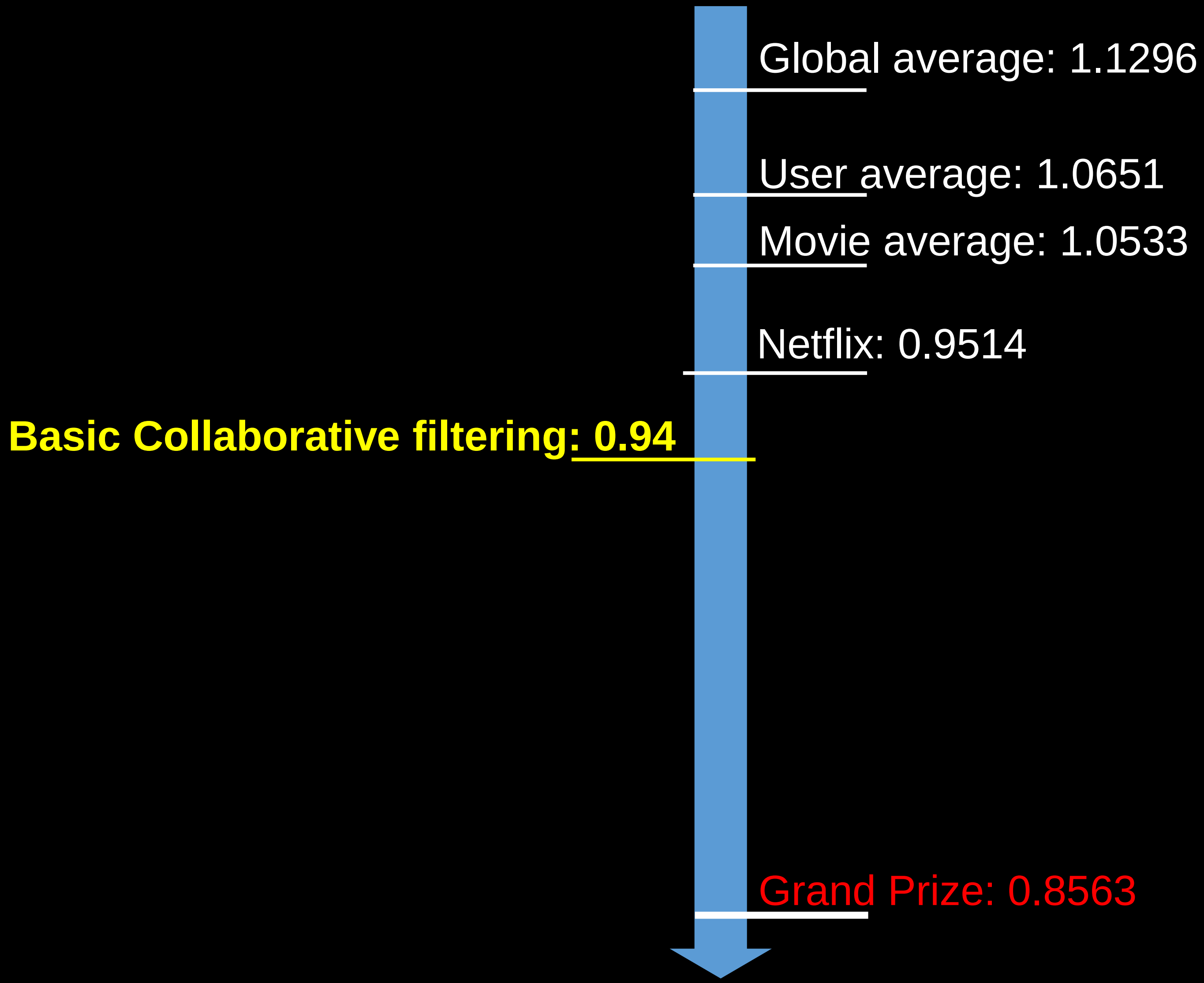
Utility Matrix R: Evaluation



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

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

Predicted rating


Global average: 1.1296
User average: 1.0651
Movie average: 1.0533
Netflix: 0.9514
Basic Collaborative filtering: 0.94
Grand Prize: 0.8563

Contrasting Recommendation Methods

Recall: Utility Matrix

Goal: ■ - estimate rating of movie **1** by user **5**

users

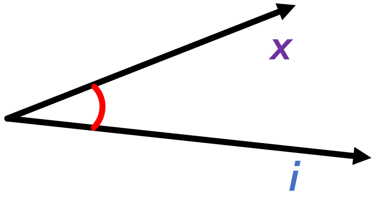
movies	1	2	3	4	**5**	6	7	8	9	10	11	12
1	1		3		?	5			5		4	
2			5	4			4			2	1	3
3	2	4		1	2		3		4	3	5	
4		2	4		5			4			2	
5			4	3	4	2					2	5
6	1		3		3			2			4	

Content-Based Recommendations

- We construct for each item a vector $\boldsymbol{i}$ ("**item profile**") and for each user a vector $\boldsymbol{x}$ ("**user profile**")
  - Item profile $\boldsymbol{i}$: k "natural" attributes of an item
  - User vector $\boldsymbol{x}$: a combination of item profiles for <u>similar</u> items rated by this user (also a k-vector)

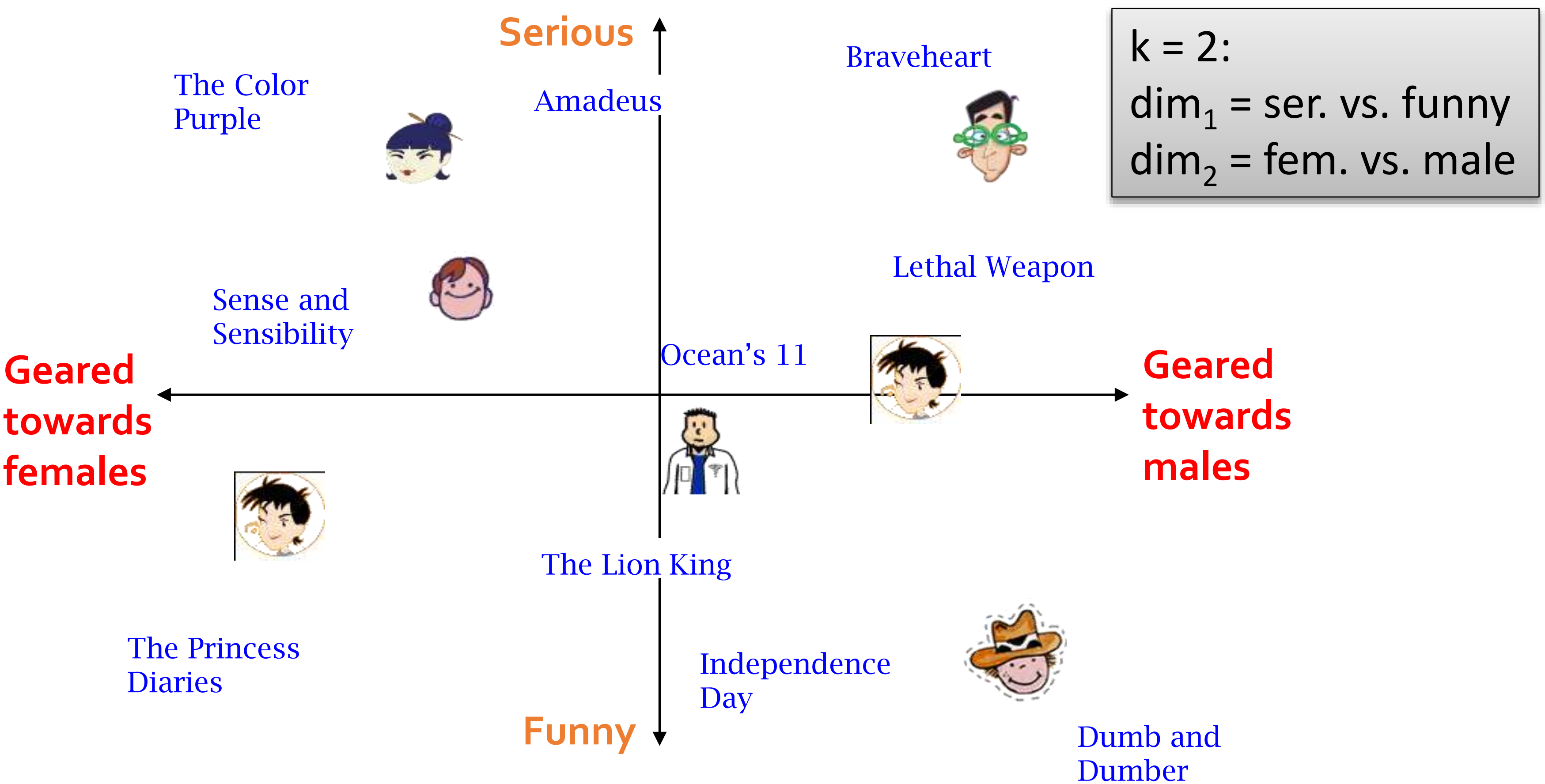
- Prediction heuristic:
  - A content-based prediction $\mathbf{r}_{\mathbf{x,i}}$ is approximated as similarity of these two k-vectors: $\mathbf{r}_{\mathbf{x,i}} = \boldsymbol{u}(\boldsymbol{x}, \boldsymbol{i})$
  - I.e., given a user profile $\boldsymbol{x}$ and item profile $\boldsymbol{i}$, estimate their similarity as:

$$\boldsymbol{u}(\boldsymbol{x}, \boldsymbol{i}) = \cos(\boldsymbol{x}, \boldsymbol{i}) = \frac{\boldsymbol{x} \cdot \boldsymbol{i}}{||\boldsymbol{x}|| \cdot ||\boldsymbol{i}||}$$



Content-Based R.: Interpretation

- In content-based recommendation, we represented each item and each user as a vector in a k-dimensional space
- => Item **i** close to a user **x** gets a high recommendation rating



Item-Item Collaborative Filtering

users

items

	1	2	3	4	**5**	6	7	8	9	10	11	12	sim(1,m)
1	1		3		?	5			5		4		1.00
2			5	4			4			2	1	3	-0.18
3	2	4		1	2		3		4	3	5		**0.41**
4		2	4		5			4			2		-0.10
5			4	3	4	2					2	5	-0.31
6	1		3		3			2			4		**0.59**

S={3,6}

- Find set **S** of items similar to item i=1 rated by target user x=5, and predict $r_{5,1}$ as a weighted sum of ratings (by user x) over all items in **S**
- => Rating $r_{5,1}$ is predicted as a weighted sum of other rows (= item ratings)

User-User CF Collaborative Filtering

users

Q={3,11}

items

	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	

- Find set **Q** of users similar to target user x=5 <u>who have rated item i=1</u>, and predict $\mathbf{r_{5,1}}$ as a weighted sum of ratings for item 1 by all users in **Q**
- => Rating $\mathbf{r_{5,1}}$ is predicted as a weighted sum of other columns (~ users)

Merging Content-Based and CF Methods

users

items	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	

$$? = \frac{p_x \cdot q_i}{\| p_x \| \cdot \| q_i \|} = C_{x,i} \cdot \begin{bmatrix} p_{x,1} \\ \vdots \\ p_{x,k} \end{bmatrix} \cdot \begin{bmatrix} q_{i,1} \\ \vdots \\ q_{i,k} \end{bmatrix}$$

Content-based: each rating is a product of two k-vectors "outside" the utility matrix

CF: each rating is a linear combination of other ratings from the utility matrix

Can we combine both worlds:
=> product of vectors from UM?

Latent Factor Models

Improving Content-Based Approach

users

items

	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	

$$? = \frac{p_x \cdot q_i}{\| p_x \| \cdot \| q_i \|} = C_{x,i} \cdot \begin{bmatrix} p_{x,1} \\ \vdots \\ p_{x,k} \end{bmatrix} \cdot \begin{bmatrix} q_{i,1} \\ \vdots \\ q_{i,k} \end{bmatrix}$$

- Representing each item by a k-vector $\mathbf{q_i}$ and each user by k-vector $\mathbf{p_x}$ is a very good idea
- But <u>can we replace "hand-crafted" item-profiles by synthetic profiles</u> derived from the utility matrix?
  - Similarly, for user profiles?

Latent Factor Models

Other view: **factorize** the utility matrix R

- = represent as product of two "thin" matrices)

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	

items

R

≈

factors (length k)

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

items

Q

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

factors (length k)

$\boldsymbol{P^T}$

- Let's assume we can approximate the utility matrix ***R*** as a product of "thin" $Q \cdot P^T$
- ***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

- Prediction: estimating the missing rating of user ***x*** for item ***i***

Similar to $\cos(q_i, p_x)$ but no scalar factor

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

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

q_i = row i of Q

p_x = column x of P^T

users (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 (rows) × factors (columns):

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

•

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

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

Ratings as Products of Factors

- Prediction: estimating the missing rating of user ***x*** for item ***i***

users (columns) × items (rows):

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	

≈

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

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

q_i = row ***i*** of ***Q***

p_x = column ***x*** of P^T

Q — items (rows) × factors (columns):

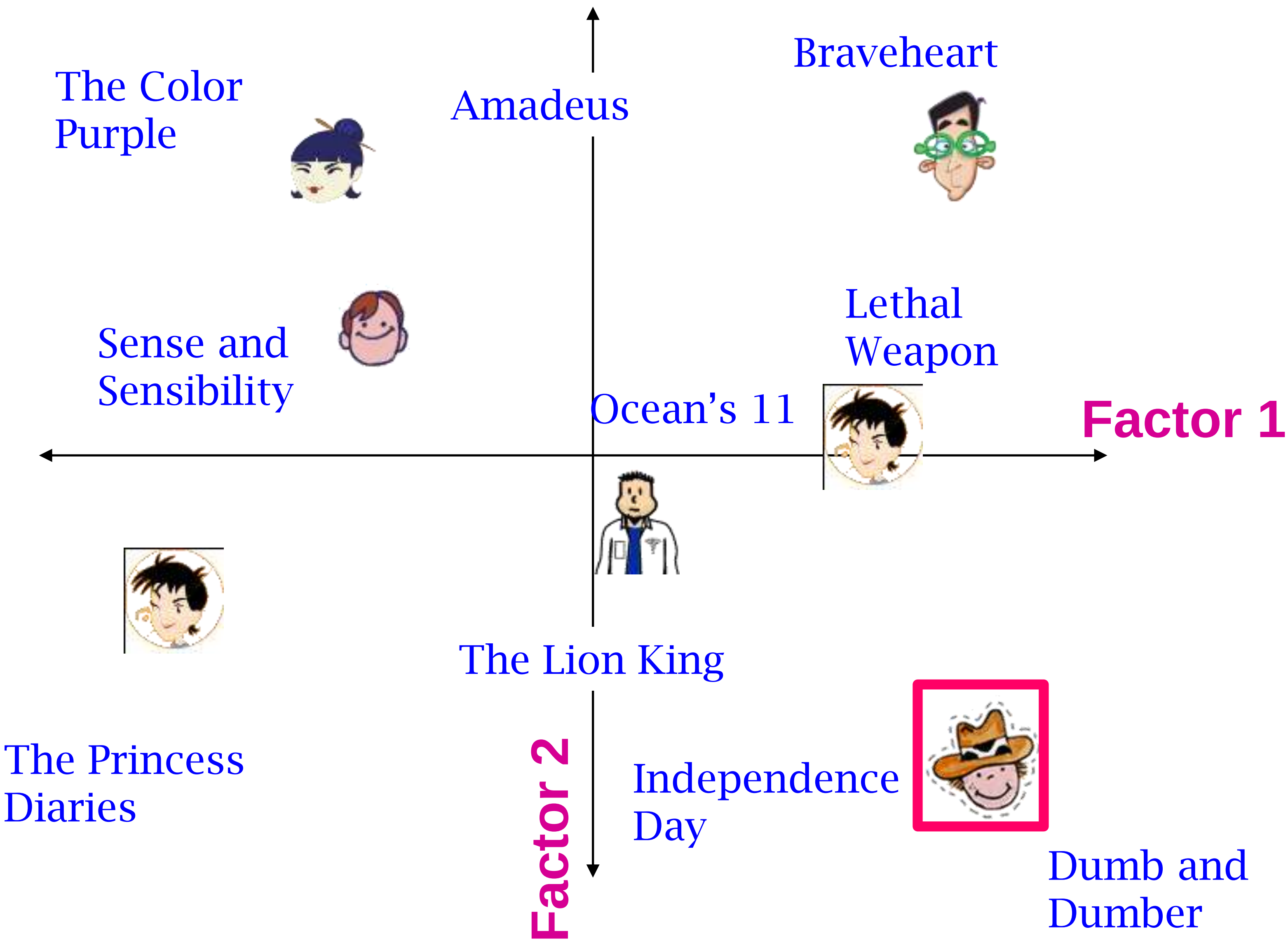
.1	-.4	.2
-.5	**.6**	**.5**
-.2	.3	.5
1.1	2.1	.3
-.7	2.1	-2
-1	.7	.3

•

P^T — factors (rows) × users (columns):

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

Latent Factor Models



Finding the Latent Factors

Latent Factor Models

- Our goal is to find matrices **P** and **Q** such that: $\min_{P,Q} \sum_{(i,x)\in R} (r_{xi} - q_i \cdot p_x)^2$

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	

≈

factors

items

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

Q

users

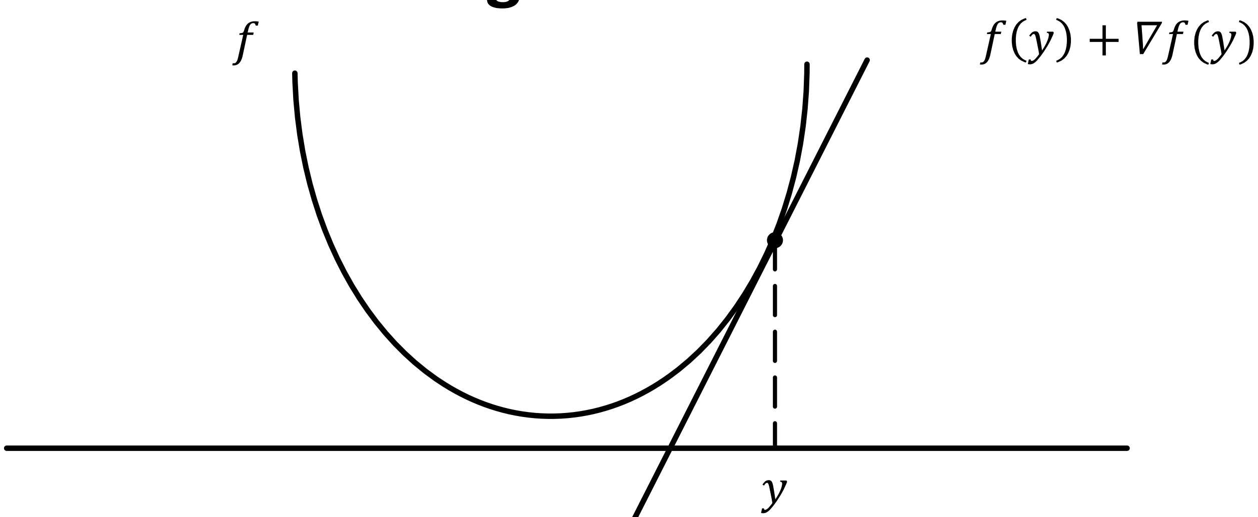
factors

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

$\boldsymbol{P}^{\mathrm{T}}$

Minimizing a function

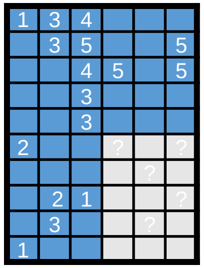
- **A simple way to minimize a function $f(x)$:**
  - Compute a derivative ∇f
  - **Start at some point y and evaluate $\nabla f(y)$**
  - **Make a step in the reverse direction of the gradient: $y = y - \nabla f(y)$**
  - **Repeat until converged**



Back to Our Problem

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

- This is a classic 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 generalize** well to unseen test data

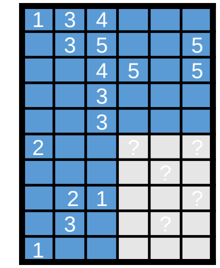


Avoiding Overfitting

- To solve overfitting we introduce **regularization**:
  - Allow <u>rich</u> model where there are <u>sufficient data</u>
  - Use <u>scarce</u> model where <u>data quantity is low</u>
- What is a rich/scarce model in our case?
  - For a user x we control factors in p_x (for item i: q_i)

- <u>Scarce</u> model could be:
  - for user x: lot of zeros in p_x
  - For item i: lot of zeros in q_i
- But function „number of zeros“ is hard to optimize
- => Use **squared norm**: $\|p_x\|^2 = p_{x,1}^2 + \cdots + p_{x,k}^2$
  - A fair approximation of „number of zeros“

Avoiding Overfitting



- **Regularization**:
  - Allow rich model where there are sufficient data
  - Shrink model where data are scarce

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

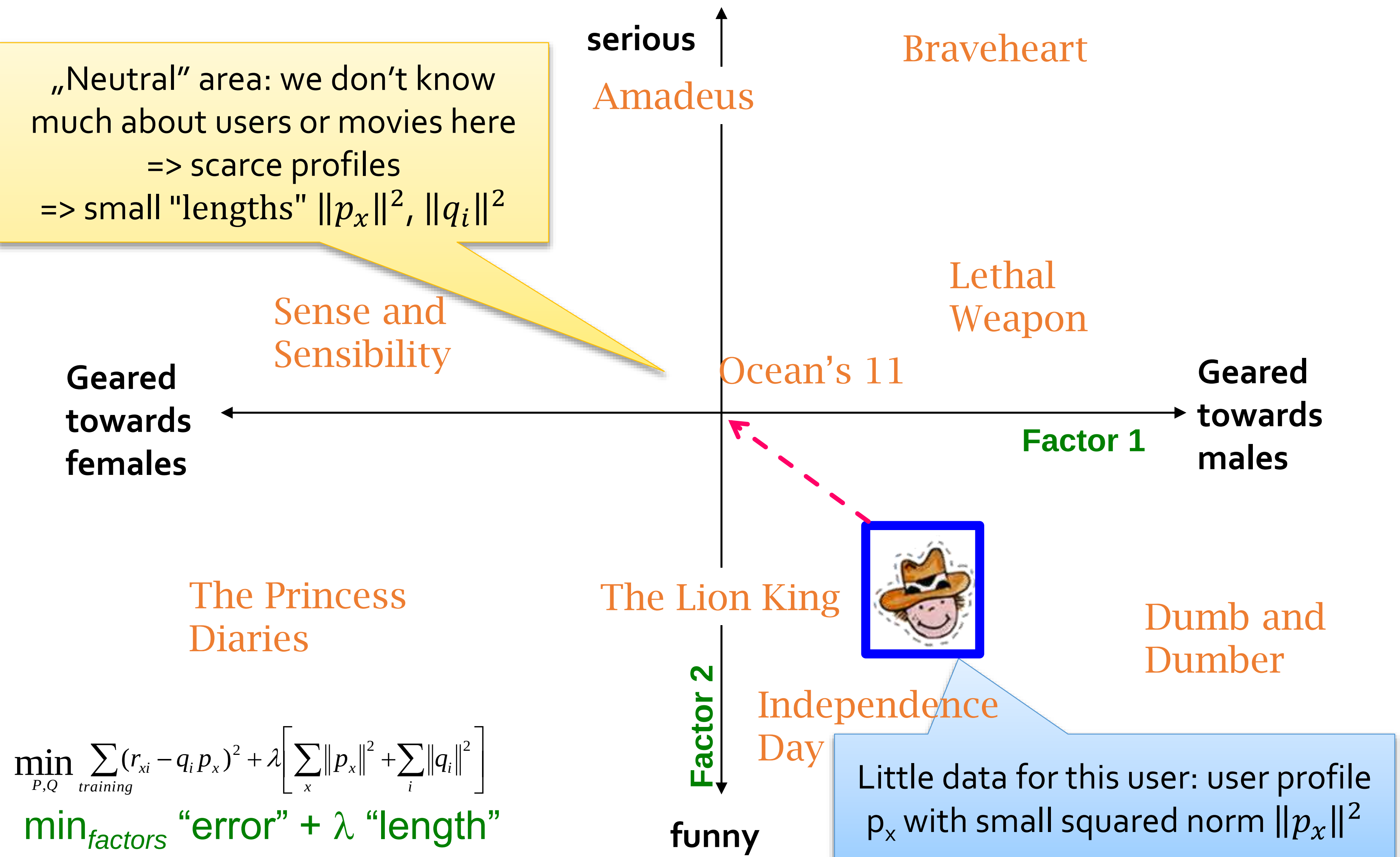
λ_1, λ_2 … user set regularization parameters

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

Role of „Length“

- Assume that user x made <u>only 1 rating</u> r_{xi}
  - We use a simple model e.g. $p_x = 0$ as the error term $(r_{xi} - q_i p_x)^2$ is at most r_{xi}^2
  - => The regularization “penalty” $\|p_x\|^2$ is also small
- Assume that user y made <u>100 ratings</u>
  - It make sense to make p_y complex ($\|p_y\|^2$>> 0) so that the sum of 100 errors $(r_{yi} - q_i p_y)^2$ remain small
  - Large $\|p_y\|^2$ is not good for minimizing the objective function but still better than having 100 large errors!
- The same for items i (freq. rated <=> „rich“ q_i)

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”

Optimizing by Stochastic Gradient Descent

Fitting a Line

- Example: Fit a straight line $y = w_1 + w_2 x$ to a set of points $(x_1, y_1), \dots . (x_n, y_n)$
- Objective function is :

$$\mathrm{Q(w)} = \sum_{i=1}^{n} Q_i(w) = \sum_{i=1}^{n} (w_1 + w_2 x_i - y_i)^2$$

(where $w = [w_1, w_2]^T$ is a 2-vector)

- **Gradient descend**:
  - We iterate over values of vector w until $\mathrm{Q(w)}$ does not improve
  - Each step changes w opposite to the direction of "fastest growth" of $\mathrm{Q(w)}$, where w is our optimization variable (2 scalars)
- We get the direction of "fastest growth" as a **gradient of $\nabla \mathrm{Q}(w)$ at a current value of w**
  - Gradient in respect to w, all other vars in $\mathrm{Q(w)}$ are constant!
  - Gradient = [first derivative of $\mathrm{Q(w)}$ by w_1; … of $\mathrm{Q(w)}$ by w_2]$^\mathrm{T}$

Gradient Descent

Procedure GD(Q(w)) #for minimization of Q(w)

- Input: Objective function Q(w)
  - w is a vector of m parameters to be optimized
- Init: Assign w a start value (may be random)
- Repeat until convergence (e.g. Q(w) gets no smaller): $\mathrm{w} \coloneqq \mathrm{w} - \alpha \nabla Q(w)$

- Example for $\mathrm{Q(w)} = \sum_{i=1}^{n} Q_i(w) = \sum_{i=1}^{n} (w_1 + w_2 x_i - y_i)^2$
- Since $\mathrm{Q(w)} = \sum_{i=1}^{n} Q_i(w)$, we have $\nabla Q(w) = \sum_{i=1}^{n} \nabla Q_i(w)$
- Each $\nabla Q_i(w)$ is (use chain rule for derivatives):

$$\nabla Q_i(w) = \begin{bmatrix} \frac{dQ_i(w)}{dw_1} \\ \frac{dQ_i(w)}{dw_2} \end{bmatrix} = \begin{bmatrix} 2(w_1 + w_2 x_i - y_i) \\ 2(w_1 + w_2 x_i - y_i) x_i \end{bmatrix}$$

Stochastic Gradient Descent /1

- Assume that the objective function Q(w) is a sum $Q(w) = \sum_{i=1}^{n} Q_i(w)$
  - Typically a $Q_i(w)$ comes from i-th training sample
  - => gradient looks like this: $\nabla Q(w) = \sum_{i=1}^{n} \nabla Q_i(w)$
- **Stochastic Gradient Descent**
  - Instead of computing all $\nabla Q_1(w)$,…, $\nabla Q_n(w)$ and then making a step $w := w - \alpha \nabla Q(w)$,…
  - … we make a step after computing each of the "partial gradients" $\nabla Q_i(w)$

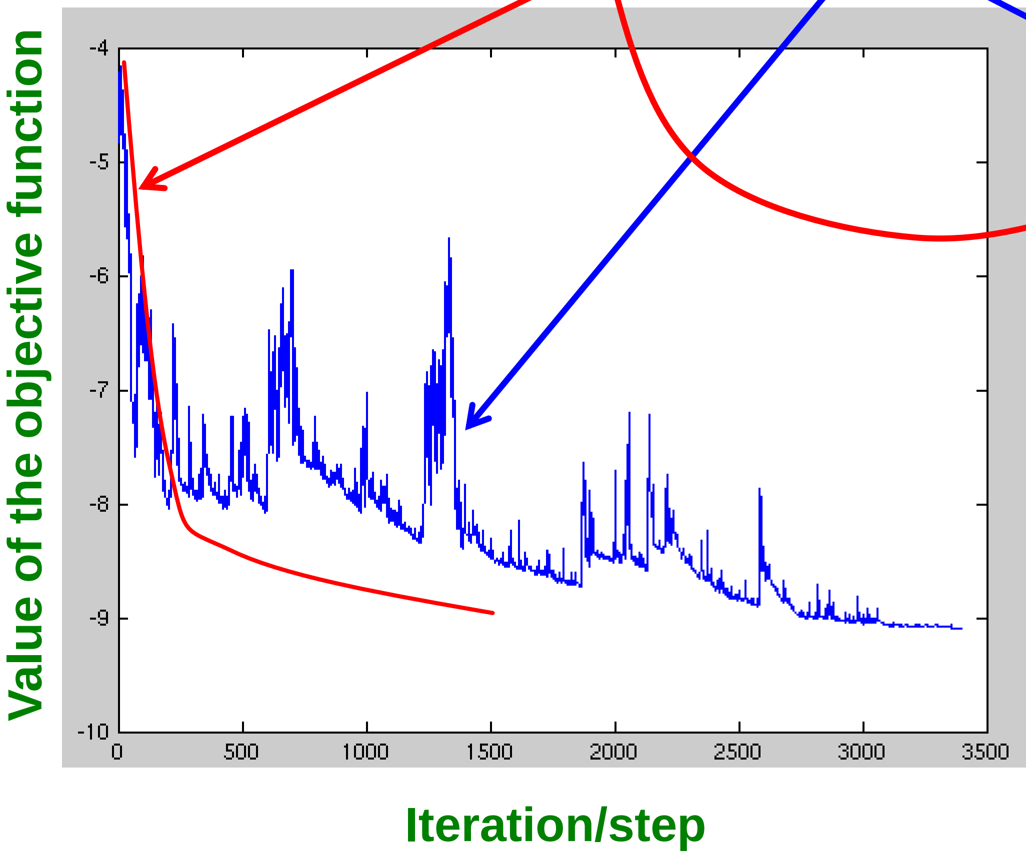
Stochastic Gradient Descent /2

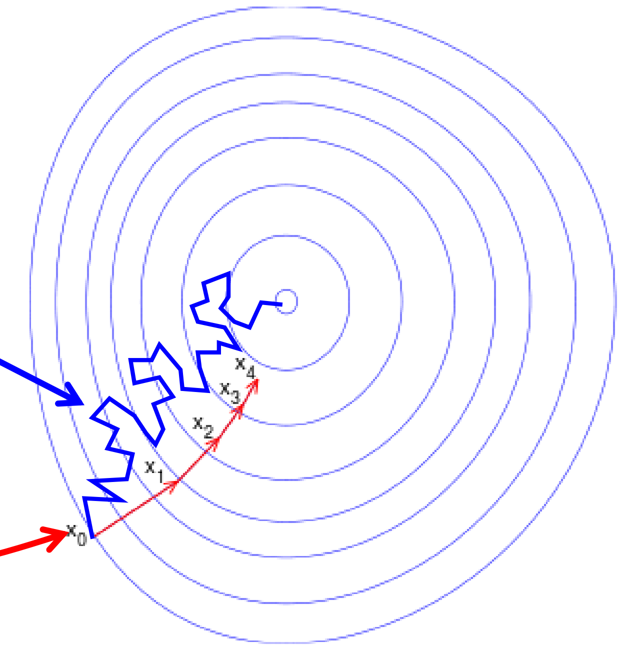
Procedure SGD(Q(w)) #for minimization of Q(w)

- Input: Objective function Q(w)
  - w is a vector of m parameters to be optimized
- Init: Assign w a start value (may be random)
- Repeat until convergence: # outer loop
  - For i = 1 to n: # inner loop
    - $\mathrm{w} \coloneqq \mathrm{w} - \beta \nabla Q_i(w)$

SGD vs. GD

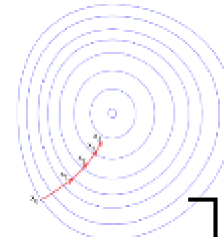
- **Convergence of GD vs. SGD**





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!

Gradient Descent for Latent Factors



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

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

- **Gradient decent:**
  - Initialize ***P*** and ***Q*** (using **SVD** with missing ratings = 0)

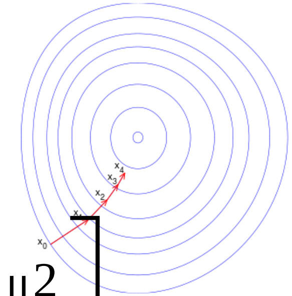
Singular Value Decomposition

  - Do gradient descent:
    - $\boldsymbol{P} \leftarrow \boldsymbol{P} - \eta \cdot \nabla \mathbf{P}$, where $\nabla \boldsymbol{P}$ is …
    - $\boldsymbol{Q} \leftarrow \boldsymbol{Q} - \eta \cdot \nabla \mathbf{Q}$
    - where $\nabla \boldsymbol{Q}$ is gradient/derivative of matrix ***Q***:
      $\nabla Q = [\nabla q_{if}]$ and $\nabla q_{if} = \sum_{x,i} -2\left(r_{xi} - q_i p_x\right) p_{xf} + 2\lambda_2 q_{if}$
      - Here $\boldsymbol{q_{if}}$ is entry ***f*** of row $\boldsymbol{q_i}$ of matrix ***Q***

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

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

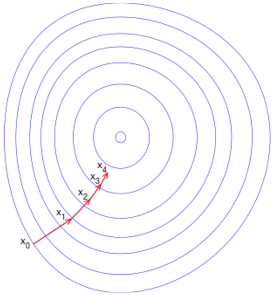
Stochastic Gradient Descent



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

- **Gradient Descent (GD) vs. Stochastic GD**
  - **Idea:** Instead of evaluating gradient over <u>all ratings</u> evaluate it for <u>an individual rating</u> and make a step
- **GD:** $\boldsymbol{Q} \leftarrow \boldsymbol{Q} - \eta \left[\sum_{\boldsymbol{r_{xi}}} \nabla \boldsymbol{Q}(\boldsymbol{r_{xi}}) \right]$
- **SGD:** $\boldsymbol{Q} \leftarrow \boldsymbol{Q} - \mu \nabla \boldsymbol{Q}(\boldsymbol{r_{xi}})$
    - $\varepsilon_{xi} = 2(r_{xi} - q_i \cdot p_x\ )$ (derivative of the "error")
    - $q_i \leftarrow q_i + \mu_1\ (\varepsilon_{xi}\ p_x - \lambda_2\ q_i)$ (update equation)
  - **Faster convergence!**
    - Need more steps but each step is computed much faster

Stochastic Gradient Descent



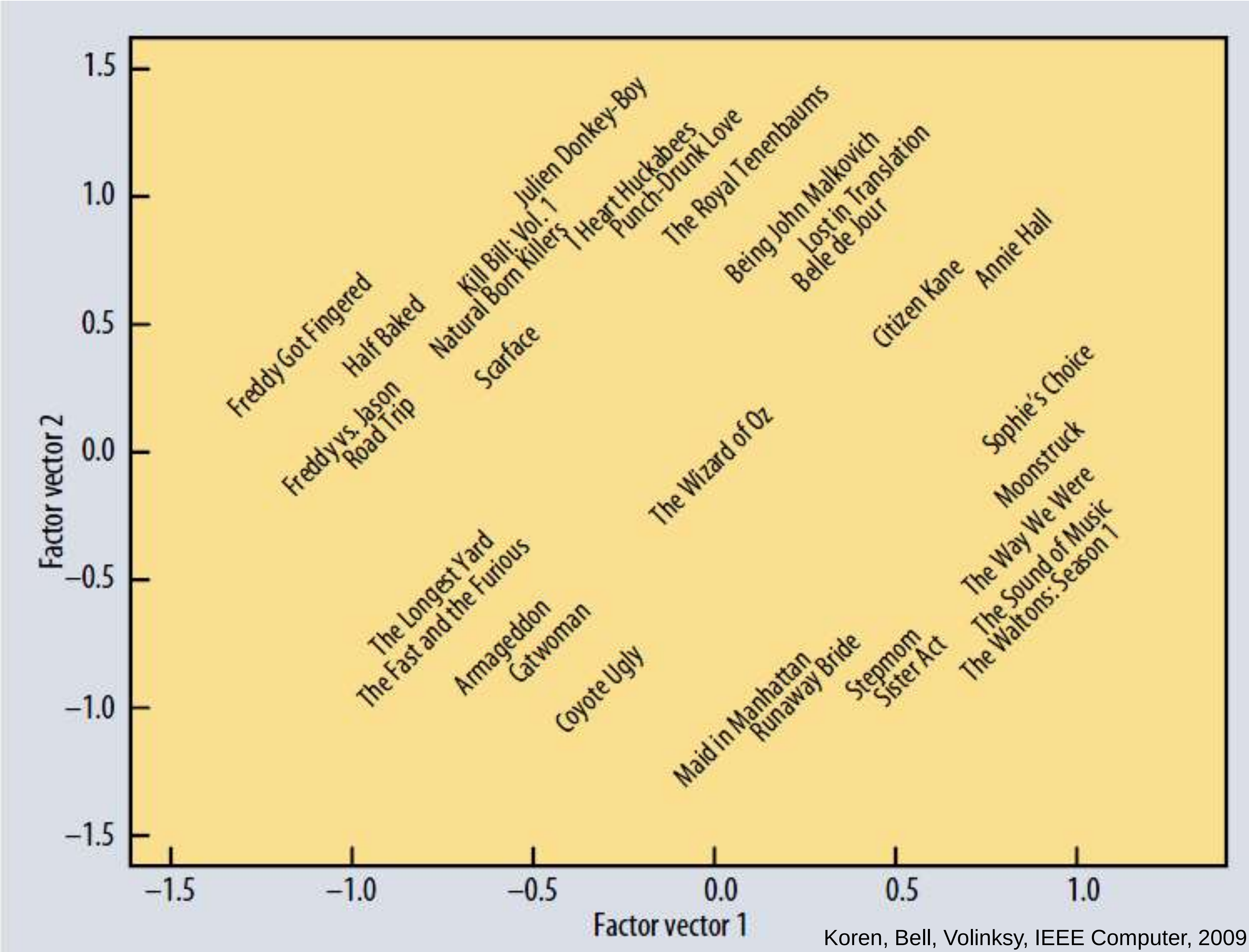
- **Stochastic gradient decent:**
  - Initialize ***P*** and ***Q*** (using SVD, pretend missing ratings are 0)
  - Then iterate over the ratings (multiple times if necessary) and update factors:

  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)

  μ … learning rates

- **Two loops:**
  - Repeat until convergence:
    - For each $\mathbf{r}_{xi}$
      - Compute gradient, do a "step"



Koren, Bell, Volinksy, IEEE Computer, 2009

The Netflix Prize:
The Winner

Modeling Local & Global Effects


- **Global:**
  - 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:**

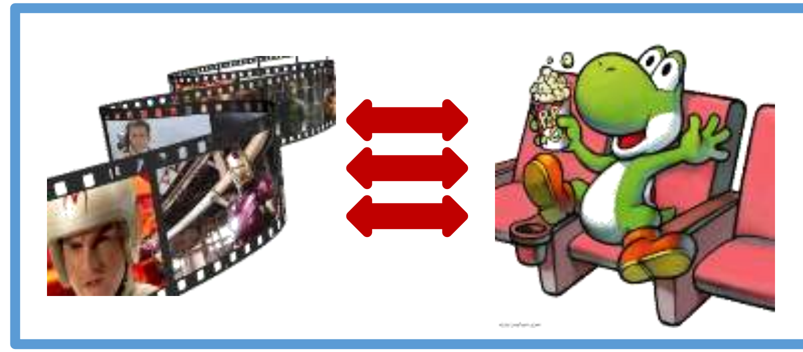
    ***Joe* will rate *The Sixth Sense* 3.8 stars**

Modeling Biases and Interactions

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 It All Together

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

Overall mean rating

Bias for user ***x***

Bias for movie ***i***

User-Movie interaction

- **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:**

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

goodness of fit

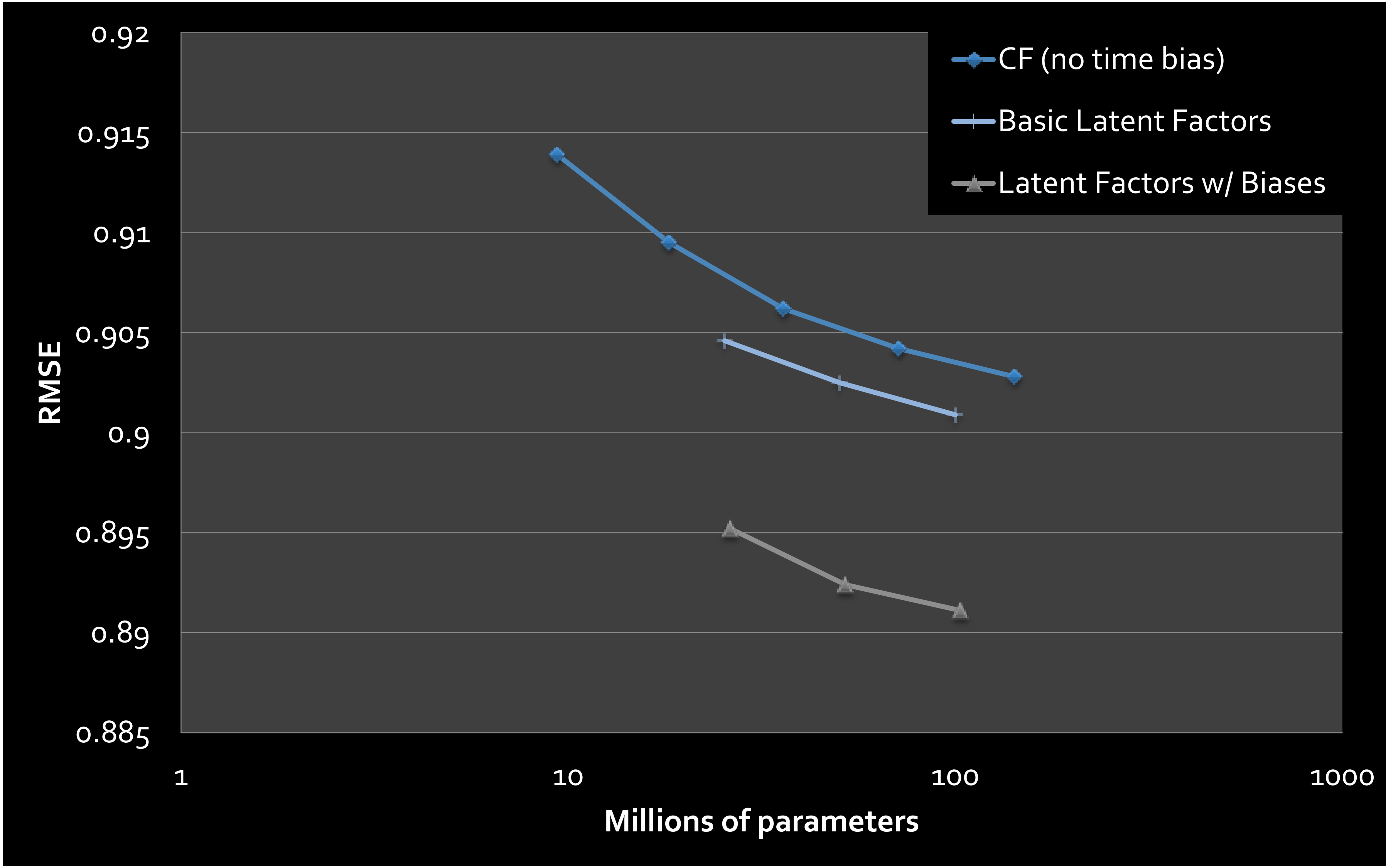
$$+\left(\lambda_1 \sum_i \|q_i\|^2 + \lambda_2 \sum_x \|p_x\|^2 + \lambda_3 \sum_x \|b_x\|^2 + \lambda_4 \sum_i \|b_i\|^2\right)$$

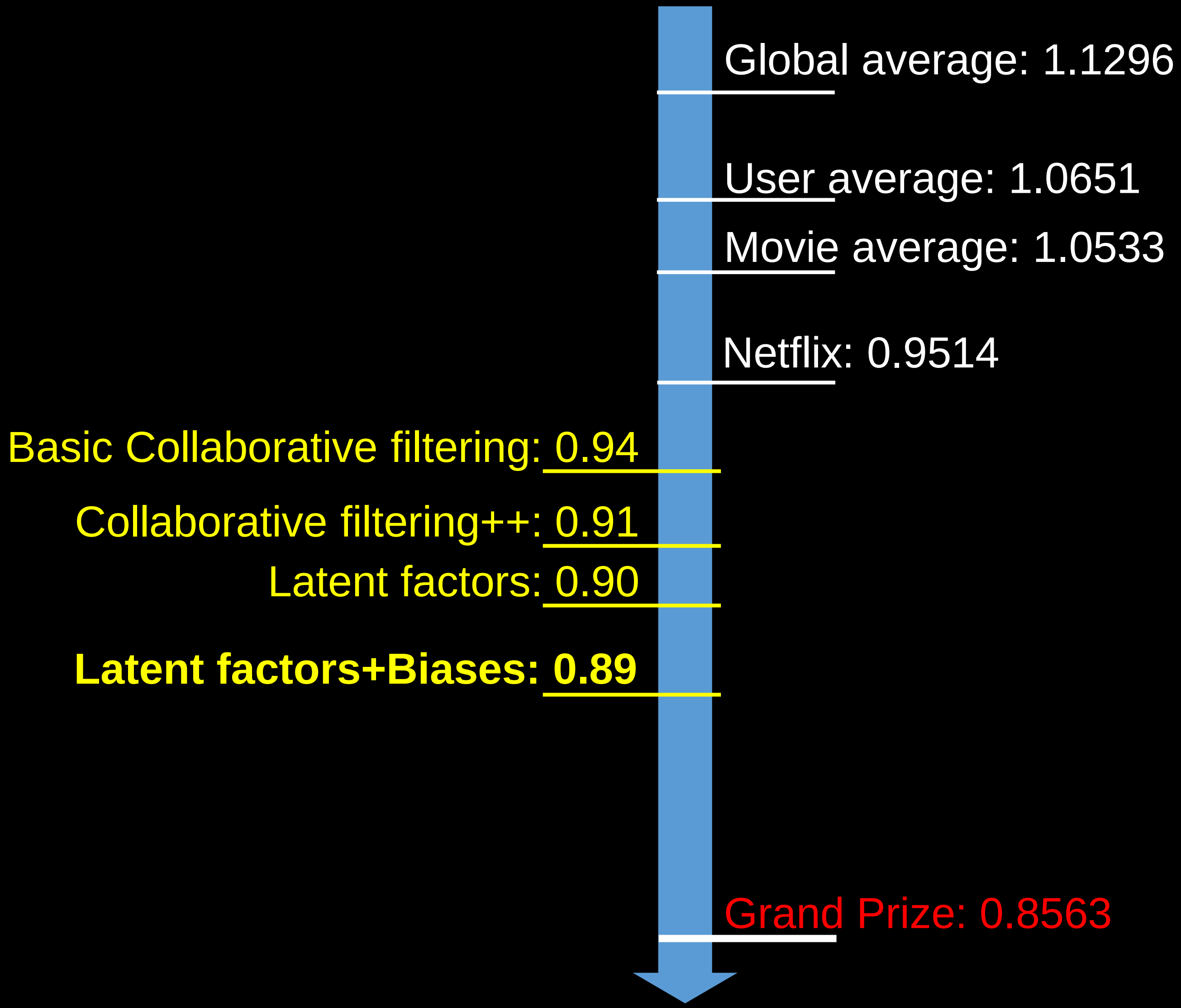
regularization

λ is selected via grid-search on a validation set

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

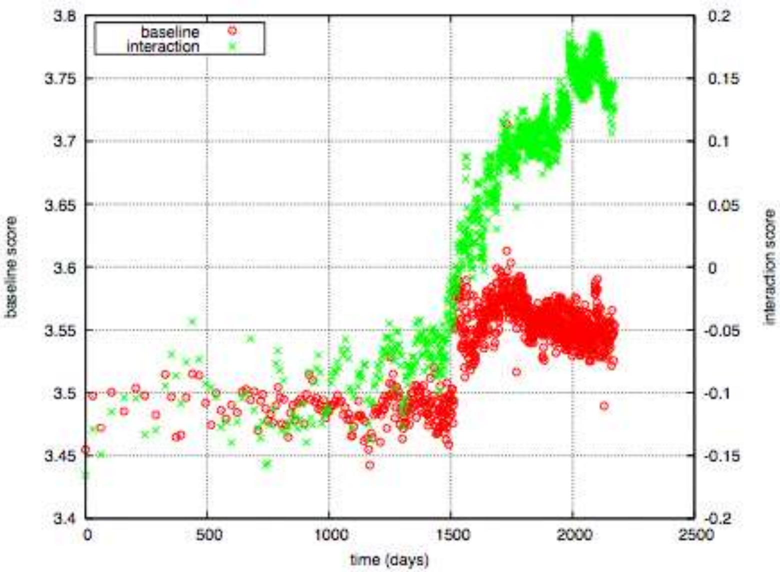
Performance of Various Methods

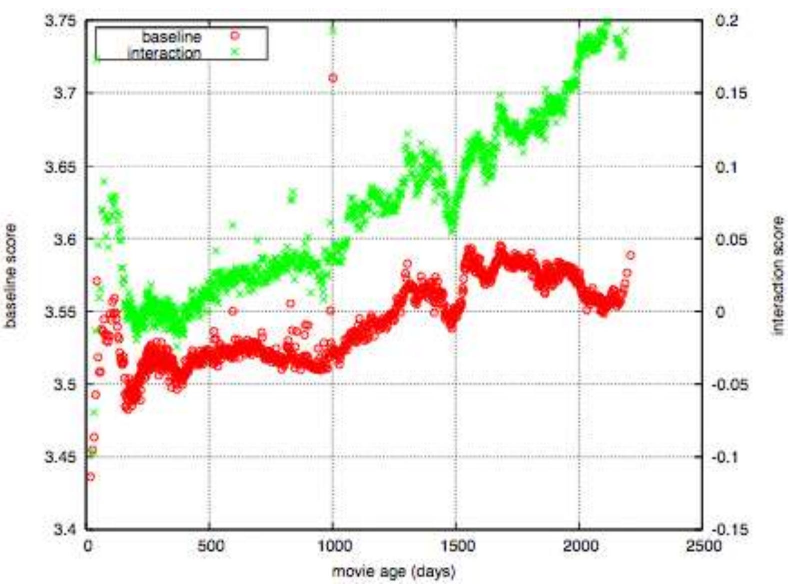



Global average: 1.1296
User average: 1.0651
Movie average: 1.0533
Netflix: 0.9514
Basic Collaborative filtering: 0.94
Collaborative filtering++: 0.91
Latent factors: 0.90
Latent factors+Biases: 0.89
Grand Prize: 0.8563

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

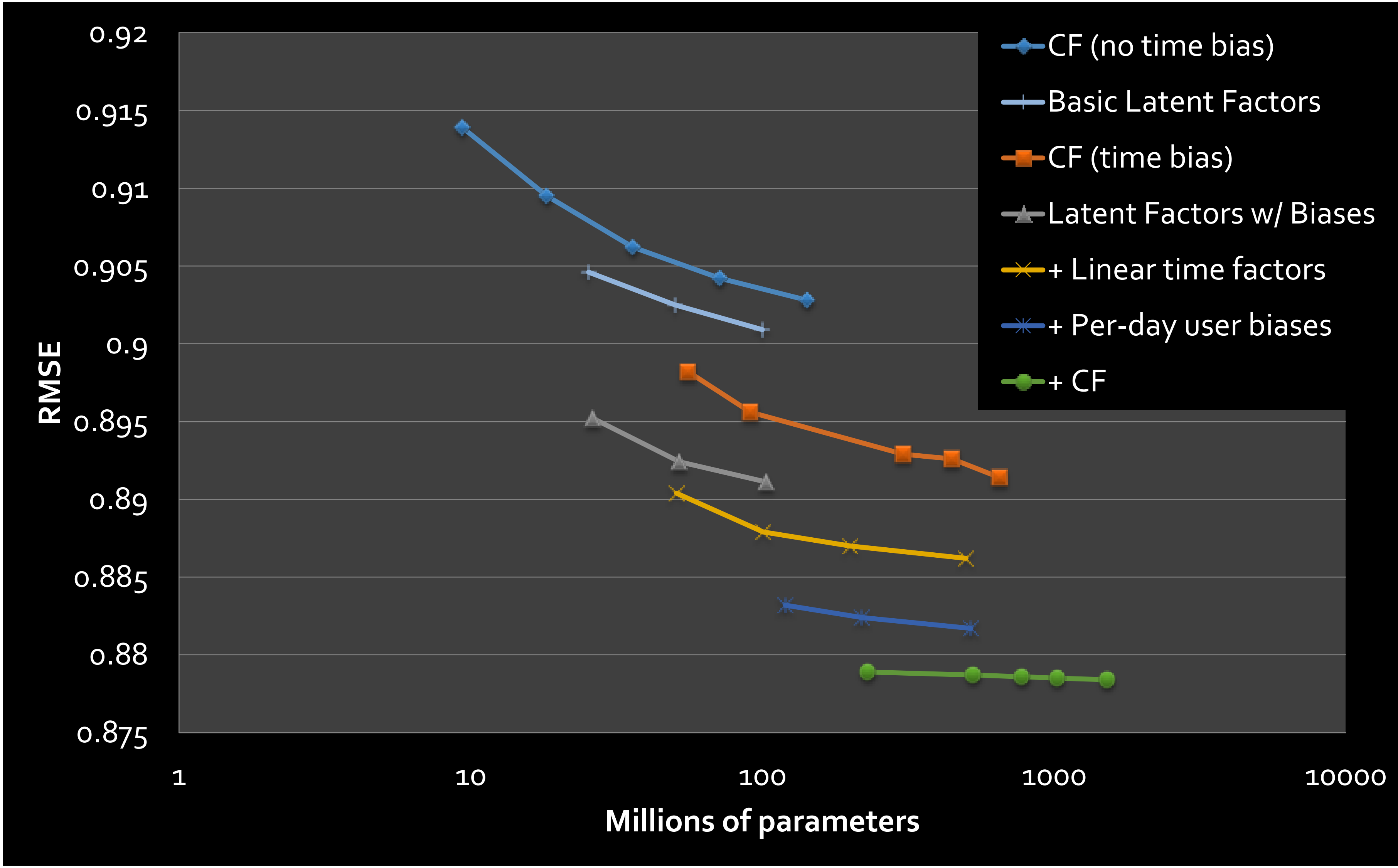
- **Original model:**
  $r_{xi} = \mu + b_x + b_i + q_i \cdot p_x$
- **Add time dependence to biases:**
  $r_{xi} = \mu + b_x(t) + b_i(t) + q_i \cdot p_x$
  - Make parameters b_x and b_i to depend on time
  - **(1)** Parameterize time-dependence by linear trends
    (2) Each bin corresponds to 10 consecutive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

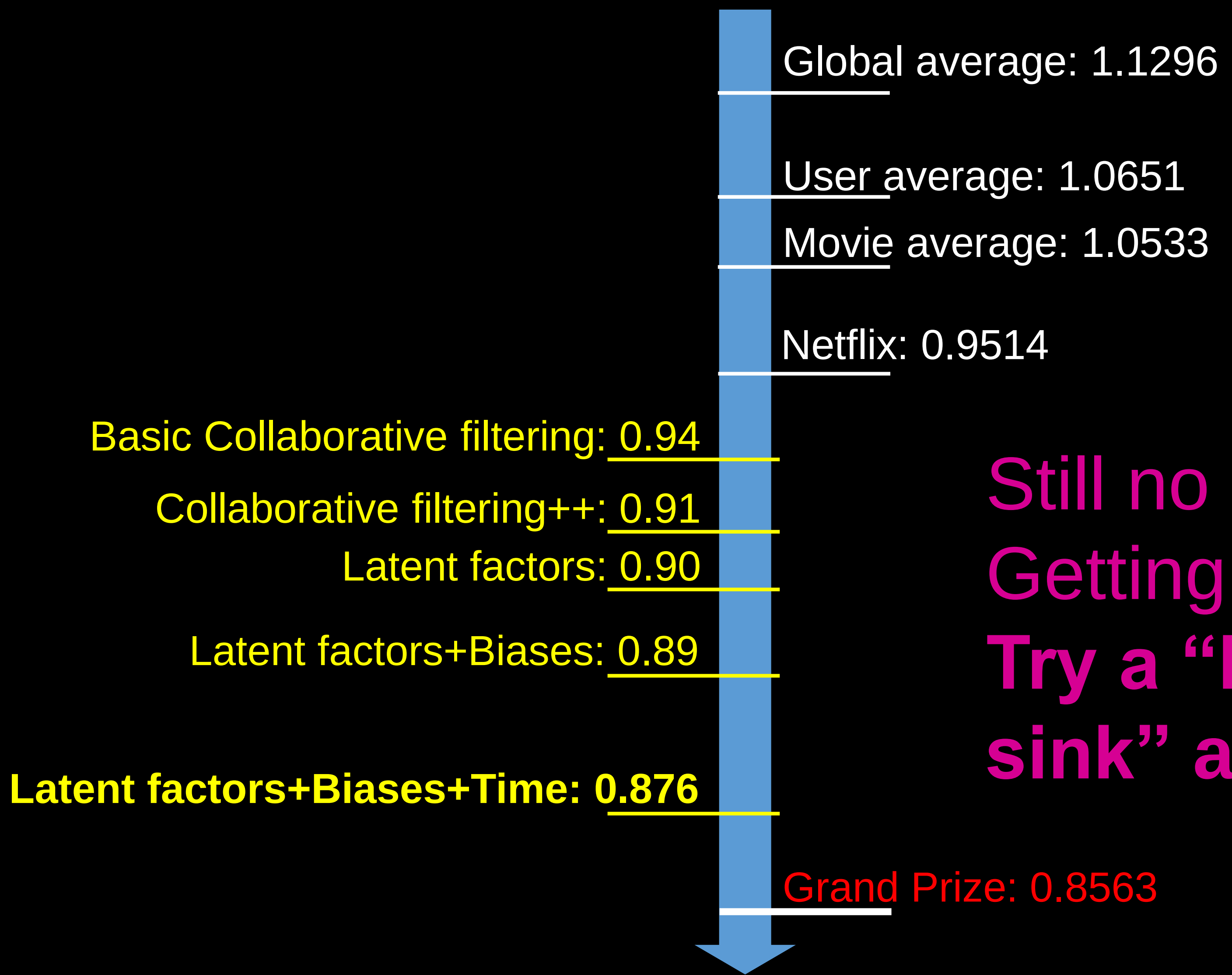
$$b_i(t) = b_i + b_{i,\mathrm{Bin}(t)}$$

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

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

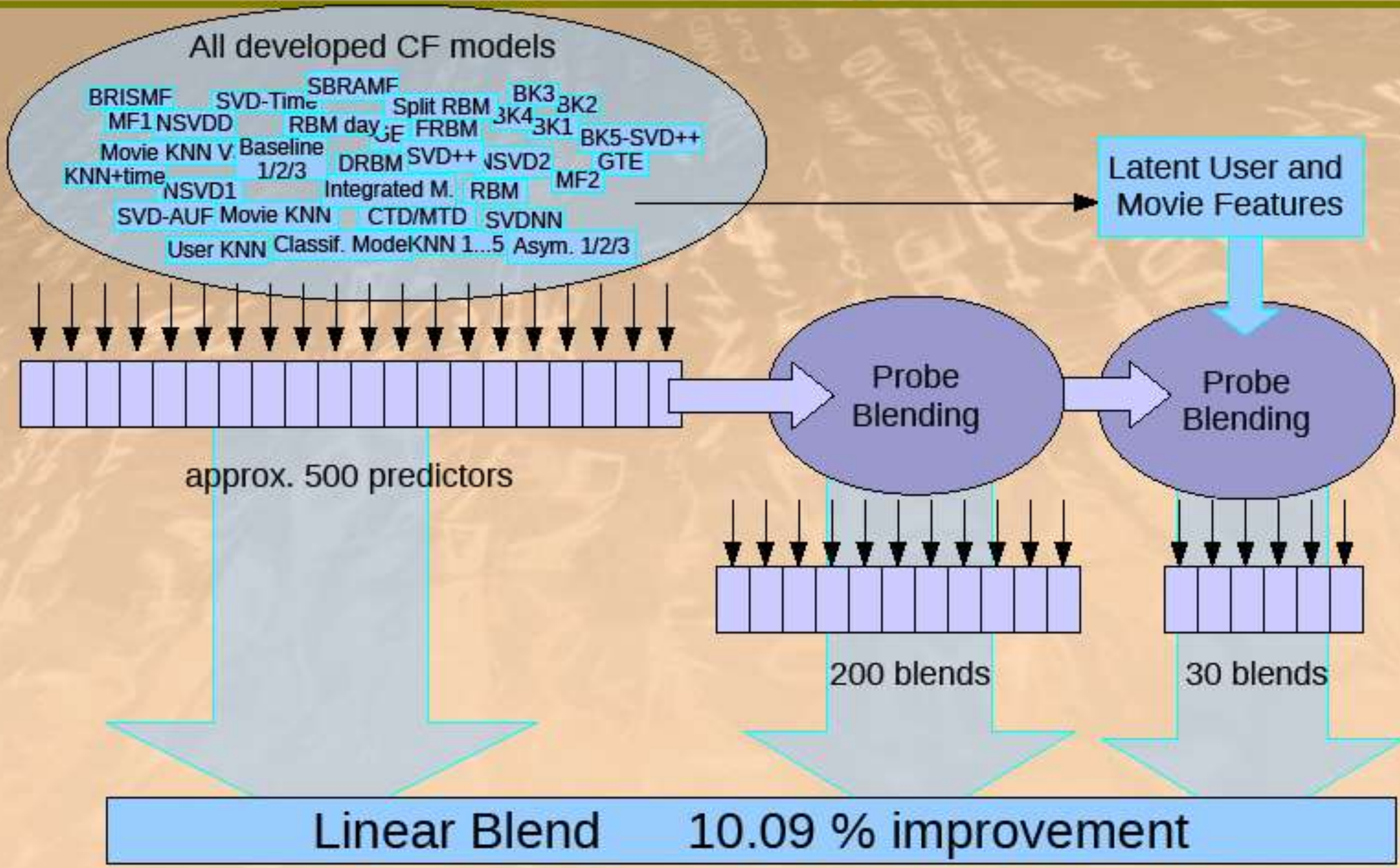
Adding Temporal Effects





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

The big picture
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

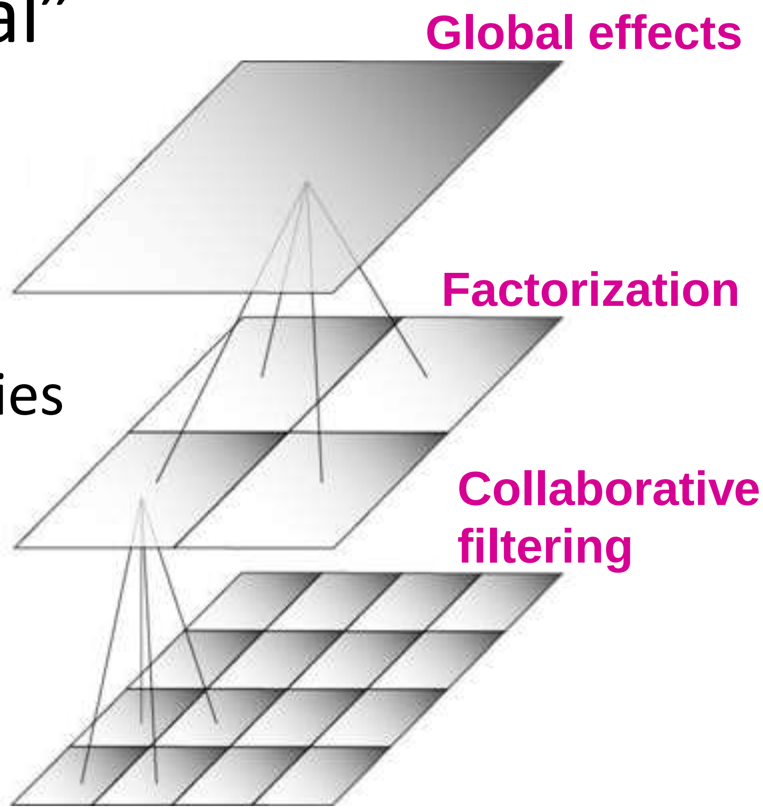
Display top 20 leaders.

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	0.8[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

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
  - **Factorization:**
    - Addressing "regional" effects
  - **Collaborative filtering:**
    - Extract local patterns



References

- Yehuda Koren, Robert Bell and Chris Volinsky: Matrix Factorization Techniques for Recommender Systems, IEEE Computer, August 2009, http://goo.gl/D2JOa9
  - Easy-to-read paper on modern recommendation techniques
- Albert Au Yeung, Matrix Factorization: A Simple Tutorial and Implementation in Python, Blog post, 16 September 2010, http://goo.gl/kzoLaO
- Fun: James Surowiecki, The Wisdom of Crowds, Doubleday; Anchor 2004
  - Wikipedia (en): http://en.wikipedia.org/wiki/The_Wisdom_of_Crowds
  - Wikipedia (de): http://de.wikipedia.org/wiki/Die_Weisheit_der_Vielen

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
  - http://www2.research.att.com/~volinsky/netflix/bpc.html
  - http://www.the-ensemble.com/

Thank you.

Questions?