Mining Massive Datasets

Lecture 4

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.

Repetition: Contrasting Recommendation Methods

Last Lecture vs. Today

- We have already seen two approaches to recommender systems:
  - A. Collaborative filtering
  - B. Content-based recommenders

- Today:
  - C. Latent factor models

Item-Item Collaborative Filtering

users

$S = \{3, 6\}$

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

- 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 (of this user x) over all items in S
- => Rating $r_{5\,1}$ is predicted as a weighted sum of other rows (= **ratings of similar items** by this user)

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	

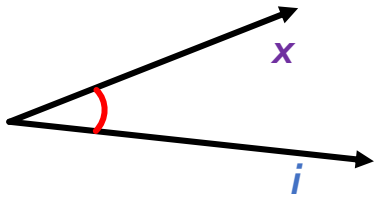
- Find set Q of **users** similar to target user x=5 who have rated item $i = 1$, and predict $r_{5\,1}$ as a weighted sum of ratings for item 1 by all users in Q
- => Rating $r_{5,1}$ is predicted as a weighted sum of other columns (ratings of similar **other users**)

Recall: 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 similar items rated by this user (also a k-vector)

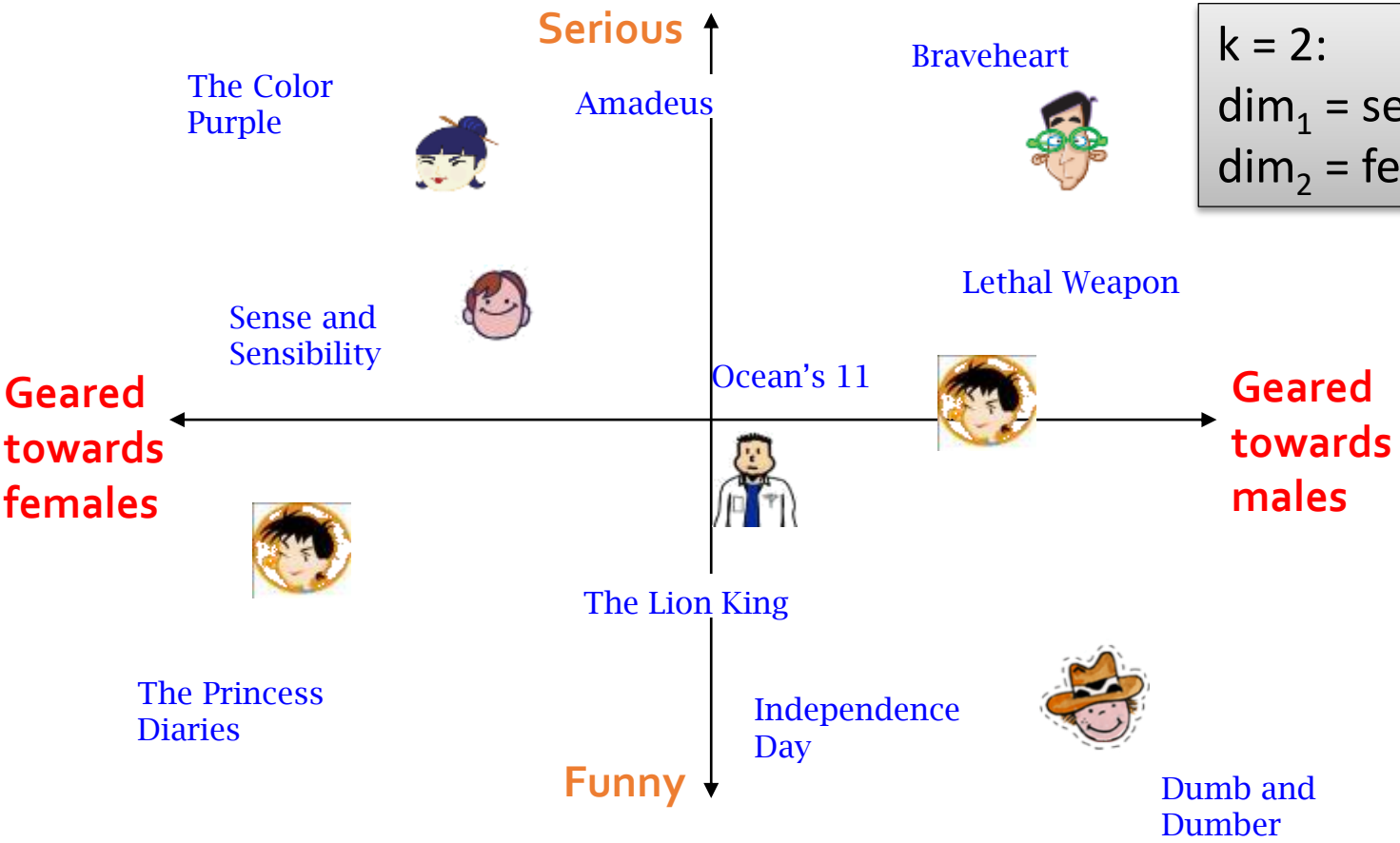
- Prediction heuristic:
  - A content-based prediction $\boldsymbol{r_{x\,i}}$ is approximated as similarity of these two k-vectors: $\boldsymbol{r_{x\,i} = u(x, i)}$
  - I.e., given a user profile $\boldsymbol{x}$ and item profile $\boldsymbol{i}$, estimate their similarity as:

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



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



C. Latent Factor Models

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 **weighted sum of other ratings** from the utility matrix

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

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	

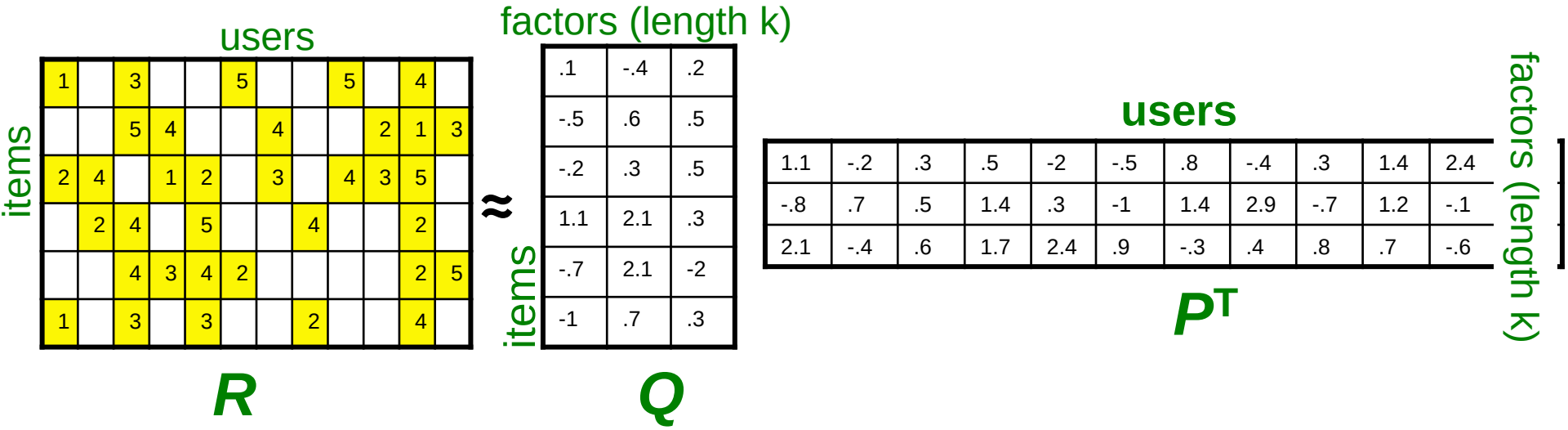
$$? = \frac{p_x \cdot q_i}{\| p_x \| \cdot \| q_i \|} = C_{x,i} \cdot \begin{bmatrix} p_{x,1} \\ \vdots \\ p_{x,k} \end{bmatrix} \cdot \begin{bmatrix} q_{i,1} \\ \vdots \\ q_{i,k} \end{bmatrix}$$

- Representing each item by a k-vector q_i and each user by k-vector p_x is a promising idea
- But can we replace **hand-crafted** item-profiles by (new) **synthetic profiles** 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)



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

- 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
  - We want the reconstruction error to be small on known ratings
  - 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

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	

≈

Q

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

items

factors

•

users

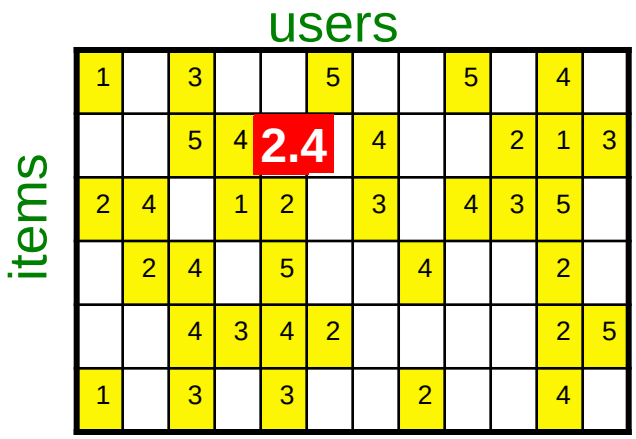
factors

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

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



$\approx$

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

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

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

items × factors

Q

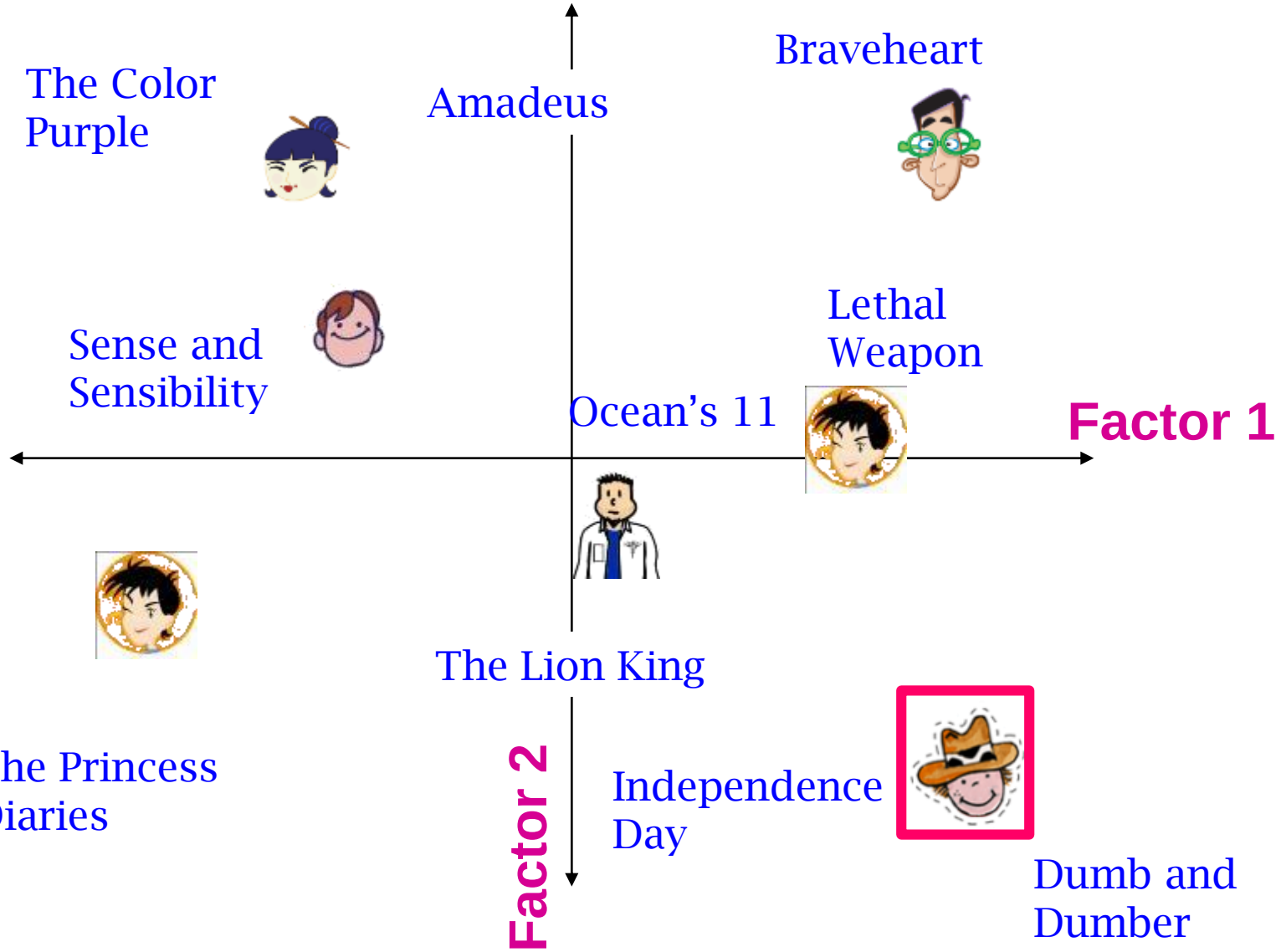
•

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



Finding the Latent Factors

Evaluating Predictions

- How to compare predictions with known ratings?
- **Root-mean-square error** (**RMSE**), details: link
  - $\sqrt{\frac{1}{N}\sum_{xi}(r_{xi} - r^*_{xi})^2}$
  - where $\boldsymbol{r_{xi}}$ is predicted, $\boldsymbol{r^*_{xi}}$ is the true rating of ***x*** on ***i,***
  - ***N*** is the number of ratings (= # of ***(x,i)*** combinations)
- Assume that number of ratings N is fixed
- Equivalent to this this is **sum-of-squared-errors** (**SSE**):
  - $\sum_{xi}(r_{xi} - r^*_{xi})^2$
- Why equivalent?

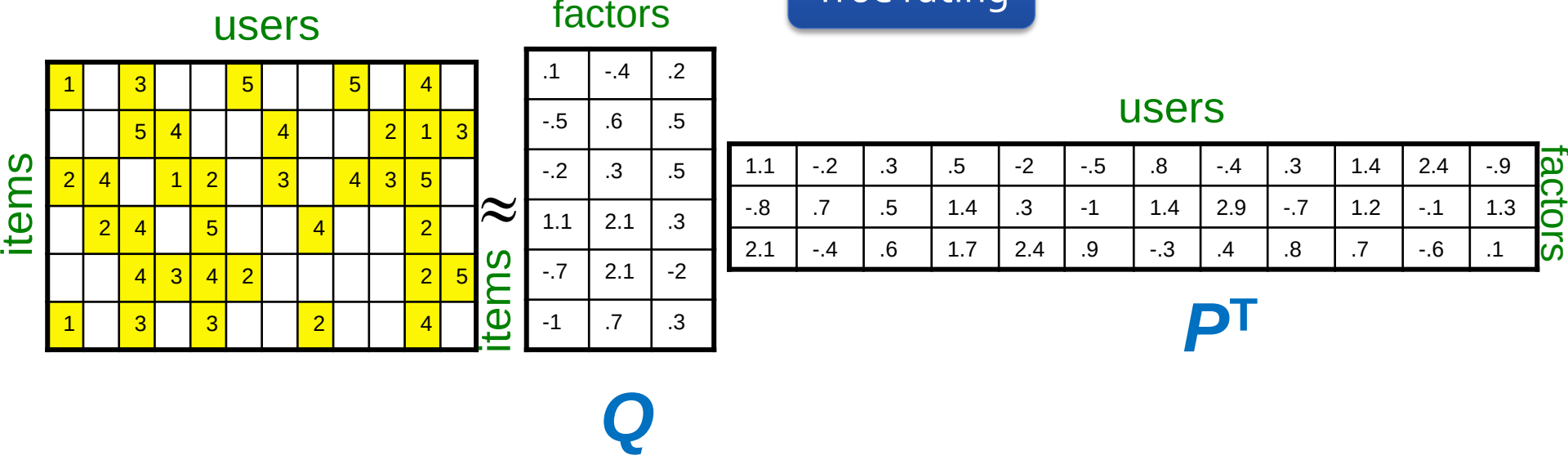
Latent Factor Models

- Our goal is to find matrices **P** and **Q** such which minimize SSE:

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

Estimation

True rating



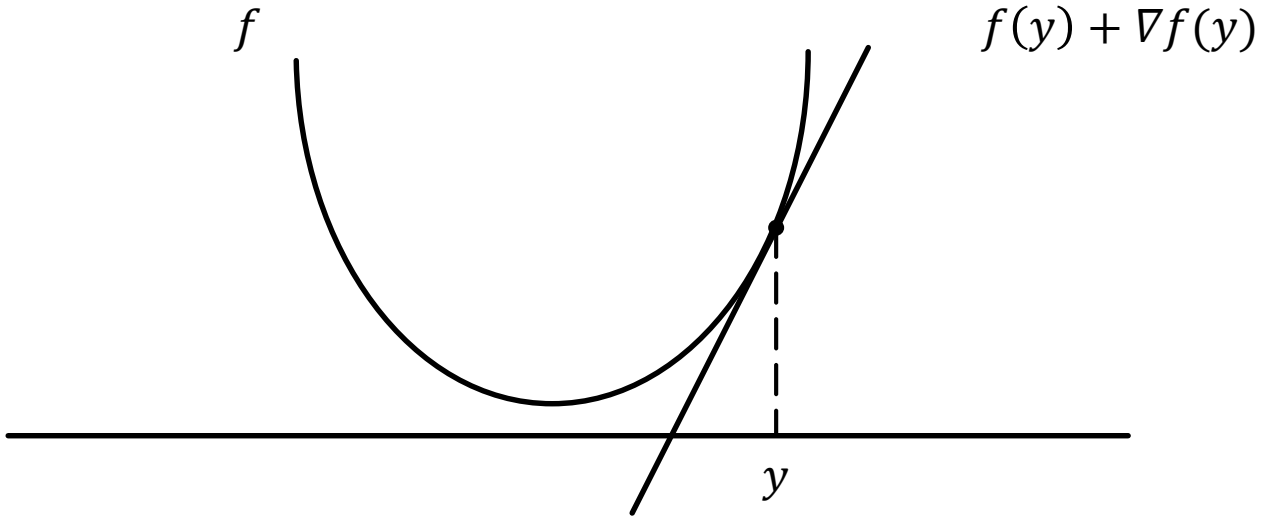
1		3			5			5		4	
		5	4			4			2	1	3
2	4		1	2		3		4	3	5	
	2	4		5			4			2	
		4	3	4	2					2	5
1		3		3			2			4	

.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

Minimizing a function

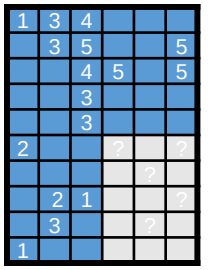
- **A simple way to minimize a function $f(x)$:**
  - Compute a derivative ∇f
  - Start at some point y and evaluate $\nabla f(y)$
  - Make a step in the reverse direction of the gradient: $y = y - \nabla f(y)$
  - Repeat until converged



Back to Our Problem

- We want to minimize SSE for unseen test data
- **Idea: Minimize SSE on training data**
  - We want a large ***k*** (# of factors) to capture all complexity
  - But, **SSE** on **test 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 to irrelevant details
  - That is it fits too well the training data and thus **not generalize** well to unseen test data



Overfitting = Fitting to Irrelevant Details

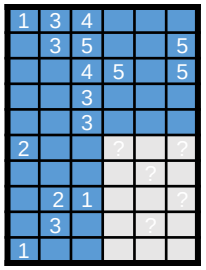
How could an *overfitting AI* interpret these signs?

- *Red cars are not allowed to overtake black cars*
- *Red "square" tracks are not allowed to overtake black cars*

Avoiding Overfitting

- To solve overfitting we introduce **regularization**:
  - Allow rich model where there are sufficient data
  - Use scarce model where data quantity is low
- What is a rich/scarce model in our case?
  - For a user x we control factors in p_x (for item i: q_i)
- Scarce 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**: $\|\boldsymbol{p_x}\|^2 = \boldsymbol{p_{x,1}^2} + \cdots + \boldsymbol{p_{x,k}^2}$
  - A fair approximation of „number of zeros“

Avoiding Overfitting



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

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

"error" "length"

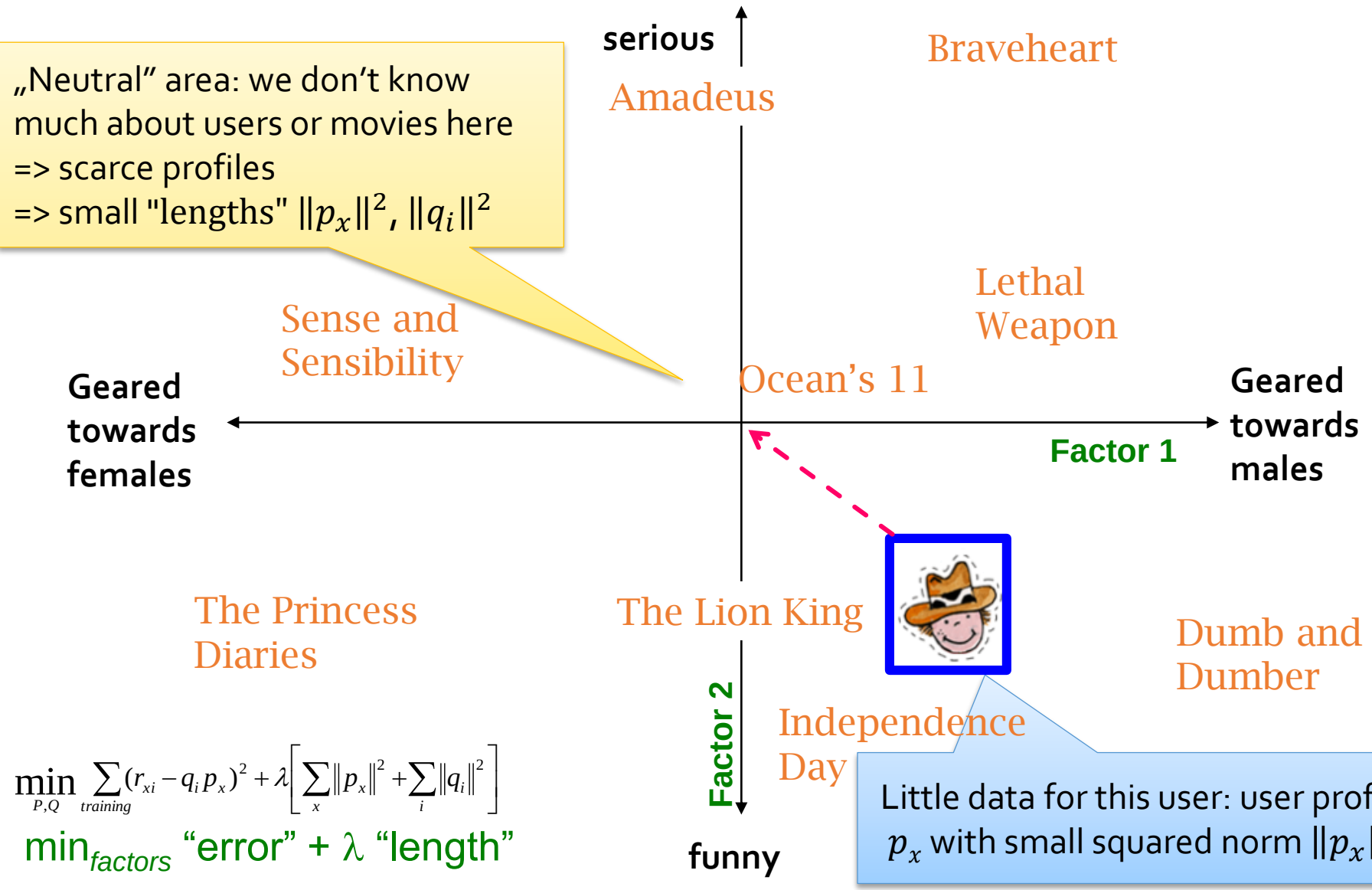
λ_1, λ_2 … user set regularization parameters

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

Role of „Length“

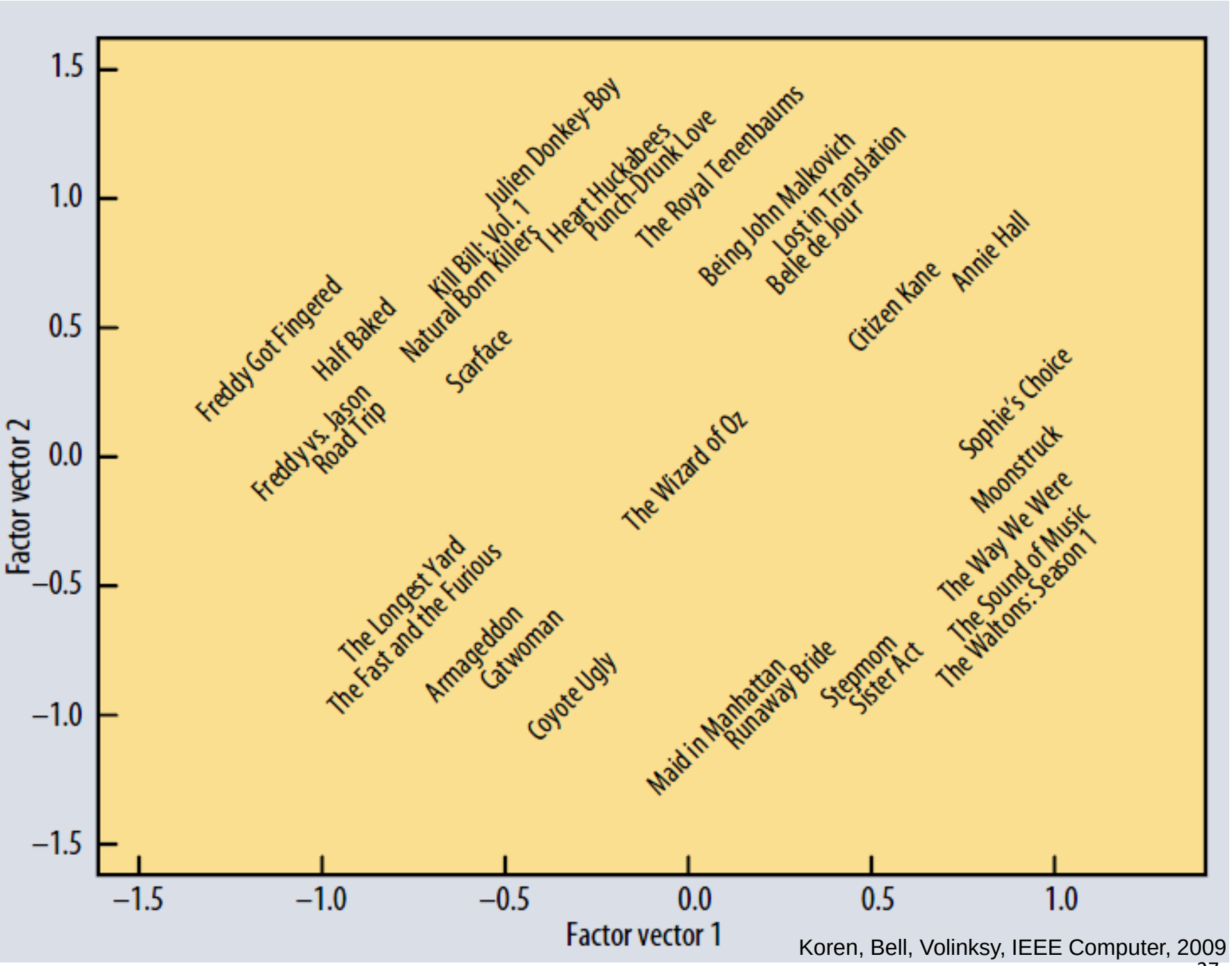
- Assume that user x made only 1 rating r_{xi}
  - We use a simple model, e.g. $p_x = 0$ as the error term $(r_{xi} - q_i p_x)^2$ is at most r_{xi}^2
  - => The regularization “penalty” $\|p_x\|^2$ is also small
- Assume that user y made 100 ratings
  - It make sense to make p_y complex ($\|p_y\|^2 >> 0$) 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"



Koren, Bell, Volinksy, IEEE Computer, 2009

Optimizing by Stochastic Gradient Descent

Example: Fitting a Line

- We want to fit a straight line $w_1 + w_2 x$ in $\mathbb{R}^2$ to a set of points $(x_1, y_1), \ldots . (x_n, y_n)$
- => Find "best" values for the parameters w_1, w_2
  - Let $w = [w_1, w_2]^T$ is a 2-vector to be optimized
- How to do this? And how we measure how "good" is w?

- 1. We introduce an objective function $\mathrm{Q(w)}$ to "measure" w:

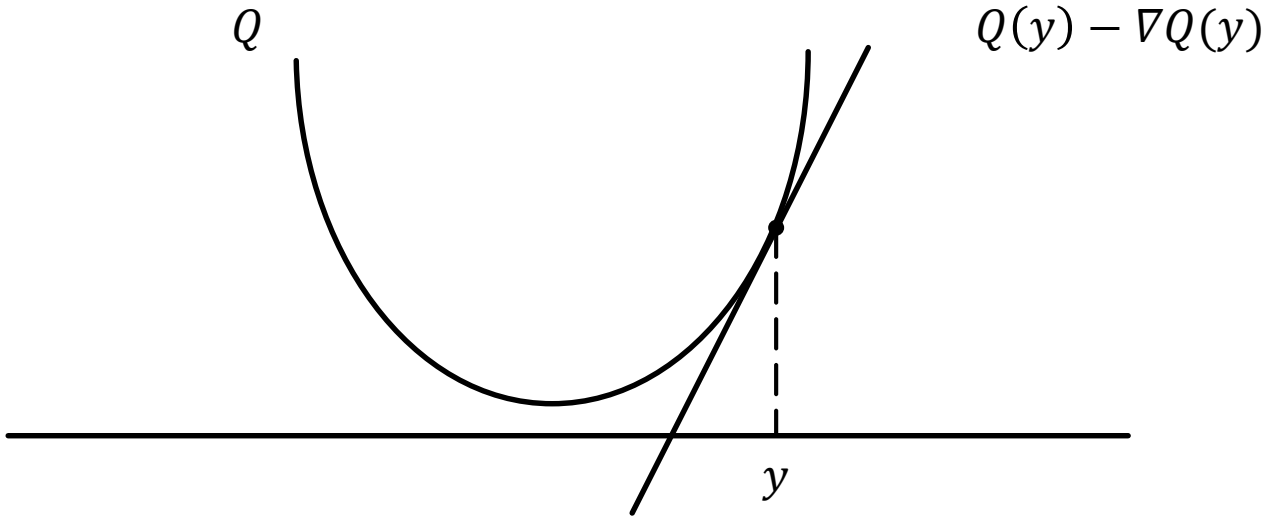
$$\mathrm{Q(w)} = \sum_{\mathrm{i=1}}^{\mathrm{n}} \mathrm{Q_i(w)} = \sum_{\mathrm{i=1}}^{\mathrm{n}} (\mathrm{w_1 + w_2 x_i - y_i})^2$$

  - $\mathrm{Q}(w)$ is just a sum of squared errors (SSE) for this w
  - What is n?

- 2. We need to minimize $\mathrm{Q}(w)$!

Recall: Minimizing a Function

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



The Gradient Descent Method

- We iterate over values of w until $Q(w)$ does not improve
- At each step, we change w opposite to the direction of "fastest growth" of $Q(w)$
- We get the direction of "fastest growth" as a gradient $\nabla Q(w)$ at a current value of w
  - Gradient of $Q(w)$ in respect to w, all other vars in $Q(w)$ are constant!

Procedure GD(Q(w)) # for minizing 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): $w := w - \alpha \nabla Q(w)$

α is a parameter ("meta-parameter") called **learning rate**

GD is Like Going Down a Hill

- Our current position $\approx w$, i.e. current values of all parameters to be optimized (e.g. $w = [w_1, w_2]^T$)
- "Territory" = surface created by the error function Q in a space with (#parameters)+1 dimensions

Algorithm:

- Repeat until stopping criterion
  - We compute the „tangent vector" $\nabla Q(w)$ at current w
    - i.e. $\nabla Q(w)$ is a vector pointing in the direction of the steepest rise from the current position w
  - Then we move (= change w) by some small step $\alpha \nabla Q(w)$ in the opposite direction …
  - .. and we arrive at a new position $\mathrm{w}' = \mathrm{w} - \alpha \nabla Q(w)$

GD Example – Computing Gradients

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):

$$w := w - \alpha \nabla Q(w)$$

- We have $Q(w) = \sum_{i=1}^{n} Q_i(w) = \sum_{i=1}^{n} (w_1 + w_2 x_i - y_i)^2$
- Since $\nabla(P + Q) = \nabla P + \nabla Q$, we have:

$$\nabla Q(w) = \sum_{i=1}^{n} \nabla Q_i(w)$$

- $\nabla Q_i(w) = \begin{bmatrix} 1^{st} \text{ derivative of } Q_i(w) \text{ by } w_1 \\ 1^{st} \text{ derivative of } Q_i(w) \text{ by } w_2 \end{bmatrix}$
- $\nabla Q_i(w) = \begin{bmatrix} \frac{dQ_i(w)}{dw_1} \\ \frac{dQ_i(w)}{dw_2} \end{bmatrix} = \begin{bmatrix} 2(w_1 + w_2 x_i - y_i) \\ 2(w_1 + w_2 x_i - y_i) x_i \end{bmatrix}$

One such 2-vector for each data point (i=1..n)

Stochastic Gradient Descent /1

- Assume that the objective function $\mathrm{Q}(\mathrm{w})$ is a sum $\mathrm{Q}(\mathrm{w}) = \sum_{\mathrm{i}=1}^{\mathrm{n}} \mathrm{Q}_{\mathrm{i}}(\mathrm{w})$
  - Typically $Q_i(w)$ comes from i-th training sample
- By linearity of the gradient $\nabla(P + Q) = \nabla P + \nabla Q$
- … we have $\nabla Q(w) = \sum_{i=1}^{n} \nabla Q_i(w)$

- **Stochastic Gradient Descent**
  - Instead of computing all $\nabla Q_1(w)$,…, $\nabla Q_n(w)$ and then making single step $\mathrm{w} := \mathrm{w} - \alpha \nabla Q(w)$,…
  - … 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 $\mathrm{Q}(\mathrm{w}) = \sum_{i=1}^{n} Q_i(w)$
  - 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
    - $\mathrm{w} := \mathrm{w} - \beta \nabla Q_i(w)$

Can you replace the double-loop with something else?

Difference GD vs. SGD

- Our exact gradient v (= vector of a steepest ascend) is a sum of 1000s of (imprecise) vectors $v^1,\ldots,v^n$
- Or: an exact "3D map of a hill" is a sum of 1000s of 3D maps, each possibly a bit blurry/imprecise

Then, in each (inner) iteration step:

GD: You compute exact gradient v …

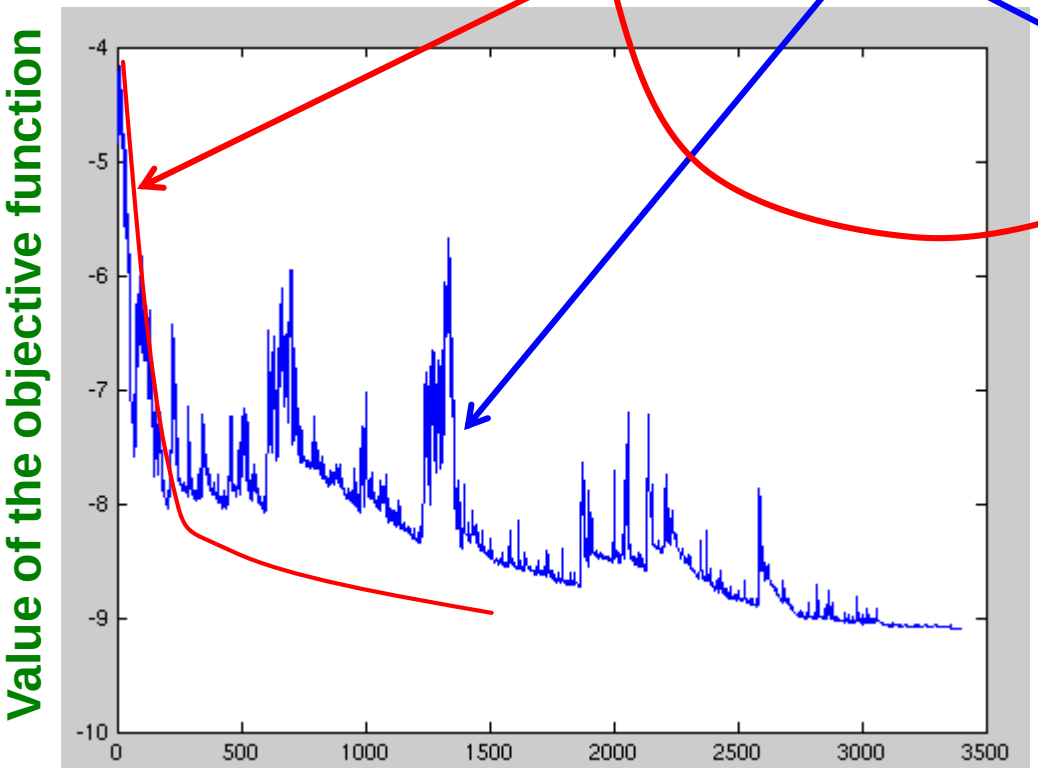
SGD: You compute one of the imprecise gradients v^i…

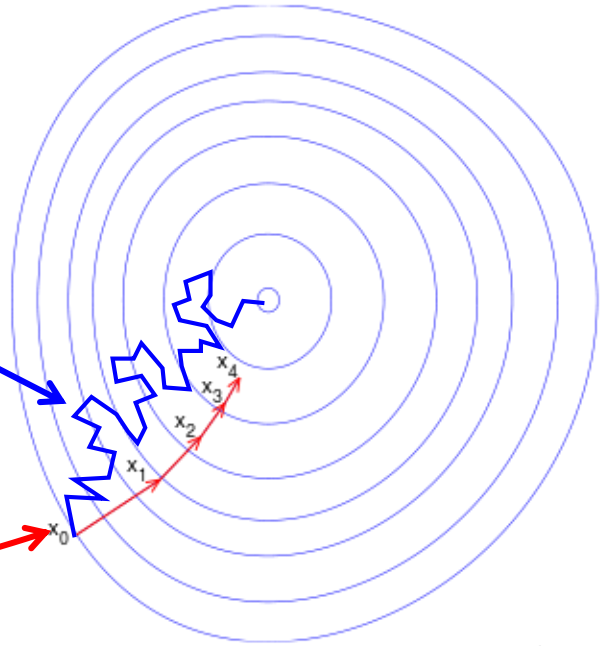
… and change your position according to this gradient

GD or SGD?

SGD vs. GD

- **Convergence of GD vs. SGD**



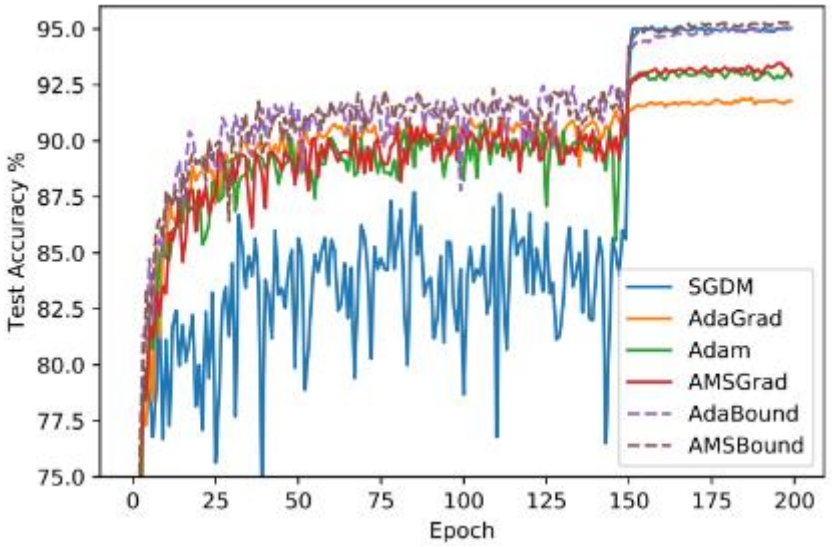


GD improves the value of the objective function at every step.
SGD improves the value in a "noisy" way.
GD takes fewer steps to converge but each step takes much longer to compute.
In practice, SGD is much faster!

SGD is a Generic Tool

- SGD is a generic and widely used method
- It can optimize (essentially) any function for which gradients can be computed
- Especially, it can train neural networks in ML
- Some improved versions are used today:

- Adam: uses adaptive learning rate + moving average of gradient
- Others (link):
  - Momentum, Nesterov accelerated gradient, Adagrad, Adadelta, RMSprop, AdaMax, Nadam, AMSGrad, …



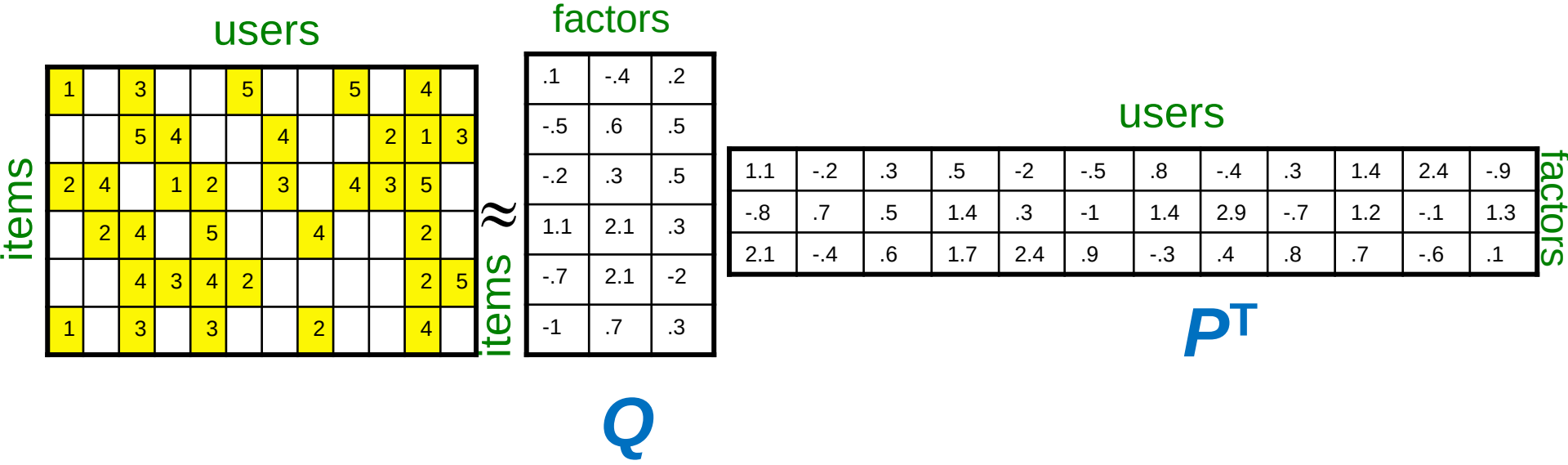
From https://openreview.net/pdf?id=Bkg3g2R9FX

Stochastic Gradient Descent for Latent Factors

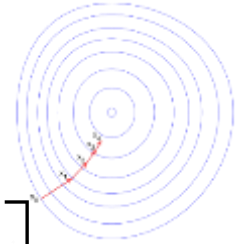
Recall: Latent Factor Models

- Our goal is to find matrices **P** and **Q** which minimize SSE + regularization term:

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



Gradient Descent for Latent Factors



- We want to find matrices P and Q with:

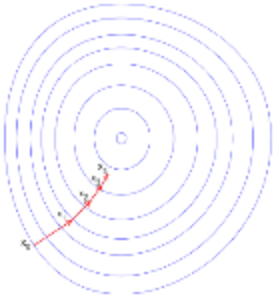
$$\min_{P,Q} \sum_{(x,i)\in R} (r_{xi} - q_i p_x)^2 + \left[\lambda_1 \sum_x \|p_x\|^2 + \lambda_2 \sum_i \|q_i\|^2 \right]$$

- **Gradient decent:**
  - Initialize P and Q (using **SVD** with missing ratings = 0)
  - Do gradient descent (iteration step):
    - $Q \leftarrow Q - \eta \cdot \nabla Q$
    - $P \quad P - \eta \cdot \nabla P$, where ∇P is …(later)
  - ∇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 q_{if} is entry f of row q_i of matrix Q
- Observation: Computing gradients is slow!

Singular Value Decomposition

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

Stochastic Gradient Descent

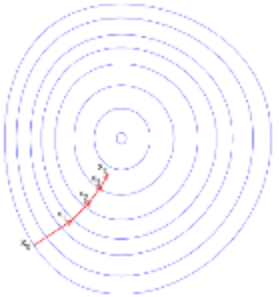


$$\min_{P,Q} \sum_{(x,i)\in R} (r_{xi} - q_i p_x)^2 + \left[\lambda_1 \sum_x \|p_x\|^2 + \lambda_2 \sum_i \|q_i\|^2 \right]$$

Gradient Descent (GD) vs. Stochastic GD

- Instead of evaluating gradient over all ratings evaluate it for an individual rating and make a step
- GD: $Q \leftarrow Q - \eta[\sum_{r_{xi}} \nabla Q_{xi}(r_{xi})]$
- SGD: $Q \leftarrow Q - \mu \nabla Q_{xi}(r_{xi})$ — for one r_{xi} at a time
  - $\varepsilon_{xi} = 2(r_{xi} - q_i \cdot p_x)$ — (derivative of the "error")
  - $q_i \leftarrow q_i + \mu_1(\varepsilon_{xi}\, p_x - \lambda_2\, q_i)$ — (update equation)
- SGD: We need more steps but each step is computed much faster

SGD for LF – Full Algorithm



Stochastic gradient decent:

- Initialize P and Q (using SVD, pretend missing ratings are 0)
- Iterate over the ratings (multiple times if necessary) and update matrices P and Q:

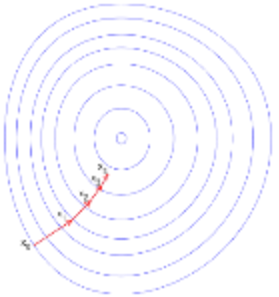
  Step: for each r_{xi} …

  μ_1, μ_2 are learning rates

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

- Two loops:
  - Repeat until convergence:
    - For each r_{xi}
      - Compute gradient, do a "step"

Key Numbers



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

- Q1. How many parameters to optimize are there, and where are they?
- A1. The parameters are entries of P and Q, and so we have (#users)*(#factors)+(#items)*(#factors)
- Q2. How many terms (summands) are there in the 1st term (SSE)? (Why do I ask this?)
- A2. The first term (SSE) has as many entries as there are non-empty ratings in the utility matrix
  - This determines the length of „inner loop“

Link: https://pingo.coactum.de/147633

Pingo Quiz

- Mark true statements (multiple-choice possible)
  a. If you increase the number of factors in the LF model, then it is advisable to choose larger regularization parameters λ_1 and λ_2.
  b. In SGD (inner) iteration step, we change our position (=parameter choice) only in few selected dimensions; in a GD step, we change position in (possibly) all dims.
  c. In the SGD for LF model, the number of steps of the inner loop (of SGD) is always (#users)*(#items).
  d. If your regularization parameters are very large, GD or SGD might give you matrices P and Q with only 0's.

References

- Yehuda Koren, Robert Bell and Chris Volinsky: Matrix Factorization Techniques for Recommender Systems, IEEE Computer, August 2009, https://ieeexplore.ieee.org/document/5197422
  - 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

Thank you.

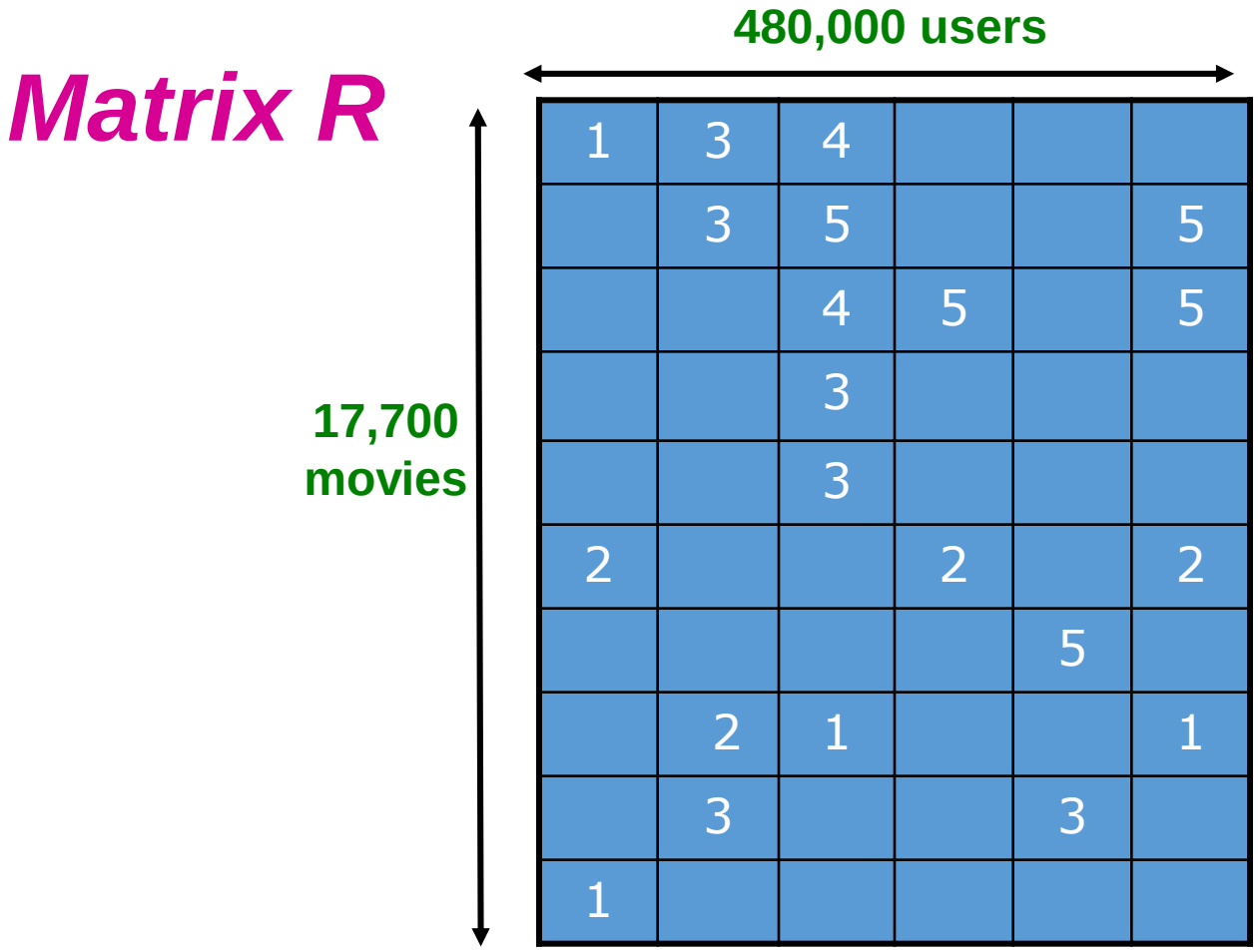
Questions?

The Netflix Prize: Introduction

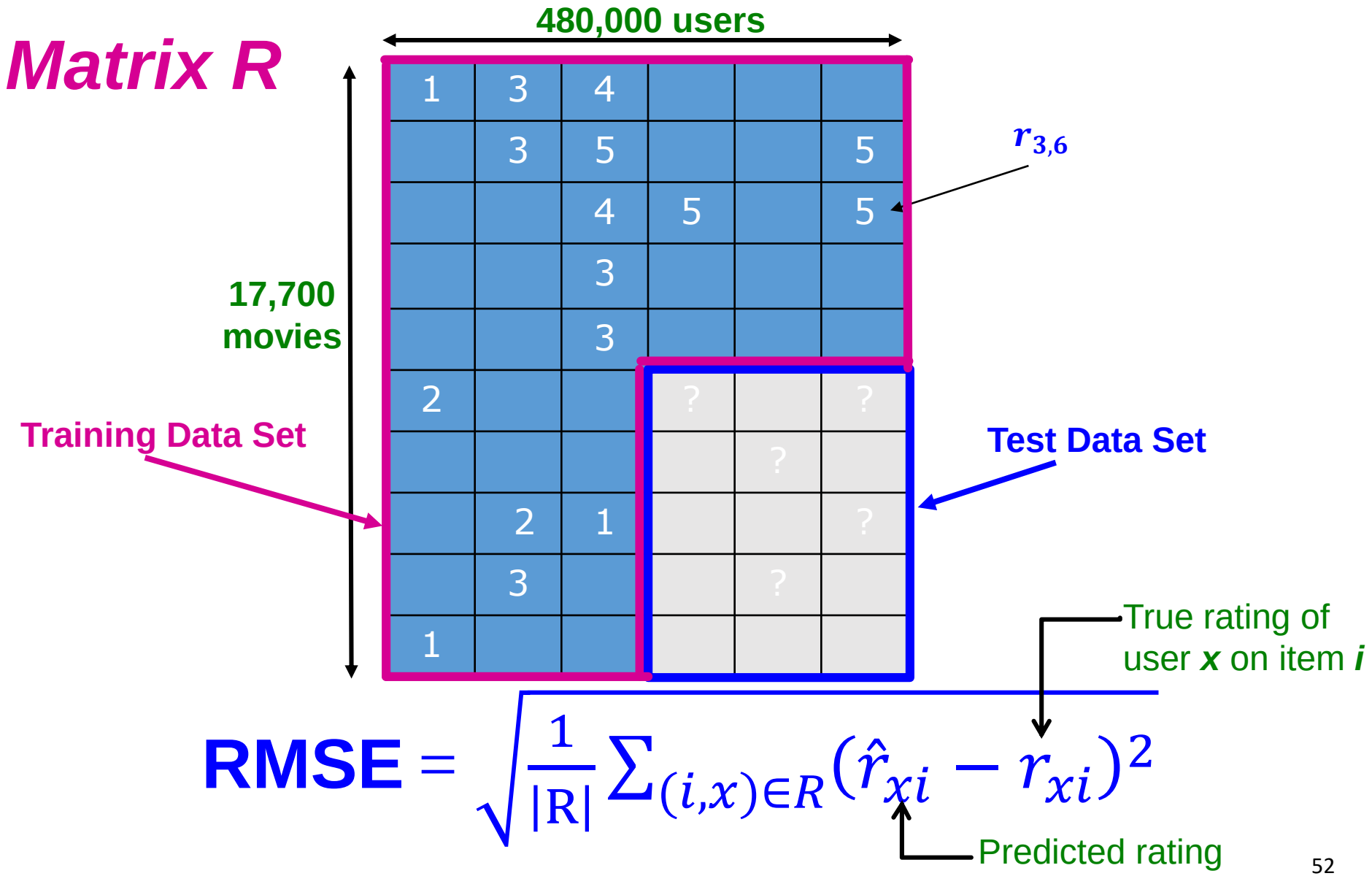
The Netflix Prize

- **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

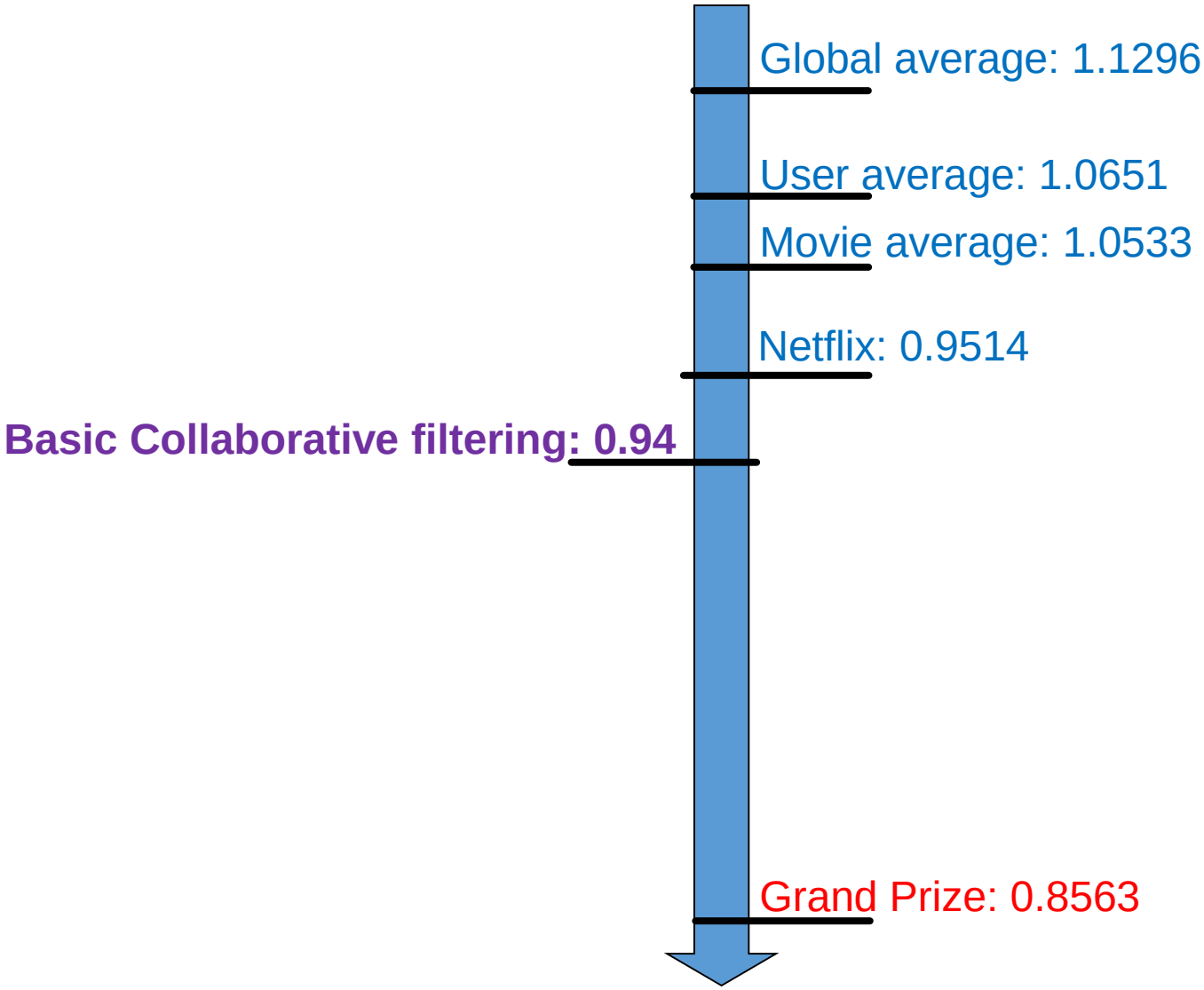
The Netflix Utility Matrix *R*



Utility Matrix *R*: Evaluation



Performance of Various Methods



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

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 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

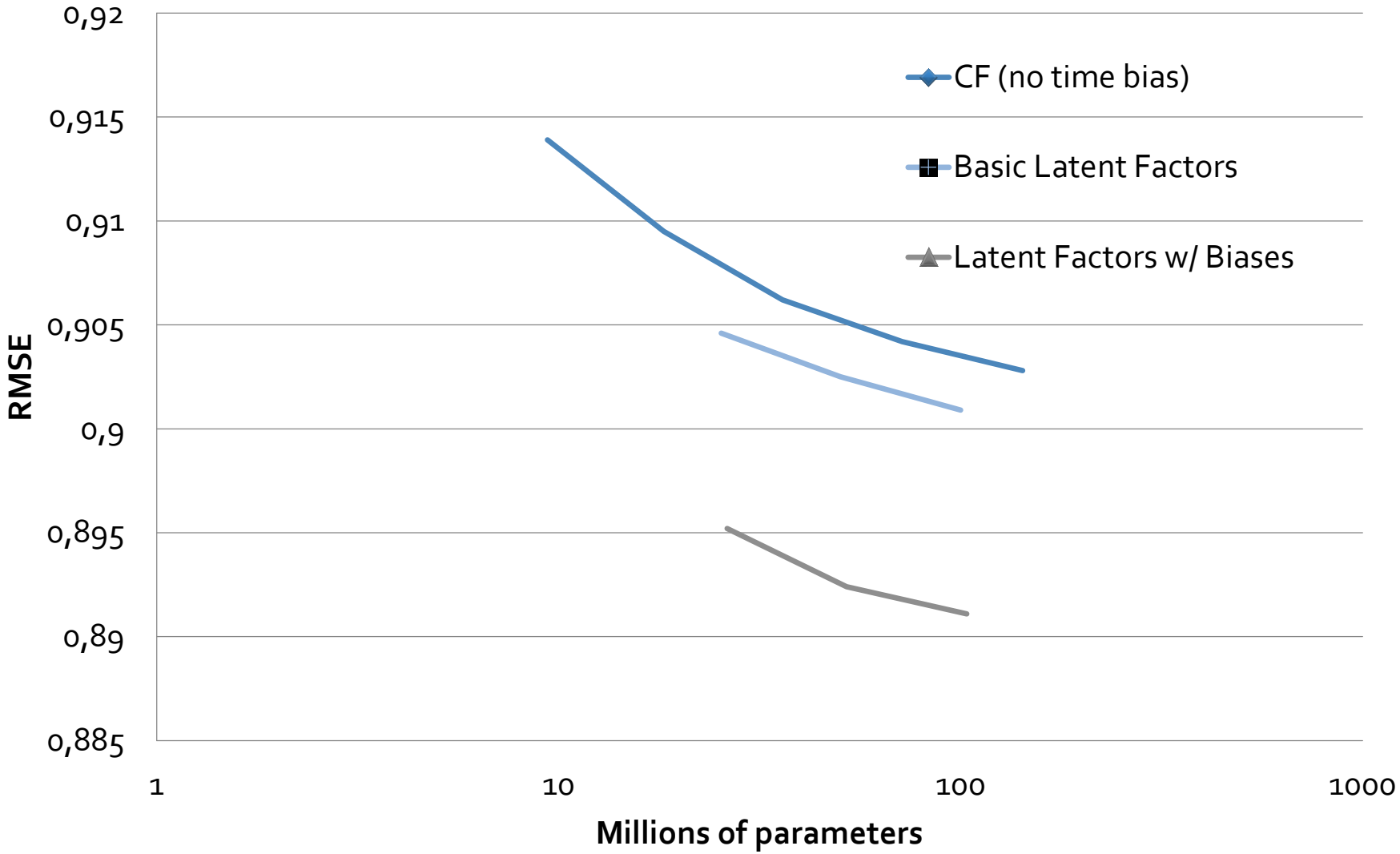
$$+ \left(\lambda_1 \sum_i \|q_i\|^2 + \lambda_2 \sum_x \|p_x\|^2 + \lambda_3 \sum_x \|b_x\|^2 + \lambda_4 \sum_i \|b_i\|^2 \right)$$

regularization

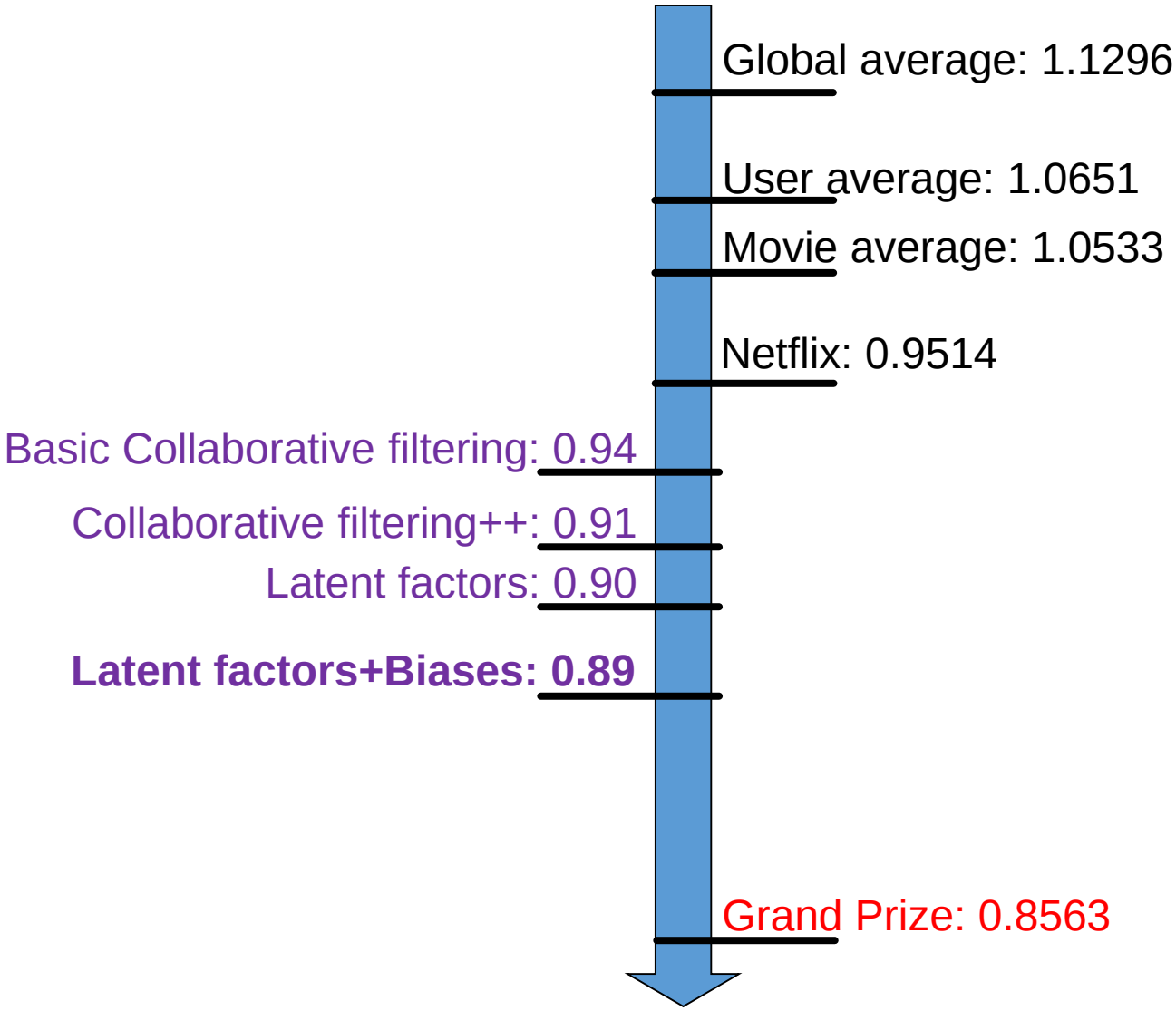
λ is selected via grid-search on a validation set

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

Performance of Various Methods

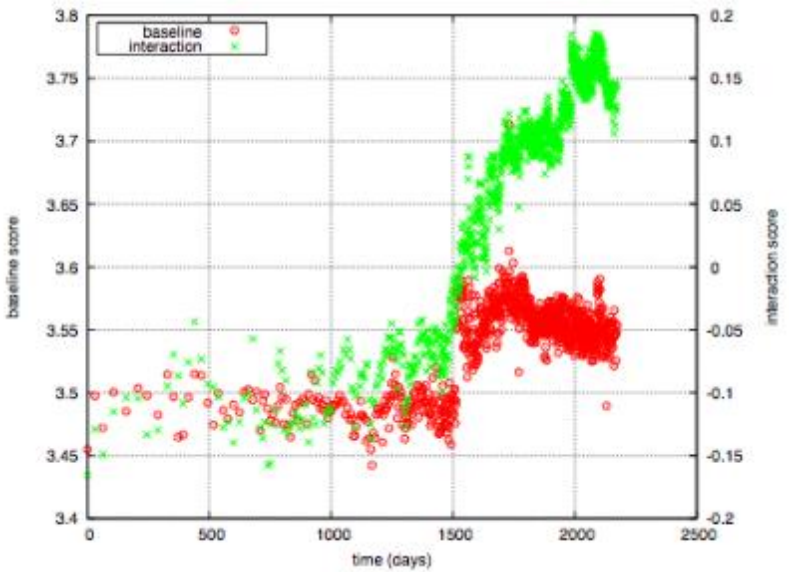


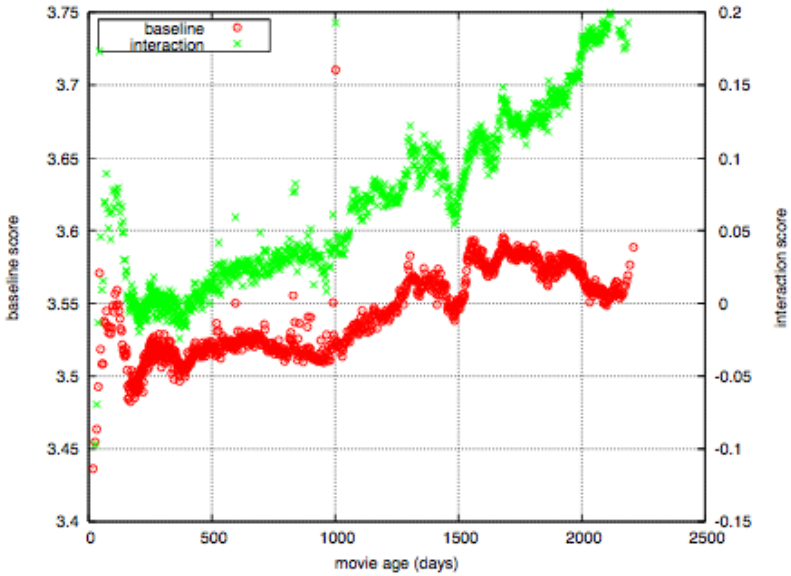
Performance of Various Methods



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

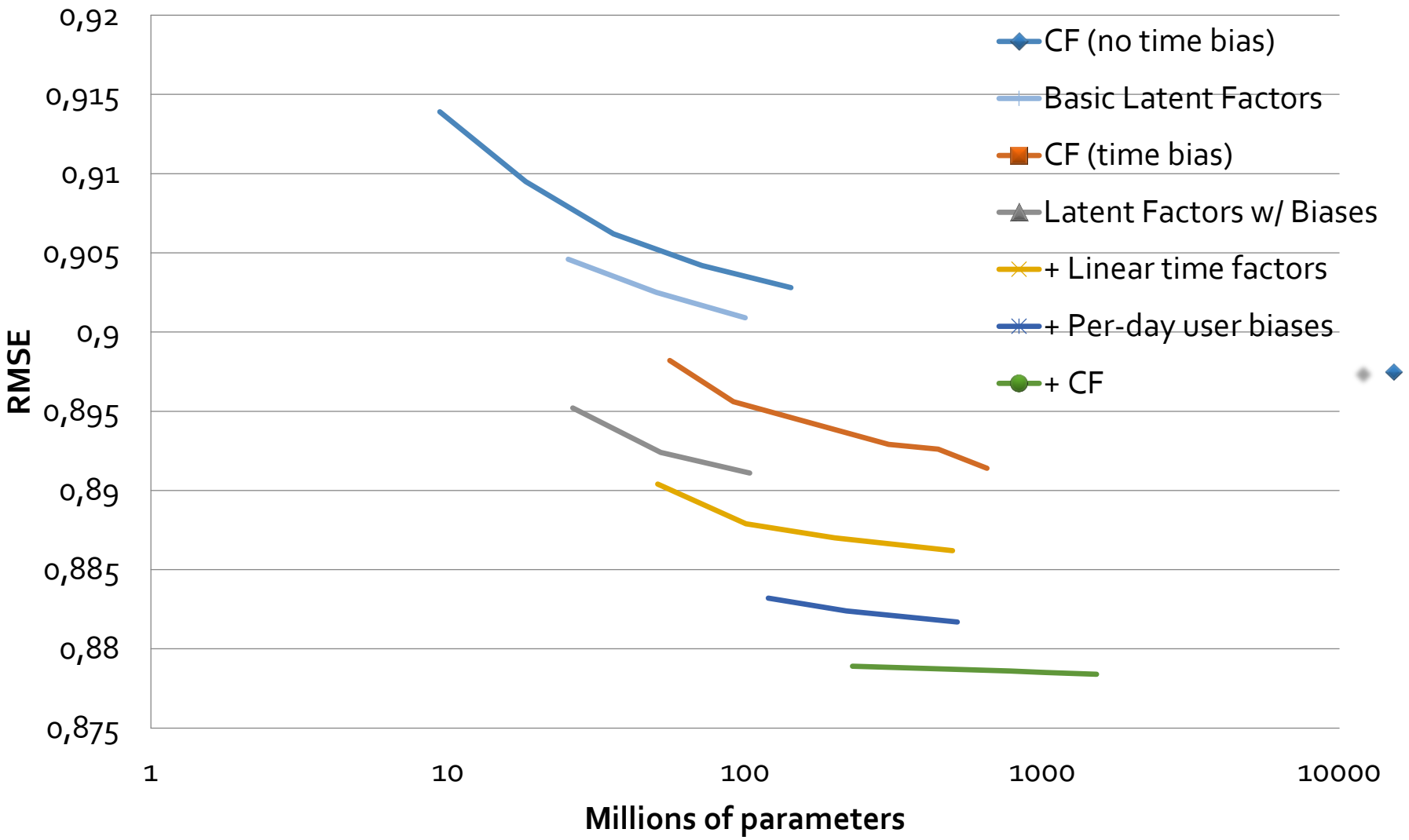
    $$b_i(t) = b_i + b_{i,\mathrm{Bin}(t)}$$

- **Add temporal dependence to factors**

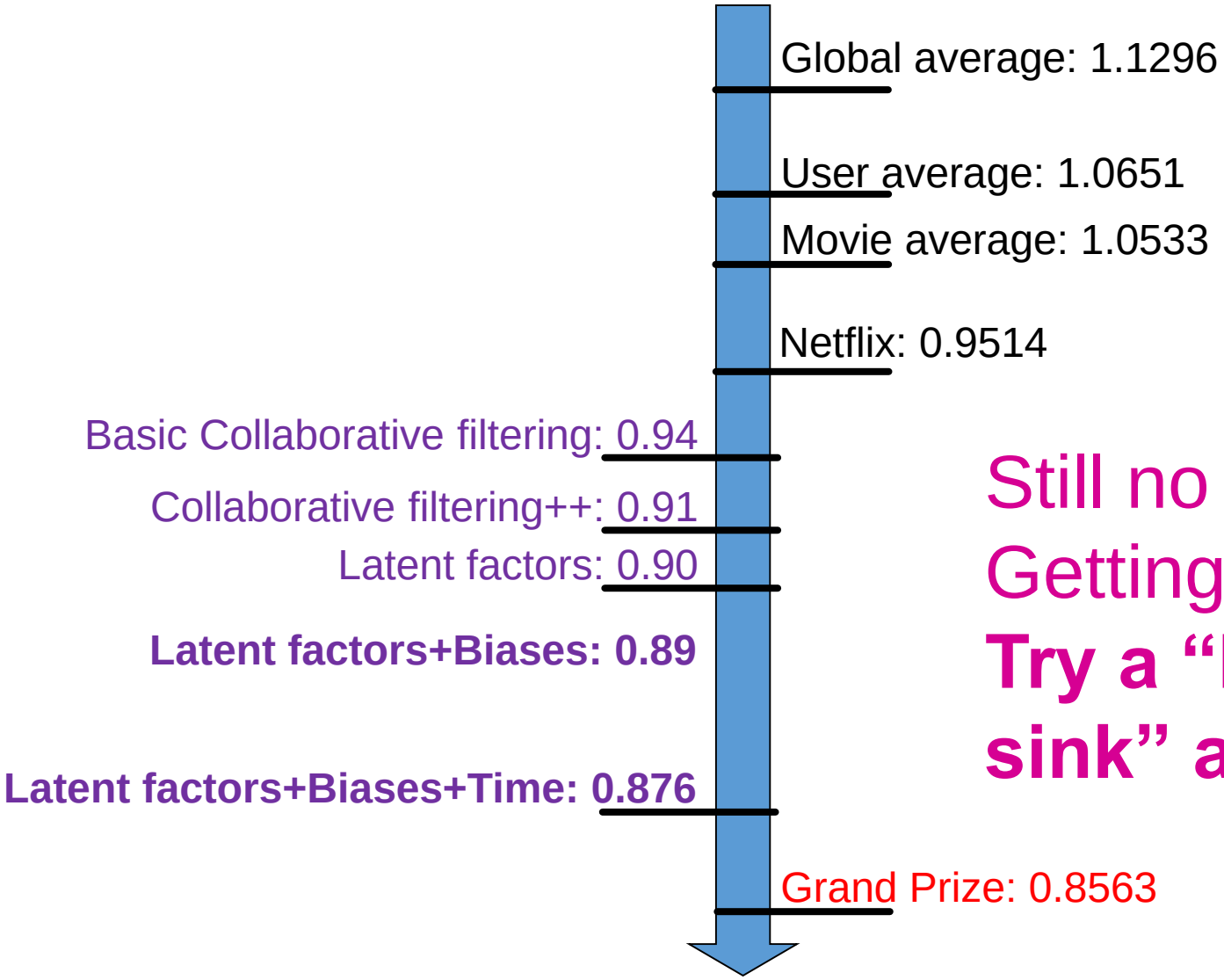
  - $p_x(t)$… user preference vector on day t

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

Adding Temporal Effects



Performance of Various Methods



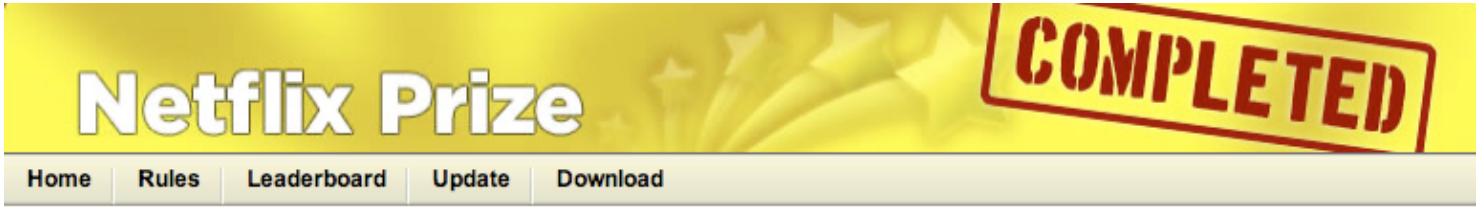
Still no prize! ☹
Getting desperate.
Try a "kitchen sink" approach!

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….**



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

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