

**IERG4300/ IEMS5709**  
**Web-Scale Information Analytics**

**Recommender Systems:**

Prof. Wing C. Lau  
Department of Information Engineering  
[wclau@ie.cuhk.edu.hk](mailto:wclau@ie.cuhk.edu.hk)

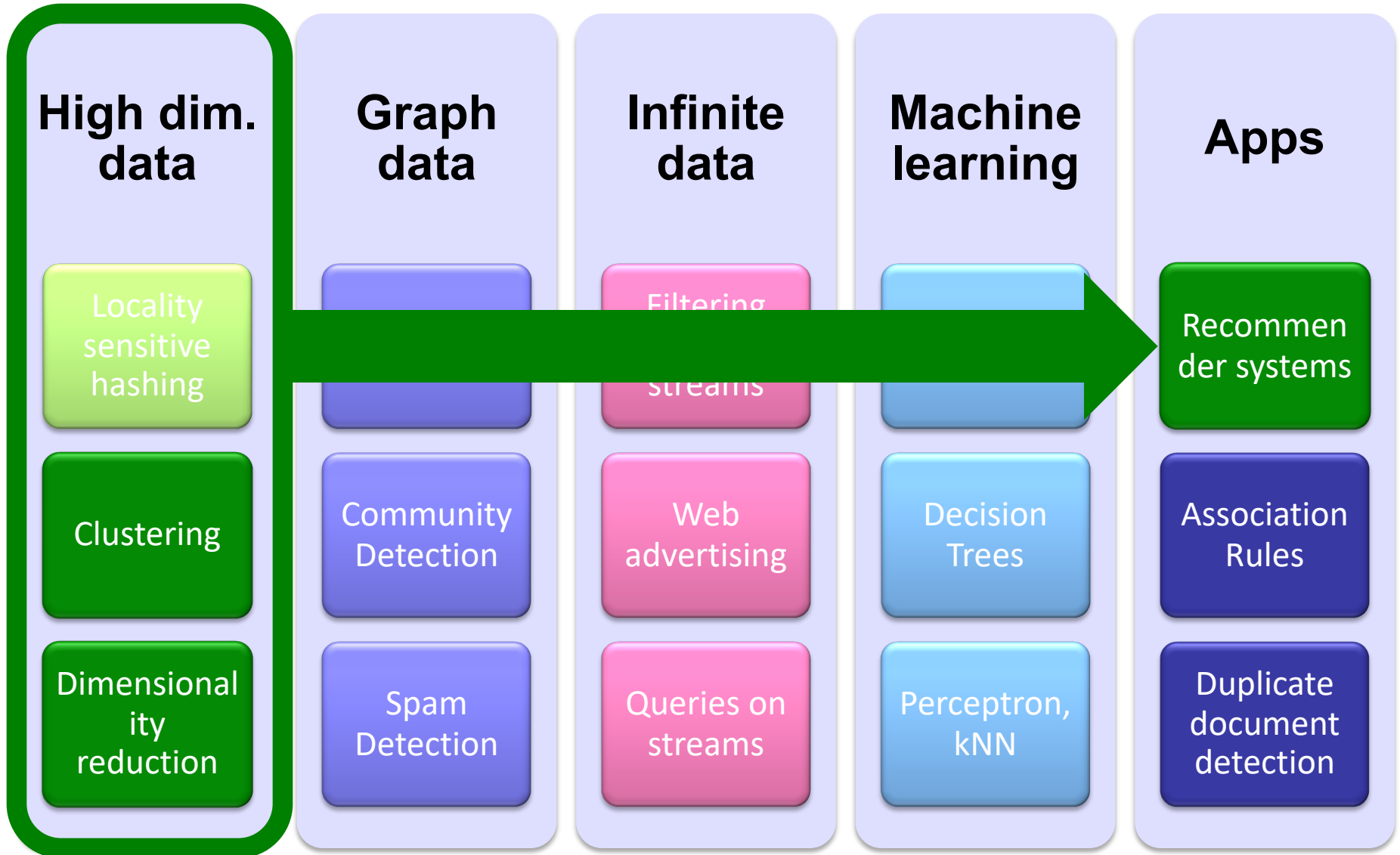
# Acknowledgements

- The slides used in this chapter are adapted from:
  - CS246 Mining Massive Data-sets, by Jure Leskovec, Stanford University.
  - Yehuda Koren, Robert Bell & Chris Volinsky, “Lessons from the Netflix Prize,” AT&T Research.
  - Some slides and plots borrowed from Yehuda Koren, Robert Bell and Padhraic Smyth

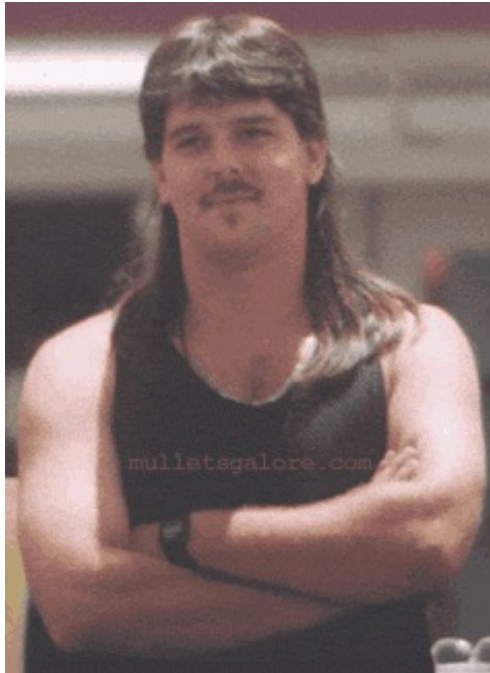
with the author’s permission. All copyrights belong to the original author of the material.

# Content-based Systems & Collaborative Filtering

# High Dimensional Data



# Example: Recommender Systems



## ○ Customer X

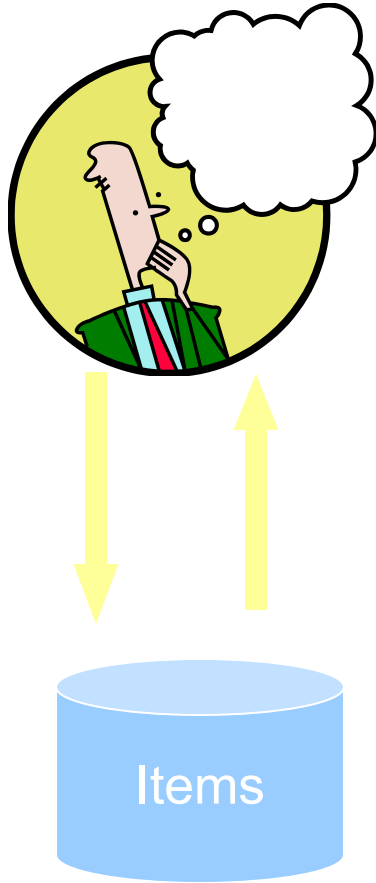
- Buys Metallica CD
- Buys Megadeth CD



## ○ Customer Y

- Does search on Metallica
- Recommender system suggests Megadeth from data collected about customer X

# Recommendations



## Examples:

amazon.com.



**m o v i e l e n s**  
helping you find the *right* movies

last.fm™  
the social music revolution

Google™  
News

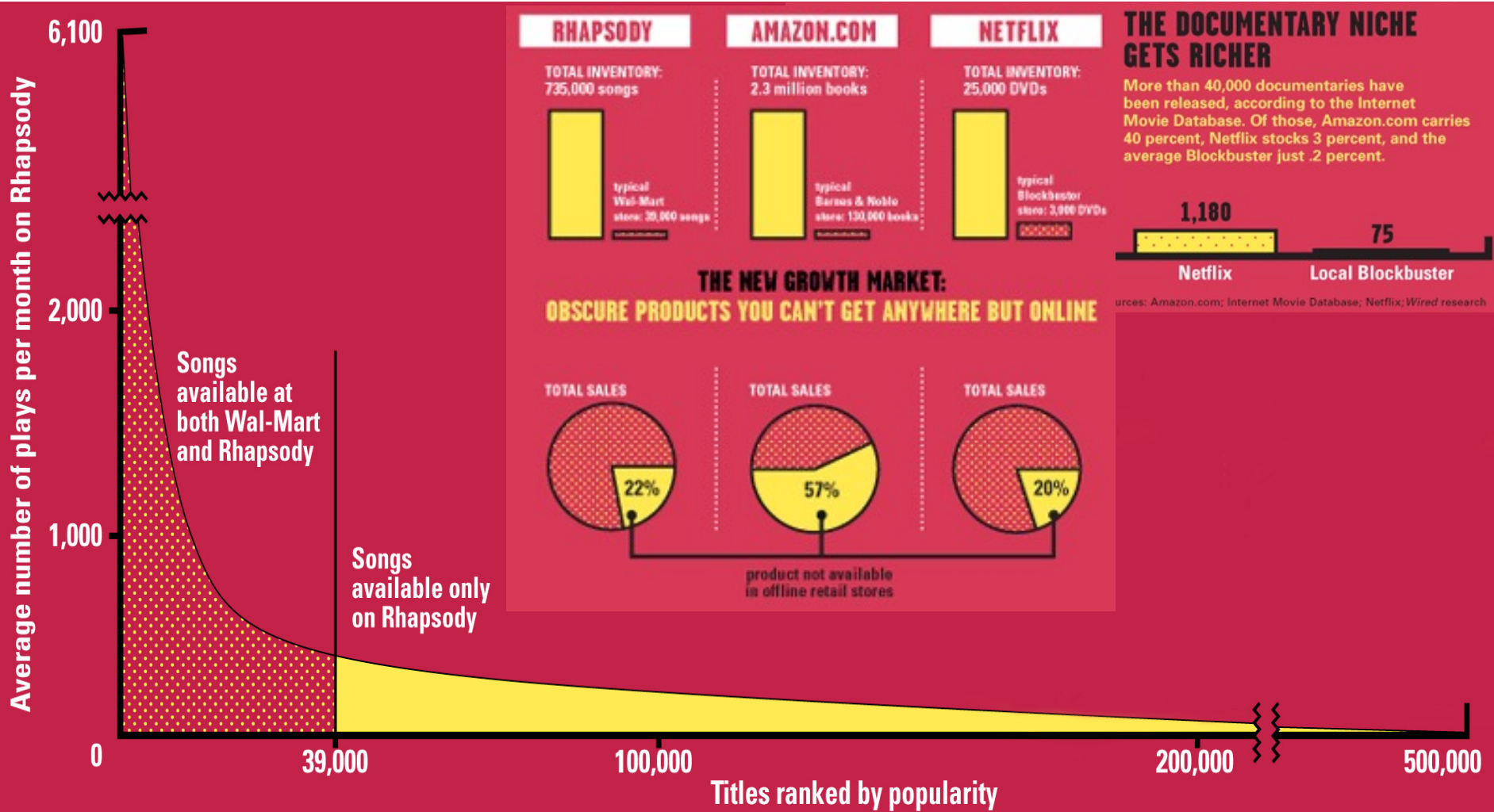
You Tube

XBOX  
LIVE

# From Scarcity to Abundance

- **Shelf space is a scarce commodity for traditional retailers**
  - Also: TV networks, movie theaters,...
- **Web enables near-zero-cost dissemination of information about products**
  - From scarcity to abundance
- **More choice necessitates better filters**
  - Recommendation engines
  - How **Into Thin Air** made **Touching the Void** a bestseller:  
<http://www.wired.com/wired/archive/12.10/tail.html>

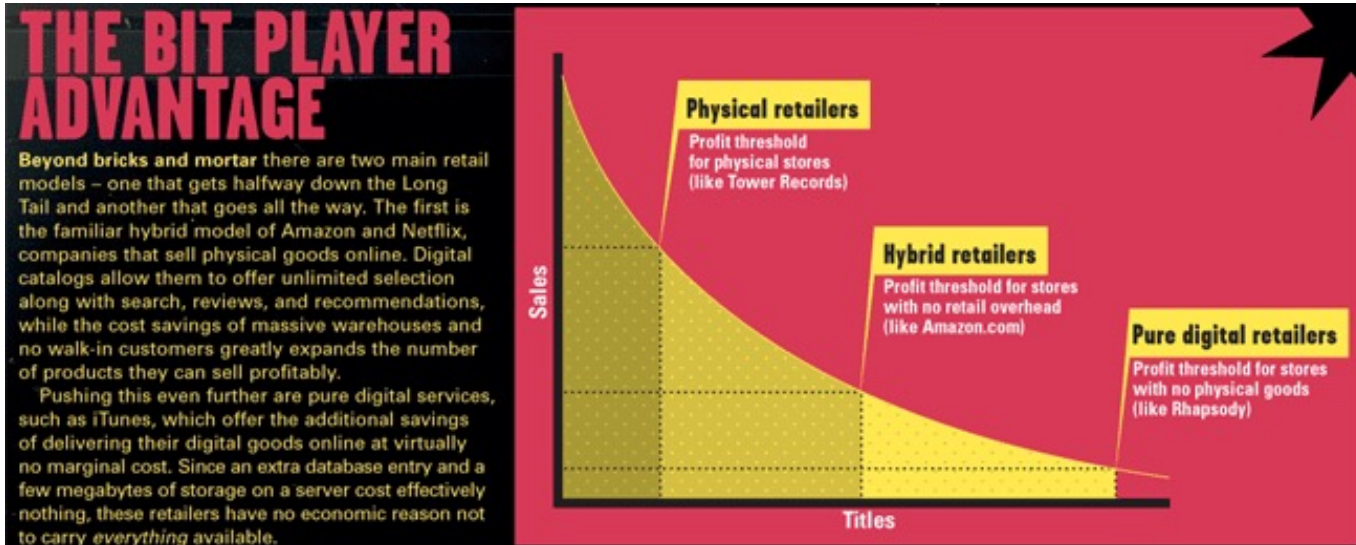
# The Long Tail



Sources: Erik Brynjolfsson and Jeffrey Hu, MIT, and Michael Smith, Carnegie Mellon; Barnes & Noble; Netflix; RealNetworks



# Physical vs. Online



Read <http://www.wired.com/wired/archive/12.10/tail.html> to learn more!

# Types of Recommendations

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

# Formal Model

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

# Utility Matrix

	Avatar	LOTR	Matrix	Pirates
Alice	1		0.2	
Bob		0.5		0.3
Carol	0.2		1	
David				0.4

# Key Problems

- **(1) Gathering “known” ratings for matrix**
  - How to collect the data in the utility matrix
- **(2) Extrapolate unknown ratings from the known ones**
  - Mainly interested in high unknown ratings
    - We are not interested in knowing what you don't like but what you like
- **(3) Evaluating extrapolation methods**
  - How to measure success/performance of recommendation methods

# (1) Gathering Ratings

## ○ **Explicit**

- Ask people to rate items
- Doesn't work well in practice – people can't be bothered

## ○ **Implicit**

- Learn ratings from user actions
  - E.g., purchase implies high rating
- What about low ratings?

## (2) Extrapolating Utilities

- **Key problem:** matrix  $U$  is **sparse**
  - Most people have not rated most items
  - **Cold start:**
    - New items have no ratings
    - New users have no history
- **Three approaches to recommender systems:**
  - 1) Content-based
  - 2) Collaborative Filtering
    - Memory-based
      - User-based Collaborative Filtering
      - Item-based Collaborative Filtering
    - Latent factor based

# Content-based Recommender Systems



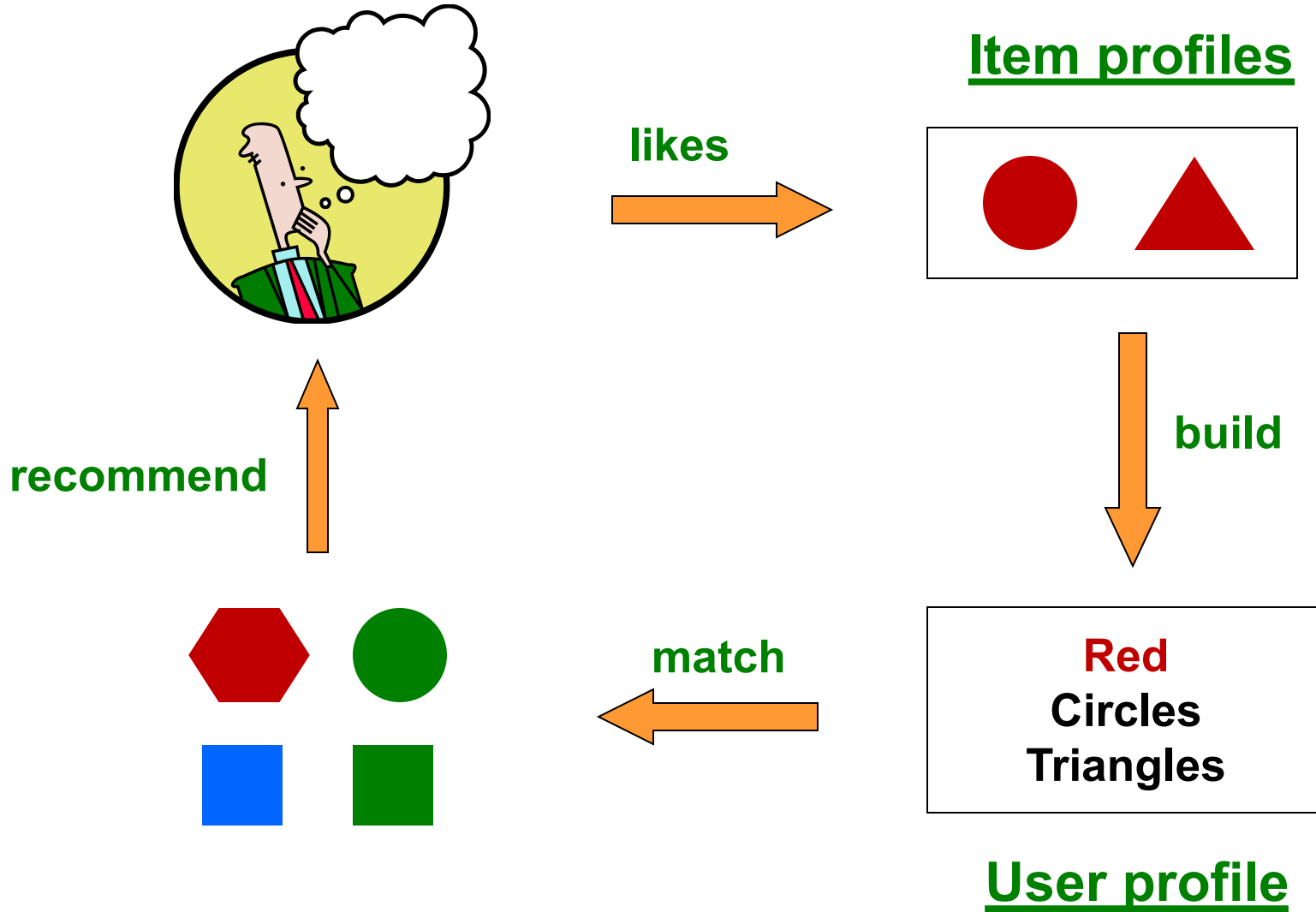
# Content-based Recommendations

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

## *Example:*

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

# Plan of Action



# Item Profiles

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

# Sidenote: TF-IDF

$f_{ij}$  = frequency of term (feature)  $i$  in doc (item)  $j$

$$TF_{ij} = \frac{f_{ij}}{\max_k f_{kj}}$$

**Note:** we normalize TF to discount for “longer” documents

$n_i$  = number of docs that mention term  $i$

$N$  = total number of docs

$$IDF_i = \log \frac{N}{n_i}$$

**TF-IDF score:**  $w_{ij} = TF_{ij} \times IDF_i$

**Doc profile** = set of words with highest **TF-IDF** scores, together with their scores

# User Profiles and Prediction

## ○ User profile possibilities:

- Weighted average of rated item profiles
- **Variation:** weight by difference from average rating for item
- ...

## ○ Prediction heuristic:

- Given user profile  $\mathbf{x}$  and item profile  $\mathbf{i}$ , estimate

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

# Pros: Content-based Approach

- **+: No need for data on other users**
  - No cold-start or sparsity problems
- **+: Able to recommend to users with unique tastes**
- **+: Able to recommend new & unpopular items**
  - No first-rater problem
- **+: Able to provide explanations**
  - Can provide explanations of recommended items by listing content-features that caused an item to be recommended

# Cons: Content-based Approach

- **–: Finding the appropriate features is hard**
  - E.g., images, movies, music
- **–: Overspecialization**
  - Never recommends items outside user's content profile
  - People might have multiple interests
  - Unable to exploit quality judgments of other users
- **–: Recommendations for new users**
  - **How to build a user profile?**

# Collaborative Filtering



# Collaborative filtering

- Recommend items based on past transactions of users
- Analyze relations between users and/or items
- Specific data characteristics are irrelevant
  - **Domain-free: user/item attributes are not necessary**
  - Can identify elusive aspects

amazon.com

Customers who bought items in your Recent History also bought:



I Own It  Not interested  
x|☆☆☆☆☆ Rate it  
[Add to Cart](#) [Add to Wish List](#)



I Own It  Not interested  
x|☆☆☆☆☆ Rate it  
[Add to Cart](#) [Add to Wish List](#)



I Own It  Not interested  
x|☆☆☆☆☆ Rate it  
[Add to Cart](#) [Add to Wish List](#)

# Collaborative Filtering (CF)

Memory-based  
(e.g.,  $k$ -nearest neighbors)

Model-based  
(e.g., matrix factorization)

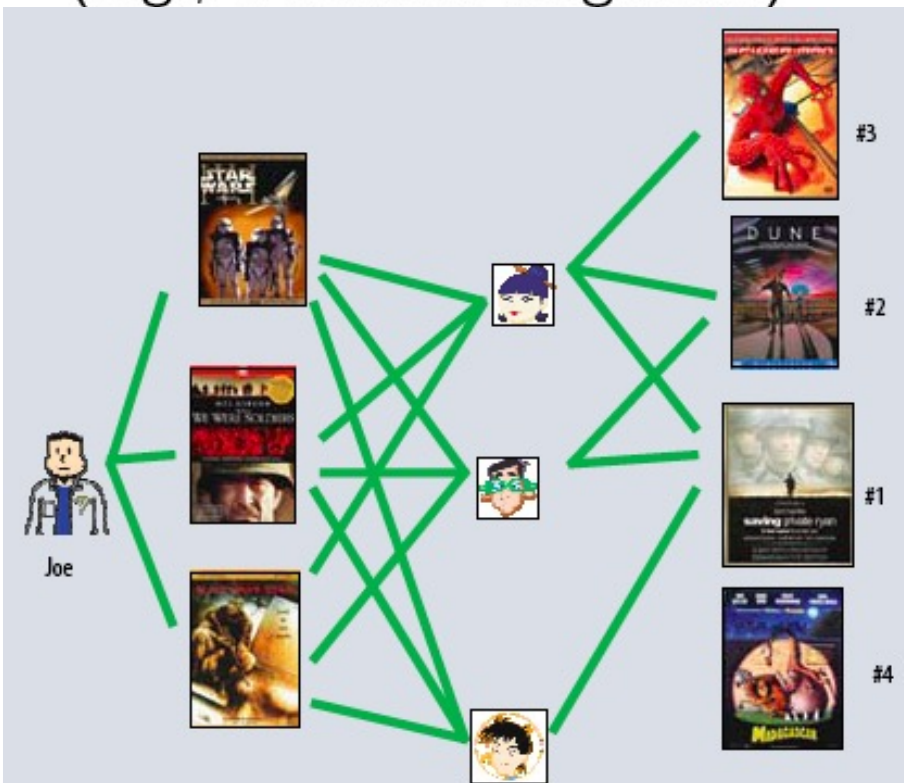


Figure 1. The user-oriented neighborhood method. Joe likes the three movies on the left. To make a prediction for him, the system finds similar users who also liked those movies, and then determines which other movies they liked. In this case, all three liked *Saving Private Ryan*, so that is the first recommendation. Two of them liked *Dune*, so that is next, and so on.

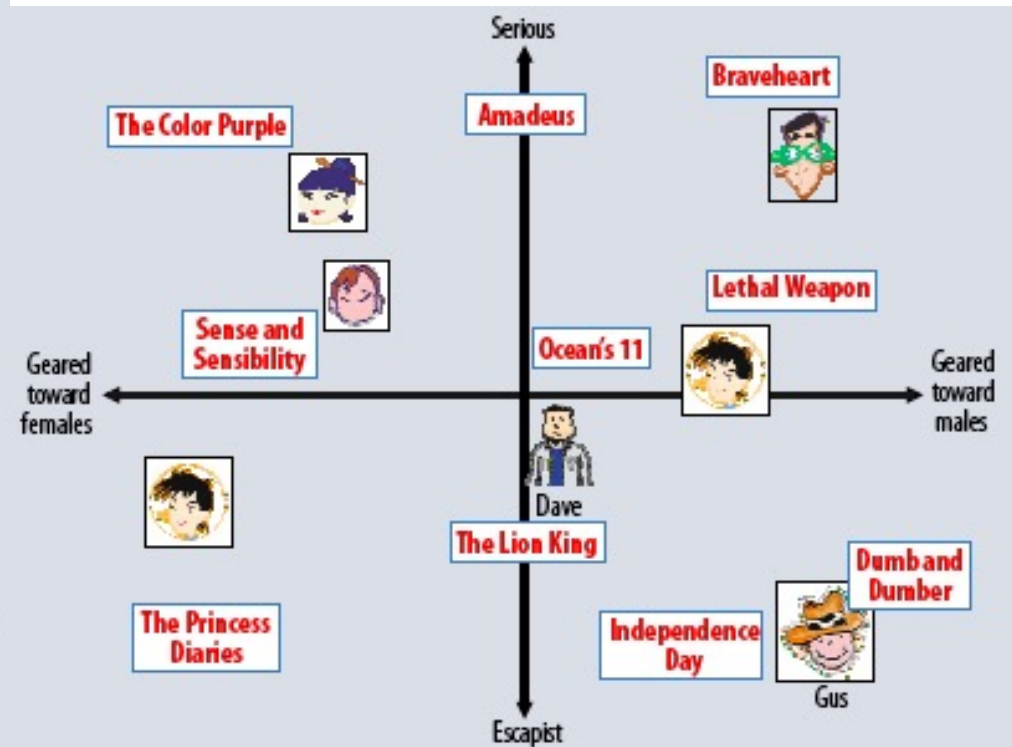
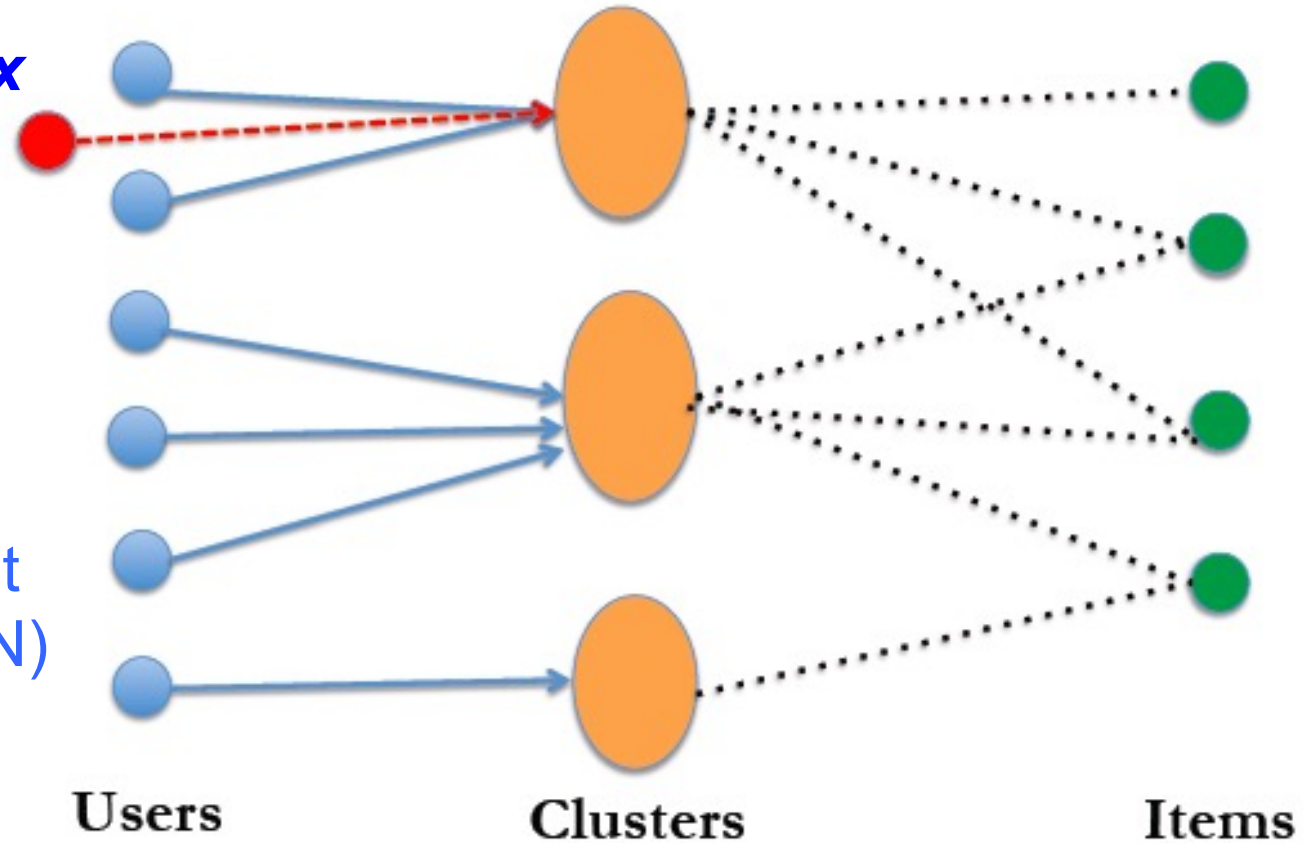


Figure 2. A simplified illustration of the latent factor approach, which characterizes both users and movies using two axes—male versus female and serious versus escapist.

<http://research.yahoo.com/pub/2859>

# Example of Memory-based Collaborative Filtering: User-User Collaborative Filtering

1. Consider user  $x$
2. Find set  $N$  of other users whose ratings are “similar” to  $x$ 's ratings, e.g. using K-nearest neighbors (KNN)
3. Recommend items to  $x$  based on the weighted ratings of items by users in  $N$



# Similar Users

$$\mathbf{r}_x = \begin{bmatrix} * & \_ & \_ & * & *** \\ * & \_ & ** & ** & \_ \end{bmatrix}$$

○ Let  $\mathbf{r}_x$  be the vector of user  $\mathbf{x}$ 's ratings

○ **Jaccard similarity measure**

- **Problem:** Ignores the value of the rating

*$\mathbf{r}_x, \mathbf{r}_y$  as sets:*  
 $\mathbf{r}_x = \{1, 4, 5\}$   
 $\mathbf{r}_y = \{1, 3, 4\}$

○ **Cosine similarity measure**

- $\text{sim}(\mathbf{x}, \mathbf{y}) = \cos(\mathbf{r}_x, \mathbf{r}_y) = \frac{\mathbf{r}_x \cdot \mathbf{r}_y}{\|\mathbf{r}_x\| \cdot \|\mathbf{r}_y\|}$

*$\mathbf{r}_x, \mathbf{r}_y$  as points:*  
 $\mathbf{r}_x = \{1, 0, 0, 1, 3\}$   
 $\mathbf{r}_y = \{1, 0, 2, 2, 0\}$

- **Problem:** Treats missing ratings as “negative”

○ **Pearson correlation coefficient**

$$\text{simil}(x, y) = \frac{\sum_{i \in I_{xy}} (r_{x,i} - \bar{r}_x)(r_{y,i} - \bar{r}_y)}{\sqrt{\sum_{i \in I_{xy}} (r_{x,i} - \bar{r}_x)^2 \sum_{i \in I_{xy}} (r_{y,i} - \bar{r}_y)^2}}$$

*$\bar{r}_x, \bar{r}_y \dots$  avg. rating of  $\mathbf{x}, \mathbf{y}$*

where  $I_{xy}$  is the set of items rated by both user  $\mathbf{x}$  and user  $\mathbf{y}$ .

# Similarity Metric

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

- **Intuitively we want:**  $\text{sim}(A, B) > \text{sim}(A, C)$
- **Jaccard similarity:**  $1/5 < 2/4$
- **Cosine similarity:**  $0.386 > 0.322$ 
  - Considers missing ratings as “negative”
  - **Solution: subtract the (row) mean**

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

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

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

# Rating Predictions

- Let  $r_x$  be the vector of user  $x$ 's ratings
- Let  $N$  be the set of  $k$  users most similar to  $x$   
who have rated item  $i$

- Prediction for item  $i$  of user  $x$ :

- $r_{xi} = \frac{1}{k} \sum_{y \in N} r_{yi}$

- $r_{xi} = \frac{\sum_{y \in N} s_{xy} \cdot r_{yi}}{\sum_{y \in N} s_{xy}}$

Shorthand:

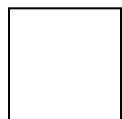
$$s_{xy} = \text{sim}(x, y)$$

- Other options?

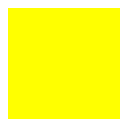
- Many other tricks possible...

# User-User CF ( $|N|=3$ )

	movies												
	1	2	3	4	5	6	7	8	9	10	11	12	
1	1		3			5			5		4		$S_{1,2}$
2			5	4	?		4			2	1	3	$S_{2,2}$
3	2	4		1	2		3		4	3	5		$S_{3,2}$
4		2	4		5			4			2		$S_{4,2}$
5			4	3	4	2					2	5	$S_{5,2}$
6	1		3		3			2			4		$S_{6,2}$



- unknown rating



- rating between 1 to 5



# Another type of Memory-based Collaborative Filtering: : Item-Item based Collaborative Filtering

- So far: **User-user collaborative filtering**
- **Another view: Item-item**
  - For item  $i$ , find other similar items
  - Estimate rating for item  $i$  based on the target user's ratings on items similar to item  $i$
  - Can use same similarity metrics and prediction functions as in user-user model

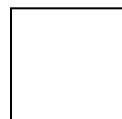
$$r_{xi} = \frac{\sum_{j \in N(i;x)} s_{ij} \cdot r_{xj}}{\sum_{j \in N(i;x)} s_{ij}}$$

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



# Item-Item CF ( $|N|=2$ )

	users											
	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	



- unknown rating



- rating between 1 to 5

# Item-Item CF ( $|N|=2$ )

**users**

	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	



- estimate rating of movie 1 by user 5

# Item-Item CF ( $|N|=2$ )

		users												
		1	2	3	4	5	6	7	8	9	10	11	12	
movies	1	1		3		?	5			5		4		1.00
	2			5	4			4			2	1	3	-0.18
	<u>3</u>	2	4		1	2		3		4	3	5		<u>0.41</u>
	4		2	4		5			4			2		-0.10
	5			4	3	4	2					2	5	-0.31
	<u>6</u>	1		3		3			2			4		<u>0.59</u>

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

Here we use Pearson correlation as similarity:  
 1) Subtract mean rating  $m_i$  from each movie  $i$   
 $m_1 = (1+3+5+5+4)/5 = 3.6$   
 row 1:  $[-2.6, 0, -0.6, 0, 0, 1.4, 0, 0, 1.4, 0, 0.4, 0]$   
 2) Compute cosine similarities between rows

# Item-Item CF ( $|N|=2$ )

		users												
		1	2	3	4	5	6	7	8	9	10	11	12	$\text{sim}(1,m)$
movies	1	1		3		?	5			5		4		1.00
	2			5	4			4			2	1	3	-0.18
	<u>3</u>	2	4		1	2		3		4	3	5		<u>0.41</u>
	4		2	4		5			4			2		-0.10
	5			4	3	4	2					2	5	-0.31
	<u>6</u>	1		3		3			2			4		<u>0.59</u>

Compute similarity weights:

$s_{13}=0.41$ ,  $s_{16}=0.59$

# Item-Item CF ( $|N|=2$ )

**users**

	1	2	3	4	5	6	7	8	9	10	11	12
1	1		3		2.6	5			5		4	
2			5	4			4			2	1	3
<u>3</u>	2	4		1	2		3		4	3	5	
4		2	4		5			4			2	
5			4	3	4	2					2	5
<u>6</u>	1		3		3			2			4	

Predict by taking weighted average:

$$r_{15} = (0.41*2 + 0.59*3) / (0.41+0.59) = 2.6$$

# Common Practice for Item-Item Collaborative Filtering

- Define **similarity**  $s_{ij}$  of items  $i$  and  $j$
- Select  $K$  nearest neighbors (KNN):  $N(i; \mathbf{x})$ 
  - Set of Items most similar to item  $i$ , **that were rated by  $\mathbf{x}$**
- Estimate rating  $r_{xi}$  as the weighted average:

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

$N(i; \mathbf{x})$  = set of items similar to item  $i$  that were rated by  $\mathbf{x}$

$s_{ij}$  = similarity of items  $i$  and  $j$

$r_{xj}$  = rating of user  $\mathbf{x}$  on item  $j$

# Item-Item vs. User-User

	Avatar	LOTR	Matrix	Pirates
Alice	1		0.8	
Bob		0.5		0.3
Carol	0.9		1	0.8
David			1	0.4

- In practice, it has been observed that item-item often works better than user-user
- **Why?** Items are simpler, users have multiple tastes

# Pros/Cons of Collaborative Filtering

- **+ Works for any kind of item**
  - No feature selection needed
- **- Cold Start:**
  - Need enough users in the system to find a match
- **- Sparsity:**
  - The user/ratings matrix is sparse
  - Hard to find users that have rated the same items
- **- First rater:**
  - Cannot recommend an item that has not been previously rated
  - New items, Esoteric items
- **- Popularity bias:**
  - Cannot recommend items to someone with unique taste
  - Tends to recommend popular items



# Hybrid Methods

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

# Remarks & Practical Tips

- **Evaluation**
- **Error metrics**
- **Complexity / Speed**

# Evaluation

**movies**

**users**

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

# Evaluation

**movies**

**users**

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

**Test Data Set**

# Evaluating Predictions

- **Compare predictions with known ratings**
  - **Root-mean-square error (RMSE)**
    - where  $\hat{r}_{xi}$  is predicted,  $r_{xi}$  is the true rating of  $x$  on  $i$
  - **Precision at top 10:**
    - % of those in top 10
  - **Rank Correlation:**
    - Spearman's *correlation* between system's and user's complete rankings

# Evaluating Predictions

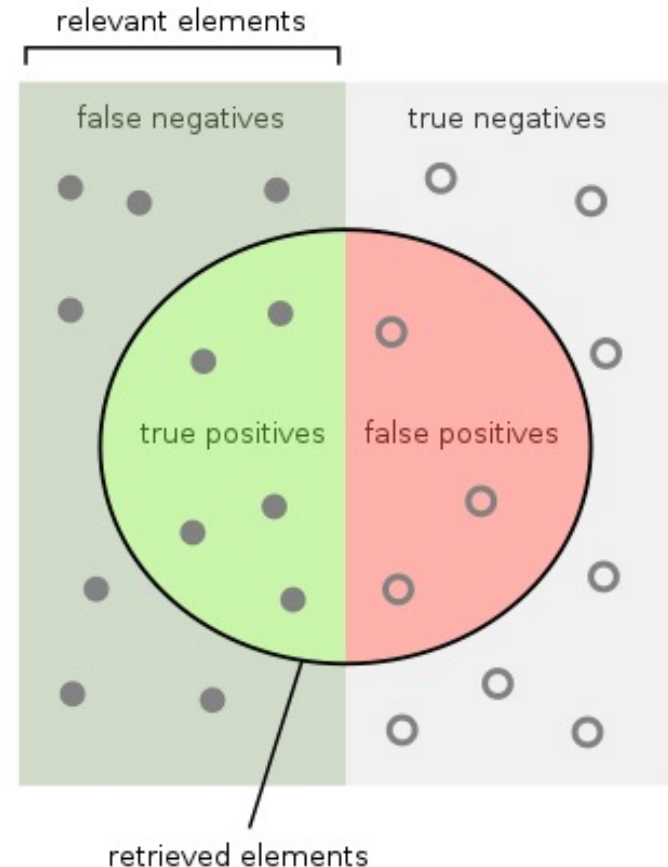
## Another approach: 0/1 model

- **Coverage:**
  - Number of items/users for which system can make predictions
- **Precision** =  $TP / (TP + FP)$
- **Accuracy** =  $(TP+TN) / (TP + FP + TN + FN)$
- **Receiver Operating characteristic (ROC) Curve**

Y-axis: True Positive Rates (TPR) ; X-axis False Positive Rates (FPR)

- $TPR$  (aka Recall) =  $TP / P = TP / (TP+FN)$  ;
- $FPR = FP / N = FP / (FP + TN)$
- See

[https://en.wikipedia.org/wiki/Precision\\_and\\_recall](https://en.wikipedia.org/wiki/Precision_and_recall)



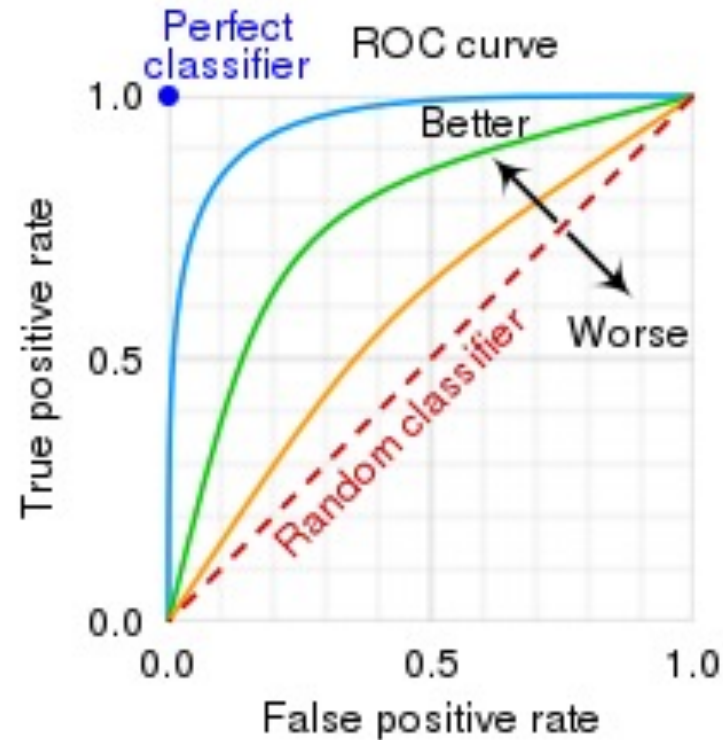
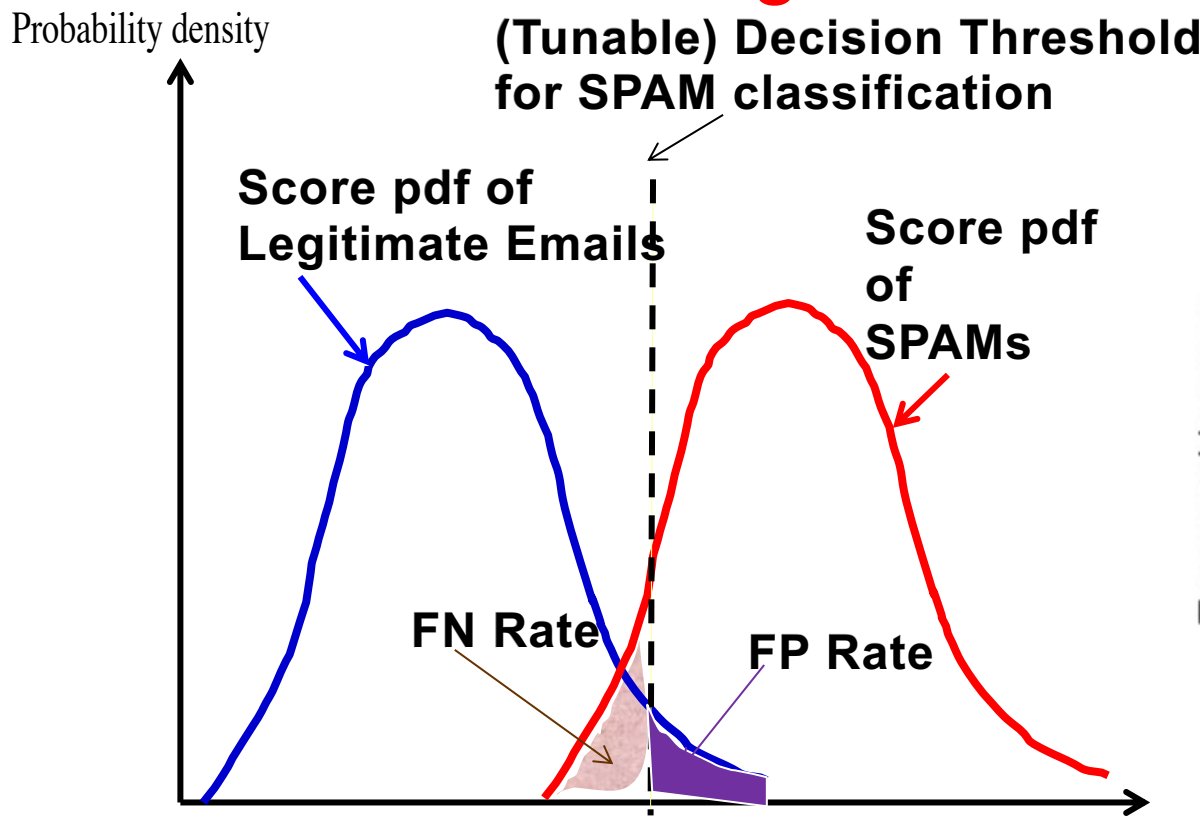
How many retrieved items are relevant?

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

How many relevant items are retrieved?

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

# Evaluating Predictions Example



## Suspicious Score computed by SPAM filtering Engine

- **Precision** =  $TP / (TP + FP)$
- **Accuracy** =  $(TP+TN) / (TP + FP + TN + FN)$
- **Receiver Operating characteristic (ROC) Curve**  
Plot True Positive Rates (TP Rate) against False Positive Rates (FP Rate)
- Y-axis: TP Rate (aka Recall) =  $1 - FN Rate = TP / P = TP / (TP+FN)$  ;
- X-axis: FP Rate (= false-alarm prob ; Fall-out) =  $FP / N = FP / (FP + TN)$

# Problems with Error Measures

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



# Item-Item CF (e.g. $K = 2$ )

		users												
		1	2	3	4	5	6	7	8	9	10	11	12	
movies	1	1		3		?	5			5		4		1.00
	2			5	4			4			2	1	3	-0.18
	<u>3</u>	2	4		1	2		3		4	3	5		<u>0.41</u>
	4		2	4		5			4			2		-0.10
	5			4	3	4	2					2	5	-0.31
	<u>6</u>	1		3		3			2			4		<u>0.59</u>

## K Nearest Neighbor (KNN) selection:

Identify movies similar to movie 1, rated by user 5

Here we use Pearson correlation as similarity:

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

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

$$\text{row 1: } [-2.6, 0, -0.6, 0, 0, 1.4, 0, 0, 1.4, 0, 0.4, 0]$$

2) Compute cosine similarities between rows

# Collaborative Filtering: Complexity

- Expensive step is finding  $k$  most similar customers:  $O(|X|)$ 
  - Recall that  $X$  = set of customers in the system
- **Too expensive to do at runtime**
  - Could pre-compute using clustering as approx.
- Naïve pre-computation takes time  $O(N \cdot |C|)$ 
  - $|C|$  = # of clusters =  $k$  in the  $k$ -means ;  $N$  = # of data points ;
- **We already know how to do this!**
  - Near-neighbor search in high dimensions (**LSH**)
  - Clustering
  - Dimensionality reduction

# Tip: Add Data

## ○ Leverage all the data

- Don't try to reduce data size in an effort to make fancy algorithms work
- Simple methods on large data do best

## ○ Add more data

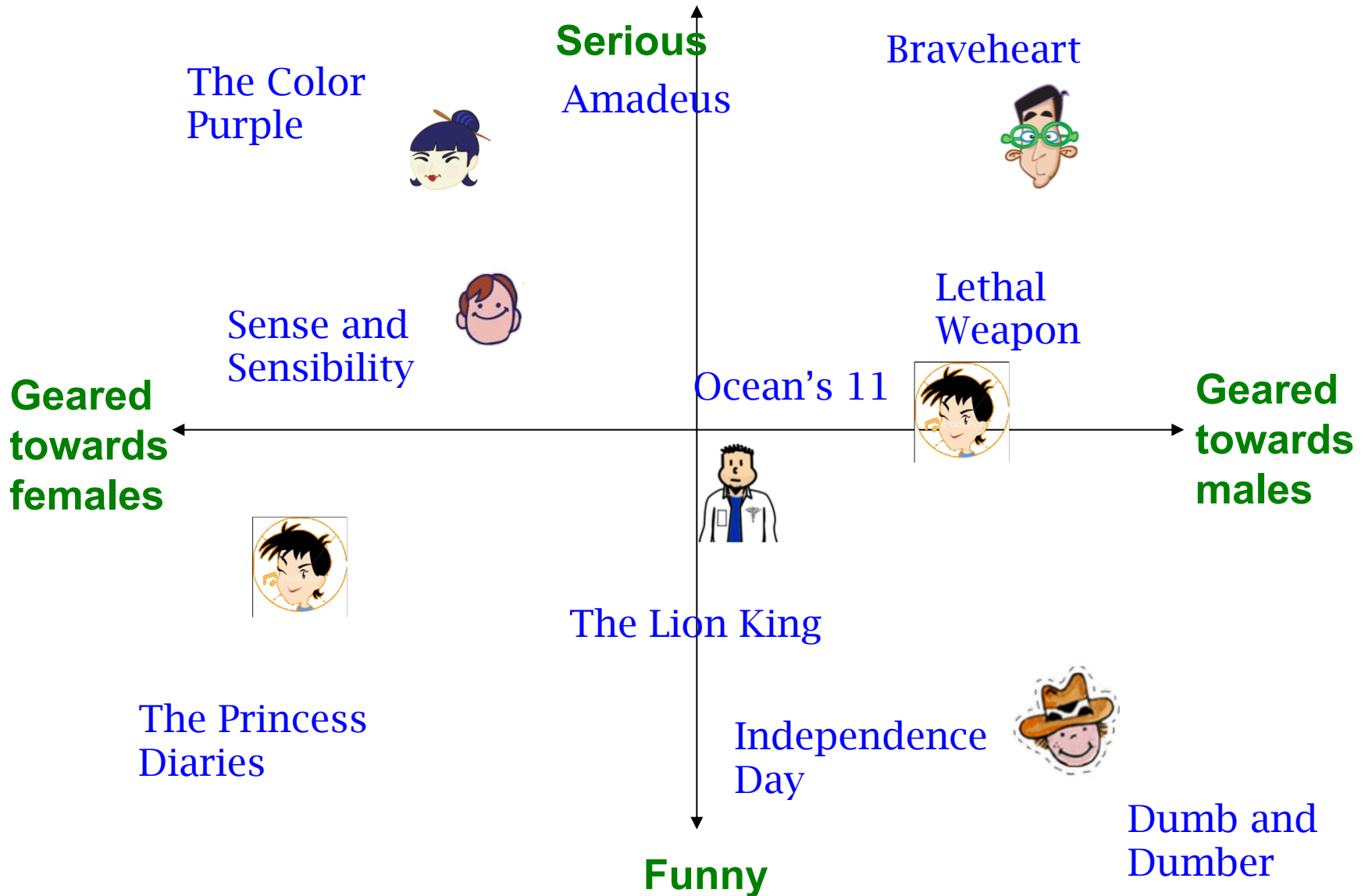
- e.g., add IMDB data on genres

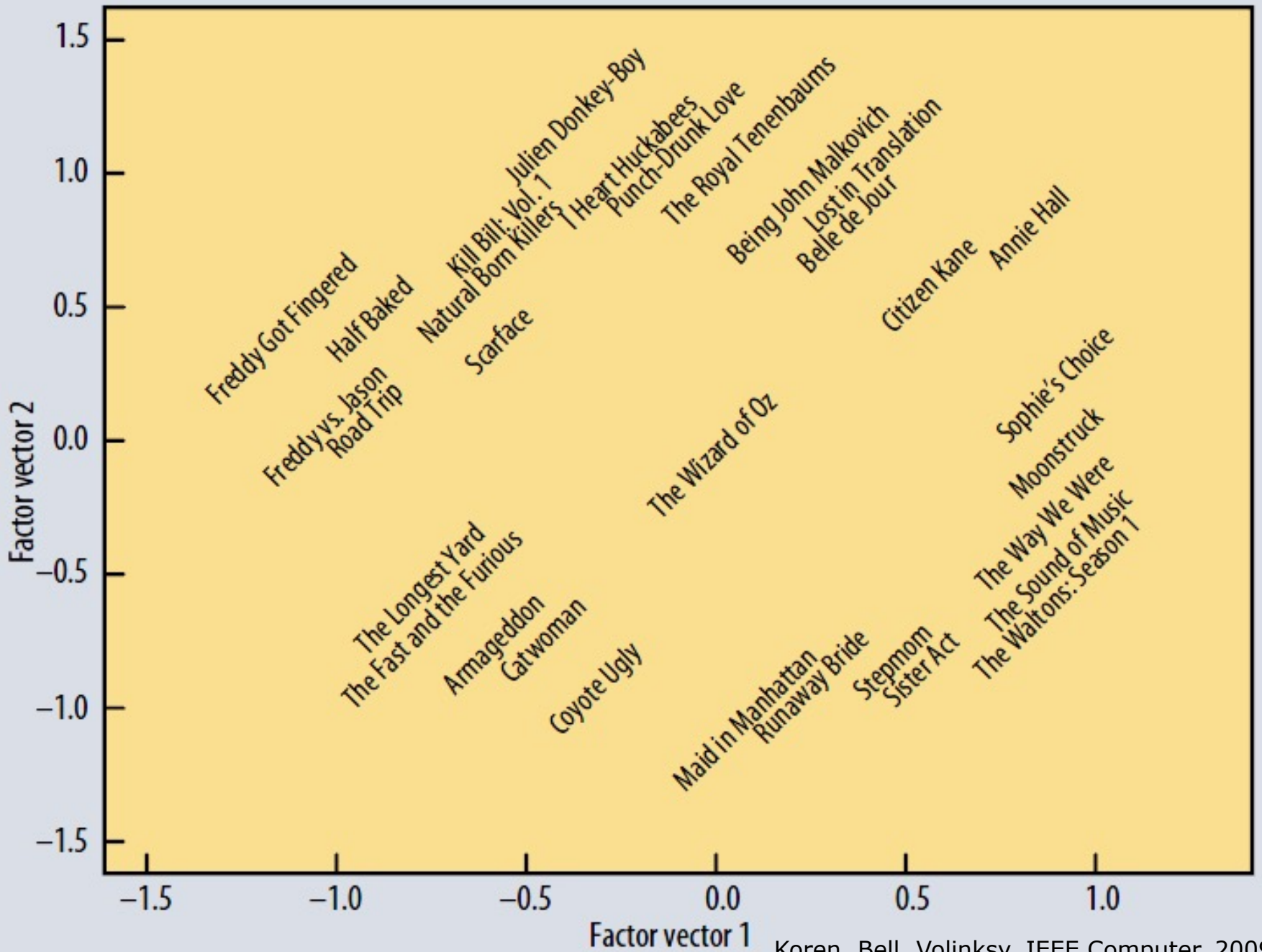
## ○ More data beats better algorithms

<http://anand.typepad.com/datawocky/2008/03/more-data-usual.html>

# Recommender Systems: Latent Factor Models

# Collaborative Filtering via Latent Factor Models (e.g., SVD)





# 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					

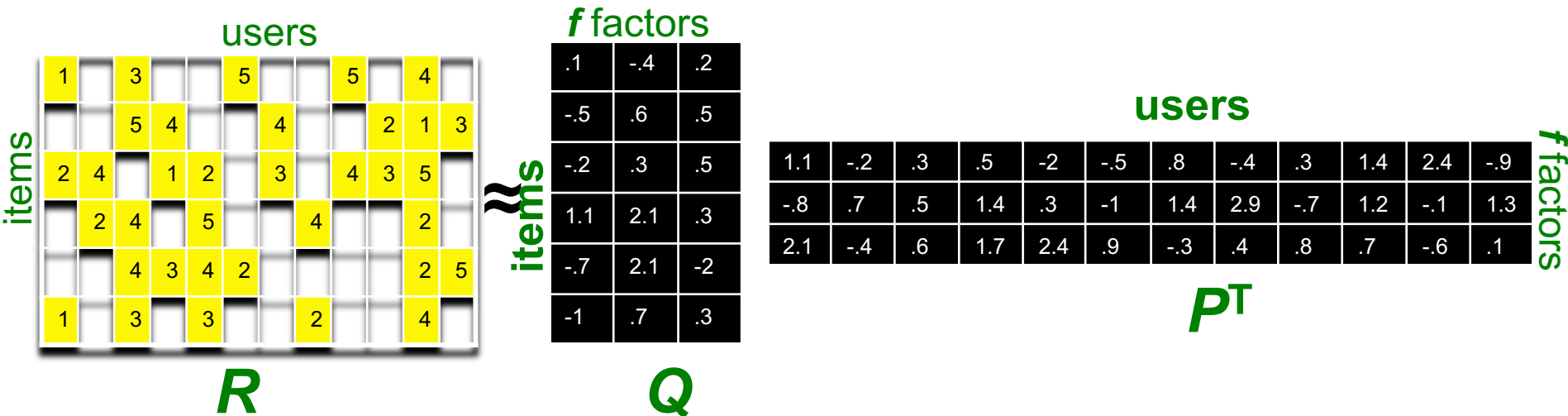




# Latent Factor Models

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

$$\text{SVD: } A = U \Sigma V^T$$



- For now let's assume we can approximate the rating 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

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

users

	1	3		5		5	4		
items		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	

≈

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

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

$q_i$  = 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

$f$  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

$P^T$

# 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	?	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			

≈

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

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

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

	.1	-.4	.2
items	-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

$P^T$

# Ratings as Products of Factors

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

users

	1	3		5		5	4	
items		5	4	2.4	4		2	1
	2	4		1	2	3	4	3
		2	4	5		4		2
		4	3	4	2			2
	1	3	3		2		4	

≈

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

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

$q_i$  = 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

$f$  factors

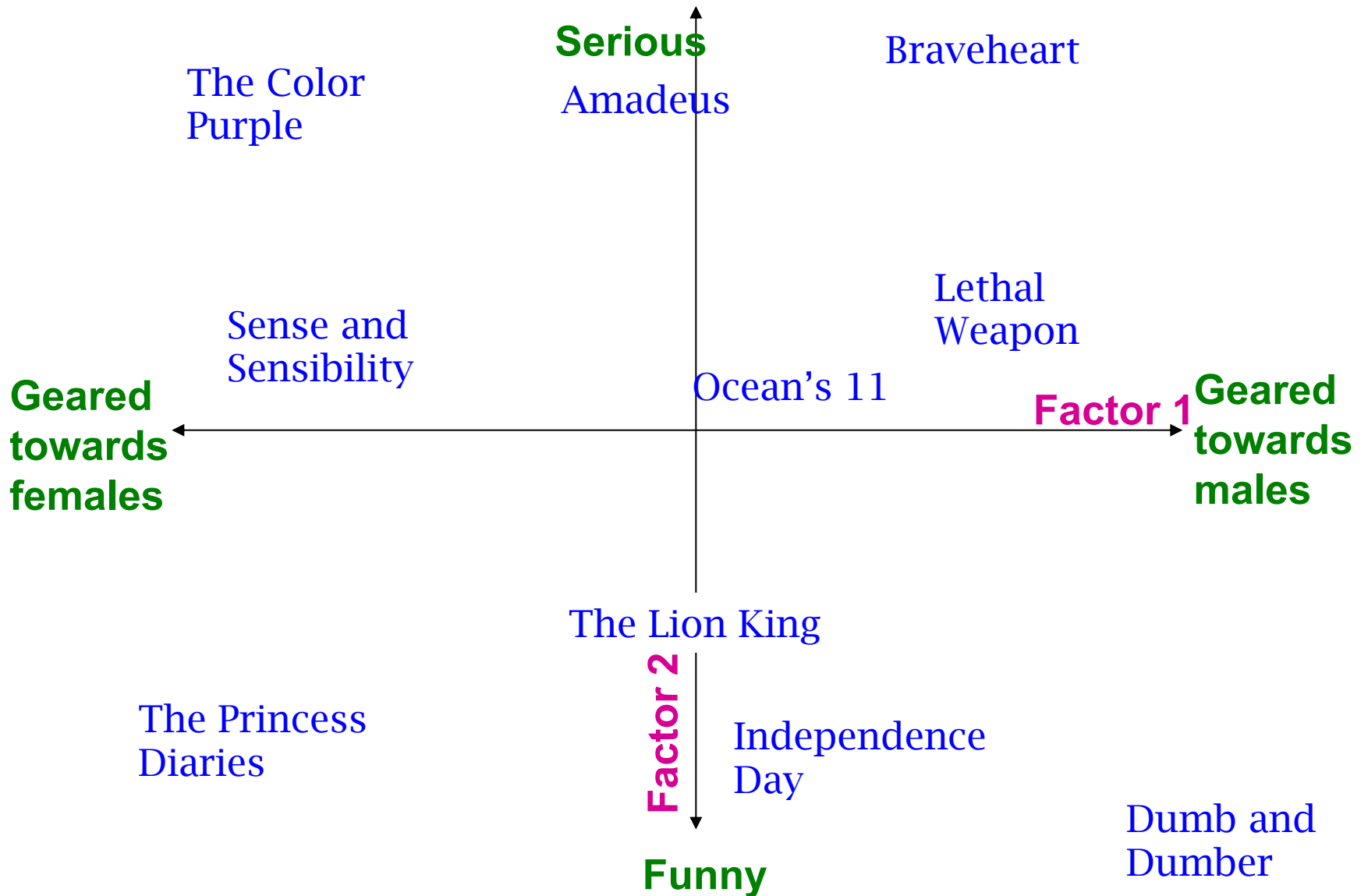
$Q$

users

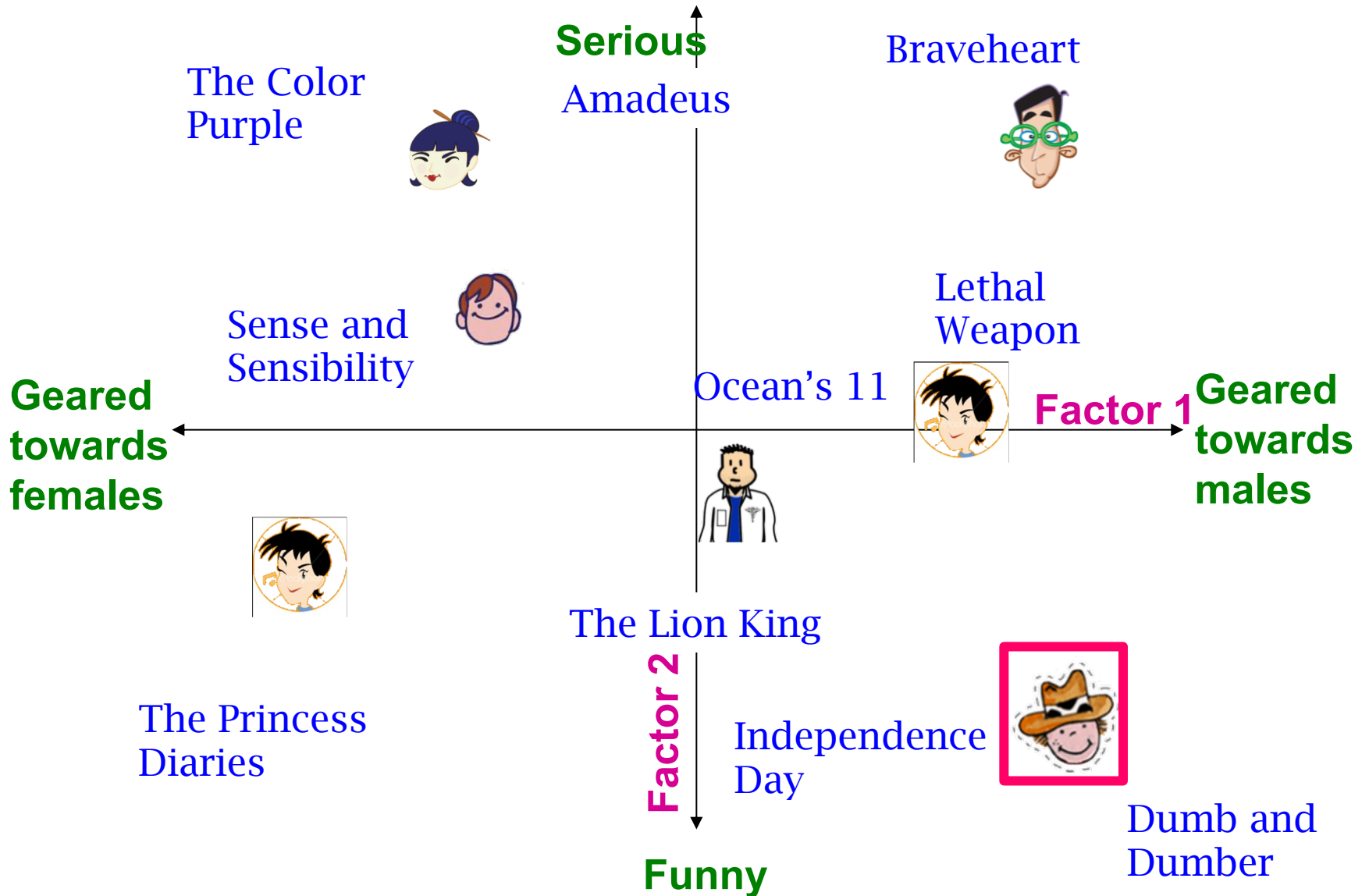
$f$ factors	1.1	-.2	.3	.5	-.2	-.5	.8	-.4	.3	1.4	2.4	-.9
	-.8	.7	.5	1.4	.3	-.1	1.4	2.9	-.7	1.2	-.1	1.3
	2.1	-.4	.6	1.7	2.4	.9	-.3	.4	.8	.7	-.6	.1

$P^T$

# Latent Factor Models



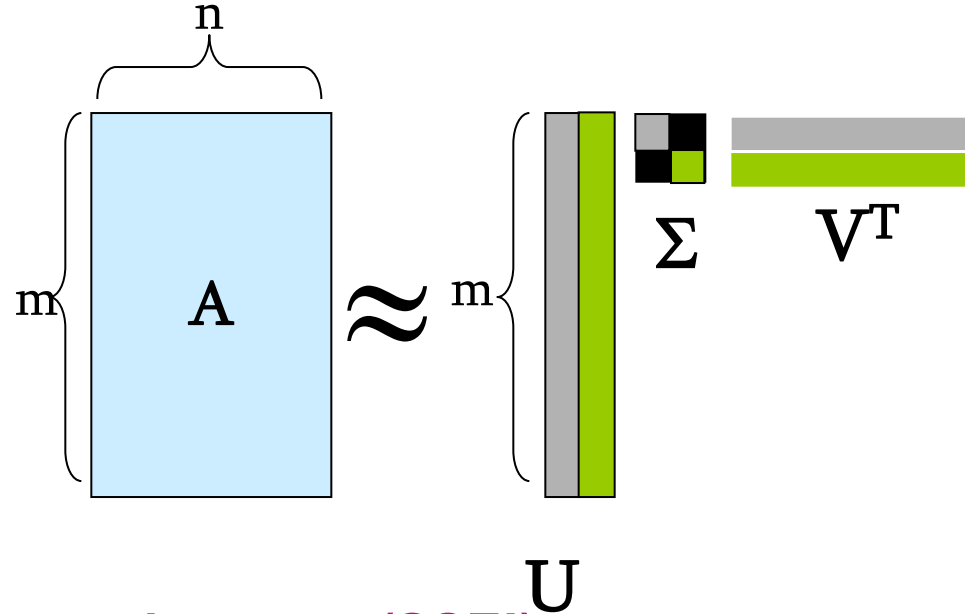
# Latent Factor Models



# Recap: SVD

## Remember SVD:

- **A**: Input data matrix
- **U**: Left singular vecs
- **V**: Right singular vecs
- $\Sigma$ : Singular values



- **SVD gives minimum reconstruction error (SSE!)**

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

The sum goes over all entries.

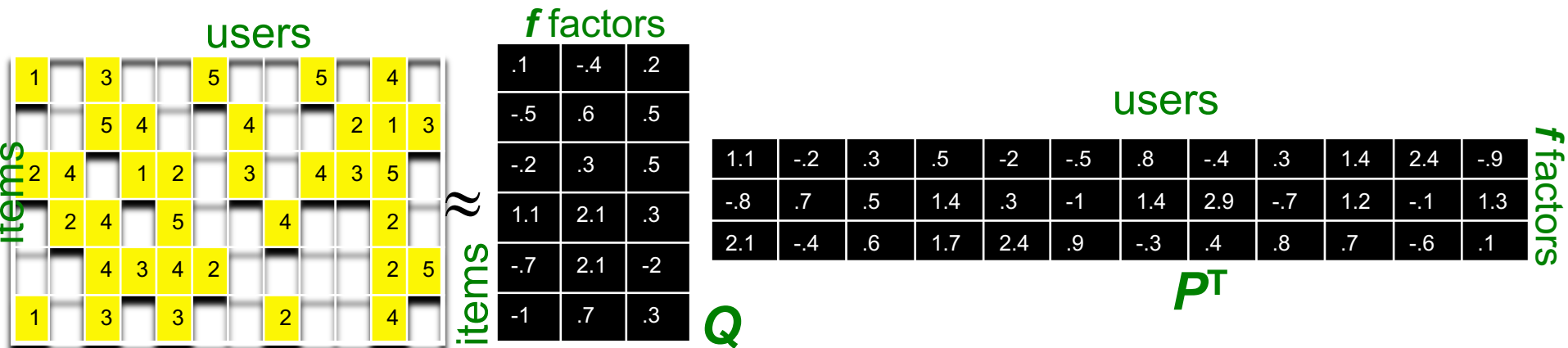
But our **R** has missing entries!

- So in our case, “SVD” on Netflix data:  $R \approx Q \cdot P^T$

- $A = R, Q = U, P^T = \Sigma V^T$   $\hat{r}_{xi} = q_i \cdot p_x^T$

- **But, we are not done yet! R has missing entries!**

# Latent Factor Models



- SVD isn't defined when entries are missing!

- Use specialized methods to find  $P$ ,  $Q$

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

- **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



Degression to the lecture notes of  
Regression and Gradient Descent  
by Andrew Ng's  
Machine Learning course from Coursera

# Dealing with Missing Entries

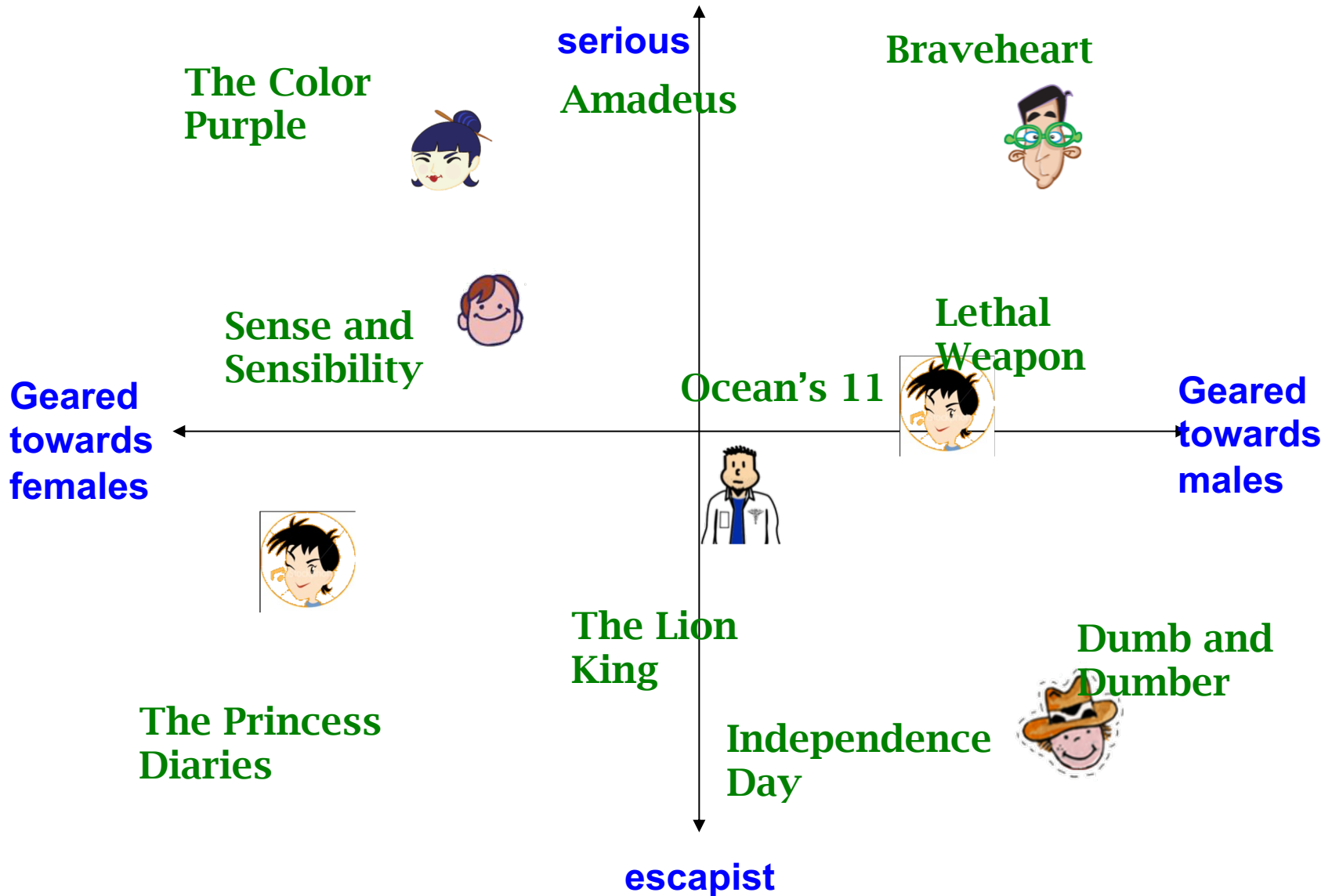
- **Want to minimize SSE for unseen test data**
- **Idea: Minimize SSE on training data**
  - Want large  $f$  (# of factors) to capture all the signals
  - But, **SSE** on test data begins to rise for  $f > 2$
- **Regularization is needed to avoid Overfitting !**
  - Allow rich model where there are sufficient data
  - Shrink aggressively where data are scarce

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

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

$\lambda$ ... regularization parameter

# Recommendations via Latent Factor Models (e.g., SVD++ by the [Bellkor Team])



# Dealing with Missing Entries

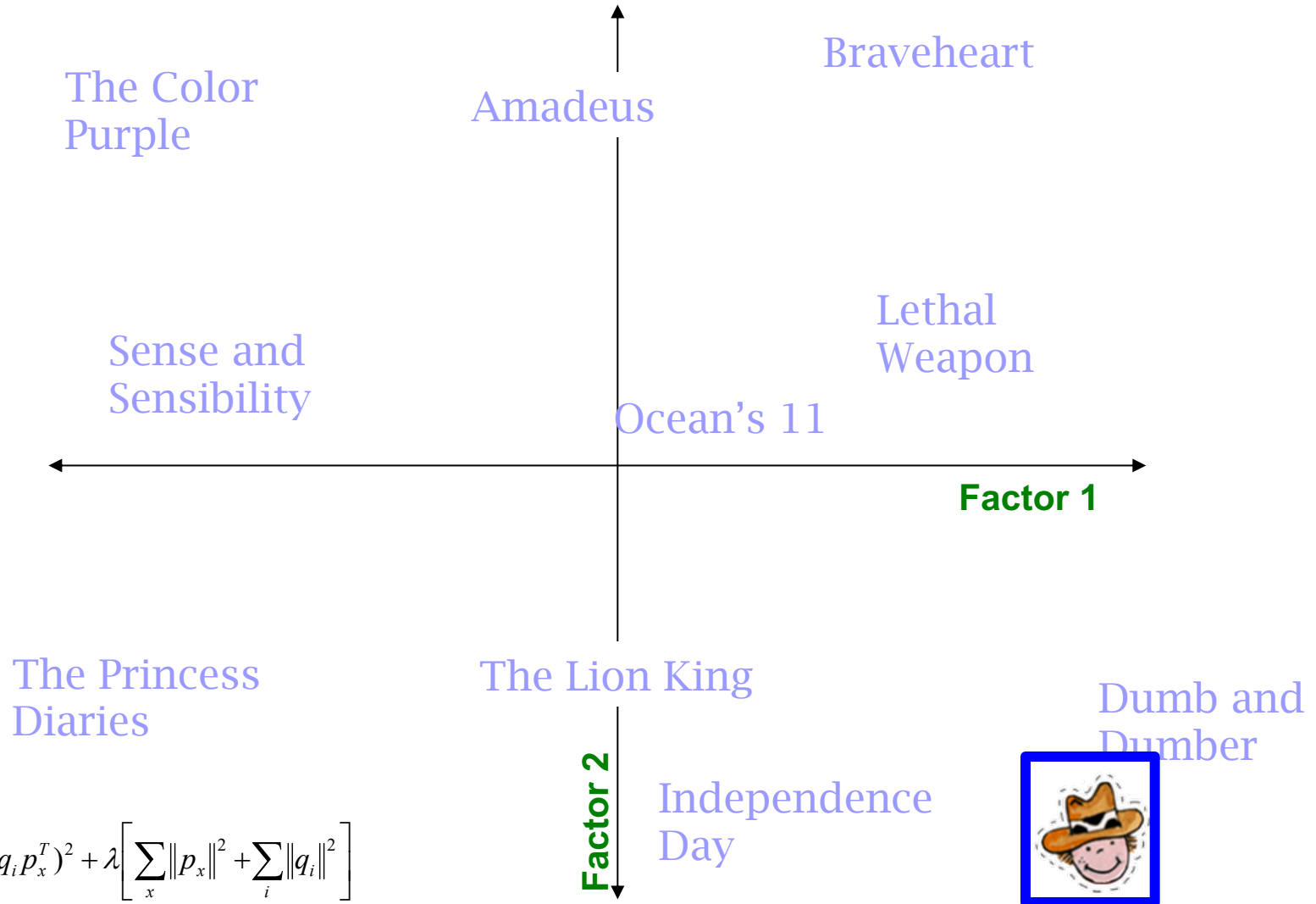
- **Want to minimize SSE for unseen test data**
- **Idea: Minimize SSE on training data**
  - Want large  $f$  (# of factors) to capture all the signals
  - But, **SSE** on test data begins to rise for  $f > 2$
- **Regularization is needed!**
  - Allow rich model where there are sufficient data
  - Shrink aggressively where data are scarce

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

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

$\lambda$ ... regularization parameter

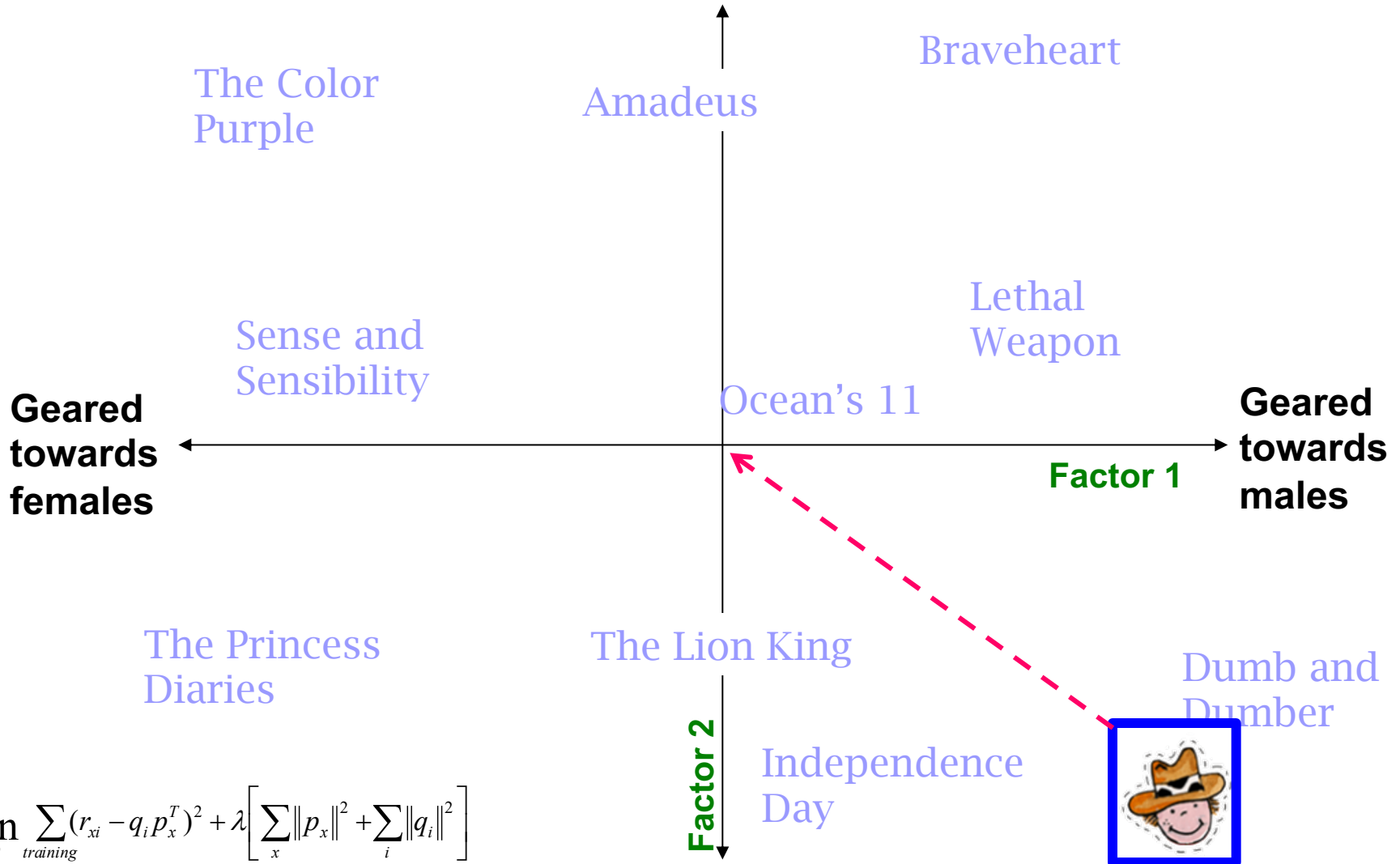
# The Effect of Regularization



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

$\min_{\text{factors}}$  "error" +  $\lambda$  "length"

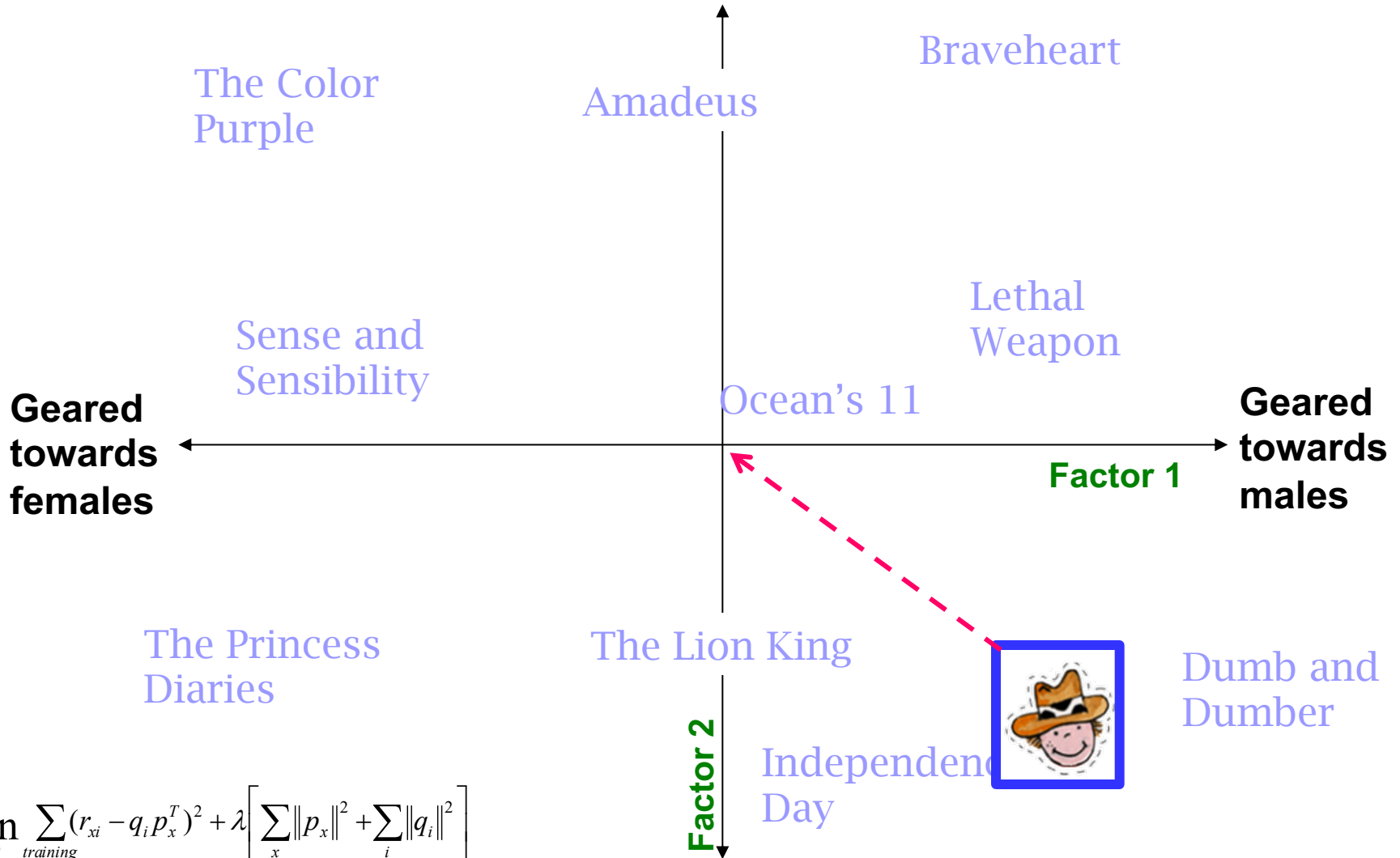
# The Effect of Regularization



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

$\min_{\text{factors}}$  "error" +  $\lambda$  "length"

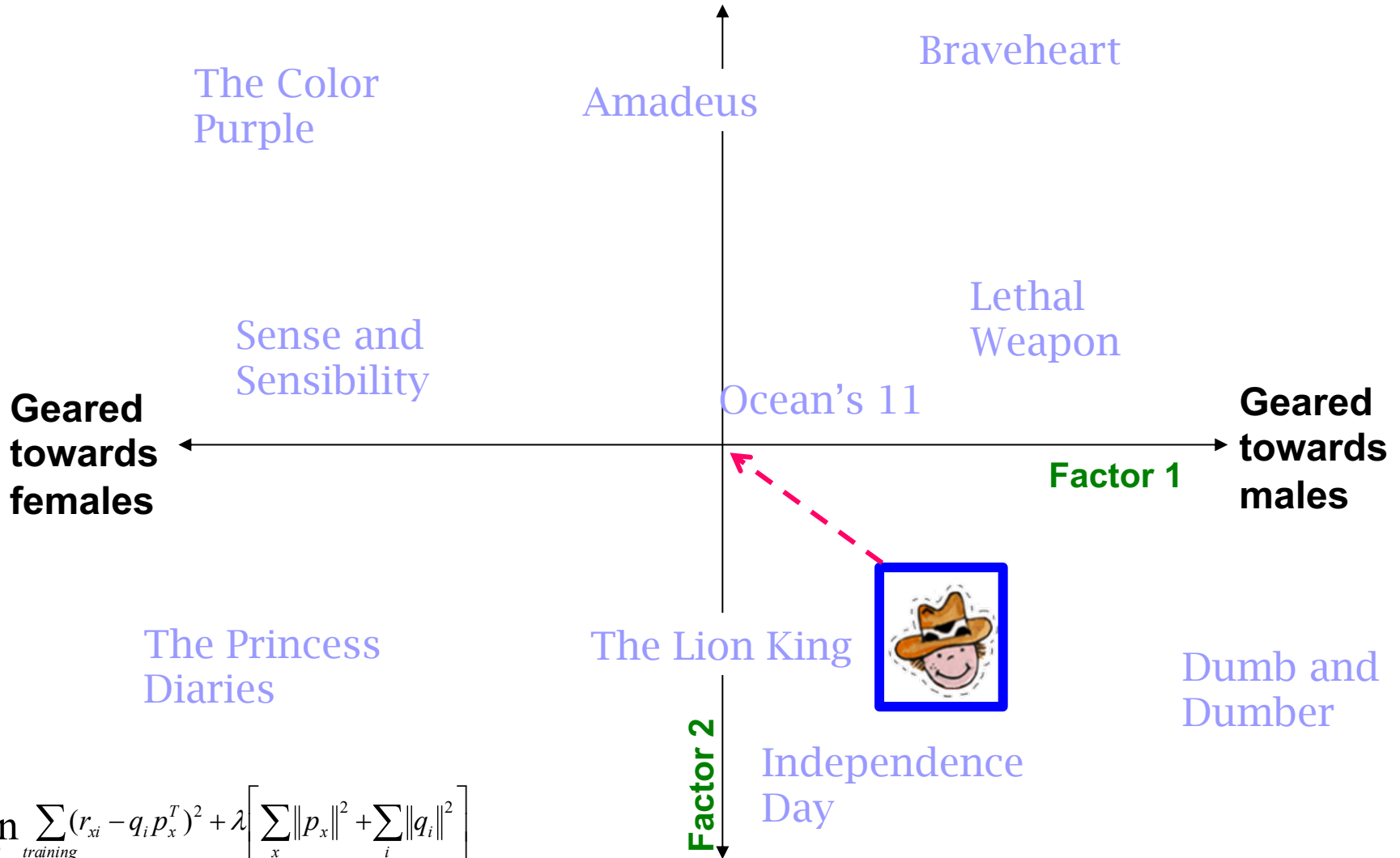
# The Effect of Regularization



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

$\min_{\text{factors}}$  “error” +  $\lambda$  “length”

# The Effect of Regularization

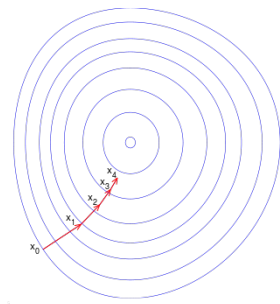


$$\min_{P, Q} \sum_{\text{training}} (r_{xi} - q_i p_x^T)^2 + \lambda \left[ \sum_x \|p_x\|^2 + \sum_i \|q_i\|^2 \right]$$

$\min_{\text{factors}}$  "error" +  $\lambda$  "length"



# Use Gradient Descent to search for the optimal settings



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

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

- Gradient descent:

- Initialize  $P$  and  $Q$  (using SVD, pretend missing ratings are 0)

- Do gradient descent:

- $P \leftarrow P - \eta \cdot \nabla P$

- $Q \leftarrow Q - \eta \cdot \nabla Q$

- Where  $\nabla Q$  is gradient/derivative of matrix  $Q$ :

$$\nabla Q = [\nabla q_{if}] \text{ and } \nabla q_{if} = \sum_{x,i} -2(r_{xi} - q_i p_x^T) p_{xf} + 2\lambda q_{if}$$

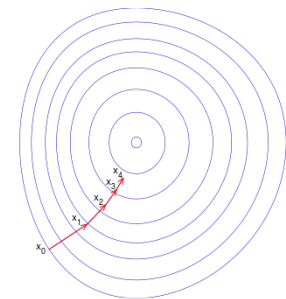
- Here  $q_{if}$  is entry  $f$  of row  $q_i$  of matrix  $Q$

- Observation: Computing gradients is slow!

How to compute gradient of a matrix?

Compute gradient of every element independently!

# (Batch) Gradient Descent



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

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

- Gradient descent:

- Initialize  $P$  and  $Q$  (using SVD, pretend missing ratings are 0)

- Do gradient descent:

- $P \leftarrow P - \eta \cdot \nabla P$

- $Q \leftarrow Q - \eta \cdot \nabla Q$

- Where  $\nabla Q$  is gradient/derivative of matrix  $Q$ :

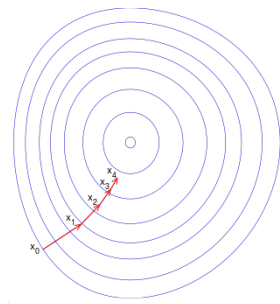
$$\nabla Q = [\nabla q_{if}] \text{ and } \nabla q_{if} = \sum_{x,i} -2(r_{xi} - q_i p_x^T) p_{xf} + 2\lambda q_{if}$$

- Here  $q_{if}$  is entry  $f$  of row  $q_i$  of matrix  $Q$

- Observation: Computing gradients is slow!

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

# Stochastic Gradient Descent



## ○ Gradient Descent (GD) vs. Stochastic GD

- **Observation:**  $\nabla Q = [\nabla q_{if}]$  where

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

- Here  $q_{if}$  is entry  $f$  of row  $q_i$  of matrix  $Q$

- $Q = Q - \eta \nabla Q = Q - \eta [\sum_{x,i} \nabla Q(r_{xi})]$

- **Idea:** Instead of evaluating gradient over all ratings evaluate it for each individual rating and make a step

## ○ GD: $Q \leftarrow Q - \eta [\sum_{r_{xi}} \nabla Q(r_{xi})]$

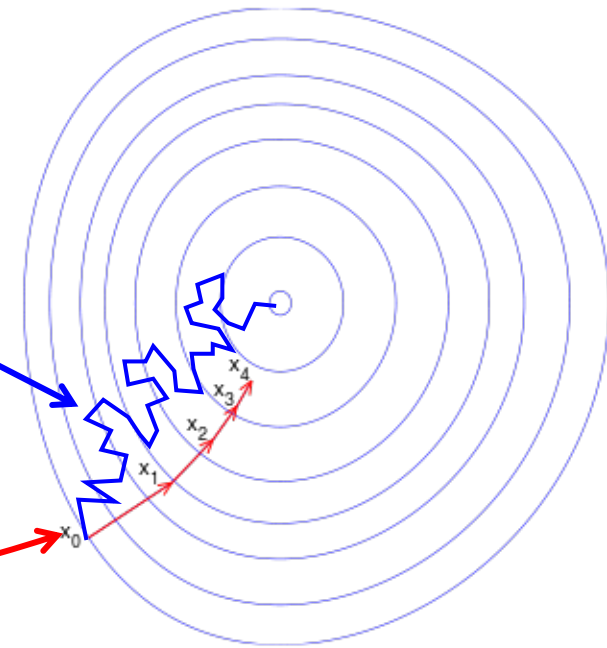
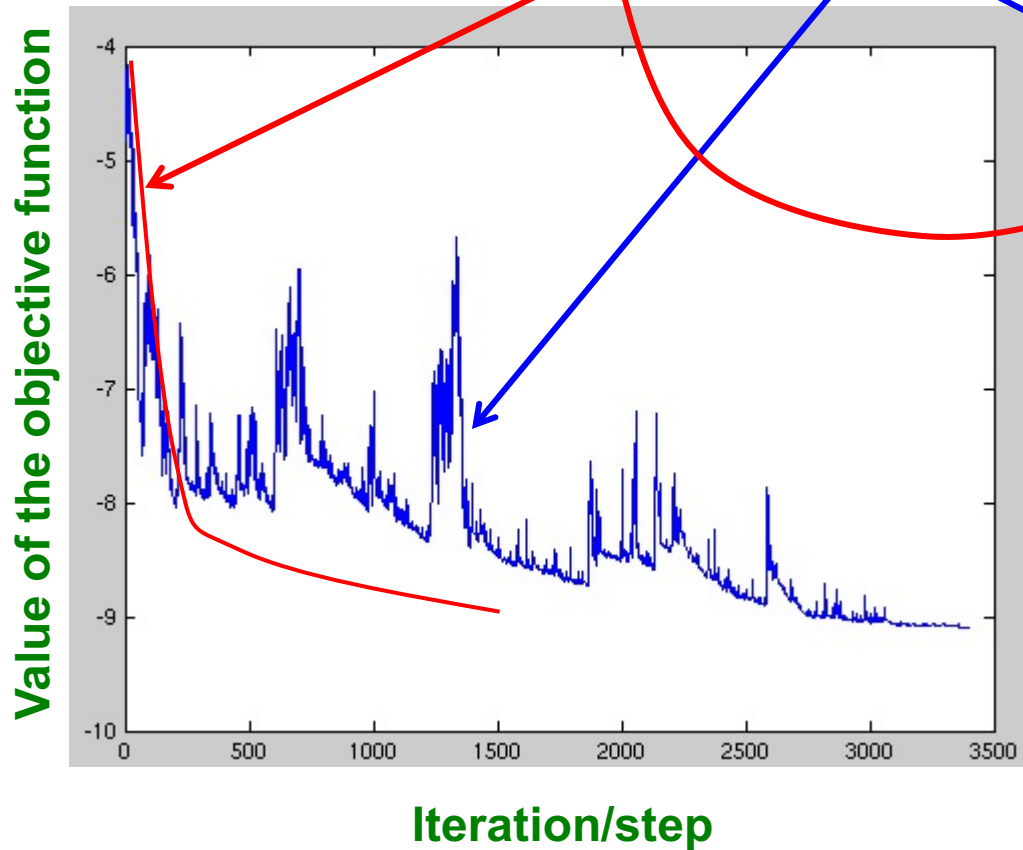
## ○ SGD: $Q \leftarrow Q - \eta \nabla Q(r_{xi})$

- **Faster convergence!**

- Need more steps but each step is computed much faster

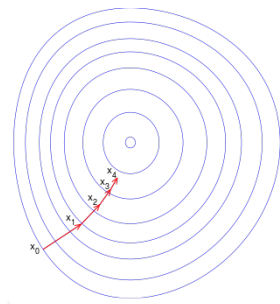
# SGD vs. GD

- 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  $\mathbf{P}$  and  $\mathbf{Q}$  (using SVD, pretend missing ratings are 0)
- Then iterate over the ratings (multiple times if necessary) and update factors:

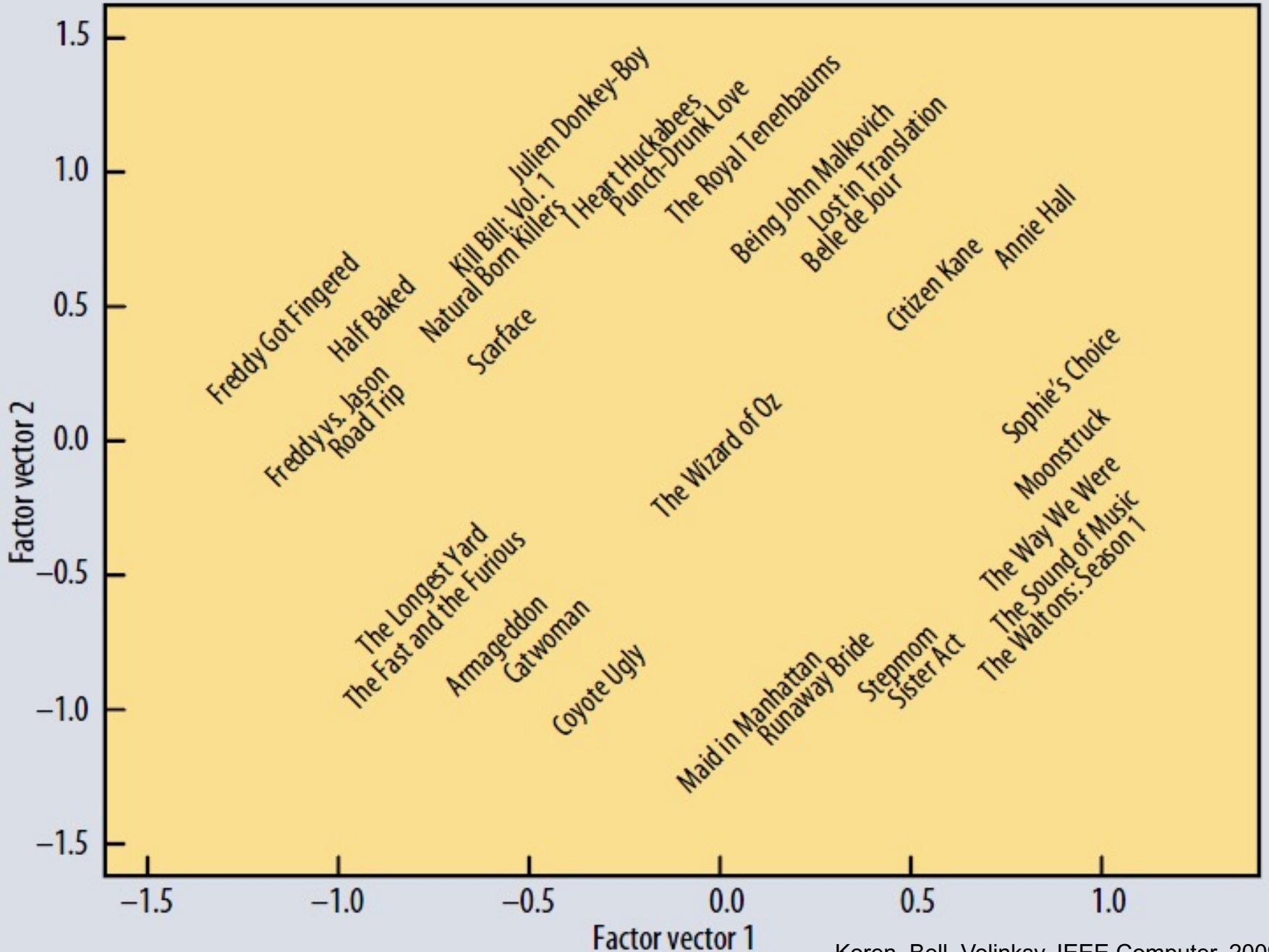
For each  $r_{xi}$ :

- $error = \varepsilon_{xi} = r_{xi} - q_i \cdot p_x^T$  1. Take derivative of the square of error
- $q_i \leftarrow q_i + \eta (\varepsilon_{xi} p_x - \lambda q_i)$  2. Update  $q_i$
- $p_x \leftarrow p_x + \eta (\varepsilon_{xi} q_i - \lambda p_x)$  3. Update  $p_x$

## ○ 2 for loops:

- For until convergence:
  - For each  $r_{xi}$ 
    - Compute gradient, do a “step”

... learning rate



# Summary: Recommendations via Optimization

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

- **Goal:** Make good recommendations
  - Quantify goodness using **SSE**:  
So, **Lower SSE means better recommendations**
  - We want to make good recommendations on items that some user has not yet seen.
  - **Let's set values for  $P$  and  $Q$  such that they work well on known (user, item) ratings**
  - And **hope** these values for  $P$  and  $Q$  will predict well the unknown ratings
- This is the a case where we apply **Optimization methods**

# Backup Slides

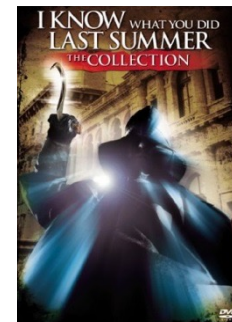
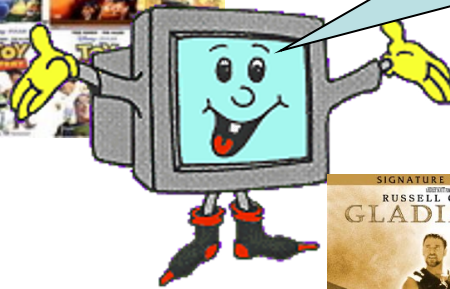


# The Netflix Challenge: 2006-09

# Recommender systems



We Know What You Ought  
To Be Watching This  
Summer



# Netflix Prize

Home Rules Leaderboard Register Update Submit Download

NETFLIX

Browse Recommendations Friends Queue Buy DVDs  
Home Genres New Releases Previews Netflix Top 100 Crib

## Movies For You

Randy, the following movies were chosen based on your interest in:  
[Bowling for Columbine](#)  
[Carnivale: Season 1](#)  
[Fahrenheit 9/11](#)



### The Big One

★★★★☆  
Aer subversive  
by from  
/



### Carnivale: Season 2

★★★★☆  
Disc Serie

Daniel Krau  
rivetingly cre  
series conti  
document t  
ventures of a motley cre  
mies who've made the C  
stbowl their ... [Read Mo](#)



### Roger & Me

★★★★☆  
In this bl  
satin

All Discs Guaranteed!

## You really liked it...

Now only for just \$5.99

Shop as low titles

Original art

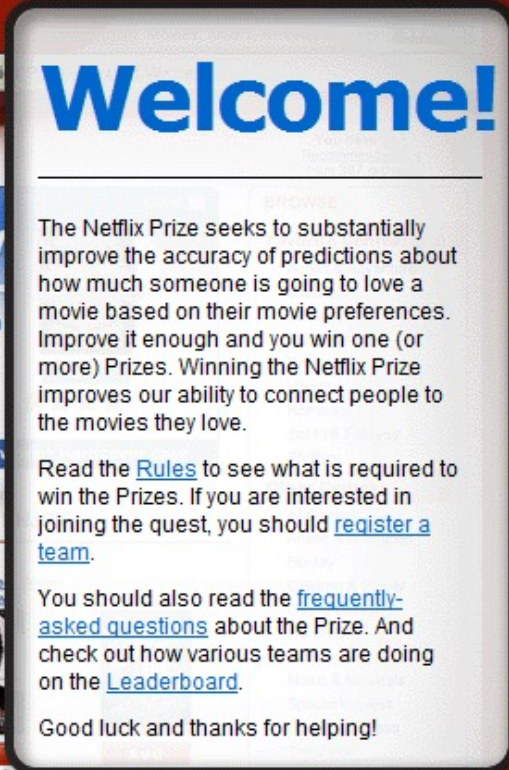
## Welcome!

The Netflix Prize seeks to substantially improve the accuracy of predictions about how much someone is going to love a movie based on their movie preferences. Improve it enough and you win one (or more) Prizes. Winning the Netflix Prize improves our ability to connect people to the movies they love.

Read the [Rules](#) to see what is required to win the Prizes. If you are interested in joining the quest, you should [register a team](#).

You should also read the [frequently-asked questions](#) about the Prize. And check out how various teams are doing on the [Leaderboard](#).

Good luck and thanks for helping!



# 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 squared error (RMSE)
  - Netflix Cinematch RMSE: 0.9514
- Competition
  - 2700+ teams
  - \$1 million grand prize for 10% improvement on Cinematch result
  - \$50,000 2007 progress prize for 8.43% improvement

# Movie rating data

Training data

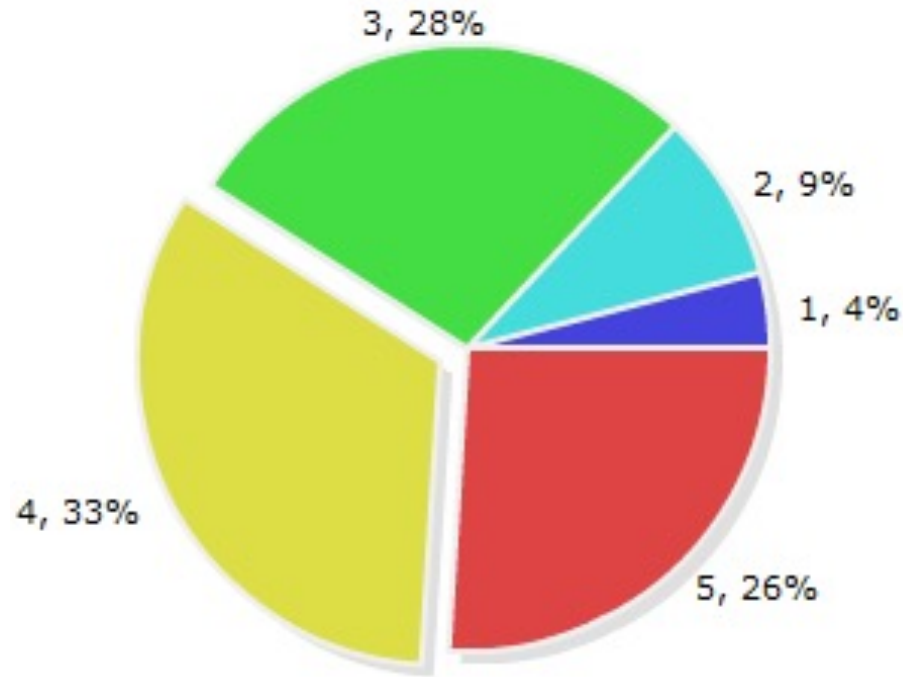
user	movie	score
1	21	1
1	213	5
2	345	4
2	123	4
2	768	3
3	76	5
4	45	4
5	568	1
5	342	2
5	234	2
6	76	5
6	56	4

Test data

user	movie	score
1	62	?
1	96	?
2	7	?
2	3	?
3	47	?
3	15	?
4	41	?
4	28	?
5	93	?
5	74	?
6	69	?
6	83	?

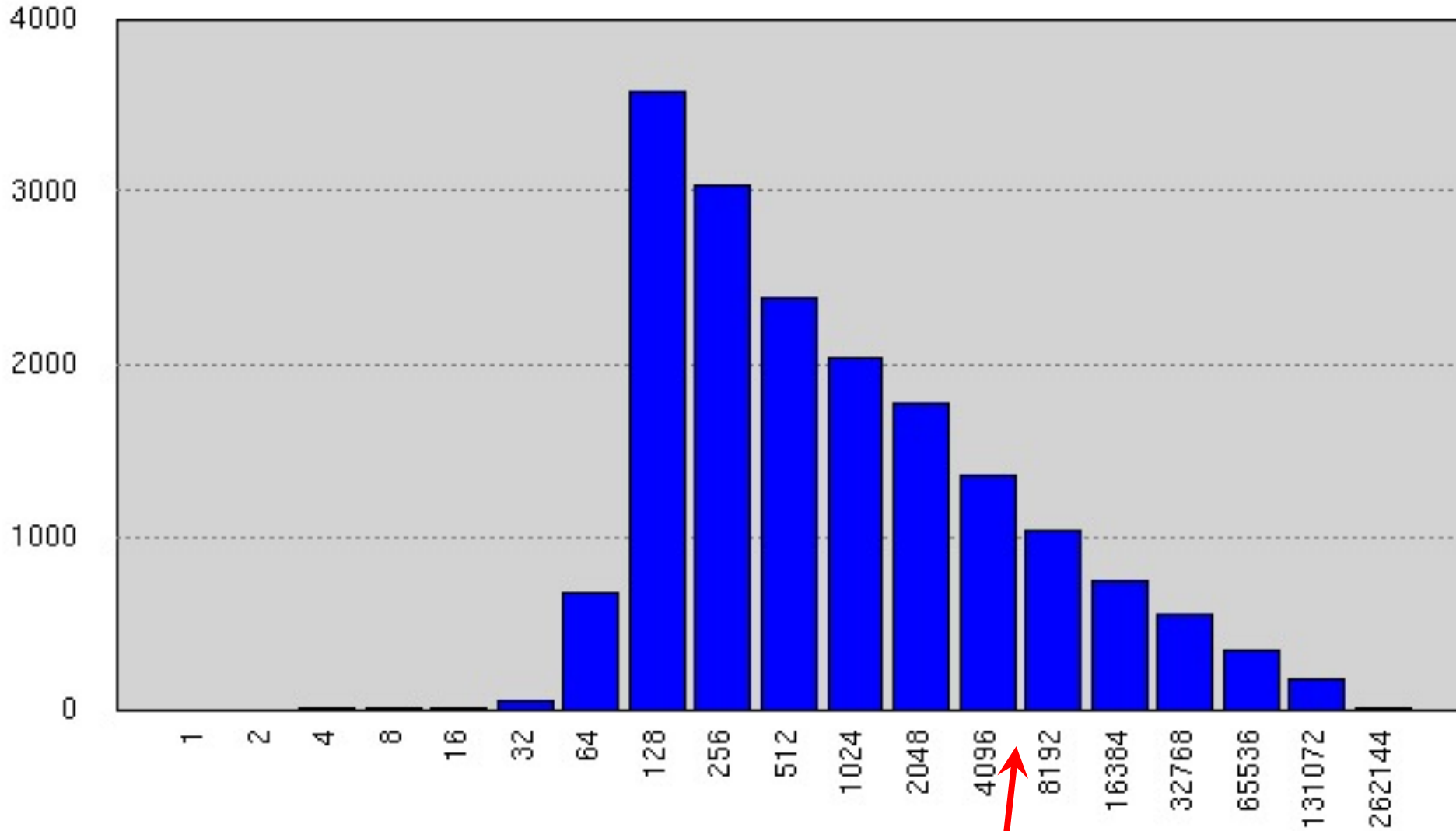


# Overall rating distribution



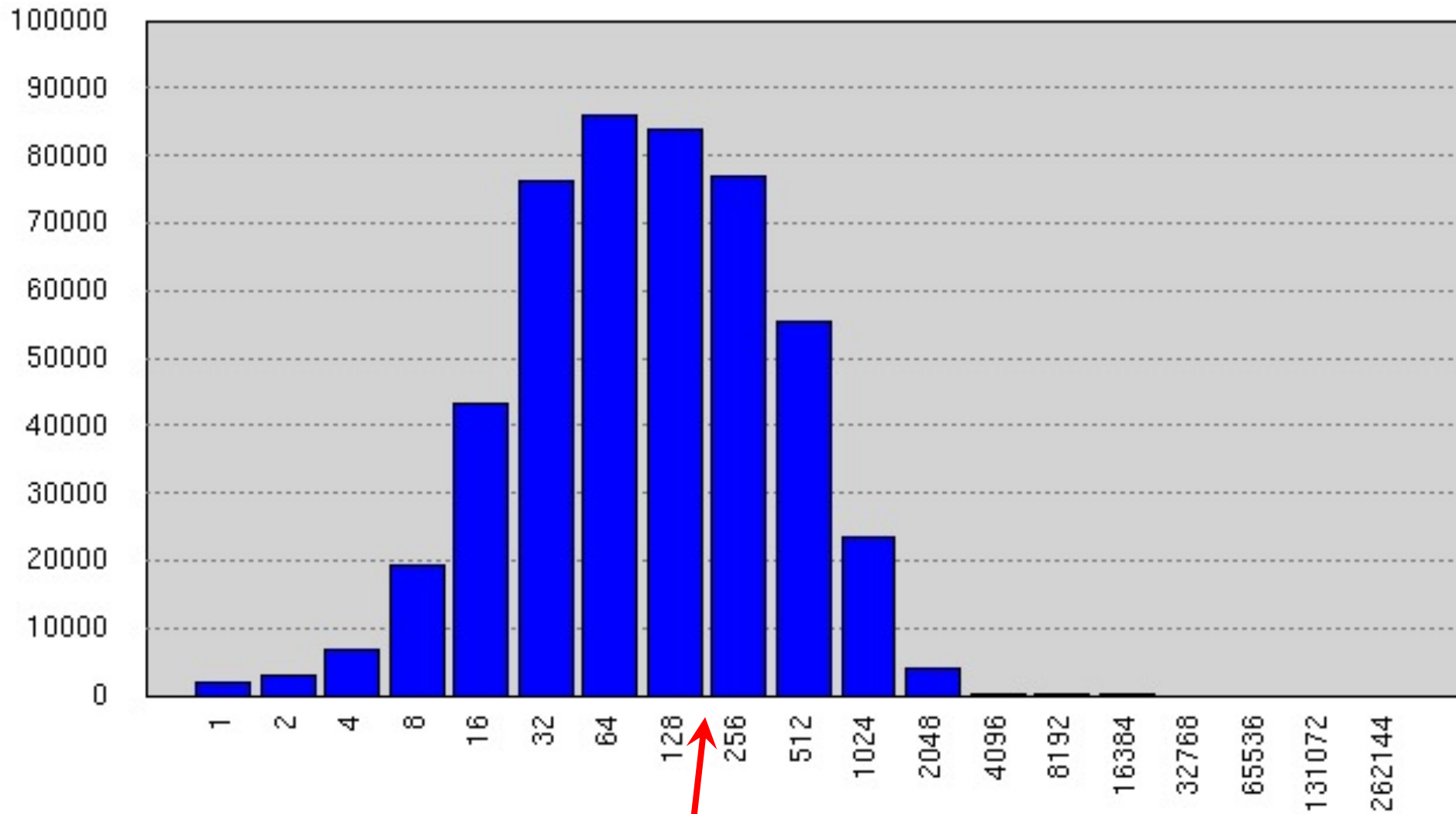
- Third of ratings are 4s
- Average rating is 3.68

# #ratings per movie



- Avg #ratings/movie: 5627

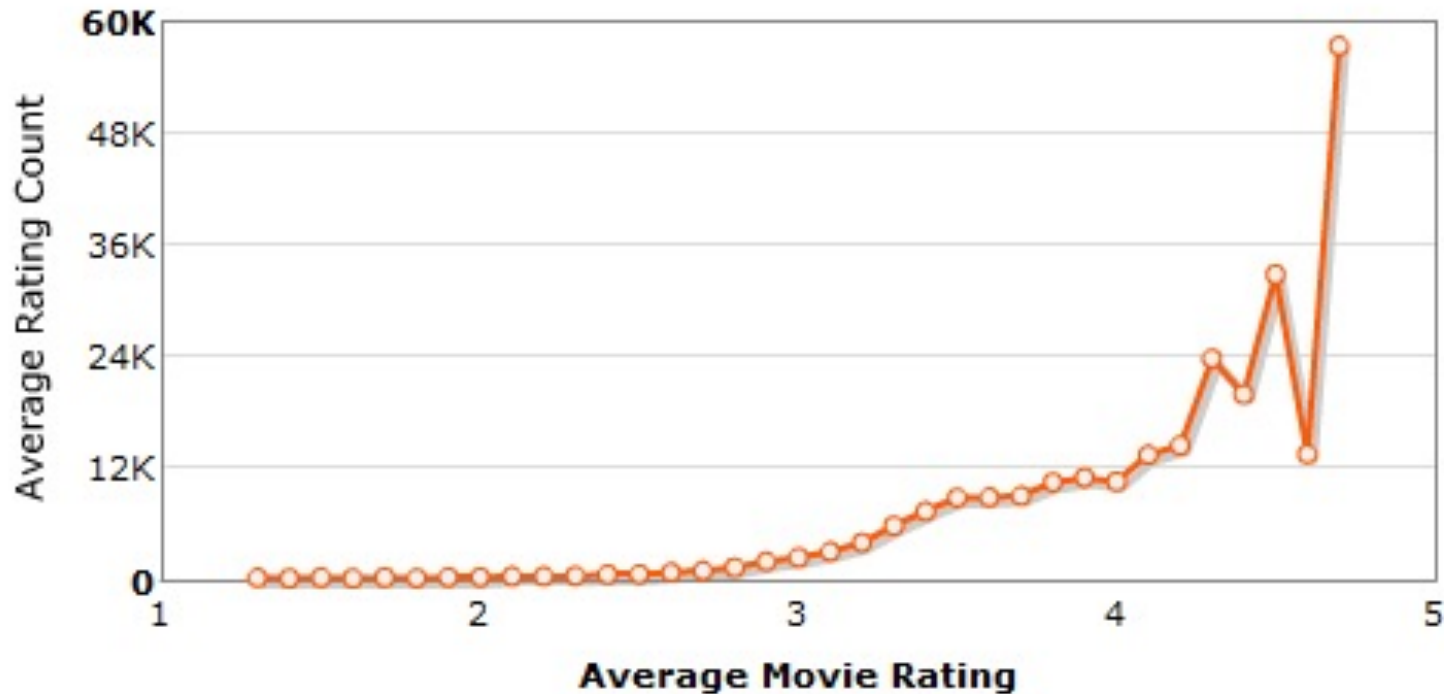
# #ratings per user



- Avg #ratings/user: 208



# Average movie rating by movie count



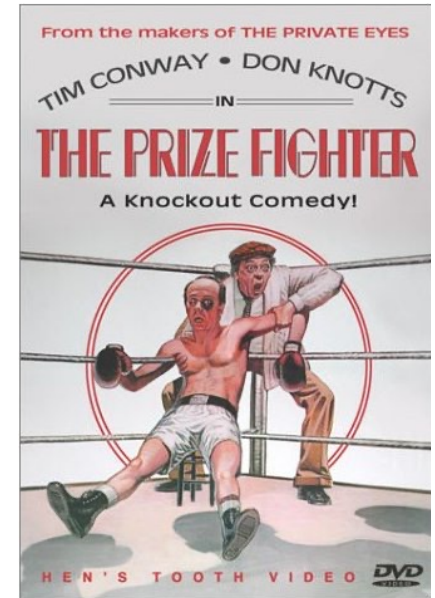
- More ratings to better movies

# Most loved movies

Title	Avg rating	Count
The Shawshank Redemption	4.593	137812
Lord of the Rings: The Return of the King	4.545	133597
The Green Mile	4.306	180883
Lord of the Rings: The Two Towers	4.460	150676
Finding Nemo	4.415	139050
Raiders of the Lost Ark	4.504	117456
Forrest Gump	4.299	180736
Lord of the Rings: The Fellowship of the ring	4.433	147932
The Sixth Sense	4.325	149199
Indiana Jones and the Last Crusade	4.333	144027

# Challenges

- Size of data
  - Scalability
  - Keeping data in memory
- Missing data
  - 99 percent missing
  - Very imbalanced
- Avoiding overfitting
- Test and training data differ significantly



movie #16322

# The BellKor recommender system

- Use an ensemble of **complementing** predictors
- Two, half tuned models worth more than a single, fully tuned model

# Extending Latent Factor Model to Include Biases

# Modeling Biases and Interactions

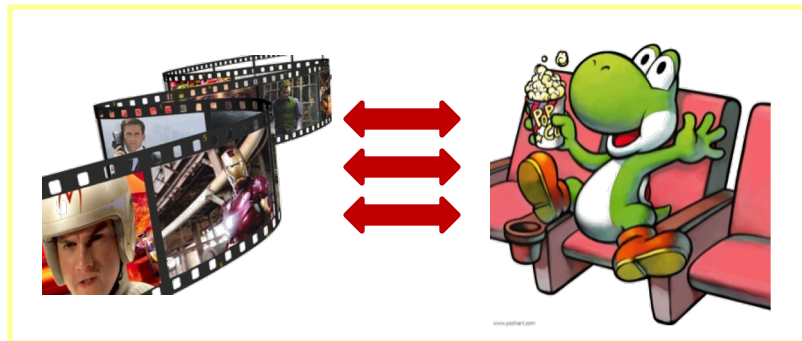
user bias



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

- $\mu$  = 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

$$r_{xi} = \underbrace{\mu}_{\text{Overall mean rating}} + \underbrace{b_x}_{\text{Bias for user } x} + \underbrace{b_i}_{\text{Bias for movie } i} + \underbrace{q_i \cdot p_x^T}_{\text{User-Movie interaction}}$$

## ○ 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:

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

goodness of fit

$$+ \lambda \left( \sum_i \|q_i\|^2 + \sum_x \|p_x\|^2 + \sum_x \|b_x\|^2 + \sum_i \|b_i\|^2 \right)$$

regularization

$\lambda$  is selected via grid-search on a validation set

- Stochastic gradient descent to find parameters

- **Note:** Both biases  $b_u$ ,  $b_i$  as well as interactions  $q_i$ ,  $p_u$  are treated as parameters (we estimate them)

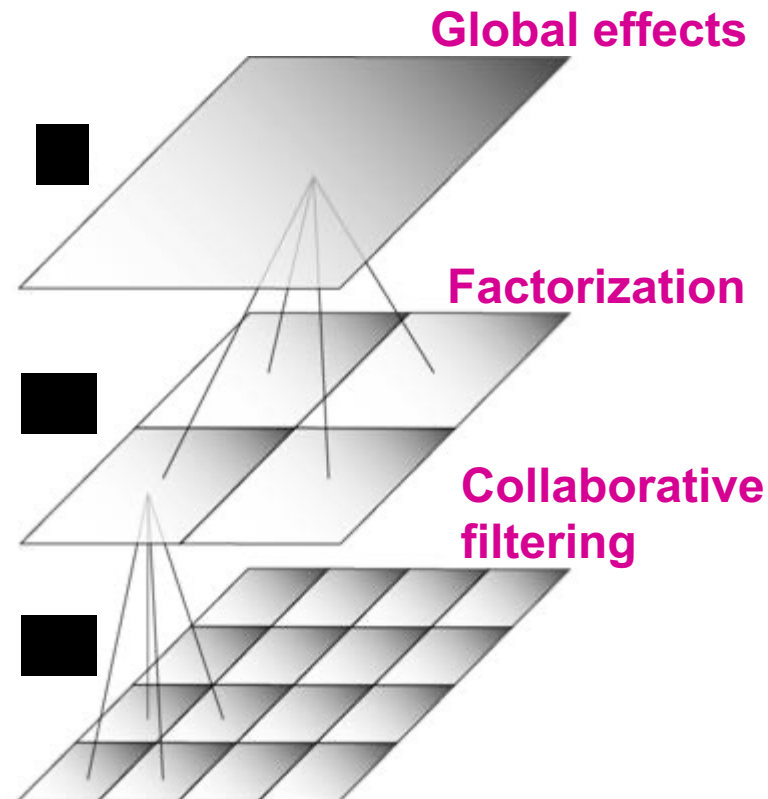
# BellKor Recommender System

- **The winner of the Netflix Challenge**

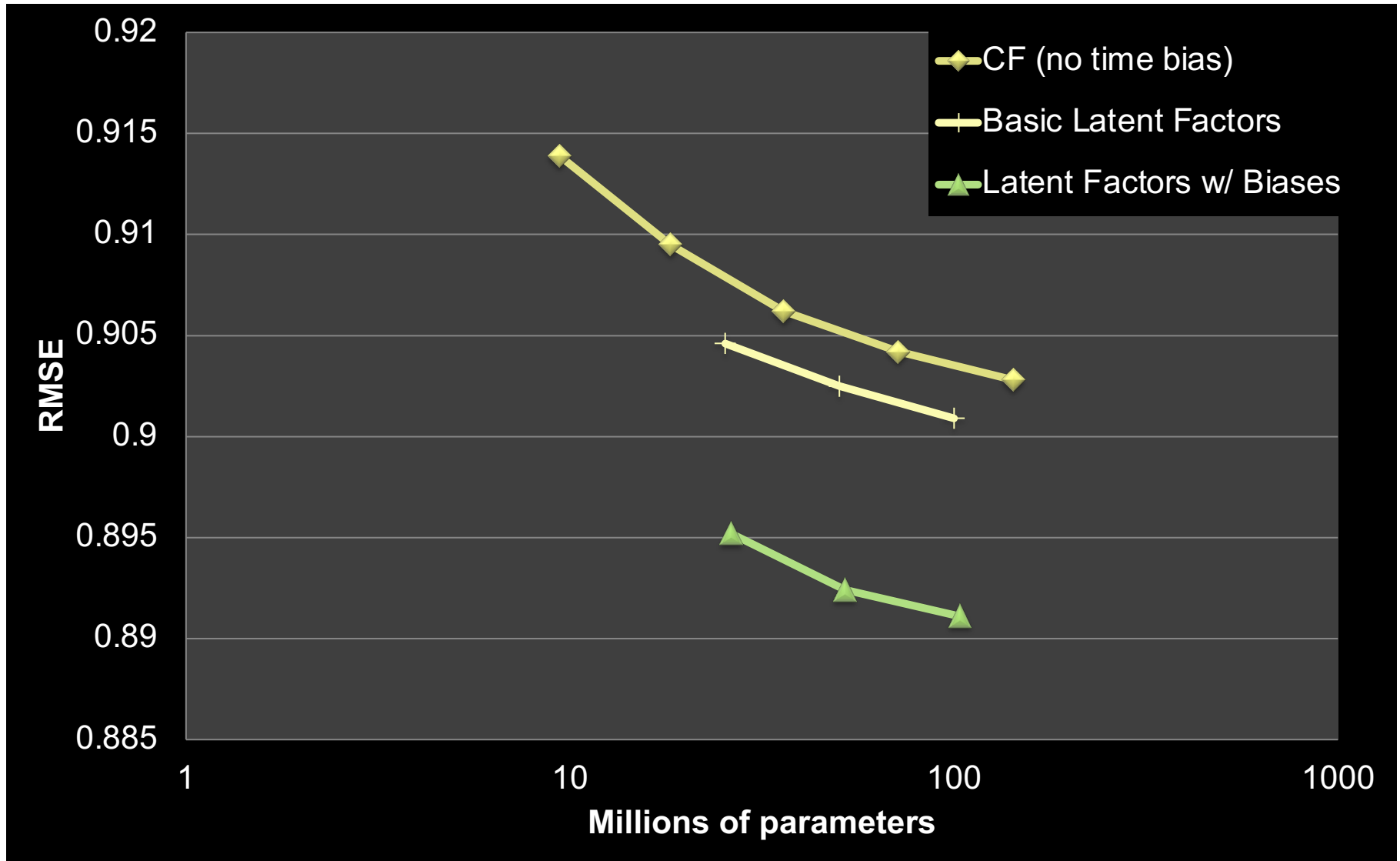
- **Multi-scale modeling of the data:**

Combine top level, “regional” modeling of the data, with a refined, local view:

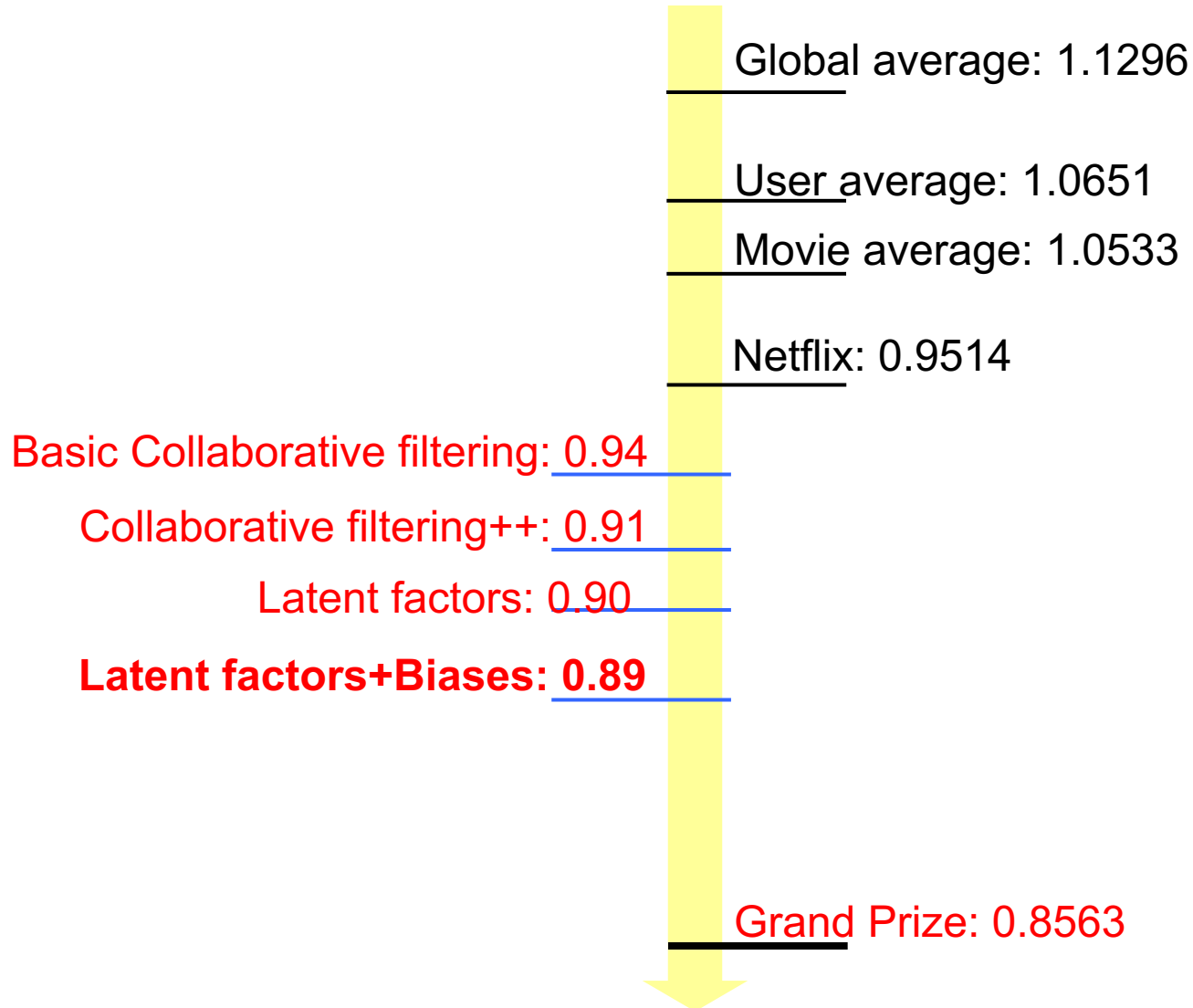
- **Global:**
  - Overall deviations of users/movies
- **Factorization:**
  - Addressing “regional” effects
- **Collaborative filtering:**
  - Extract local patterns



# Performance of Various Methods



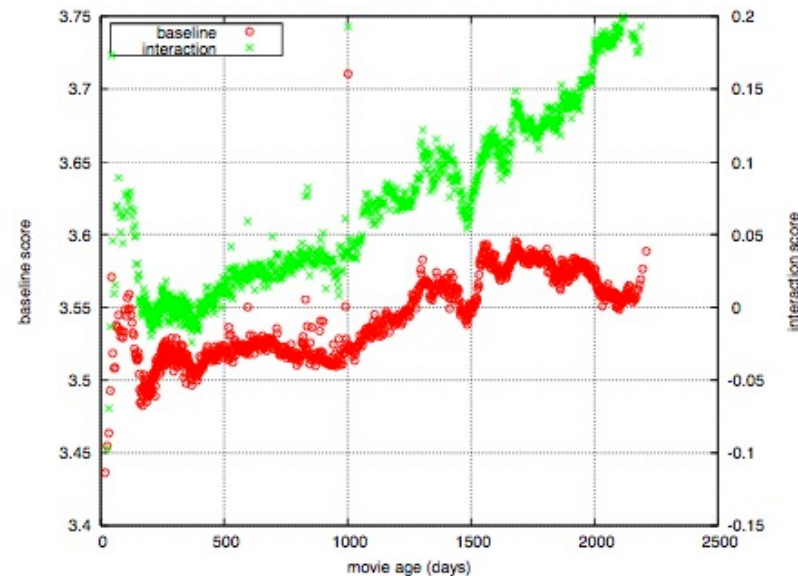
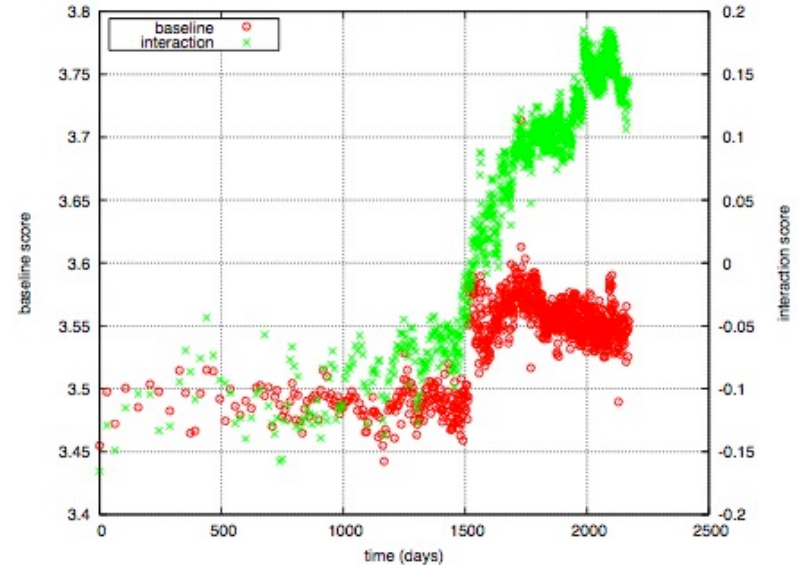
# Performance of Various Methods



# 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:**

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

- **Add time dependence to biases:**

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

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

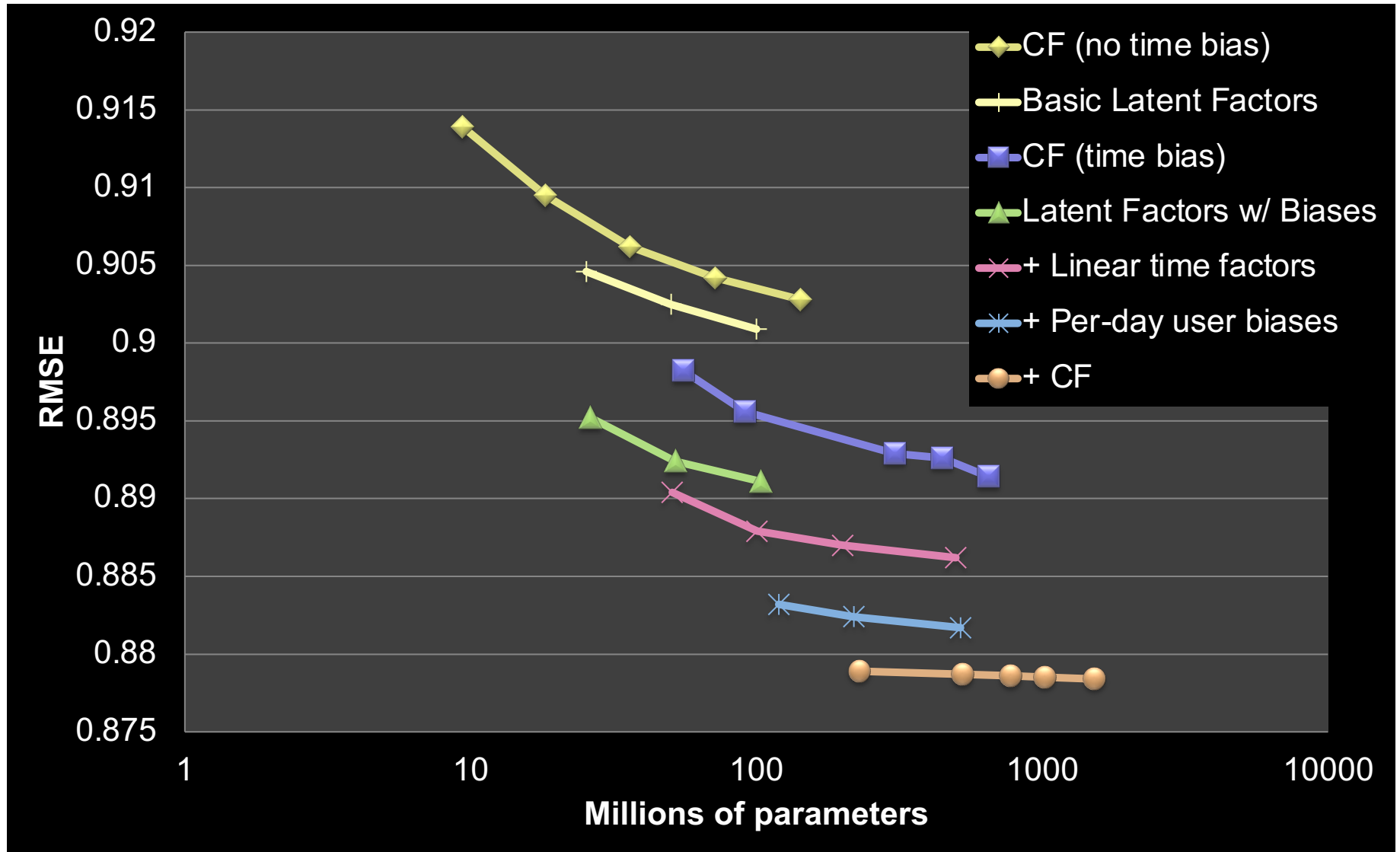
- **Add temporal dependence to factors**

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

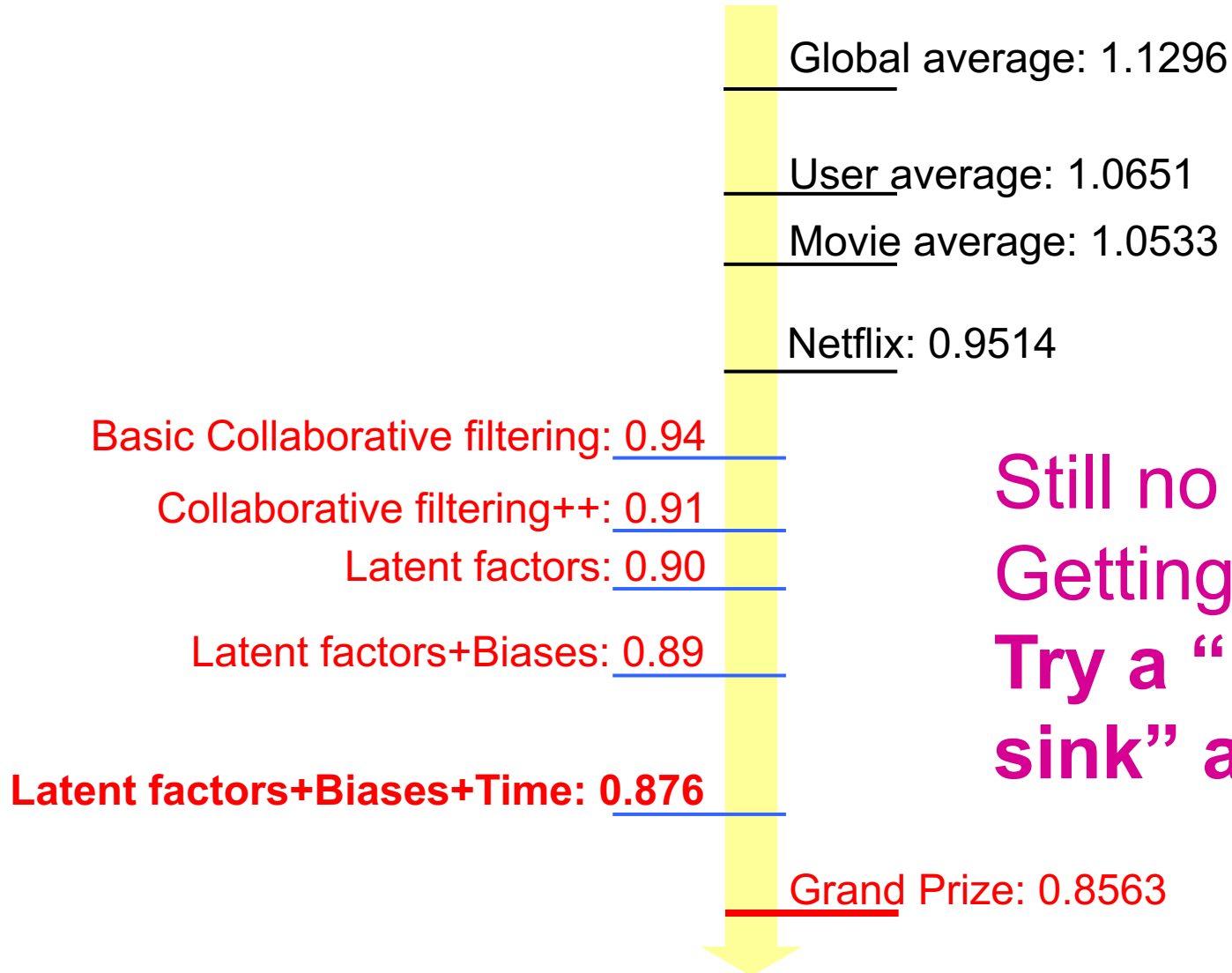
$$b_i(t) = b_i + b_{i, \text{Bin}(t)}$$

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

# Adding Temporal Effects



# Performance of Various Methods

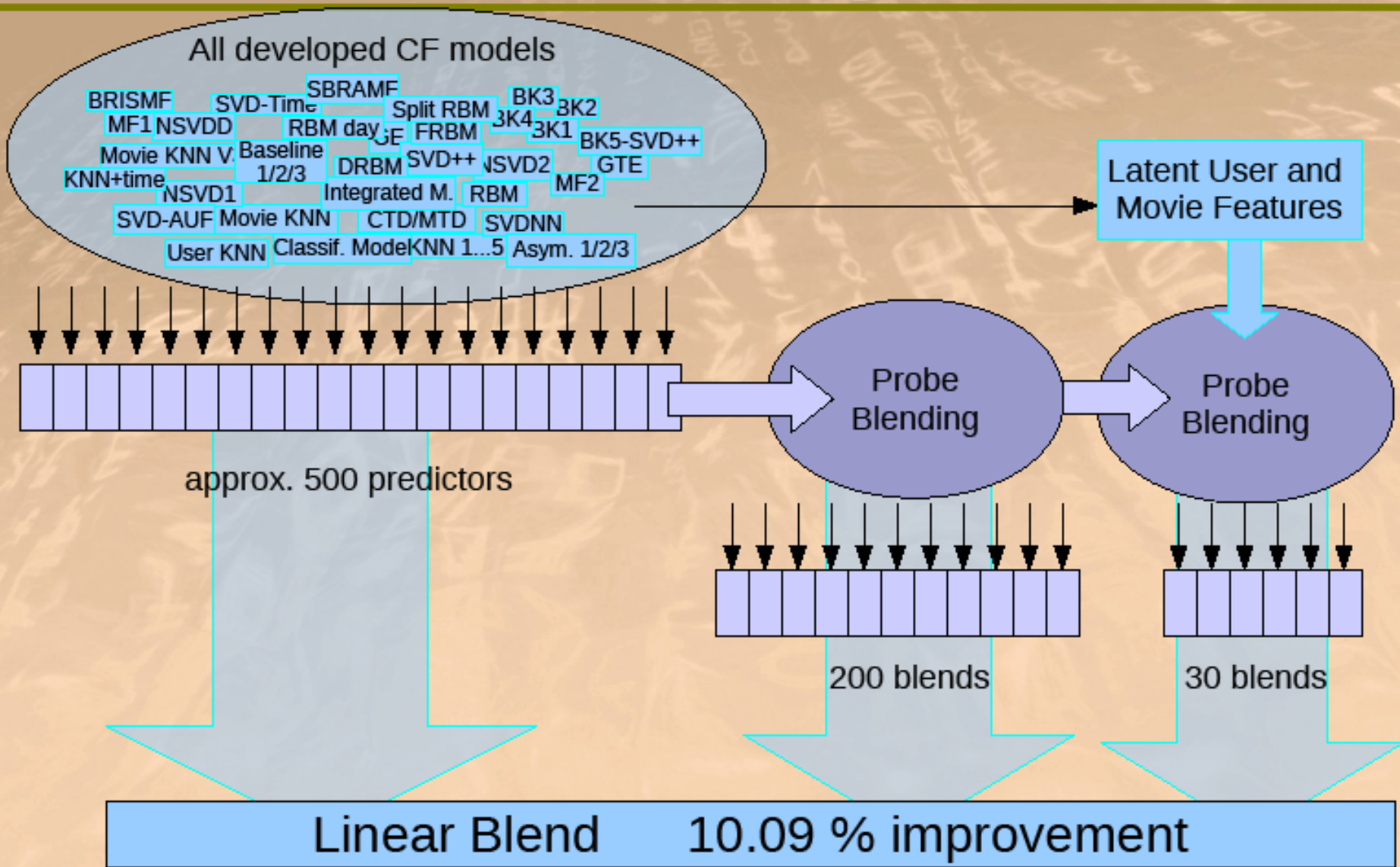


Still no prize! 😞  
Getting desperate.  
Try a “kitchen sink” approach!

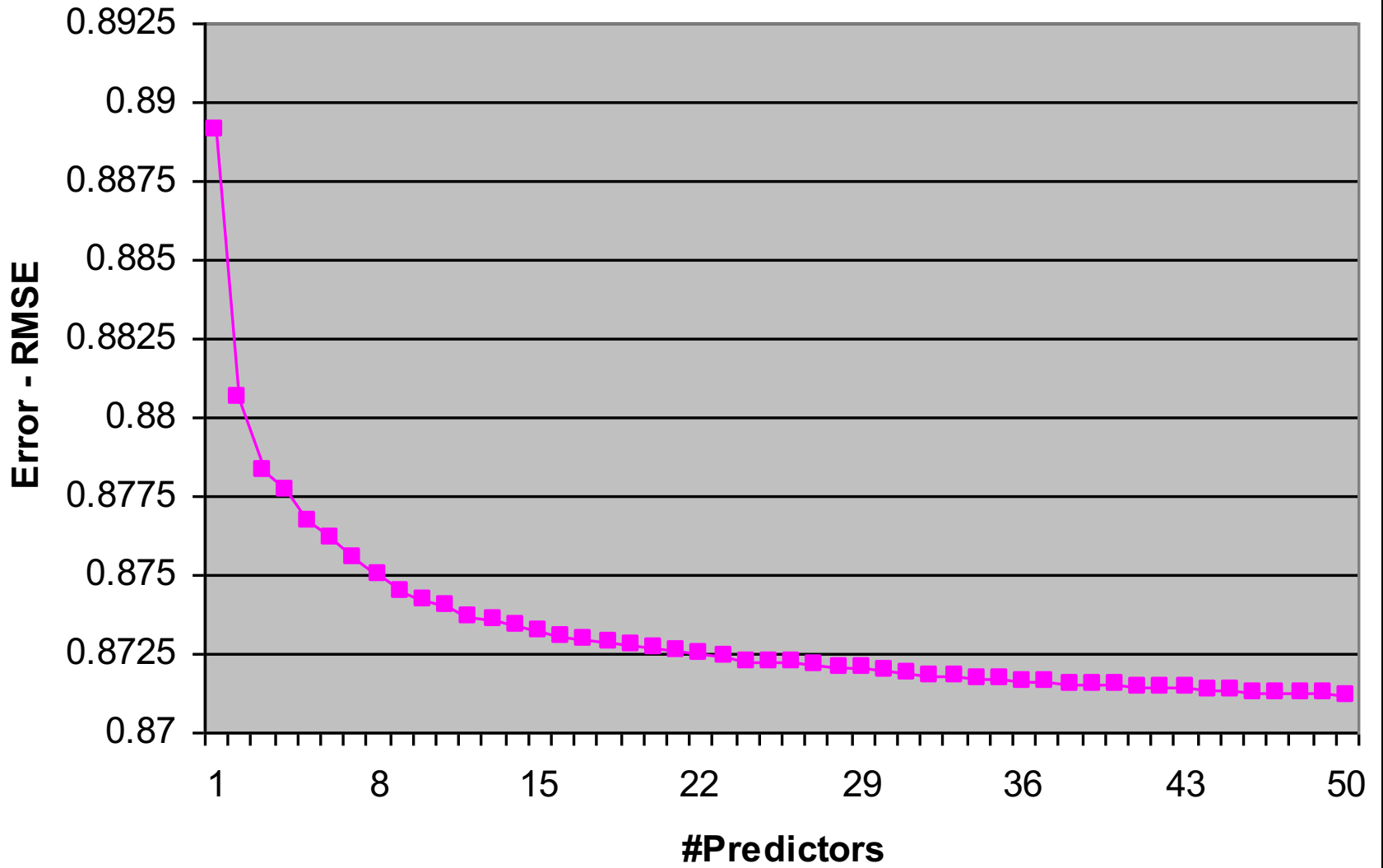


# The big picture

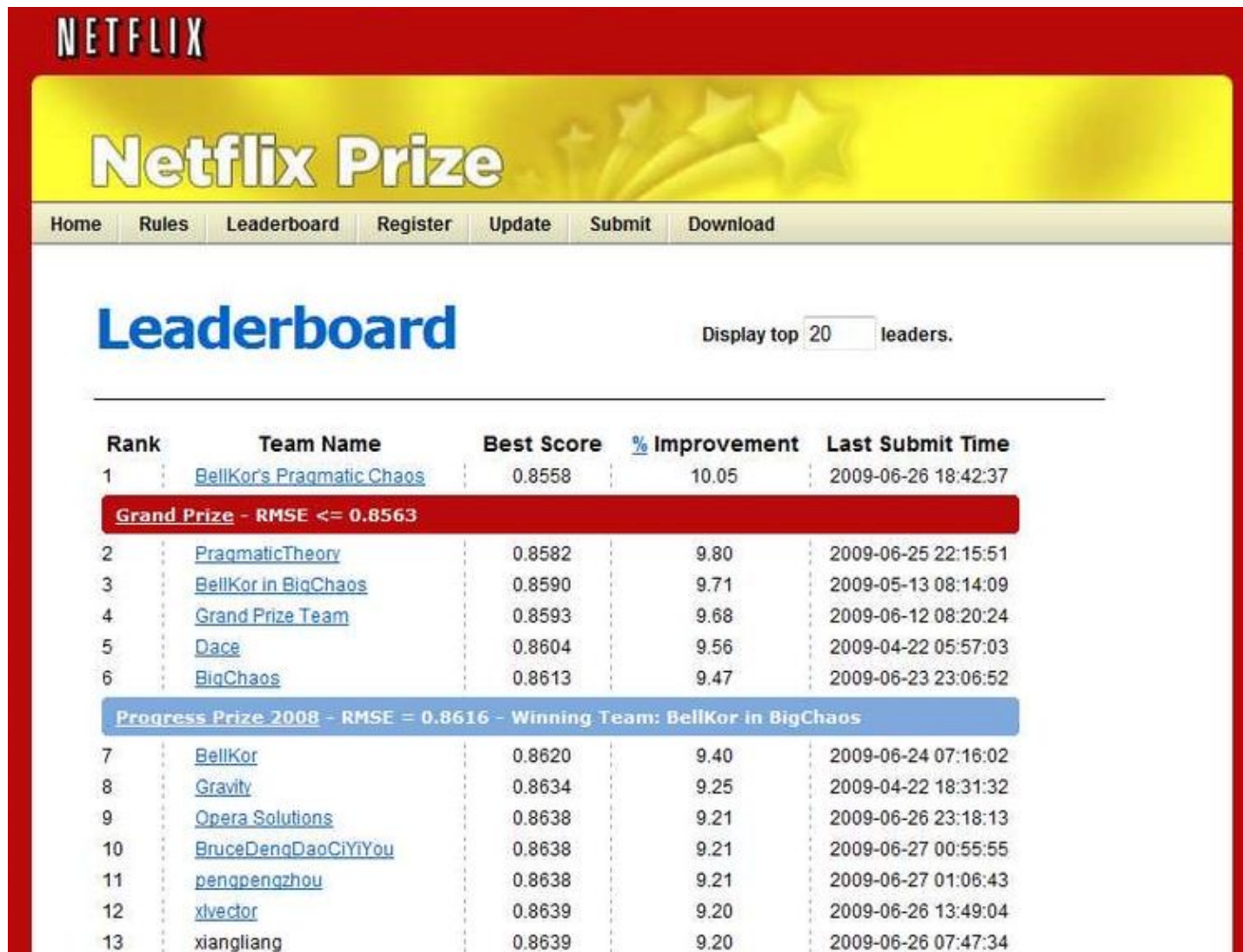
## Solution of BellKor's Pragmatic Chaos



## Effect of ensemble size



# Standing on June 26<sup>th</sup> 2009



NETFLIX

## Netflix Prize

Home Rules Leaderboard Register Update Submit Download

### Leaderboard

Display top  leaders.

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

June 26<sup>th</sup> 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 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 Prize



[Home](#)
[Rules](#)
[Leaderboard](#)
[Update](#)
[Download](#)

## Leaderboard

Showing Test Score. [Click here to show quiz score](#)

Display top  leaders.

Rank	Team Name	Best Test Score	% Improvement	Best Submit Time
<b>Grand Prize - RMSE = 0.8567 - Winning Team: BellKor's Pragmatic Chaos</b>				
1	<a href="#">BellKor's Pragmatic Chaos</a>	0.8567	10.06	2009-07-26 18:18:28
2	<a href="#">The Ensemble</a>	0.8567	10.06	2009-07-26 18:38:22
3	<a href="#">Grand Prize Team</a>	0.8582	9.89	2009-07-16 21:11:40
4	<a href="#">Opera Solutions and Vandelay United</a>	0.8588	9.84	2009-07-10 01:12:31
5	<a href="#">Vandelay Industries!</a>	0.8591	9.81	2009-07-10 00:32:20
6	<a href="#">PragmaticTheory</a>	0.8594	9.77	2009-06-24 12:06:56
7	<a href="#">BellKor in BigChaos</a>	0.8601	9.70	2009-05-13 08:14:09
8	<a href="#">Dace</a>	0.8612	9.59	2009-07-24 17:18:43
9	<a href="#">Feeds2</a>	0.8622	9.48	2009-07-12 13:11:51
10	<a href="#">BigChaos</a>	0.8623	9.47	2009-04-07 12:33:59
11	<a href="#">Opera Solutions</a>	0.8623	9.47	2009-07-24 00:34:07
12	<a href="#">BellKor</a>	0.8624	9.46	2009-07-26 17:19:11

### Progress Prize 2008 - RMSE = 0.8627 - Winning Team: BellKor in BigChaos

13	<a href="#">xiangliang</a>	0.8642	9.27	2009-07-15 14:53:22
14	<a href="#">Gravity</a>	0.8643	9.26	2009-04-22 18:31:32
15	<a href="#">Ces</a>	0.8651	9.18	2009-06-21 19:24:53
16	Invisible Ideas	0.8653	9.15	2009-07-15 15:53:04
17	<a href="#">Just a guy in a garage</a>	0.8662	9.06	2009-05-24 10:02:54
18	<a href="#">J Dennis Su</a>	0.8666	9.02	2009-03-07 17:16:17
19	<a href="#">Craig Carmichael</a>	0.8666	9.02	2009-07-25 16:00:54
20	<a href="#">acmehill</a>	0.8668	9.00	2009-03-21 16:20:50

### Progress Prize 2007 - RMSE = 0.8723 - Winning Team: KorBell



# Million \$ Awarded Sept 21<sup>st</sup> 2009



# 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/>