서울대학교
융합과학기술대학원
Seoul National University
Graduate School of Convergence
Science and Technology

Recommendation Systems

Recommendations

Search

Recommendations

Items

Products, web sites, blogs, news items, …

Examples:

amazon.com

PANDORA

StumbleUpon

del.icio.us

NETFLIX

movielens
helping you find the *right* movies

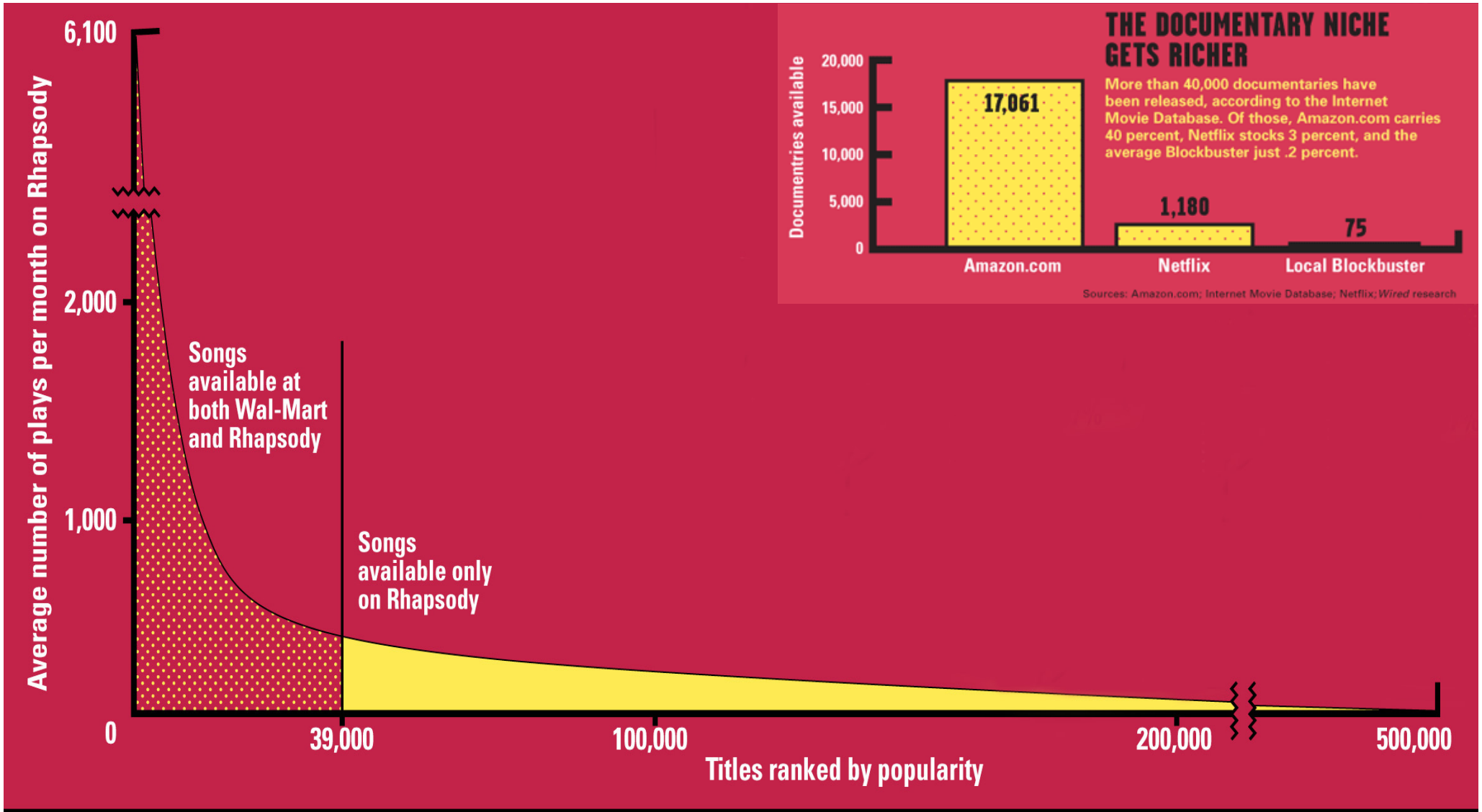
last·fm
the social music revolution

Google News

YouTube

XBOX LIVE

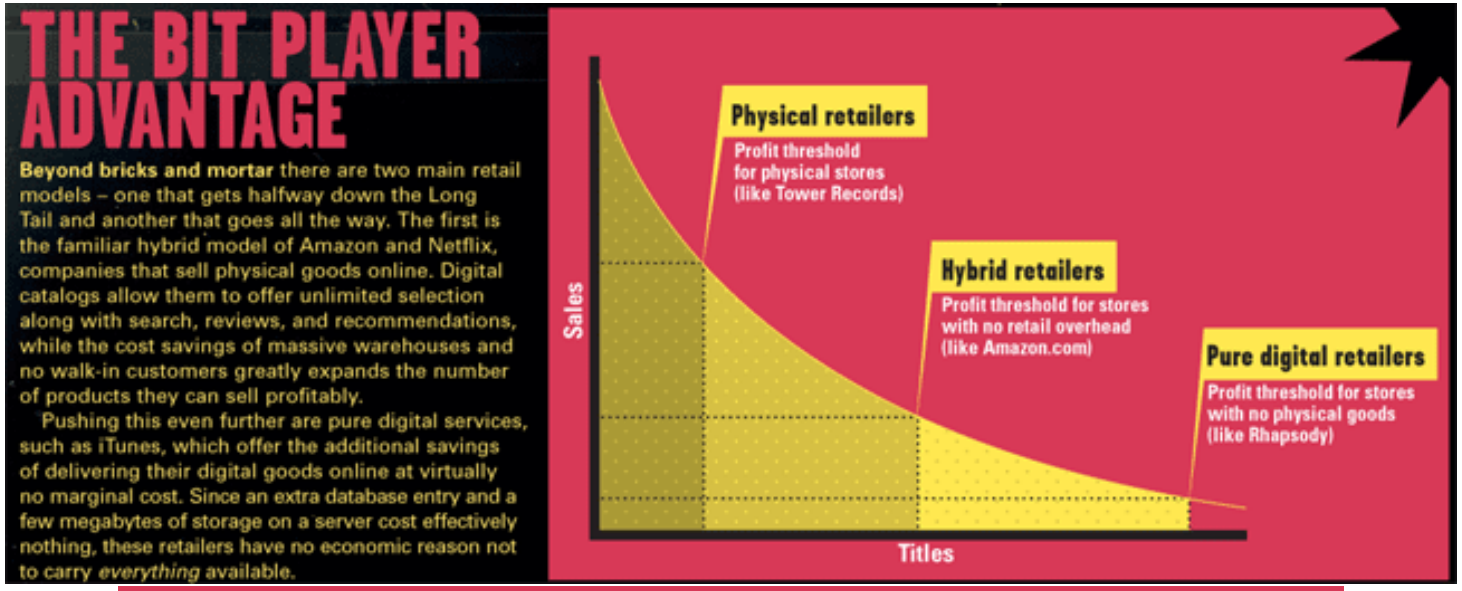
Sidenote: The Long Tail



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

Source: Chris Anderson (2004)

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 <u>what you don't like</u> but <u>what you like</u>

(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: Utility 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
  - 3) Latent factor based

Content-based Recommender Systems

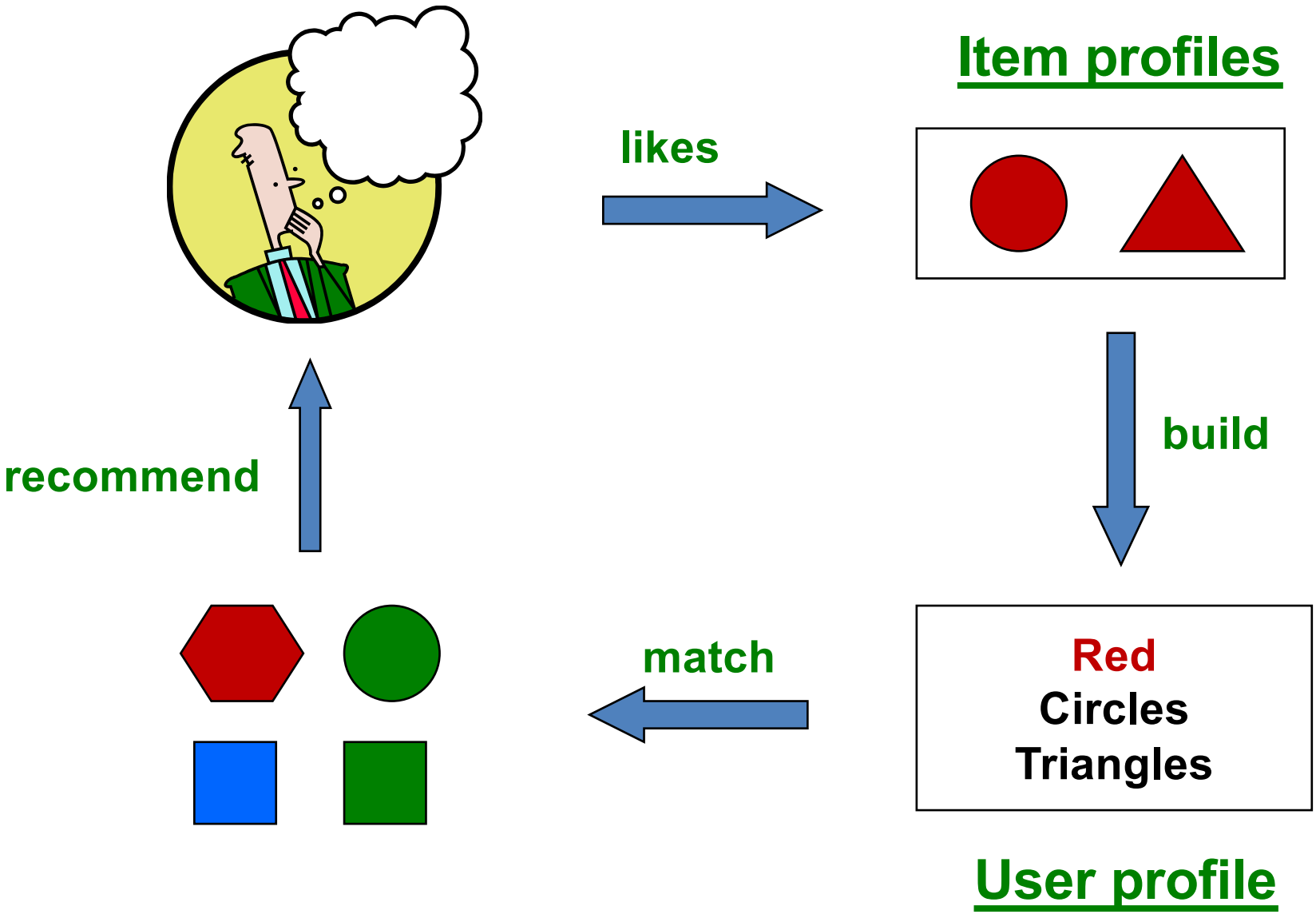
Content-based Recommendations

- 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

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 $\boldsymbol{x}$ and item profile $\boldsymbol{i}$, estimate
  - $u(\boldsymbol{x}, \boldsymbol{i}) = \cos(\boldsymbol{x}, \boldsymbol{i}) = \frac{\boldsymbol{x} \cdot \boldsymbol{i}}{||\boldsymbol{x}|| \cdot ||\boldsymbol{i}||}$

Example: Star Ratings

- Same example, 1–5 ratings
  - Actor A's movies: 3, 5 start
  - Actor B's movies: 1, 2, and 4 stars

- Normalize ratings by subtracting user's mean rating (3)
  - Actor A's normalized ratings = 0, +2
    - Profile weight = (0 + 2) / 2 = 1
  - Actor B's normalized ratings = −2, −1, +1
    - Profile weight = −2/3

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

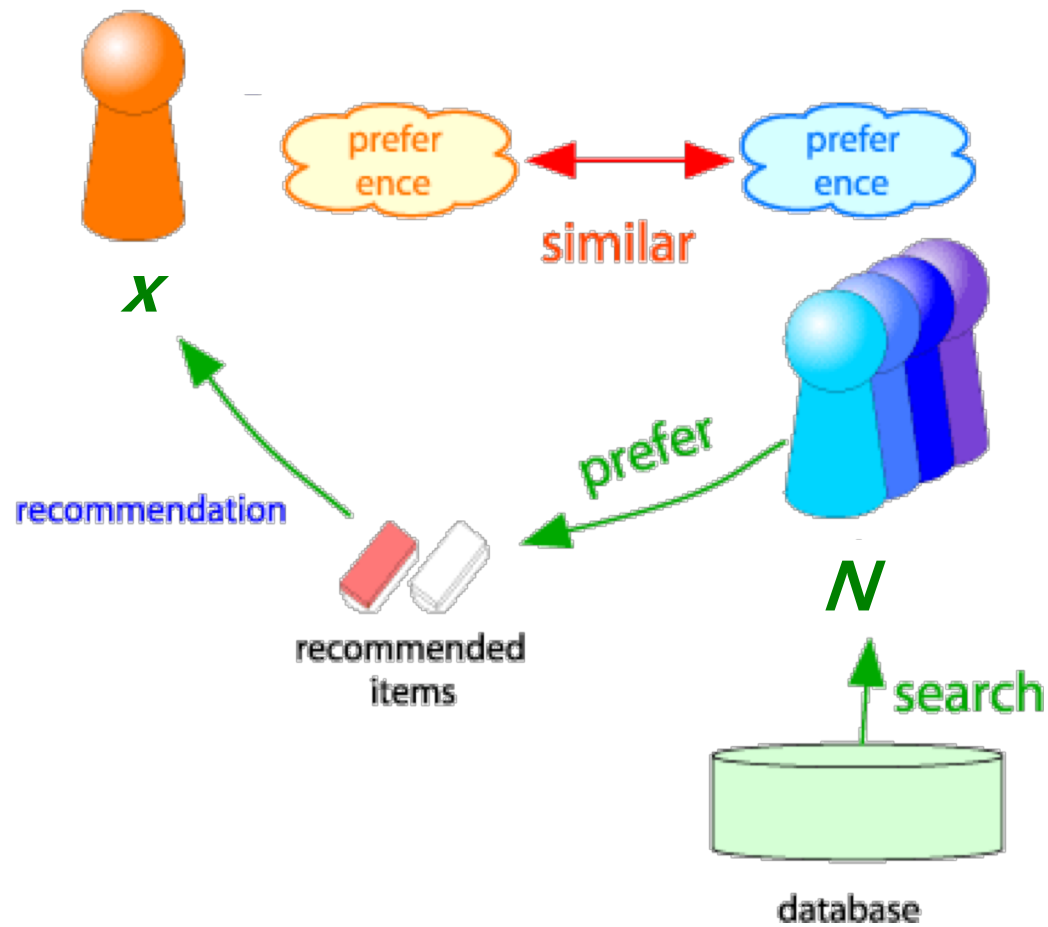
- – : Finding the appropriate features is hard
  - E.g., images, movies, music
- – : Recommendations for new users
  - How to build a user profile?
- – : Overspecialization
  - Never recommends items outside user's content profile
  - People might have multiple interests
  - Unable to exploit quality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

Collaborative Filtering

Harnessing quality judgments of other users

Collaborative Filtering

- Consider user x
- Find set N of other users whose ratings are "similar" to x's ratings
- Estimate x's ratings based on ratings of users in N



Finding "Similar" Users

- Let r_x be the vector of user x's ratings
- Jaccard similarity measure
  - Problem: Ignores the value of the rating
- Cosine similarity measure
  - $\text{sim}(x, y) = \cos(r_x, r_y) = \frac{r_x \cdot r_y}{||r_x|| \cdot ||r_y||}$
  - Problem: Treats missing ratings as "negative"
- Pearson correlation coefficient
  - S_{xy} = items rated by both users x and y

$$sim(x, y) = \frac{\sum_{s \in S_{xy}} (r_{xs} - \overline{r_x})(r_{ys} - \overline{r_y})}{\sqrt{\sum_{s \in S_{xy}} (r_{xs} - \overline{r_x})^2} \sqrt{\sum_{s \in S_{xy}} (r_{ys} - \overline{r_y})^2}}$$

r_x, r_y … avg. rating of **x**, **y**

r_x = [*, _, _, *, ***]
r_y = [*, _, **, **, _]

r_x, r_y as sets:
r_x = {1, 4, 5}
r_y = {1, 3, 4}

r_x, r_y as points:
r_x = {1, 0, 0, 1, 3}
r_y = {1, 0, 2, 2, 0}

Similarity Metric

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

- Intuitively we want: sim(A, B) > 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 (e.g 10/3 for A)

Missing rate → average
Tough raters vs. easy raters

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

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

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

Rating Predictions

From similarity metric to recommendations:

- 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*:
  - $r_{xi} = \frac{1}{k} \sum_{y \in N} r_{yi}$ 유사한 사용자의 평균 rating
  - $r_{xi} = \frac{\sum_{y \in N} s_{xy} \cdot r_{yi}}{\sum_{y \in N} s_{xy}}$ 유사한 사용자의 평균 rating이나 비슷한 사용자의 rating에 가중치를 둠
  - Other options?
- Many other tricks possible…

Shorthand:
$s_{xy} = sim(x, y)$

Item-Item 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 ratings for similar items
  - 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***

Similarity Matrix

	Similiarity Scores: Co-Occurrence Matrix					Daniel's Purchase History		Recommendation Score		
American Hustle	4	5	3	4	×	0	=	4x0 + 5x0 + 3x1 + 4x0	=	3 ▶ Recommend American Hustle
Dallas Buyers Club	5	5	0	0		0		5x0 + 5x0 + 0x1 + 0x0		0
The Monuments Men	3	0	3	0		1		3x0 + 0x0 + 3x1 + 0x0		3 ▶ Do not recommend since already purchased
12 Years a Slave	4	0	0	4		0		4x0 + 0x0 + 0x1 + 4x0		0
	American Hustle	Dallas Buyers Club	The Monuments Men	12 Years a Slave						

Item−Item CF (|N|=2)

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	

- unknown rating

- rating between 1 to 5

Item−Item CF (|N|=2)

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	

- estimate rating of movie **1** by user **5**

Item−Item CF (|N|=2)

users

movies

	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**

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

movies

	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

Compute similarity weights:

$s_{1,3}$=0.41, $s_{1,6}$=0.59

Item−Item CF (|N|=2)

users

movies

	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
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	

Predict by taking weighted average:

$$r_{ix} = \frac{\sum_{j \in N(i;x)} s_{ij} \cdot r_{jx}}{\sum s_{ij}}$$

$r_{1.5}$ **= (0.41*2 + 0.59*3) / (0.41+0.59) = 2.6**

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 <u>item-item</u> 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

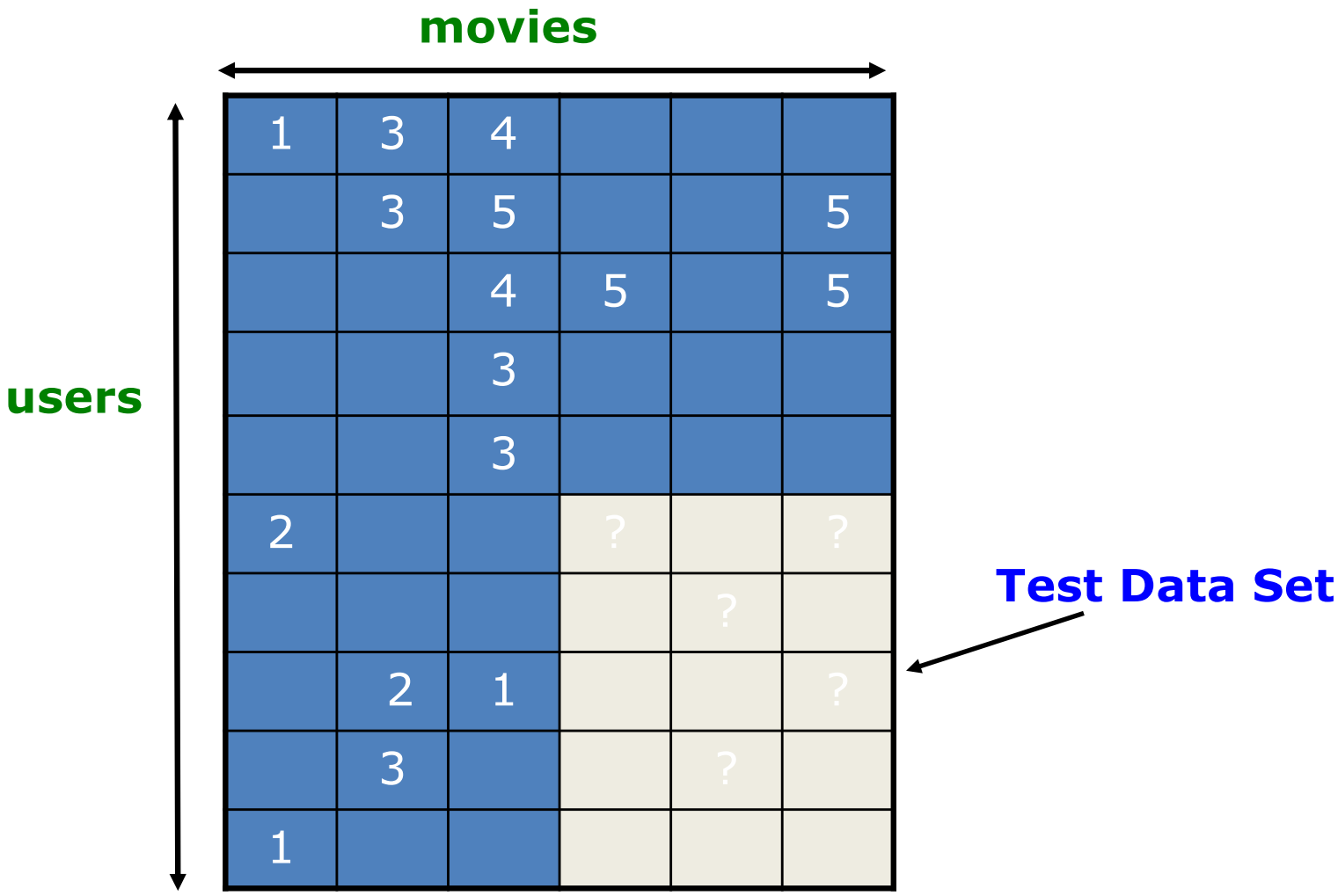
Evaluation

movies

users

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

Evaluation



Evaluating Predictions

- Compare predictions with known ratings
  - Root-mean-square error (RMSE)
    - $\sqrt{\sum_{xi}(r_{xi} - r^*_{xi})^2}$ / N where $\boldsymbol{r_{xi}}$ is predicted, $\boldsymbol{r^*_{xi}}$ is the true rating of $\boldsymbol{x}$ on $\boldsymbol{i}$
  - Precision at top k:
    - % of those in top k items e.g. top 10
  - Rank Correlation:
    - Spearman's *correlation* between system's and user's complete rankings

- Another approach: 0/1 model
  - Coverage:
    - Number of items/users for which system can make predictions
  - Precision:
    - Accuracy of predictions
  - Receiver operating characteristic (ROC)
    - Tradeoff curve between false positives and false negatives

Problems with Error Measures

- Narrow focus on accuracy sometimes misses the point
  - Prediction Diversity
    - 해리포터 1을 좋아했다면 해리포터 2를 추천 → 당연하고 다양성이 없음
    - 모든 추천이 비슷하다
  - Prediction Context
    - TV를 사면 한동안은 안산다
  - Order of predictions
    - 사는 물건에도 순서가 있다
- In practice, we care only to predict high ratings:
  - RMSE might penalize a method that does well for high ratings and badly for others

Collaborative Filtering: Complexity

- Expensive step is finding *k* most similar customers: $O(|X|)$
- Too expensive to do at runtime
  - Could pre-compute
- Naïve pre-computation takes time $O(k \cdot |X|)$
  - X … set of customers
- 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

The Netflix Prize

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

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

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

The Netflix Utility Matrix R

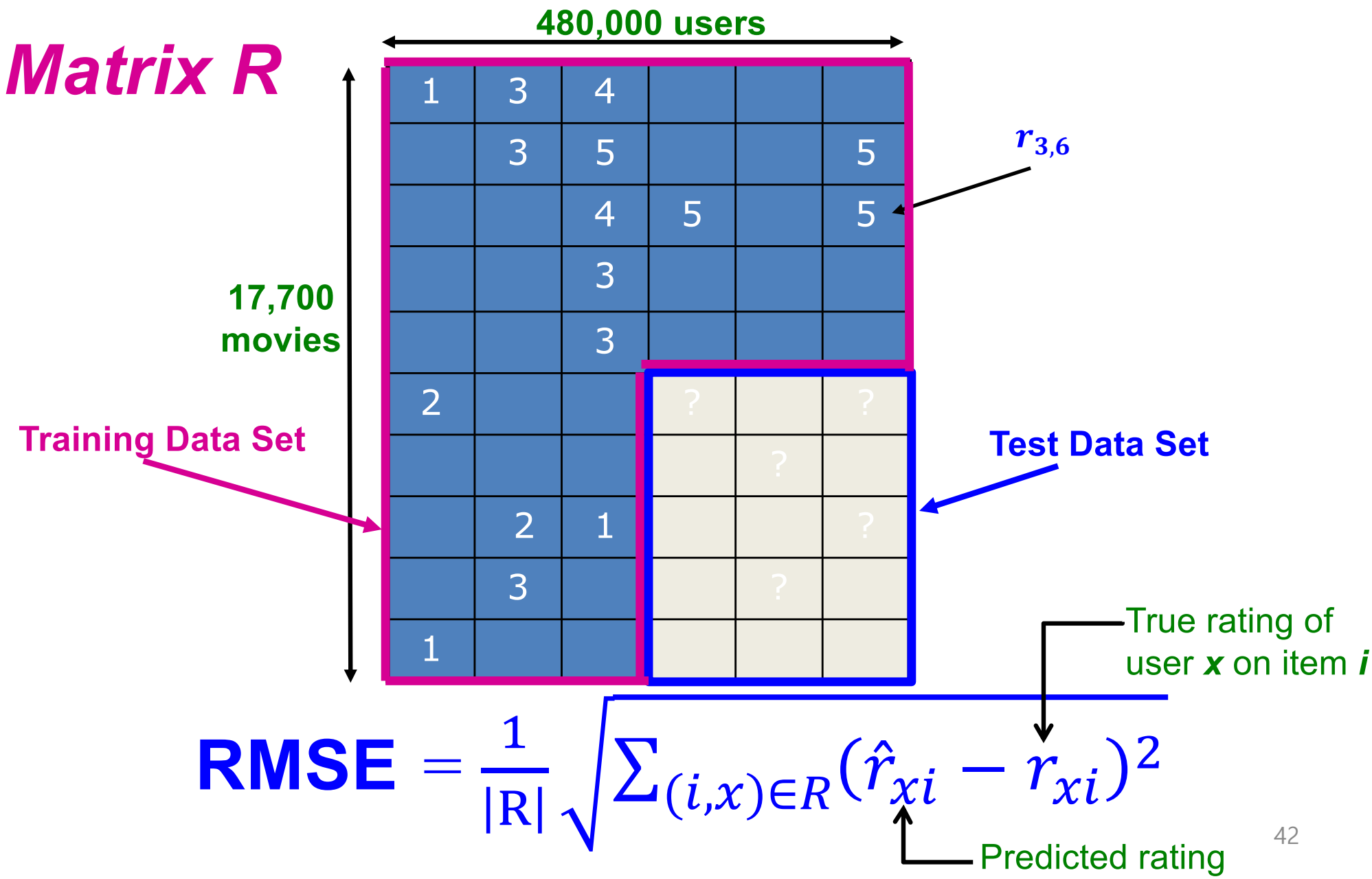
Matrix R

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

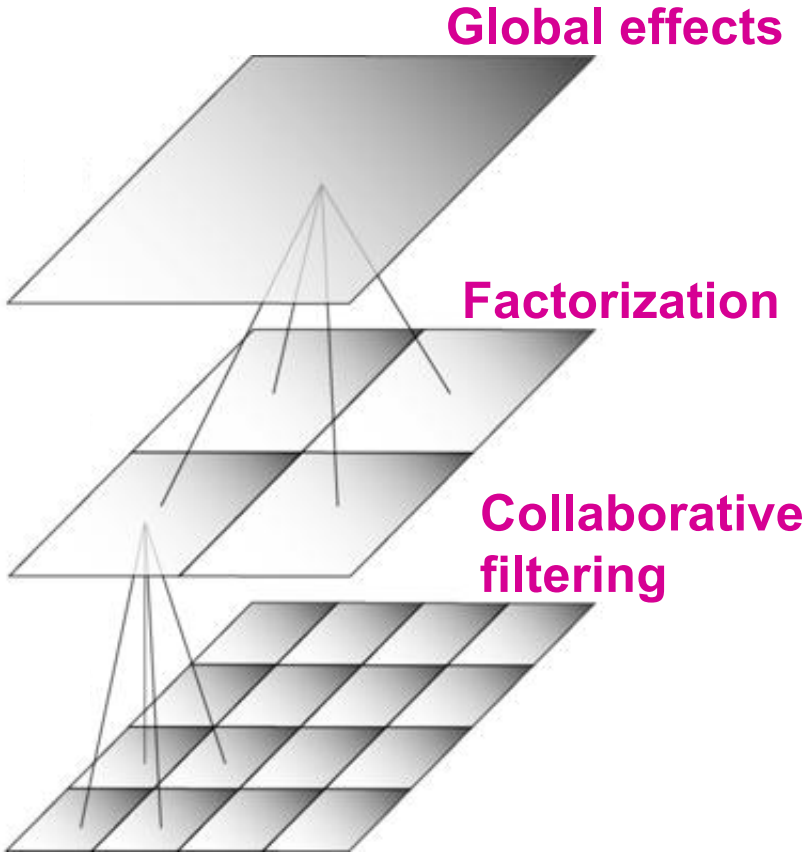


$$\textbf{RMSE} = \frac{1}{|R|}\sqrt{\sum_{(i,x)\in R}(\hat{r}_{xi} - r_{xi})^2}$$

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



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

Recap: Collaborative Filtering (CF)

- Earliest and most popular collaborative filtering method
- Derive unknown ratings from those of "**similar**" movies (item–item variant)
- Define similarity measure s_{ij} of items i and j
- Select k–nearest neighbors, compute the rating
  - $N(i; x)$: items <u>most similar to i</u> that <u>were rated by x</u>

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

s_{ij}… similarity of items i and j

r_{xj}…rating of user x on item j

$N(i;x)$… set of items similar to item i that were rated by x

Modeling Local & Global Effects

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

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

baseline estimate for r_{xi}

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

μ = overall mean rating

b_x = rating deviation of user x

= (*avg. rating of* user x) – μ

b_i = (*avg. rating of* movie i) – μ

Problems/Issues:

1) Similarity measures are "arbitrary"

2) Pairwise similarities neglect interdependencies among users

3) Taking a weighted average can be restricting

Solution:

Instead of s_{ij} use w_{ij} that we estimate directly from data

Idea: Interpolation Weights w_{ij}

- Use a weighted sum rather than weighted avg.:

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

- A few notes:
  - $\boldsymbol{N(i;x)}$ ⋯ set of movies rated by user $\boldsymbol{x}$ that are similar to movie $\boldsymbol{i}$
  - $\boldsymbol{w_{ij}}$ is the interpolation weight (some real number)
    - We allow: $\sum_{j \in N(i,x)} \boldsymbol{w_{ij}} \neq \mathbf{1}$
  - $\boldsymbol{w_{ij}}$ models interaction between pairs of movies (영화 i, j의 유사도)
    - It does not depend on user $\boldsymbol{x}$

Idea: Interpolation Weights w_{ij}

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

    $\sum_{(i,x)\in R}(\hat{r}_{xi} - r_{xi})^2$
  - Find w_{ij} that minimize **SSE** on **training data!**
    - Models relationships between item i and its neighbors j
  - w_{ij} can be learned/estimated based on x and all other users that rated i

Recommendations via Optimization

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

- **Goal:** Make good recommendations
  - Quantify goodness using **RMSE:**

    Lower RMSE ⇒ better recommendations

  - 사용자가 안 본 아이템에 대해서 RMSE를 구할 수 있는가?
  - Can't really do this!

  - **알고 있는 아이템(user, item) rating에 대해서 잘 적용되는 weight를 찾고 unknown ratings에 대해서도 잘 되기를 바래본다.**

Recommendations via Optimization

- Let's set values w such that they work well on known (user, item) ratings
- How to find such values w?
  - Idea: objective function을 구하고 optimization problem으로 해결한다

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

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

- Think of w as a vector of numbers

잠깐 다른 얘기 Convex Optimization

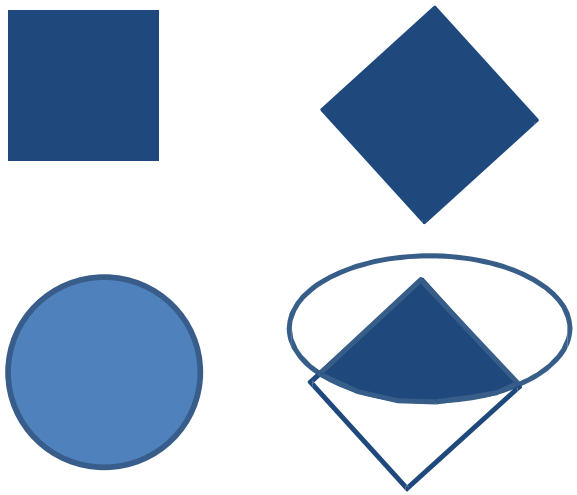
- Find minimum of a function subject to solution constraints
- Business/economics/ game theory
  - Resource allocation
  - Optimal planning and strategies
- Statistics and Machine Learning
  - All forms of regression and classification
  - Unsupervised learning
- Control theory
  - Keeping planes in the air!

Convex Sets

- A set C is convex if $\forall x, y \in C$ and $\forall a \in [0,1]$

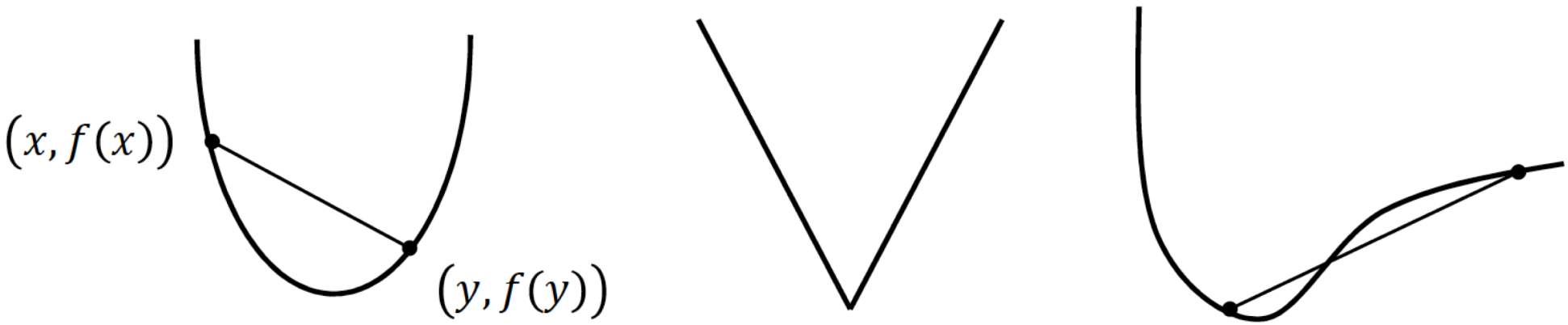
$$ax + (1 - a)\, y \in C$$

  - Line segment between points in C also lies in C
- Example
  - Intersection of halfspaces
  - L_p balls
  - Intersection of convex sets



Convex Functions

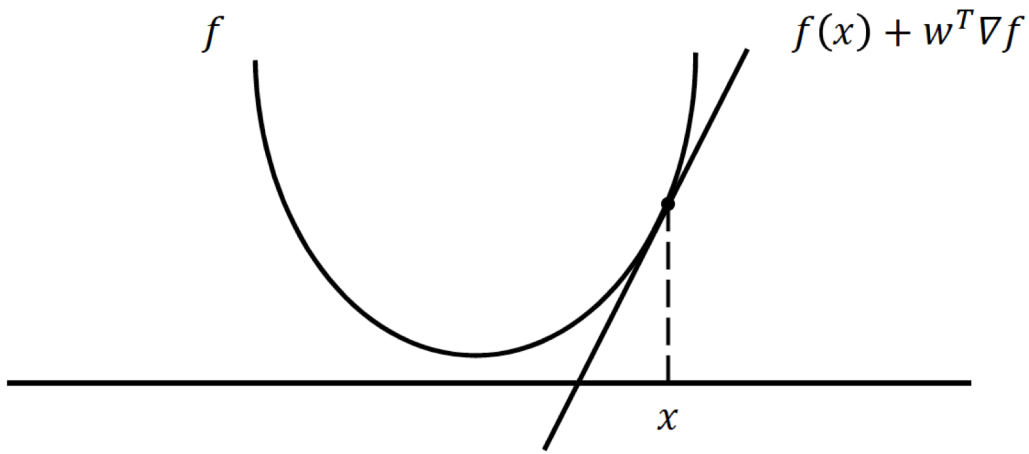
- A = [0..1] 일때 f(ax + (1−a)y) <= a *f(x) + (1−a)* f(y)
- Graph of f upper bounded by line segment between points on graph
- 임의의 두점간의 아래에 그래프가 놓이게 된다



Gradient 기울기

- Differentiable convex f
- Gradient ∇f at x gives linear approximation

$$\nabla f = \left[\frac{\delta f}{\delta x_1} \quad \cdots \quad \frac{\delta f}{\delta x_d}\right]^T$$

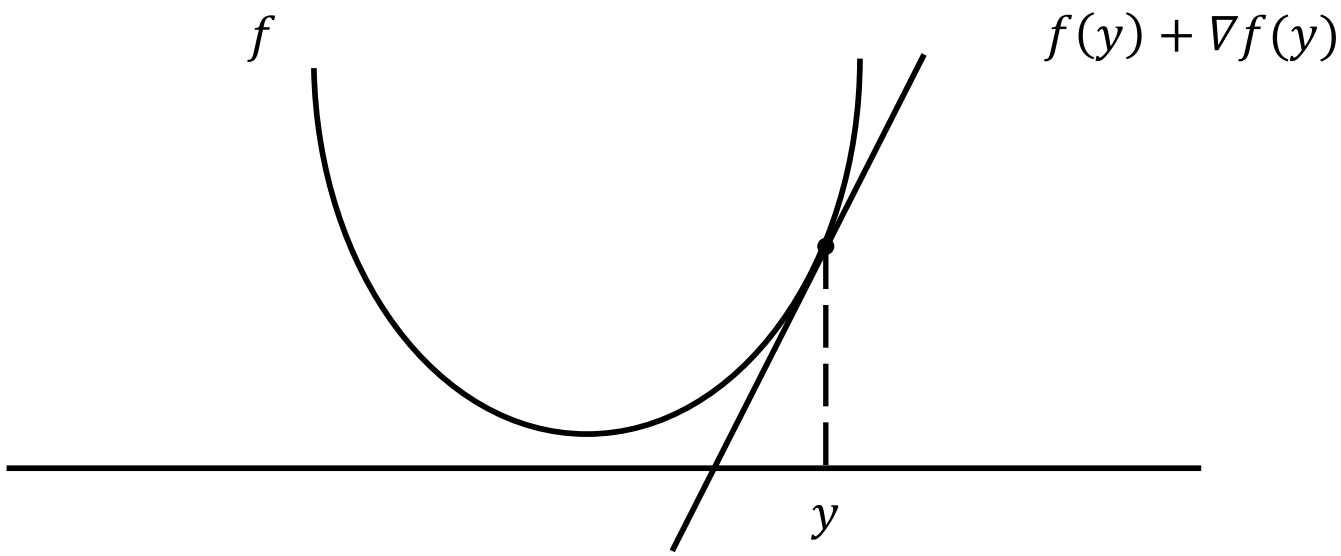


Gradient Descent

- To minimize f move down gradient
  - But not too far!
  - Optimum when Vf = 0
- Given f, learning rate a, starting point x_0
  - $x = x_0$
  - Do until $\nabla f = 0$
  - $x = x - a \nabla f$

최소값을 찾는 방법

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



Interpolation Weights

- We have the optimization problem, now what?
- Gradient decent:

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

  - Iterate until convergence: $\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \boldsymbol{\nabla_w J}$
  - where $\boldsymbol{\nabla_w J}$ is the gradient (derivative evaluated on data)

$\boldsymbol{\eta}$ … learning rate

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

for $\boldsymbol{j} \in \{\boldsymbol{N}(\boldsymbol{i};\boldsymbol{x}), \forall \boldsymbol{i}, \forall \boldsymbol{x}\}$ else $\frac{\partial J(w)}{\partial w_{ij}} = \boldsymbol{0}$

- .For every movie $\boldsymbol{j} \in \boldsymbol{N}(\boldsymbol{i};\boldsymbol{x})$, we compute $\frac{\partial \boldsymbol{J}(\boldsymbol{w})}{\partial \boldsymbol{w_{ij}}}$

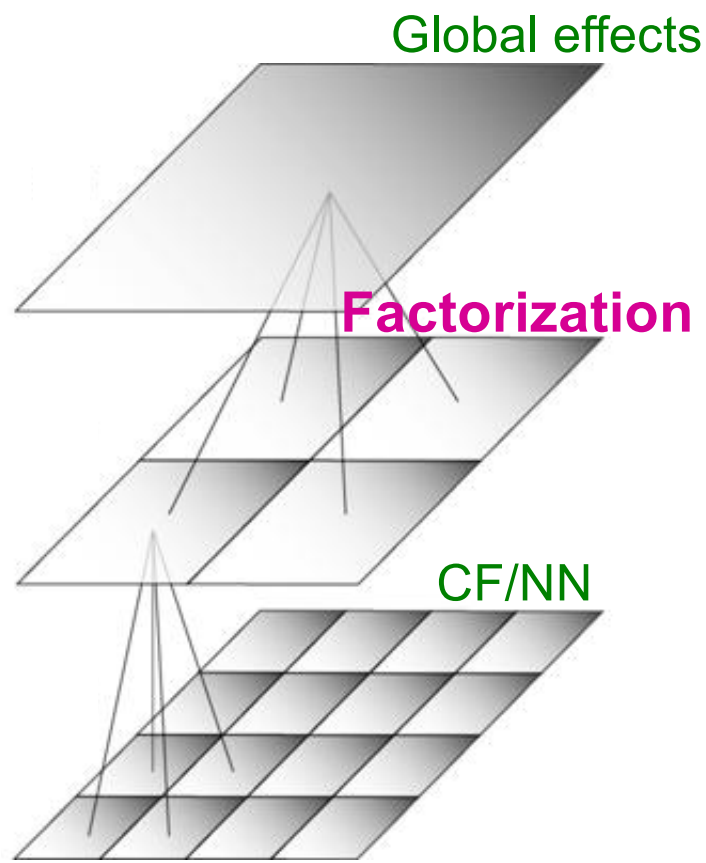
while $|w_{new} - w_{old}| > \varepsilon$**:**

$w_{old} = w_{new}$

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

Interpolation Weights

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



Global average: 1.1296

User average: 1.0651

Movie average: 1.0533

Netflix: 0.9514

Basic Collaborative filtering: 0.94

CF+Biases+learned weights: 0.91

Grand Prize: 0.8563

Latent Factor Models (e.g. SVD)

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

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

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	

R

≈

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
-.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

P^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*?

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

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

users

items

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

≈

items

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

factors

Q

•

users

factors

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

P^T

Ratings as Products of Factors

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

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

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

q_i = row i of Q

p_x = column x of P^T

users

items

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

≈

items

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

factors

Q

•

users

factors

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

P^T

Ratings as Products of Factors

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

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

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

users

items

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

≈

items

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

f factors

Q

•

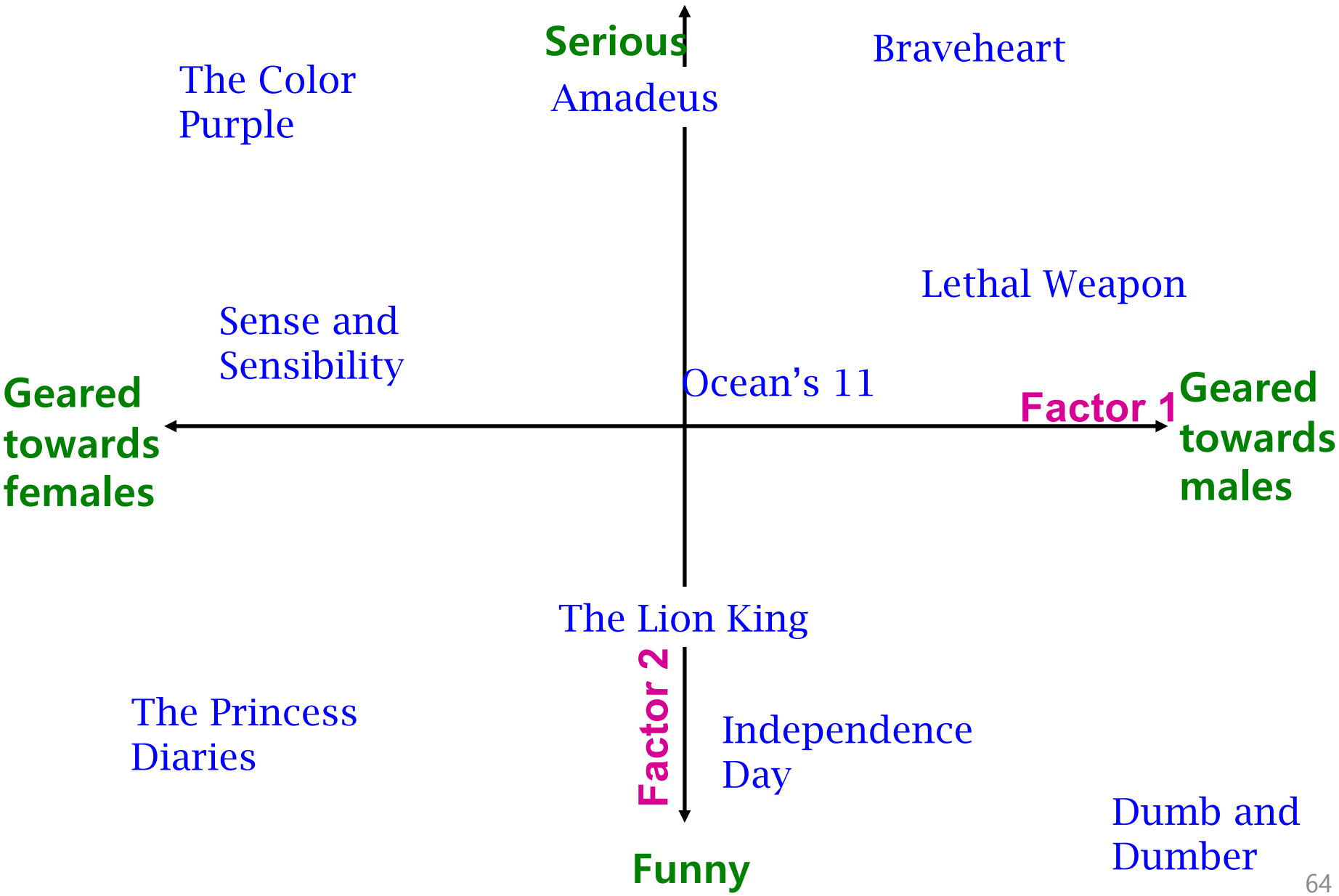
users

f factors

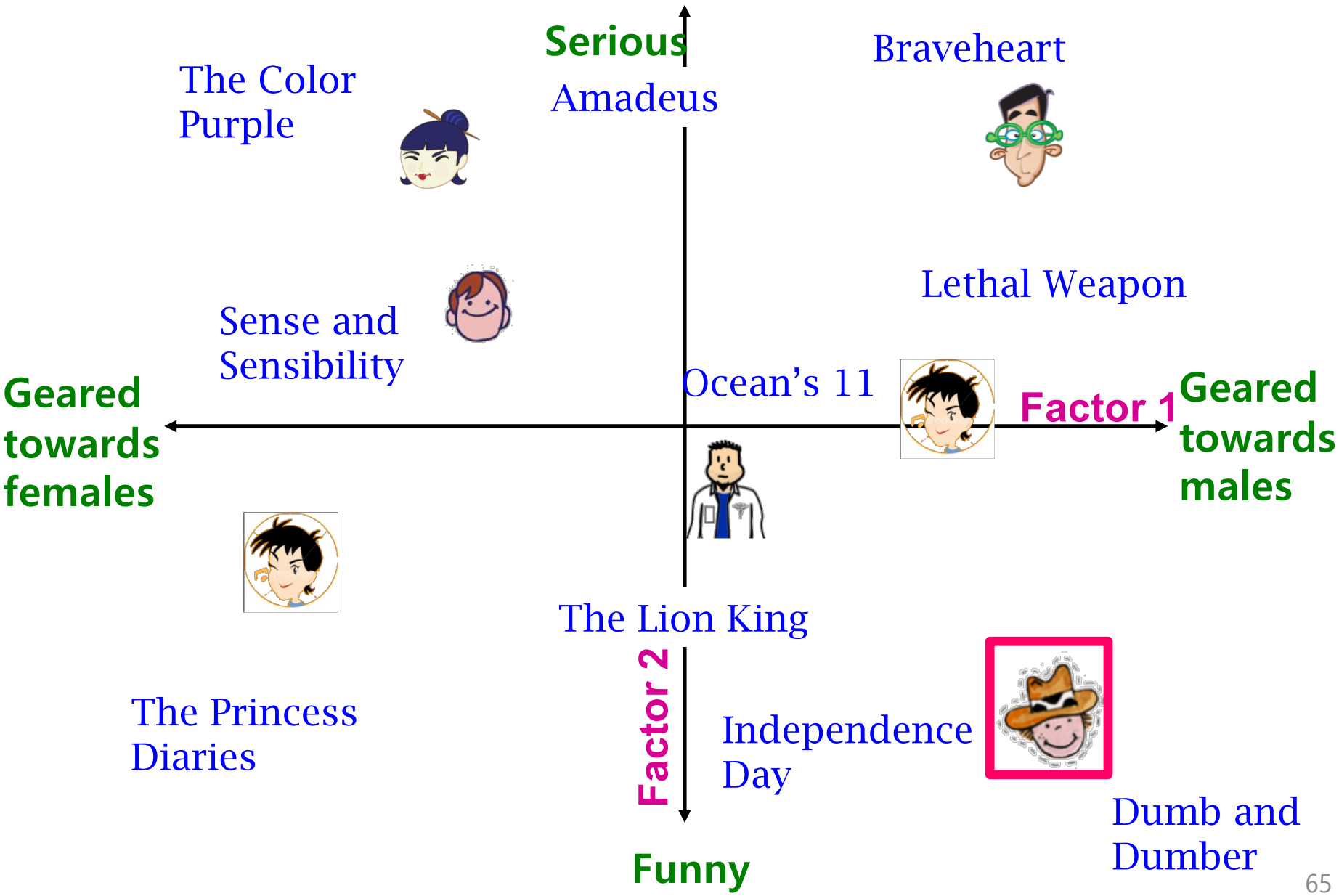
1.1	-.2	.3	.5	-2	-.5	.8	-.4	.3	1.4	2.4	-.9
-.8	.7	.5	1.4	.3	-1	1.4	2.9	-.7	1.2	-.1	1.3
2.1	-.4	.6	1.7	2.4	.9	-.3	.4	.8	.7	-.6	.1

P^T

Latent Factor Models

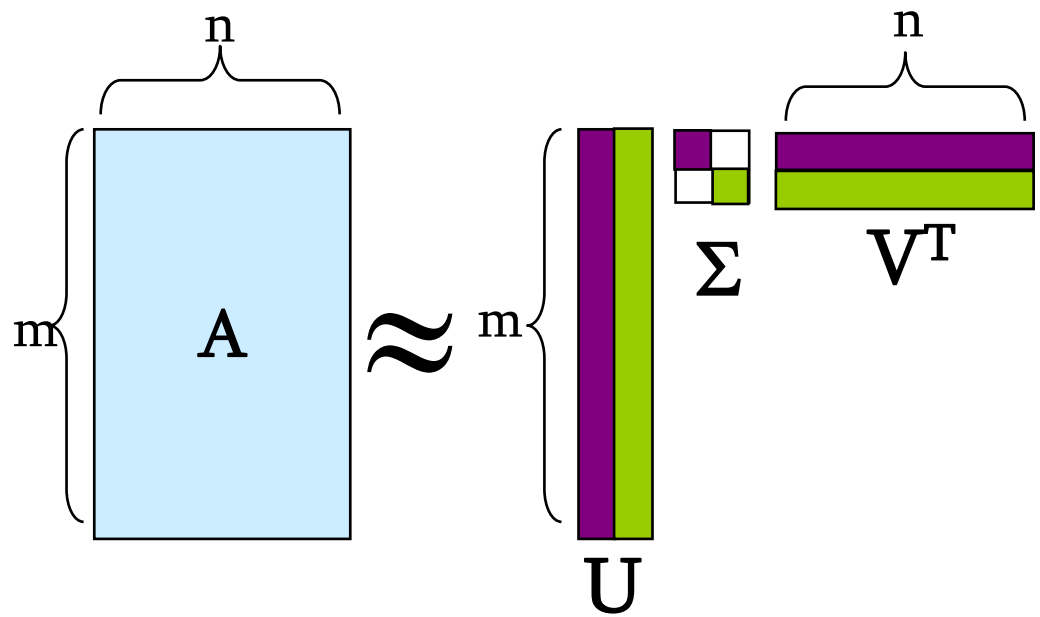


Latent Factor Models



Recap: SVD

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



- So in our case:

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

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

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

SVD: More good stuff

- We already know that SVD gives minimum reconstruction error (Sum of Squared Errors):

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

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

Latent Factor Models

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

P^T

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

Finding the Latent Factors

Latent Factor Models

- Our goal is to find P and Q such tat:

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

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

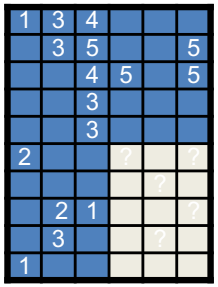
factors

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

P^T

Back to Our Problem

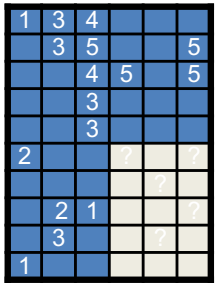
- Want to minimize SSE for unseen test data
- Idea: Minimize SSE on <u>training</u> data
  - Want large k (# of factors) to capture all the signals
  - But, SSE on <u>test</u> data begins to rise



- This is a classical example of **overfitting**:
  - With too much freedom (too many free parameters) the model starts fitting noise
  - That is it fits too well the training data and thus **not generalizing** well to unseen test data

Dealing with Missing Entries

- To solve overfitting we introduce regularization:
  - Allow rich model where there are sufficient data
  - 데이터가 많은 부분을 강조
  - 데이터가 없는 부분은 aggressively shrink

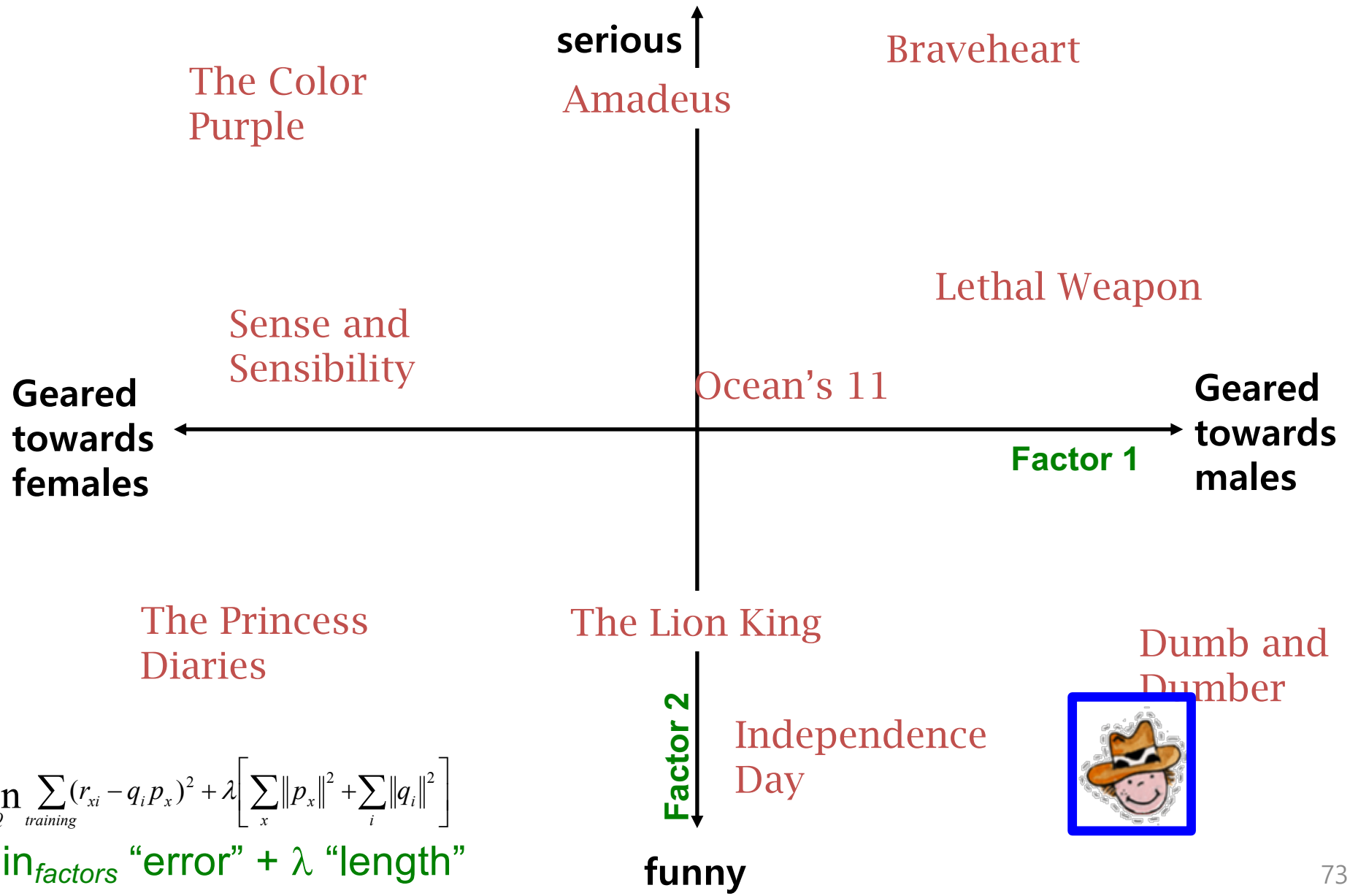


$$\min_{P,Q} \underbrace{\sum_{training} (r_{xi} - q_i p_x)^2}_{\text{“error”}} + \underbrace{\left[\lambda_1 \sum_x \|p_x\|^2 + \lambda_2 \sum_i \|q_i\|^2 \right]}_{\text{“length”}}$$

λ_1, λ_2 … user set regularization parameters

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

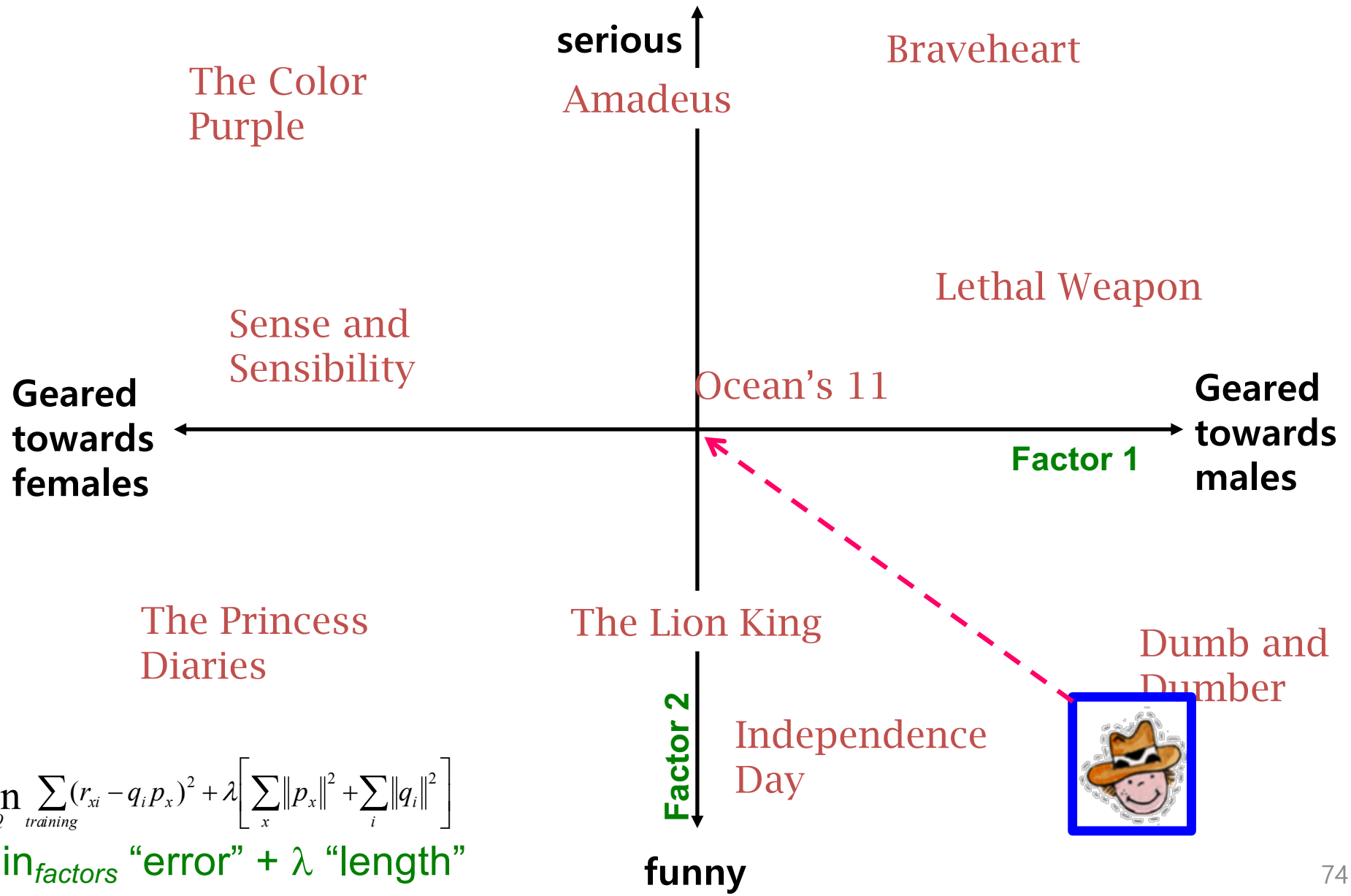
The Effect of Regularization



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

$\min_{factors}$ "error" + λ "length"

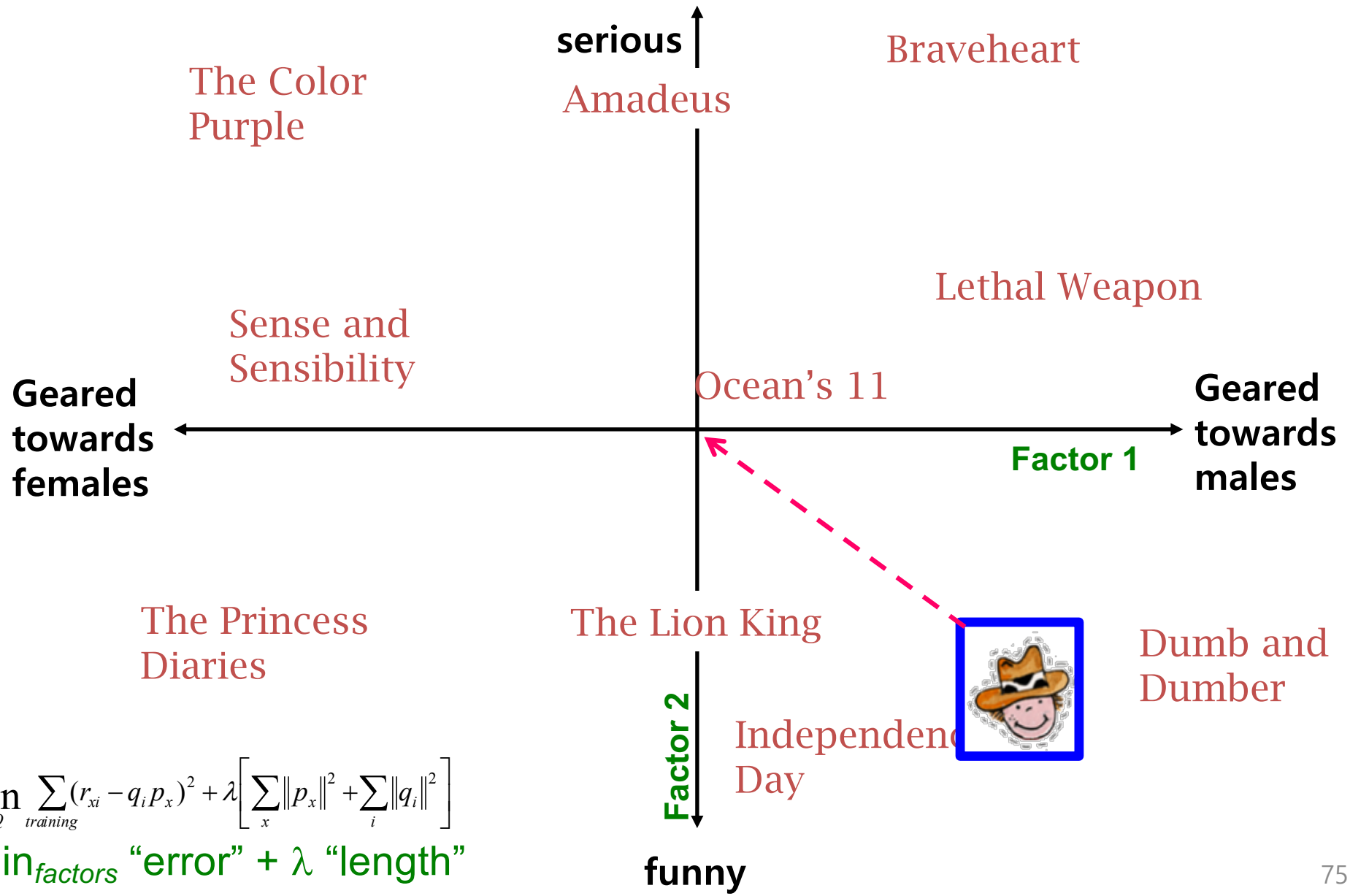
The Effect of Regularization



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

$\min_{factors}$ "error" + λ "length"

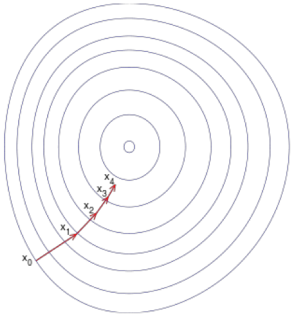
The Effect of Regularization



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

$\min_{factors}$ "error" + λ "length"

Stochastic Gradient Descent



- Want to find matrices P and Q:

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

- Gradient decent:
  - Initialize P and Q (using SVD, pretend missing ratings are 0)
  - Do gradient descent:
    - $P \leftarrow P - \eta \cdot \nabla P$
    - $Q \leftarrow Q - \eta \cdot \nabla Q$
    - where ∇Q is gradient/derivative of matrix Q:

      $\nabla Q = [\nabla q_{if}]$ and $\nabla q_{if} = \sum_{x,i} -2(r_{xi} - q_i p_x) p_{xf} + 2\lambda_2 q_{if}$
      - Here $\boldsymbol{q_{if}}$ is entry f of row q_i of matrix Q
      - r_{xj}…rating of user x on item j
      - *P movie – factor, Q: user - factor*

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

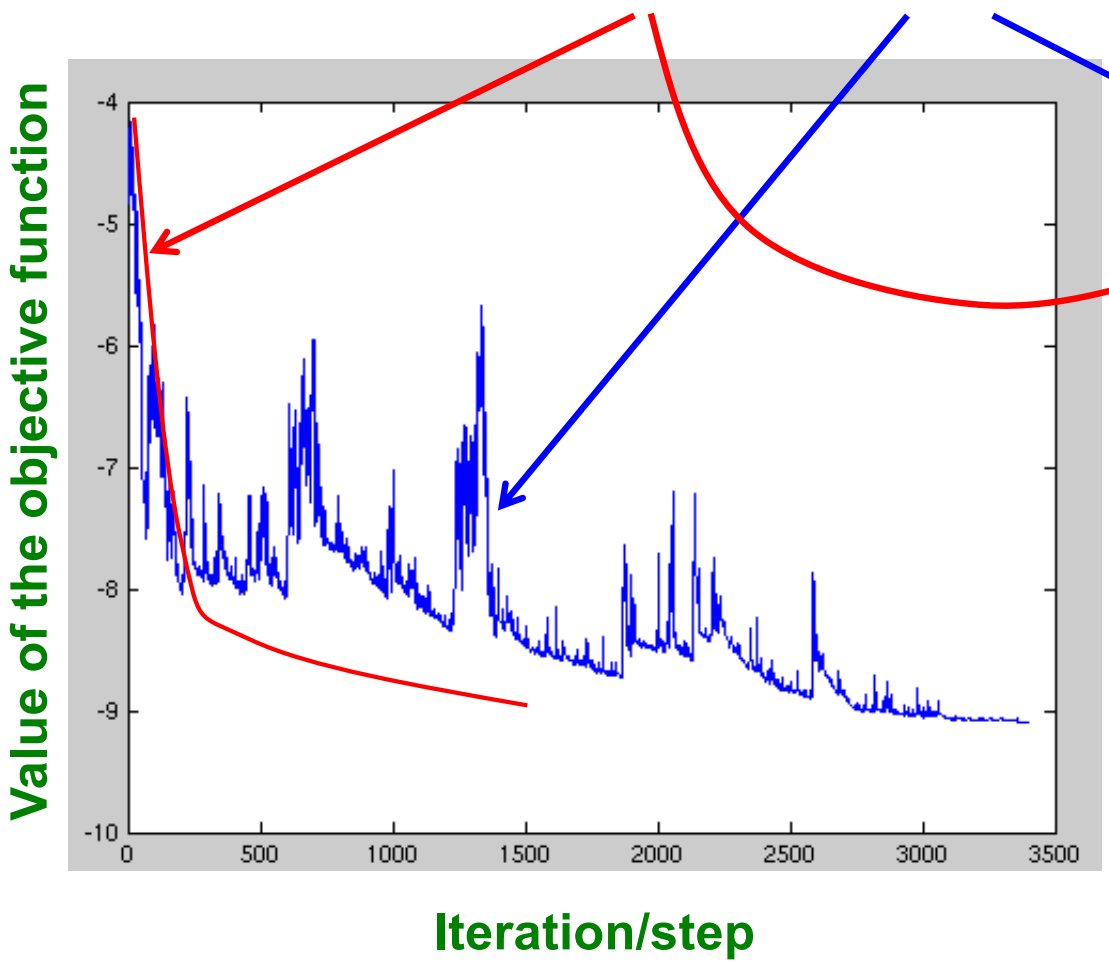
  - Observation:
    - 모든 포인트에 대해서 계산해야 하므로 Gradients 계산이 너무 느려진다

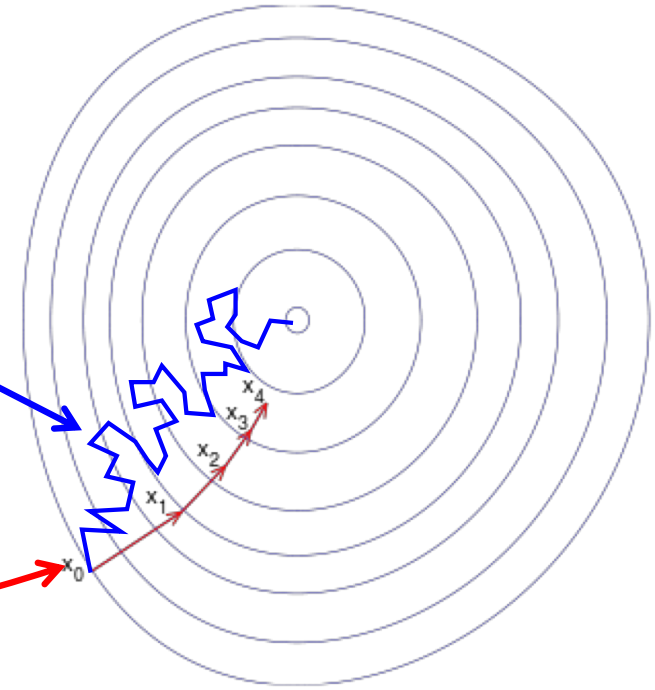
Stochastic Gradient Descent

- Computing gradient requires iterating over all points, can be too costly
- Instead, compute gradient at single training example
- Evaluate gradient for each individual rating and make a step

SGD vs. GD

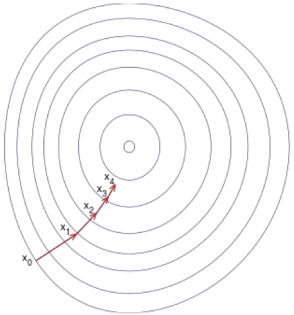
- Convergence of GD vs. SGD





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

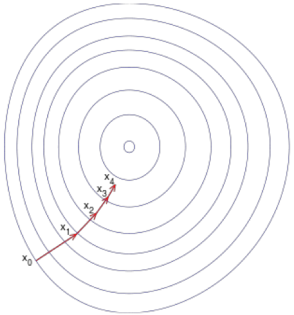


- Gradient Descent (GD) vs. Stochastic GD
  - Observation: $\nabla Q = [\nabla q_{if}]$ where

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

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

Stochastic Gradient Descent



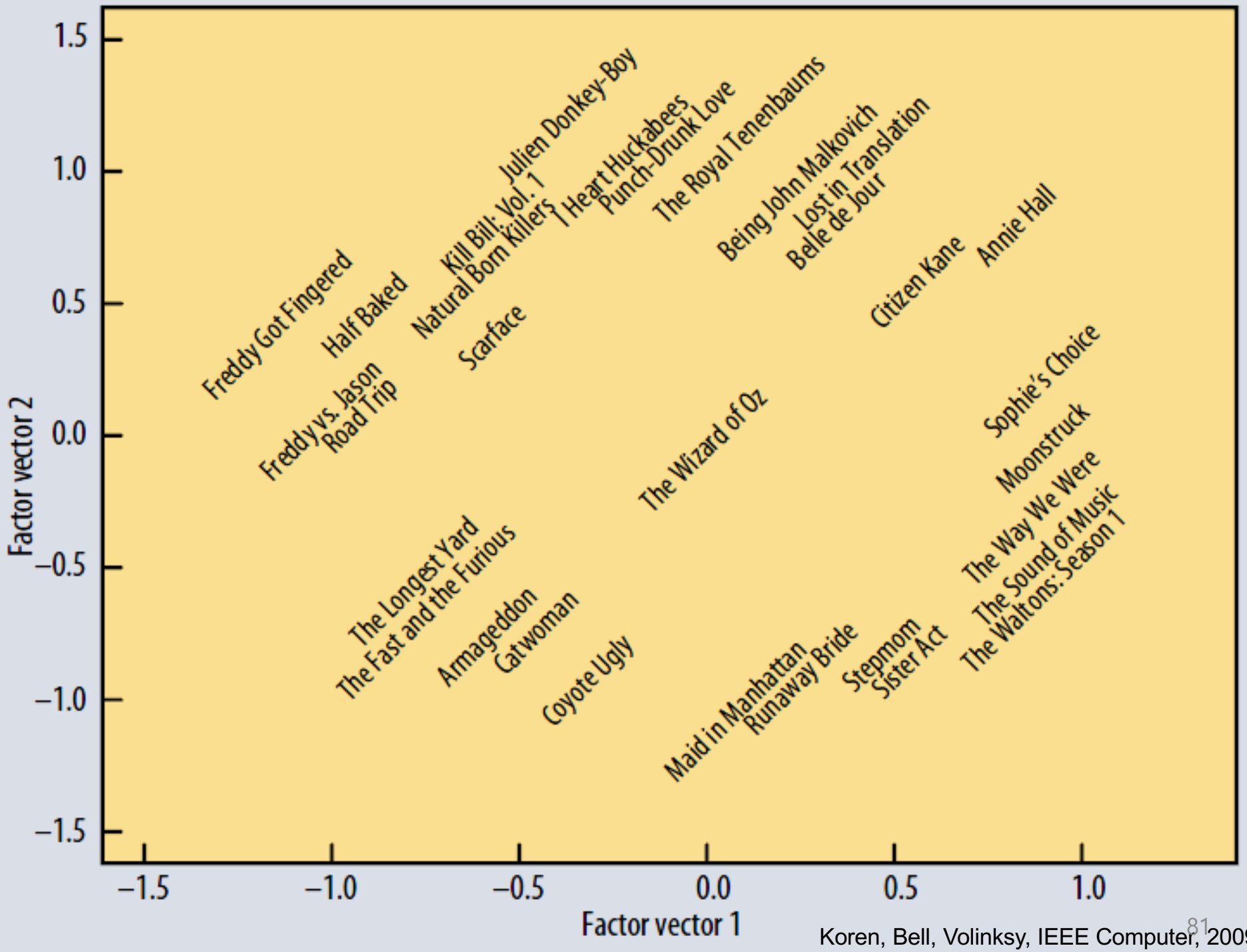
- Stochastic gradient decent:
  - Initialize P and Q (using SVD, pretend missing ratings are 0)
  - Then iterate over the ratings (multiple times if necessary) and update factors:

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

- $\varepsilon_{xi} = 2(r_{xi} - q_i \cdot p_x)$ (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 rate

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

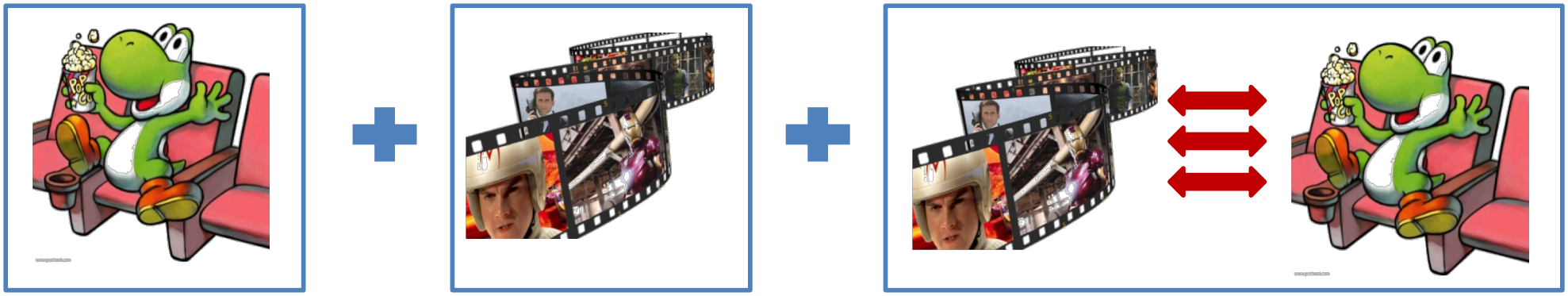


Koren, Bell, Volinksy, IEEE Computer, 2009

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 be havior
- Among the main practical contribut ions 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

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

Overall mean rating | Bias for user ***x*** | Bias for movie ***i*** | 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:

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

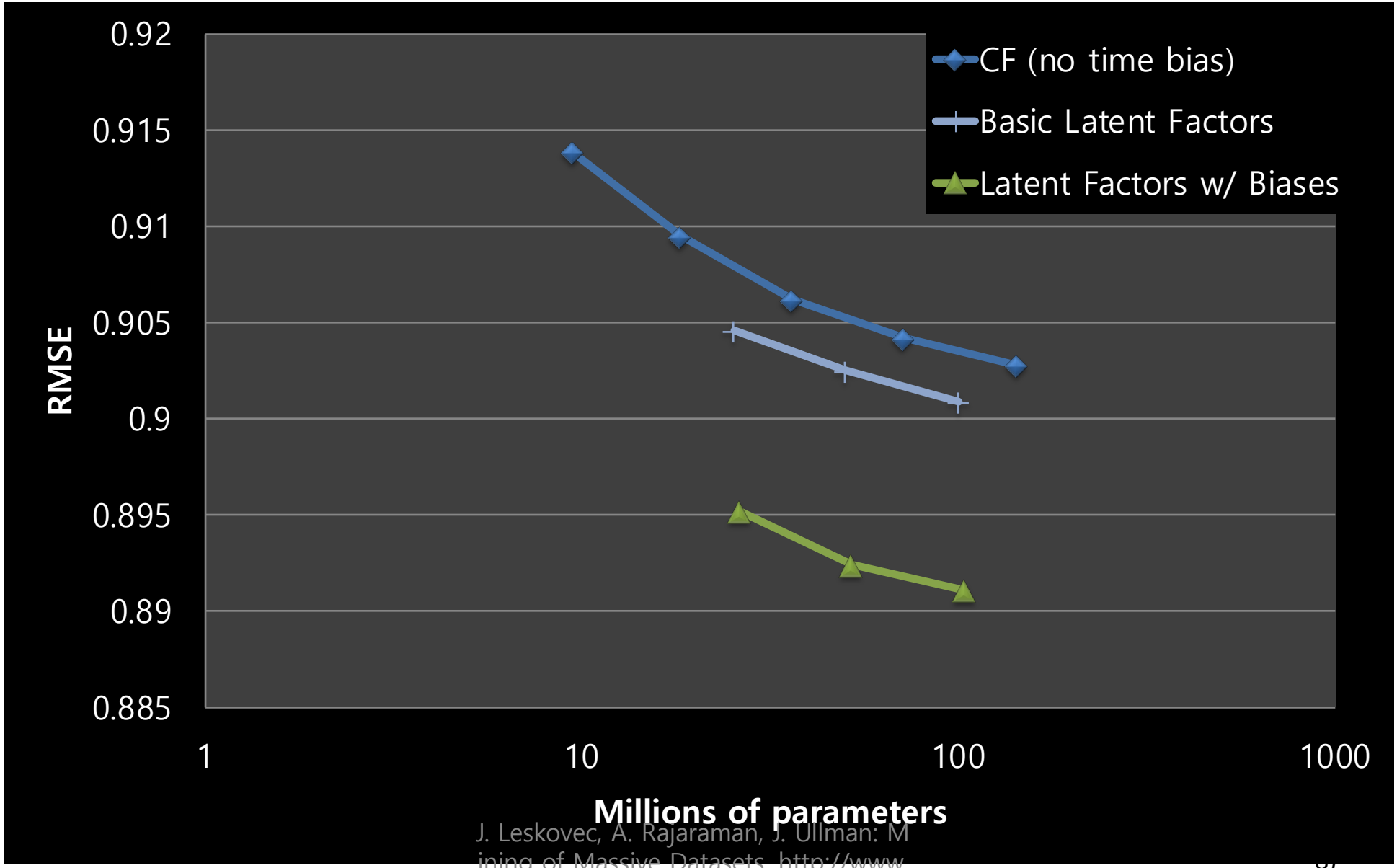
goodness of fit

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

regularization

- 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



J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org

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

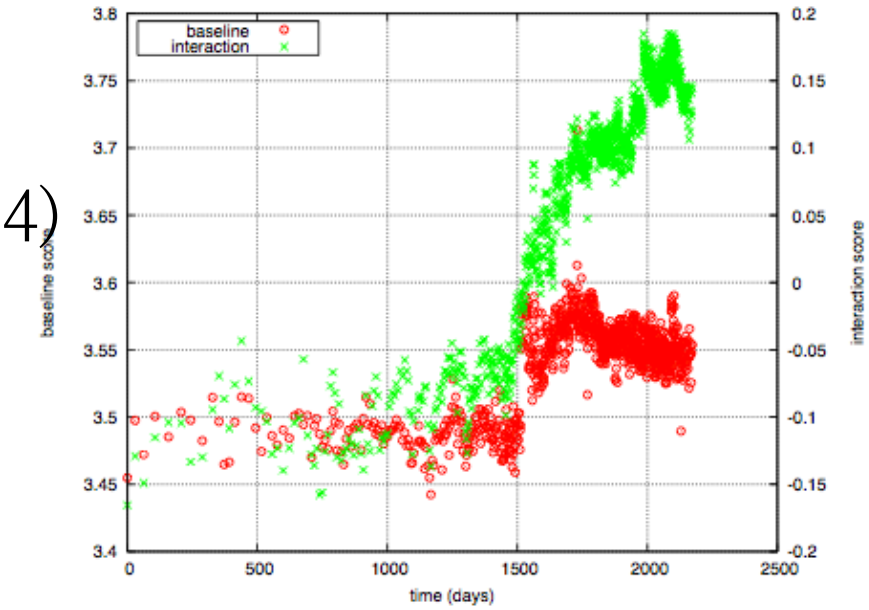
Latent factors+Biases: 0.89

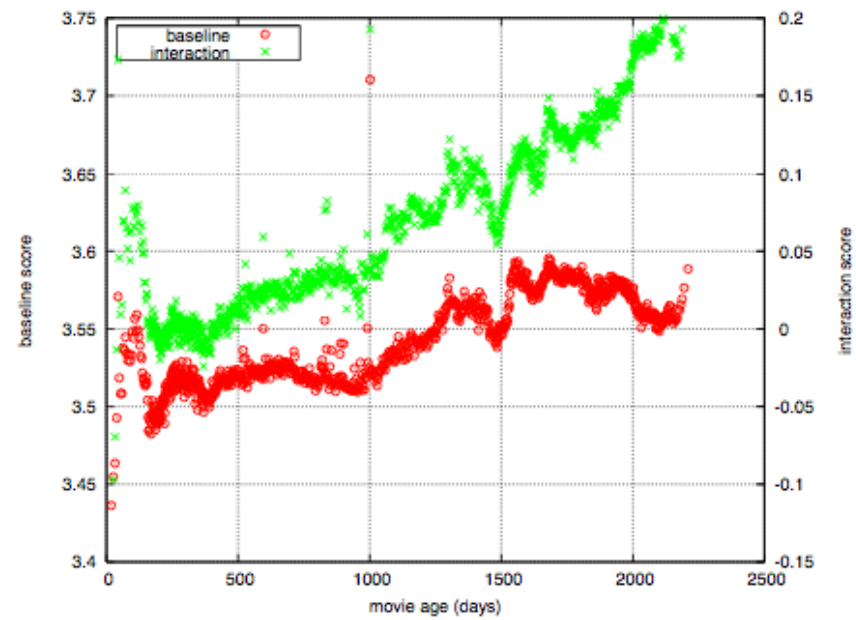
Grand Prize: 0.8563

The Netflix Challenge: 2006–09

Temporal Biases Of Users

- Sudden rise in the average movie rating (early 2004)
  - Improvements in Netflix
  - GUI improvements
  - Meaning of rating changed
- 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

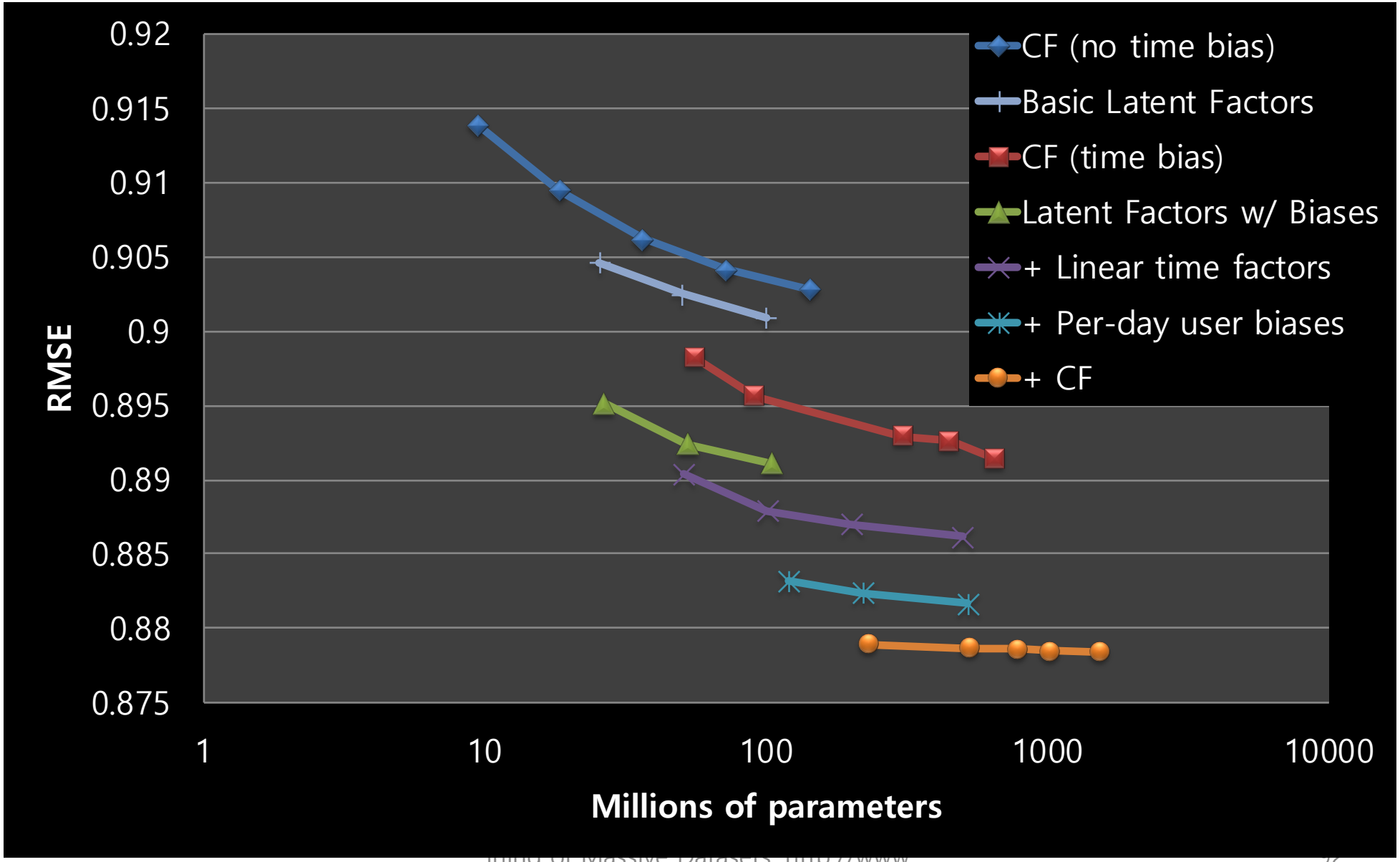
    (2) [illegible] consecutive weeks

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

- 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



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

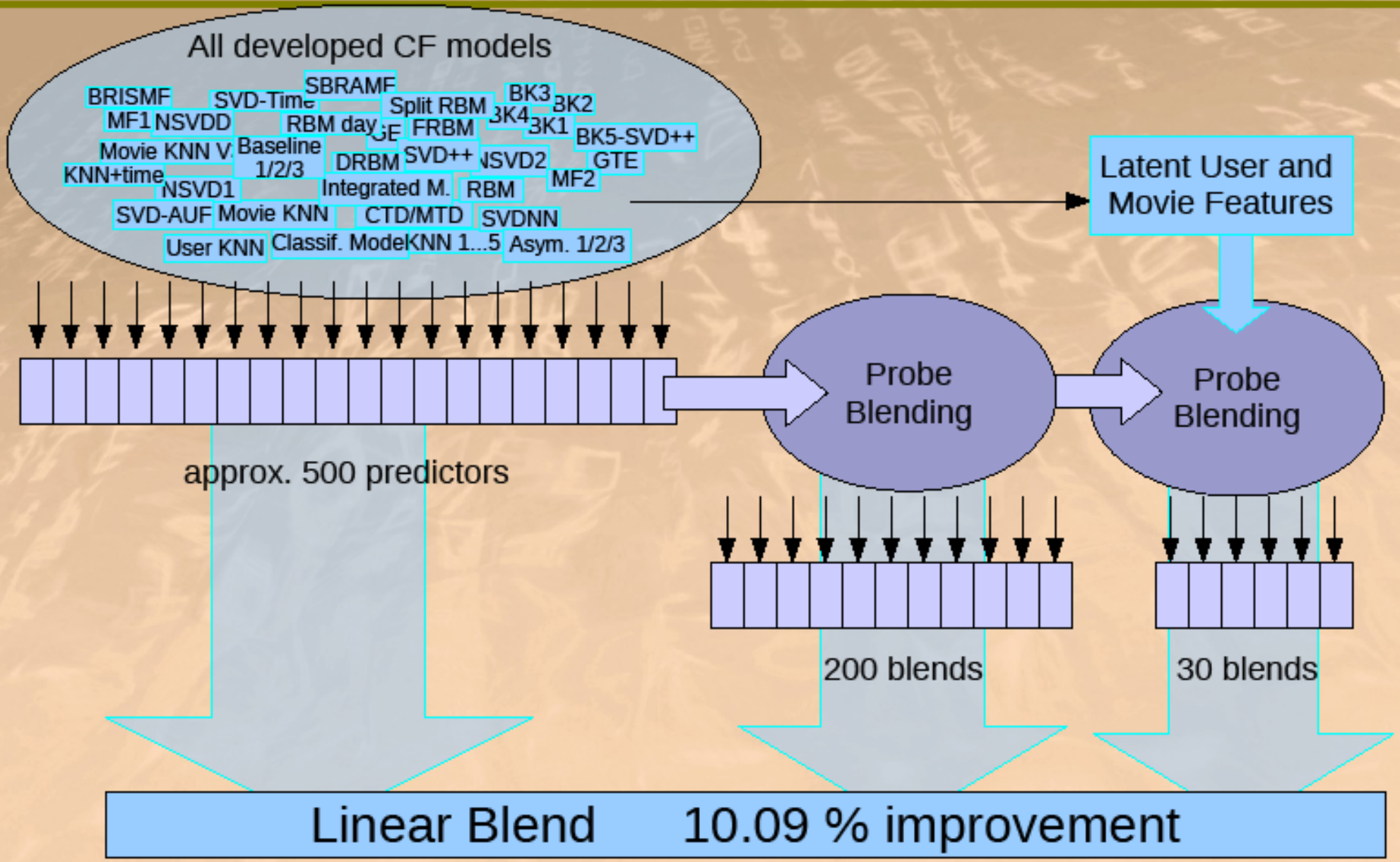
Latent factors+Biases+Time: 0.876

Grand Prize: 0.8563

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

The big picture

Solution of BellKor's Pragmatic Chaos



Standing on June 26th 2009

NETFLIX

Netflix Prize

Home | Rules | Leaderboard | Register | Update | Submit | Download

Leaderboard

Display top 20 leaders.

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

June 26th submission triggers 30-day "last call"

NETFLIX

Netflix Prize

COMPLETED

Home | Rules | Leaderboard | Update | Download

Leaderboard

Showing Test Score. Click here to show quiz score

Display top 20 leaders.

Rank	Team Name	Best Test Score	% Improvement	Best Submit Time
Grand Prize - RMSE = 0.8567 - Winning Team: BellKor's Pragmatic Chaos				
1	BellKor's Pragmatic Chaos	0.8567	10.06	2009-07-26 18:18:28
2	The Ensemble	0.8567	10.06	2009-07-26 18:38:22
3	Grand Prize Team	[illegible]	[illegible]	[illegible]
4	Opera Solutions and Vandelay United	0.8588	9.84	2009-07-10 01:12:31
5	Vandelay Industries !	0.8591	9.81	2009-07-10 00:32:20
6	PragmaticTheory	0.8594	9.77	2009-06-24 12:06:56
7	BellKor in BigChaos	0.8601	9.70	2009-05-13 08:14:09
8	Dace_	0.8612	9.59	2009-07-24 17:18:43
9	Feeds2	0.8622	9.48	2009-07-12 13:11:51
10	BigChaos	0.8623	9.47	2009-04-07 12:33:59
11	Opera Solutions	0.8623	9.47	2009-07-24 00:34:07
12	BellKor	0.8624	9.46	2009-07-26 17:19:11
Progress Prize 2008 - RMSE = 0.8627 - Winning Team: BellKor in BigChaos				
13	xiangliang	0.8642	9.27	2009-07-15 14:53:22
14	Gravity	0.8643	9.26	2009-04-22 18:31:32
15	Ces	0.8651	9.18	2009-06-21 19:24:53
16	Invisible Ideas	0.8653	9.15	2009-07-15 15:53:04
17	Just a guy in a garage	0.8662	9.06	2009-05-24 10:02:54
18	J Dennis Su	0.8666	9.02	2009-03-07 17:16:17
19	Craig Carmichael	0.8666	9.02	2009-07-25 16:00:54
20	acmehill	0.8668	9.00	2009-03-21 16:20:50
Progress Prize 2007 - RMSE = 0.8723 - Winning Team: KorBell				

J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org

Million $ Awarded Sept 21st 2009

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
  - Koren, Bell, & Volinsky, Matrix Factorization Techniques for Recommender Systems, IEEE Computer 2009
- Open Innovation