Mining Massive Datasets
Recommender Systems

Outline

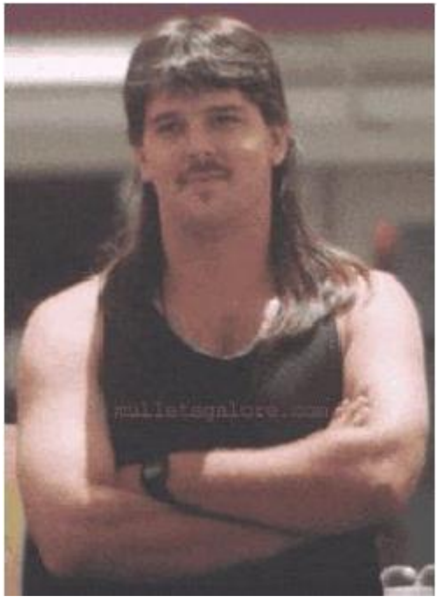
1.1 Recommender Systems

1.2 Content-based Recommendations

1.3 Collaborative Filtering

1.4 Latent Factor Models

Example



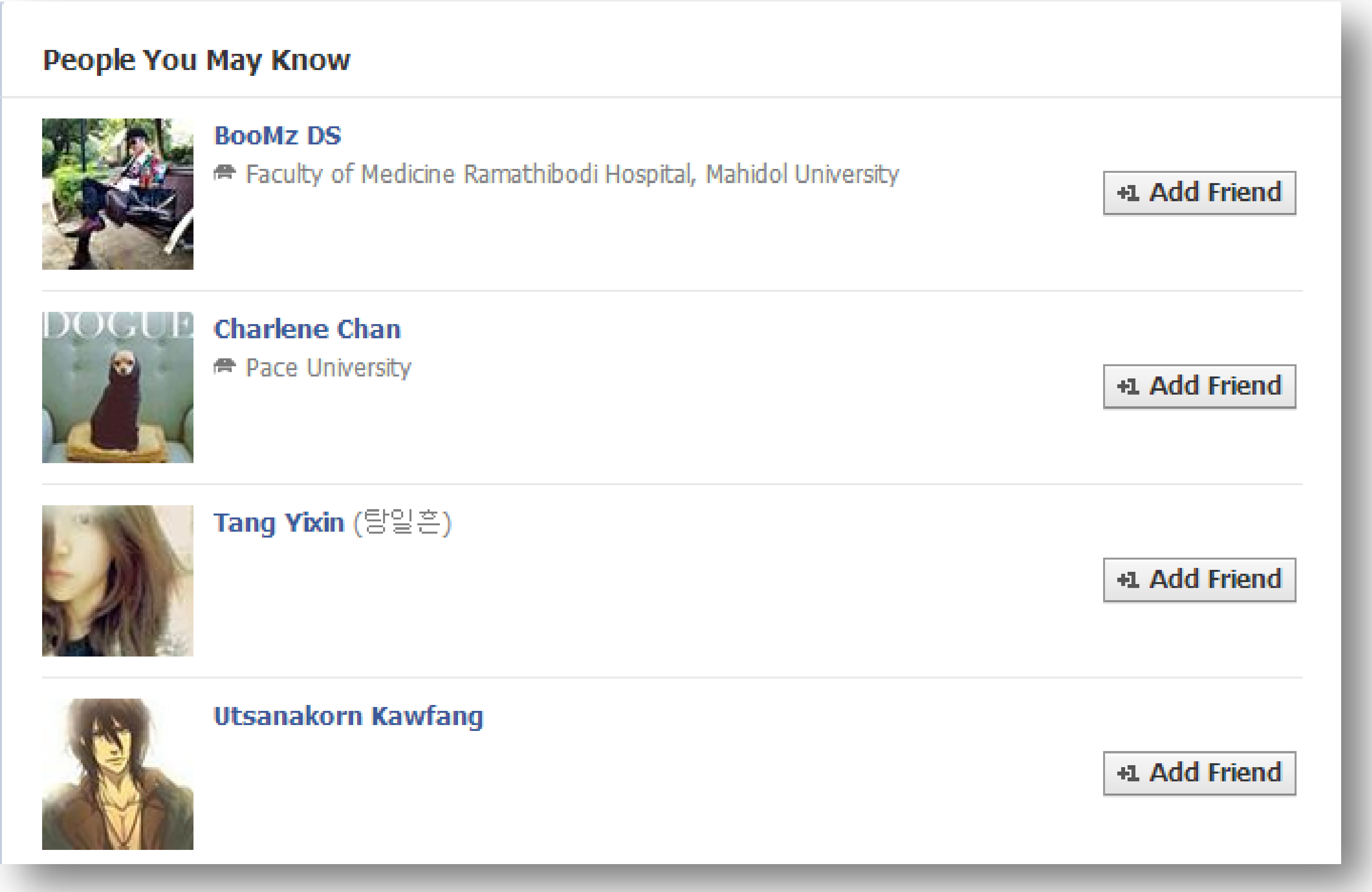
Customer X

- Buys Metallica CD
- Buys Megadeth CD

Customer Y

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

Example



People You May Know

BooMz DS
Faculty of Medicine Ramathibodi Hospital, Mahidol University

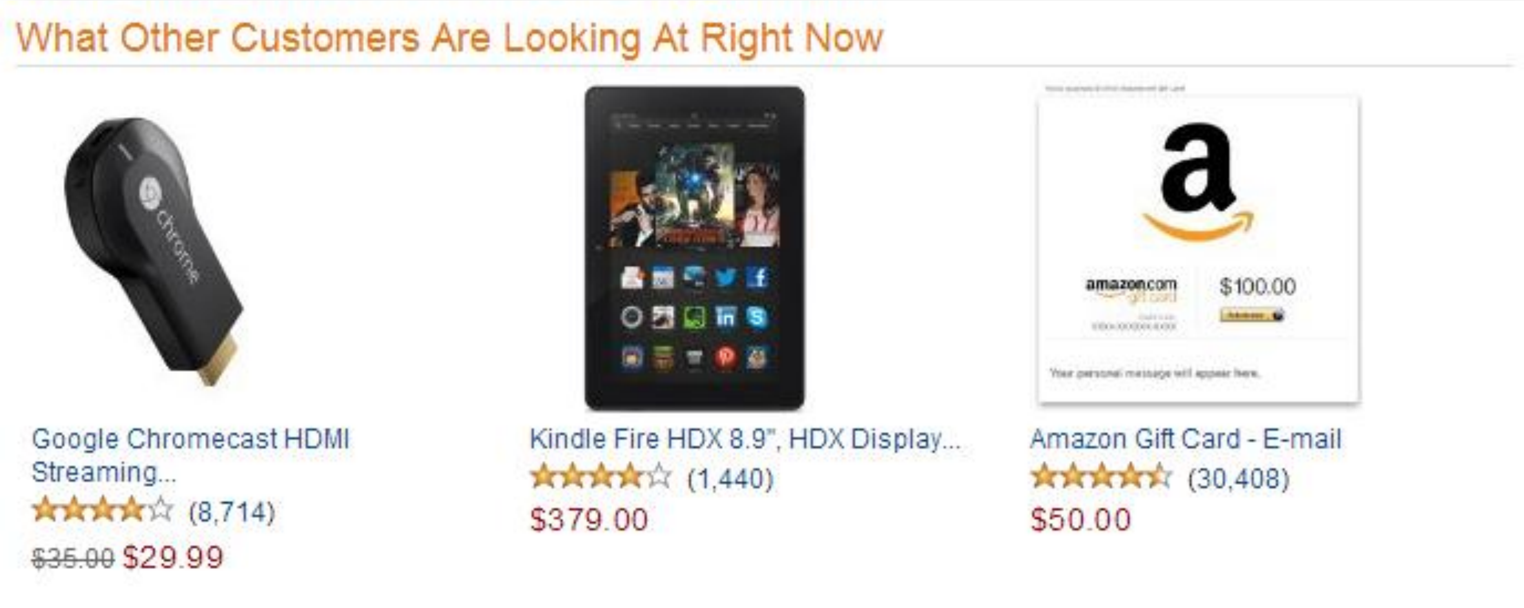
Charlene Chan
Pace University

Tang Yixin (탕일흔)

Utsanakorn Kawfang

www.facebook.com

Example



What Other Customers Are Looking At Right Now

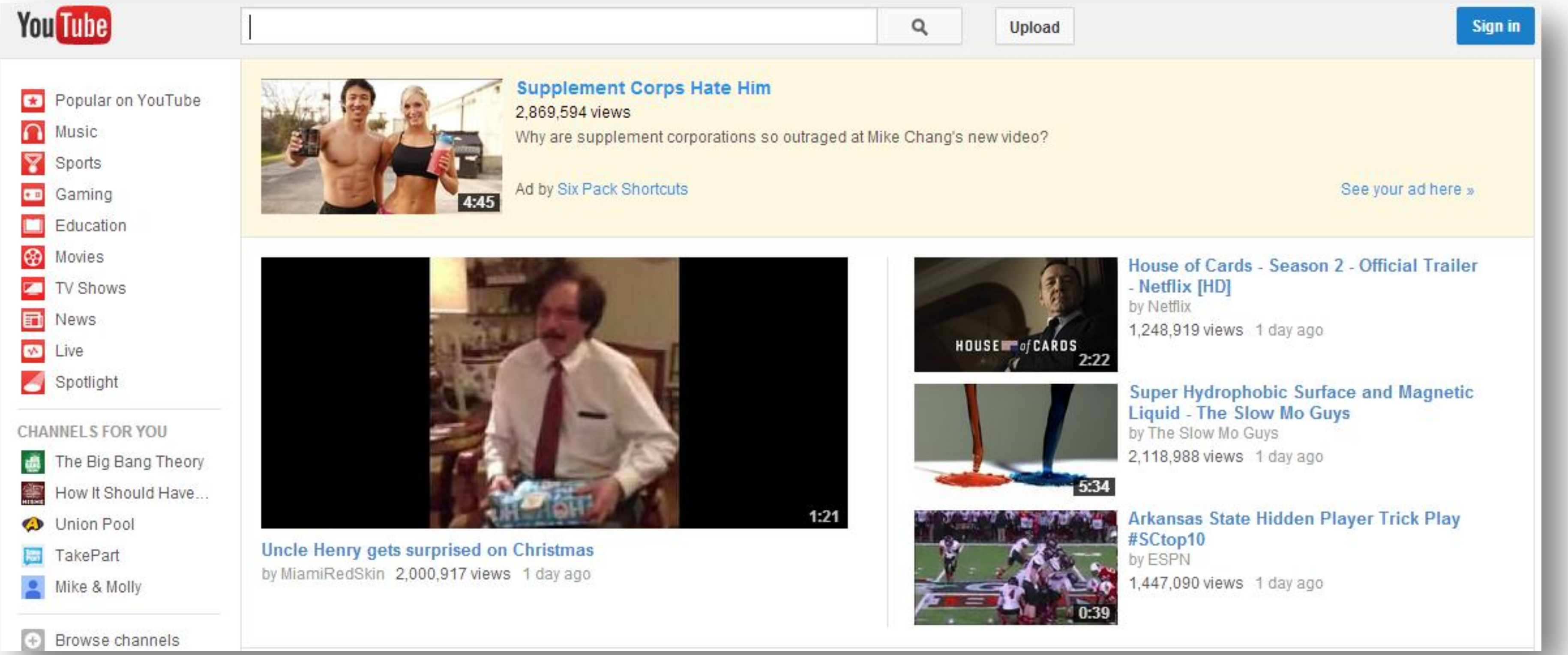
Google Chromecast HDMI Streaming...
★★★★☆ (8,714)
~~$35.00~~ $29.99

Kindle Fire HDX 8.9", HDX Display...
★★★★☆ (1,440)
$379.00

Amazon Gift Card - E-mail
★★★★☆ (30,408)
$50.00

www.amazon.com

Example



Supplement Corps Hate Him
2,869,594 views
Why are supplement corporations so outraged at Mike Chang's new video?
Ad by Six Pack Shortcuts
See your ad here »
4:45

Uncle Henry gets surprised on Christmas
by MiamiRedSkin 2,000,917 views 1 day ago
1:21

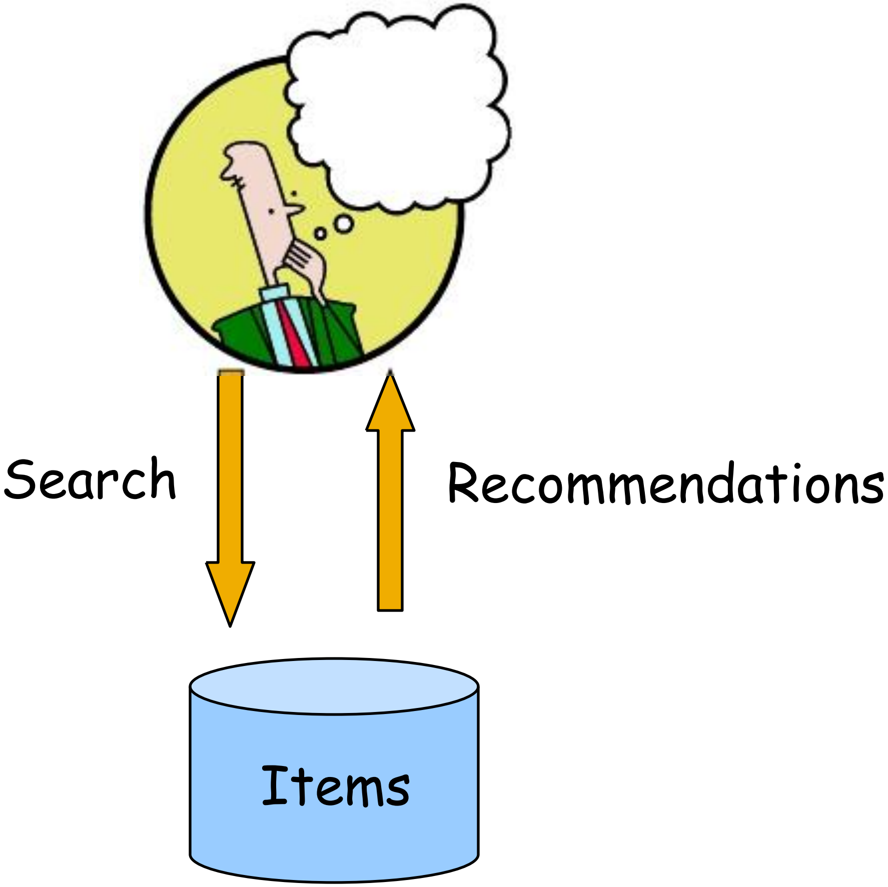
House of Cards - Season 2 - Official Trailer - Netflix [HD]
by Netflix
1,248,919 views 1 day ago
2:22

Super Hydrophobic Surface and Magnetic Liquid - The Slow Mo Guys
by The Slow Mo Guys
2,118,988 views 1 day ago
5:34

Arkansas State Hidden Player Trick Play #SCtop10
by ESPN
1,447,090 views 1 day ago
0:39

www.youtube.com

Recommendations



Search

Recommendations

Items

Products, web sites, blogs, news items, …

Recommendations

Shelf space is a scarce commodity for traditional retailers

- Also: TV networks, movie theaters,…

Web enables near-zero-cost dissemination of information about products

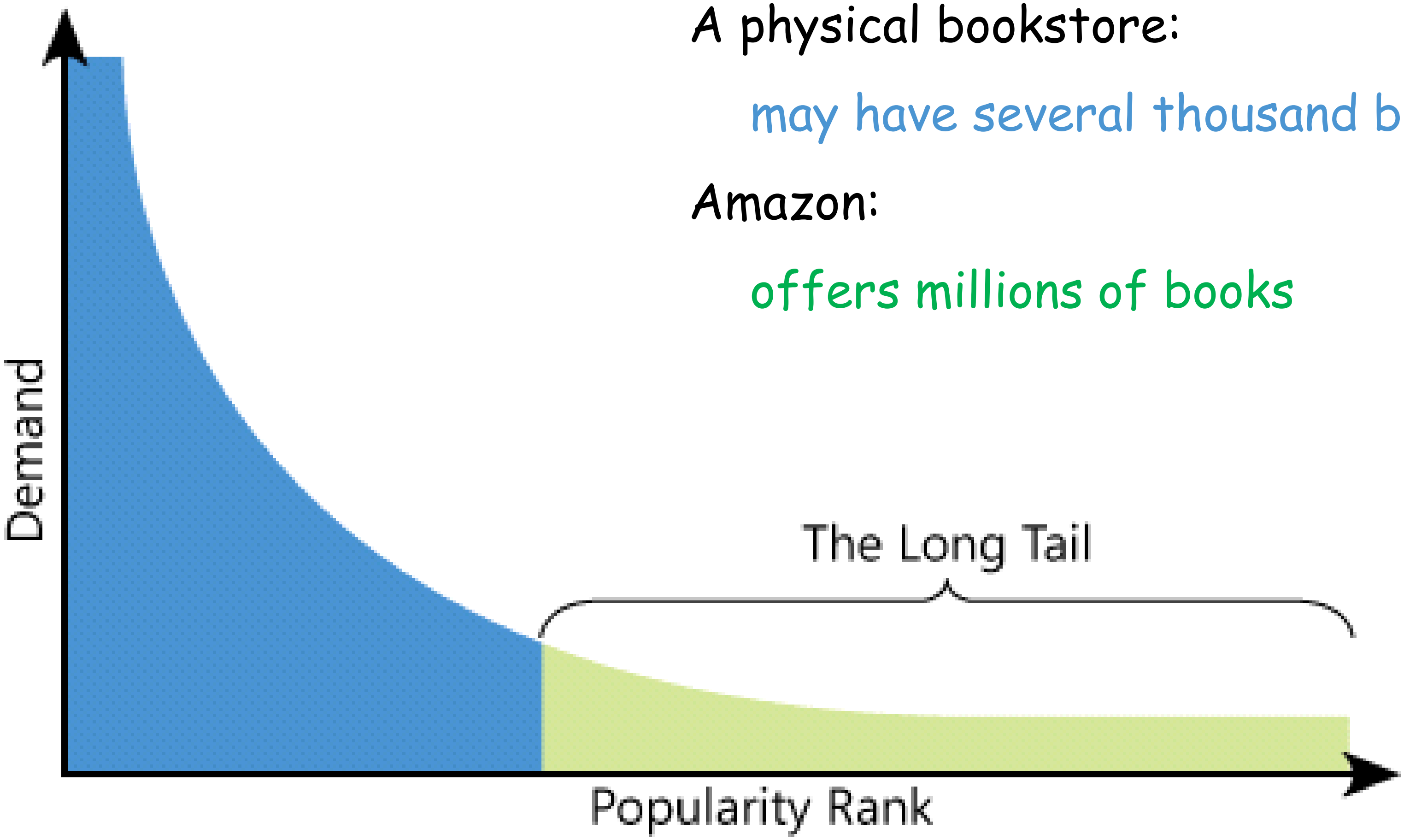
- From scarcity to abundance

More choice necessitates better filters

- Recommendation engines

Into Thin Air & Touching the Void a
bestseller: http://www.wired.com/wired/archive/12.10/tail.html

The Long Tail



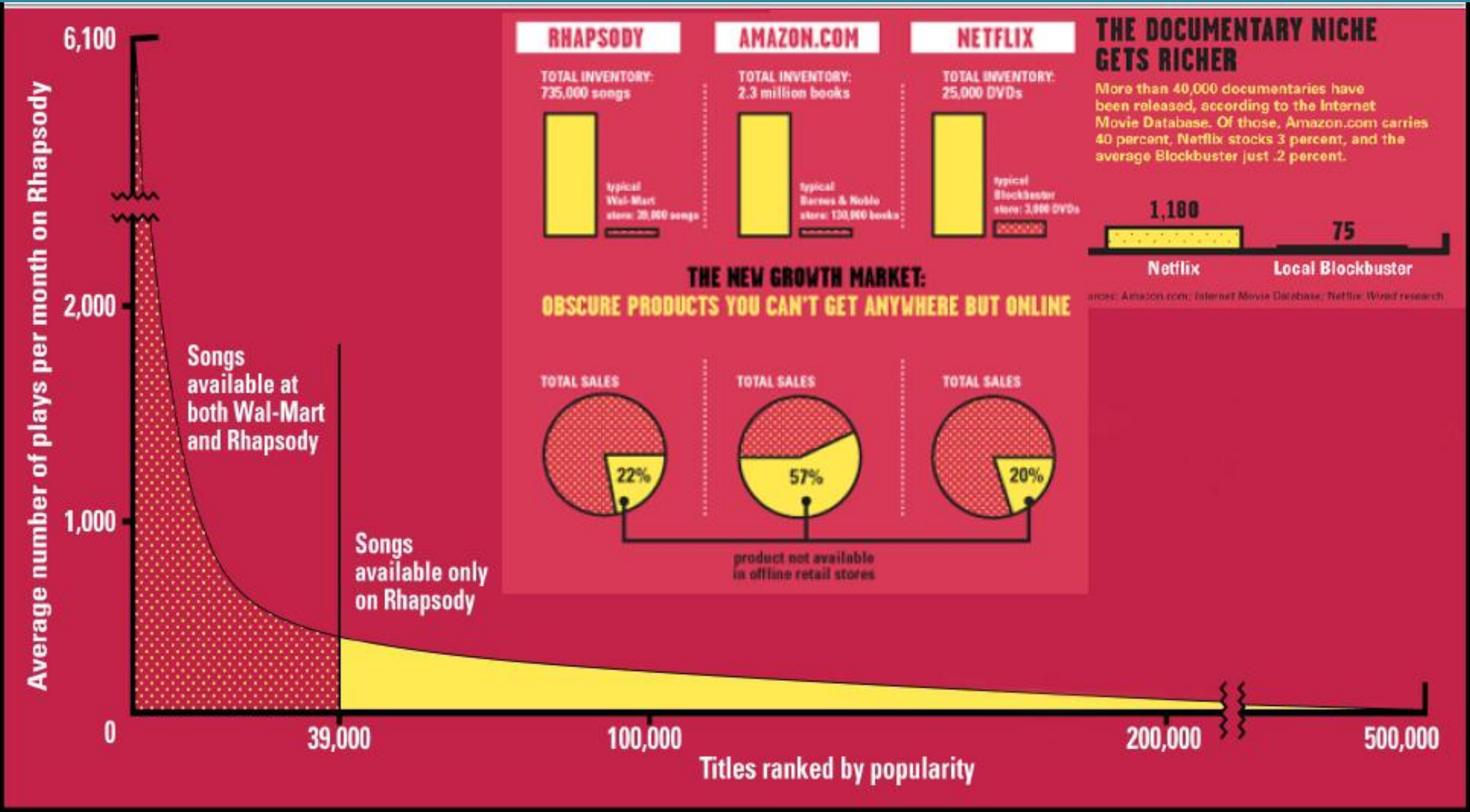
A physical bookstore:

may have several thousand books

Amazon:

offers millions of books

The Long Tail



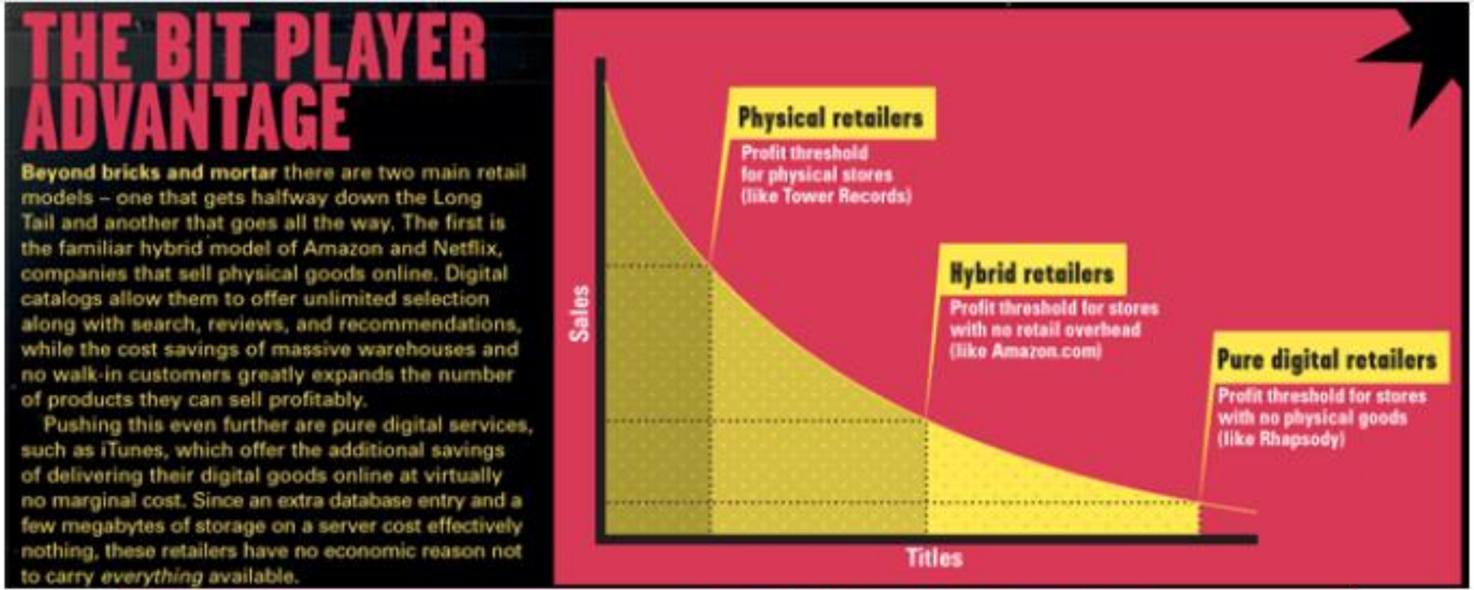
More than 40,000 documentaries have been released, according to the Internet Movie Database. Of those, Amazon.com carries 40 percent, Netflix stocks 3 percent, and the average Blockbuster just .2 percent.

urces: Amazon.com; Internet Movie Database; Netflix; Wired research

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

Source: Chris Anderson (2004)

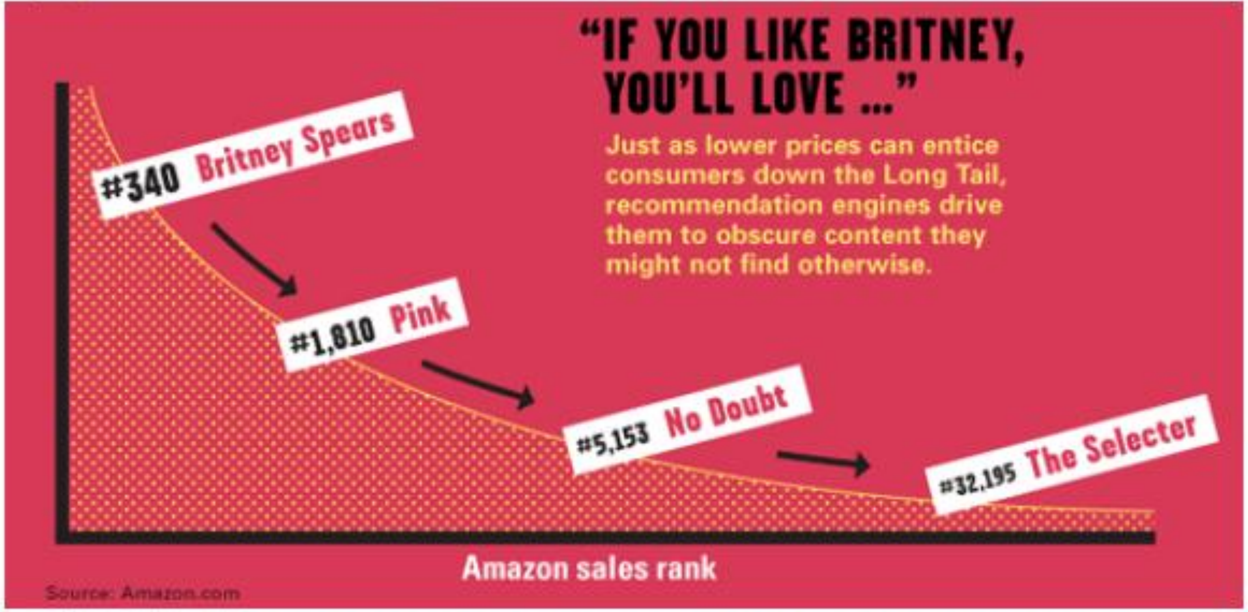
Physical vs. online



THE BIT PLAYER ADVANTAGE

Beyond bricks and mortar there are two main retail models – one that gets halfway down the Long Tail and another that goes all the way. The first is the familiar hybrid model of Amazon and Netflix, companies that sell physical goods online. Digital catalogs allow them to offer unlimited selection along with search, reviews, and recommendations, while the cost savings of massive warehouses and no walk-in customers greatly expands the number of products they can sell profitably.

Pushing this even further are pure digital services, such as iTunes, which offer the additional savings of delivering their digital goods online at virtually no marginal cost. Since an extra database entry and a few megabytes of storage on a server cost effectively nothing, these retailers have no economic reason not to carry *everything* available.



"IF YOU LIKE BRITNEY, YOU'LL LOVE ..."

Just as lower prices can entice consumers down the Long Tail, recommendation engines drive them to obscure content they might not find otherwise.

Source: Amazon.com

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

Types

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 $u : X \times S \to R$

- 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
Carl	0.2		1	
David				0.4

Key Problem

- (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
  - 3) Latent factor based

Outline

1.1 Recommender Systems

1.2 Content-based Recommendations

1.3 Collaborative Filtering

1.4 Latent Factor Models

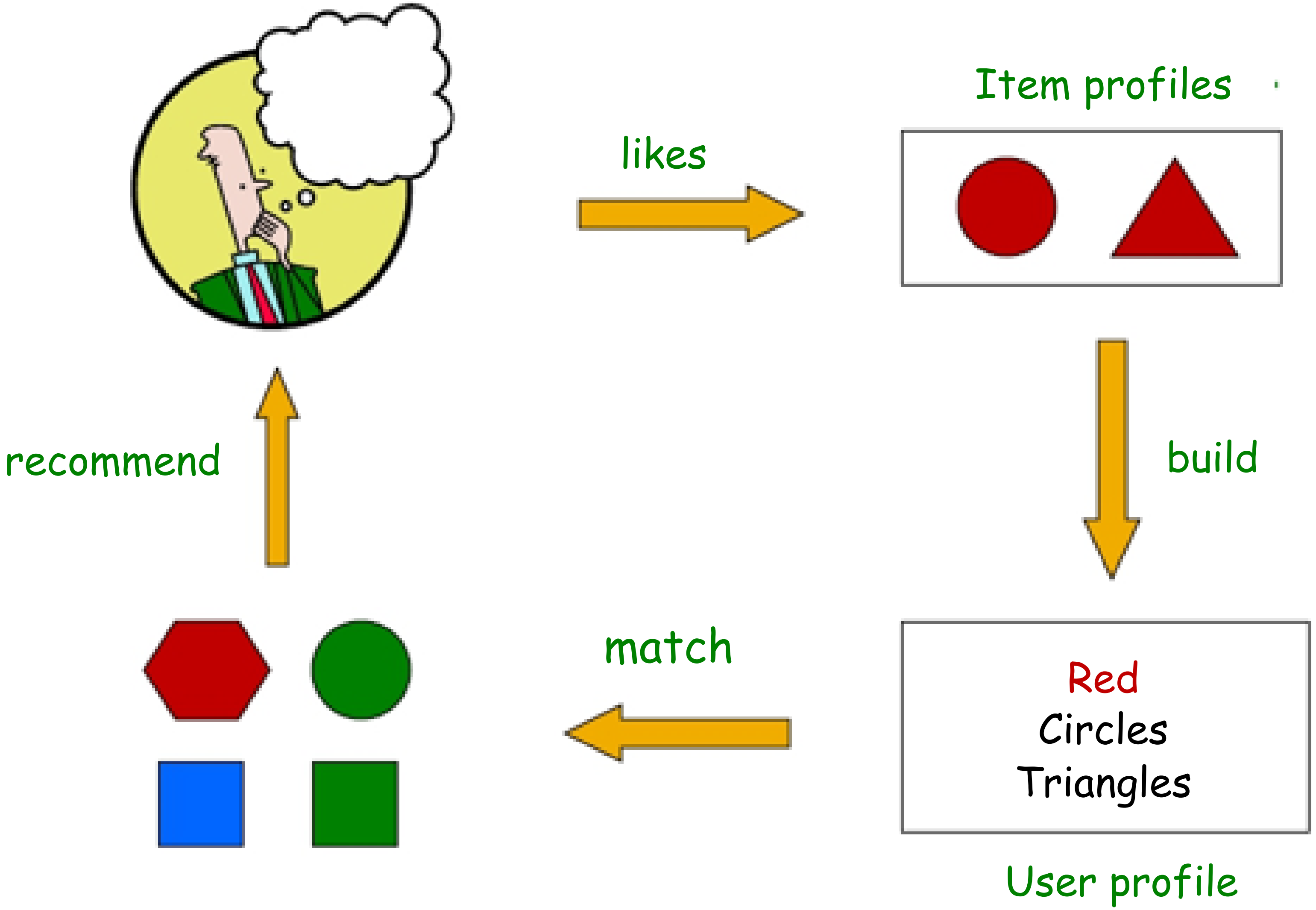
Content-based

- 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



likes

Item profiles

build

Red
Circles
Triangles

User profile

match

recommend

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

- User profile possibilities:
  - Weighted average of rated item profiles
  - Variation: weight by difference from average rating for item
  - …
- Prediction heuristic:
  - Given user profile *x* and item profile *i*, estimate

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

Pros

- +: 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

- -: 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?

Outline

1.1 Recommender Systems

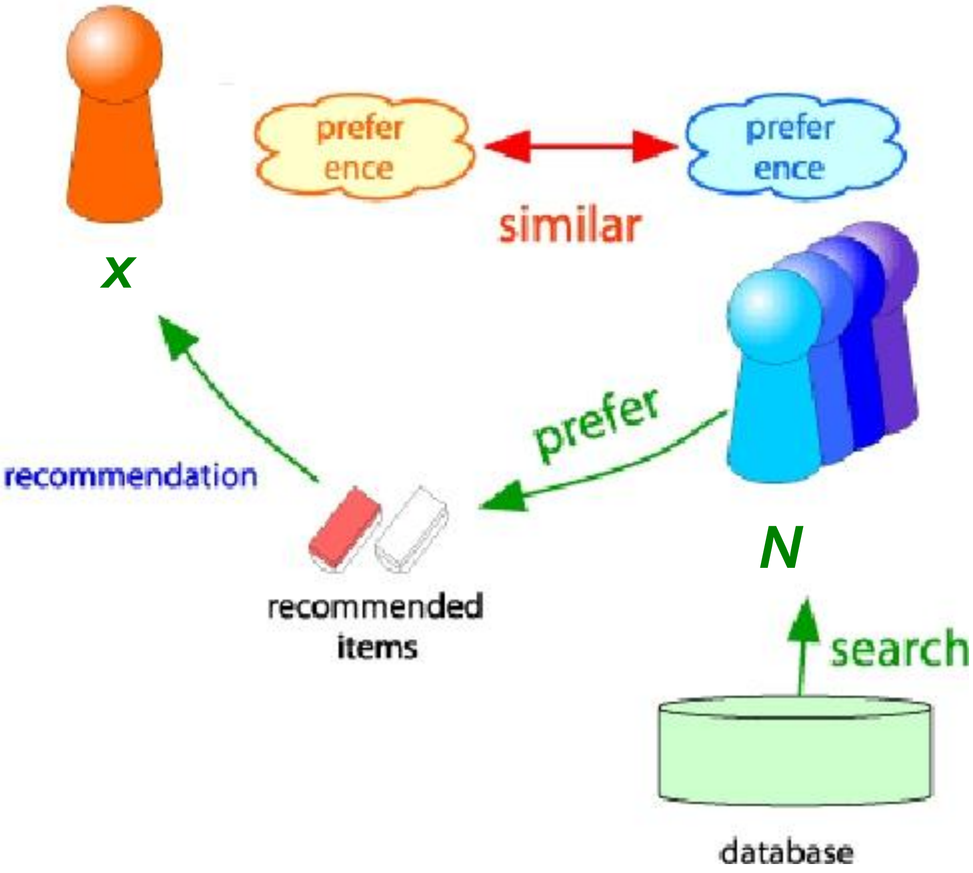
1.2 Content-based Recommendations

1.3 Collaborative Filtering

1.4 Latent Factor Models

Similar Users

- 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*



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
  - sim(x, y) = cos(r_x, r_y) =
  - Problem: Treats missing ratings as "negative"
- Pearson correlation coefficient

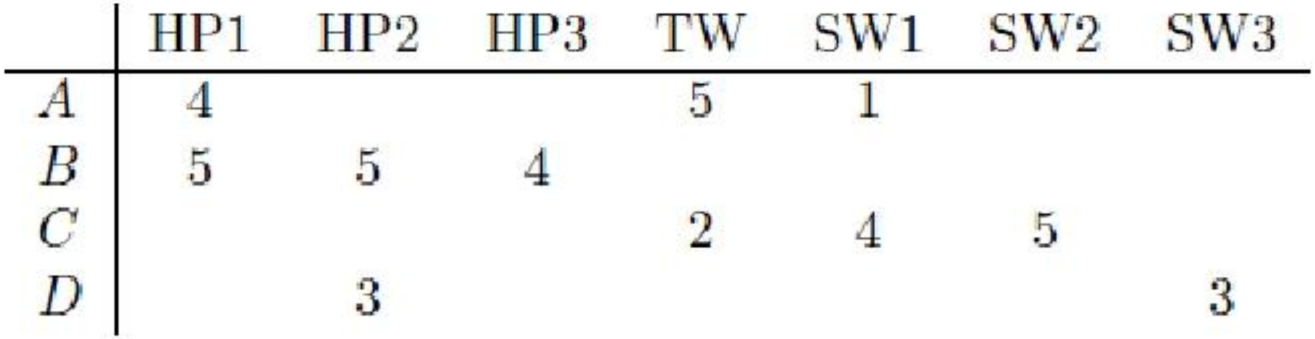
$$sim(x,y) = \frac{\sum_{s\in S_{xy}}(r_{xs}-\overline{r_x})(r_{ys}-\overline{r_y})}{\sqrt{\sum_{s\in S_{xy}}(r_{xs}-\overline{r_x})^2}\sqrt{\sum_{s\in S_{xy}}(r_{ys}-\overline{r_y})^2}}$$

rx, ry as sets:
rx = {1, 4, 5}
ry = {1, 3, 4}

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

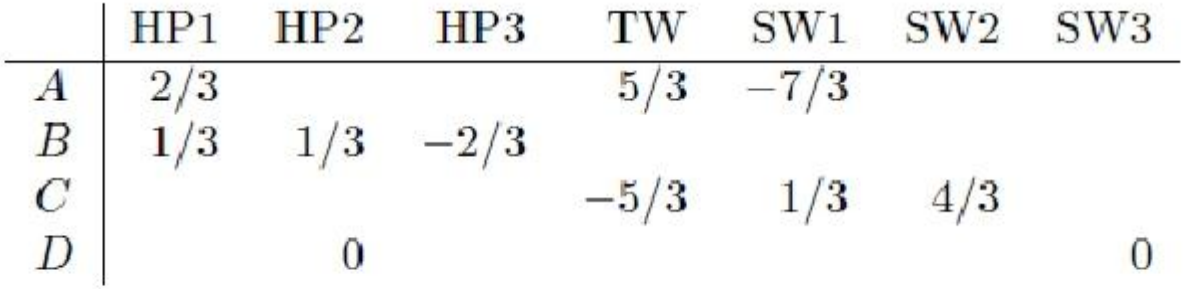
$\overline{r_x}$, $\overline{r_y}$... avg.
rating of x, y

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



	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

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 *s* of *user x*:
  - $r_{xi} = \frac{1}{k} \sum_{y \in N} r_{yi}$
  - $r_{xi} = \frac{\sum_{y \in N} s_{xy} \cdot r_{yi}}{\sum_{y \in N} s_{xy}}$
  - Other options?
- Many other tricks possible…

Similar Items

- 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

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

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

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 mi from each movie i
m1 = (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:

s13=0.41, s16=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:

r15=(0.41*2+0.59*3)/(0.41+0.59) = 2.6

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

Item-Item CF (|N|=2)

	Avatar	LOTR	LOTR	Pirates
Alice	1		0.2	
Bob		0.5		0.3
Bob	0.2		1	
David				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

- + 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

CF: Common Practice

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

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

Baseline estimate for r_{xi}

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

- μ = overall mean movie rating
- b_x = rating deviation of user *x*
  = *(avg. rating of user x)* – μ
  b_i = rating deviation of movie *i*

Outline

1.1 Recommender Systems

1.2 Content-based Recommendations

1.3 Collaborative Filtering

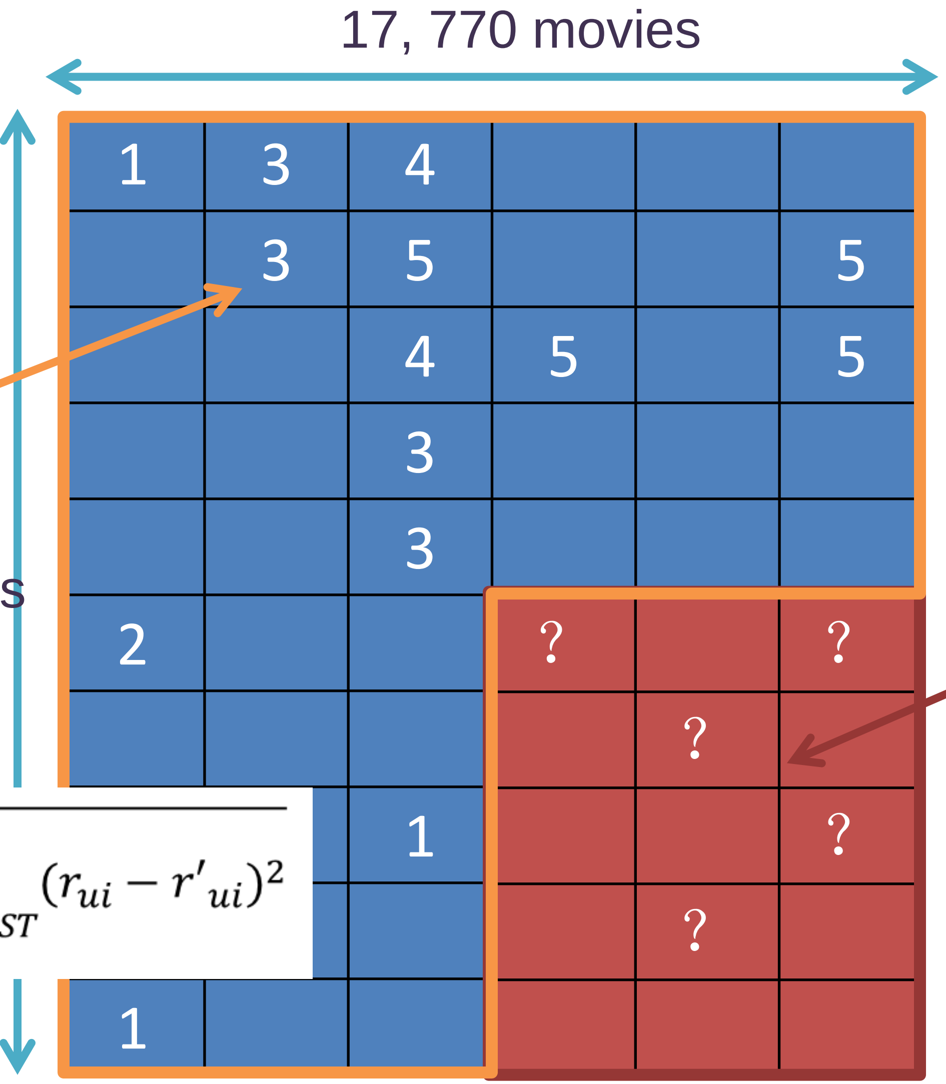
1.4 Latent Factor Models

The Netflix Prize

- **$1 million prize**
- **Training data**
  - 100 million ratings, 480, 000 users, 17, 770 movies
  - 7 years of data: 98-05
- **Test data**
  - Last few ratings of each user (2.8 million)
- **Evaluation criterion**
  - 10% improvement on Netflix in Root Mean Square Error (RMSE), that's 90%×0.9525=0.8572
- **Competition**
  - Until 2009, 5, 000 teams, 40, 000 submits

The Netflix Utility Matrix R

Matrix R



17, 770 movies

480, 000 users

Training data

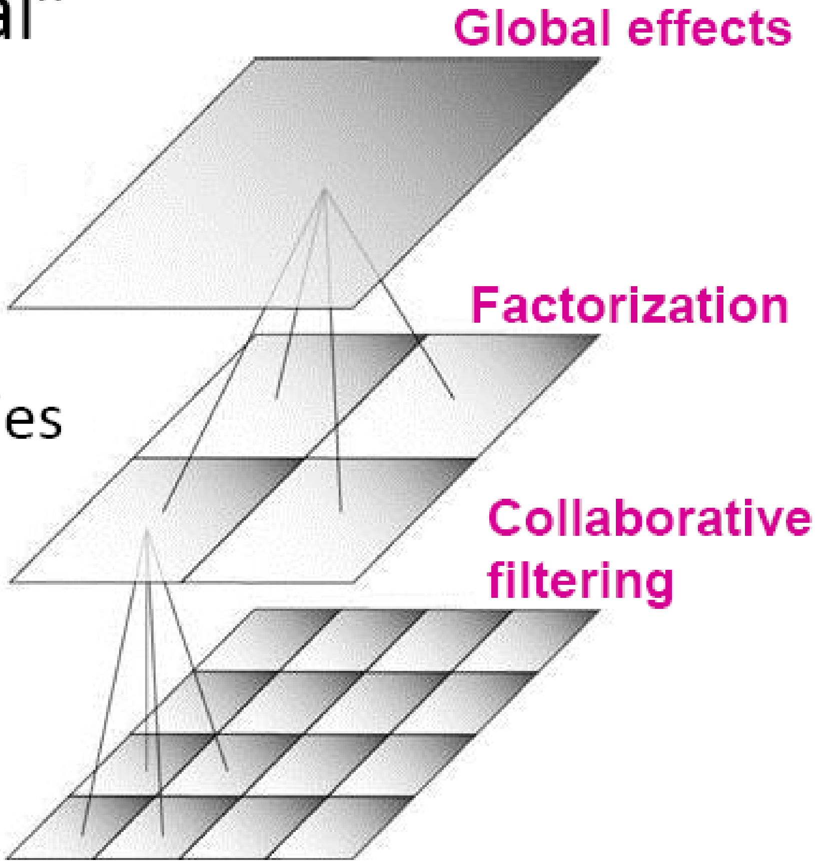
Test data

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

$$RMSE = \sqrt{\sum_{(u,i)\in TEST} (r_{ui} - r'_{ui})^2}$$

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 Effect

- **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 most similar to ***i*** that were rated by ***x***

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

s_{ij}… similarity of items ***i*** and ***j***
r_{uj}…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 Effect

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

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

baseline estimate for r_{xi}

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

$\boldsymbol{\mu}$ = overall mean rating

$\boldsymbol{b_x}$ = rating deviation of user $\boldsymbol{x}$

= (*avg. rating of* user $\boldsymbol{x}$) – $\boldsymbol{\mu}$

$\boldsymbol{b_i}$ = (*avg. rating of* movie $\boldsymbol{i}$) – $\boldsymbol{\mu}$

Problems/Issues:

1) Similarity measures are "arbitrary"

2) Pairwise similarities neglect interdependencies among users

3) Taking a weighted average can be restricting

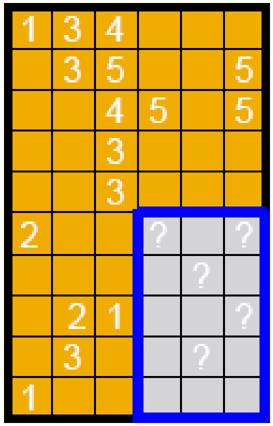
Solution: **Instead of s_{ij} use w_{ij} that we estimate directly from data**

Idea: Interpolation Weights W_{ij}

- $\widehat{r_{xi}} = b_{xi} + \sum_{j \in N(i,x)} w_{ij}(r_{xj} - b_{xj})$
- **How to set *w_{ij}*?**
  - Remember, error metric is **SSE:** $\sum_{(i,u) \in R}(\hat{r}_{ui} - r_{ui})^2$
  - Find ***w_{ij}*** that minimize **SSE** on **training data!**
    - Models relationships between item ***i*** and its neighbors ***j***
  - ***w_{ij}*** can be **learned/estimated** based on ***x*** and all other users that rated ***i***

Why is this a good idea?

Recommendations Via Optimization



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

- **Here is what we just did:**
  - **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. Can't really do this. Why?
    - **Let's set values *w* such that they work well on known (user, item) ratings**
      And **hope** these ***w***s will predict well the unknown ratings
- This is the first time in the class that we see **Optimization methods**

Recommendations Via Optimization

- **Idea: Let’s set values *w* such that they work well on known (user, item) ratings**
- **How to find such values *w*?**
- **Idea:** Define an objective function and solve the optimization problem
- Find ***w*** w_{ij} that minimize **SSE** on **training data**!

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

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

Interpolation Weights

- We have the optimization problem, now what? ⍰ $$\min_{w_{ij}} \sum_x \left(\left[b_{xi} + \sum_{j \in N(i;x)} w_{ij}(r_{xj} - b_{xj}) \right] - r_{xi} \right)^2$$
- Gradient decent
  - Iterate until convergence: $\boldsymbol{w} \quad \boldsymbol{w} - \eta \boldsymbol{\nabla w}$ $\quad \eta$ … learning rate
  - where $\boldsymbol{\nabla w}$ is gradient (derivative evaluated on data):

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

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

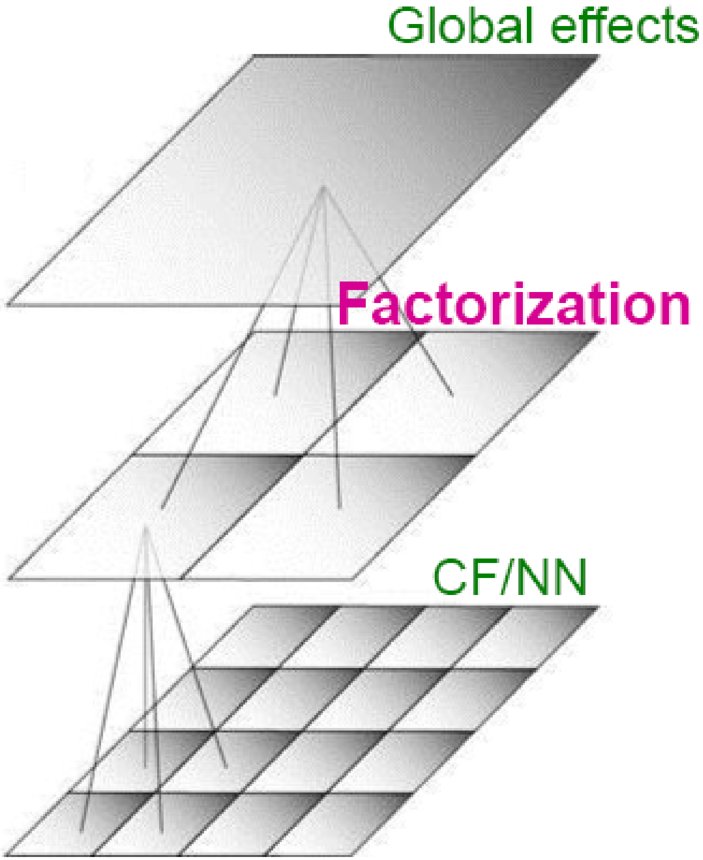
else $\dfrac{\partial}{\partial w_{ij}} = \mathbf{0}$

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

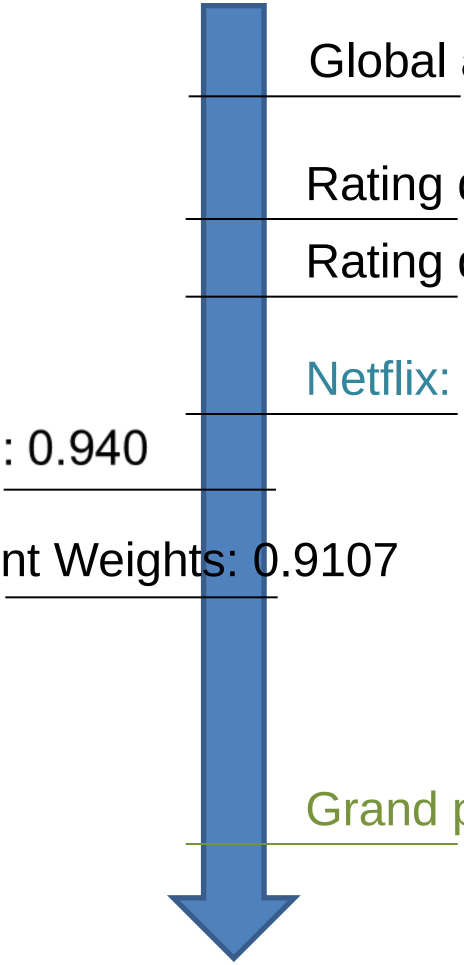
```
while |w_new - w_old| > ε:
    w_old = w_new
    w_new = w_old - η · ∇w_old
```

Interpolation Weights

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



Performance



Global average: 1.1296

Rating deviation of user included: 1.0651

Rating deviation of movie included: 1.0533

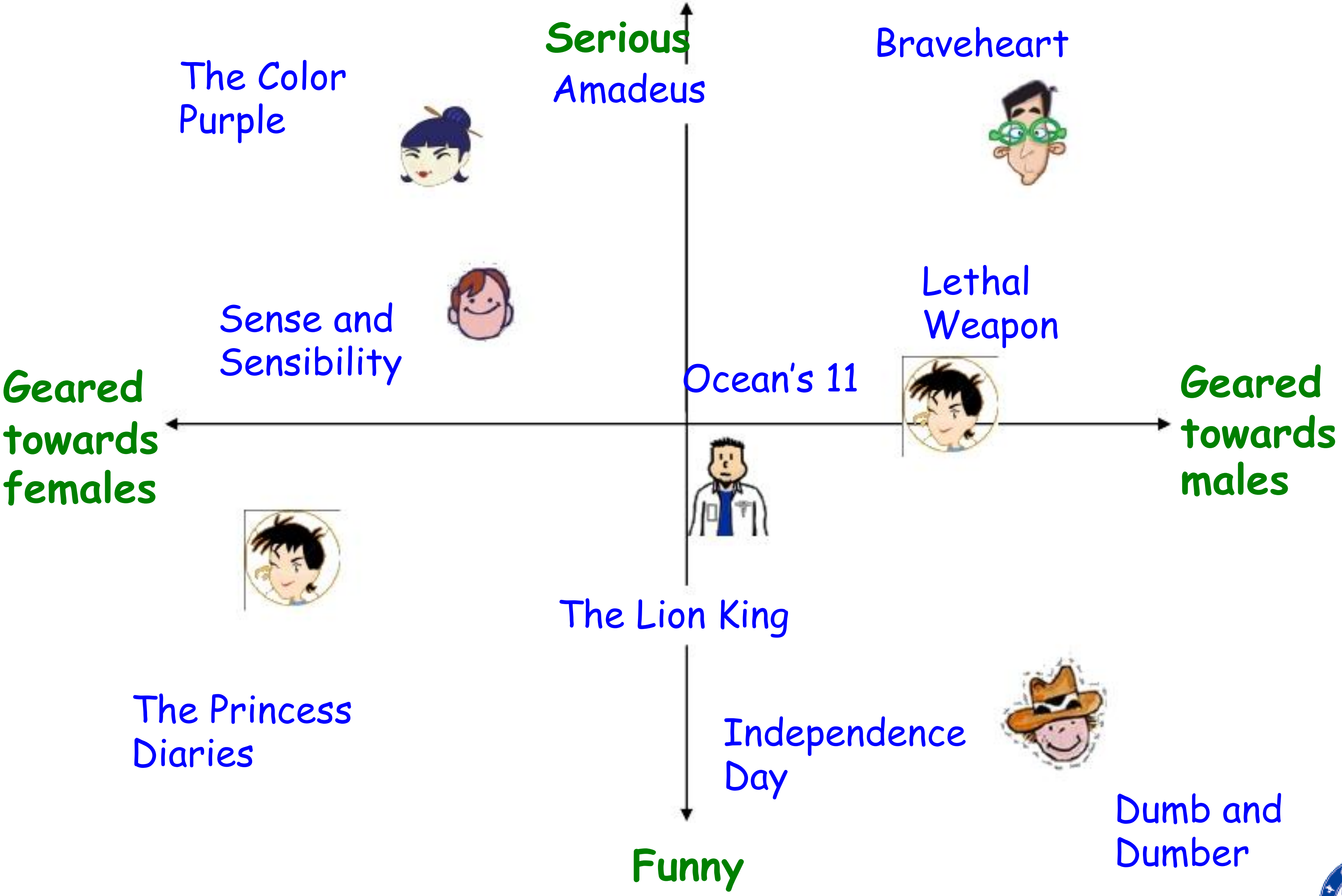
Netflix: 0.9514

Neighborhood model (k=20): 0.940

Neighborhood model + Learnt Weights: 0.9107

Grand prize: 0.8563

Latent Factor Models (e.g., SVD)



Serious

Funny

Geared towards females

Geared towards males

The Color Purple

Amadeus

Braveheart

Sense and Sensibility

Lethal Weapon

Ocean's 11

The Lion King

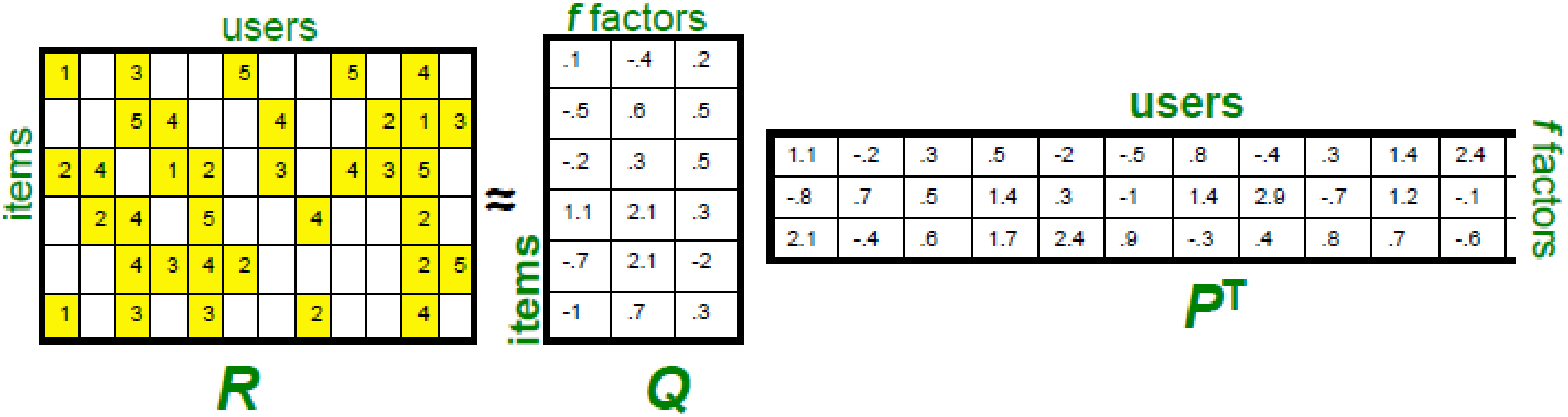
The Princess Diaries

Independence Day

Dumb and Dumber

Latent Factor Models

- "SVD" on Netflix data: R≈ Q·P^T



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

- For now let's assume we can approximate the rating matrix R as a product of "thin" Q·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

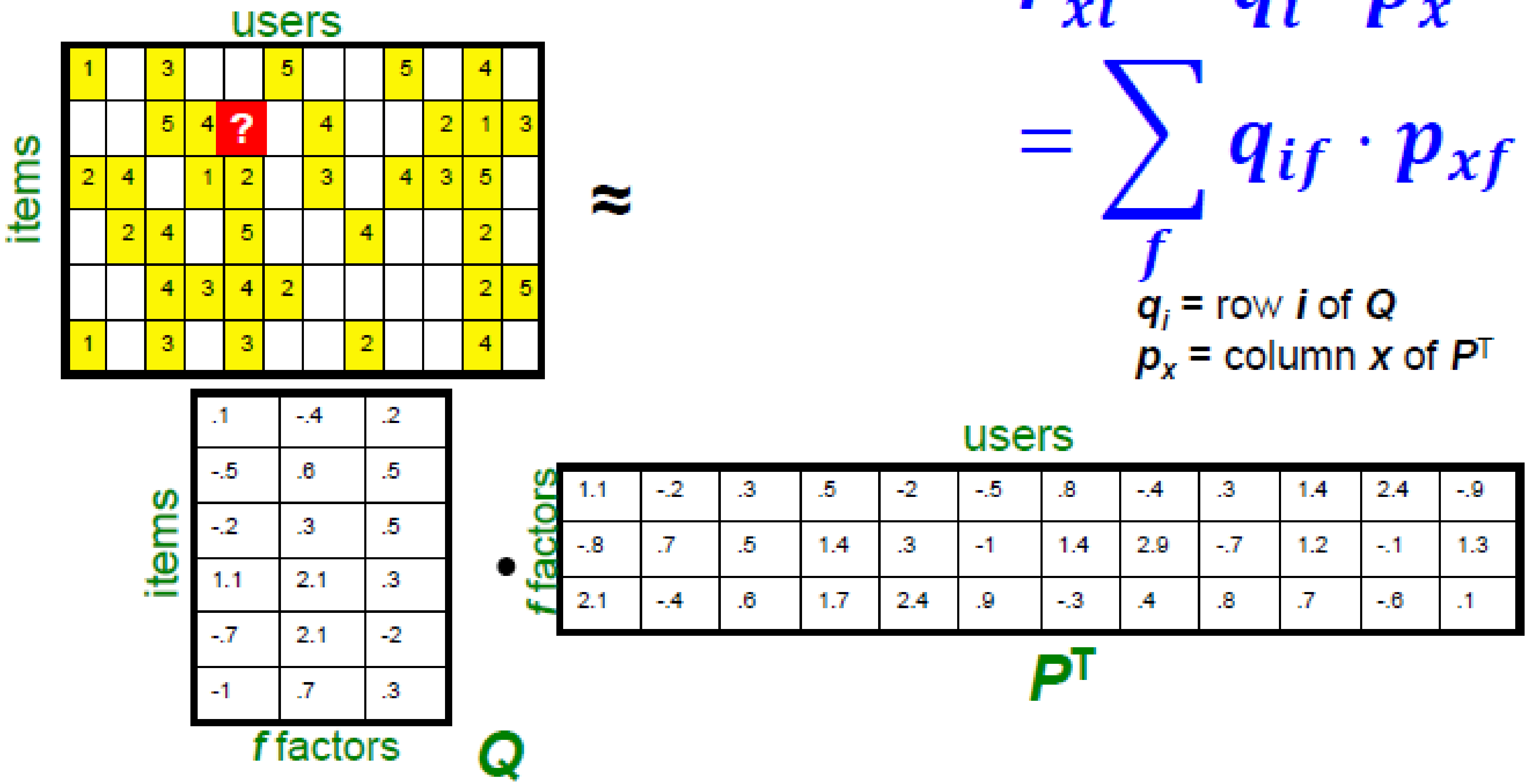
- How to estimate the missing rating of user x for item i?

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

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

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 (items × f factors)

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

•

P^T (f factors × users)

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

Ratings as Products of Factors

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

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

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

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

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

$\approx$

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

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

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

$\bullet$

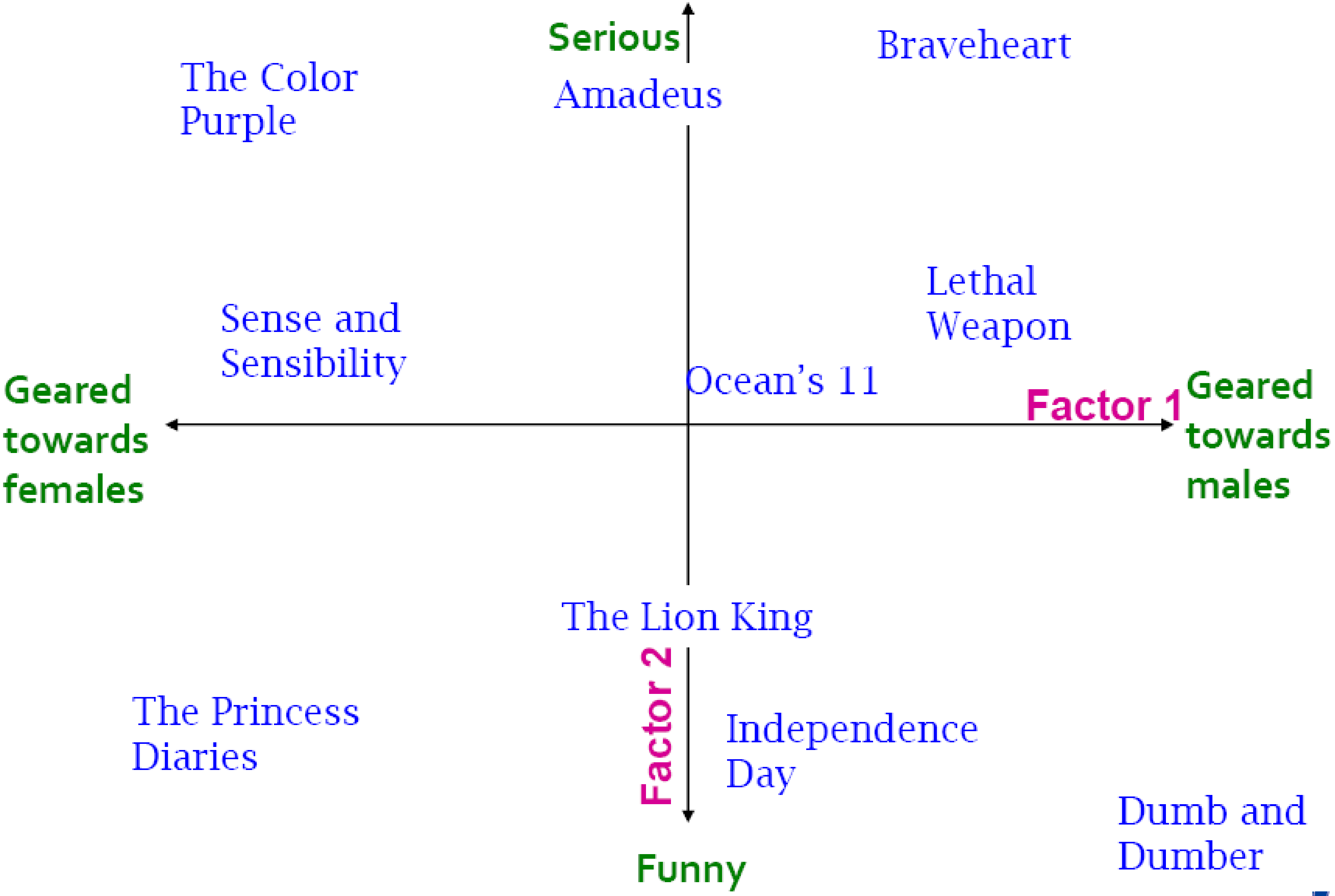
users

f factors

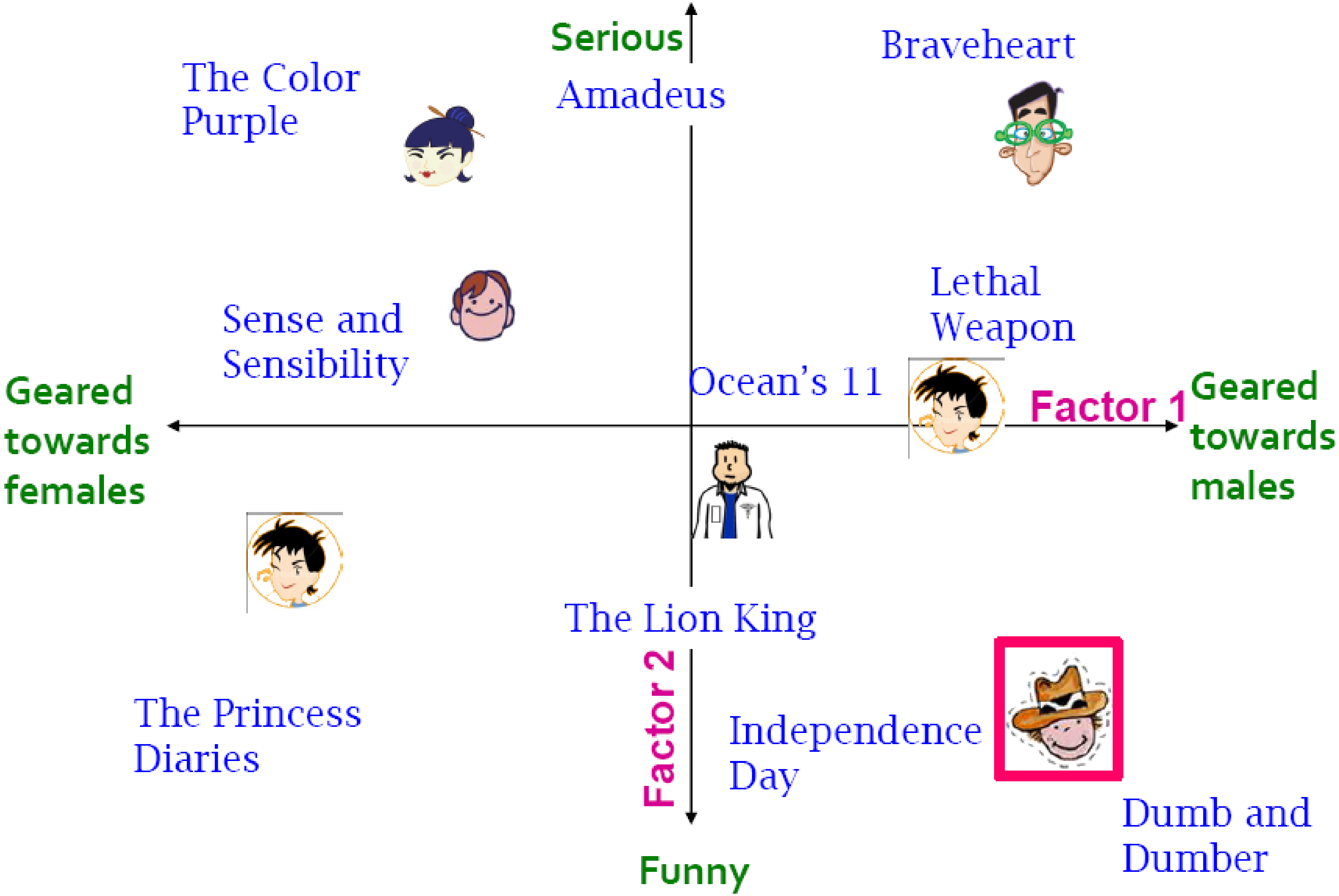
1.1	-.2	.3	.5	-2	-.5	.8	-.4	.3	1.4	2.4	-.9
-.8	.7	.5	1.4	.3	-1	1.4	2.9	-.7	1.2	-.1	1.3
2.1	-.4	.6	1.7	2.4	.9	-.3	.4	.8	.7	-.6	.1

P^T

Latent Factor Models

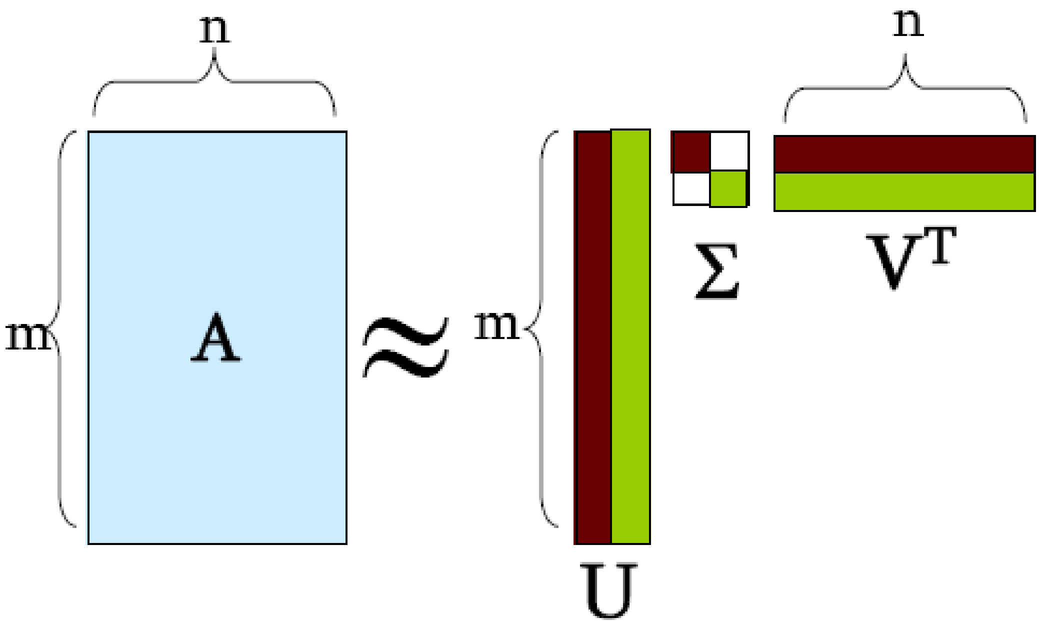


Latent Factor Models



Recap: SVD

- **Remember SVD:**
  - **A**: Input data matrix
  - **U**: Left singular vecs
  - **V**: Right singular vecs
  - Σ: Singular values
  - **SVD gives minimum reconstruction error (SSE!)**



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

The sum goes over all entries.
But our ***R*** has missing entries!

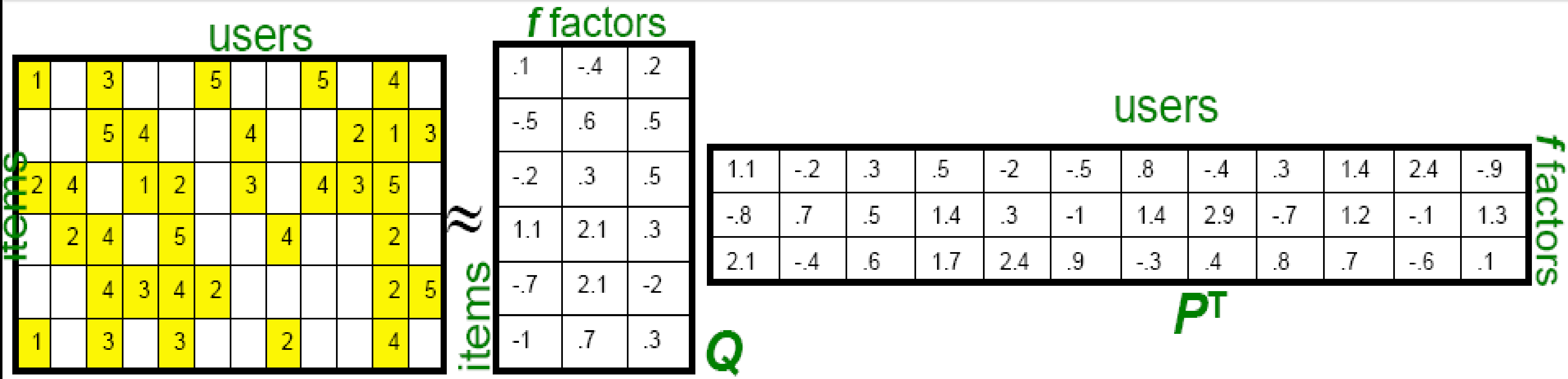
- So in our case, "SVD" on Netflix data: ***R* ≈ *Q* · *P*^{T}**

***A* = *R*, *Q* = *U*, *P*^{T} = Σ *V*^{T}**

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

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

Latent Factor Models



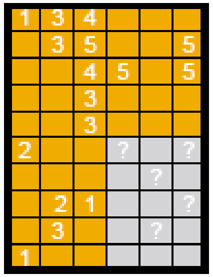
1		3			5			5		4	
		5	4			4			2	1	3
2	4		1	2		3		4	3	5	
	2	4		5			4			2	
		4	3	4	2					2	5
1		3		3			2			4	

.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

- **SVD isn't defined when entries are missing!**
- **Use specialized methods to find *P*, *Q***
  - $\min_{P,Q} \sum_{(i,x)\in \mathrm{R}} (r_{xi} - q_i \cdot p_x^T)^2$ $\qquad \hat{r}_{xi} = q_i \cdot p_x^T$
  - **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

Dealing With Missing Entries



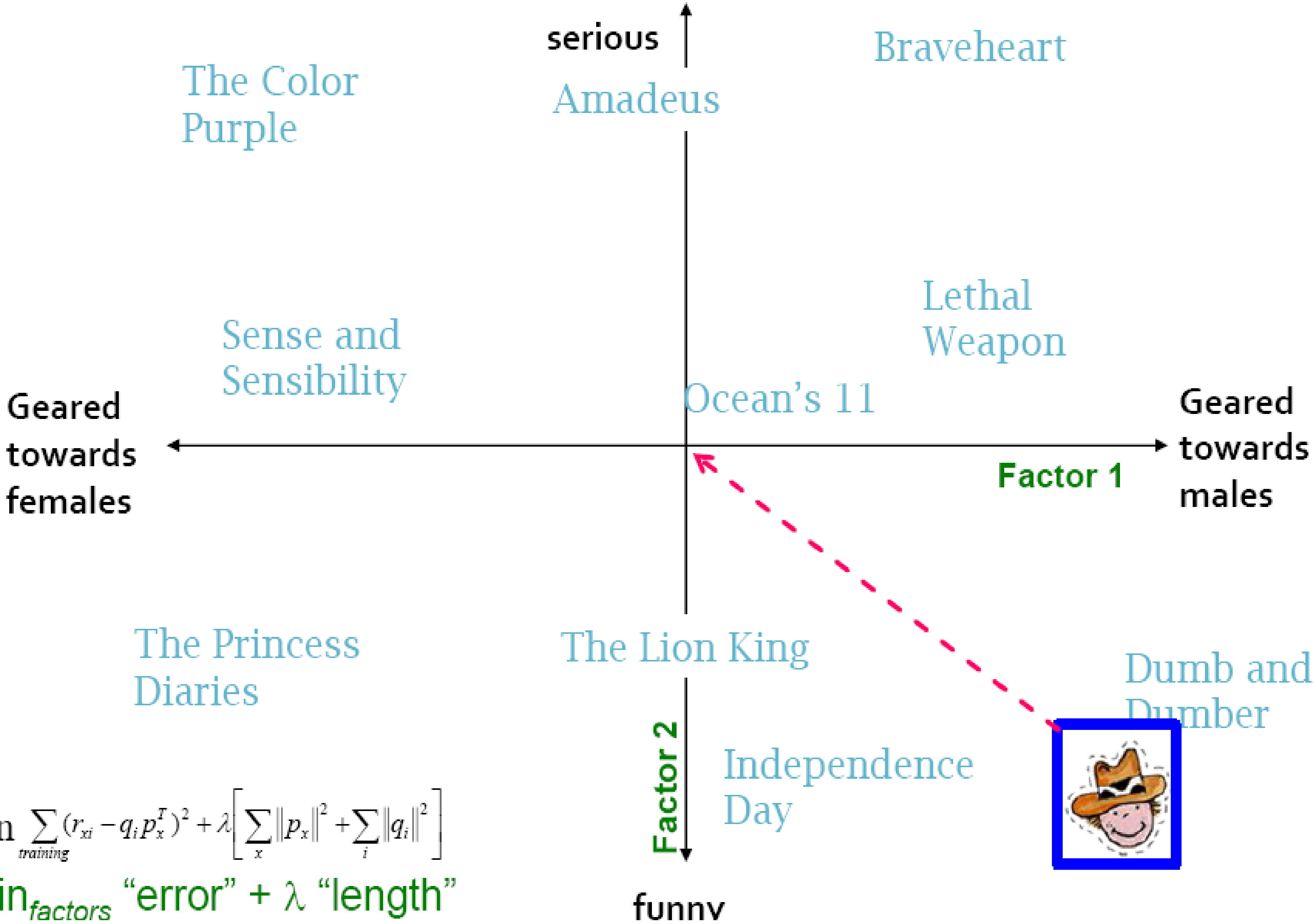
1	3	4			
	3	5			5
		4	5		5
		3			
		3			
2			?		?
				?	
	2	1			?
	3			?	
1					

- 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

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

λ… regularization parameter

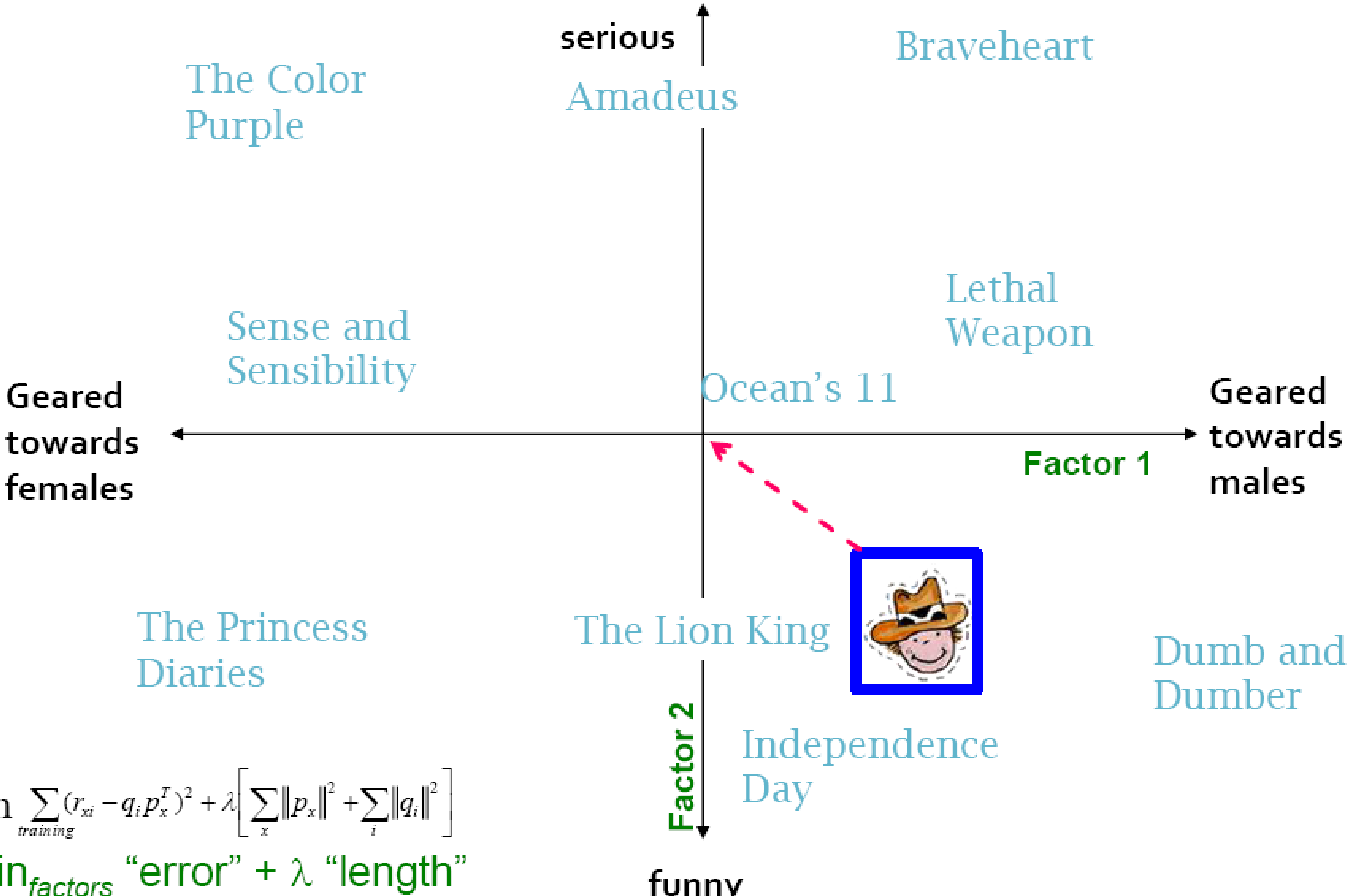
The Effect of Regularization



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

$\min_{factors}$ "error" + λ "length"

The Effect of Regularization



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

$\min_{factors}$ "error" + λ "length"

Stochastic Gradient Descent

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

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

- **Gradient decent:**
  - Initialize ***P*** and ***Q*** (using SVD, pretend missing ratings are 0)
  - Do gradient descent:
    - $\boldsymbol{P} \leftarrow \boldsymbol{P} - \eta \cdot \nabla \mathbf{P}$
    - $\boldsymbol{Q} \leftarrow \boldsymbol{Q} - \eta \cdot \nabla \mathbf{Q}$
    - Where ***∇Q*** is gradient/derivative of matrix ***Q***:
      $\nabla Q = [\nabla q_{if}]$ and $\nabla q_{if} = \sum_{x,i} -2(r_{xi} - q_i p_x^T) p_{xf} + 2\lambda q_{if}$
      - Here $\boldsymbol{q_{if}}$ is entry ***f*** of row ***q***$_i$ of matrix ***Q***

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

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

Stochastic Gradient Descent

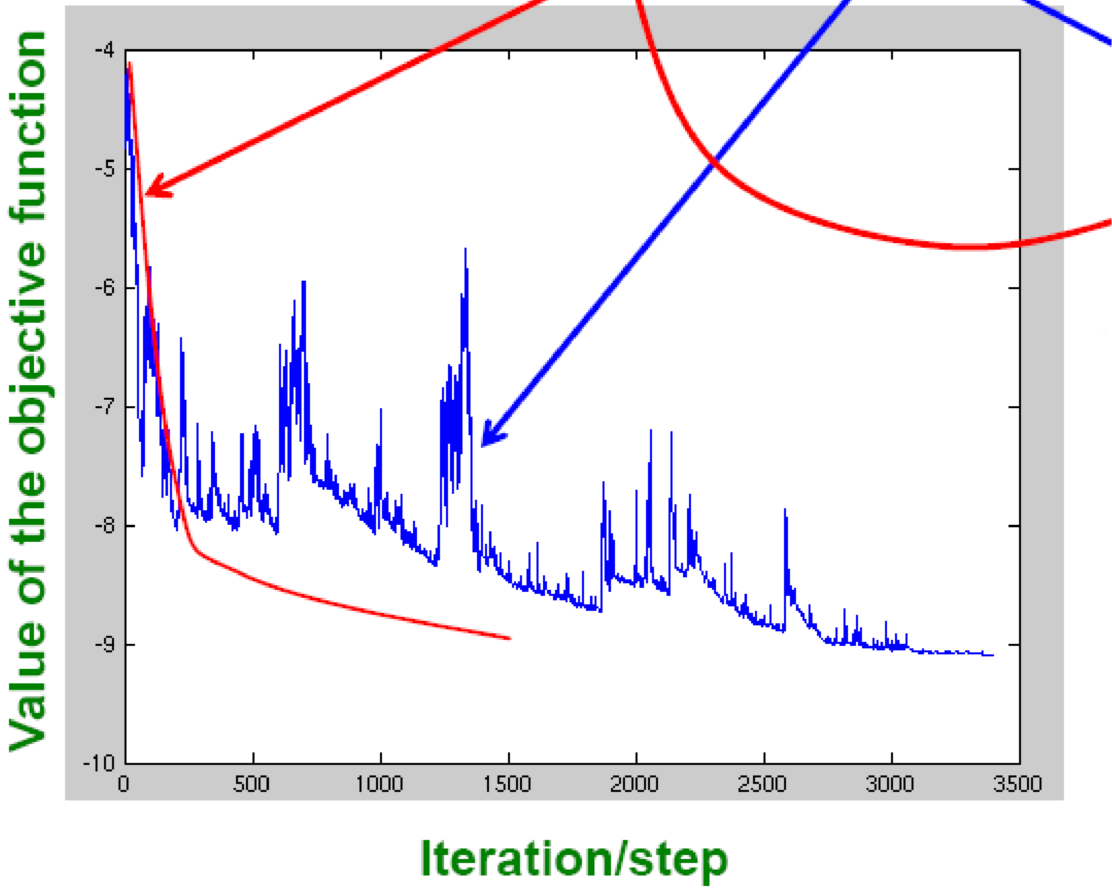
- **Gradient Descent (GD) vs. Stochastic GD**
  - **Observation:** $\nabla Q = [\nabla q_{if}]$ where

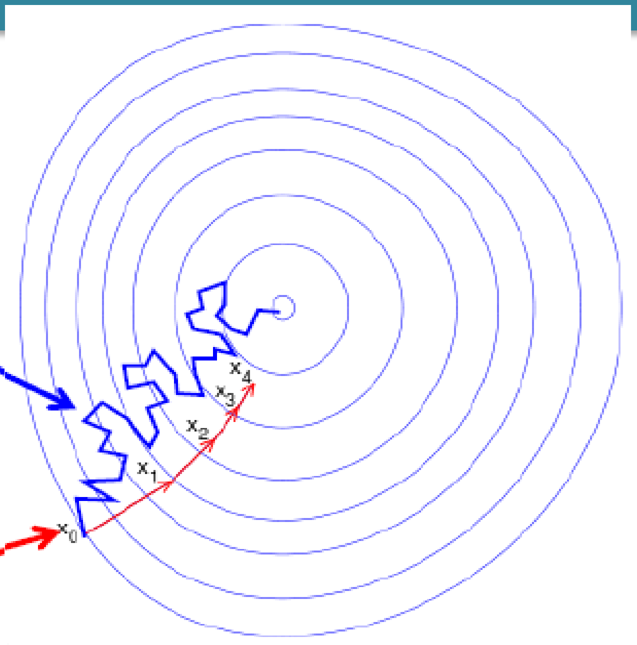
    $$\nabla q_{if} = \sum_{x,i} -2(r_{xi} - q_{if}p_{xf})p_{xf} + 2\lambda q_{if} = \sum_{x,i} \nabla \boldsymbol{Q}\,(\boldsymbol{r}_{xi})$$

    - Here $\boldsymbol{q}_{if}$ is entry $\boldsymbol{f}$ of row $\boldsymbol{q}_i$ of matrix $\boldsymbol{Q}$
  - $\boldsymbol{Q} = \boldsymbol{Q} - \eta\nabla\boldsymbol{Q} = \boldsymbol{Q} - \eta\left[\sum_{x,i} \nabla \boldsymbol{Q}\,(\boldsymbol{r}_{xi})\right]$
  - **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:** $\boldsymbol{Q} \leftarrow \boldsymbol{Q} - \eta\nabla\boldsymbol{Q}(\boldsymbol{r}_{xi})$
  - **Faster convergence!**
    - Need more steps but each step is computed much faster

Stochastic Gradient Descent

- **Convergence of GD vs. SGD**





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

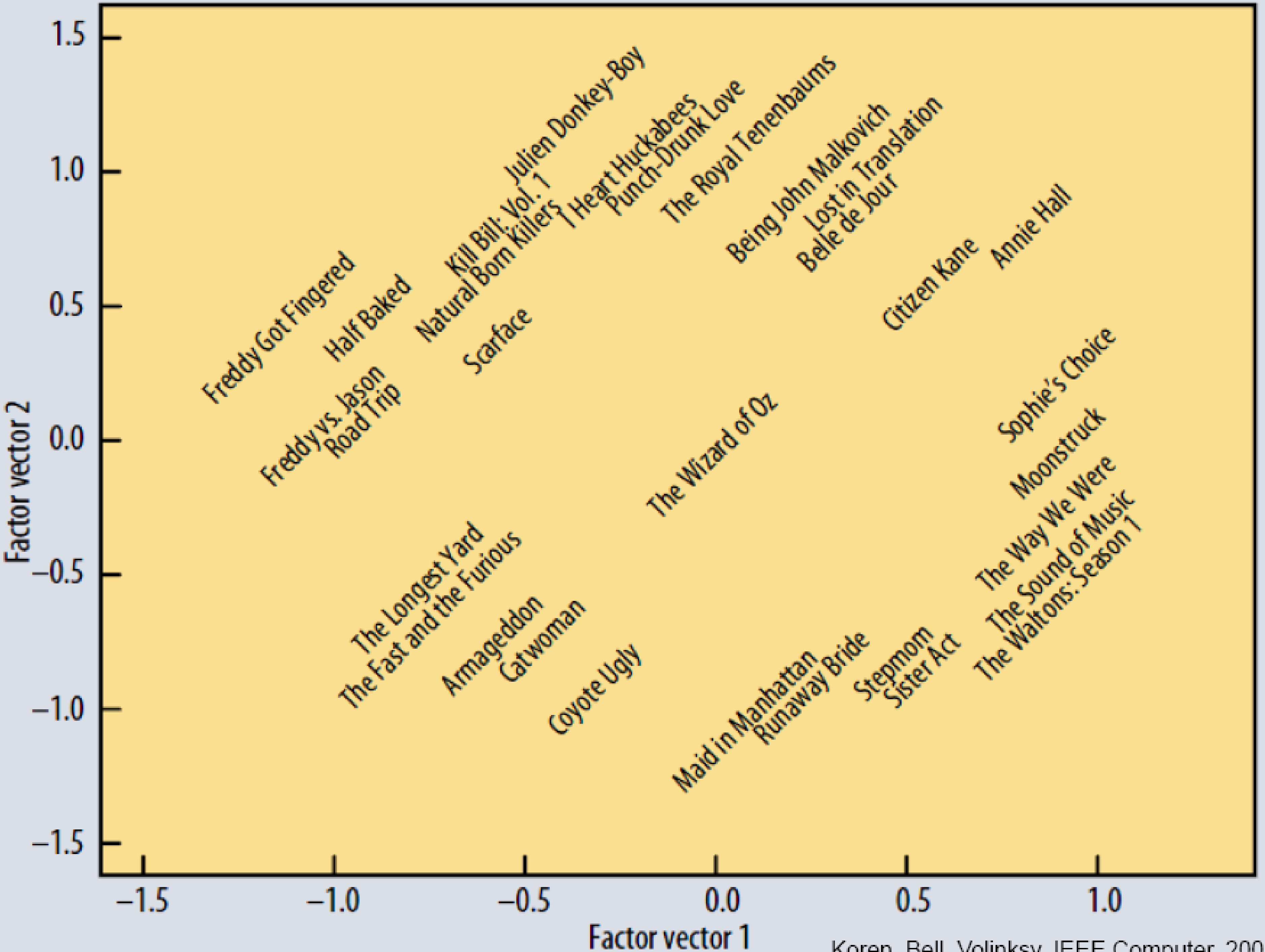
Stochastic Gradient Descent

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

    For each *r*_*xi*_:

    - $\varepsilon_{xi} = r_{xi} - q_i \cdot p_x^T$ (derivative of the "error")
    - $q_i \leftarrow q_i + \eta\,(\varepsilon_{xi}\, p_x - \lambda\, q_i)$ (update equation)
    - $p_x \leftarrow p_x + \eta\,(\varepsilon_{xi}\, q_i - \lambda\, p_x)$ (update equation)

    η … learning rate
- **2 for loops:**
  - For until convergence:
    - For each **r**$_{\mathbf{xi}}$
      - Compute gradient, do a "step"



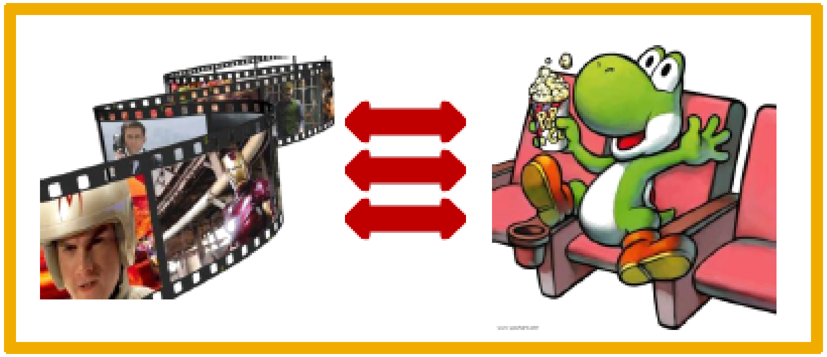
Koren, Bell, Volinksy, IEEE Computer, 2009

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

- 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

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

- **Example:**
  - Mean rating: ***μ = 3.7***
  - You are a critical reviewer: your ratings are 1 star lower than the mean: ***b_x = -1***
  - Star Wars gets a mean rating of *0.5* higher than average movie: ***b_i = + 0.5***
  - Predicted rating for you on Star Wars:
    = 3.7 - 1 + 0.5 = 3.2

Fitting the New Model

- **Solve:**

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

goodness of fit

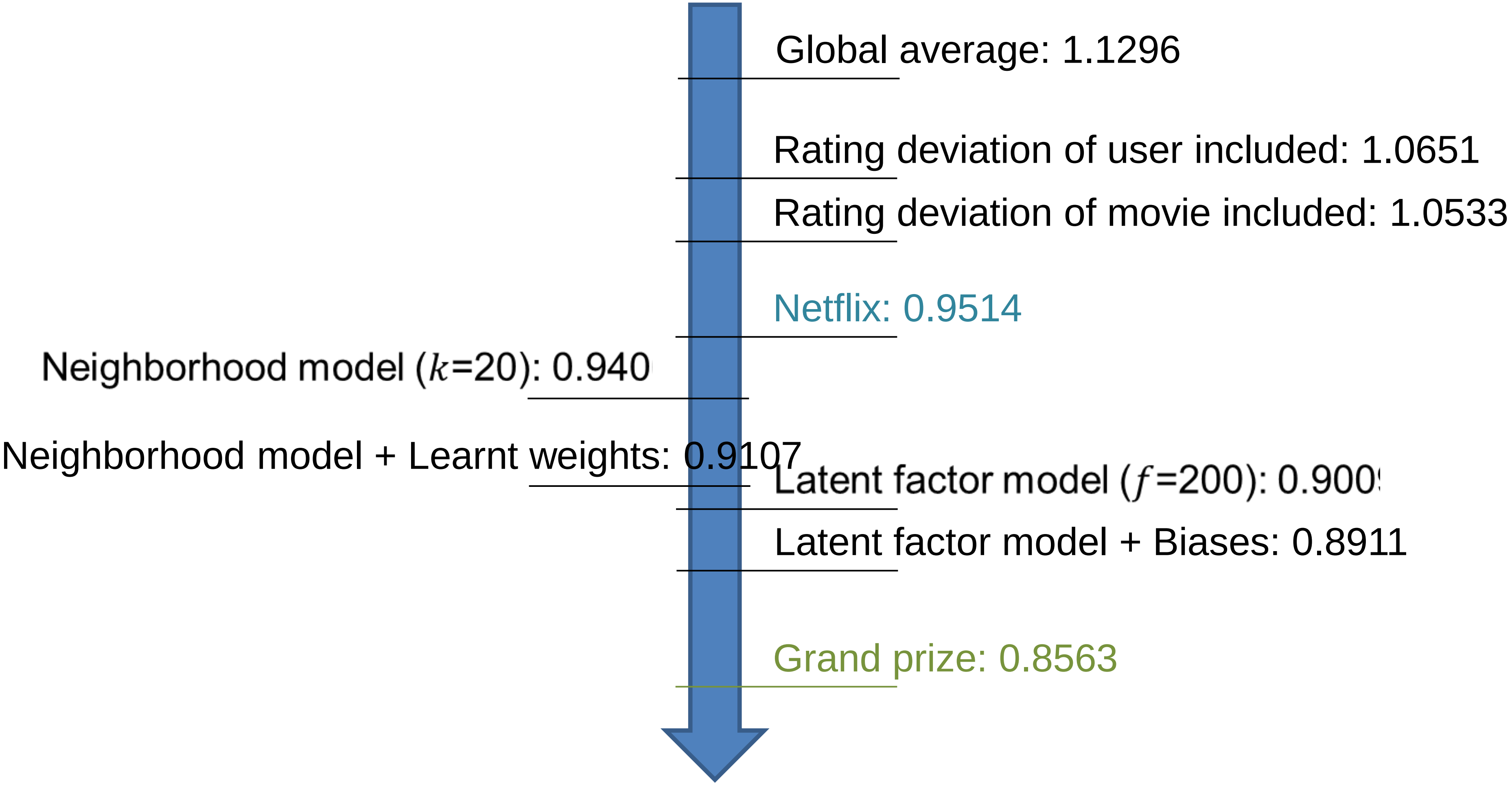
$$+ \lambda \left(\sum_i \|q_i\|^2 + \sum_x \|p_x\|^2 + \sum_x \|b_x\|^2 + \sum_i \|b_i\|^2 \right)$$

regularization

λ is selected via grid-search on a validation set

- **Stochastic gradient decent to find parameters**
  - **Note:** Both biases $\boldsymbol{b}_u$, $\boldsymbol{b}_i$ as well as interactions $\boldsymbol{q}_i$, $\boldsymbol{p}_u$ are treated as parameters (we estimate them)

Performance of Various Methods



Global average: 1.1296

Rating deviation of user included: 1.0651

Rating deviation of movie included: 1.0533

Netflix: 0.9514

Neighborhood model (k=20): 0.940

Neighborhood model + Learnt weights: 0.9107

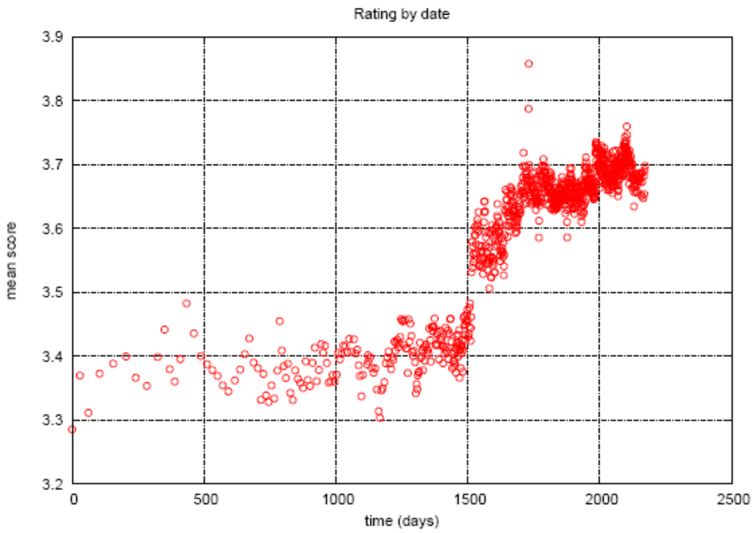
Latent factor model (f=200): 0.900

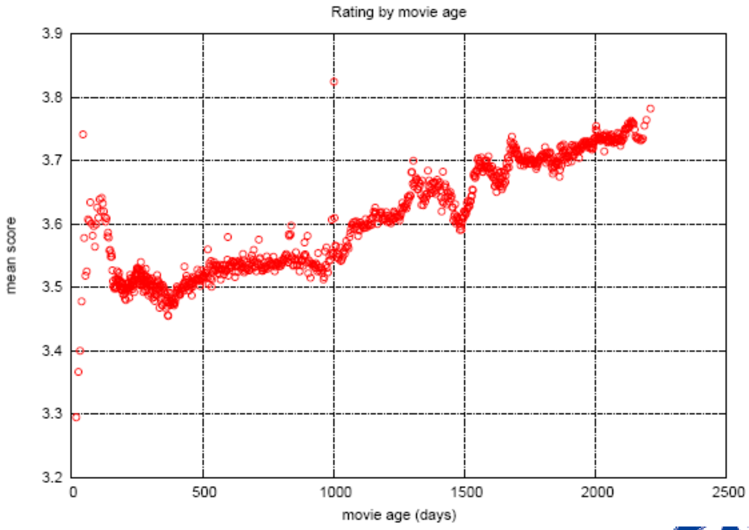
Latent factor model + Biases: 0.8911

Grand prize: 0.8563

Temporal Biases of Users

- Sudden rise in the average movie rating (early 2004)
- Older movies are just inherently better than newer ones
- Add time dependence to biases
  - $r'_{ui} = \mu + b_u(t) + b_i(t) + U_u V_i'$





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

Temporal Biases of Users

- **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 $\boldsymbol{b_u}$ and $\boldsymbol{b_i}$ to depend on time
  - **(1)** Parameterize time-dependence by linear trends

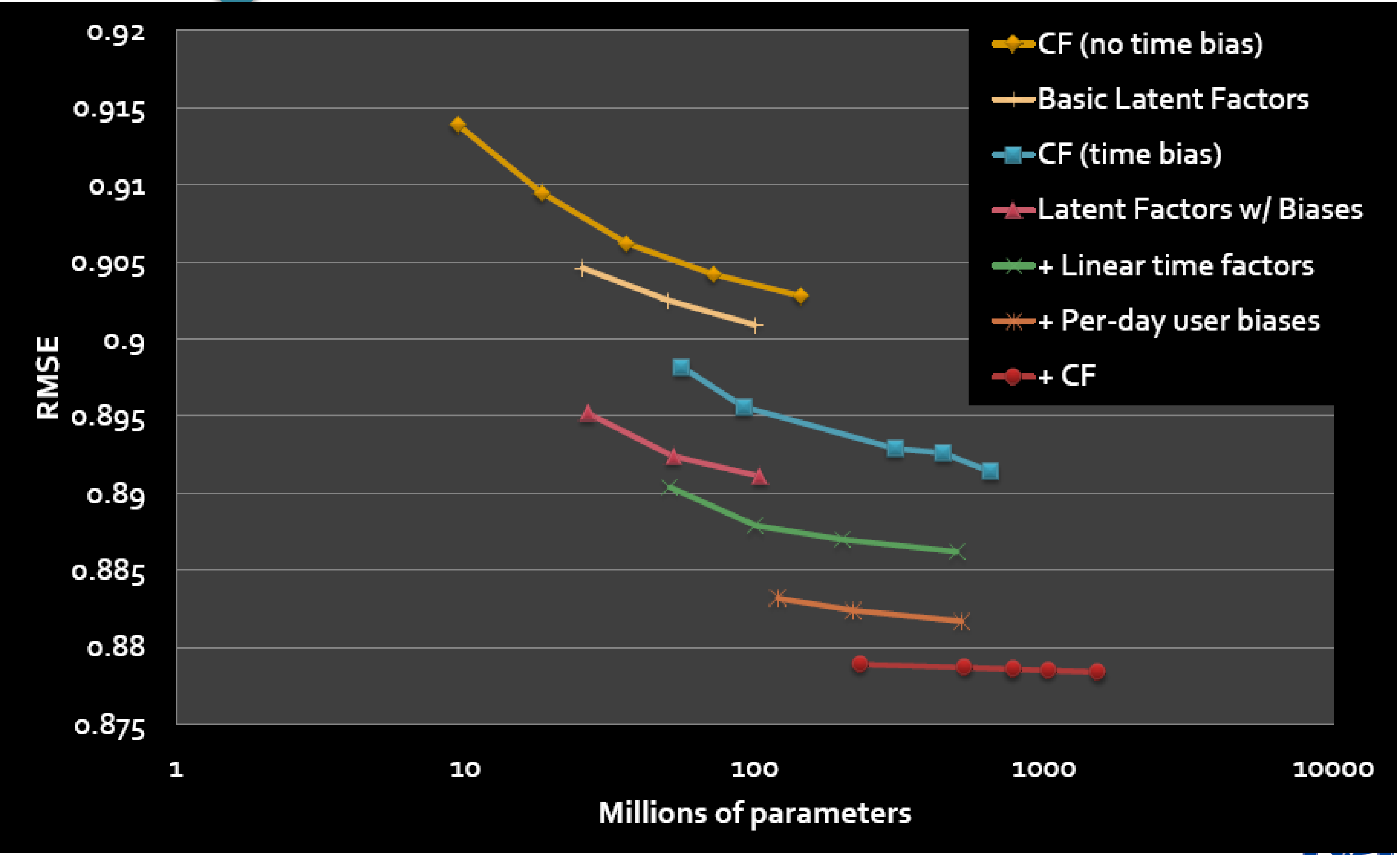
    (2) Each bin corresponds to 10 consecutive weeks

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

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

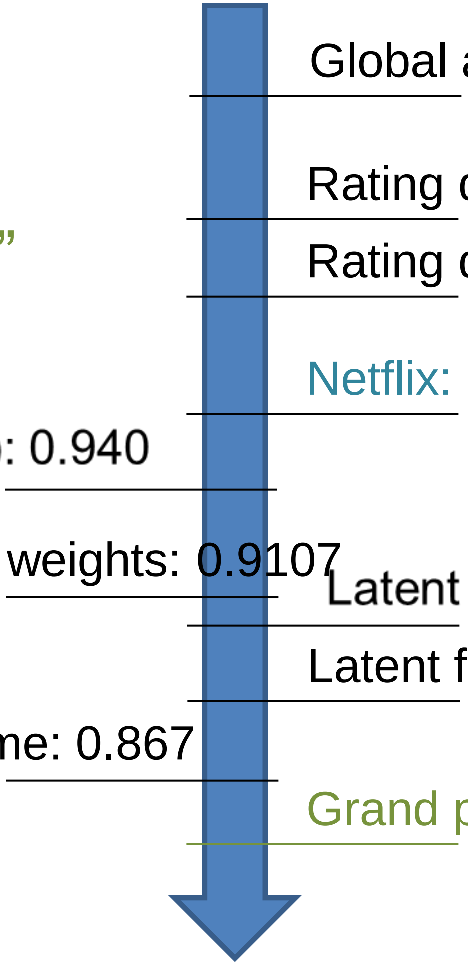
Y. Koren, Collaborative filtering with temporal dynamics, KDD '09

Adding Temporal Effects



Performance

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



Global average: 1.1296

Rating deviation of user included: 1.0651

Rating deviation of movie included: 1.0533

Netflix: 0.9514

Neighborhood model (k=20): 0.940

Neighborhood model + Learnt weights: 0.9107

Latent factor model (f=200): 0.900

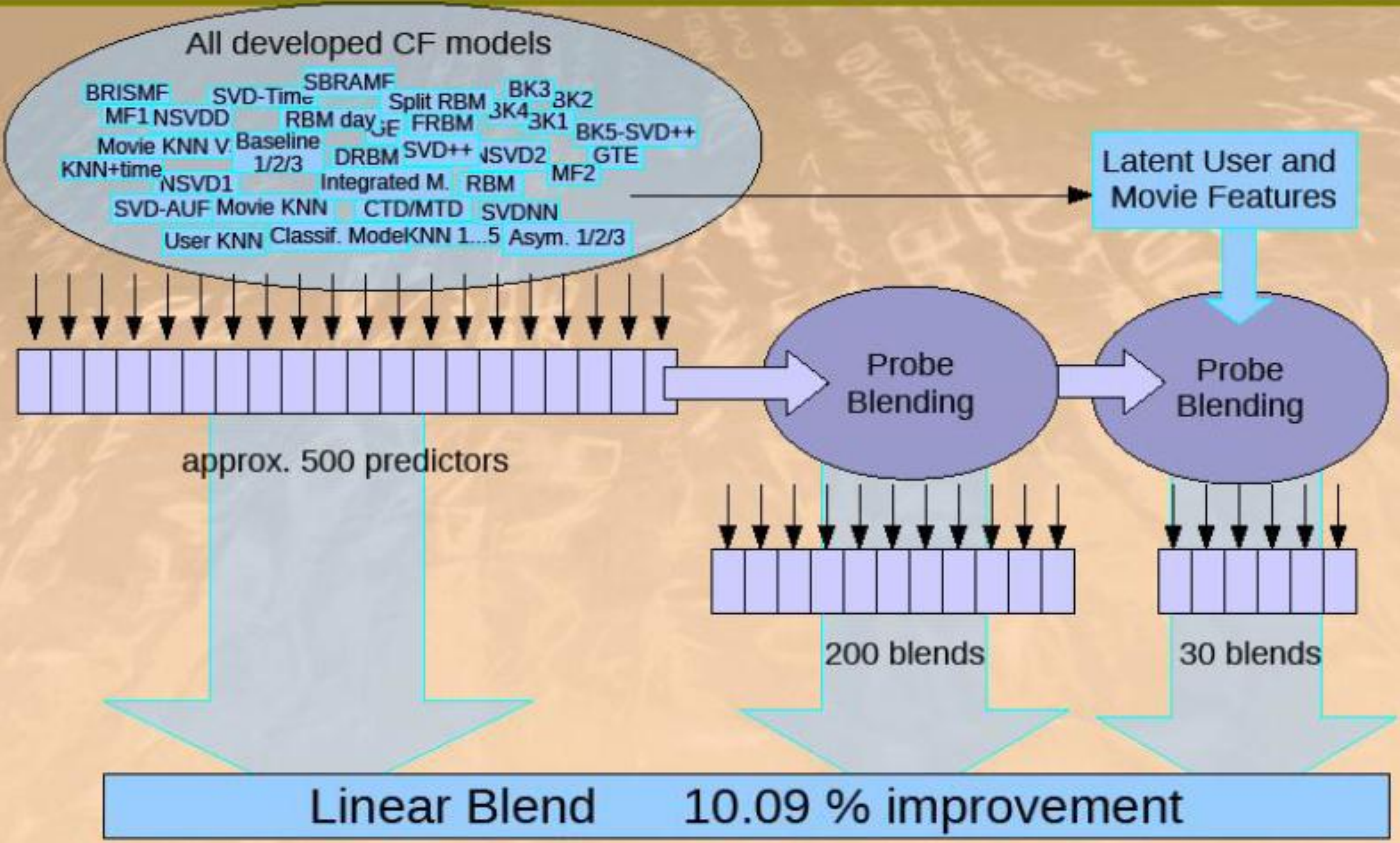
Latent factor model + Biases: 0.8911

Latent factors + Biases + Time: 0.867

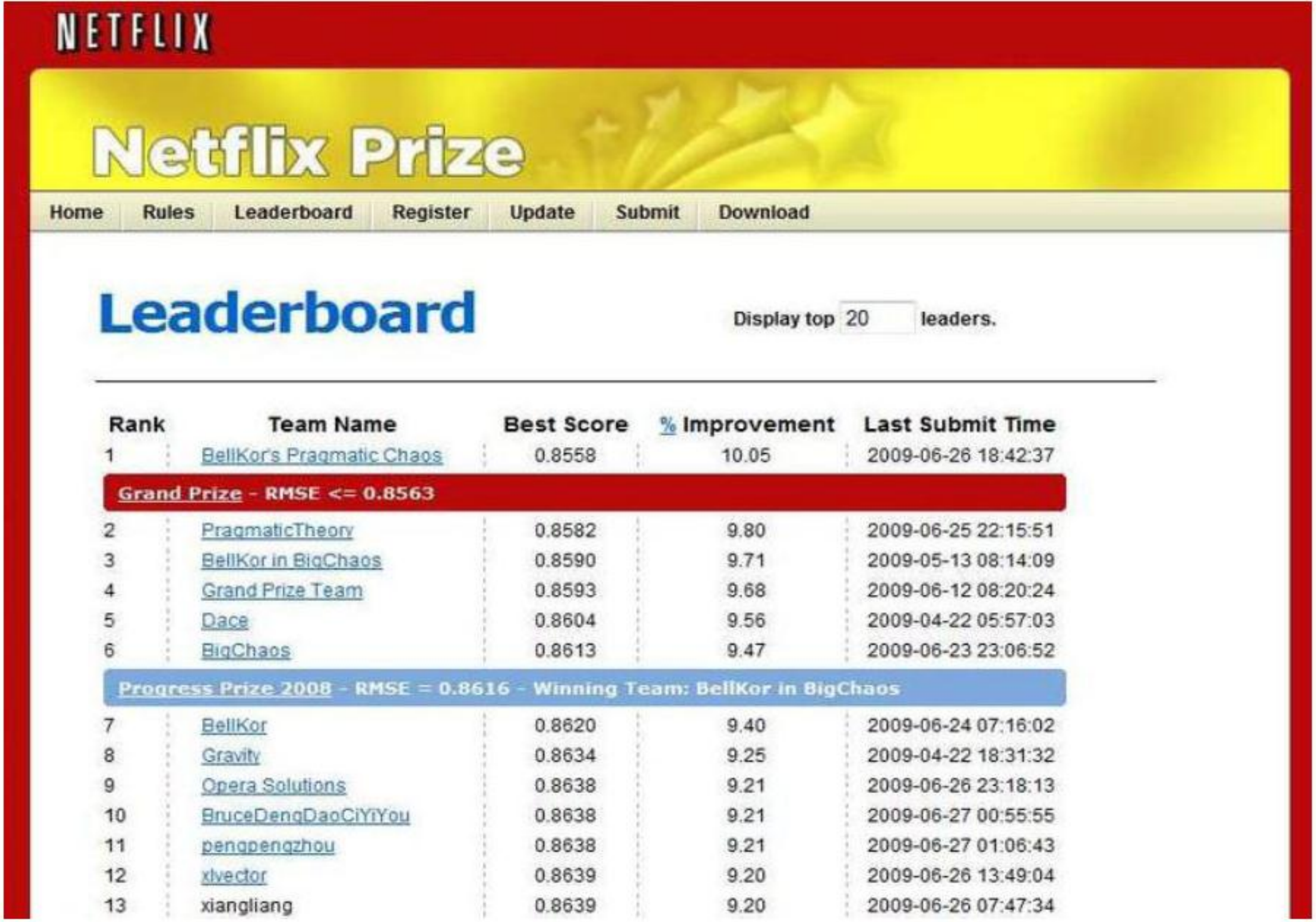
Grand prize: 0.8563

The big picture

Solution of BellKor's Pragmatic Chaos



Standing on June 26th 2009



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

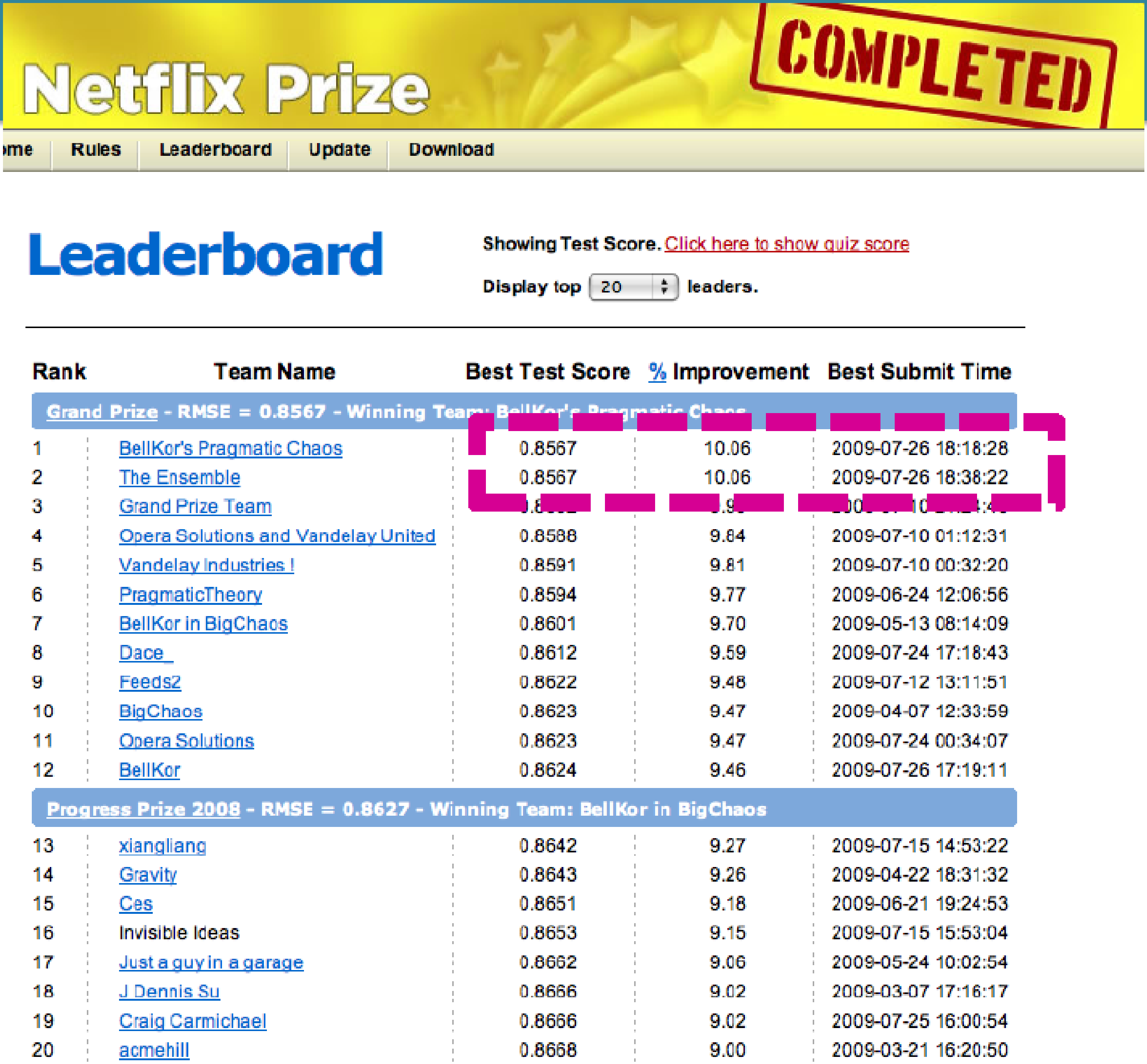
June 26th submission triggers 30-day "last call"

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

Million $ Awarded Sept. 21th 2009

Tell me and I forget.
Show me and I remember.
Involve me and I understand.

Thank you!   Q&A