MIE524 Data Mining

Recommender Systems: Latent Factor Models

Some slides modified from:
Leskovec, Rajaraman, Ullman (http://www.mmds.org), Dimitris Sacharidis

Announcements

- **Lab tomorrow:**
  - Quiz 4
  - Lab for Assignment 5 (no quiz for A5)
- **Grades for Quiz 3 and Assignment 3 will be posted today**
  - Grades for A2 will be posted soon
- **Practice questions for the final will be posted early next week**
- **Office hours for the final exam:**
  - Wednesday December 4 after (last) lecture
  - Friday December 6: 11am-12pm, 1-2pm

Why do we need recommender systems?

- **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 systems play crucial role

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, …

Content-based vs. Collaborative Filtering

- **Content-based algorithms**
  - Items have profiles (e.g., movie’s genre, director, actors, plot, year)
  - Recommend items to customers similar to previous items rated highly

- **Collaborative Filtering (covered today)**
  - Does not require profiles (features) for items or users
  - Instead, only requires user-item interaction data
    - User A clicks/likes/rates item B
  - Performs very well in practice

Formal Model

- ***X*** = set of **Customers**
- ***S*** = set of **Items**

- **Utility function** ***u*****:** ***X*** **×** ***S*** **→** ***R***
  - ***R*** = set of interaction feedbacks
  - ***Explicit*** **feedback**
    - e.g., 0-5 stars
  - ***Implicit*** **feedback**
    - e.g., user watched movie (0/1)

Case study for this lecture:
<u>Online streaming service</u>
- **Customers = Users**
- **Items = Movies**

Utility Matrix

	Avatar	LOTR	Matrix	Pirates
Alice	5		1	
Bob		4		3
Carol	4			
David				3

Explicit
(focus today)

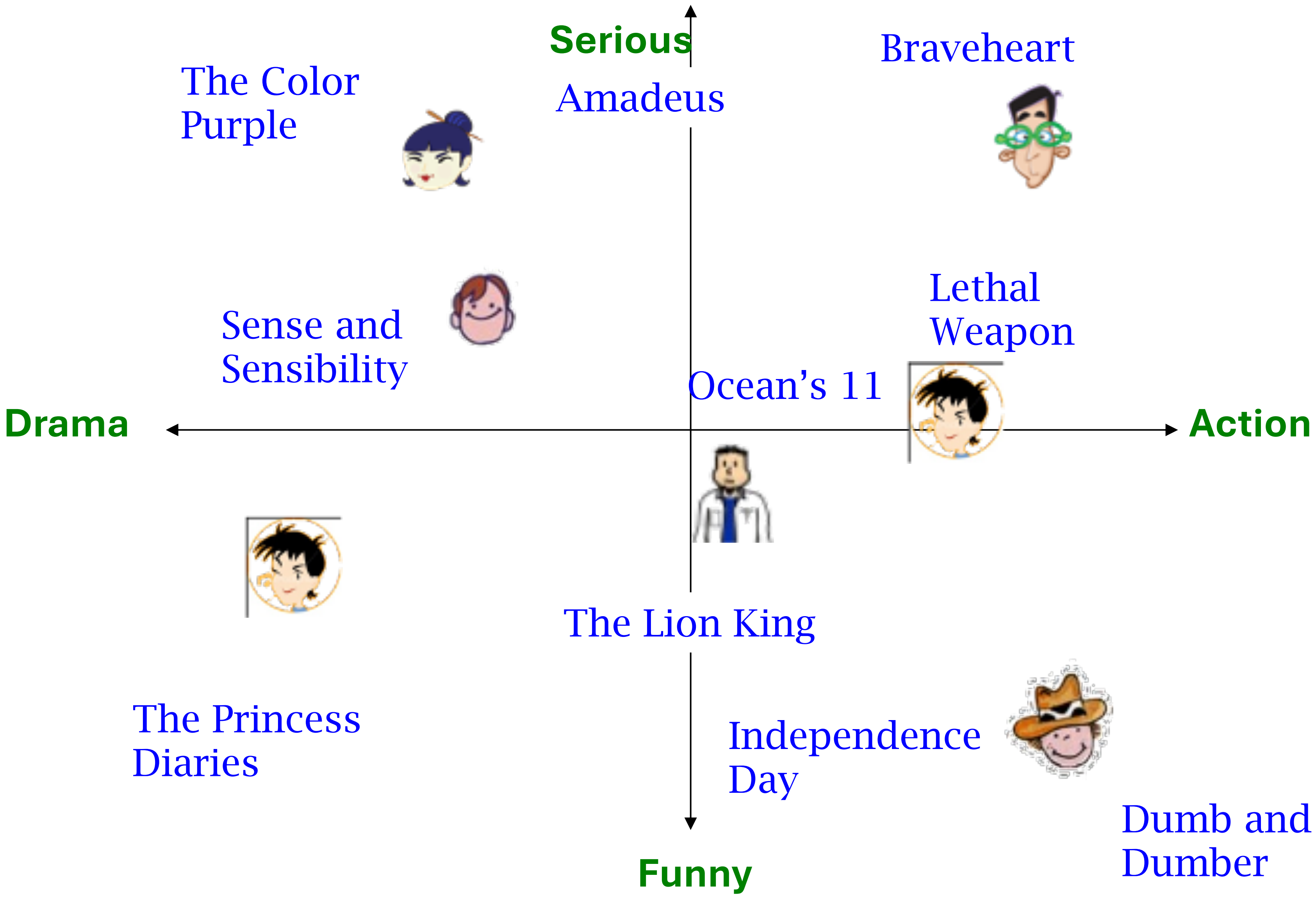
	Avatar	LOTR	Matrix	Pirates
Alice	1	0	1	0
Bob	0	1	0	1
Carol	1	0	0	0
David	0	0	0	1

Implicit

Key Problems

- **Gathering “known” ratings for matrix**
  - How to collect the data in the utility matrix
- **Extrapolate unknown ratings from the known ones**
  - In practice: typically interested in high unknown ratings
    - We are not interested in knowing what you don’t like but what you like
- **Evaluating extrapolation methods**
  - How to measure success/performance of recommendation methods

Intuition: Latent Factor Models



- Users and movies are described by vectors of latent features in a shared vector space
- Use similarity in latent space to measure match

Latent Factor Models

- Approximate the rating matrix **R** as product of matrices $\mathbf{Q} \cdot \mathbf{P}^T$

users (columns) × items (rows)

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

R

≈

factors (columns) × items (rows)

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

Q

users (columns) × factors (rows)

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

$\boldsymbol{P^T}$

Latent Factor Models

- Approximate the rating matrix **R** as product of matrices **Q · P^T**
  - **<u>Problem</u>: R** has missing entries
  - Learn Q and P that minimize the **reconstruction error** on known ratings and ignore the missing values

users (columns) / items (rows)

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

R

≈

factors (columns) / items (rows)

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

Q

users (columns) / factors (rows)

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

Latent Factor Models

- Approximate the rating matrix **R** as product of matrices **Q · P^T**
  - **<u>Problem</u>: R** has missing entries
  - Learn Q and P that minimize the **reconstruction error** on known ratings and ignore the missing values

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

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

R (items × users)

1		3			5			5		4	
		$\boxed{5}$	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 × factors)

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

P^T (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

Latent Factor Models

- **How to estimate missing ratings?**
- Estimate using latent factors $\hat{r}_{xi} = q_i \cdot p_x = \sum_f q_{if} \cdot p_{xf}$

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	

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

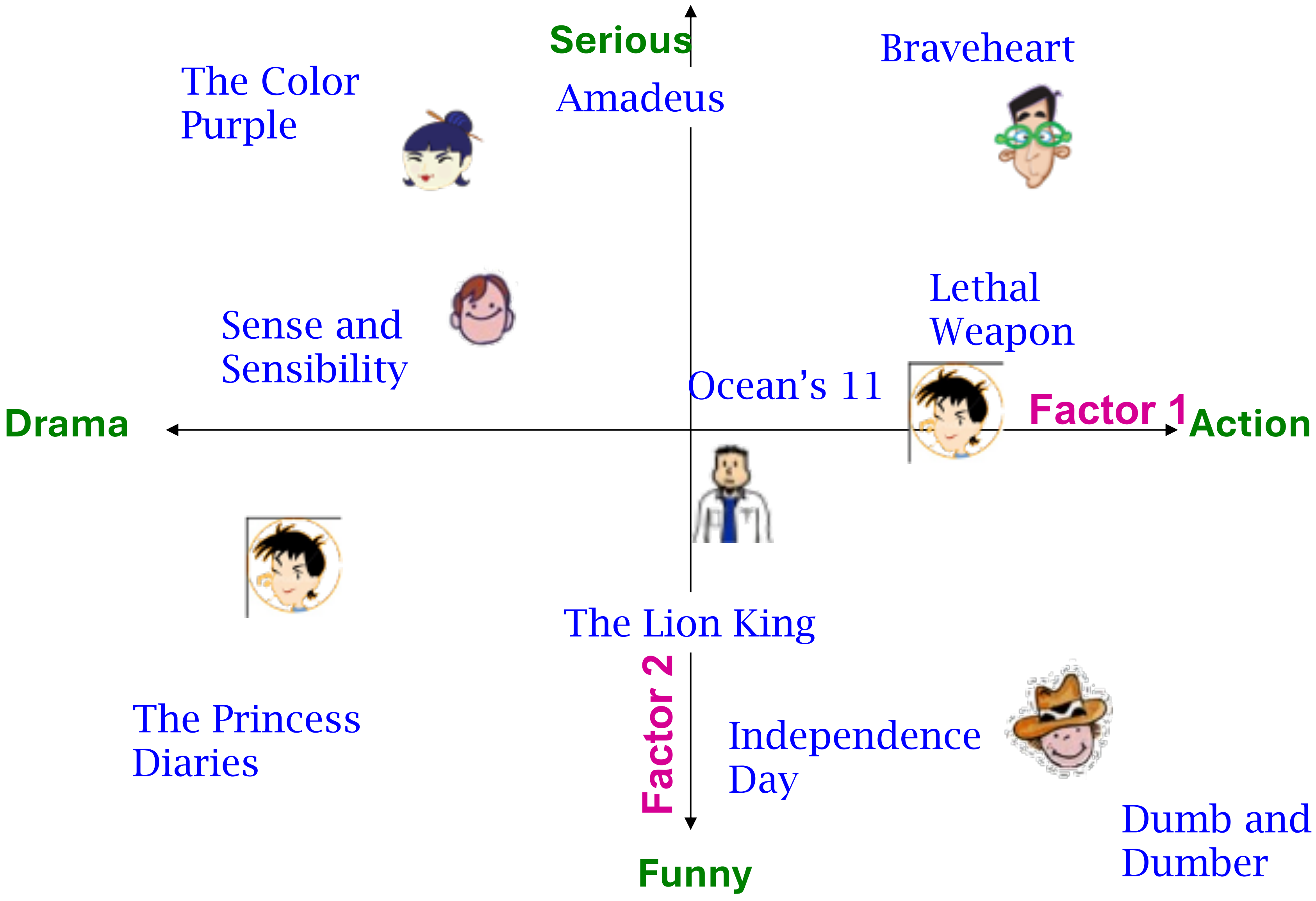
users

factors

1.1	-.2	.3	.5	-2	-.5	.8	-.4	.3	1.4	2.4	-.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

Intuition: Latent Factor Models



- Users and movies are described by vectors of latent features in a shared vector space
- Q and P provide the latent representation for each item/user
- Inner (dot) product measures similarity
- Larger inner product between user and movie = higher predicted rating

Latent Factor Models

- How to estimate missing ratings?
- Estimate using latent factors $\hat{r}_{xi} = q_i \cdot p_x = \sum_f q_{if} \cdot p_{xf}$

  q_i = row i of Q

  p_x = column x of P^T

- **<u>Can we use SVD to compute P and Q?</u>**

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

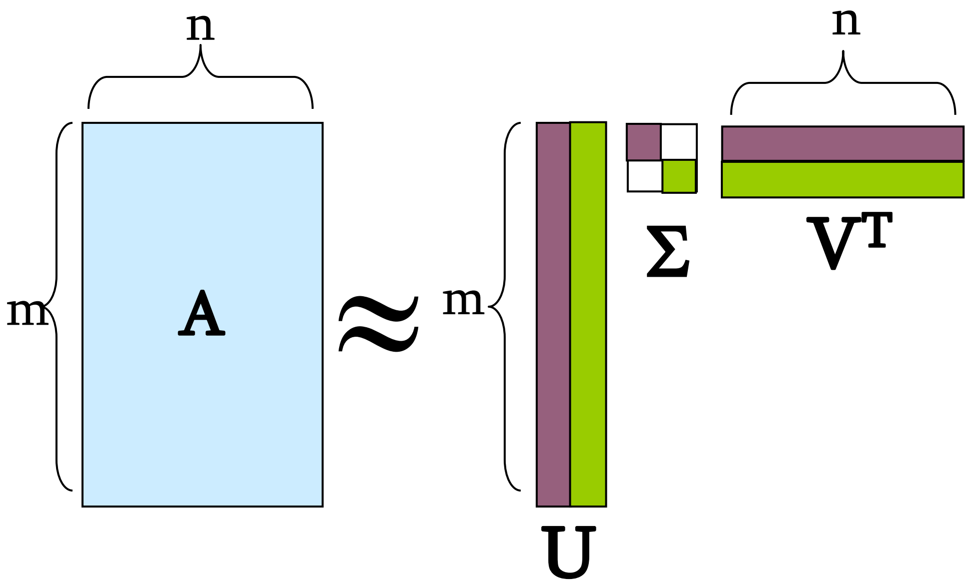
factors

1.1	-.2	.3	.5	-2	-.5	.8	-.4	.3	1.4	2.4	-.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

Reminder: SVD

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



Reminder: SVD

U **is "user-to-concept" similarity matrix**

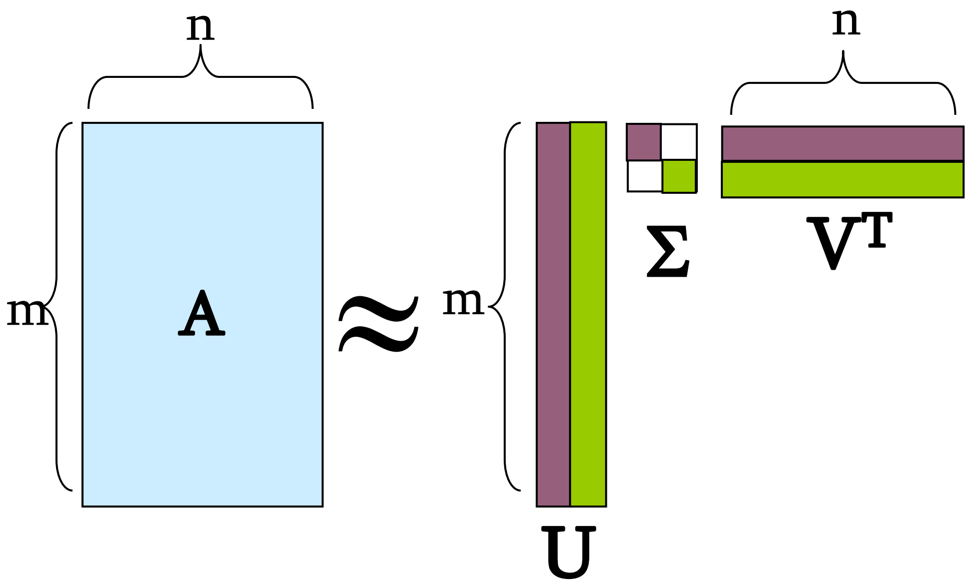
$$
\begin{array}{cc}
 & \begin{array}{ccccc}\text{Matrix} & \text{Alien} & \text{Serenity} & \text{Casablanca} & \text{Amelie}\end{array} \\
\text{Users} & \begin{bmatrix}
\mathbf{1} & \mathbf{1} & \mathbf{1} & 0 & 0 \\
\mathbf{3} & \mathbf{3} & \mathbf{3} & 0 & 0 \\
\mathbf{4} & \mathbf{4} & \mathbf{4} & 0 & 0 \\
\mathbf{5} & \mathbf{5} & \mathbf{5} & 0 & 0 \\
0 & \mathbf{2} & 0 & \mathbf{4} & \mathbf{4} \\
0 & 0 & 0 & \mathbf{5} & \mathbf{5} \\
0 & \mathbf{1} & 0 & \mathbf{2} & \mathbf{2}
\end{bmatrix}
\end{array}
=
\begin{bmatrix}
\mathbf{0.13} & 0.02 & -0.01 \\
\mathbf{0.41} & 0.07 & -0.03 \\
\mathbf{0.55} & 0.09 & -0.04 \\
\mathbf{0.68} & 0.11 & -0.05 \\
0.15 & \mathbf{-0.59} & \mathbf{0.65} \\
0.07 & \mathbf{-0.73} & \mathbf{-0.67} \\
0.07 & \mathbf{-0.29} & \mathbf{0.32}
\end{bmatrix}
\times
\begin{bmatrix}
\mathbf{12.4} & 0 & 0 \\
0 & \mathbf{9.5} & 0 \\
0 & 0 & \mathbf{1.3}
\end{bmatrix}
\times
\begin{bmatrix}
\mathbf{0.56} & \mathbf{0.59} & \mathbf{0.56} & 0.09 & 0.09 \\
0.12 & -0.02 & 0.12 & \mathbf{-0.69} & \mathbf{-0.69} \\
0.40 & \mathbf{-0.80} & 0.40 & 0.09 & 0.09
\end{bmatrix}
$$

V **is "movie-to-concept" similarity matrix**

Reminder: SVD

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

- **In our case: $R \approx Q \cdot P^T$**
  - $A = R$, $Q = U$, $P^T = \Sigma V^T$



Can we use SVD?

- What we want: minimize reconstruction error on **existing** values
- What SVD does: minimize reconstruction errors on **all** values
  - SVD is not defined when entries are missing!

- In previous lecture, we used "0" for missing values
  - **Problem**: this is interpreted as a low score for that movie
  - In real world, majority of entries are zeros per user
  - SVD will learn Q,P while maximizing reconstruction of many zeros

Funk's "SVD" [Simon Funk, 2006]

- Specialized method to find P, Q
  - P, Q map users/movies to latent (concept) space
- Maximizes reconstruction of **existing** ratings by P and Q
  - Ignores the missing rating
- Predicting rating: $\hat{\boldsymbol{r}}_{\boldsymbol{xi}} = \boldsymbol{q}_{\boldsymbol{i}} \cdot \boldsymbol{p}_{\boldsymbol{x}}$
- Unlike SVD, columns of P and Q are not required to be orthonormal

Funk's "SVD" [Simon Funk, 2006]

- **Idea: let's think about this as ML problem:**
  - Minimize sum of squared error (SSE) on existing ratings ("training set")
    - $\min_{P,Q} \sum_{(i,x)\in \text{training}} \left(r_{xi} - q_i \cdot p_x\right)^2$
  - Predict ratings for missing entries with high accuracy
    - Evaluate using a held-out (unseen) "test set"

The Utility Matrix *R*

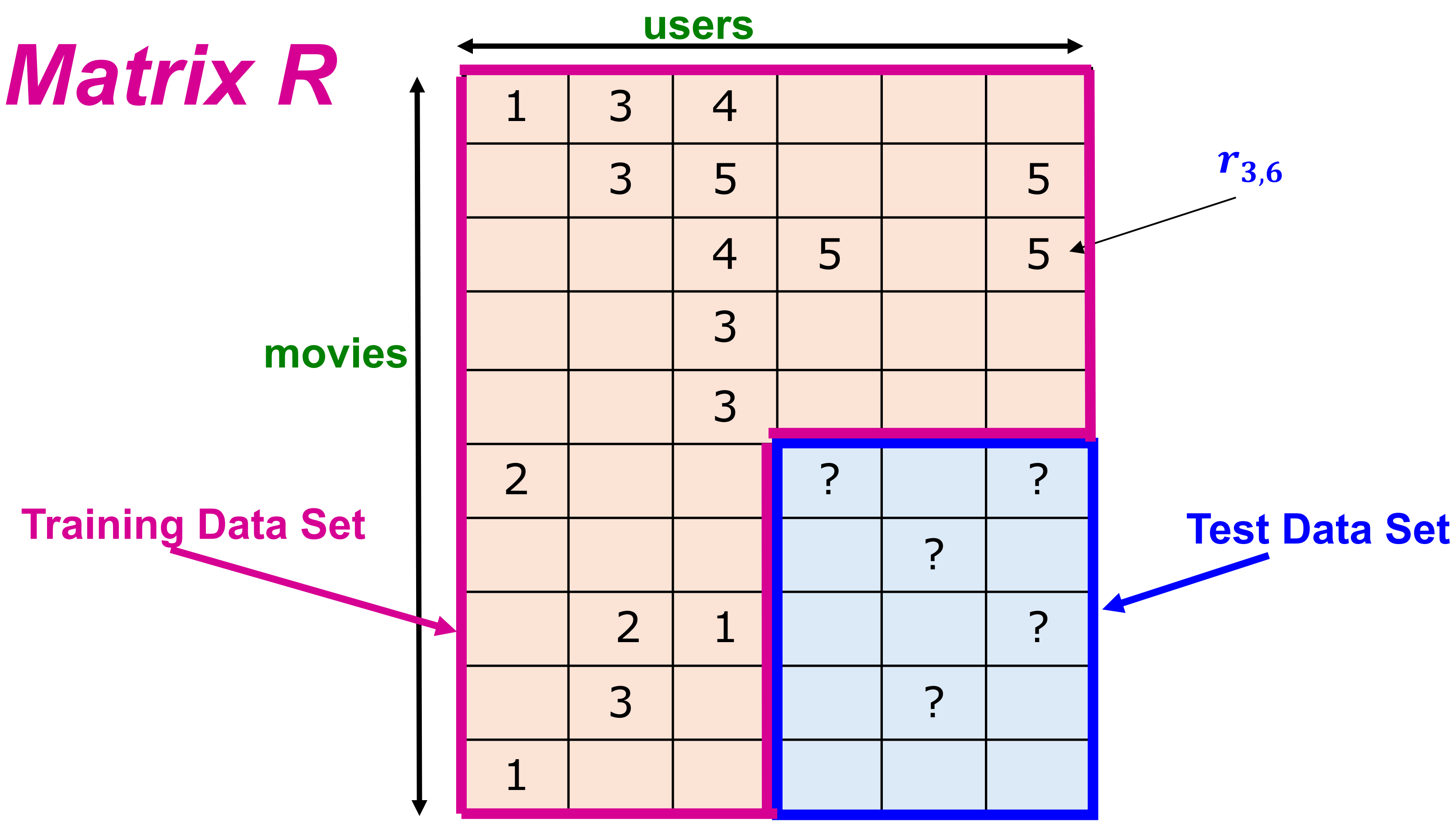
Matrix R

users

movies

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

Utility *R*: Evaluation



Evaluate how close the true rating of user ***x*** on item ***i*** *to the predicted rating*

Funk's "SVD" [Simon Funk, 2006]

- **Idea: let's think about this as ML problem:**
  - Minimize sum of squared error (SSE) on existing ratings ("training set")
    - $\min_{P,Q} \sum_{(i,x)\in\text{training}} \left(r_{xi} - q_i \cdot p_x\right)^2$
  - Predict ratings for missing entries with high accuracy
    - Evaluate using a held-out (unseen) "test set"

- **Questions:**
  - Q1: Would the learned P,Q accurately predict the missing entries?
  - Q2: How to learn P,Q based only on existing ratings?

Q1: Predicting missing ratings

- **Ideally: the learned P,Q minimize SSE for unseen test data**

- **In practice: Minimize SSE on training data**
  - Larger k (# of factors) will capture all the signals and reduce loss
  - But, SSE on **test data** begins to rise for *larger k*

- **This is a classical example of overfitting:**
  - Too many free parameters (high variance): the model starts fitting noise
    - Not generalizing well to unseen test data

Q1: Predicting missing ratings

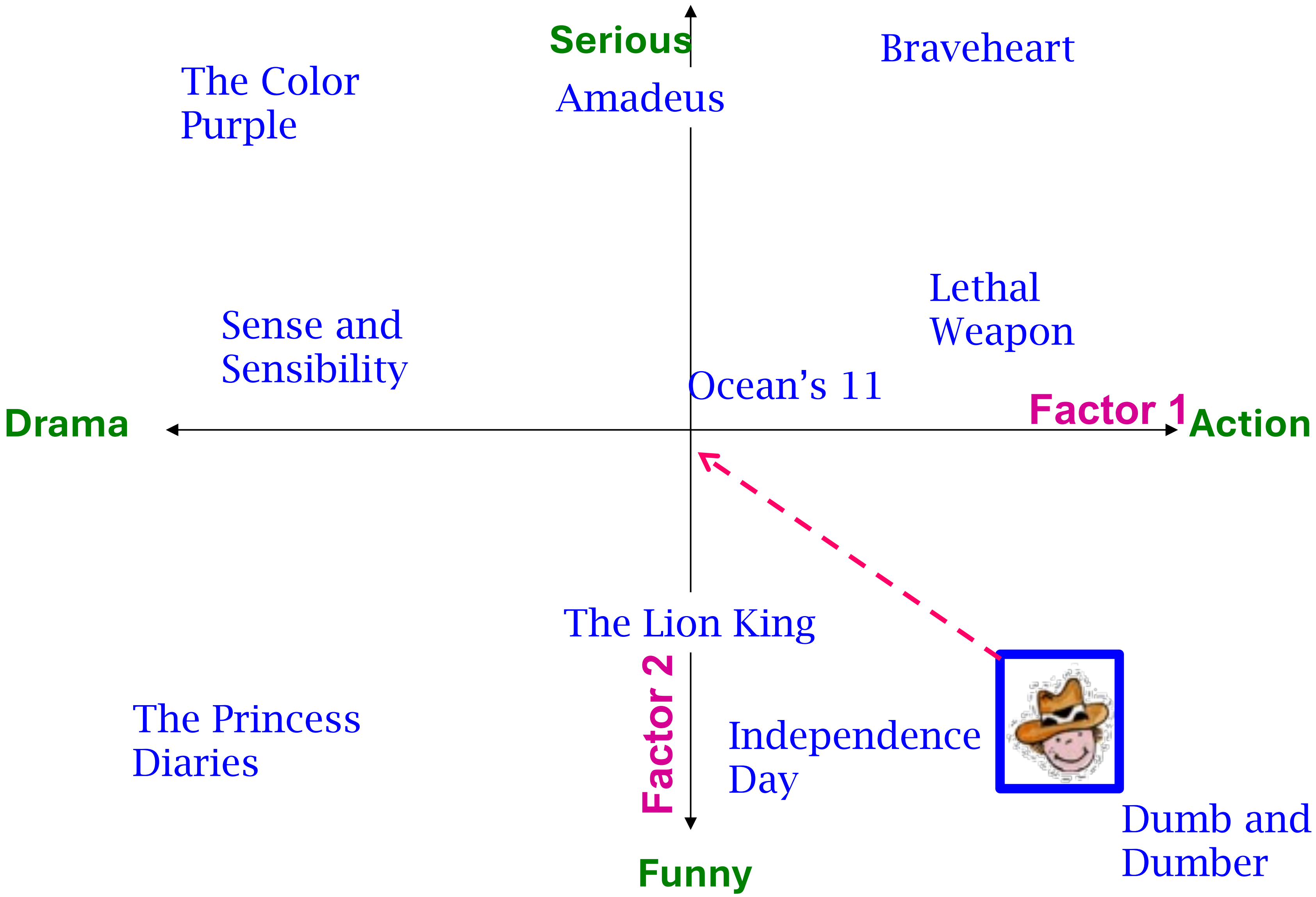
- **To address overfitting: use regularization**
  - Allow rich model where there are sufficient data
  - Shrink aggressively where data are scarce

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

λ: regularization weight (strength)

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

Intuition: Regularization



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

Q2: How to learn P,Q?

- Goal: optimize

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

- Two main approaches:
  - Alternating Least Squares (covered in lab & assignment 5)
  - **(Stochastic) Gradient Descent**

Koren et al., 2009

Gradient Descent for Funk's "SVD"

- Want to find matrices P and Q: $\min \sum_{training} (r_{xi} - q_i \cdot p_x)^2 + \lambda \left(\sum_x \|p_x\|^2 + \sum_i \|q_i\|^2 \right)$

- **Gradient decent:**
  - Initialize ***P*** and ***Q*** (run SVD with '0's for missing ratings)
  - Do gradient descent:
    - $\boldsymbol{P \leftarrow P - \eta \cdot \nabla P}$
    - $\boldsymbol{Q \leftarrow Q - \eta \cdot \nabla Q}$ [where ***∇Q*** is gradient of matrix ***Q***]

  - **How to compute gradients of matrix?**
    - **Compute gradients of each element independently**
      $\nabla Q = [\nabla q_{if}]$ and $\nabla q_{if} = \sum_{x,i} -2(r_{xi} - q_i p_x) p_{xf} + 2\lambda q_{if}$

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

Gradient Descent for Funk's "SVD"

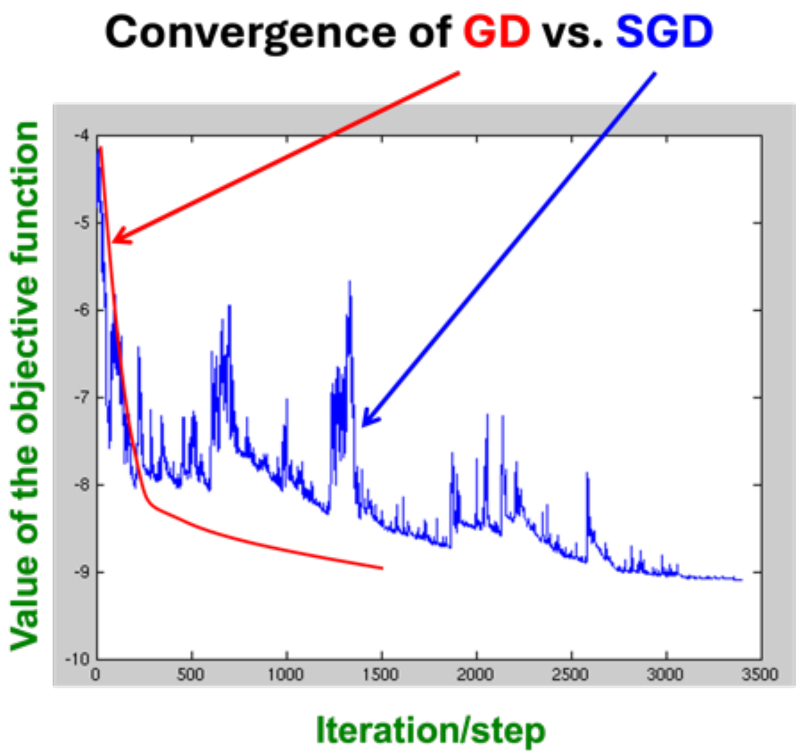
- **Gradient Descent:**

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

- **Stochastic Gradient Descent:**
  - Instead of evaluating gradient over all ratings, evaluate it for each individual rating and make a step
  - **SGD converges faster**
    - It requires more steps, but each step is computed much faster

Gradient Descent for Funk's "SVD"

- **Stochastic Gradient Descent:**
  - Initialize P and Q (using SVD, pretend missing ratings are 0)
  - Iterate until convergence:
    - For each rating r_{xi}:
      - $e_{ui} = (r_{xi} - q_i \cdot p_x)$
      - $q_i = q_i + \eta(e_{xi}p_x - \lambda q_i)$
      - $p_i = p_i + \eta(e_{ui}p_i - \lambda p_i)$



Note: the term '2' in the gradients is absorbed by the learning rate

The Netflix Prize

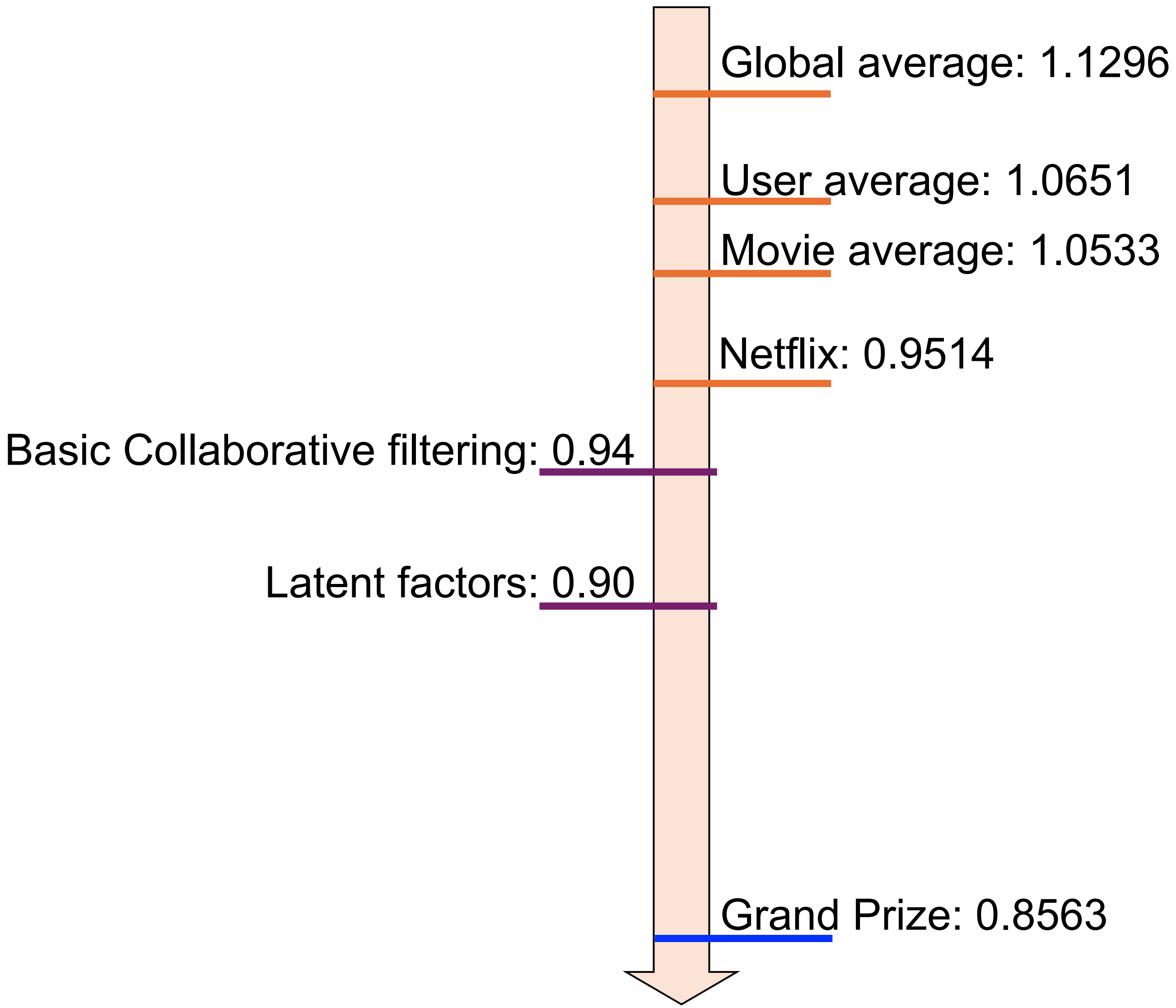
- **Training data**
  - 100 million ratings, 480,000 users, 17,770 movies
  - Collected for 6 year: 2000-2005

- **Test data**
  - Last few ratings of each user (2.8 million)
  - Evaluation using 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

*** SSE and RMSE are monotonically related (Funk's SVD optimizes RMSE)**

- **The Netflix Prize Competition**
  - 2,700+ teams
  - **$1 million prize for 10% improvement on Netflix**

Netflix Prize Performance



Summary

- The problem of Collaborative Filtering
- Latent Factor Models approach to CF
- Funk’s “SVD” algorithm

What’s next:

- The remaining improvements for the Netflix Prize
- Evaluation beyond RMSE
- Neural approaches for collaborative filtering