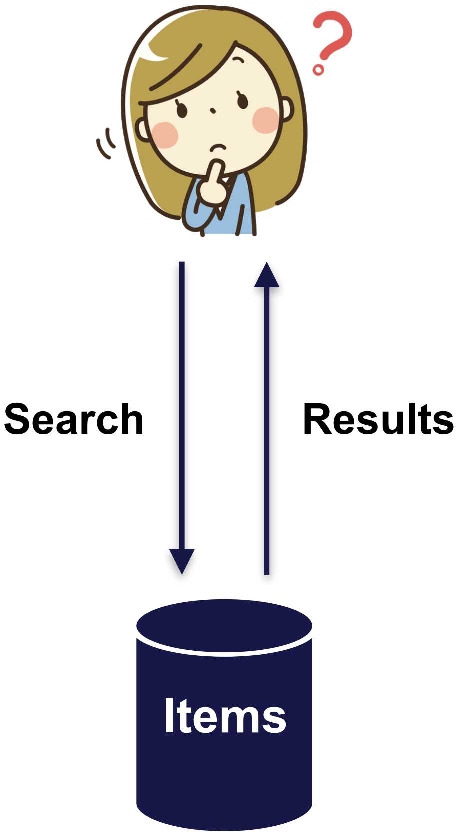
42578 Advanced Business Analytics

Recommender Systems

Stanislav Borysov [stabo@dtu.dk] | Adapted from J. Leskovec, A. Rajaraman, J. Ullman (“Mining of Massive Datasets”, http://www.mmds.org, Stanford University)

Recommendations

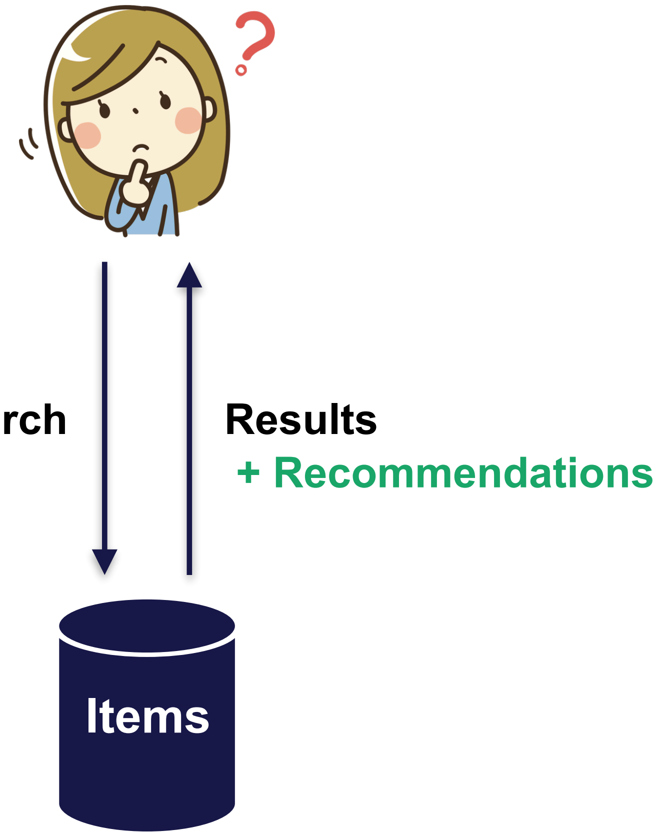


Search

Results

Items

Recommendations



Examples

amazon

NETFLIX

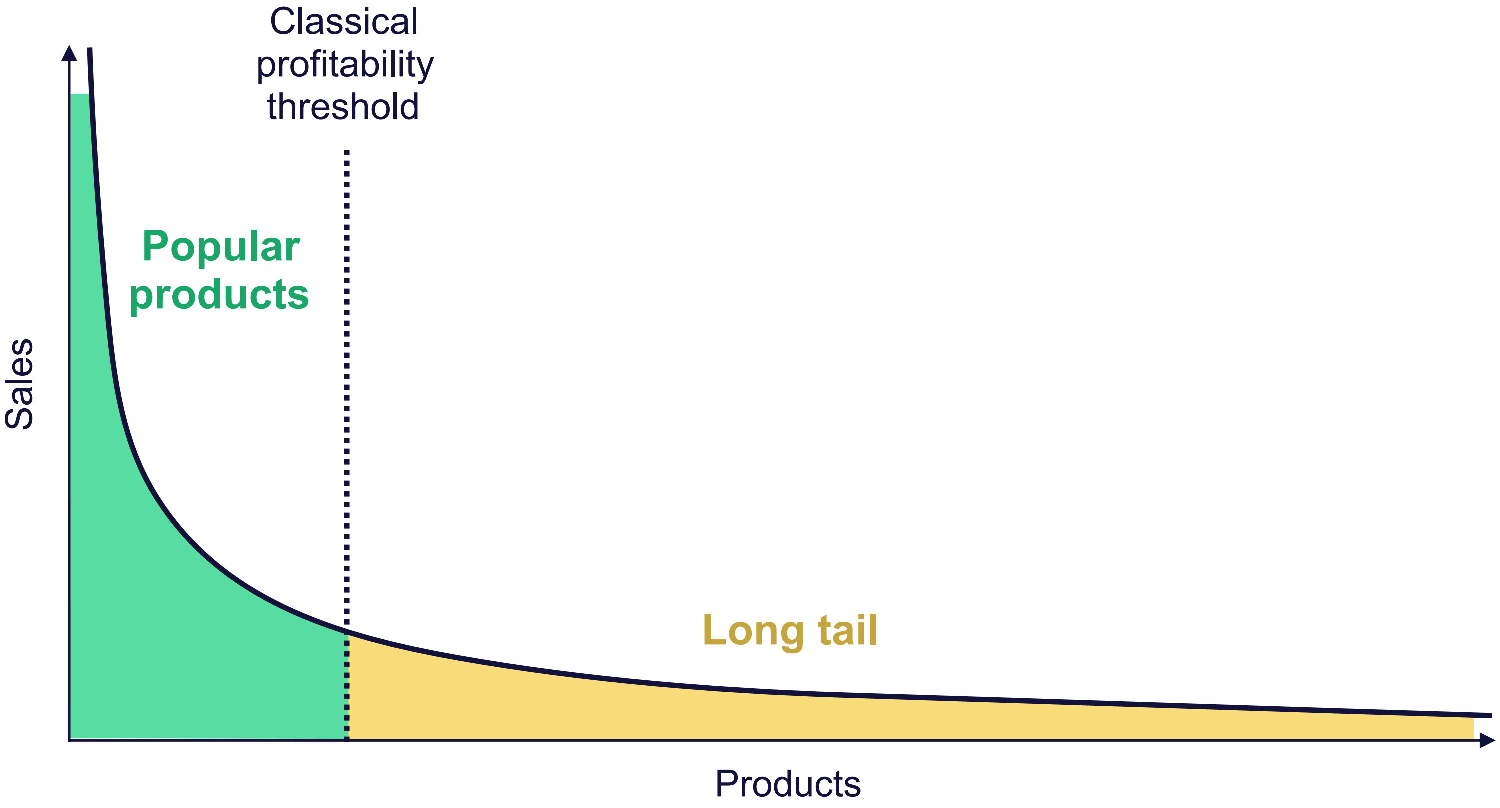
YouTube

Spotify

From Scarcity to Abundance

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

The Long Tail



Types of Recommendations

- **Editorial and hand-curated**
  - Editors' picks
  - Lists of “essential” items

- **Simple aggregates**
  - Top 10, Most Popular, Recent Uploads

- **Tailored to individual users**
  - Amazon, Netflix, …

Formal Model

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

- **Utility function u: X × I → R**
  - **R** = set of ratings
  - **R** is a totally ordered set
  - e.g., **0-5** stars, real number in **[0,1]** (purchase probability), …

Rating (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 the matrix**
  - How to collect the data in the rating 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 Ratings

- **Key problem:** Rating matrix **R** is **sparse**
  - Most people have not rated most items
  - **Cold start:**
    - New items have no ratings
    - New users have no history

- **Two approaches to recommender systems:**
  - **(1)** Content-based recommendations
  - **(2)** Collaborative filtering
    - Memory-based approach: Item-Item, User-User
    - Model-based approach: Matrix factorisation, Latent factors, PCA, Neural nets, …
    - Hybrid
  - Hybrid

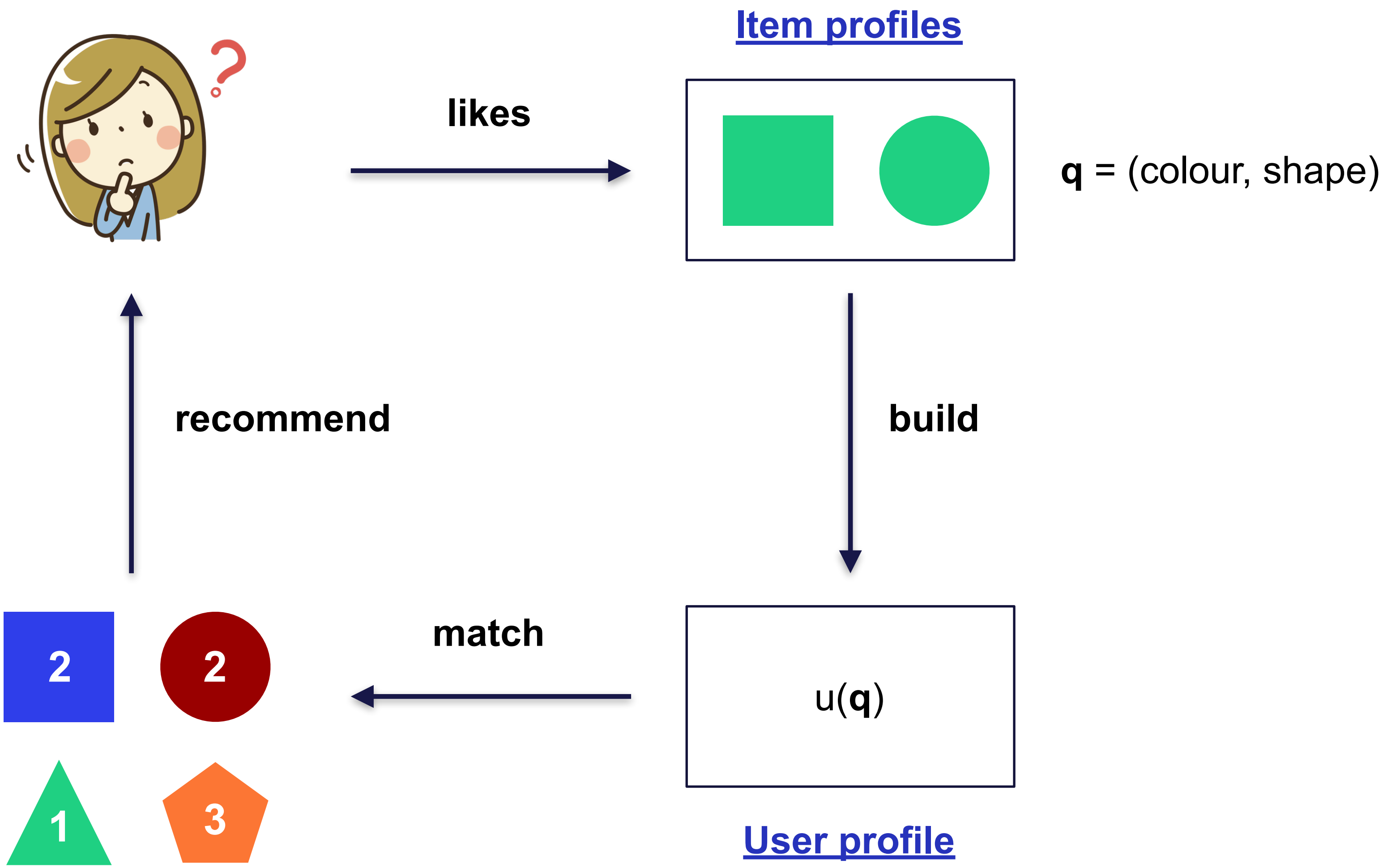
Content-based Recommender Systems

Content-based Recommendations

- **Main idea:** Recommend items to the 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

Main Idea



Item profiles

likes

$\mathbf{q}$ = (colour, shape)

build

$u(\mathbf{q})$

User profile

match

recommend

Item Profiles

- For each item, create an **item profile,** q_i

- **Item profile is a set (vector) of item features**
  - **Movies:** author, title, actor, director,…
  - **Text:** Set of "important" words in document
    - TF-IDF, topics, …
  - **Images, music:**
    - Hand-crafted features
    - Unsupervised learning ("embeddings"), e.g., Clustering, Generative modelling

User Profile

- **General case: Utility function $u_x(q)$**

- **Simplest case: Linear model**
  - User profile is a vector of weights $\mathbf{p}_x$ for features in $\mathbf{q}$
  - linear regression
  - logistic regression (probability of “like”)

- **Nonlinear “black-box” models**
  - Lack of interpretability

Pros: Content-based Approach

\+ Does not heavily depend on data on other users

\+ Able to recommend to users with unique tastes

\+ Able to recommend new & unpopular items

Cons: Content-based Approach

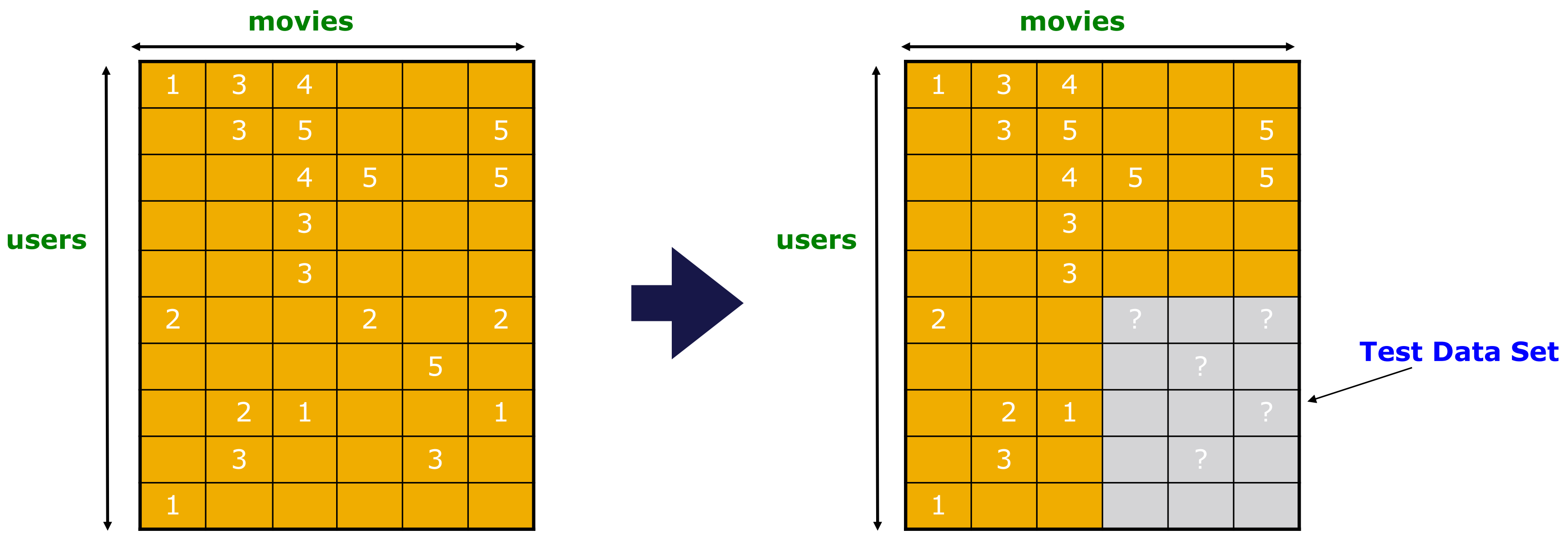
- **Finding the appropriate features is hard**
  - E.g., images, movies, music

- **Recommendations for new users**
  - Irrelevant recommendations in the beginning

- **Overspecialisation**
  - Never recommends items outside user's content profile
  - People might have multiple interests
  - **Unable to exploit quality judgments of other users**

Evaluation of Recommender Systems

Evaluation of Recommender Systems



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

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

- Train / Test split
- Cross-validation

Performance Measure

- **Root-mean-square error** (RMSE)

  $$\sqrt{\frac{1}{N}\sum_{xi}\left(\hat{r}_{xi}-r_{xi}\right)^2}$$ where $\hat{r}_{xi}$ is predicted and r_{xi} is true rating of *x* on *i*

  – **cons:** In practice, we care only to predict high ratings, and RMSE might penalise a method that does well for high ratings and badly for others

- **Precision at top *k***
  - % of those in top k predictions (in practice, often *k*=10)

- **Rank (Spearman's) correlation**

Baselines

- **Global mean rating, μ**
  - average of all ratings
- **User mean rating, μ_x**
  - average of all ratings from the user *x*
- **Item mean rating, μ_i**
  - average of all ratings for the item *i*
- **Combination**
  - $r_{xi} = \mu + \boxed{b_x} + \boxed{b_i} = \mu + (\mu_x - \mu) + (\mu_i - \mu)$

User's bias
(deviation of the user's mean from the global mean)

Item's bias
(deviation of the item's mean from the global mean)

Exercises

- “Recommender Systems.ipynb”: Sections 1 - 3.1 (Content-based filtering)

Memory-based Collaborative Filtering

Person A

- likes private jets
- likes beer

https://www.buzzfeednews.com/article/miriamelder/russians-who-look-like-hollywood-stars

Person A

- likes private jets
- likes beer

Person B

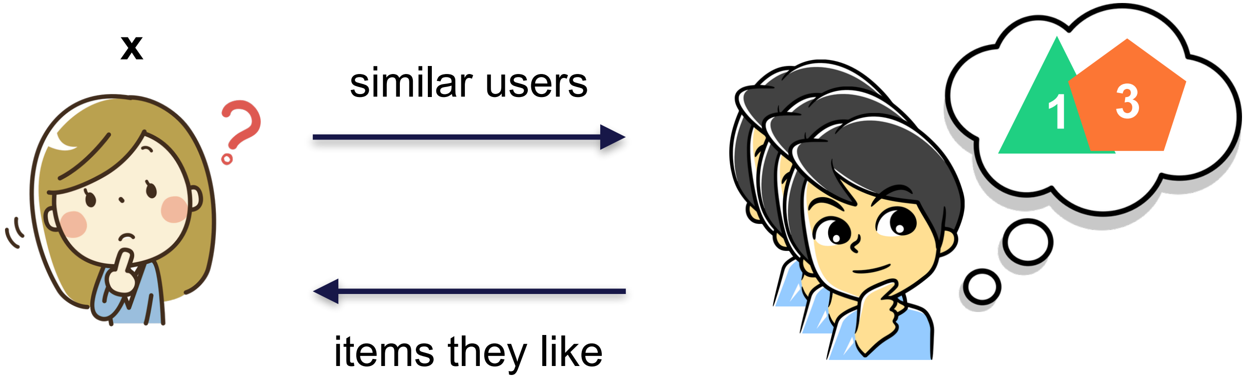
- wants to buy beer
- recommend to buy a private jet?

* Do not judge a person based on just one photograph

https://www.buzzfeednews.com/article/miriamelder/russians-who-look-like-hollywood-stars

User-User Collaborative Filtering

- Consider user **x**
- Find similarity of **x** to **all other users**
- Estimate **x**’s ratings using ratings of the other users weighted by their similarity to x
- **K-Nearest-Neighbor approach:** all other users → *K* most similar users



x

similar users

items they like

Finding “Similar” Users

- Let **r**$_{\mathbf{x}}$ be the vector of user **x**’s ratings: **r**$_{\mathbf{x}}$ = (1, ?, ?, 4, 5) and **r**$_{\mathbf{y}}$ be the vector of user **y**’s ratings: **r**$_{\mathbf{y}}$ = (?, 1, ?, 3, 4)

- **Pearson correlation coefficient**
  - **S**$_{\mathbf{xy}}$ = items rated by both users **x** and **y**: (4, 5) and (3, 4)

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

Rating Predictions

- Predict $\mathbf{r_{xi}}$ — rating of the item **i** by the user **x**

- Let **K** be the set of *K* users most similar to **x** who have rated item **i**

- **Prediction:**
  - $$r_{xi} = \frac{1}{K} \sum_{y \in \mathbf{K}} r_{yi}$$
  - $$r_{xi} = \frac{\sum_{y \in \mathbf{K}} sim(x, y) r_{yi}}{\sum_{y \in \mathbf{K}} sim(x, y)}$$

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 the user-user model

$$r_{xi} = \frac{\sum_{j \in \mathbf{K}} sim(i,j) r_{xj}}{\sum_{j \in \mathbf{K}} sim(i,j)}$$

  - **K** is a set of *K* most similar items to *i* rated by *x*

Item-Item Collaborative Filtering: Example (*K*=2)

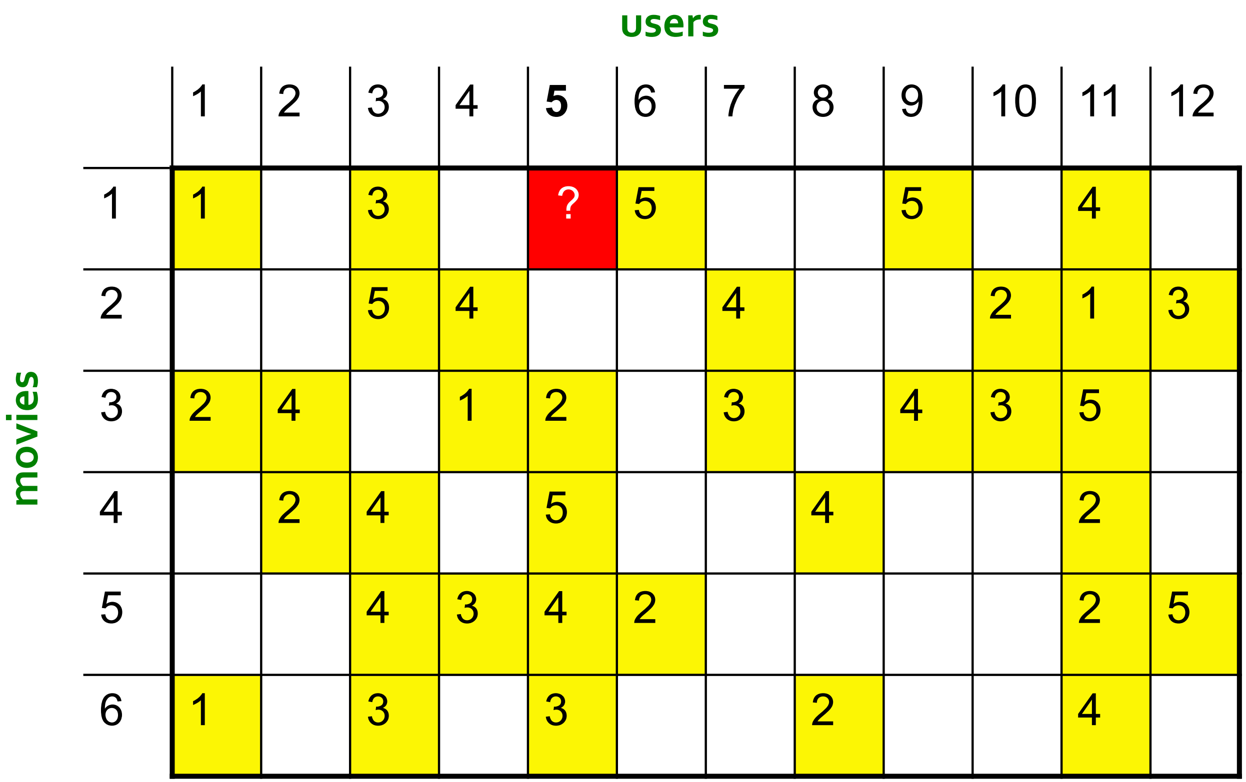
users

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 Collaborative Filtering: Example (*K*=2)



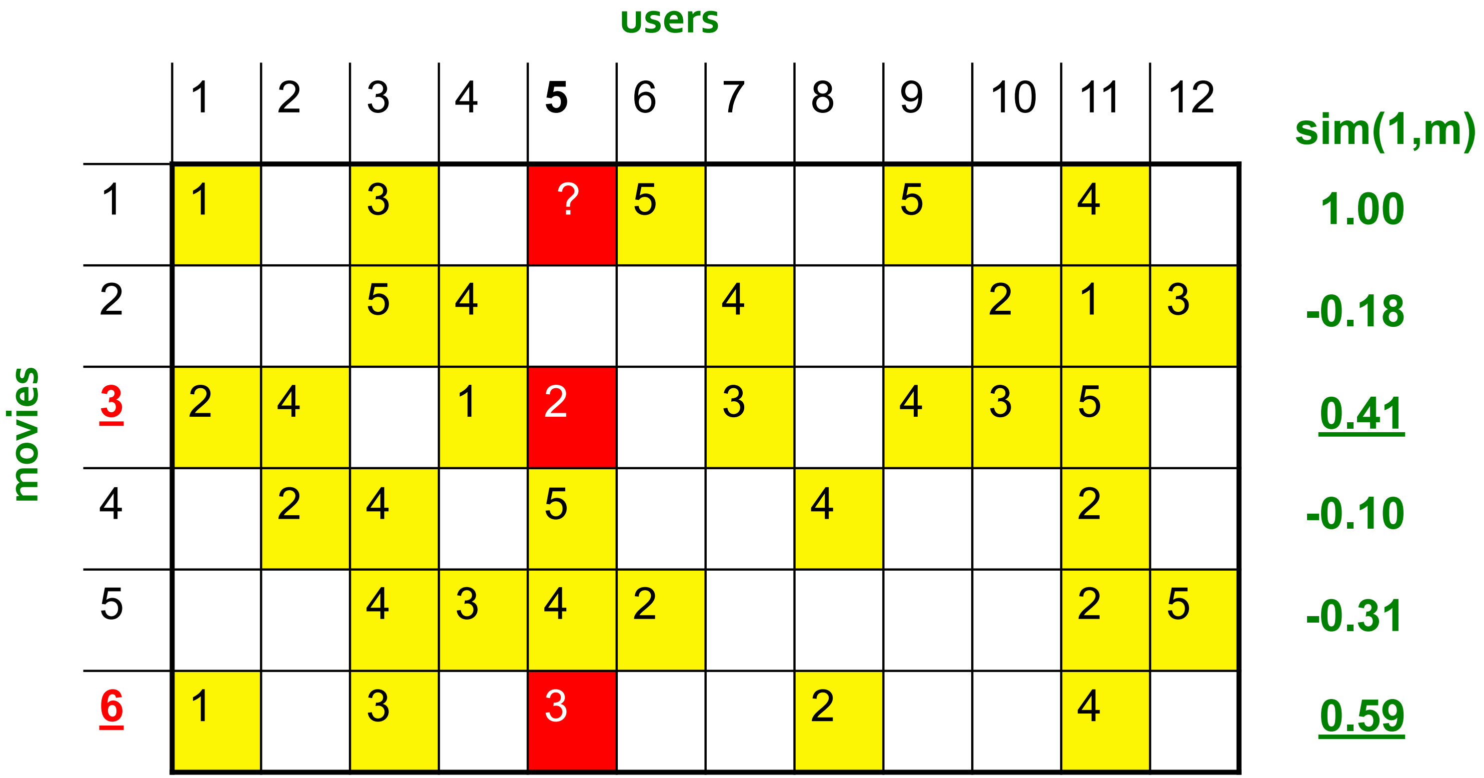
users

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 Collaborative Filtering: Example (*K*=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
$m_1 = (1+3+5+5+4)/5 = \mathbf{3.6}$
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 Collaborative Filtering: Example (*K*=2)

users

movies

	1	2	3	4	5	6	7	8	9	10	11	12	sim(1,m)
1	1		3		**2.6**	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

Similarity weights:

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

Predict by taking weighted average:

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

Collaborative Filtering: Common Practice

$$r_{xi} = \frac{\sum_{y \in \mathbf{K}} sim(x,y) r_{yi}}{\sum_{y \in \mathbf{K}} sim(x,y)}$$

$$\Downarrow$$

$$r_{xi} = \underbrace{\boxed{b_{xi}}}_{\text{Baseline estimate for } r_{xi}} + \underbrace{\boxed{\frac{\sum_{y \in \mathbf{K}} sim(x,y)\left(r_{yi} - b_{yi}\right)}{\sum_{y \in \mathbf{K}} sim(x,y)}}}_{\text{Deviation from the baseline}}$$

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

Item-Item vs. User-User

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

Pros/Cons of Collaborative Filtering

+ Works for any kind of item
  - No feature selection needed

- Cold Start:
  - Need enough users in the system to find a match

- Sparsity:
  - The user/ratings matrix is sparse
  - Hard to find users that have rated the same items

- First rater:
  - Cannot recommend an item that has not been previously rated
  - New items, Esoteric items

- Popularity bias:
  - Cannot recommend items to someone with unique taste
  - Tends to recommend popular items

- Expensive step is finding K most similar customers/items (linear complexity)
  - Solution: caching, indexing, nearest-neighbour search, etc.

Hybrid Methods

- **Implement two or more different recommenders and combine predictions**
  - Perhaps using a linear model

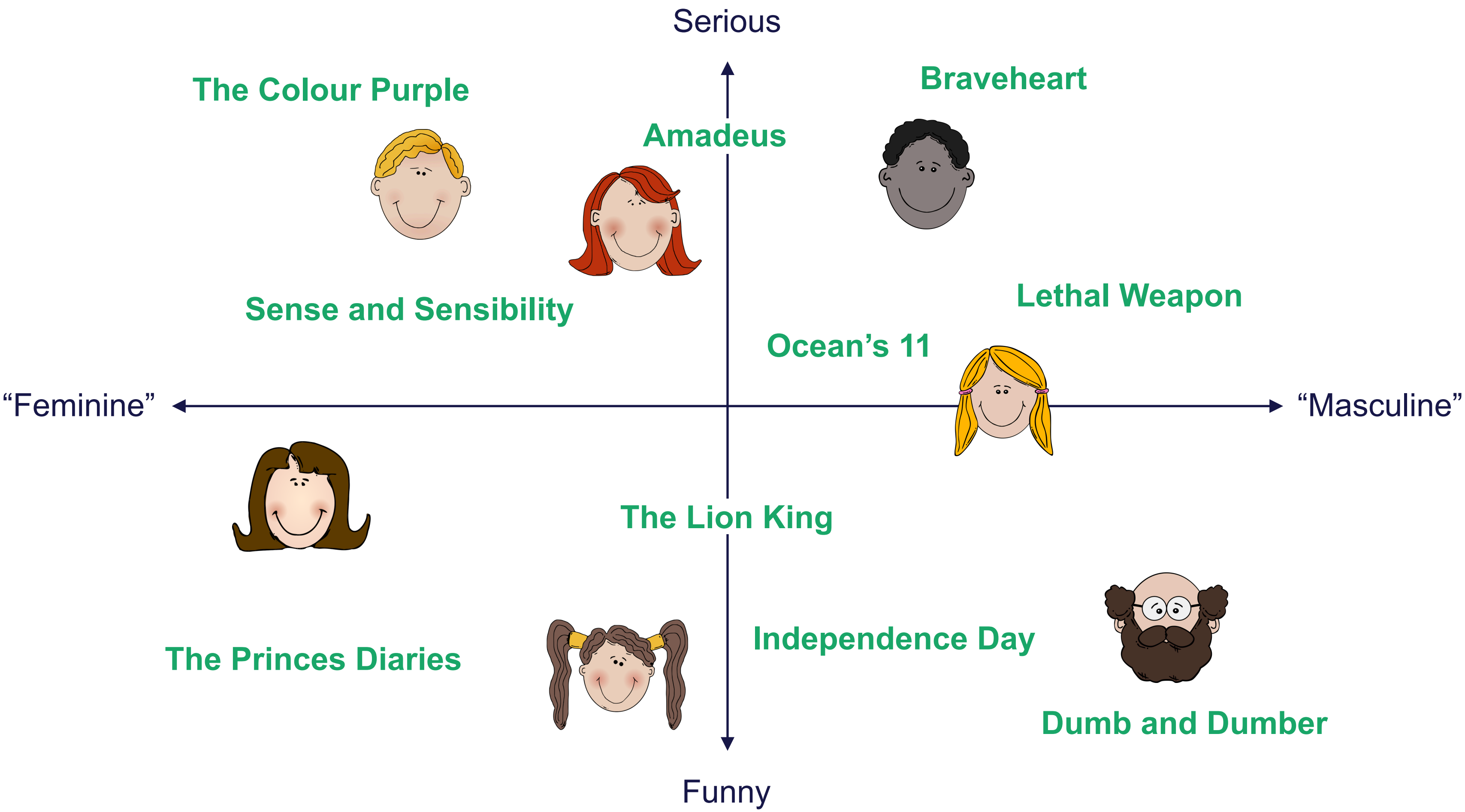
- **Add content-based methods to collaborative filtering**
  - Item profiles for new item problem
  - Demographics to deal with new user problem

Exercises

- Part 2 - Memory-based recommendations

Model-based Collaborative Filtering

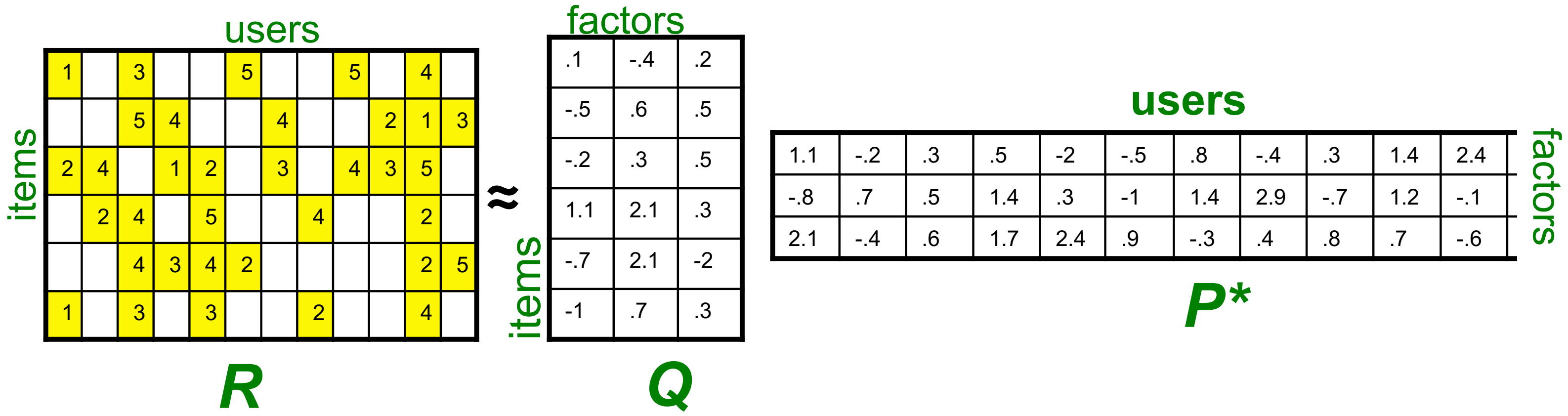
Latent Factor Models



Latent Factor Models

- “SVD” on the utility matrix: R ≈ Q · P*

SVD: D=UΣT*



R

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

Q

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

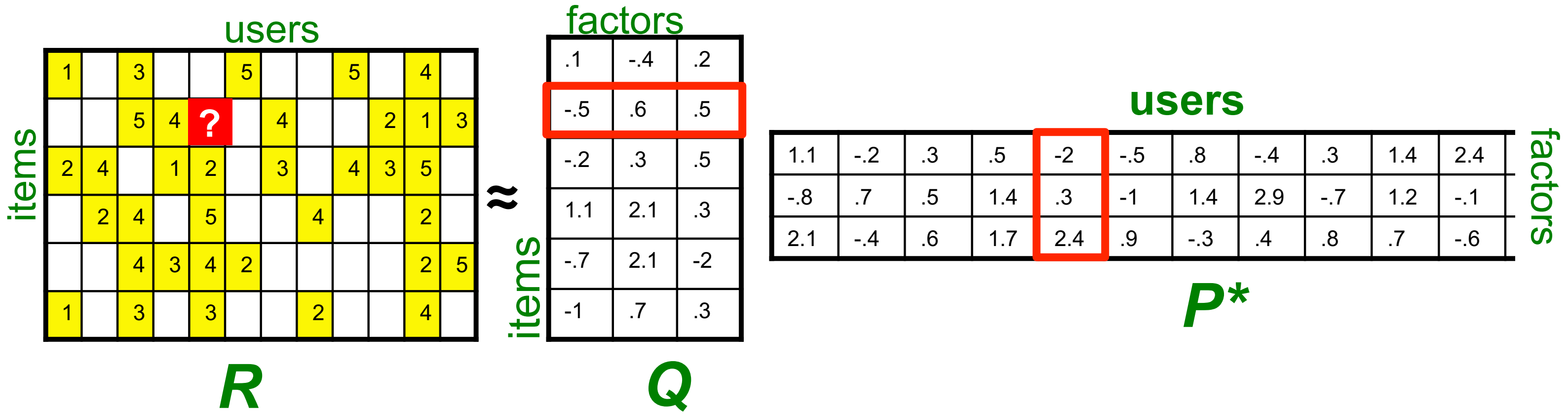
P*

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***
  - 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*?**



R

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

Q

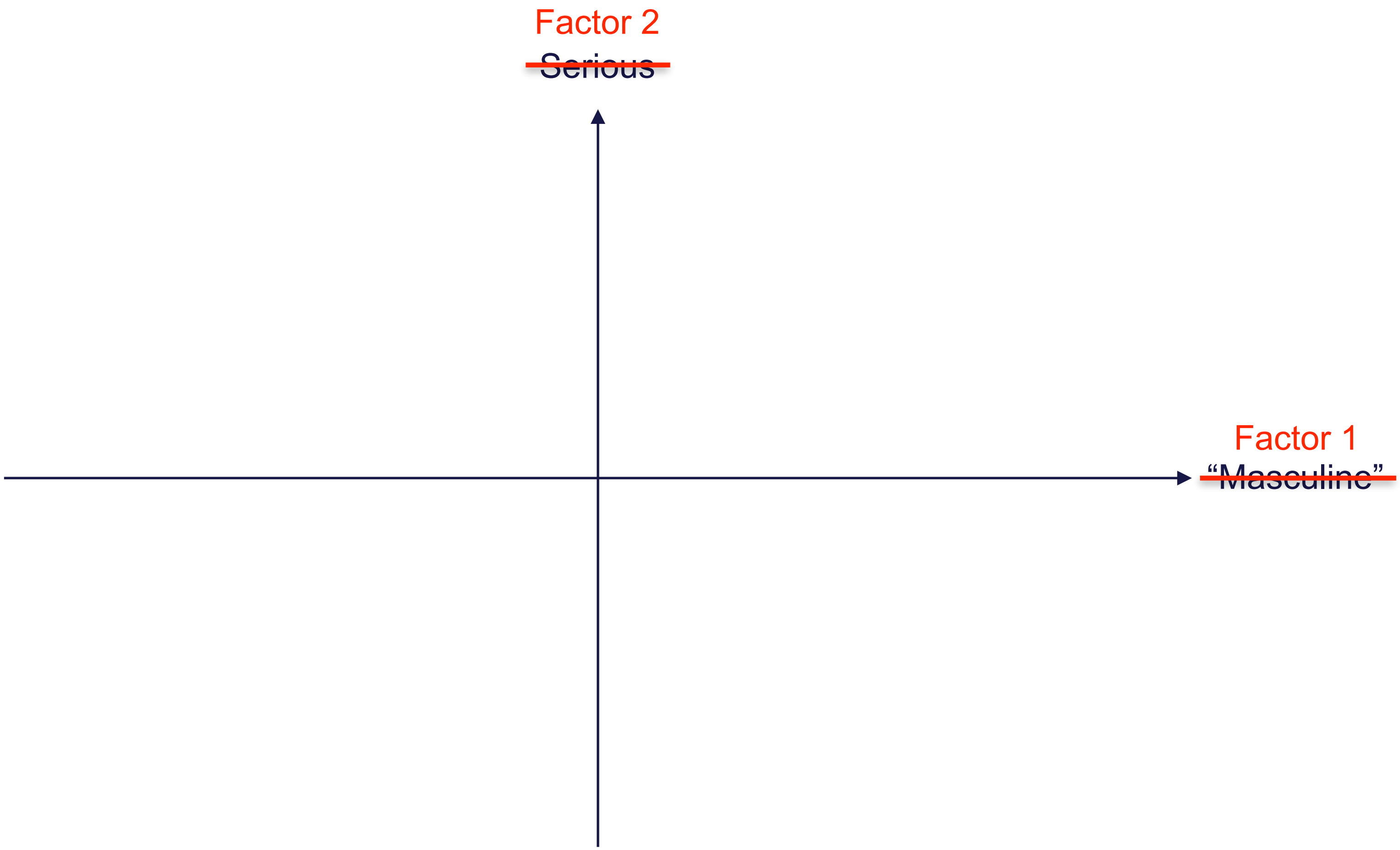
.1	-.4	.2
-.5	.6	.5
-.2	.3	.5
1.1	2.1	.3
-.7	2.1	-2
-1	.7	.3

P*

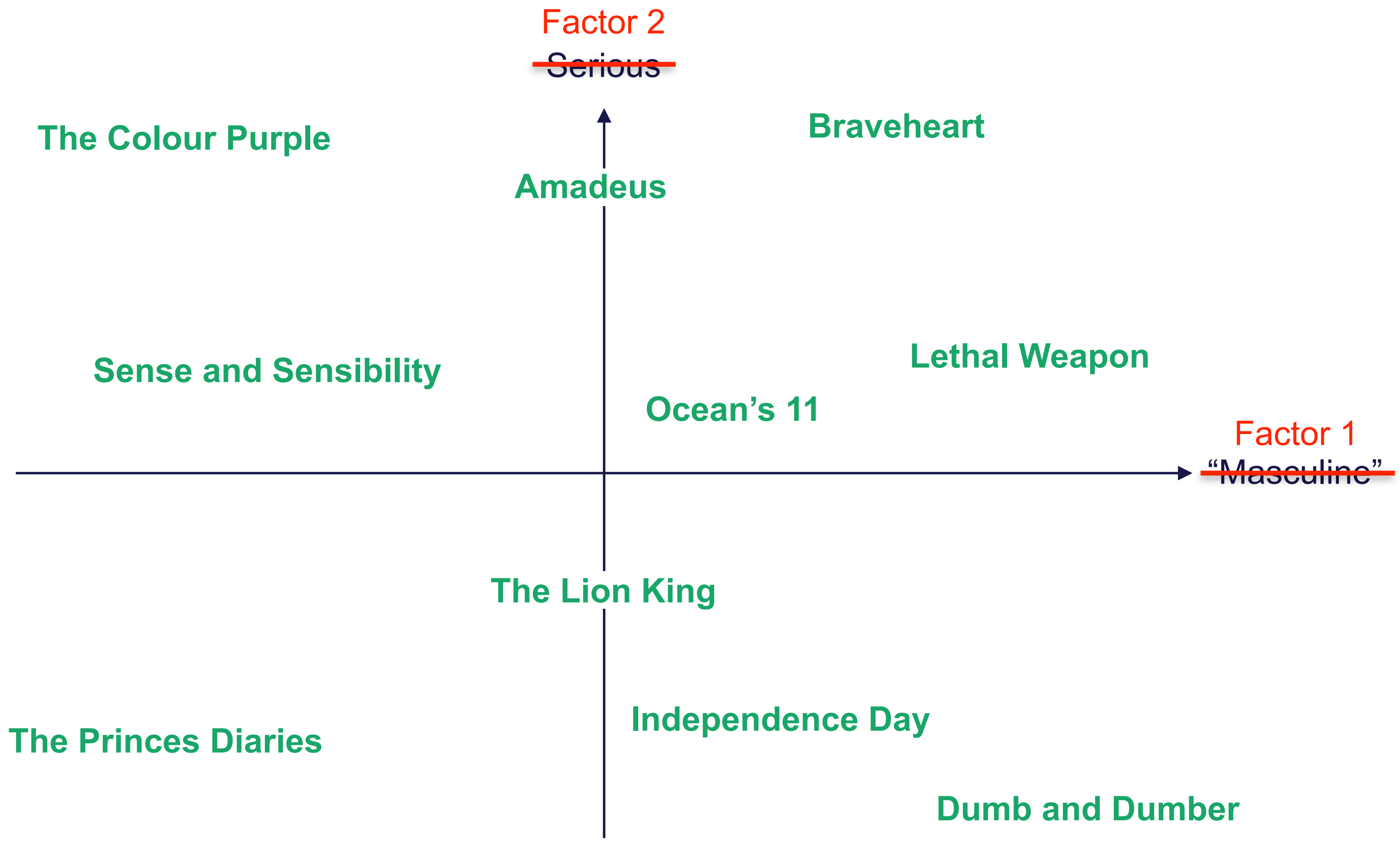
1.1	-.2	.3	.5	-2	-.5	.8	-.4	.3	1.4	2.4
-.8	.7	.5	1.4	.3	-1	1.4	2.9	-.7	1.2	-.1
2.1	-.4	.6	1.7	2.4	.9	-.3	.4	.8	.7	-.6

- $\hat{r}_{xi} = \mathbf{q}_i \mathbf{p}_x{}^* = \sum_j q_{ij} p_{xj}$ = -0.5 * -2 + 0.6 * 0.3 + 0.5 * 2.4 = 2.4

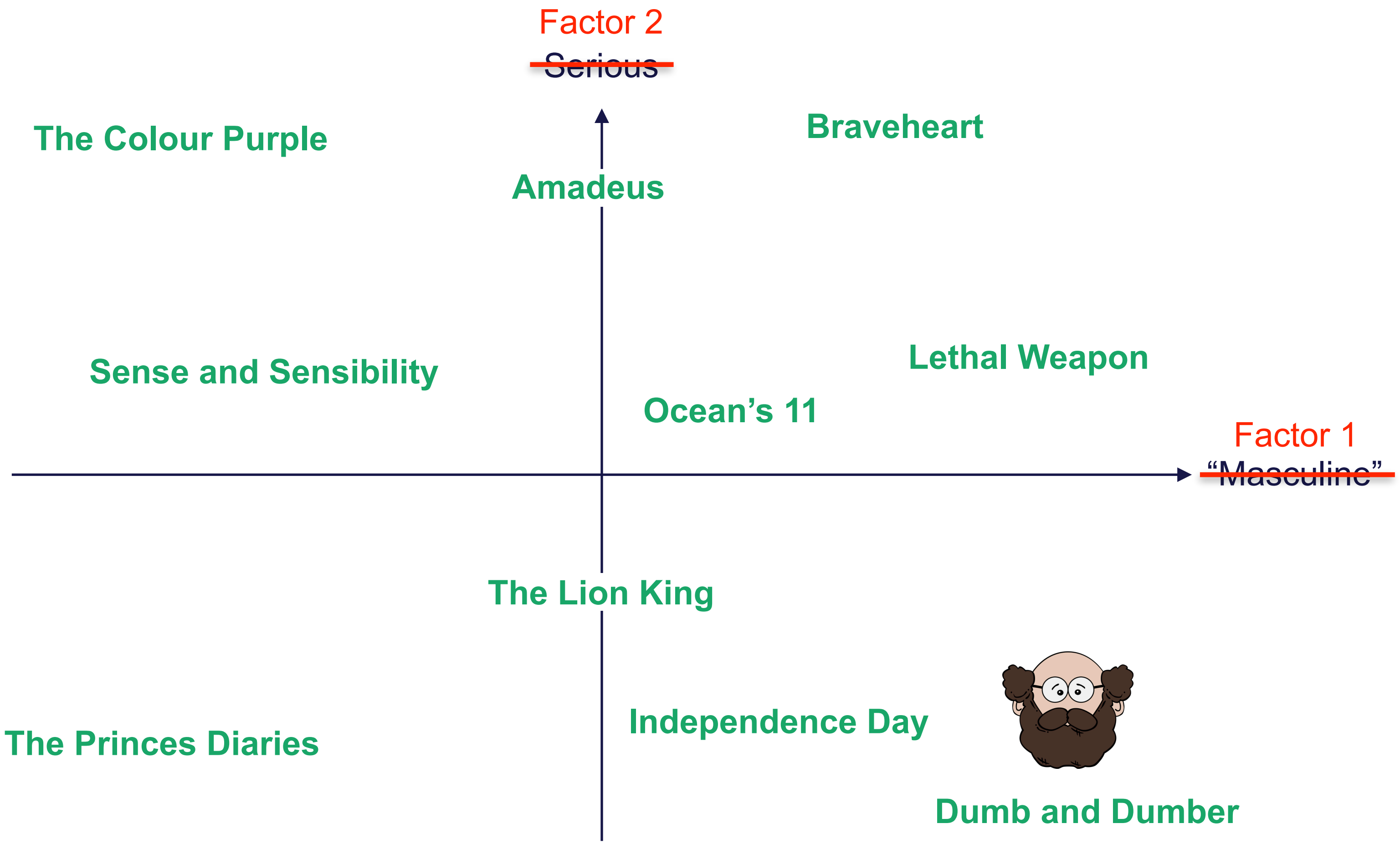
Latent Factor Models



Latent Factor Models



Latent Factor Models



Factor 2

~~Serious~~

The Colour Purple

Braveheart

Amadeus

Sense and Sensibility

Lethal Weapon

Ocean’s 11

Factor 1

~~“Masculine”~~

The Lion King

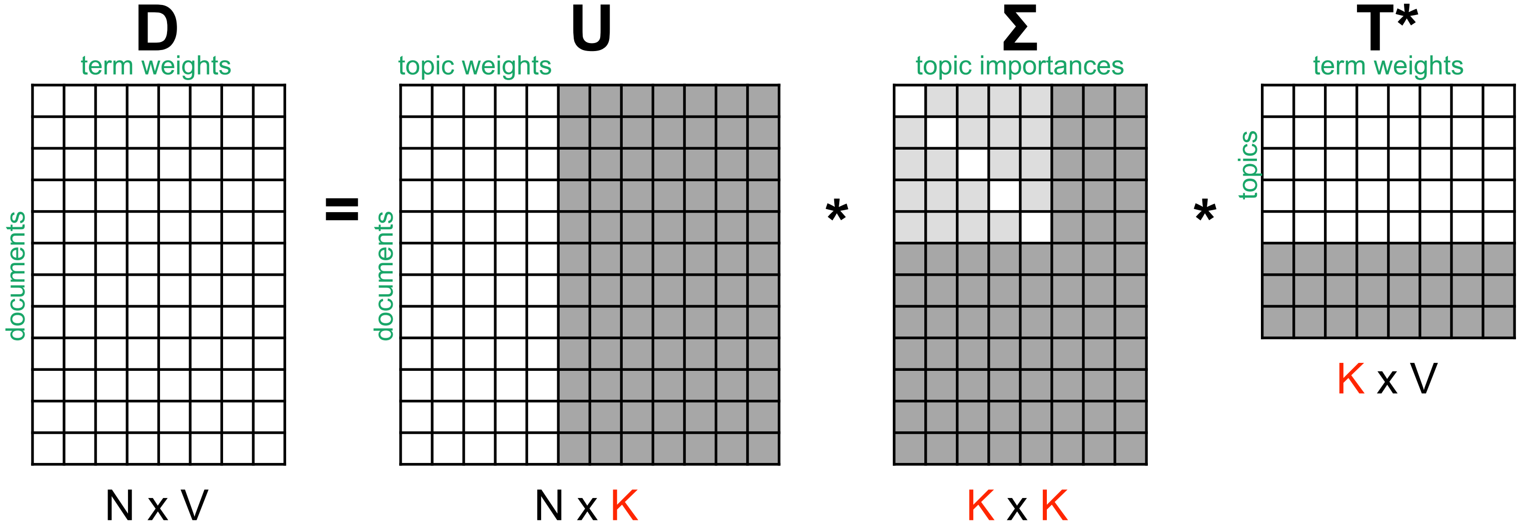
Independence Day

The Princes Diaries

Dumb and Dumber

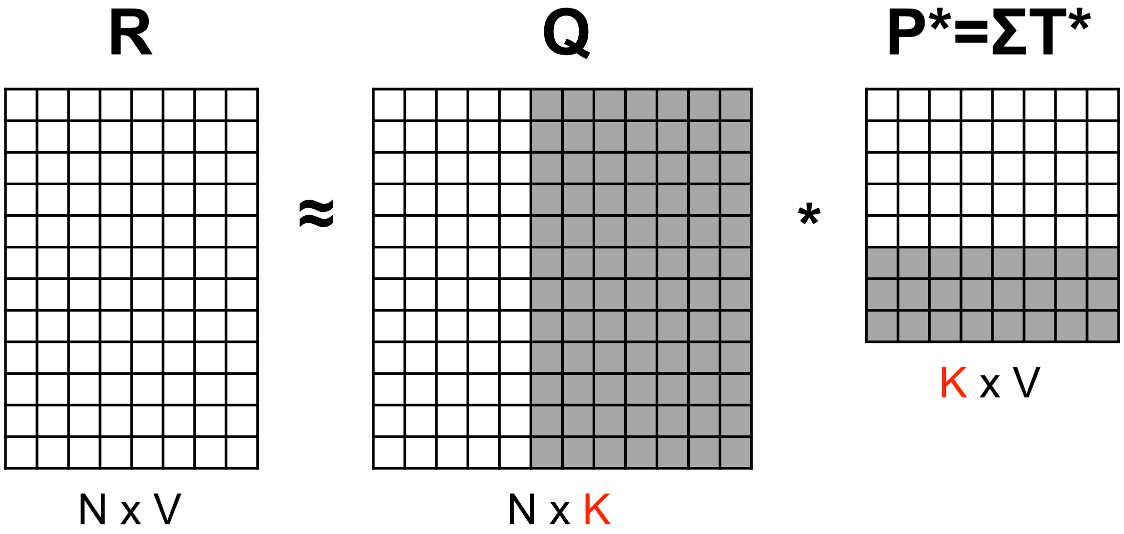
Recap: SVD

- It is a Singular Value Decomposition (generalisation of eigenvalue decomposition) of the document-term matrix **D=UΣT***



- **D** is a document-term matrix
- **U** is a document-topic map ("topic distribution")
- **Σ** is an ordered diagonal matrix of singular values ("topic importance")
- **T** is a term-topic map ("term distribution")

Low-Rank Matrix Factorisation



- **R** is a utility matrix
- **Q** is a item-factor map
- **P*** is a user-factor map

SVD vs Low-Rank Matrix Factorisation

- **SVD gives minimum reconstruction error (Sum of Squared Errors):**

  $$\min_{U,\Sigma,T} \sum_{i,j} \left(d_{ij} - \left[U \Sigma T^* \right]_{ij} \right)^2$$

  which is monotonically related to 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!**

- **Use optimisation methods to find elements of P and Q (e.g., Gradient Descent)**

  $$\min_{P,Q} \sum_{i,j \in R} \left(r_{ij} - \left[P Q^* \right]_{ij} \right)^2$$

- **Note:**
  - We don't require cols of **P**, **Q** to be orthogonal/unit length
  - **Non-convex :(**

Finding P and Q

Number of latent factors

- **Loss function**

$$J(P,Q) = \sum_{\text{training data}} \sum_{i,j \in R} \left(r_{ij} - \sum_{k=1}^{\boxed{K}} q_{ik} p_{kj} \right)^2$$

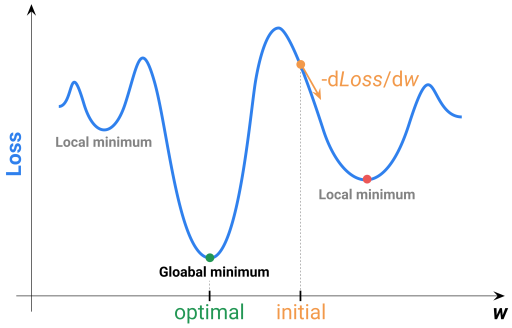
- **Gradients for q_{ij} (the same for p_{ij})**

$$\frac{\partial J}{\partial q_{ij}} = \sum_{\text{training data}} \sum_{i,j \in R} -2 \left(r_{ij} - \sum_{k=1}^{K} q_{ik} p_{kj} \right) q_{ij}$$

- **Updates (the same for p_{ij}) — until convergence**

$$q_{ij} \leftarrow q_{ij} - \boxed{\eta} \frac{\partial J}{\partial q_{ij}}$$

Learning rate



Finding P and Q

Number of latent factors

- **Loss function**

$$J(P,Q) = \sum_{\text{training data}} \sum_{i,j \in R} \left(r_{ij} - \sum_{k=1}^{\boxed{K}} q_{ik} p_{kj} \right)^2$$

- **Gradients for q$_{ij}$ (the same for p$_{ij}$)**

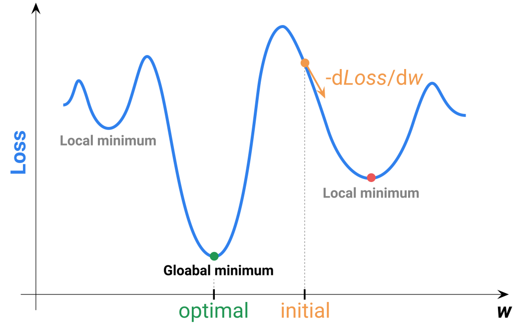
$$\frac{\partial J}{\partial q_{ij}} = \sum_{\text{~~training data~~}} \sum_{i,j \in R} -2 \left(r_{ij} - \sum_{k=1}^{K} q_{ik} p_{kj} \right) q_{ij}$$

A few training data points ("batch") — Stochastic Gradient Descent — Faster & Better in high dimensions

- **Updates (the same for p$_{ij}$) — until convergence**

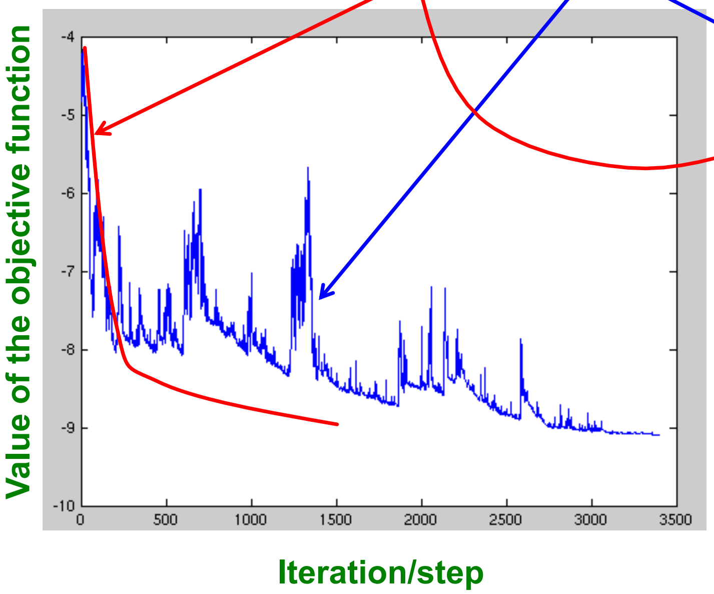
$$q_{ij} \leftarrow q_{ij} - \boxed{\eta} \frac{\partial J}{\partial q_{ij}}$$

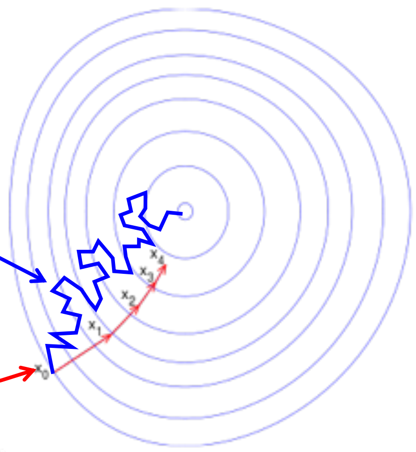
Learning rate



Sidenote: GD vs. SGD

- **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!

Finding P and Q

Number of latent factors

Hyperparameter responsible for the model's complexity (**overfitting!**)

- **Loss function**

$$J(P,Q) = \sum_{\text{training data}} \sum_{i,j\in R} \left(r_{ij} - \sum_{k=1}^{\boxed{K}} q_{ik} p_{kj} \right)^2$$

- **Gradients for q_{ij} (the same for p_{ij})**

$$\frac{\partial J}{\partial q_{ij}} = \sum_{\text{training data}} \sum_{i,j\in R} -2 \left(r_{ij} - \sum_{k=1}^{K} q_{ik} p_{kj} \right) q_{ij}$$

- **Updates (the same for p_{ij}) — until convergence**

$$q_{ij} \leftarrow q_{ij} - \eta \frac{\partial J}{\partial q_{ij}}$$

Finding P and Q

- **Loss function**

$$J(P,Q) = \sum_{\text{training data}} \sum_{i,j \in R} \left(r_{ij} - \sum_{k=1}^{\boxed{K}} q_{ik} p_{kj} \right)^2$$

Number of latent factors

Hyperparameter responsible for the model's complexity (**overfitting!**)

- regularisation
  $$+\lambda_1 \sum_{ij} q_{ij}^2 + \lambda_2 \sum_{ij} p_{ij}^2$$
  (two more HPs to tune)
- direct tuning on a separate validation set / cross-validation on train data

- **Gradients for q_{ij} (the same for p_{ij})**

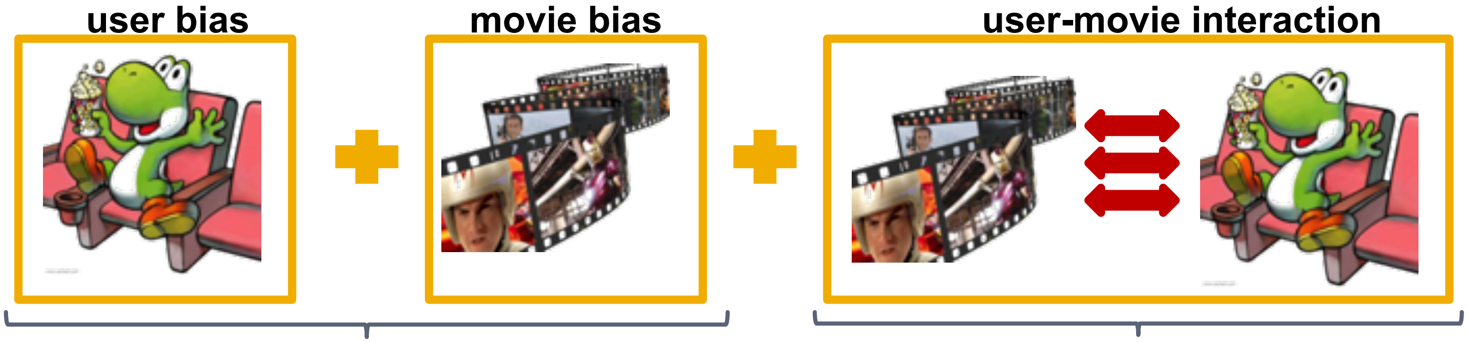
$$\frac{\partial J}{\partial q_{ij}} = \sum_{\text{training data}} \sum_{i,j \in R} -2 \left(r_{ij} - \sum_{k=1}^{K} q_{ik} p_{kj} \right) q_{ij}$$

- **Updates (the same for p_{ij}) — until convergence**

$$q_{ij} \leftarrow q_{ij} - \eta \frac{\partial J}{\partial q_{ij}}$$

One Last Thing…

Modelling Biases and Interactions



Baseline predictor

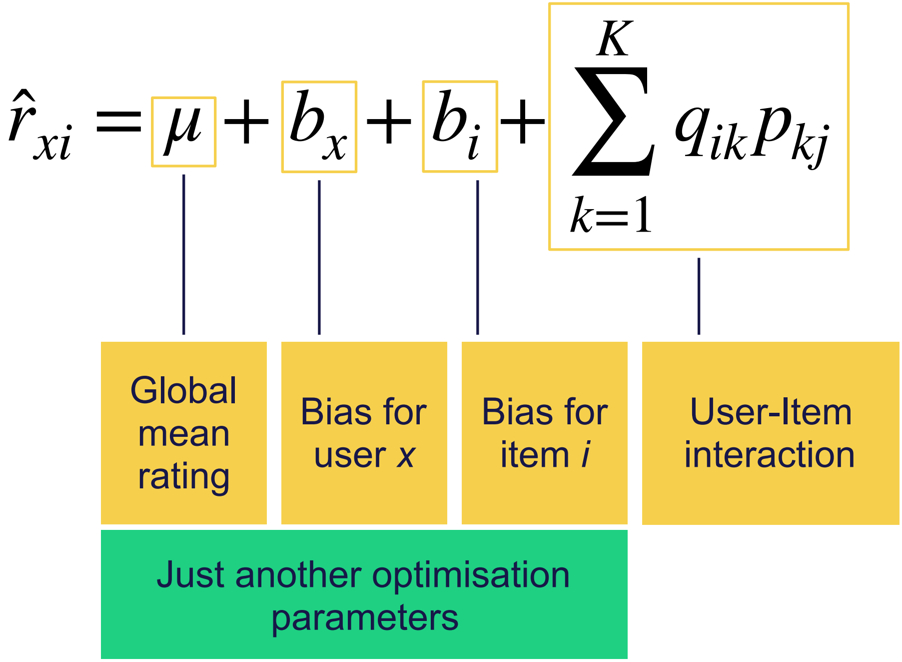
- Separates users and movies
- Benefits from insights into user's behavior
- Among the main practical contributions of the competition

User-Movie interaction

- Characterizes the matching between users and movies
- Attracts most research in the field
- Benefits from algorithmic and mathematical innovations

- μ = overall mean rating
- b_x = bias of user x
- b_i = bias of movie i

Putting It All Together



$$\hat{r}_{xi} = \boxed{\mu} + \boxed{b_x} + \boxed{b_i} + \boxed{\sum_{k=1}^{K} q_{ik} p_{kj}}$$

Global mean rating

Bias for user *x*

Bias for item *i*

User-Item interaction

Just another optimisation parameters

$$J(P,Q) = \sum_{\text{training data}} \sum_{i,x \in R} \left(r_{ix} - (\mu + b_x + b_i) - \sum_{k=1}^{K} q_{ik} p_{kj} \right)^2$$

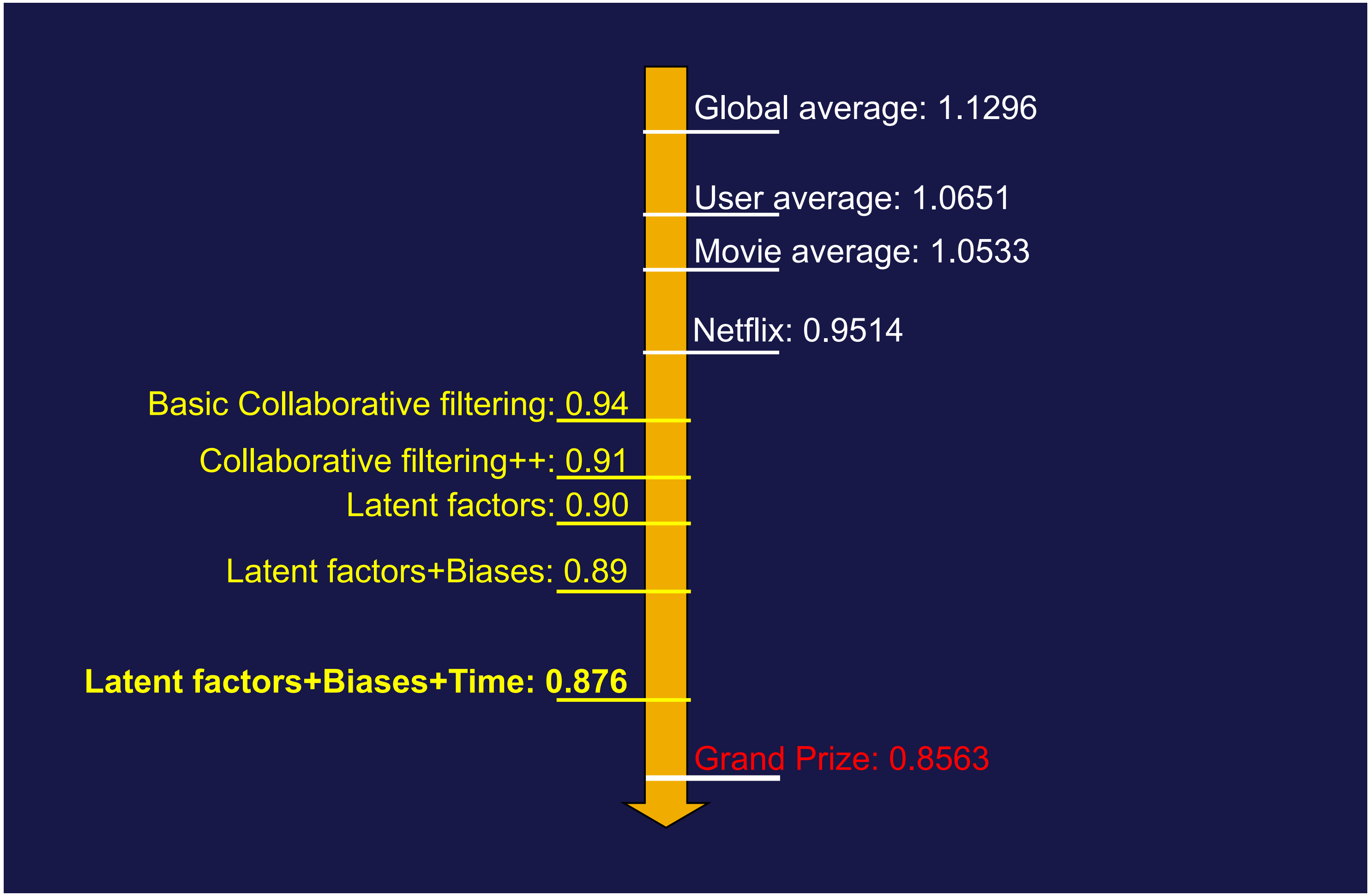
The Netflix Prize (2009)

- **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)
  - **Netflix's system RMSE: 0.9514**

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

Performance of Various Methods



Global average: 1.1296

User average: 1.0651

Movie average: 1.0533

Netflix: 0.9514

Basic Collaborative filtering: 0.94

Collaborative filtering++: 0.91

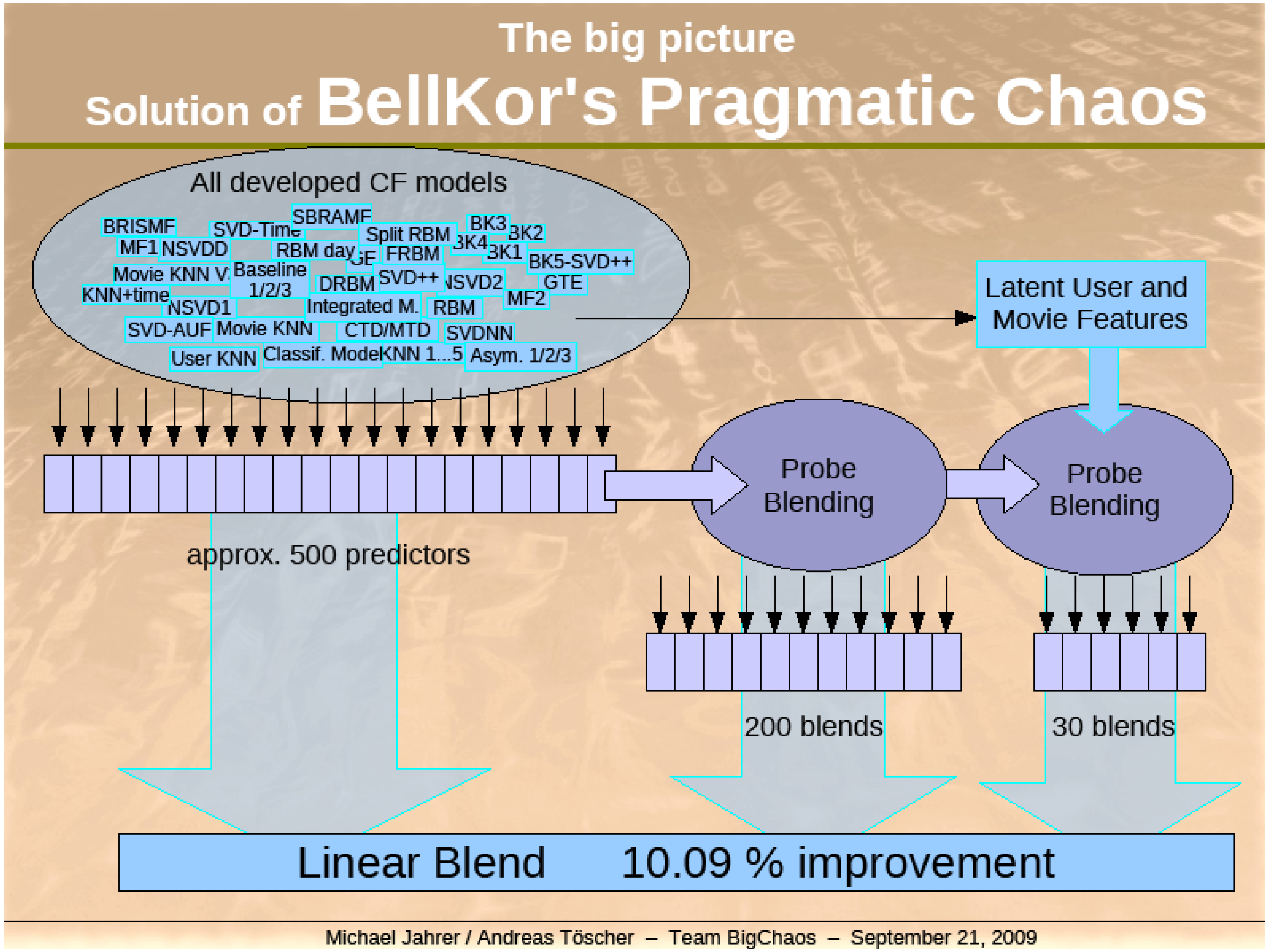
Latent factors: 0.90

Latent factors+Biases: 0.89

Latent factors+Biases+Time: 0.876

Grand Prize: 0.8563

Winner: A “kitchen sink” approach



The big picture

Solution of BellKor's Pragmatic Chaos

Michael Jahrer / Andreas Töscher – Team BigChaos – September 21, 2009

Exercises

- “Recommender Systems.ipynb”: Sections 3.2 - 5

Recommender reading

- Chapter 9, “Mining of Massive Datasets” by Jure Leskovec, Anand Rajaraman, and Jeff Ullman http://infolab.stanford.edu/~ullman/mmds/ch9.pdf