MÁSTER EN DATA SCIENCE

Recommender Systems

Machine Learning

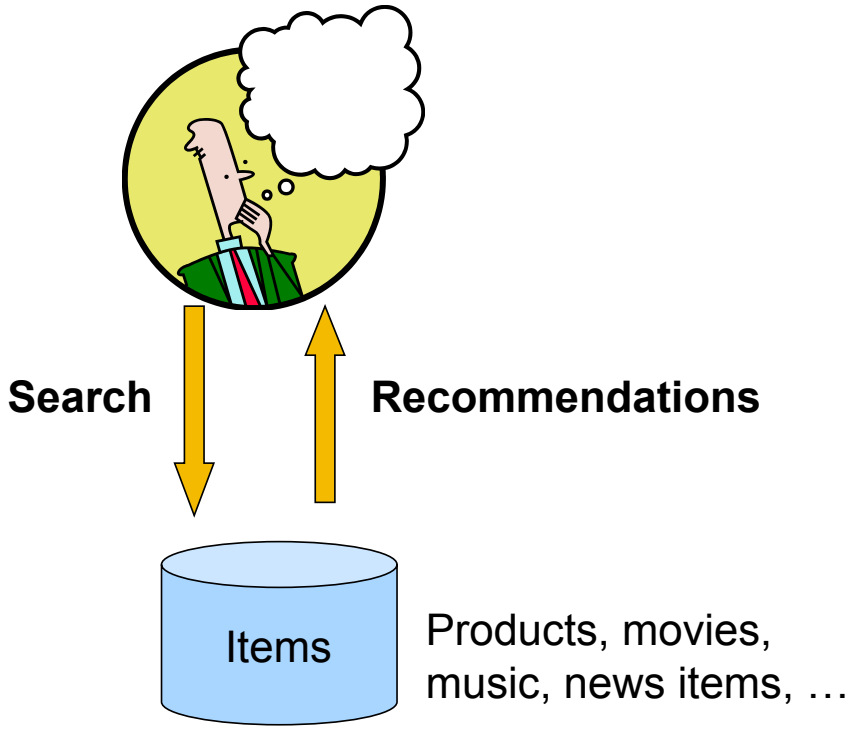
Felipe Alonso-Atienza

Universidad Rey Juan Carlos

Outline

1. Introduction
2. Simple recommenders
3. Content-based recommenders
4. Collaborative Filtering
5. Evaluating recommender systems
6. Latent Factors models
7. Recommenders using Deep Learning
8. Recommenders in action (next lecture)

What's a RS?



Given a set of items S and a set of users U,

we want to recommend item $i \subset S$ to user $u \subset U$

Introduction

- Relatively novel machine learning discipline, which has received low attention in academia during the last years
- However, one of the most popular applications of machine learning in technology companies:
  - Substantial fraction of the revenue
    - Amazon: 35% of total sales
    - Netflix: over 80% of watched TV shows/movies by recommendation
    - YouTube: recommendation algorithm generates more than 70% of views





The origin: the Netflix prize

- From renting DVDs to content creator

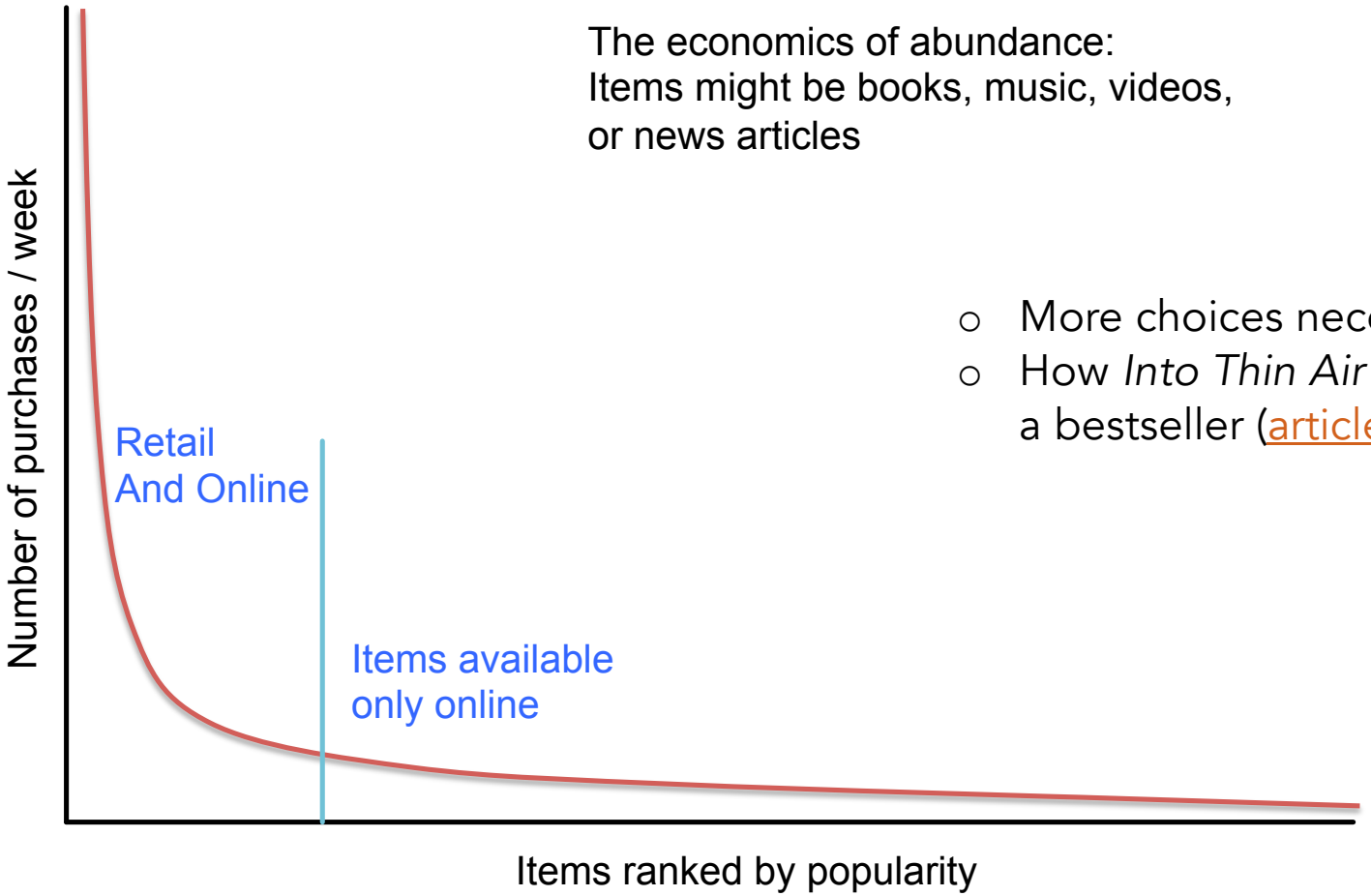
- **1M$ prize**
- Training data
  - 100 million rating,
  - 480.000 users,
  - 17.770 movies
  - 6 years data
- Test data
  - Last few ratings of each user
  - 2.8 million

The Netflix prize

- *"The Netflix Prize abstracted the recommendation problem to a simplified proxy of accurately* ***predicting ratings****. It is now clear that this is just one of many components in an effective industrial recommendation system. They also need to account for factors like* ***diversity, context, evidence, freshness, and novelty****."*

[Xavier Amatriain et. Al. Past, Present, and Future of Recommender Systems: An Industry Perspective, Recsys '16]

Motivation: the long tail



The economics of abundance:
Items might be books, music, videos, or news articles

- More choices necessitate better filters
- How *Into Thin Air* made *Touching the Void* a bestseller (article)

Examples

- **Products**/items
  - Global vs session recommendations

- Need of personalized **content**: coherent and diverse recommendations
  - Video, movies

  - Songs (exploiting music characteristics), radio

  - News

Examples

- People / friends
  - Items are also users



- Search results
  - Pandora vs Pandora

What's a RS?

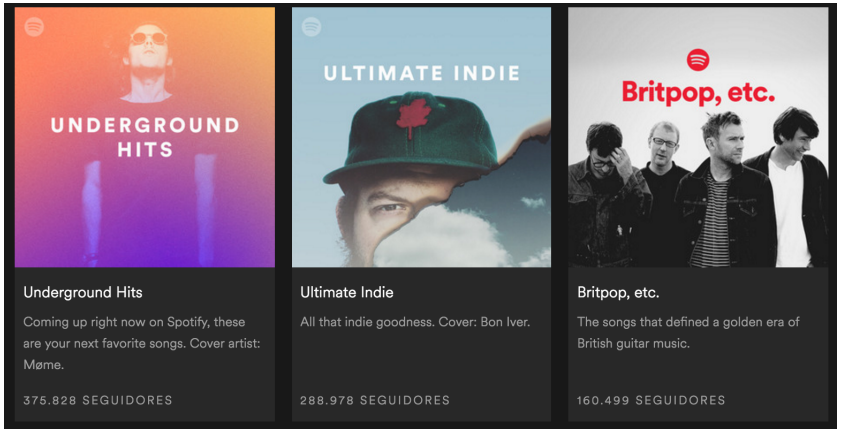
- Provide a list of (top-N) recommendations instead of predicting ratings
- Amazon does not show your predicted rating

Outline

1. Introduction
2. Simple recommenders
3. Content-based recommenders
4. Collaborative Filtering
5. Evaluating recommender systems
6. Latent Factors models
7. Recommenders using Deep Learning
8. Recommenders in action (next lecture)

Non-personalized recommenders

- **Lists**
  - Favorites
  - Essentials



Non-personalized recommenders

- **Recommendations based on popularity**
  - Population behavior can be used to guide suggestions
    - "All people love this, so you might love it too"
    - Most read articles…
    - Most watched video…
    - Most played song…
    - Most recent …
  - Most popular restaurant?
  - Highest rated movie?

Non-personalized recommenders

- Ranking in news aggregators / communities
  - Popularity / age

Personalized recommendations

- How to generate personalized recommendations?
  1. **User** information/tastes
  2. **Item**/product characteristics
  3. **Interaction** between users and items

- How do we know what users would like?
  - Explicit feedback: ask the customer
    - Extra work -> sparse data
    - Different standards
  - Implicit feedback
    - Click data (not reliable, fraud)
    - Purchases (money) / video consumption (time)

Personalized recommendations

- Utility matrix: U x S → R
  - U = set of users/customers
  - S = set of Items/products
  - R = set of ratings; e.g., [0-1]; 1-5 stars.
    - What does an empty cell mean?

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

Personalized recommendations: Key problems

1. **Gathering ratings:**
   I. Asking people
   II. Estimate from user actions (e.g. purchase implies high rating)

2. **Estimate unknown ratings from the known ones**
   I. Most people have not rated most items
   II. Cold start:
      - New items have no ratings
      - New users have no history

3. **Evaluate the success/performance of recommendation methods**

Outline

1. Introduction
2. Simple recommenders
3. **Content-based recommenders**
4. Collaborative Filtering
5. Evaluating recommender systems
6. Latent Factors models
7. Recommenders using Deep Learning
8. Recommenders in action (next lecture)

Content-based systems

- Use properties of the items (metadata) to recommend
  - Movies: **genre**, **year of release**, set of actors, …

Movie	action	adventure	animation	children's	comedy	crime	documentary	drama	fantasy	film-noir	horror	musical	western	mystery	romance	sci-fi	thriller	war	western2
Toy Story	0	1	1	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0
Jumanji	0	1	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
Grumpier Old Men	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0
Waiting to Exhale	0	0	0	0	1	0	0	1	0	0	0	0	0	0	1	0	0	0	0
Father of the Bride	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

```
def computeYearSimilarity(self, movie1, movie2, years):
        diff = abs(years[movie1] - years[movie2])
        sim = math.exp(-diff / 10.0)
        return sim
```

  - sim_total = sim_genre * sim_years

https://sundog-education.com/recsys/

Content-based systems

- Use properties of the items (metadata) to recommend
  - Movies: genre, year of release, set of actors, …

Movie	Alice (1)	Bob (2)	Carol (3)	Dave (4)	x_1 (romance)	x_2 (action)
Love at last	5	5	0	0	0.9	0
Romance forever	5	?	?	0	1.0	0.01
Cute puppies of love	?	4	0	?	0.99	0
Nonstop car chases	0	0	5	4	0.1	1.0
Swords vs. karate	0	0	5	?	0	0.9

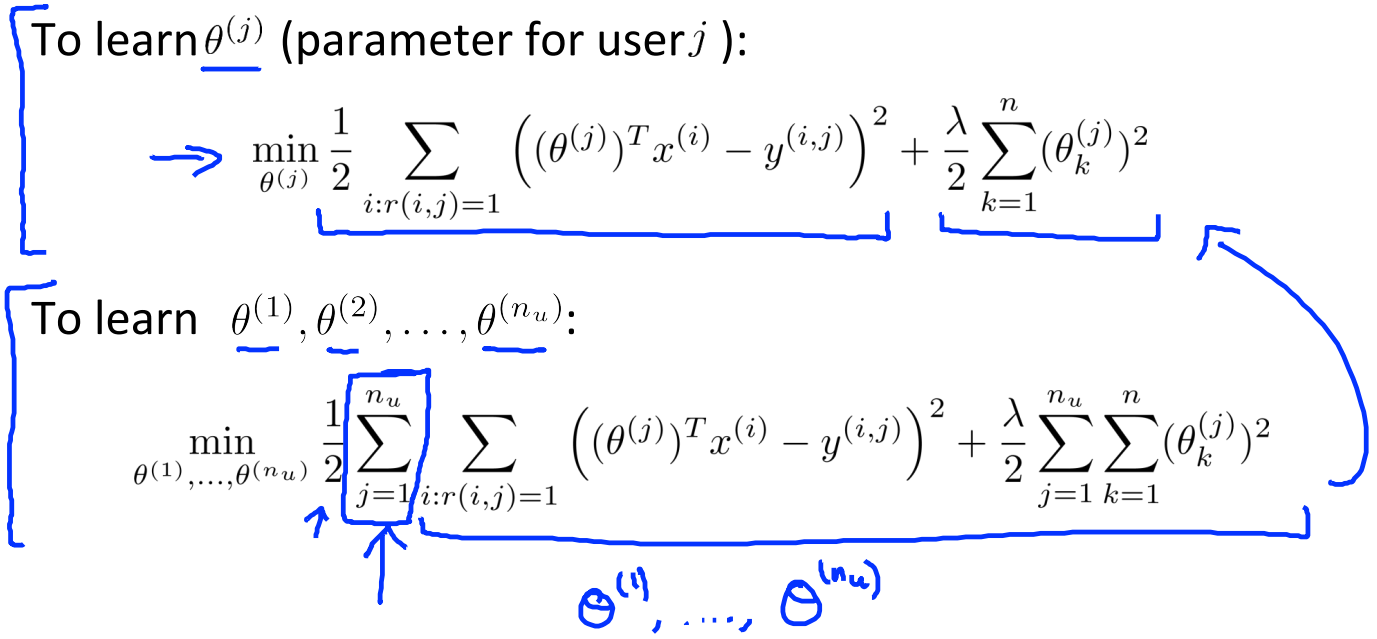
- This item information can be combined with the user profile and the session information:
  - User info: age, purchase history, …
  - Session information: day of the purchase, time of the purchase, etc…

Formulation

- This problem can be formulated as a classical machine learning problem
- Regression problem
  - Estimate unknown ratings for each user based on item features (and on user and session features)
- Classification problem
  - Determine if a user would like/dislike an item based on item features (and on user and session features)

Formulation

- **Regression problem**
  - Estimate unknown ratings for each user based on item features (and on user and session features)



From: Andrew Ng. Machine Learning (Coursera)

Pros & cons

- PROS
  1. Personalized recommendations
  2. Features can capture context (time of the day, what I just saw)
  3. Handles missing information (age of the user, limited purchase history)
  4. No need data for other users (items easy to add)
  5. Explanation for recommended items

- CONS
  1. Features might not be available or the are hard to find
  2. Overspecialization:
     - Never recommends items outside user's content profile
     - Unable to exploit quality judgments of other user
  3. Cold-start problem for new users

Outline

1. Introduction
2. Simple recommenders
3. Content-based recommenders
4. **Collaborative Filtering**
5. Evaluating recommender systems
6. Latent Factors models
7. Recommenders using Deep Learning
8. Recommenders in action (next lecture)

Collaborative Filtering (CF)

- IDEA: exploit information on what other users have done (rated/purchased) on different items

"People who bought this also bought..."

- US PATENT 7113917 B2: Personalized recommendations of items represented within a database, 2006

Collaborative Filtering (CF)

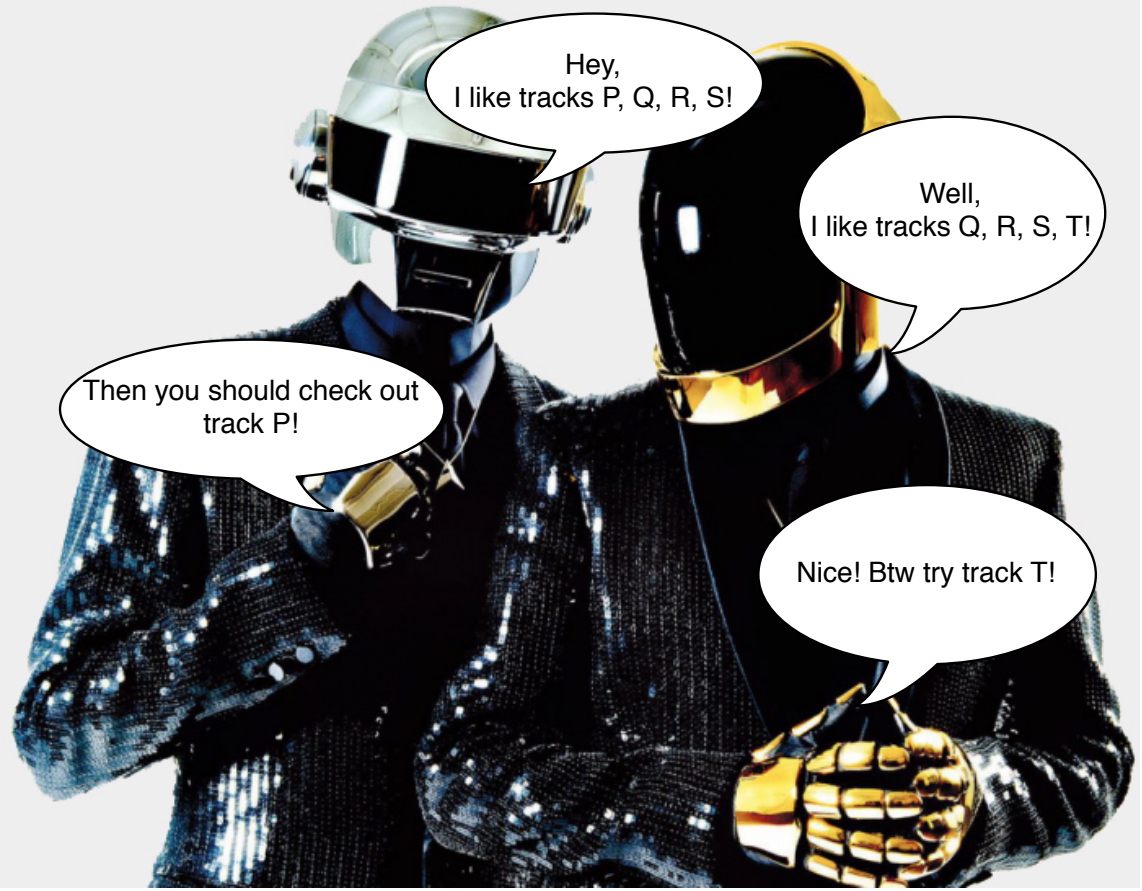


Image via Erik Bernhardsson

CF taxonomy

1. Co-occurrence matrix: "*People who bought this also bought…*"
   a. Beers and diapers
2. Memory-based CF
   a. user-user: "*Users who are similar to you also liked …*"
      - If two users have rated similarly a number of movies, then they are probably similar
   b. item-item: "*Users who liked this movie also liked …*"
      - If two movies get similar ratings from lots of users, then they are probably similar
3. Model-based CF, also known as Latent factor models

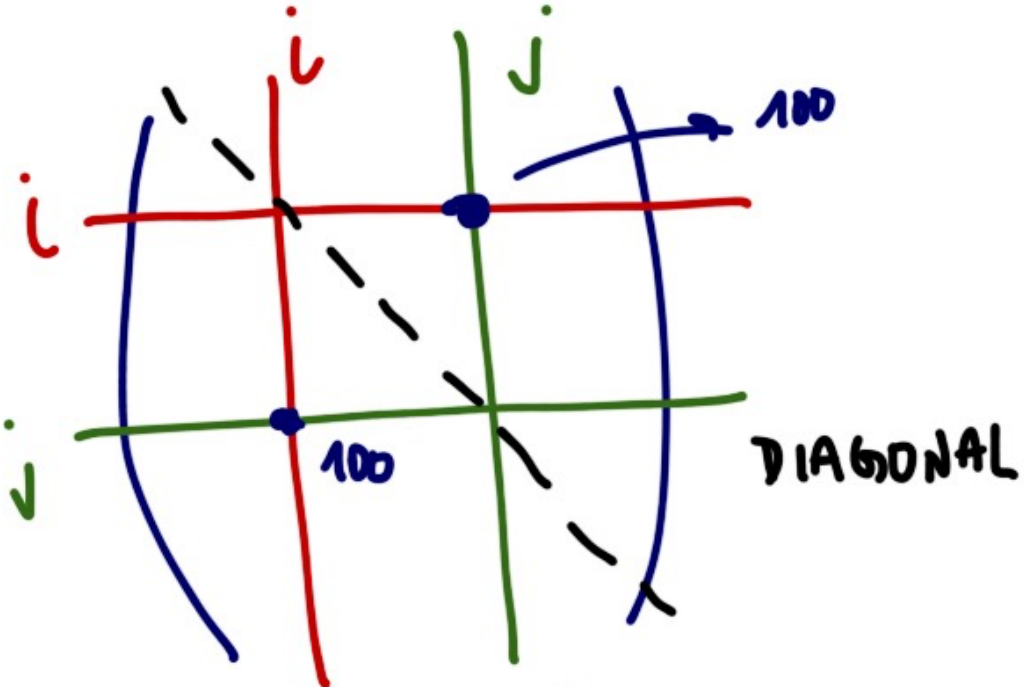
- Memory-based CF:
  - Find similarities among users
  - Find similarities among items

Outline

1. Introduction
2. Simple recommenders
3. Content-based recommenders
4. Collaborative Filtering
   I. Co-occurrence matrix
5. Evaluating recommender systems
6. Latent Factors models
7. Recommenders using Deep Learning
8. Recommenders in action (next lecture)

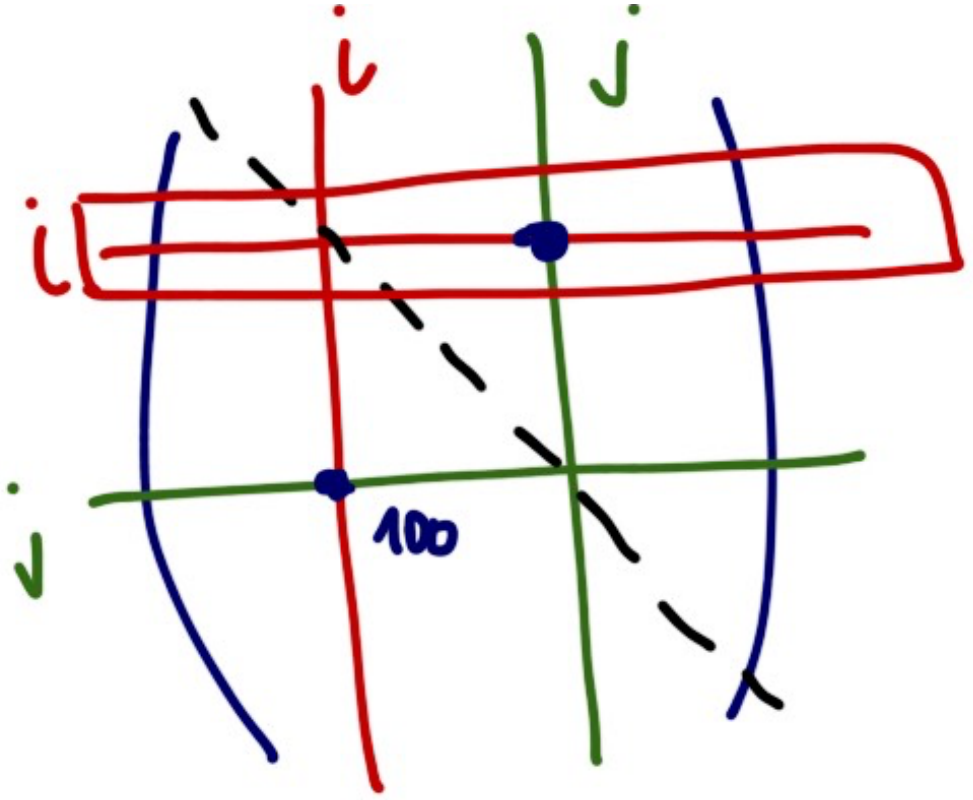
Co-occurrence matrix

- # of users who purchased both items *i* & *j*
  - *i* = mountain boots
  - *j* = mountain jacket



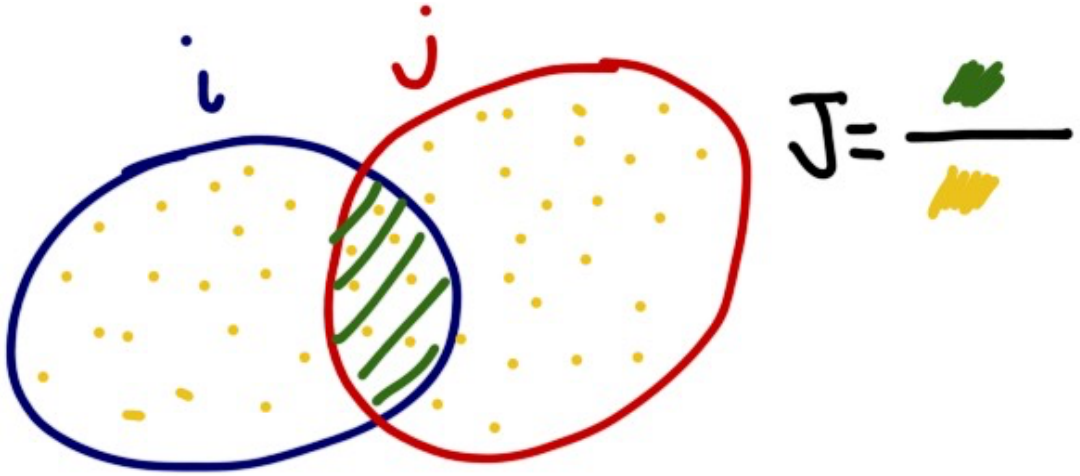
Making recommendations

- Look at the specific row *i*
- Recommend other items with largest counts



Effect of popular items

- What if there are very popular items?
- Normalize co-occurrence Matrix
- Jaccard similarity
  - who purchased **i and j** divided by
  - who purchased **i or j**



Co-occurrence matrix

- Also known as association rules or market basket analysis
  - Mlxtend library

- Cons:
  - Only current situation, no history
    - Weighted average of purchases
  - No personalized features

Outline

1. Introduction
2. Simple recommenders
3. Content-based recommenders
4. Collaborative Filtering
   I. Co-occurrence matrix
   II. user-user CF
5. Evaluating recommender systems
6. Latent Factors models
7. Recommenders using Deep Learning
8. Recommenders in action (next lecture)

user-user CF

- *"Users who are **similar** to you also liked …"*
- Find similarities among users

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

HP = *Harry Potter*
TW = *Twilight* (Crepúsculo)
SW = *Star Wars*

user-user CF: Jaccard Similarity

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

- $J(A,B) = |r_A \cap r_B| / |r_A \cup r_B| = 1 / 5$
- $J(A,C) = 2 / 4$
- $J(A,B) < J(A,C)$ …. expected?

- KO: ignores rating values!

user-user CF: cosine distance

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

- $\cos(A,B) = r_A \cdot r^T_B \ / \ \|r_A\| \cdot \|r_B\| = 5*4 \ / \ (\sqrt{42} * \sqrt{66}) = 0.38$
- $\cos(A,C) = 0.32$

- KO: treats missing ratings as negative

user-user CF: centered cosine

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

	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

- Normalize ratings by subtracting row mean

user-user CF: centered cosine

	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

- cos(A,B) = 0.09
- cos(A,C) = -0.56

- Treats missing ratings as average
- Also known as Pearson Correlation

Making recommendations

- Let r_u be the vector of user u's ratings
- The prediction for user u and item i

$$\hat{r}_{ui} = \frac{\sum_{u'} sim(u, u') r_{u'i}}{\sum_{u'} |sim(u, u')|}$$

- Options:
  1. Use only the set of k users most similar to u
  2. Taking into account user bias

$$\hat{r}_{ui} = \bar{r}_u + \frac{\sum_{u'} sim(u, u')(r_{u'i} - \bar{r}_{u'})}{\sum_{u'} |sim(u, u')|}$$

Outline

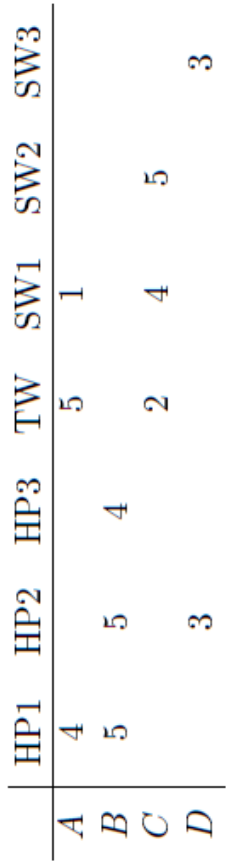
1. Introduction
2. Simple recommenders
3. Content-based recommenders
4. Collaborative Filtering
   I. Co-occurrence matrix
   II. user-user CF
   III. item-item CF
5. Evaluating recommender systems
6. Latent Factors models
7. Recommenders using Deep Learning
8. Recommenders in action (next lecture)

item-item CF

- *"Users who liked this movie also liked …"*
- **Find similarities among items**

- Same procedure as un user-user CF
  - Transpose your utility matrix

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



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

Outline

1. Introduction
2. Simple recommenders
3. Content-based recommenders
4. Collaborative Filtering
   I. Co-occurrence matrix
   II. user-user CF
   III. item-item CF
   IV. Global baseline
5. Evaluating recommender systems
6. Latent Factors models
7. Recommenders using Deep Learning
8. Recommenders in action (next lecture)

CF + Global baseline

- Model local and global effects
- We want to estimate Joe's rating for the movie The Sixth Sense
- Global Baseline approach
  - Mean movie rating: **3.7 stars**
  - Joe rates **0.2** stars below avg.
  - The Sixth Sense is **0.5** stars above avg.
  - Baseline estimate: 3.7 + 0.5 – 0.2 = 4 stars

$$b_{ui} = \mu + b_u + b_i = 3.7 - 0.2 + 0.5$$

$$\hat{r}_{ui} = b_{ui} + \frac{\sum_{u'} sim(u, u')(r_{u'i} - b_{u'i})}{\sum_{u'} |sim(u, u')|} \qquad \hat{r}_{ui} = b_{ui} + \frac{\sum_{j} sim(i, j)(r_{uj} - b_{uj})}{\sum_{j} |sim(i, j)|}$$

CF Comparison

- **Item-item**
  1. More stable (allows pre-calculations)*.
  2. Useful when having more users than items.
  3. Easy to justify recommendations
     - *"you get a recommendation for movie Y because you liked movie X, which is similar to movie Y"*

- **User-user**
  1. Better way to give recommendations
     - Serendipity

* Linden et al. "Amazon.com Recommendations Item to Item Collaborative Filtering"

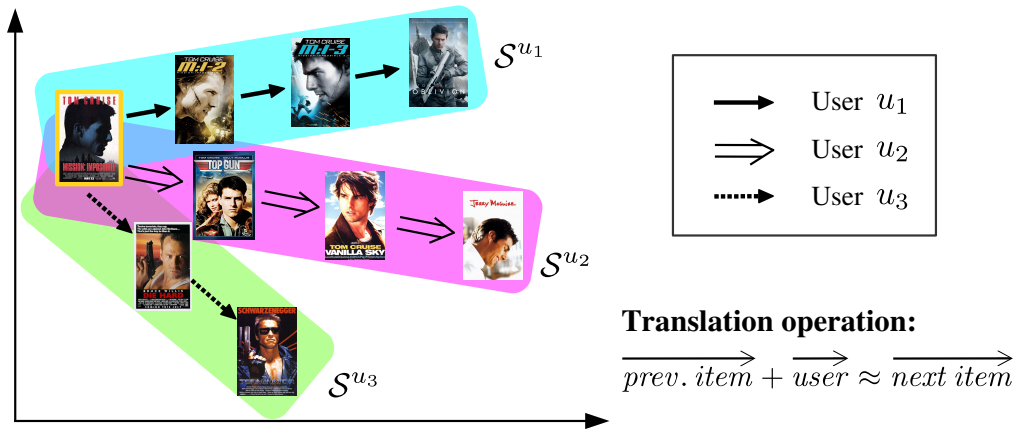
CF Comparison Pros & cons

- PROS
  1. Easy to compute, specially successful in generating personalized recommendations.
  2. Works for any kind of item

- CONS
  1. Utility matrix is sparse:
     - can lead to bad prediction if the article is unpopular and very few users have given feedback about them
  2. It is difficult to find related users or items for users with extraordinary taste
  3. Cold start (at least 20+ ratings before produce accurate recommendations)

CF possible variations

- Only consider items with high rating
- Only consider user with high similarity score
- Consider sequence of events:
  - What would be the next movie to watch?



https://cseweb.ucsd.edu/~jmcauley/pdfs/recsys17.pdf

Outline

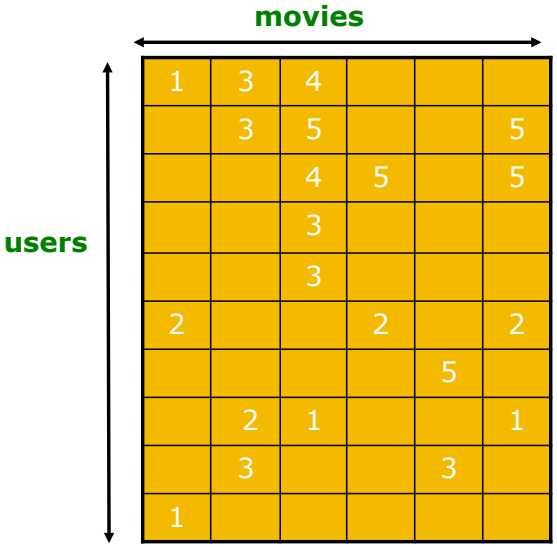
1. Introduction
2. Simple recommenders
3. Content-based recommenders
4. Collaborative Filtering
5. **Evaluating recommender systems**
6. Latent Factors models
7. Recommenders using Deep Learning
8. Recommenders in action (next lecture)

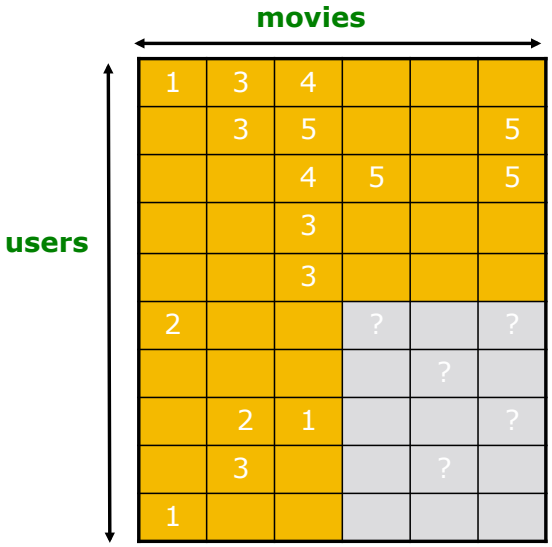
Evaluation (GO TO Slide 17)

- Off-line
  - Regression: RMSE, MAE
  - Classification:
    - Be on top-N list
    - Rating > threshold
  - Ranking: precision@k, Kendall's tau.
  - Recommenders:
    - Hit rate, average reciprocal hit rate, coverage, diversity
- On-line
  - A/B testing

Evaluation: RMSE

- Popular metric: RMSE over a test set (T)





$$\mathrm{RMSE} = \sqrt{\frac{\sum_{(u,i)\in T} \left(r_{ui} - \hat{r}_{ui}\right)^2}{N_T}}$$

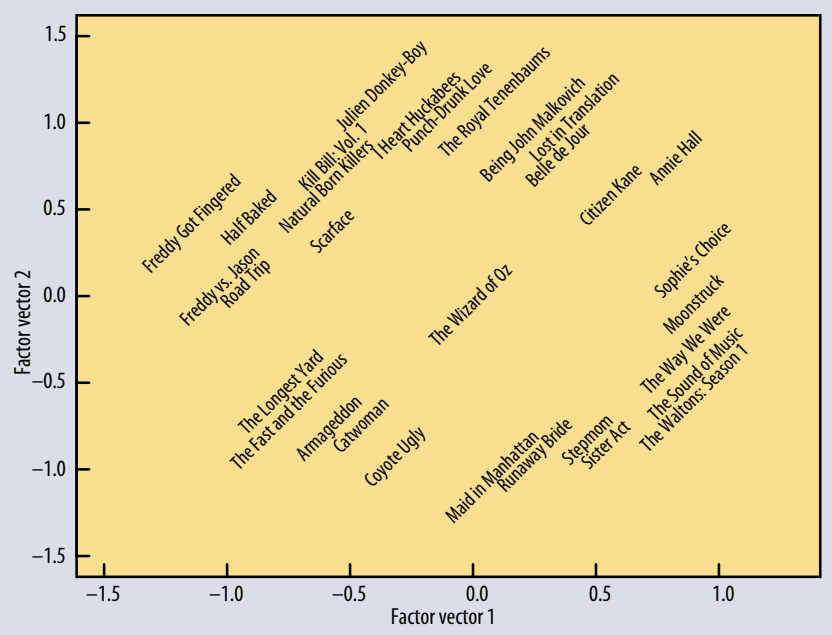
- KO:
  - lack of prediction context
  - Same penalization for high and low ratings

Outline

1. Introduction
2. Simple recommenders
3. Content-based recommenders
4. Collaborative Filtering
5. Evaluating recommender systems
6. Latent Factors models
7. Recommenders using Deep Learning
8. Recommenders in action (next lecture)

Latent factors

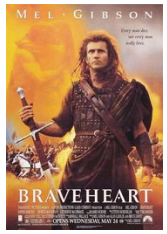
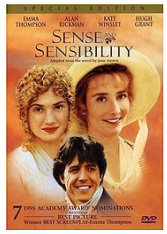
- Idea:
  - Ratings are influenced by a set of factors (e.g. amount of action in movies, comedy vs drama)
  - These factors are not obvious



  - The goal is to infer those so called latent factors from the rating data

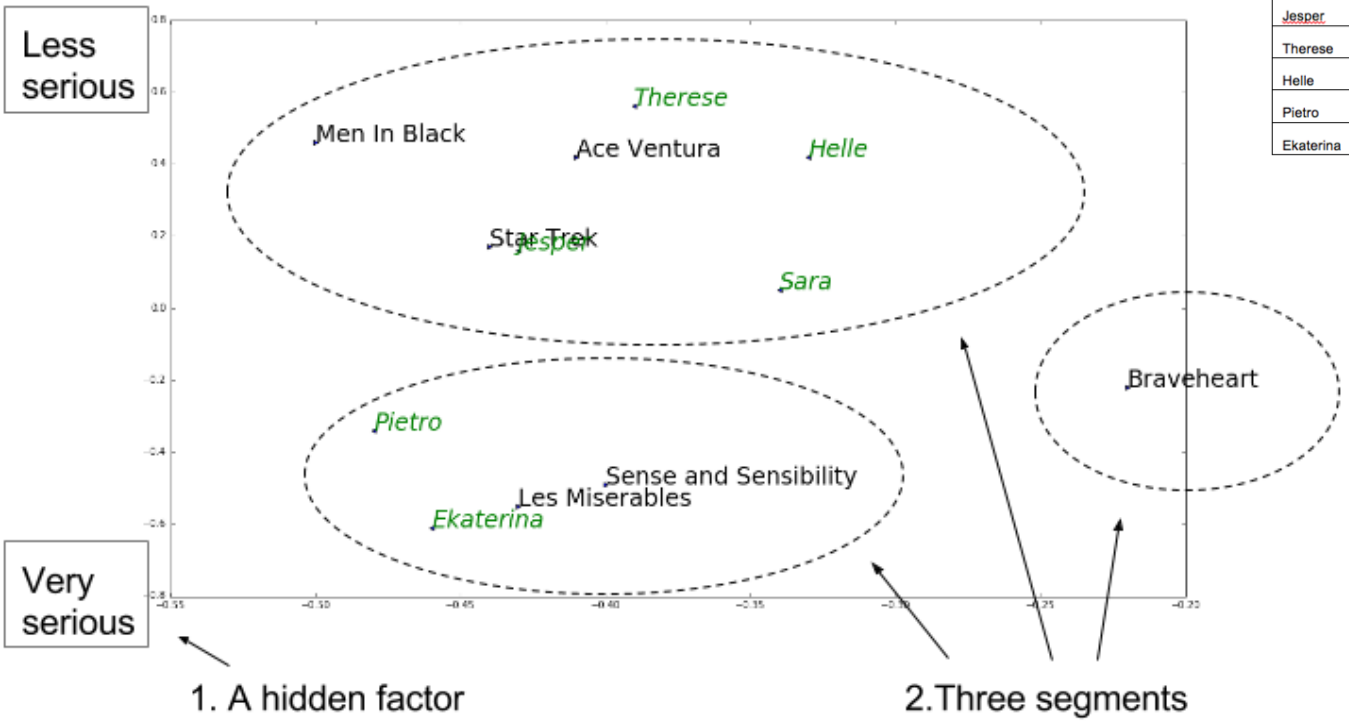
Latent factors

- Idea: matrix factorization

						
	Comedy	Action	Comedy	Action	Drama	Drama.
Sara	5	3		2	2	2
Jesper	4	3	4		3	3
Therese	5	2	5	2	1	1
Helle	3	5	3		1	1
Pietro	3	3	3	2	4	5
Ekaterina	2	3	2	3	5	5

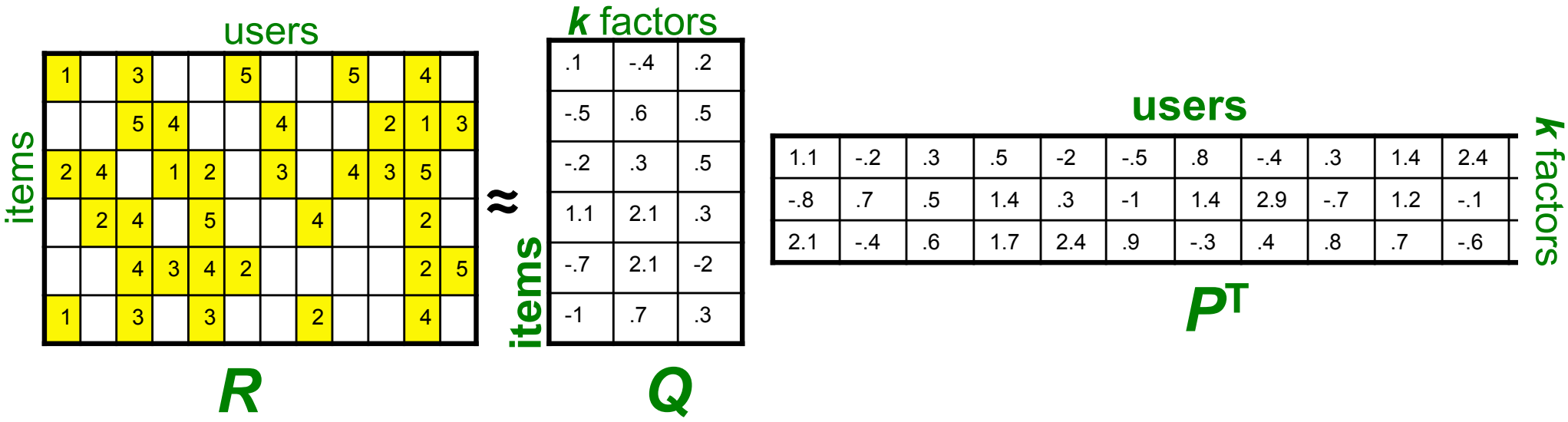
Latent factors

- Idea: matrix factorization



Latent factors

- IDEA: the utility matrix R can be factorized, such that $R = Q \cdot P^T$



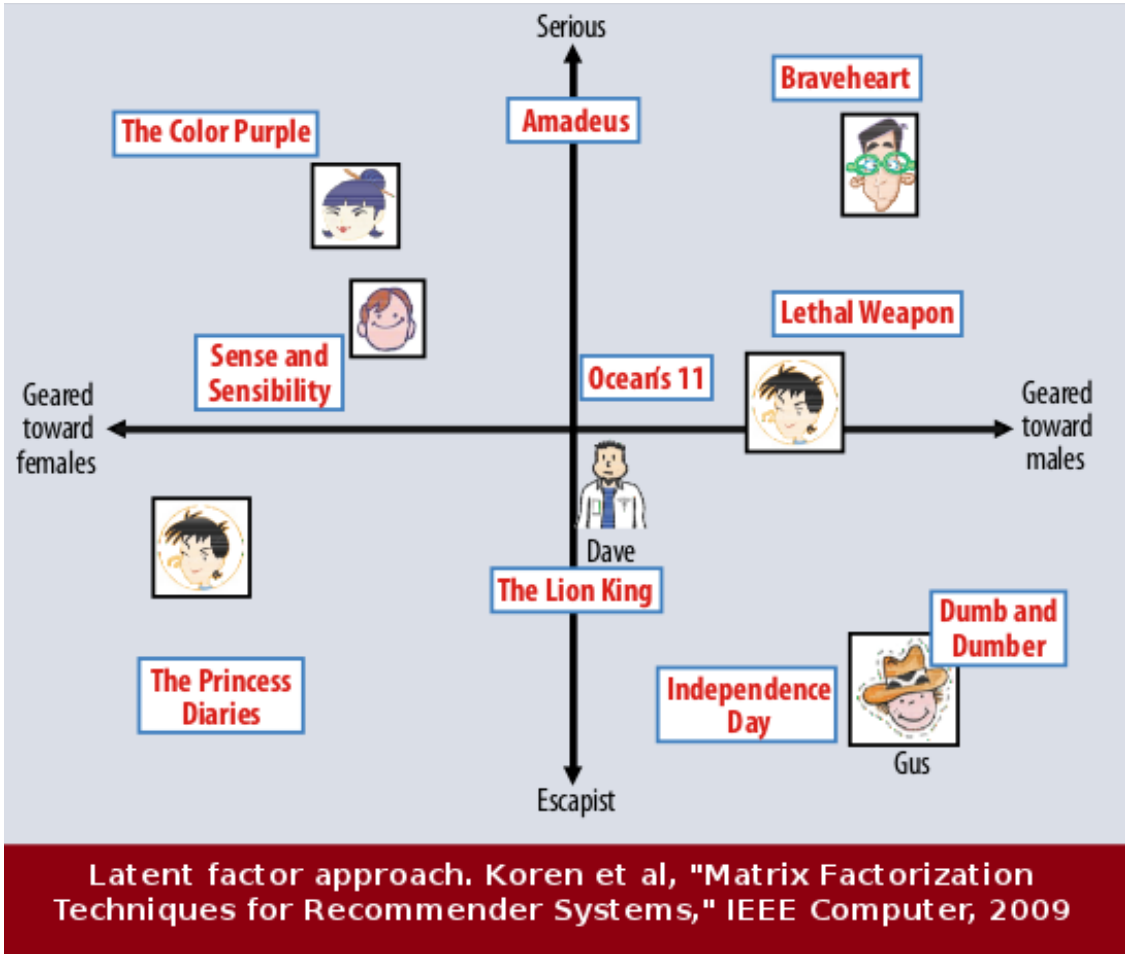
1		3			5			5		4	
		5	4			4			2	1	3
2	4		1	2		3		4	3	5	
	2	4		5			4			2	
		4	3	4	2					2	5
1		3		3			2			4	

.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

- Factors = new features
- They might not necessarily be intuitive understandable

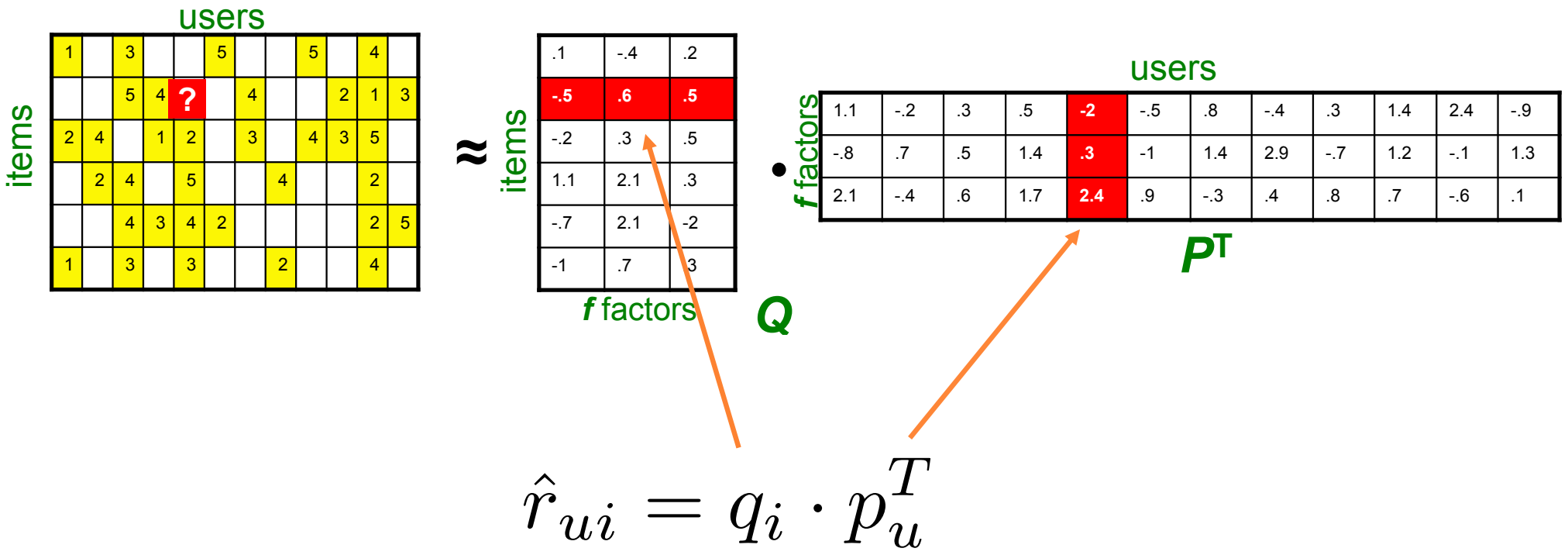
Latent factors



Latent factor approach. Koren et al, "Matrix Factorization Techniques for Recommender Systems," IEEE Computer, 2009

Rating estimation

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



$$\hat{r}_{ui} = q_i \cdot p_u^T$$

Problem formulation

- **How to find *Q* and P^T?**

- SVD could be an option [U,S,V] = svd(R)
  - $R = U \cdot S \cdot V^T$
  - Q = U and $P^T = S \cdot V^T$
  - KO: SVD is not defined when entries are missing

- Regularized solution

$$\min_{q^*,p^*} \sum_{(u,i)\in\kappa} (r_{ui} - q_i^T p_u)^2 + \lambda(|| q_i ||^2 + || p_u ||^2)$$

which can be minimized using:

- Alternating least squares
- Stochastic gradient descent

Adding biases

- Static global effects

$$\hat{r}_{ui} = \mu + b_i + b_u + q_i^T p_u$$

$$\min_{p^*, q^*, b^*} \sum_{(u,i) \in \kappa} (r_{ui} - \mu - b_u - b_i - p_u^T q_i)^2 + \lambda$$
$$(|| p_u ||^2 + || q_i ||^2 + b_u^2 + b_i^2)$$

- Temporal dynamics

$$\hat{r}_{ui}(t) = \mu + b_i(t) + b_u(t) + q_i^T p_u(t)$$

  - Rise in the average movie rating (Netflix)
  - Users prefer new movies without any reason

Other approaches

SLIM: Sparse Linear Methods for Top-N Recommender Systems

Xia Ning and George Karypis
Computer Science & Engineering
University of Minnesota, Minneapolis, MN
Email: {xning,karypis@cs.umn.edu}

http://glaros.dtc.umn.edu/gkhome/node/774

Outline

1. Introduction
2. Simple recommenders
3. Content-based recommenders
4. Collaborative Filtering
5. Evaluating recommender systems
6. Latent Factors models
7. Recommenders using Deep Learning
8. Recommenders in action (next lecture)

Deep Learning RS

- Restricted Boltzmann Machines for collaborative filtering
  - Generative (autoencoder)

- Autorec
  - Denoising autoencoder

Deep Learning RS

- Deep Matrix Factorization
  - CTR prediction
  - Embeddings

```
#Using keras
emb_u = Embedding(N,K)
emb_i = Embedding(M,K)

u = input()
i = input()

eu = emb_u(u)
ei = emb_i(i)

r = dot(eu,ei)
```

Deep Learning RS

- RNN
  - What comes next?

Outline

1. Introduction
2. Simple recommenders
3. Content-based recommenders
4. Collaborative Filtering
5. Evaluating recommender systems
6. Latent Factors models
7. Recommenders using Deep Learning
8. Recommenders in action (next lecture)

References

- J. Leskovec, A. Rajaraman, J. Ullman. Mining of Massive Datasets
- Recommender Systems, The Textbook. CC Agarwal. Springer 2016.
- Practical Recommender Systems, Kim Falk. Manning 2019.
- Building Recommender Systems with Machine Learning and AI. Frank Kane.
- Recommender Systems and Deep Learning in Python. Lazy Programer Inc., Udemy