Mining Massive Datasets

Lecture 3

Artur Andrzejak
http://pvs.ifi.uni-heidelberg.de

RUPRECHT-KARLS-
UNIVERSITÄT
HEIDELBERG

Note on Slides

A substantial part of these slides come (either verbatim or in a modified form) from the book Mining of Massive Datasets by Jure Leskovec, Anand Rajaraman, Jeff Ullman (Stanford University). For more information, see the website accompanying the book: http://www.mmds.org.

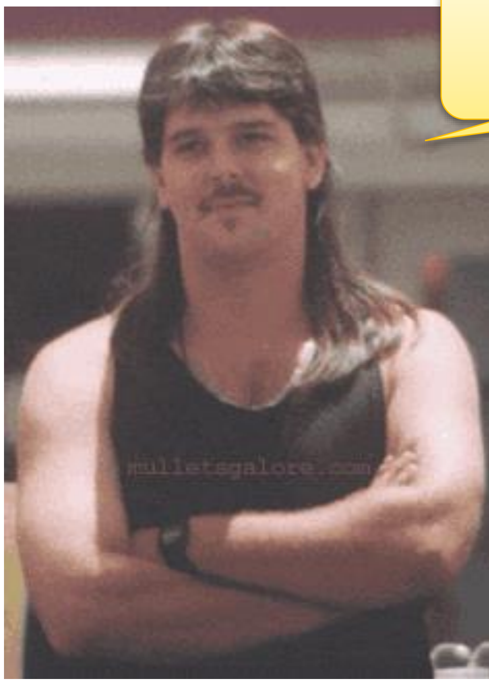
Current Topic

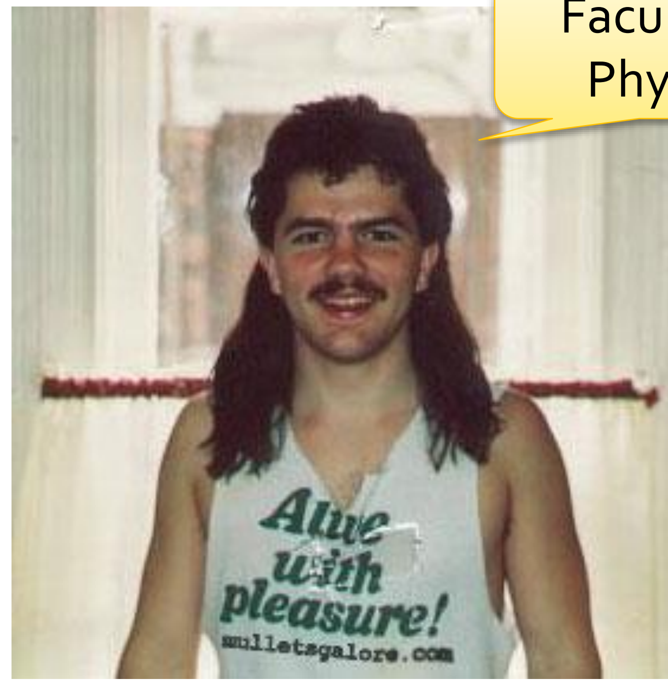
High dim. data	Graph data	Infinite data	Machine learning	Apps
Locality sensitive hashing	PageRank, SimRank	Filtering data streams	SVM	Recommender systems
Clustering	Community Detection	Web advertising	Decision Trees	Association Rules
Dimensio-nality reduction	Spam Detection	Queries on streams	Perceptron, kNN	Duplicate document detection

Programming in Spark & MapReduce

Recommender Systems: Motivation and Overview

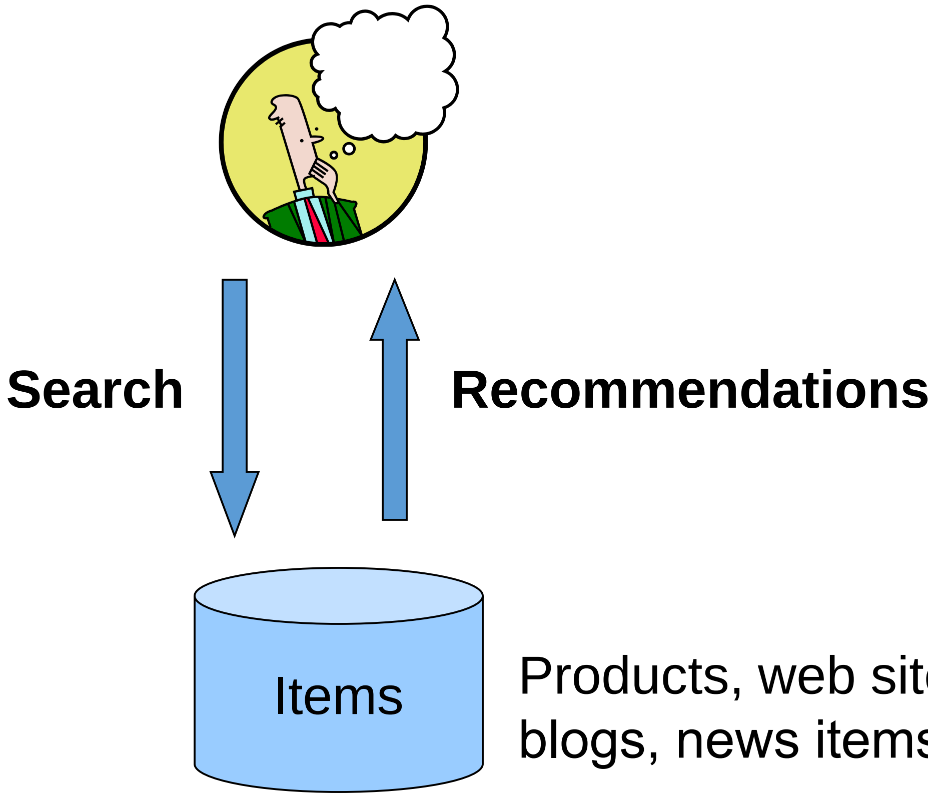
Recommender Systems - Example





- **Scholar X**
  - Buys Metallica CD
  - Buys Megadeth CD
- **Scholar Y**
  - Does search on Metallica
  - Recommender system suggests Megadeth
    - From data about **X**

Recommender Systems



Products, web sites, blogs, news items, …

Examples:

amazon.com
PANDORA
StumbleUpon
NETFLIX
del.icio.us
movielens helping you find the *right* movies
last.fm the social music revolution
Google News
YouTube
XBOX LIVE

Why using Recommender Systems?

- Value for users
  - Find things that are interesting
  - Narrow down the set of choices
  - Discover new things …

- Value for providers
  - Increase trust and customer loyalty
  - Increase sales, click rates, conversion etc.
  - Opportunities for promotion

Formal Model

- $\boldsymbol{X}$ = set of Customers
- $\boldsymbol{S}$ = set of Items

- Utility function $\boldsymbol{u}$: $\boldsymbol{X} \times \boldsymbol{S} \rightarrow \boldsymbol{R}$
  - $\boldsymbol{R}$ = set of ratings
  - $\boldsymbol{R}$ is a totally ordered set
  - e.g., **0-5** stars, real number in **[0,1]**

Utility Matrix

users

movies

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

Utility Matrix

Goal: ■ - estimate rating of movie **1** by user **5**

users

items (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	

Key Problems

- (1) **Gathering “known” ratings for matrix**
  - How to collect the data in the utility matrix
- (2) **Extrapolate unknown ratings from the known ones**
  - Mainly interested in what people like (high scores)
  - Key problem: Utility matrix ***U*** is sparse
    - Cold start: New items have no ratings / New users have no history
- (3) **Evaluating extrapolation methods**
  - How to measure success/performance of recommendation methods

Overview of the Approaches

A. Collaborative filtering (CF)

- Two Versions: user-user and item-item

B. Content-based recommenders

- Difficult in practice due to manually drafted features

C. Latent factor models (LF)

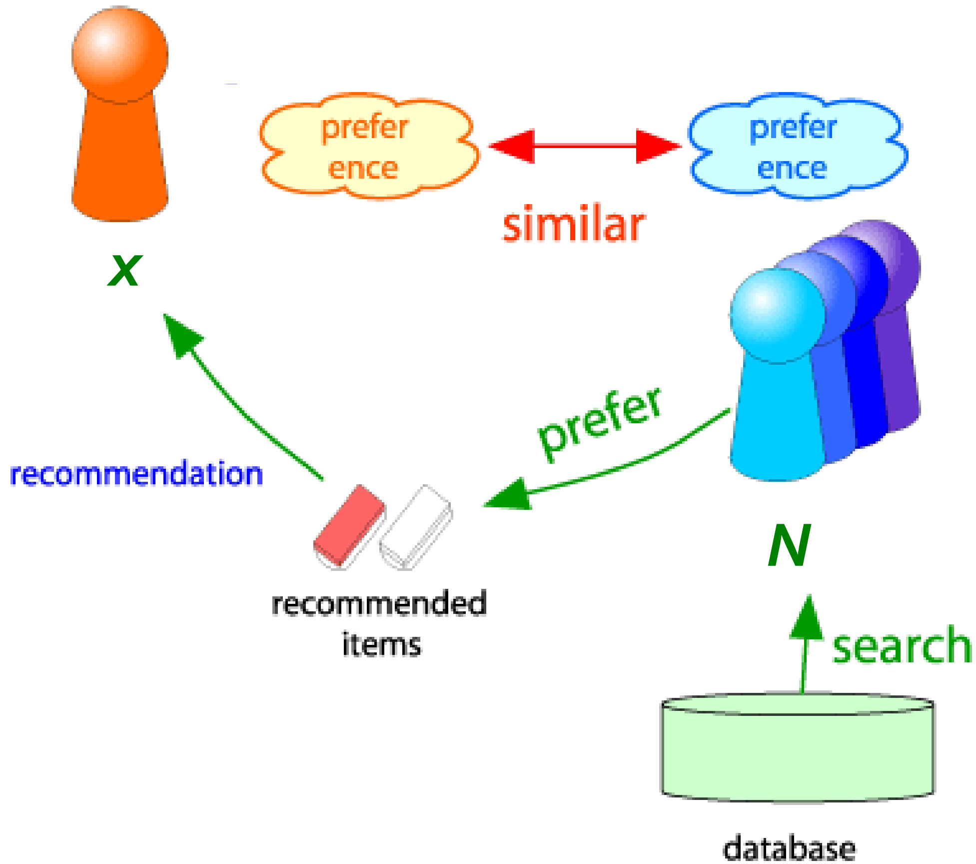
- Improves on B by finding latent features *automagically*

A1. Collaborative Filtering

Version: User-User

User-User Collaborative Filtering

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



Algorithm for User-User CF

- Let r_x be the vector of user x's ratings
- Let N be the set of k "neighbors" of x
  - = users most similar to x (by metric $sim(x, y)$, later)
  - … who have already rated item i
- The predicted rating of user x for item i is:

$$r_{xi} = \frac{1}{k} \sum_{y \in N} r_{yi}$$

Average rating of k "neighbors" of x for item i

Rating Predictions - Improved

- Let r_x be the vector of user x's ratings
- Let N be the set of k "neighbors" of x
  - =users most similar to x (by $sim(x, y)$)
  - … who have already rated item i
- The predicted rating of user x for item i is:

$$r_{xi} = \frac{\sum_{y \in N} sim(x, y) \cdot r_{yi}}{\sum_{y \in N} |sim(x, y)|}$$

Improvement here: we scale the recommendations of other user y by similarity $sim(x, y)$ of x to this "neighbor" y

Note: Expression $sim(x, y)$ for $y \in N$ should be positive since y and x are by definition similar; but for general case, use $|sim(x,y)|$ in denominator!

Example: User-User CF

Computing $\boldsymbol{N}$ = the set of k users <u>most similar</u> to x

- According to $sim(x, y)$, y's have already rated item i

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

users

items	1	2	3	4	**5**	6
1	1		3		?	5
2			5	4		
3	2	4		1	2	
4		2	4		5	
5			4	3	4	2
6	1		3		3	

- 1. Find all users y who have already rated item 1

Example: User-User CF

Computing N = the set of k users most similar to x

- According to $sim(x, y)$, y's have already rated item i

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

users

items	1	2	3	4	**5**	6
1	1		3		?	5
2			5	4		
3	2	4		1	2	
4		2	4		5	
5			4	3	4	2
6	1		3		3	

- 1. Find all users y who have already rated item 1
  - Solution: $y \in \{1, 3, 6\}$
- 2. Compute $sim(x, y)$ for users y
  - Let $s_{5,1} = sim(5,1)$, $s_{5,3} = \ldots$

Example: User-User CF

Computing N = the set of k users <u>most similar</u> to x

- According to $sim(x, y)$, y's have already rated item i

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

users

items	1	2	3	4	**5**	6
1	1		3		?	5
2			5	4		
3	2	4		1	2	
4		2	4		5	
5			4	3	4	2
6	1		3		3	

- 1. Find all users who have already rated item 1
- 2. For such users y (here $y \in \{1, 3, 6\}$) compute $sim(x, y)$
  - Let $s_{5,1} = sim(5,1)$, $s_{5,3} = \dots$
- 3. Among $\{s_{5,1}, s_{5,3}, s_{5,6}\}$, find top k values (say k=2)
  - E.g. $s_{5,3}, s_{5,6}$ largest => $N = \{3,6\}$

Finding "Similar" Users by Cosine

- Let $\boldsymbol{r_z}$ be the column of user $\boldsymbol{z}$'s ratings:

$\boldsymbol{r_x}$ = [*, _, _, *, ***]
$\boldsymbol{r_y}$ = [*, _, **, **, _]

$\boldsymbol{r_x}, \boldsymbol{r_y}$ *as vectors, NaN's -> 0's:*
$\boldsymbol{r_x}$ = {1, 0, 0, 1, 3}
$\boldsymbol{r_y}$ = {1, 0, 2, 2, 0}

  - Turn stars to numbers, fill missing values with 0's
  - => We can consider r_x, r_y as vectors
- Use **Cosine similarity measure**
  - $sim(\boldsymbol{x}, \boldsymbol{y}) = \cos(\boldsymbol{r_x}, \boldsymbol{r_y}) = \frac{r_x \cdot r_y}{||r_x|| \cdot ||r_y||}$
  - $\cos(\boldsymbol{r_x}, \boldsymbol{r_y})$ is just the angle bw. r_x, r_y, from 1 (=similar) to -1 ("opposite")

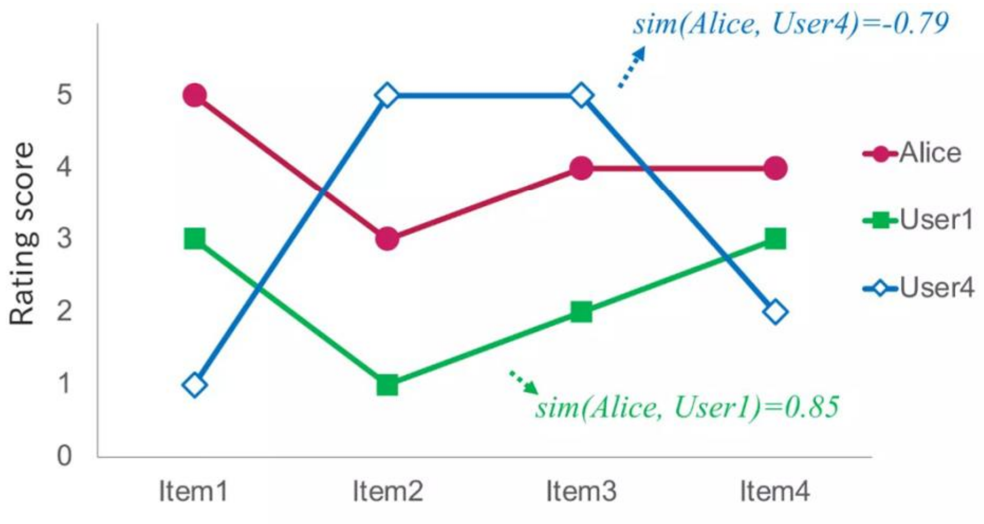
Problems with Cosine

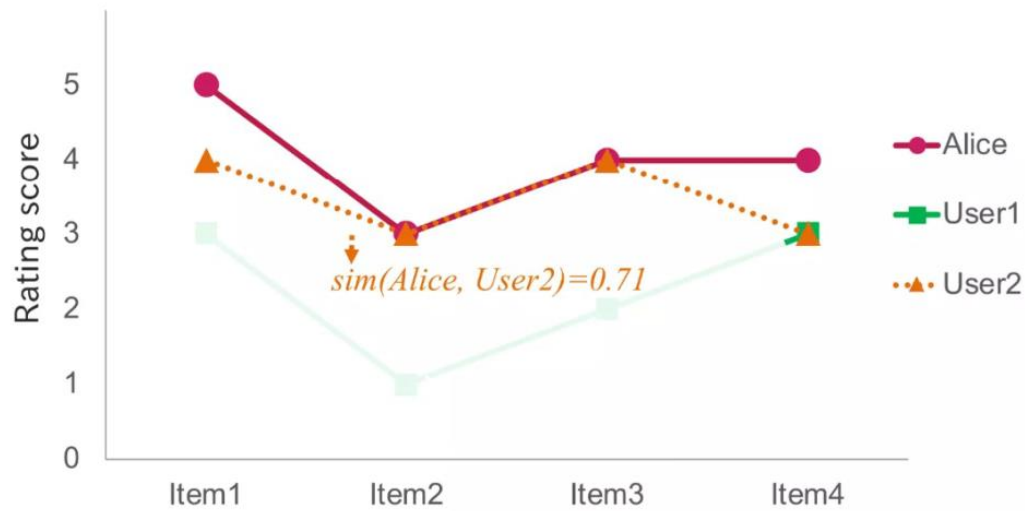
- **Cosine similarity measure**
  - $sim(\boldsymbol{x}, \boldsymbol{y}) = \cos(\boldsymbol{r}_x, \boldsymbol{r}_y) = \frac{r_x \cdot r_y}{||r_x|| \cdot ||r_y||}$

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

- Problem 1: Missing ratings become low ratings!
  - 0's are interpreted as (very) low ratings
  - Items not rated by a user are considered as <u>disliked</u>!
- Problem 2: Users have different bias
  - I.e. some give higher ratings on average, others low ratings on average

Example: User Bias vs. Similarity





From Cosine to Pearson

$\boldsymbol{r_x}$ = [*, _, _, *, ***]
$\boldsymbol{r_y}$ = [*, _, **, **, _]

- Solution for problems 1 & 2
  - For P1, *consider only parts of the vectors* r_x, r_y
    - Vector entries for items rated by <u>both</u> users x and y
  - For P2, *normalize* or *center* each vector: substract average user's rating
- => Use **Pearson correlation coefficient** on partial vectors
  - Pearson CC is computed as cosine similarity between <u>centered</u> vectors
- Definition and formula
  - $\boldsymbol{S_{xy}}$ = items rated by both users x and y
  - $\bar{r}_x$, $\bar{r}_y$,… = average ratings of users x, y, …

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

Computing Most Similar Users

- For recommendations, we use only most similar users, or “neighbors” $\mathrm{N}(x)$ of x

- A. Set a threshold for user similarity
  - If a user has higher similarity than a threshold, he/she can be regarded as a “similar” user

- B. Focus on top k similar users (kNN method)
  - If a user ranks at the top k similarity, he/she can be regarded as a similar user
  - k is often set to between 50 ~ 200
  - In worst cases, a system uses rating information of users with low similarity

Computing Predictions

- The predicted rating for item i and user x is

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

- But it is good consider the bias or baseline of user x:

$$r_{xi} = \overline{\boldsymbol{r_x}} + \frac{\sum_{y \in N(x)} sim(x,y) \cdot (r_{yi} - \overline{\boldsymbol{r_x}})}{\sum_{y \in N(x)} sim(x,y)}$$

A2. Collaborative Filtering

Version: Item-Item

Problems with User-Based CF

- Lack of data: If users haven’t rate the same items yet, user similarity cannot be computed
- Instability
  - It is rare that two users rated the same item
    - => User sim. drastically changes with few new ratings
  - User preferences (user features) often change, while item features do not often change
- Computational cost
  - In general, #Users >> #Items => high cost of finding nearest neighbors (similar users)

Item-Item Collaborative Filtering

- For item i, find similar items rated by user x
  - Let $N(i;x)$ be the set of such items
  - We can use similarity metrics $sim()$ as before
- Estimate rating of user x for item i *based on her/his previous ratings* for the items in $N(i;x)$
- The predicted rating of user x for item i is:

$$r_{xi} = \frac{\sum_{j \in N(i;x)} sim(i,j) \cdot r_{xj}}{\sum_{j \in N(i;x)} sim(i,j)}$$

r_{xj} = rating of user x on item j

Example: Item-Item CF

users

items	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 (white cell) - rating between 1 to 5 (yellow cell)

Item-Item CF

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

users

items	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	

Item-Item CF (Set |N|=2)

Neighbor selection:
Identify movies similar to movie **1**, rated by user 5

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

users

items	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	
<u>3</u>	2	4		1	2		3		4	3	5		<u>0.41</u>
4		2	4		5			4			2		-0.10
5			4	3	4	2					2	5	-0.31
<u>6</u>	1		3		3			2			4		<u>0.59</u>

Item-Item CF (|N|=2)

Compute similarity weights:

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

users

items	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	-
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**

Item-Item CF (|N|=2)

$$r_{xi} = \frac{\sum_{j \in N(i;x)} sim(i,j) \cdot r_{xj}}{\sum_{j \in N(i;x)} sim(i,j)}$$

Predict by taking the weighted average:

$$r_{1\,5} = (0.41*2 + 0.59*3) / (0.41+0.59) = 2.6$$

users

items	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	
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

CF: Using Baseline Estimates

- Define similarity $sim(i,j)$ 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:

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

baseline estimate for (x,j)

baseline estimate for (x,i):

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

- $\boldsymbol{\mu}$ = overall mean item rating
- $\boldsymbol{b_x}$ = rating deviation of user $\boldsymbol{x}$
  = *(avg. rating of user* $\boldsymbol{x}$*)* $-\ \boldsymbol{\mu}$
- $\boldsymbol{b_i}$ = rating deviation of item $\boldsymbol{i}$

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 (e.g. new items, esoteric items)
- **- Popularity bias:**
  - Cannot recommend items to someone with unique taste
  - Tends to recommend popular items

Collaborative Filtering

Remarks & Practical Tips

Item-Item vs. User-User

	Alice	Bob	Carol	David
Star Wars	1		0.2	
Matrix		0.5		0.3
Avatar	0.2		1	
Pirates				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

Hybrid Methods

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

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

Evaluation

movies

users

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

Evaluation

movies

users

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

Test Data Set

Collaborative Filtering: Complexity

- Expensive step is **finding k most similar customers**: $O(|X|)$
  - X … set of customers
- Too expensive to do at runtime
- But we could pre-compute for all customers
  - Naïve pre-computation takes time $O(|X|^2)$
- We will learn how to do this faster!
  - Near-neighbor search in high dimensions via Locality-Sensitive Hashing
  - Other means: clustering, dimensionality reduction

Tip: Add Data

- **Leverage all the data**
  - Don’t try to reduce data size in an effort to make fancy algorithms work
  - Simple methods on large data do best
- **Add more data**
  - e.g., add IMDB data on genres
- **More data beats better algorithms**
  - http://anand.typepad.com/datawocky/2008/03/more-data-usual.html

Computing Pearson Efficiently

A Side-Note

Computing Pearson Efficiently

- „Our“ formula for Pearson skips missing values (NaN‘s) in the computation
  - => We use only items in $\boldsymbol{S_{xy}}$ (= items rated by both users $\boldsymbol{x}$ and $\boldsymbol{y}$) in each of the 3 sums
  - E.g. numerator: $\sum_{s \in S_{xy}} (\boldsymbol{r_{xs}} - \overline{\boldsymbol{r_x}})(\boldsymbol{r_{ys}} - \overline{\boldsymbol{r_y}})$
- Better: A. pre-process each vector as follows:
  - 1. Normalize non-missing values in each vector
    - I.e. compute mean over (only) non-missing values, and subtract m from each non-missing value
  - 2. Replace each missing value by 0
- ... and B. use standard Pearson formula (no skipping)

Computing Pearson - Example

- Example phase A:
  - 1. Normalize <u>non-missing</u> values in each vector
    - I.e. compute mean over (only) non-missing values, and subtract m from each non-missing value
  - 2. Replace each <u>missing</u> value by 0's
- xr = [1, nan, 3, nan, nan, 5, nan, nan, 5, nan, 4, nan]
  - 1a. Compute mean: m = (1+3+5+5+4)/5 = 3.6
  - 1b. Subtract mean m from each non-missing value:
    xr-m = [-2.6, nan, -.6, nan, nan, 1.4, nan, nan, 1.4, nan, .4, nan]
  - 2. Replace NaN's by 0's:

$\Rightarrow$ x = [-2.6, 0, -0.6, 0, 0, 1.4, 0, 0, 1.4, 0, 0.4, 0]

Computing Pearson - Example

- xr = [1, nan, 3, nan, nan, 5, nan, nan, 5, nan, 4, nan]

  $\Rightarrow$ x = [-2.6, 0, -0.6, 0, 0, 1.4, 0, 0, 1.4, 0, 0.4, 0]

  yr = [2, 4, nan, 1, 2, nan, 3, nan, 4, 3, 5, nan]

  $\Rightarrow$ y = [-1, 1, 0, -2, -1, 0, 0, 0, 1, 0, 2, 0]
- B. Compute „normal“ Pearson of x, y: 0.4140393356
- Why is the result same as using $\boldsymbol{S_{xy}}$ and “skipping”?
  - If a value (rating) is missing, it becomes 0 after pre-processing => same as “skipping” it in each sum
  - E.g. numerator: $\sum_{s \in S_{xy}} (\boldsymbol{r_{xs}} - \overline{\boldsymbol{r_x}})(\boldsymbol{r_{ys}} - \overline{\boldsymbol{r_y}})$
    - $s \notin S_{xy}$ => $(\boldsymbol{r_{xs}} - \overline{\boldsymbol{r_x}}) = \boldsymbol{0}$ or $(\boldsymbol{r_{ys}} - \overline{\boldsymbol{r_y}}) = \boldsymbol{0}$
    - => No contribution to the sum, i.e. as if skipped!

Computing Pearson - Code

```
from scipy import nan
import numpy as np
from scipy.stats import pearsonr

def normalize (input: list):
   mean = np.nanmean(input)
   return input - mean

def preprocess_vec (input: list):
   "Normalize and remove NaN's"
   return np.nan_to_num(normalize(input))
```

Computing Pearson - Code

```
# Rows (items) from slide "Item-Item CF (Set |N|=2)"
# x is item 1, y is item 3
xraw = [1, nan, 3, nan, nan, 5, nan, nan, 5, nan, 4, nan]
x = preprocess_vec(xraw)
yraw = [2, 4, nan, 1, 2, nan, 3, nan, 4, 3, 5, nan]
y = preprocess_vec(yraw)

# Correlation of x and y
corr_xy, p_value  = pearsonr(x, y)
print (f"Pearson correlation of x and y is {corr_xy}")
```

=> *Pearson correlation of x and y is 0.4140393356054126*

B. Content-based Recommender Systems

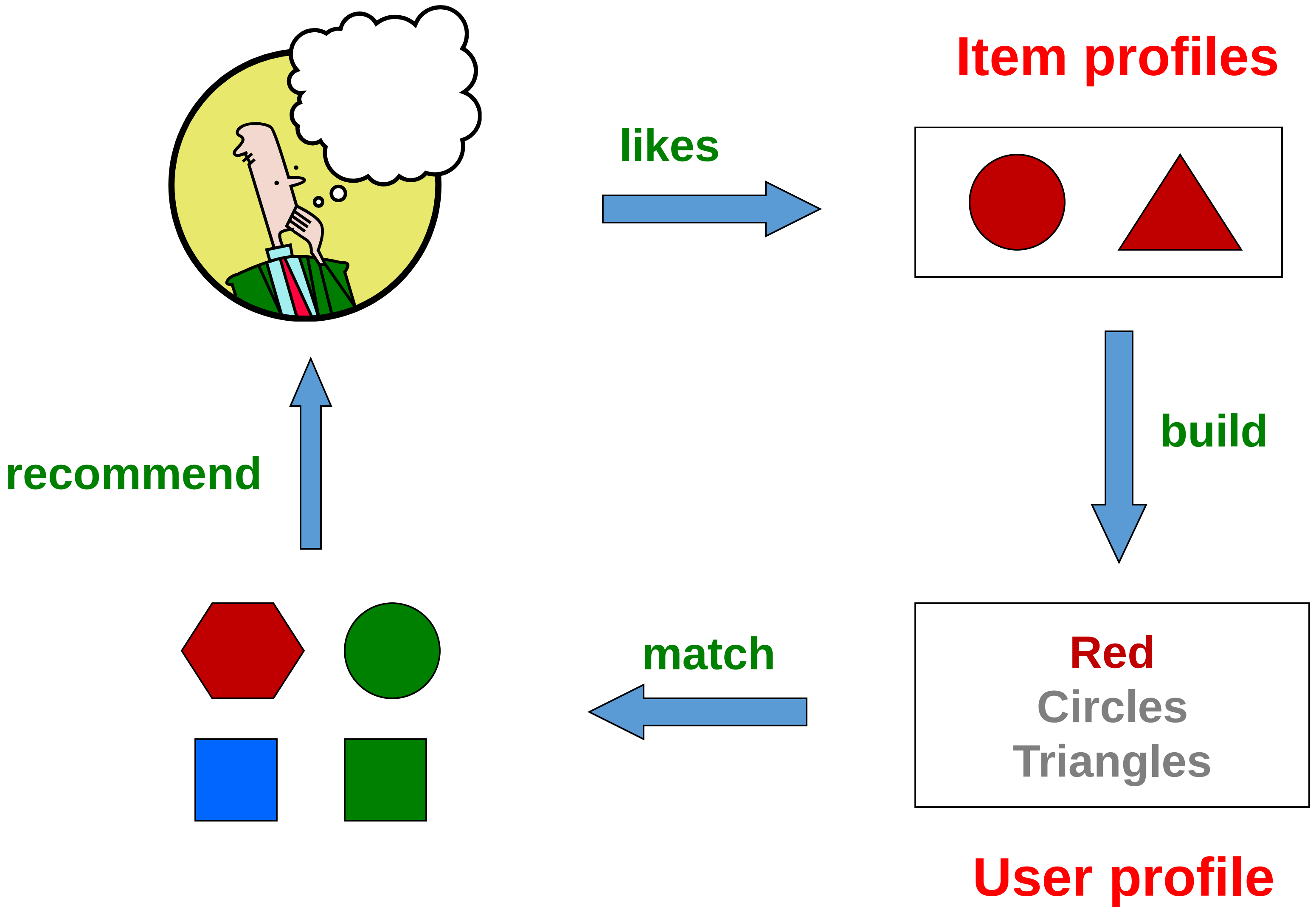
Content-based Recommendations

- **Main idea:** Recommend to customer ***x*** items similar to previous items rated highly by ***x***

Example:

- **Movie recommendations**
  - Recommend movies with same actor(s), genre, director,…
- **Websites, blogs, news**
  - Recommend other sites with “similar” content

Plan of Action



Item Profiles

- For each item, create an **item profile**
- Profile is a set (vector) v of features
  - **Movies:** author, title, actor, director,…
  - **Text:** Set of "important" words in document
- Binary encoding (0/1) is used frequently
  - Each (famous) actor gets a fixed vector position k
  - If present in the movie, v_k is set to 1, else to 0

Example: Items are Movies

- Representing item profile – a “mixed” vector

$$i_1 = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 1 \\ \vdots \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ \vdots \\ 3.2 \\ 2.9 \\ \vdots \end{bmatrix}$$

Set of actors
(as 0/1 entries)

Director(s)
(as 0/1 entries)

Ratings from various
movie DBs (as numbers)

User Profiles – A Simple Approach

- Idea: <u>average</u> the profiles of all items rated by a user
  - Weight each such item profile by the rating of this user
- Let use u give items 1, 2, …, n ratings $r_1, r_2, \ldots, r_n$
- **User profile x** = weighted average of rated item profiles

$$\mathbf{x} = (r_1 * i_1 + r_2 * i_2 + \ldots + r_n * i_n)/n$$

Weights = ratings

Profiles of items 1, 2, .., n (<u>those rated by user u</u>) - vectors

Example: Star-Based Ratings

$$i_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix},\ i_2 = \begin{bmatrix} 1 \\ 0 \end{bmatrix},\ i_3 = \begin{bmatrix} 0 \\ 1 \end{bmatrix},\ i_4 = \begin{bmatrix} 0 \\ 1 \end{bmatrix},\ i_5 = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \begin{array}{l} \leftarrow \text{actor A present} \\ \leftarrow \text{actor B present} \end{array}$$

- Items are movies, only features are "actors"
  - Item profile has 2 components (for actor A and actor B)
- User ratings are 1 to 5 stars (per movie)
- User watched 5 movies
  - Actor A – movies got 3 and 5 stars (movies 1 & 2)
  - Actor B – movies got 1, 2 and 4 stars (movies 3, 4, 5)
- Ratings are $r_1=3$, $r_2=5$, $r_3=1$, $r_4=2$, $r_5=4$
- Item profiles are as above

Example: Star-Based Ratings

- The user profile becomes:

$$(r_1 i_1 + r_2 i_2 + r_3 i_3 + r_4 i_4 + r_5 i_5)/5 =$$

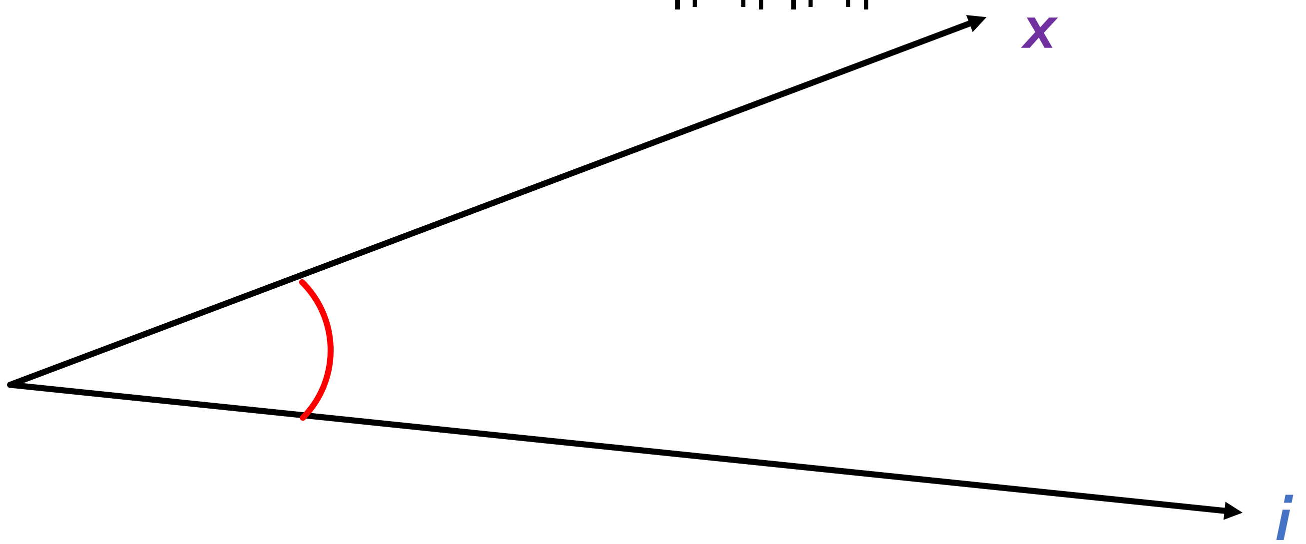
$$(3\begin{bmatrix}1\\0\end{bmatrix} + 5\begin{bmatrix}1\\0\end{bmatrix} + 1\begin{bmatrix}0\\1\end{bmatrix} + 2\begin{bmatrix}0\\1\end{bmatrix} + 4\begin{bmatrix}0\\1\end{bmatrix})/5 = \begin{bmatrix}8/5\\7/5\end{bmatrix}$$

Such simple user profiles can exhibit <u>low quality</u>
(esp. if a user rated only few items) =>
See additional slides for problems & solutions:
Slides from "Problems with Simple User Profiles"

Matching User and Item Profiles

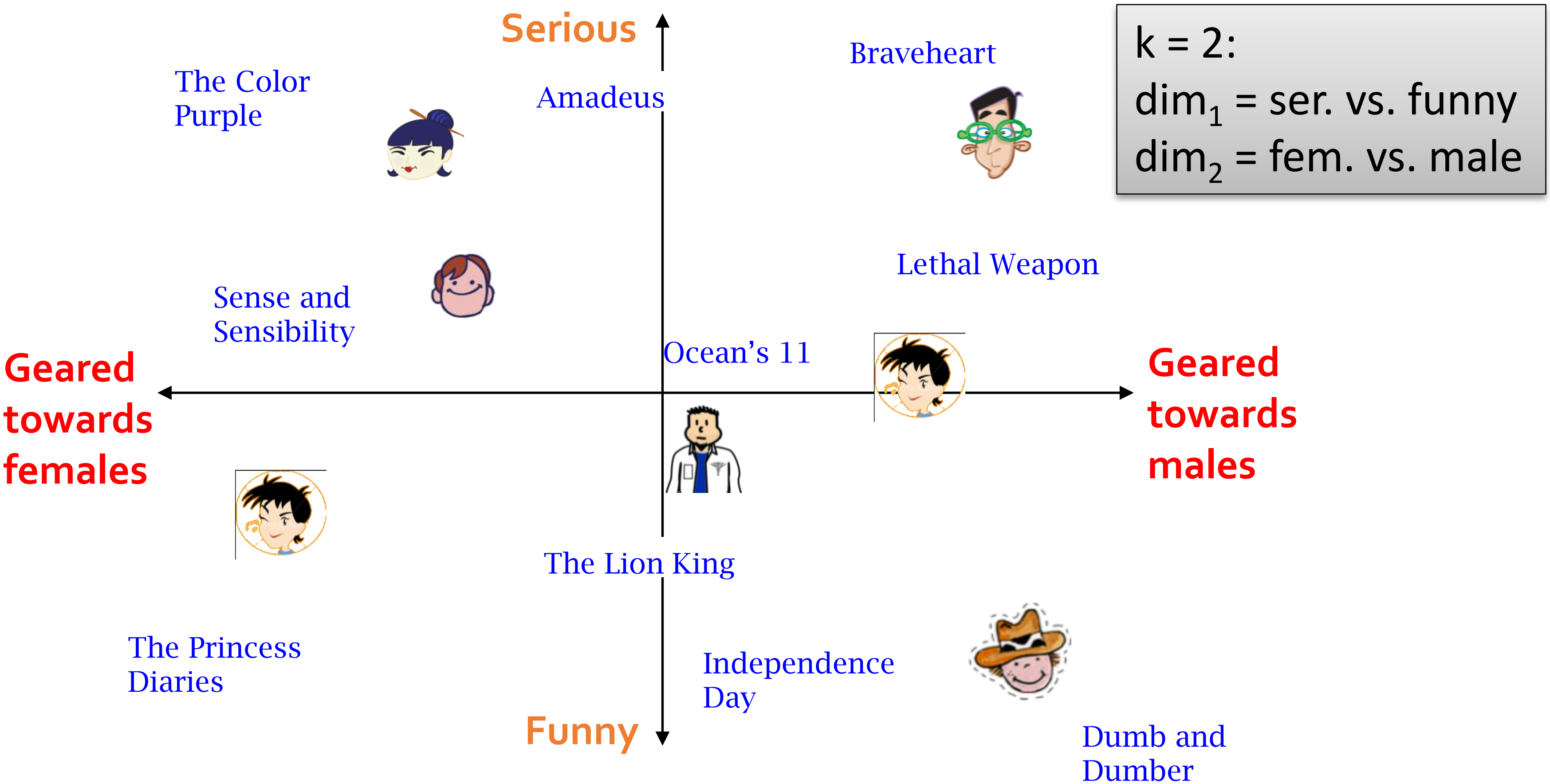
- To compute similarity of user profile and item profile, use a prediction heuristic:
  - Given user profile $\boldsymbol{x}$ and item profile $\boldsymbol{i}$, estimate

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



Content-Based R.: Interpretation

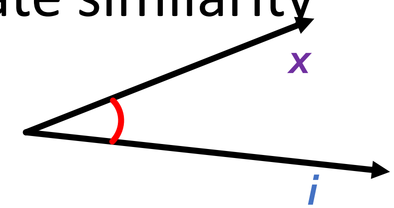
- In content-based recommendation, we represented each item and each user as a <u>vector in a k-dimensional space</u>
- => Item **i** close to a user **x** gets a high recommendation rating



Summary: Content-Based R.

- We construct a vector $\boldsymbol{i}$ **for each item** ("**item profile**") and a vector $\boldsymbol{x}$ (of size s) **for each user**
  - Item profile $\boldsymbol{i}$: "natural" attributes of an item
  - User vector $\boldsymbol{x}$: combination of item profiles rated by this user ("synthetic" profile)
- **Prediction heuristic**:
  - Given a user vector $\boldsymbol{x}$ and item vector $\boldsymbol{i}$, estimate similarity

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



  - For a user with vector $\boldsymbol{x}$, recommend by various criteria:
    - E.g. all items with $u(\boldsymbol{x}, \boldsymbol{i})$ > **threshold**
    - Rank items by $u(\boldsymbol{x}, \boldsymbol{i})$, recommend top k (e.g. k=5)

Pros: Content-based Approach

- **+: No need for data on other users**
  - No cold-start or sparsity problems
- **+: Able to recommend to users with unique tastes**
- **+: Able to recommend new & unpopular items**
  - No first-rater problem
- **+: Able to provide explanations**
  - Can provide explanations of recommended items by listing content-features that caused an item to be recommended

Cons: Content-based Approach

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

Thank you.

Questions?

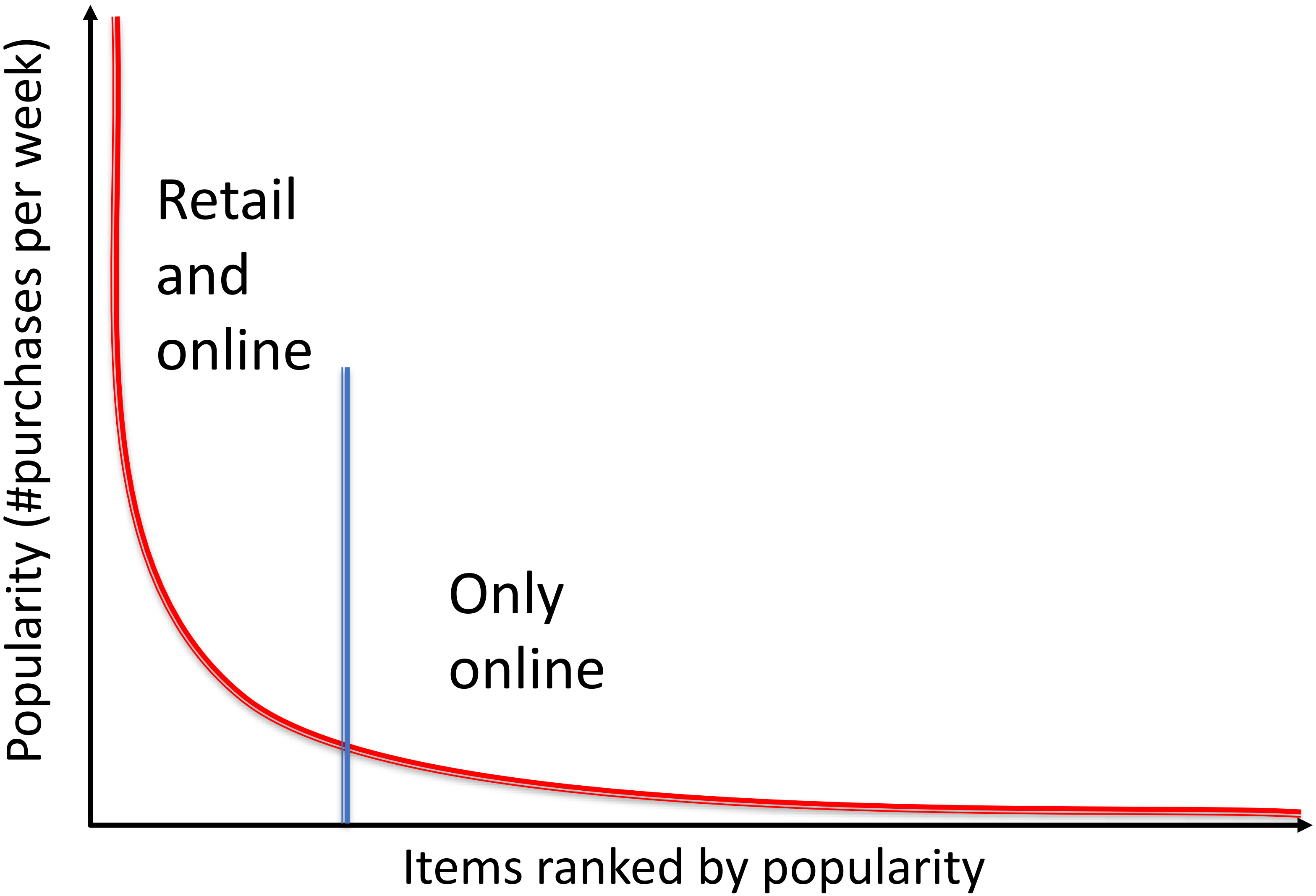
Additional Slides

Overview

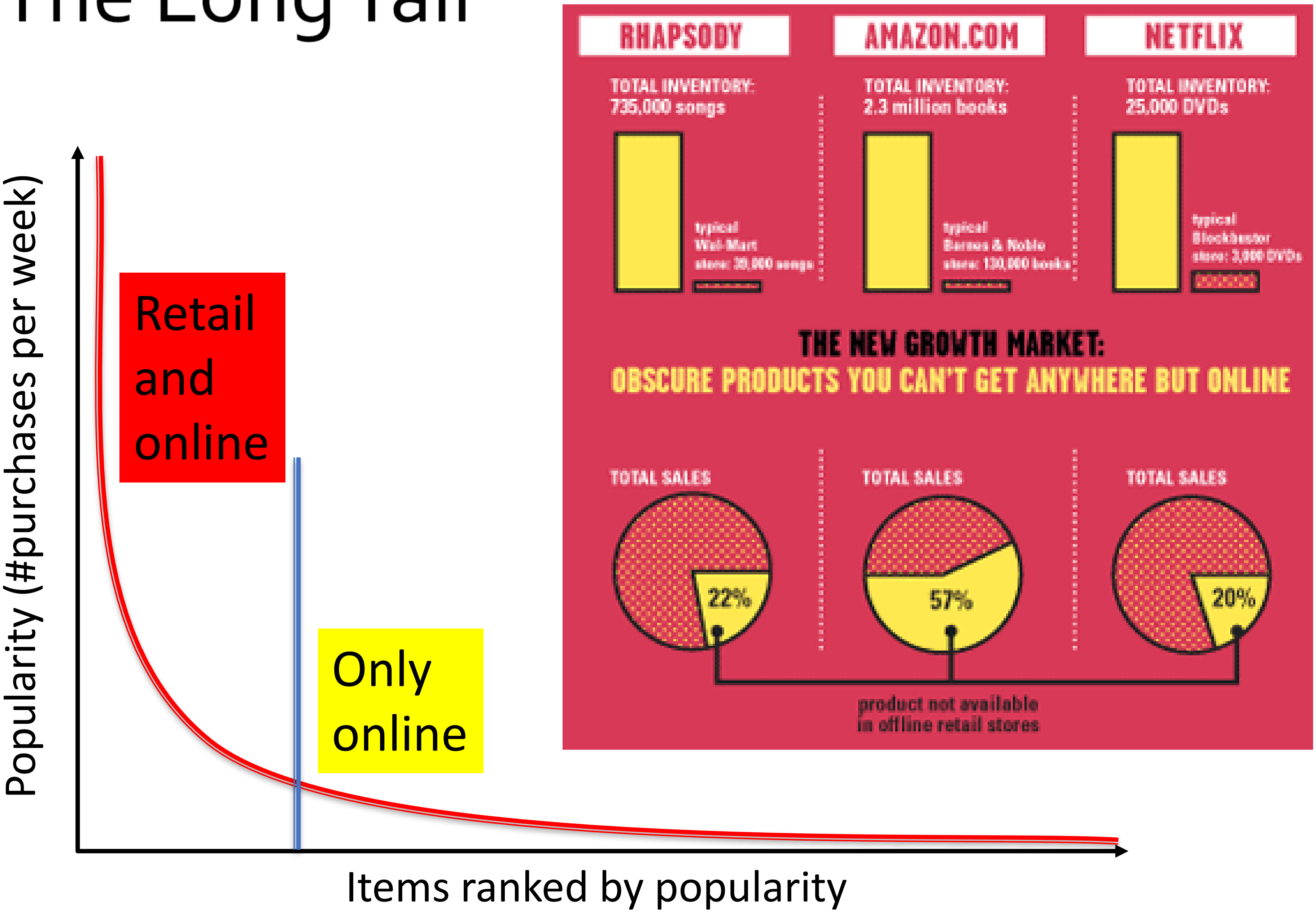
From Scarcity to Abundance

- 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
  - How **Into Thin Air** made **Touching the Void** a bestseller: http://www.wired.com/wired/archive/12.10/tail.html

The Long Tail



The Long Tail



(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?

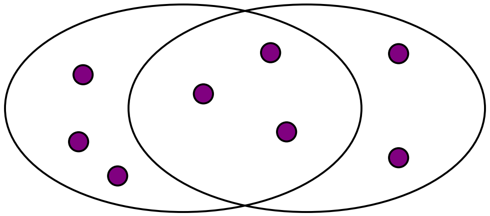
Additional Slides

Collaborative Filtering
Details on Similarity and Evaluation

Jaccard Measures

- The **Jaccard similarity** of two sets is the size of their intersection divided by the size of their union:
  $sim(C_1, C_2) = |C_1 \cap C_2| / |C_1 \cup C_2|$
- **Jaccard distance:** $d(C_1, C_2) = 1 - |C_1 \cap C_2| / |C_1 \cup C_2|$

- For measuring similarity of users, we consider only sets of items for which users voted
- Problem? Values of ratings are ignored!



3 in intersection
8 in union
Jaccard similarity = 3/8
Jaccard distance = 5/8

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

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

A Good 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: $sim(\boldsymbol{A},\boldsymbol{B}) > sim(\boldsymbol{A},\boldsymbol{C})$
  - **Jaccard similarity:** 1/5 **<** 2/4 => bad
  - **Cosine similarity:** 0.386 **>** 0.322 => not good
- **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

Notice: cos similarity is = **correlation** when data is centered at 0!

$sim\ \boldsymbol{A},\boldsymbol{B}$ vs. **A,C:**
0.092 > -0.559

=> $sim(\boldsymbol{x},\boldsymbol{y}) = \dfrac{\sum_i r_{xi} \cdot r_{yi}}{\sqrt{\sum_i r_{xi}^2} \cdot \sqrt{\sum_i r_{yi}^2}}$

Evaluating Predictions

How to compare predictions with known ratings?

- **Root-mean-square error** (RMSE), details: link
  - $\sqrt{\frac{1}{N}\sum_{xi}\left(r_{xi} - r_{xi}^{*}\right)^{2}}$
  - where $\boldsymbol{r}_{xi}$ is predicted, $\boldsymbol{r}^{*}_{xi}$ is the true rating of ***x*** on ***i,*** and ***N*** is the number of ratings (= number of used ***(x,i)*** combinations)

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

Additional Slides

Content-based
Recommender Systems

Problems with Simple User Profiles

- The user profile becomes:

$$(r_1 i_1 + r_2 i_2 + r_3 i_3 + r_4 i_4 + r_5 i_5)/5 =$$

$$(3\begin{bmatrix}1\\0\end{bmatrix} + 5\begin{bmatrix}1\\0\end{bmatrix} + 1\begin{bmatrix}0\\1\end{bmatrix} + 2\begin{bmatrix}0\\1\end{bmatrix} + 4\begin{bmatrix}0\\1\end{bmatrix})/5 = \begin{bmatrix}8/5\\7/5\end{bmatrix}$$

- Problem 1: user likes actor A more than actor B, but this shows only weakly in his profile!
- Problem 2: with more ratings, each component becomes smaller (as **n** gets larger)
  - Because components with value 0 disturb the average, but should be treated as “don’t care about corresp. rating”

For 1: Normalizing Ratings

- Solution for 1: Normalize ratings by subtracting user's mean rating (which is 3 = (3+5+1+2+4)/5)
  - Normalized ratings for actor A movies => 0, +2
  - Normalized ratings for actor A movies => -2, -1, +1
- Then the user profile is:
  - With r_1=0, r_2=+2, r_3=-2, r_4=-1, r_5=+1

$$(0\begin{bmatrix}1\\0\end{bmatrix}+2\begin{bmatrix}1\\0\end{bmatrix}-2\begin{bmatrix}0\\1\end{bmatrix}-1\begin{bmatrix}0\\1\end{bmatrix}+1\begin{bmatrix}0\\1\end{bmatrix})/5=\begin{bmatrix}2/5\\-2/5\end{bmatrix}$$

Now better: clear distinction for actor A and actor B

For 2: Per-component Weights

Only ratings for these 2 items should be counted for actor A

$$(0\begin{bmatrix}1\\0\end{bmatrix}+2\begin{bmatrix}1\\0\end{bmatrix}-2\begin{bmatrix}0\\1\end{bmatrix}-1\begin{bmatrix}0\\1\end{bmatrix}+1\begin{bmatrix}0\\1\end{bmatrix})/5=\begin{bmatrix}2/5\\-2/5\end{bmatrix}$$

Only the ratings for these 3 items should be counted for actor B

- Essence of problem 2: a 0 in an item's component (=attribute) k should mean "don't care", but now mean "one more neutral rating for attribute k"
- => Use "individual" n for each vector component
  - For actor A: $n_A = 2$, for actor B: $n_B = 3$

For 2: Per-component Weights

- => Use "individual" n for each vector component
  - For actor A: $n_A = 2$, for actor B: $n_B = 3$
  - Recall: Normalized ratings are $r_1=0$, $r_2=+2$, $r_3=-2$, $r_4=-1$, $r_5=+1$
- Then the user profile becomes:

$$\begin{bmatrix} r_1/n_a \\ 0 \end{bmatrix} + \begin{bmatrix} r_2/n_a \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ r_3/n_b \end{bmatrix} + \begin{bmatrix} 0 \\ r_4/n_b \end{bmatrix} + \begin{bmatrix} 0 \\ r_5/n_b \end{bmatrix} =$$

$$\begin{bmatrix} 0/2 \\ 0 \end{bmatrix} + \begin{bmatrix} 2/2 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ -2/3 \end{bmatrix} + \begin{bmatrix} 0 \\ -1/3 \end{bmatrix} + \begin{bmatrix} 0 \\ 1/3 \end{bmatrix} = \begin{bmatrix} 1 \\ -2/3 \end{bmatrix}$$

Sidenote: Text Features

- How to pick important text features?
- Usual heuristic from text mining is **TF-IDF** (Term-frequency * Inverse-Doc-Frequency)
  - Term == Feature
  - Doc(ument) == Item
- Now popular: word embeddings like **word2vec**
  - Each word is represented as a (sub)vector
  - Such vectors represent typical contexts (other words) in which this one occur
  - This distributed representation is learned from data

Sidenote: TF-IDF

- For term = i and doc = j, the **TF-IDF score** $\boldsymbol{w_{ij}}$ is:

$$\boldsymbol{w_{ij} = TF_{ij} * IDF_i}$$

TF (term frequency):

- Let $\boldsymbol{f_{ij}}$ be frequency of term $\boldsymbol{i}$ in document $\boldsymbol{j}$
- Then:

$$TF_{ij} = \frac{f_{ij}}{\max_k f_{kj}}$$

Note: we normalize TF to discount for "longer" docs

IDF (inverse document frequency (of a term)):

- Let $\boldsymbol{N}$ = total number of docs, $\boldsymbol{n_i}$ = number of docs that mention term $\boldsymbol{i}$
- Then:

$$IDF_i = \log \frac{N}{n_i}$$