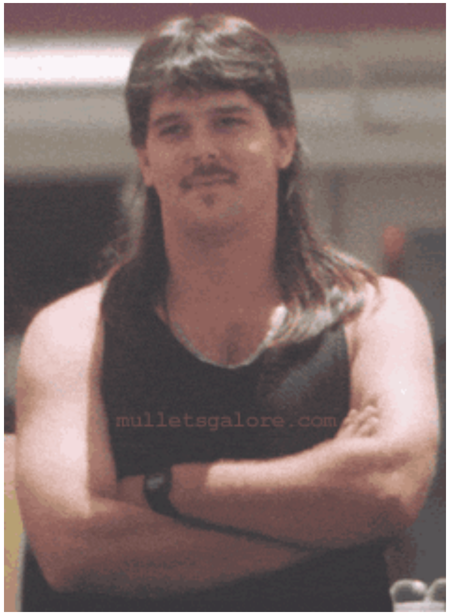
Recommender Systems

Aline Bessa
PhD Candidate, NYU

Modified from Jeff Ullman, Anand Rajaraman, and Jure Lescovek

http://www.mmds.org

I didn't know I liked that!



- **Michael**
  - Buys Metallica album
  - Buys Megadeth album
- **John**
  - Bought a Metallica album
  - Recommender system suggests Megadeth from data collected about **Michael**

pandora®

okcupid

Rotten Tomatoes®

Spotify®

twitter

allrecipes.com®

amazon

LinkedIn

IMDb

facebook

NETFLIX

The New York Times

yelp

tripadvisor®

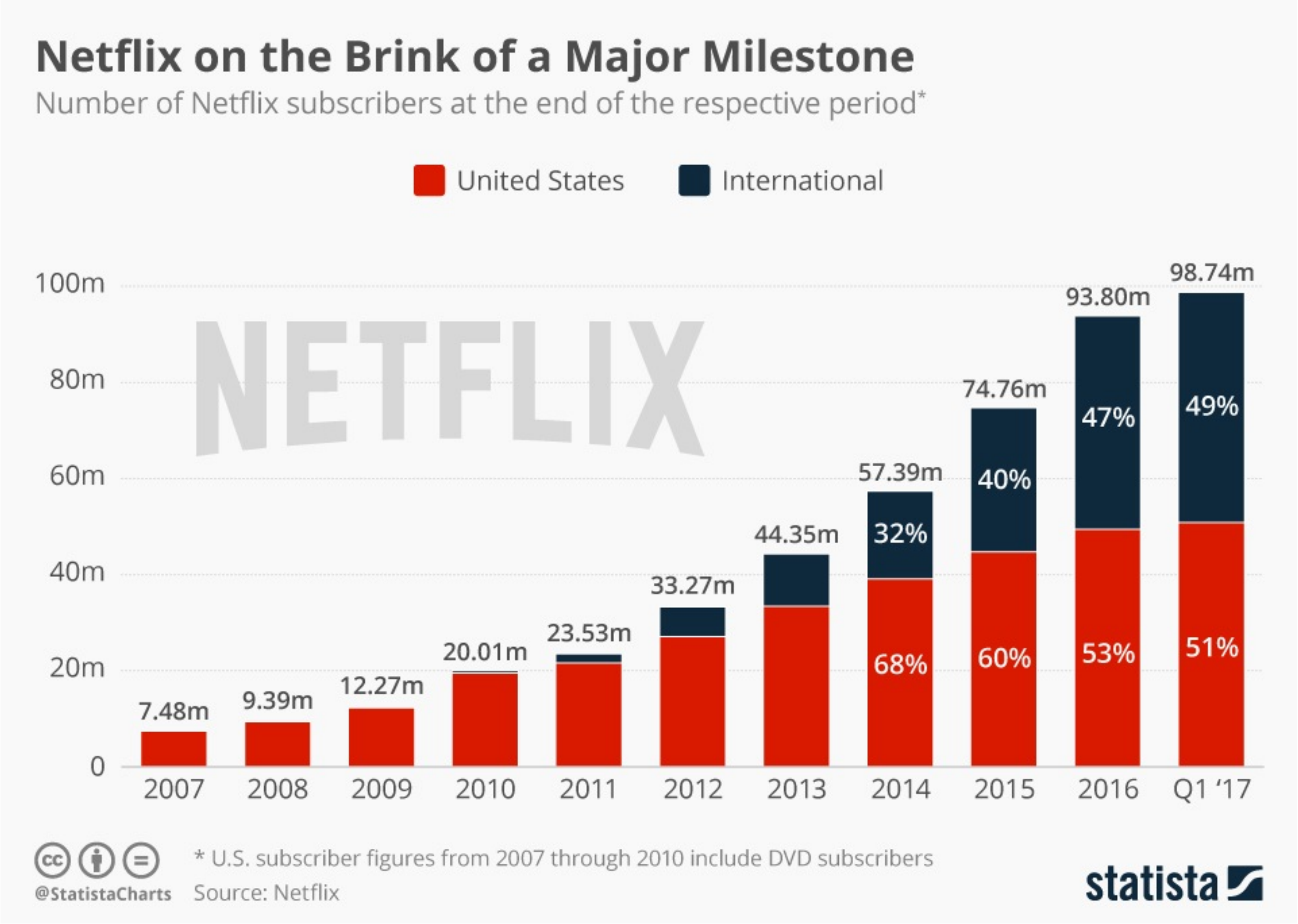
hulu

Instagram

YouTube

Relationship with Big Data

High number of users



Source: https://www.statista.com/chart/7677/netflix-subscriber-growth/

Relationship with Big Data

High number of items

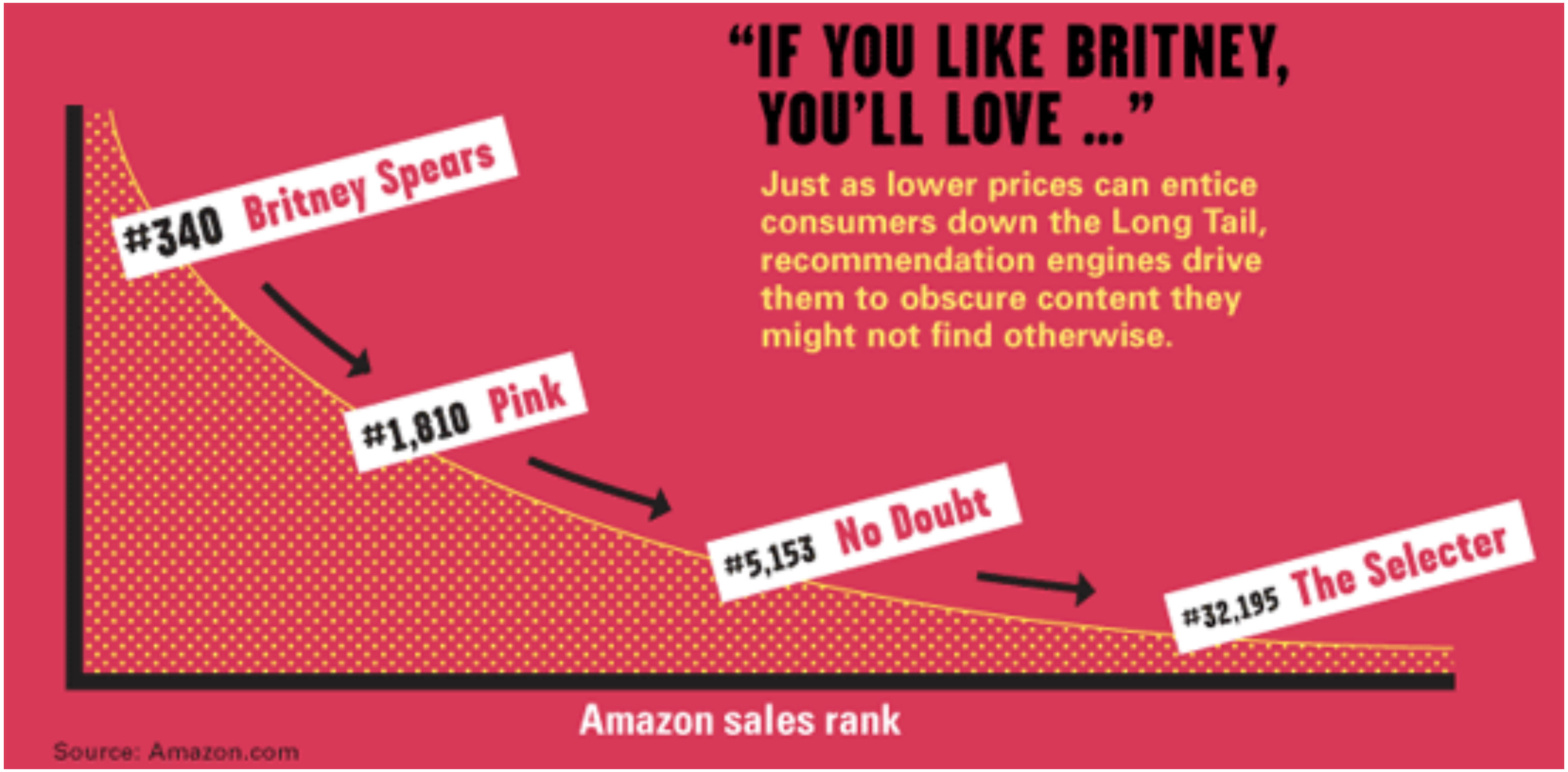
How many products does Amazon sell (2015)?

Amazon.com	USA	488 million	+235 million
Amazon.co.uk	UK	261 million	+108 million
Amazon.de	Germany	237 million	+96 million
Amazon.fr	France	209 million	+90 million
Amazon.co.jp	Japan	168 million	+60 milion
Amazon.it	Italy	165 million	+77 million
Amazon.es	Spain	160 million	+76 million
Amazon.ca	Canada	133 million	+77 million
Amazon.in	India	42 million	+18 million

Source: https://export-x.com/2015/12/11/how-many-products-does-amazon-sell-2015/

Relationship with Big Data

Exploration of a long tail of choices: finer tuned recommendations



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

More data leads to better recommendations!

- Tastes are better characterized
- Level of personalization in recommendations increases
- More data beats better algorithms
  - http://anand.typepad.com/datawocky/2008/03/more-data-usual.html
- Adding more data is usually easy
  - e.g., add IMDB data on movie genres
- Cost of keeping additional data for items is cheap

Formal Model

- ***X*** = set of **users**
- ***S*** = set of **items**
- **Utility function** ***u***: ***X*** × ***S*** → ***R***
    - ***R*** is set of ratings
    - ***R*** is a totally ordered set
    - e.g., **0-5** stars, real number in **[0,1]**

Utility Matrix

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

Utility Matrix - Goal

	Avatar	LOTR	Matrix	Pirates
Alice	1	*u(Alice, LOTR) = ?*	0.2	
Bob		0.5		0.3
Carol	0.2		1	
David				0.4

Calculating $u(x, i)$: key problems

- Gathering "known" ratings for matrix
  - How to collect the data in the utility matrix?
    - Explicit *versus* implicit rating
- Predicting ratings $u(x, i)$ 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*
- Evaluating different $u(x, i)$ functions
  - How to measure success/performance of recommendation methods

Approaches to recommender systems

- Content-based
  - Recommend items to user ***x*** similar to previous items rated highly by ***x***
- Collaborative filtering
  - Find set ***N*** of users whose ratings are similar to ***x***'s
  - Predict new ratings for ***x*** based on ratings of users in ***N***
  - Recommend items to ***x*** associated to highest predicted ratings
- Hybrid methods
  - Recommend items combining similarities with previous items rated by x and predicted ratings based on ratings of ***x***'s similar users

Approaches to recommender systems

- Content-based
  - Recommend items to user ***x*** similar to previous items rated highly by ***x***
- Collaborative filtering **TODAY!**
  - Find set ***N*** of users whose ratings are similar to ***x***'s
  - Predict new ratings for ***x*** based on ratings of users in ***N***
  - Recommend items to ***x*** associated to highest predicted ratings
- Hybrid methods
  - Recommend items combining similarities with previous items rated by x and predicted ratings based on ratings of ***x***'s similar users

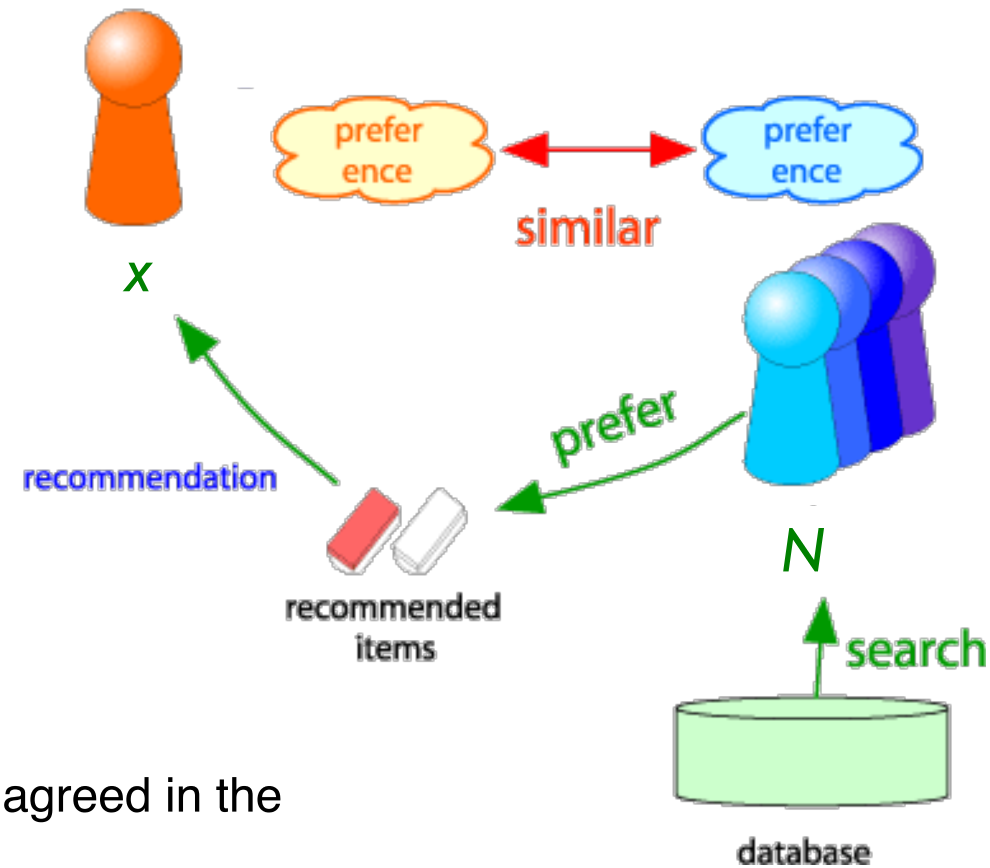
Collaborative filtering - why?

- Works for any kind of item in an agnostic fashion
  - No need to select features for similarity computations with previously rated items
  - Useful for “harder” data types (e.g., audio)
- Exploits quality judgments of other users
  - An item similar to previously rated ones can still be bad
- Significant technical advancements in the last decade
  - Netflix prize - https://en.wikipedia.org/wiki/Netflix_Prize
  - Recsys conference - https://recsys.acm.org/best-papers/
- An ***X* x *S*** rating matrix is the sole requirement
  - No additional data is used by CF algorithms

Collaborative Filtering

Consider user ***x***

1. Find set ***N*** of other users whose ratings are "**similar**" to ***x***'s ratings
2. Predict ***x***'s ratings based on ratings of users in ***N***



Underlying assumption I: People who agreed in the past will agree in the future

Underlying assumption II: The items people will like in the future are similar to the ones they liked in the past

1) Finding Similar Users

Input: Set of users *X*, value *k* for *k* most similar users

Output: Sets *N* of *k* most similar users for all *x* in *X*

```
For user x in X:
  N[x] = {}
For user x in X:
  For user y in X \ {x}:
      tmp = compute_similarity(x, y)
      if tmp is one of the k top similarities computed so far for x:
        store y in N[x]
return N
```

1) Finding Similar Users

Input: Set of users X, value k for k most similar users

Output: Sets N of k most similar users for all x in X

```
For user x in X:
  N[x] = {}
For user x in X:
  For user y in X \ {x}:
    tmp = compute_similarity(x, y)
    if tmp is one of the k top similarities computed so far for x:
      store y in N[x]
return N
```

1) Finding Similar Users

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

- **Which users are more similar to *A*?**
  - Users *A* and *C*
    - Rated **two movies** in common
    - **Opposite opinions** about them
  - Users *A* and *B*
    - Rated **one movie** in common
    - **Similar opinions** about it

Intuitively we want sim(A, B) > sim(A, C)

1) Finding Similar Users

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

- **Jaccard similarity**
  - **Problem:** Ignores values of ratings
  - A = {HP1, TW, SW1}, B = {HP1, HP2, HP3}, C = {TW, SW1, SW2}
  - Jaccard(A, B) = 1/5, Jaccard(A, C) = 1/2
  - Jaccard(A, C) > Jaccard(A, B) **Intuitively wrong!**

Note:

Jaccard(X, Y) = $|X \cap Y| / |X \cup Y|$

1) Finding Similar Users

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

- **Cosine similarity**
  - **Problem:** Treats missing rating as zero (low rating)
  - A = [4, 0, 0, 5, 1, 0, 0], B = [5, 5, 4, 0, 0, 0, 0],
    C = [0, 0, 0, 2, 4, 5, 0]
  - Cosine(A, B) = 0.380, Cosine(A, C) = 0.322
  - Cosine(A, B) > Cosine(A, C)
  - **Intuitively correct but values are close**

Note: Cosine(X, Y) = (X . Y)/(IIXII . IIYII)

1) Finding Similar Users

- Jaccard works better when the matrix is binary
- Cosine works better when ratings are normalized
- Many other options
  - Pearson correlation
  - Kullback–Leibler divergence
  - Metric learning https://en.wikipedia.org/wiki/Similarity_learning

2) Predicting Ratings

- Let r_w be the vector of user *w*'s ratings
- Let *N'* be the subset of *N* where all *n* in *N'* rated item *i*
- The predicted rating r_{xi} (shorthand for *u(x, i)*) is

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

$$r_{xi} = \frac{\sum_{y \in N'} s_{xy} \cdot r_{yi}}{\sum_{y \in N'} s_{xy}}$$

Shorthand:
$s_{xy} = sim(x, y)$

What if we focus on the items instead?

- **So far:** User-user collaborative filtering
  - Similarity computed between users
- **Another view:** Item-item collaborative filtering
  - For item ***i***, find set ***N*** of similar items
  - Predict rating for item ***i*** based on ratings for items in ***N***
  - Can use same similarity metrics and prediction functions as in user-user model

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

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

Item-Item CF (|N|=2)

users

movies

12	11	10	9	8	7	6	5	4	3	2	1	
	4		5			5			3		1	1
3	1	2			4			4	5			2
	5	3	4		3		2	1		4	2	3
	2			4			5		4	2		4
5	2					2	4	3	4			5
	4			2			3		3		1	6

- unknown rating

- rating between 1 to 5

Item-Item CF (|N|=2)

users

movies

12	11	10	9	8	7	6	**5**	4	3	2	1	
	4		5			5	?		3		1	1
3	1	2			4			4	5			2
	5	3	4		3		2	1		4	2	3
	2			4			5		4	2		4
5	2					2	4	3	4			5
	4			2			3		3		1	6

- predict rating of movie **1** by user **5**

Item-Item CF (|N|=2)

users

12	11	10	9	8	7	6	**5**	4	3	2	1	movies	sim(1,m)
	4		5			5	?		3		1	1	1.00
3	1	2			4			4	5			2	-0.18
	5	3	4		3		2	1		4	2	**3**	**0.41**
	2			4			5		4	2		4	-0.10
5	2					2	4	3	4			5	-0.31
	4			2			3		3		1	**6**	**0.59**

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

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 CF (|N|=2)

users

12	11	10	9	8	7	6	5	4	3	2	1	movies	sim(1,m)
	4		5			5	?		3		1	1	1.00
3	1	2			4			4	5			2	-0.18
	5	3	4		3		2	1		4	2	<u>3</u>	<u>0.41</u>
	2			4			5		4	2		4	-0.10
5	2					2	4	3	4			5	-0.31
	4			2			3		3		1	<u>6</u>	<u>0.59</u>

Compute similarity weights:

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

Item-Item CF (|N|=2)

users

12	11	10	9	8	7	6	5	4	3	2	1	movies
	4		5			5	**2.6**		3		1	1
3	1	2			4			4	5			2
	5	3	4		3		2	1		4	2	3
	2			4			5		4	2		4
5	2					2	4	3	4			5
	4			2			3		3		1	6

Predict by taking weighted average:

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

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

Common Practice

- So far:

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

- Common practice:

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

baseline estimate for r_{xi}

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

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

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

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

$\boldsymbol{b_i}$ = rating deviation of movie $\boldsymbol{i}$

= (*avg. rating for item* $\boldsymbol{i}$) – $\boldsymbol{\mu}$

Item-Item *versus* User-User

	Avatar	LOTR	Matrix	Pirates
Alice	1		0.8	
Bob		0.5		0.3
Carol	0.9		1	0.8
David			1	0.4

- In practice, it has been observed that **<u>item-item</u>** often works better than user-user
- **Why?** Items are simpler, users have multiple tastes

Limitations of Collaborative Filtering

- Cold Start
  - New items have no ratings and new users have no history
    - CR needs enough users in the system to build ***N***
- 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
    - Detrimental to new items or *niche* items
- Popularity bias
  - Cannot recommend items to someone with unique taste
  - Tends to recommend popular items

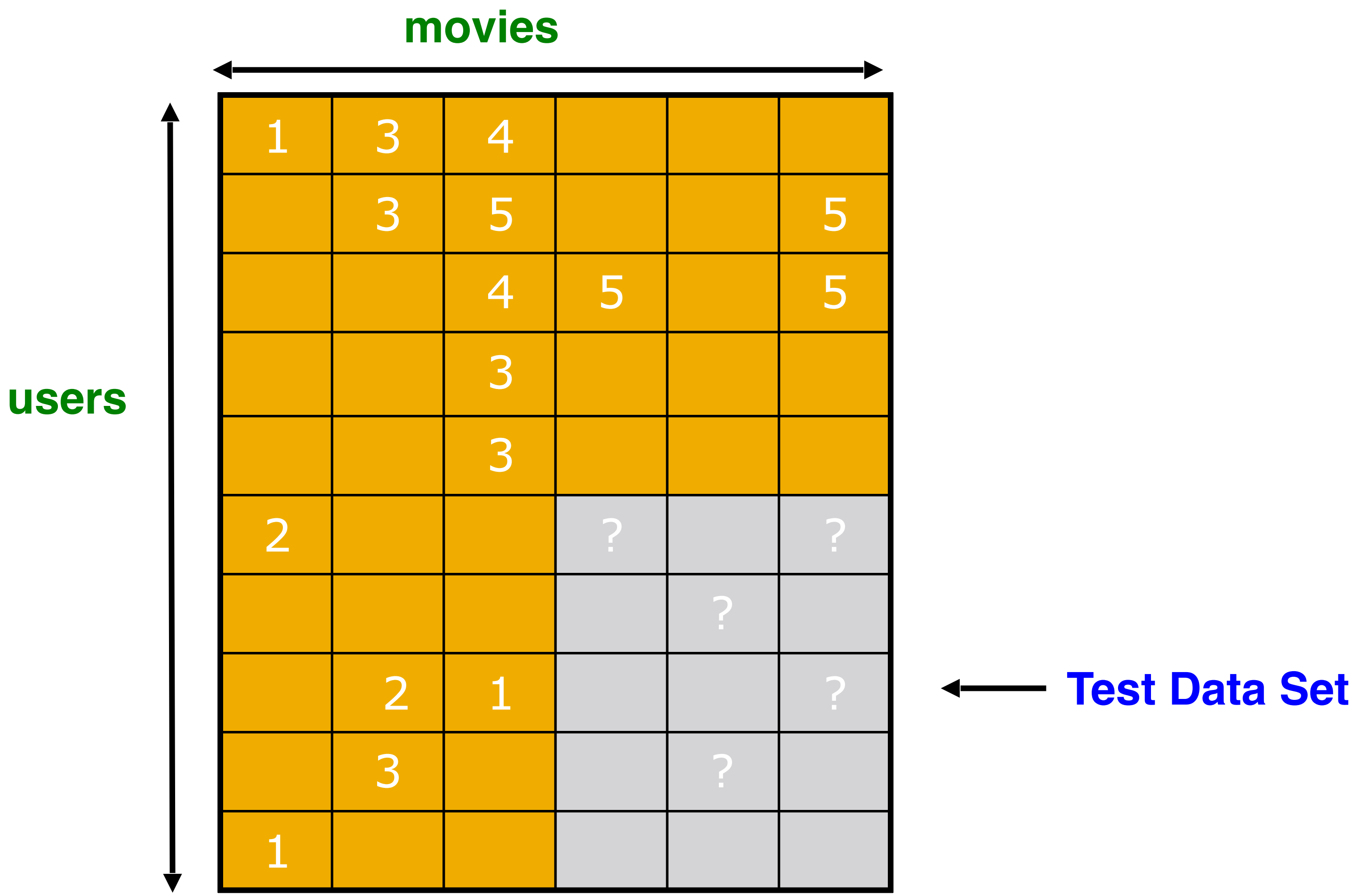
Evaluation

movies

users

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

Evaluation



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

Evaluating Predictions

- Root mean square error (**RMSE**)
  - $\sqrt{\sum_{xi}(r_{xi} - r^*_{xi})^2}$ where r_{xi} is the estimated rating and r^*_{xi} is the true one
- Precision at top *k* (**P@*k***)
  - A = {relevant items in test set for user *x*}
    - e.g., items with ratings 4 and 5
  - B = {*k* top estimated items in test set for user *x*}
  - P@*k* = |A ∩ B|/|B|

Evaluating Predictions

- Root mean square error (**RMSE**)
  - $\sqrt{\sum_{xi}(r_{xi} - r^*_{xi})^2}$ where r_{xi} is the estimated rating and r^*_{xi} is the true one
- Precision at top *k* (**P@*k***)
  - A = {relevant items in test set for user *x*}
    - e.g., items with ratings 4 and 5
  - B = {*k* top estimated items in test set for user *x*}
  - P@*k* = |A ∩ B|/|B|

Focus on accuracy of recommendations!

Beyond Accuracy

- In the past ~10 years, there has been a growing concern about other aspects of good recommendations
  - Coverage - Number of items/users for which the system can make predictions
  - Diversity - It is more likely to find a suitable item if the recommended items are diverse, unless the user expressed narrow set of preferences
  - Novelty - Recommendations for items that the user did not know about are relevant
  - Scalability - Large collections of items in real-life scenarios require algorithmic tradeoffs (e.g., efficiency and accuracy)
  - Co-utility - The probability that an item has of being useful is affected by other recommended items
  - Adaptivity - Item collections and user interests change over time

Complexity of CF

- Finding the set ***N*** of most similar users/items is expensive
  - O(|X|) per user or O(|S|) per item

Too expensive to do at runtime!

- Pre-computation of ***N*** is common practice
  - Naïve pre-computation
  - Near-neighbor search in high dimensions (LSH)
  - Parallelization with MapReduce

Naïve pre-computation (item-item)

Input: Set of items *I*, value *k* for *k* most similar items

Output: Sets *N* of *k* most similar items for all *i* in *I*

```
For item i in I:
  N[i] = {}
For item i in I:
  For item j in I \ {i}:
      tmp = compute_similarity(i, j)
      if tmp is one of the k top similarities computed so far for i:
        store j in N[i]
return N
```

Near-neighbor search (LSH)

Input: Set of items I, value k for k most similar items

Output: Sets N of k most similar items for all i in I

Note I: we abstracted the parameters for the family of hash functions in the LSH here

H = Hashtable for items
For item i in I:
  compute_LSH(H, i)

Note II: Items a and b are in the same bucket in H, with a probability >= p, if dist(a, b) <= r. dist, p and r are parameters

For item i in I:
  candidates = {all items in i's bucket in H} \ {i}
  N[i] = {top k most similar items in candidates}
return N

Parallelization with MapReduce

Mapper input: Item i and a set of items J

Note: Items are either rows or columns in a utility matrix

Map function: Compute sim(j, i) for all j in J; keep local set of top similarities $K = \{(j_1, \text{sim}(j_1, i)), \ldots, (j_k, \text{sim}(j_k, i))\}$; output tuple ($i$, K)

Reduce function: For each key i, merge the sets of its local top similarities and generate a global set K_{global}; output (i, K_{global})

Ethics and recommender systems

- Acquisition of user data has to be transparent
- Privacy of users
  - https://www.wired.com/2009/12/netflix-privacy-lawsuit/
- Transparency in filtering choices
- Explainability of recommendations

More!

- Ricci, F., Rokach, L., Shapira, B., & Kantor, P. B. (Eds). (2011). **Recommender systems handbook.** New York: Springer.
- Mahmood, T., Ricci, F. **Learning and adaptivity in interactive recommender systems.** In: Proceedings of the 9th International Conference on Electronic Commerce, ICEC'07, pp. 75–84. ACM Press, New York, NY, USA (2007)
- Collaborative filtering and hadoop: http://aimotion.blogspot.com/2012/08/introduction-to-recommendations-with.html
- Winning the Netflix Prize: http://blog.echen.me/2011/10/24/winning-the-netflix-prize-a-summary/