

Recommender Systems: Part 1

Web Science Lecture

15 March 2020

Sagnik Ray Choudhury

src@di.ku.dk

<http://sagnik.in>

UNIVERSITY OF COPENHAGEN



Organization

- Part 1: Problem definition, motivations.
- Part2: Intro to content-based methods.
- Part3: Collaborative filtering.
- Part 4: Latent factor models: introduction.
- Acknowledgements:
 - Jure Leskovec, Anand Rajaraman, Jeff Ullman([Mining of Massive Datasets](#))
 - Isabelle Augenstein ([Web Science 2020](#))
 - Charu C Agarwal ([Recommender Systems, The Textbook](#))
 - Internet.

Announcement

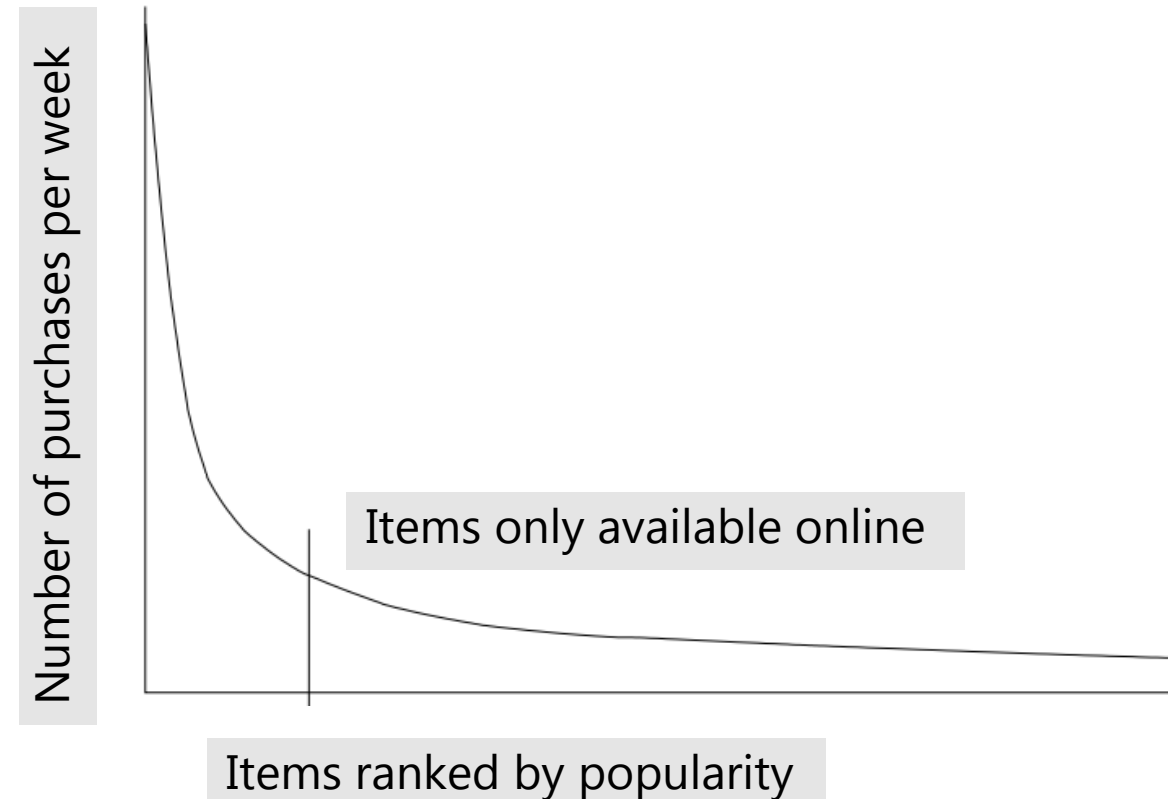
- On **March 15** you will receive a link for the course evaluation in your KU e-mailbox.
- The deadline to complete the course evaluation is **March 28**.
- Please remember to fill out the course evaluation, even if you do not have anything wrong to report!
- Your feedback is very important to us, so we can understand how to improve this course for the next years.

Recommender Systems: Use

- Movies: Netflix
- Friends or connections: Facebook, LinkedIn
- Products: Amazon
- Are recommender systems good?
 - Recommender systems outperform human recommenders, even in a domain where people have a lot of experience and well-developed tastes: predicting what people will find funny.
- Do people *trust* recommender systems?
 - People do not trust these recommender systems. They do not use them to make recommendations for others, and they prefer to receive recommendations from other people instead.
 - Why? Machine recommendations are hard to understand (we will see why).
 - Simple explanations can help! (Netflix: "because you watched", Amazon: "because you bought")

From Scarcity to Abundance

- Shelf space is a scarce commodity for traditional retailers
 - Also, consider TV networks before Netflix
- The Web enables near-zero-cost dissemination of information about products.
 - Scarcity to abundance (number of products in Amazon is much higher than any physical space)
 - “Long tail” phenomenon



Into Thin Air and Touching the Void

An extreme example of how the long tail, together with a well designed recommendation system can influence events is the story told by Chris Anderson about a book called *Touching the Void*. This mountain-climbing book was not a big seller in its day, but many years after it was published, another book on the same topic, called *Into Thin Air* was published. Amazon's recommendation system noticed a few people who bought both books, and started recommending *Touching the Void* to people who bought, or were considering, *Into Thin Air*. Had there been no on-line bookseller, *Touching the Void* might never have been seen by potential buyers, but in the on-line world, *Touching the Void* eventually became very popular in its own right, in fact, more so than *Into Thin Air*.

Formal Model

- C = set of customers/users
- S = set of items (movies, products)
- Utility function: $u: C \times S \rightarrow R$
 - R = set of ratings (real values within a range, eg, 0-5)
 - Totally ordered set

	Fargo	Splash	Titanic	Juno	Gladiator
John	4			5	1
Mary	5		4		
Lars			2	4	5
Mette		3		3	

We want to fill up the missing values in the matrix. We care less about the correctness of “low” ratings than high ratings.

Key Problems

- Gathering “known” ratings for the matrix
- Extrapolate unknown ratings from the known ones
 - We really care about high ratings
- Evaluate extrapolation methods

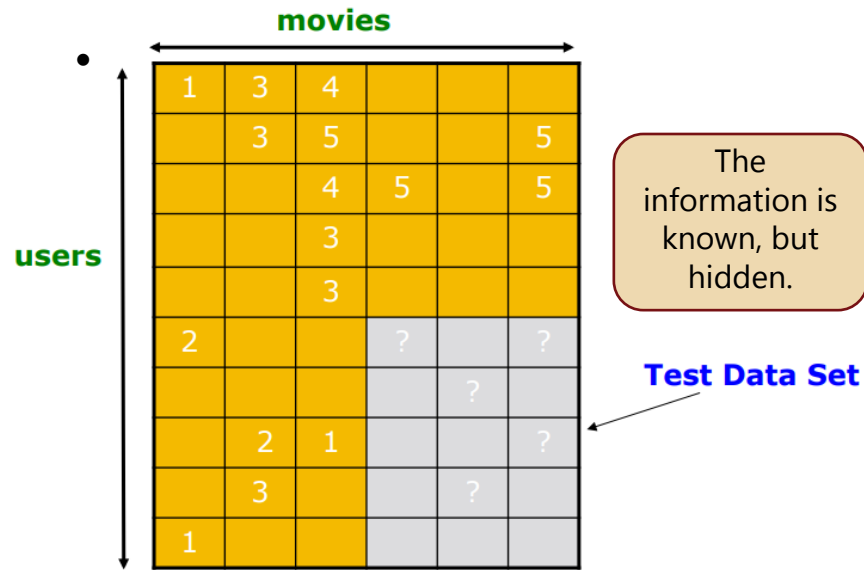
Gathering Known Ratings

- Explicit
 - Ask people to rate items
 - Doesn't work well in practice, people can not be bothered.
- Implicit
 - Learn rating from user actions: purchase/watching/having conversation with somebody implies high rating. (very scalable)
 - What about low ratings?
- Most recommender systems use a combination of both.

Extrapolate Unknown Ratings from the Known Ones

- Key Problem: Utility matrix U is sparse.
 - Most people have not rated most items.
 - Cold start:
 - New items have no ratings.
 - New users have no history.
 - It's just not enough that one user provides ratings for multiple items (or one item has multiple user ratings), multiple users must rate multiple items.
- Three approaches to recommender systems
 - Content based.
 - Collaborative filtering.
 - Latent factor based.
- But first, evaluation!

Evaluating Recommender Systems



Compare predictions against withheld known ratings (test set T)

Root Mean Square Error or RMSE:

$$\sqrt{\frac{\sum_{(x,i) \in T} (r_{xi} - r_{xi}^*)^2}{N}}$$

$N = |T|$

r_{xi} is the predicted rating

r_{xi}^* is the actual rating

But this narrow focus on accuracy will sometimes get us into problems.

Problems with RMSE

- **Prediction diversity:** If you have only seen Harry Potter movies, you will probably like LOTR as well, but this method will penalize you)
- **Prediction context:** Suppose you went on a trip to the beach and bought accordingly. Your recommendations should not *a/ways* be based on that.
- **Order of predictions:** Does not handle an ordered list of preferences.
- **High ratings:** RMSE treats all ratings as equal.
 - Method A: good for high ratings and bad for the low ones.
 - Method B: good for low ratings and bad for the high ones.
 - In practice, we want method A over method B, but RMSE treats them equally.

Alternative Evaluation Schemes

	LOTR	Inception	Memento	Harry Potter	Fantastic Beasts
John	5	2	3	4	5
Mary	3	5	5	2	3
Lars			2	4	5
Mette		3		3	

		LOTR	labels	Inception	Memento	Harry Potter	Fantastic Beasts
John	Data points	1		0	0	1	1
Mary		0		1	1	0	0

Recommender system predicts K labels (movies) for each data point (user). This is *multi label* classification

R1(John) -> LOTR, Inception, Harry Potter, Memento.

Recall@K: percentage of true labels captured when we predict K labels per user.

Precision@K: percentage of predictions we get right.

John has true labels LOTR, HP, FB.

	LOTR	Inception	HP	Memento
# of true labels captured	1	1	2	2
Recall@K	1/3	1/3	2/3	2/3
# of predictions	1	2	3	4
Precision@K	1/1	1/2	2/3	2/4

Alternate Evaluation Schemes: Contd.

- So far

$$F1@k = \frac{2}{\frac{1}{\text{precision@k}} + \frac{1}{\text{recall@k}}}$$

It is hard to maximize precision and recall simultaneously.

- RMSE
- Precision@K and Recall@K

- What is a real scenario:

- We are given a set of recommendations
- The higher the precision, the better
- We want the good (relevant) recommendations to come early

$$\text{avgpre@k} = \frac{1}{k} * \sum_{j=1}^k \text{precision@j.relevant@j}$$

- Average precision:

$$\text{precision@8} = 0.5$$

$$\text{Avg precision@8} = (1 + (2/3) + (3/4) + (4/7))/8 = 0.37$$

R1	correct 1/1	incorrect 0	correct 2/3	correct 3/4	incorrect 0	incorrect 0	correct 4/7	incorrect 0
R2	correct 1/1	correct 2/2	correct 3/3	incorrect 0	correct 4/5	incorrect 0	incorrect 0	incorrect 0

$$\text{Avg precision@8} = (1 + 1 + 1 + (4/5))/8 = 0.47$$

Average Precision@K, Contd.

$$\text{avgpre@k} = \frac{1}{k} * \sum_{j=1}^k \text{precision@j.relevant@j}$$

- What if number of possible labels (L) < K ? Or, the algorithm predicts all relevant results, but then some?

$$\text{avgpre@k} = \frac{1}{\min(k, L)} * \sum_{j=1}^k \text{precision@j.relevant@j}$$

Another view:

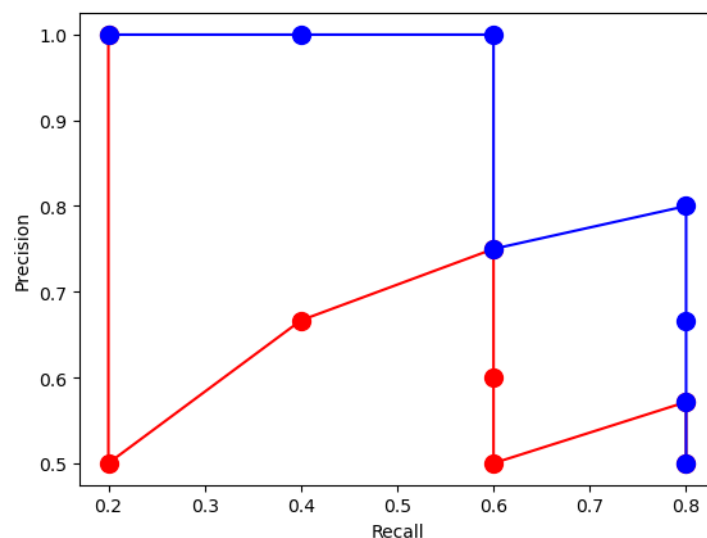
$$\text{avgpre@k} = \frac{1}{\min(k, L)} * \sum_{j=1}^k \text{precision@j} \cdot \Delta r(j)$$

$\Delta r(j)$ is the change in recall from $(j-1)^{\text{th}}$ subset to j^{th} subset.

Mean Average Precision (MAP@K): Average of avgpre@K across all users

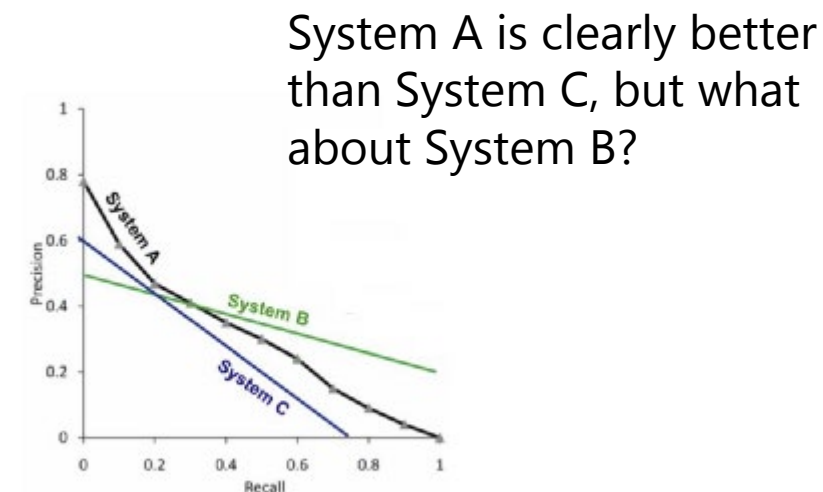
Average Precision@K, Contd.

- Precision Recall Curve
- We want to maximize the area under the precision recall curve
- This is what average precision measures (roughly)



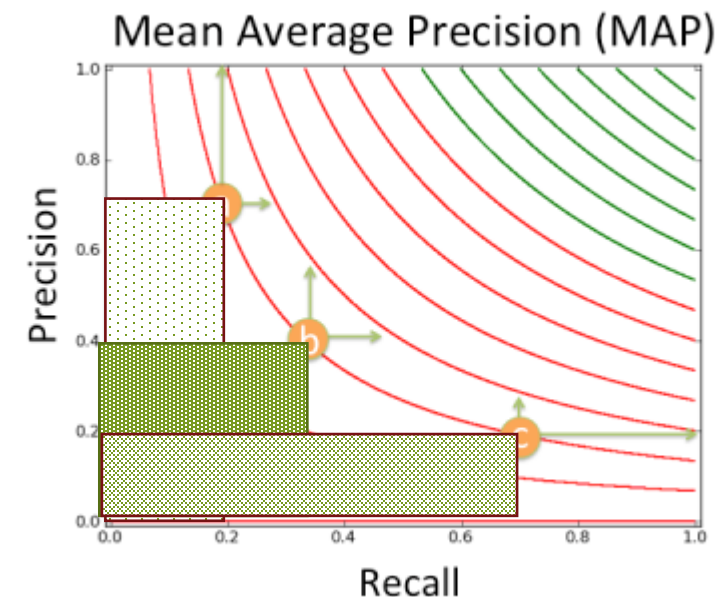
R1	1	0	1	1	0	0	1	0
R2	1	1	1	0	1	0	0	0

From previous slide



Summary of Binary Metrics

- Evaluation metrics so far:
 - RMSE
 - Precision and Recall @K
 - Average Precision and Mean Average Precision
 - Average precision approximates the area under the precision recall curve
- Problems with MAP:
 - Points a, b, and c all have the same MAP value. Vertical green arrows are how much Precision would have to increase to have the same affect as the increase in Recall marked by the horizontal arrows. The shorter the arrow the easier to achieve. The long arrows are particularly hard to achieve.
 - If you are at "a", improving precision is not going to help you a lot.



Mean Average Precision is "mean"

All metric so far have considered a recommendation either valid or invalid (0/1). What about a *notion of importance* for recommendations? In other words, recommendations are not just valid or invalid, some recommendations are more valid than the others.

Cumulative Gains

ranking	relevance
LOTR	1
Fargo	2
HP	2
La La Land	0
HP	1
CG	6
DCG	3.64

- Cumulative Gain (CG@K) $CG@K = \sum_{i=1}^K rel_i$
- Note: Relevancy score is not a rating that the system predicts: the system just produces the ranked list. The relevancy score is the rating that the evaluator provided for the movie.
- Suppose John rated 10 movies, and the system predicted this ranking. We can get the CG for the predictions.
- Does not account for the ranking of the items

- Discounted Cumulative Gain:
- DCG@5 from the table is 3.65. If you order them in decreasing order of relevance, DCG@5 goes to 4.19.

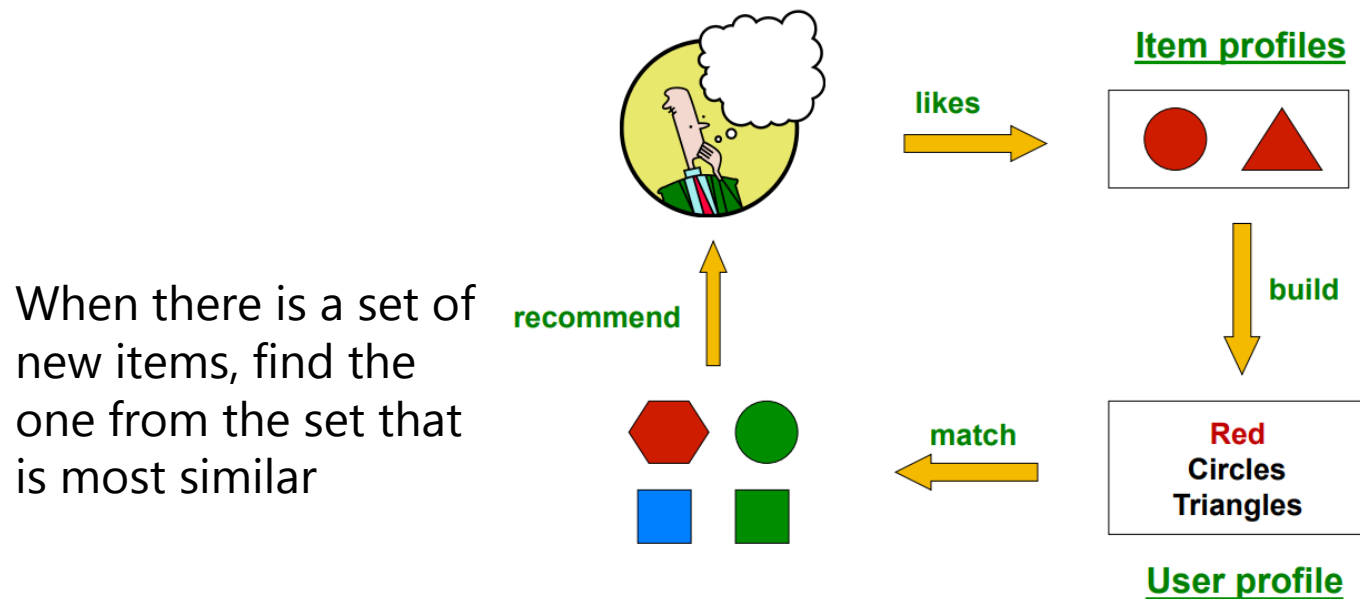
$$DCG@K = \sum_{i=1}^K \frac{rel_i}{\log_2(i + 1)}$$

Summary

- Recommender Systems: Problem Definition and Challenges
- Different Evaluation Metrics

Content Based Recommendation

- Recommend items to customer x that are similar to previous items rated highly by x



Mining of Massive Datasets

Item	Color	Shape
item 1	Red	Circle
Item 2	Red	Triangle

Represent the user in terms of the items she likes

Building Item and User Profiles

- Profile is a set(vector) of features
 - Movies: title, director, genre
 - Product: manufacturer, usage
- We can complement these features with text features:
 - From the reviews, get the most important words (remember tf-idf)?
- User Profile:
 - Average of the item profiles the user has rated.
 - We can use a weighted average (likes one movie more than the other)
- Prediction:
 - Given user profile \mathbf{x} and item profile \mathbf{i} compute the distance between them.
 - Cosine/ Euclidian
 - When to use Cosine vs Euclidian (when the magnitude is not important)
 - Exercise: What happens when the vectors are normalized, i.e., $||\mathbf{x}||_2 = ||\mathbf{i}||_2 = 1$?

Content-based Approach: Pros

- Recommendation is not directly inferred from the rating matrix.
- No need for data on other users
 - Avoids cold start problems

Cold start:

- New items have no ratings.
 - New users have no history.
 - It's not enough that one user provides ratings for multiple items (or one item has multiple user ratings), multiple users must rate multiple items.
- Able to recommend new and unpopular items
 - No first rater problem
 - Able to provide explanations:
 - If you recommend a news article that talks about finance, you can say that the user spent a lot of times in the past reading financial news.

Content-based Approach: Cons

- Finding the appropriate features is hard.
 - What is a “good” representation of a movie? Actor/Director (explicit) or Genre/Use of color palate (implicit)?
 - What about images
- Overspecialization
 - Unable to exploit information from other users: John has rated Harry Potter + LOTR movies, Mary has rated LOTR and Fantastic Beast, John will never be recommended FB or Mary HP.
- Cold start problem for “new users”
 - How to build an item profile if no item is rated?

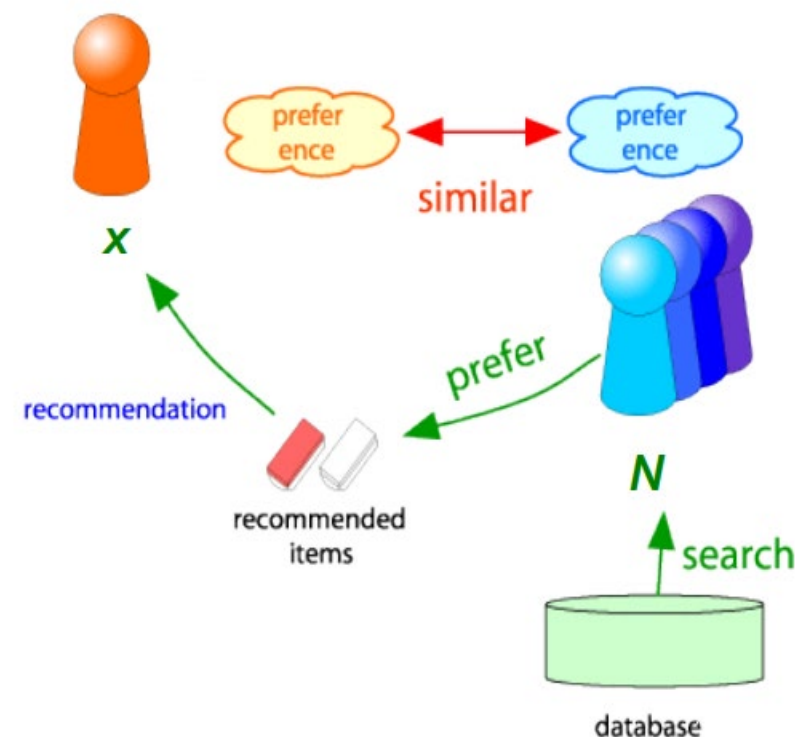
Collaborative Filtering

- Consider User X
- Find a set N of other users whose ratings are "similar" to X.
- Estimate X's ratings based on ratings of users in N.

	HP1	HP2	HP3	Twilight	SW1	SW2	SW3
John	4			5	1		
Mary	5	5	4				
Lars				2	4	5	
Mette		3					3

Can we infer the blank ratings from the users?

- Finding the appropriate features is hard.
 - What is a "good" representation of a movie? Actor/Director (explicit) or Genre/Use of color palate (implicit)?
 - What about images



Similar Users

- | | HP1 | HP2 | HP3 | Twilight | SW1 | SW2 | SW3 |
|-------|-----|-----|-----|----------|-----|-----|-----|
| John | 4 | | | 5 | 1 | | |
| Mary | 5 | 5 | 4 | | | | |
| Lars | | | | 2 | 4 | 5 | |
| Mette | | 3 | | | | | 3 |

- Consider users John, Mary and Lars
- Intuitively, John is similar to Mary because they both like Harry Potter movies.
- John is different from Lars because while John likes Twilight and hates Star Wars, the opposite is true for Lars.
- Formally, we need a similarity metric Sim such that $\text{Sim}(\text{John}, \text{Mary}) > \text{Sim}(\text{John}, \text{Lars})$

Similarity Function: Euclidian Distance and Cosine Similarity

	HP1	HP2	HP3	Twilight	SW1	SW2	SW3
John	4			5	1		
Mary	5	5	4				
Lars				2	4	5	
Mette		3					3

john	4	0	0	5	1	0	0
Mary	5	5	4	0	0	0	0
Lars	0	0	0	2	4	5	0
Mette	0	3	0	0	0	0	3

- Euclidian distance:

$$\sqrt{\sum_{i=1}^n (x_i - y_i)^2}$$

- ED(J, M) = 8.25
- ED(J, L) = 7.68

```
import numpy as np
def euclidean_distance(x, y):
    return np.sqrt(np.sum((x - y) ** 2))
```

According to ED, John is more similar to Lars than Mary

- Cosine similarity (x, y): $\frac{x \cdot y}{\|x\| \cdot \|y\|}$
- Cosine(John, Mary) = 0.38
- Cosine(John, Lars) = 0.32

```
import numpy as np
def cosine_similarity(x, y):
    return np.dot(x, y) / (np.sqrt(np.dot(x, x)) *
                           np.sqrt(np.dot(y, y)))
```

Now John is similar to Mary than Lars, but not too much

Similarity Functions: Centered Cosine

- There's a problem in measuring similarity using Euclidian and Cosine distance
 - Does not capture the notion of similarity well
- Problem: treats missing ratings as negative
 - Does not capture the notion that 0 means it was not rated, not somebody really disliked the movie.
 - Had John rated HP2 and HP3, he would have probably rated them highly.
- Centered Cosine: Normalize ratings by subtracting row mean

	HP1	HP2	HP3	Twilight	SW1	SW2	SW3	Row mean
John	4			5	1			10/3
Mary	5	5	4					14/3
Lars				2	4	5		11/3
Mette		3					3	3

Centered Cosine: Contd.

	HP1	HP2	HP3	Twilight	SW1	SW2	SW3
John	2/3			5/3	-7/3		
Mary	1/3	1/3	-2/3				
Lars				-5/3	1/3	4/3	
Mette		0					0

- We have “centered” the rows, i.e., row sums are now 0.
- We have introduced the concept of negative rating: if you dislike the movie on average than the movies you have rated, you have a –ve rating for that movie.

```
In [52]: def centered(x):
...:     avg = np.sum(x)/float(np.count_nonzero(x))
...:     for i in x:
...:         if i == 0:
...:             yield i
...:         else:
...:             yield i - avg
...:
In [53]: def cosine_similarity(x, y):
...:     x = list(centered(x))
...:     y = list(centered(y))
...:     return np.dot(x, y) / (np.sqrt(np.dot(x, x)) * np.sqrt(np.dot(y, y)))
...:
In [54]: print("john",john, "mary", mary, "lars", lars)
john [4, 0, 0, 5, 1, 0, 0] mary [5, 5, 4, 0, 0, 0, 0] lars [0, 0, 0, 2, 4, 5, 0]
In [55]: print(cosine_similarity(john, mary))
0.09245003270420475
In [56]: print(cosine_similarity(john, lars))
-0.5590852462516899
In [57]:
```

CosineSim(John, Mary) = 0.09

CosineSim(John, Lars) = - 0.56

In summary, centered cosine:

- Captures intuition better
- Handles “tough raters” and “easy raters” (one examiner may grade more leniently than the other, how to compare them?)
- Also known as **Pearson Correlation**

User Similarity to Unknown Ratings

- We have a way to measure the similarity between users.
- How do we translate that to unknown ratings (which was the goal to begin with)?

	HP1	HP2	HP3	Twilight	SW1	SW2	SW3
John	4	?		5	1		
Mary	5	5	4				
Lars				2	4	5	
Mette		3					3

- Rating of item i by user x is unknown:
 - (John, HP2)
- Let r_x be the vector of user x 's ratings
- Let N be the set of K users most similar to x who have also rated item i .
 - Can't use Mary to estimate the rating of SW2 for John.
- Average rating from all users U for item i (who have rated it).

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

$$K = |N|$$

- Weight this average by the similarity values:
$$r_{xi} = \frac{\sum_{y \in N} s_{xy} \cdot r_{yi}}{\sum_{y \in N} s_{xy}}$$

Item-Item Collaborative Filtering

- So far: User-user collaborative filtering
- Item-Item collaborative filtering
 - For item i , find other similar items (in terms of users!)
 - Estimate ratings for item i based on ratings for similar items
 - Can use the same similarity metrics as in user-user model.


- r_{xi} rating of user x for item i .
- $s_{ij} \cdot r_{xj}$ (similarity between item i and j). (rating of all items j that has been rated by user x)
- $N(i; x)$ set of items rated by user x that are similar to item i .

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

Item-Item CF: ($|N| = 2$)

	users												
	U1	U2	U3	U4	U5	U6	U7	U8	U9	U10	U11	U12	CCSim(1, m)
M1	1		3		?	5			5		4		1.
M2			5	4			4			2	1	3	-0.18
M3	2	4		4	2		3		4	3	5		0.41
M4		2	4		5			4			2		-0.10
M5			4	3	4	2					2	5	-0.31
M6	1		3		3			2			4		0.59

 unknown rating

 rating between 1-5

Want to find $r_{(M1, U5)}$

Step 1: Find similarity between movies

```
In [70]: movies = np.array(
...: [
...:     [1, 0, 3, 0, 0, 5, 0, 0, 5, 0, 4, 0],
...:     [0, 0, 5, 4, 0, 0, 4, 0, 0, 2, 1, 3],
...:     [2, 4, 0, 1, 2, 0, 3, 0, 4, 3, 5, 0],
...:     [0, 2, 4, 0, 5, 0, 0, 4, 0, 0, 2, 0],
...:     [0, 0, 4, 3, 4, 2, 0, 0, 0, 0, 2, 5],
...:     [1, 0, 3, 0, 3, 0, 0, 2, 0, 0, 4, 0]
...: ], dtype=np.float32)

In [71]: sims = [cosine_similarity(movies[0], movies[x]) for x in range(len(movies))]

In [72]: sims
Out[72]:
[0.9999999999999999,
-0.17854212213729673,
0.41403933560541256,
-0.10245014273309601,
-0.30895719032666236,
0.5870395085642742]
```

Item-Item CF: ($|N| = 2$)

users

	U1	U2	U3	U4	U5	U6	U7	U8	U9	U10	U11	U12	CCSim(1, m)
M1	1		3		?	5			5		4		1.
M2			5	4			4			2	1	3	-0.18
M3	2	4		4	2		3		4	3	5		0.41
M4		2	4		5			4			2		-0.10
M5			4	3	4	2					2	5	-0.31
M6	1		3		3			2			4		0.59

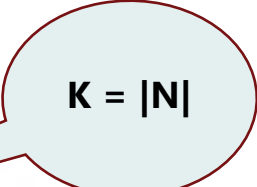
Want to find $r_{(M1, U5)}$

- Find similarity between movies (done)
- Choose neighborhood of size 2 (movie 3 and movie 6)
- $s_{1,3} = 0.41$, $s_{1,6} = 0.59$
- $r_{(M1, U5)} = (0.41*2 + 0.59*3) / (0.41 + 0.59) = 2.6$

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

Item-Item v User-User

- In theory, user-user and item-item are dual approaches
- In practice, item-item outperforms user-user in many use cases
- Why?
 - User's own rating is used to predict the recommendation (flip side: less diverse results).
 - Concrete reason for the recommendation: "because you watched X, the recommendations are List[y]"
 - More stable:
 - Number of users is much higher than number of items
 - Two users usually have a very small number of mutually rated items. Two items have a very large number of users that rated both.
 - Addition of some ratings can change the result for user vectors drastically, but not the same for the items.

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


$K = |N|$

Pros/Cons of Collaborative Filtering

- Works without explicit feature selection for items
 - Users are the features for the items and vice versa
- Cold start
 - Need enough user-rating pairs.
 - Sparsity
 - The user/ratings matrix is sparse: hard to find users that have rated the same items
 - First rater
 - Can not recommend an unrated item
- Tends to recommend popular items: if some movies are popular across the board, they will always be recommended (can you see the equivalence with the idf component in tf-idf calculation?)
- Complexity for item-item (will be similar for user-user):
 - **Offline**: calculate item neighborhoods (assume there are U users and N items):
 - Each item has a dimension of U (item is represented in the user space). $\text{Sim}(\text{item}_x, \text{item}_y) = \mathcal{O}(U)$
 - Do this for all possible item pairs, total complexity is $\mathcal{O}(N^2U)$
 - **Online**: calculate the rating based on the neighborhood (K): $\mathcal{O}(K)$

Offline is not truly offline because new user/items are added everyday

Recap So Far

- Recommender Systems: 1. Predict unseen rating for user u and item or 2. Rank a set of items for user u (a natural extension of 1).
- Methods
 - Content based: each item (i) is represented in a feature space, each user (u) is represented with an aggregation function of the items she rates, rating for a new item (i') is calculated by the $\text{Sim}(u, i')$. **Problem: need to get the feature space!**
 - Collaborative filtering or neighborhood-based methods:
 - User-User: Represent user (u) in the item space, choose a *user* neighborhood, the rating (u, i) is an aggregation of all user (u') ratings for item i such that u' is in user neighborhood for u .
 - Item-Item: Represent item (i) in the user space, choose an *item* neighborhood, the rating (u, i) is an aggregation of all item (i') ratings for user u such that i' is in item neighborhood for i .
 - This is very similar to the content-based method, just the features are not manually engineered.

Latent Factor Based Models

- Assume you have a rating matrix S with U (user) rows and I (item) columns.

1. For a target entry (u, i) determine the most similar rows/columns of the ratings matrix with the use of the cosine coefficient between rows/columns. For user-based methods rows are used, whereas for item-based methods, columns are used.
2. Predict the target entry (u, i) using a weighted combination of the ratings in the most similar rows/columns determined in the first step.

Collaborative filtering: A unified view

- Problems

- Complexity (can be reduced by doing some offline clustering)
- Sparsity: Most (u, i) values are not available, but we are calculating similarity between items/users based on them.
 - Not a very robust method. We can use a dimensionality reduction technique to find a d dimensional representation for a user (item) where $1. d < U (N)$, and the missing values are filled up.
 - We are still not looking at the cross correlation between users and items
- Can there be a method that can 1. get the missing values, 2. capture the cross correlation between users and items?

Latent Factor Models: What is Latent

Person	Height	weight	Likes color
P1	5'11"	180 lbs	Blue
P2	5'4"	140 lbs	Red
P3	6'2"	240 lbs	Blue
P4	5'10"	110 lbs	Red
P5	5'6"	170 lbs	Red
P6	5'8"	190 lbs	Blue

Suppose all females like red and all males like blue.

The model is given the dataset of height and weight and asked to predict the color the person would like.

Is the latent factor **Gender**?

If it is, you can now represent each data point in 1 dimension which is directly correlated with the color: the prediction becomes easy!

	NERO	Julius Caesar	Patriot	Sleepless Seattle	Pretty Woman	Casablanca
1	1	1	1	0	0	0
1	1	1	1	0	0	0
1	1	1	1	0	0	0
-1	-1	-1	-1	1	1	1
-1	-1	-1	-1	1	1	1
-1	-1	-1	-1	1	1	1

1. Ratings between -1 and 1.
2. No missing values

There is a latent factor: movie genre {history, romance}.

Latent Factor Models: How Do They Help Here?

NERO	Julius Caesar	Patriot	Sleepless Seattle	Pretty Woman	Casablanca
1	1	1	0	0	0
1	1	1	0	0	0
1	1	1	0	0	0
-1	-1	-1	1	1	1
-1	-1	-1	1	1	1
-1	-1	-1	1	1	1

1. Ratings between -1 and 1.
2. No missing values

There is a latent factor: movie genre {history, romance}.

- Suppose there is a method to tell me that there is one factor "genre" for the movies.
- This happens:

history	romance
1	0
1	0
1	0
-1	1
-1	1
-1	1

NERO	Julius Caesar	Patriot	Sleepless Seattle	Pretty Woman	Casablanca
1	1	1	0	0	0
0	0	0	1	1	1

Suppose a new user comes in, and ranks patriot as 1, we immediately know that the scores for all movies are going to be [1, 1, 1, 0, 0, 0]

Why is This Possible? Implicit Correlations in Data

- There are implicit correlations in data

$$\begin{bmatrix} 1 & 2 & 1 \\ -2 & -3 & 1 \\ 3 & 5 & 0 \end{bmatrix}$$

M1 (data matrix)

$R1 = R2 + R3$, let's consider them as "basis vectors"

$$\begin{bmatrix} -2 & -3 & 1 \\ 3 & 5 & 0 \end{bmatrix}$$

P (Projection matrix)

New representation of the data with these basis vectors

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}$$

M2: New representation. $M2 \times P = M1$

Last lecture

- The last matrix had 6 rows and 6 columns, but a rank of 2 (verify).
- Any $m \times n$ matrix R of rank $k \ll \min\{m, n\}$ can always be expressed in the following product form of rank- k factors: $R = UV^T$
- Even when the matrix R has rank larger than k , it can often be approximately expressed as the product of rank- k factors $R \approx UV^T$

Approximate Matrix Factorization

- Even when the matrix R has rank larger than k , it can often be approximately expressed as the product of rank- k factors $R \approx UV^T$

NERO	Julius Caesar	Cleopatra	Sleepless Seattle	Pretty Woman	Casablanca
1	1	1	0	0	0
1	1	1	0	0	0
1	1	1	0	0	0
1	1	1	1	1	1
-1	-1	-1	1	1	1
-1	-1	-1	1	1	1
-1	-1	-1	1	1	1

- Ratings between -1 and 1.
- No missing values

history	romance
1	0
1	0
1	0
1	1
-1	1
-1	1
-1	1

NERO	Julius Caesar	Cleopatra	Sleepless Seattle	Pretty Woman	Casablanca
1	1	1	0	0	0
0	0	1	1	1	1

U4 likes both history and romance and Cleopatra belongs to both.

This is an approximate decomposition because the original matrix had rank 3.

There is a latent factor: movie genre {history, romance}. This matrix has rank 3 (verify).

Latent Factor Models: Variations of Matrix Factorization

- Latent factor models “discover” a good feature space that governs the data generation.
- Most latent factor models will use different forms of matrix factorization.
- We have so far talked about fully specified matrices (all values are given), but they don't have to be.
 - We will see how in the next lecture.

Summary of this Lecture + What to Expect Next

- Recommender systems: what are they and why they are useful.
- Evaluation method for recommender systems (they are very common with information retrieval/ search engine methods)
- Three ways of developing recommender systems
 - Content based
 - Collaborative filtering
 - Latent factor models
- Next lecture
 - More on latent factor models.
 - Some tricks in collaborative filtering.
 - DNN based models for recommender systems: a very short intro.

A photograph of two scientists in a laboratory. In the foreground, a man with dark hair and glasses, wearing a white lab coat over a blue shirt, is holding a small green plastic container with both hands. Inside the container are several green, grass-like plants. He is looking down at the plants. Behind him, another man with a beard and glasses, also in a white lab coat over a plaid shirt, is looking at the same plants. The background is a bright, out-of-focus laboratory setting with windows and shelves.

Attend the QA session on Monday
for clarifications and interesting
discussions!