

ORIE 5355

Lecture 6: Recommendations – from predictions to decisions

Nikhil Garg

Announcements

Last time: Prediction (filling in missing entries)

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

Questions on prediction?

What to *do* with predictions? Naïve method

Train a single matrix factorization model using some data (what data?)

→ I have predictions for each item and each user

For example, predict $r_{ij} = u_i \cdot v_j$

For each user i , simply recommend the best item

$$\operatorname{argmax}_j u_i \cdot v_j$$

(Or K best items)

Issues with naïve method

- Capacities

What if you only have 5 of item j , and everyone likes item j ?

- Multi-sided preferences

Recommendations in freelancing markets (workers matched with clients), dating apps, volunteer platforms, etc

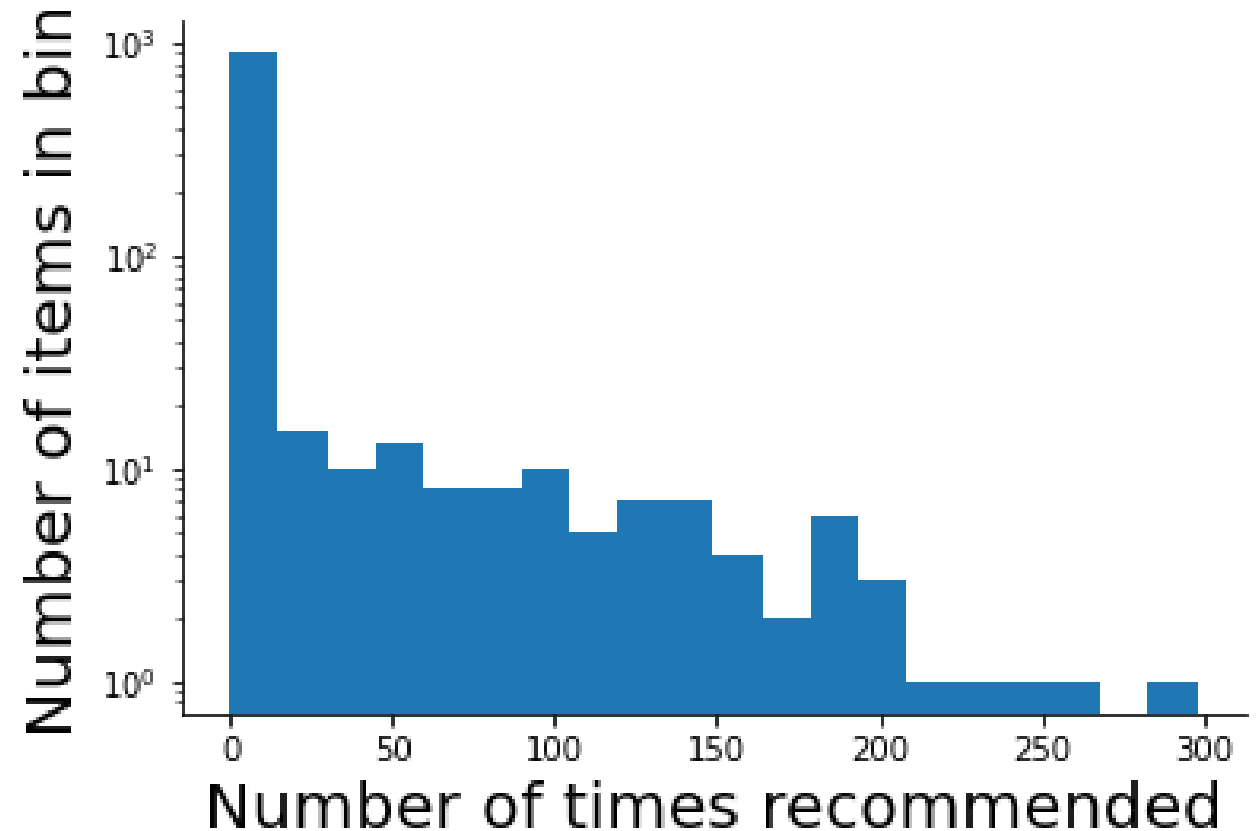
- Challenges in recommending *sets* of items

- *Diversity* of items recommended
- Behavioral effects? Recommending one item makes another item more popular

Today: going from predictions → recommendations

An example

- In the homework, we ask you to first recommend using the “naïve” method of just recommending best prediction for each user
- You’ll observe a plot like the following



Dealing with capacity constraints

Overview

- What's the challenge, exactly?
- Solving an “easier” problem: “maximum weight matching in a bipartite graph”
- Insights from the easier problem to real-life applications

The challenge

- In many (non-online-media) settings, you are recommending “items” with capacity constraints:
 - You have a finite number of each item in your warehouse
 - An Airbnb can only be booked by one customer at a time
 - Workers can’t work for every client; a client can only hire 1 person
 - People on dating apps – can’t talk to everyone
- If you ignore these capacity constraints, then everyone may be recommended the same (limited) item
 - Some people will be left out
- (How) should you factor in capacity in your recommendations?

The challenge, formally (simple version)

- You have N users and M items, but only 1 copy of each item
- You want to recommend 1 item $j(i)$ to each user i
- Each user i will consume the item that you recommend them
- You want to maximize the sum of predicted ratings of consumed items

$$\sum_i r_{ij(i)}$$

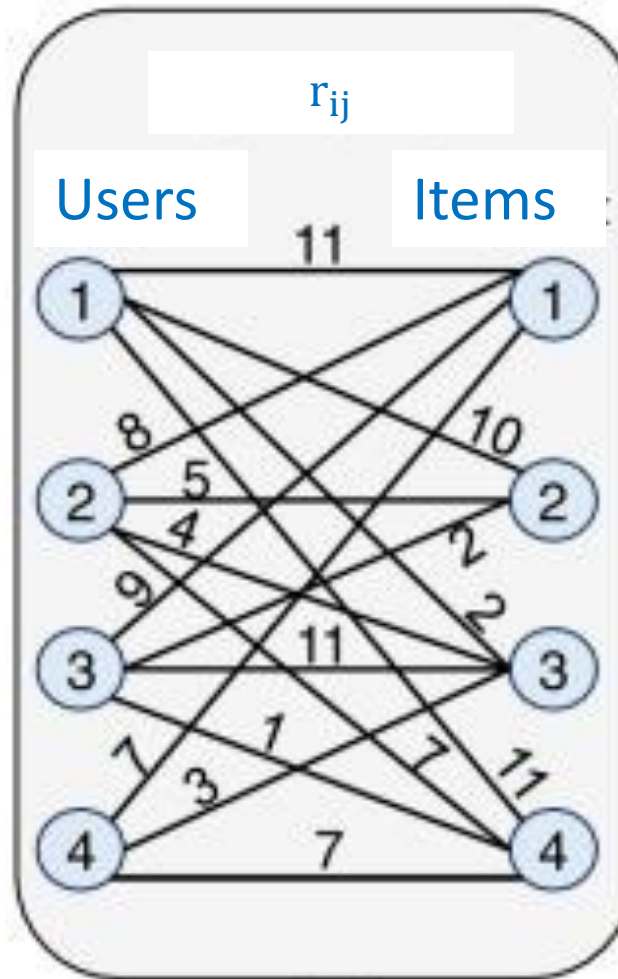
- However, each item can only be recommended once

$$j(i) \neq j(i') \text{ unless } i = i'$$

Solving the simple case

It turns out that this simple case is called “maximum weight matching”

Draw a graph with users on one side and items on the other



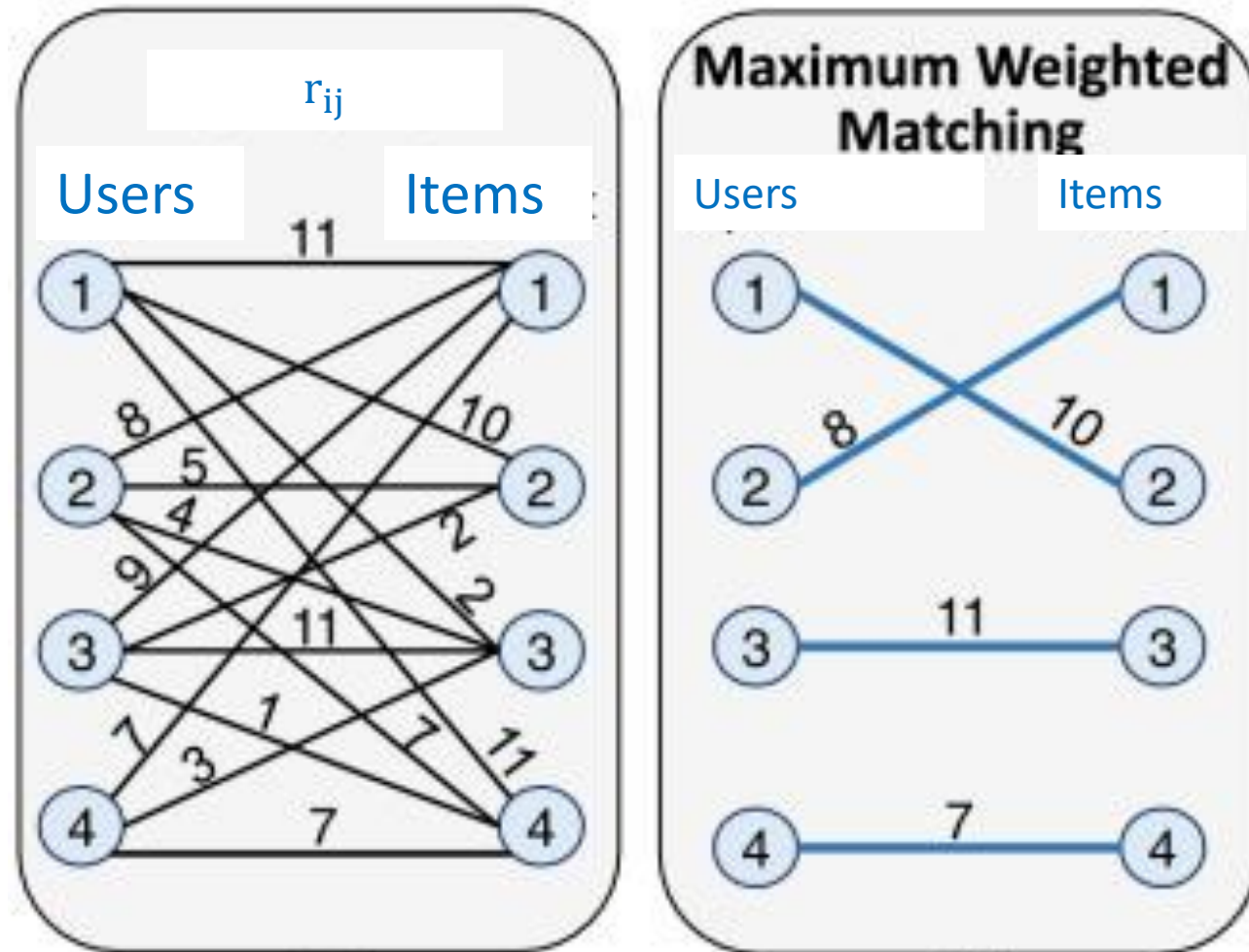
Solving the simple case

It turns out that this simple case is called “maximum weight matching”

Draw a graph with users on one side and items on the other

Find the “matching” that maximizes sum of edge weights

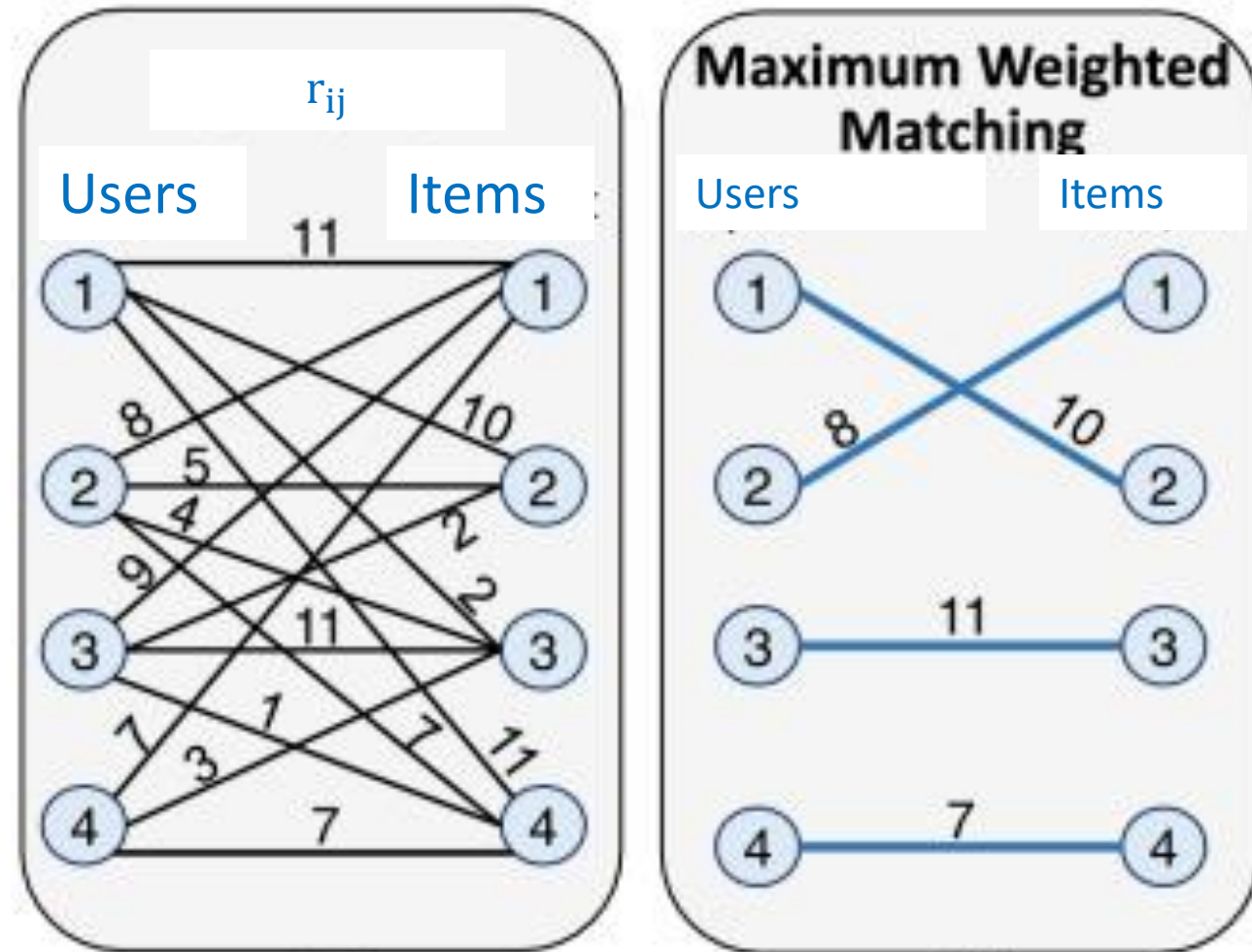
[scipy.optimize.linear_sum_assignment — SciPy v1.7.1 Manual](#)



Insights from the simple case

In general, the actual solution might be combinatorial – a complex function of all the joint preferences

- Some *users* are not matched with their most preferred item!
- Some *items* are not matched with the user that likes it the most!
- If a user likes multiple items similarly, maybe they get their 2nd choice
- Users who are “picky” (only like one item) may get that item, over users who like that item more



Challenges in using max weight matchings

- Everyone doesn't show up at once
 - New users come in tomorrow – have to leave items for them
- You can't “match” people, only recommend them items
 - Someone may not consume the item!
- “Capacity” constraints are also soft
 - New items are shipped to warehouse all the time
 - Maybe you can spend more money to expedite shipment
- Computational constraints in rerunning large scale max weight matchings with every new user

What to do in practice

- Finding a “great” solution requires a lot of careful data science + modeling work
- Some reasonable heuristics:
 - “Batching”: If you don’t have to give recommendations immediately, wait for some number of users to show up and solve max weight matching (for example, every hour)
 - “Index” policies: For each user, create a “score” for each item and just choose recommend the item(s) with the highest score(s)

Index policies

- We want a score (index) between each item j and user i : s_{ij}
- Then, for each user, pick the item with the max score: $\operatorname{argmax}_j s_{ij}$
(out of those that still have capacity)
- We've already seen an example: if the only thing that matters is predicted rating, then $s_{ij} = r_{ij}$
- Why index policies?
 - They're efficient: for each user, only need to consider their scores
 - They can be *explained* to users
 - All information about other users is contained in how score is constructed

Constructing index policies

What matters in constructing an index policy?

- The higher the ratings by other users for an item, the smaller s_{ij} should be
- The less capacity C_j left for the item, the smaller s_{ij} should be

An *example* score function

$$s_{ij} = \alpha_j \left[\frac{r_{ij}}{\bar{r}_j} \right] C_j^\beta$$

where α_j, β are some (learned) parameters over time, and $\bar{r}_j$ is the mean rating for the item (across all users)

α_j : Item is “special” and should be over-recommended

β : Relative importance of capacity. ($\beta = 0$ means ignore capacity)

Many possible score functions! Should be application specific

Capacity constraints lessons

- If you just recommend each user their highest predicted scores, then you might not be *globally* efficient
- Even if you can't implement it, taking intuition from the “optimal” solution is often valuable
- Index policies: even if “optimal” solution requires combinatorial constraints, “practical” solution can decompose the problem

Multi-sided preferences

Multi-sided preferences

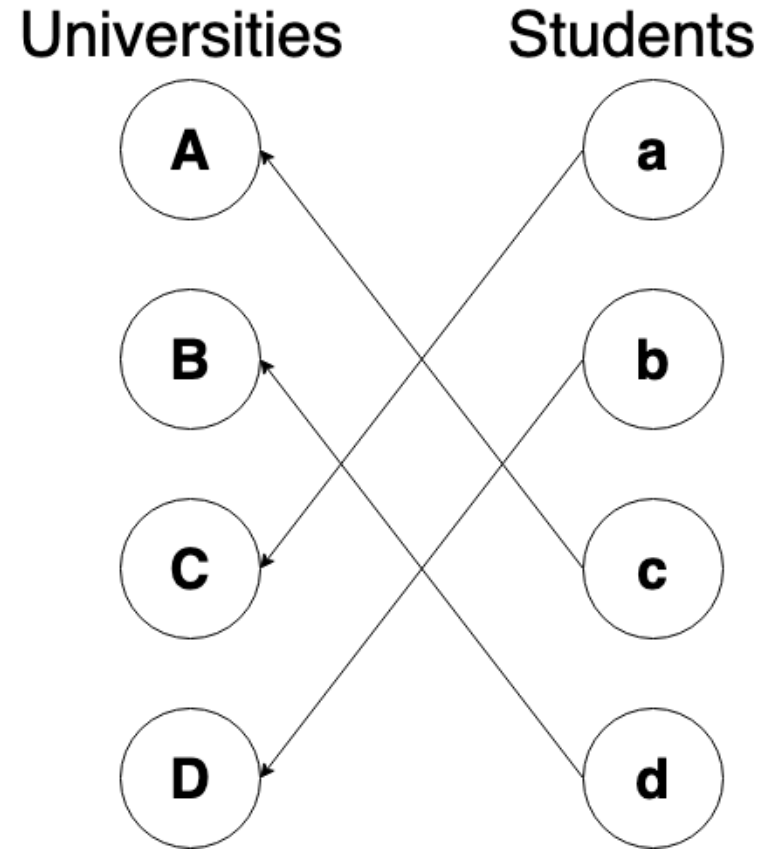
- In many modern online markets, both sides have preferences
Freelancing markets (workers matched with clients), dating apps, volunteer platforms, etc
- A match only happens if *both* sides like each other
And have capacity...

The challenge, formally (simple version)

- You have N workers and N clients
 - Each worker can only work with 1 client; each client only hires 1 worker
- Each side has preferences (predicted ratings) over the other side
- You want to create “good” matches
 - Good for who? Workers? Clients? Some combination?
- One possible goal: create “stable” matches

“Stable matching” in 1 slide

- Stable matching:
 - Given rank order preferences from each person on each side
 - Match the sides such that matches are “stable”: No potential pair wants to abandon their current partners for each other.
- Efficient to find: “Gale-Shapley algorithm”
- Used to allocate:
 - Medical students to residencies
 - Students in NYC to high schools



Challenges in using stable matching

Same as from using maximum weight matchings

- Everyone doesn't show up at once
 - New users come in tomorrow – have to leave items for them
- You can't "match" people, only recommend them items
 - Someone may not consume the item!
- "Capacity" constraints are also soft
 - New items are shipped to warehouse all the time
 - Maybe you can spend more money to expedite shipment
- Computational constraints in rerunning large scale stable matchings with every new user

Just more complicated with both sides now having preferences

Doesn't guarantee maximizing sum of utilities!

Intuition from stable matching to recommendations

What matters in constructing an index policy?

- The higher the ratings by other workers/clients, the smaller s_{ij} should be
- If either worker i or client j has been recommended to many other people in the past, the smaller s_{ij} should be
Equivalent of “capacity”
- Now, *both* i ’s rating for j and j ’s rating for i matter
- From stable matching: *both* i and j matter – one-sided high score can’t “make up” for the other side being a low score

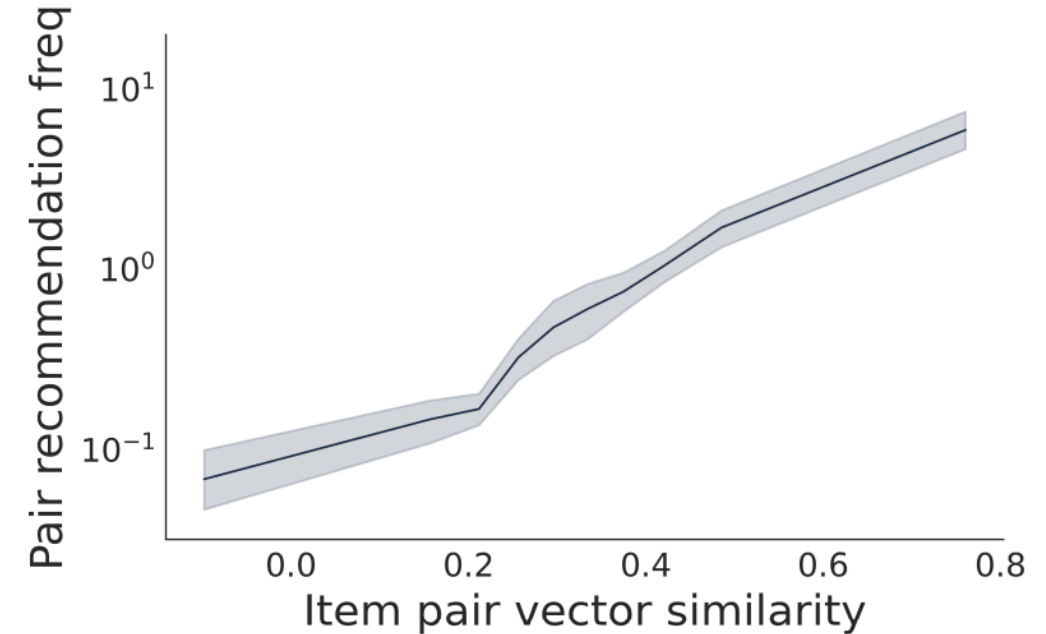
An *example* heuristic score function

$$s_{ij} = \min \left(\frac{\alpha_j r_{ij} c_j^\beta}{\bar{r}_j}, \frac{\alpha_i r_{ji} c_i^\beta}{\bar{r}_i} \right)$$

Diversity in recommendations

Diversity of recommendations

- If you do the naïve method and recommend multiple items to each user, then you're not going to recommend a *diverse set* of items
- Why? If you have a single user vector $\mathbf{u}_i$, then if two items j and k both have large dot products $\mathbf{u}_i \cdot \mathbf{v}_j$ and $\mathbf{u}_i \cdot \mathbf{v}_k$, then they tend to be similar, $\mathbf{v}_j \approx \mathbf{v}_k$

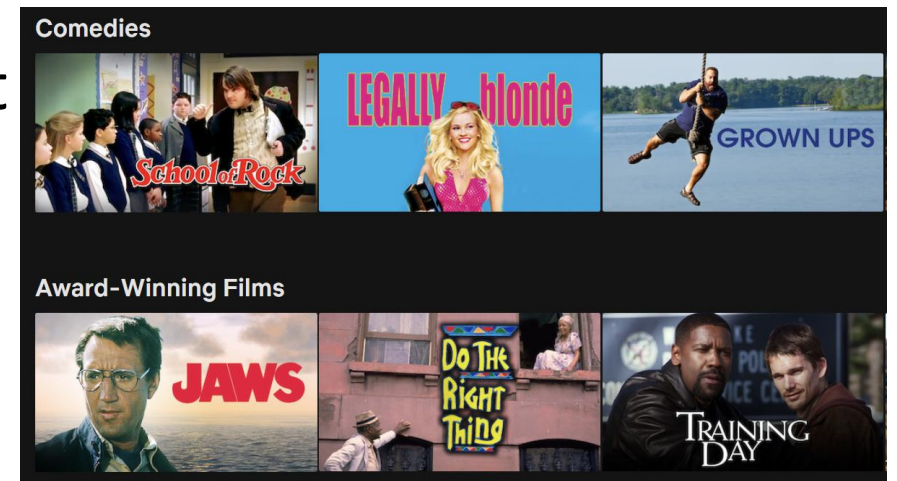


With the MovieLens dataset and recommending 2 items to each user. The more similar 2 items are, the more likely they are to be recommended together compared to their “marginal” distributions

Improving diversity of recommendations

Many possible approaches

- Create a “short list” of items based on just the prediction (“relevance”), and then select a diverse set from the short list
- Pre-select topics and then most relevant within each topic
- Start from most relevant item, filter other items that are too similar to items already recommended



Other challenges

Choice: users can reject recommendations + select something different

Feedback: Recommendations affect future training data

Agency/control: Users should be able to express their preferences

Societal level effects

- Is everyone consuming the same content? (Winners take all?)
- Or are we all consuming very personalized content? (Filter bubbles?)

Summary of recommendations

There are 3 steps to building a recommendation system:

- Choose the *data* that you will use
What does the data imply about people's opinions and future desires?
- Train a model to *predict* ratings between pairs of items and users
Different approaches (item- and user similarity, matrix factorization)
Can also combine approaches
- *Recommend* items based on predictions and other concerns
Capacity constraints, diversity, fairness considerations, long-term objectives

Questions?