ORIE 5355

Lecture 5: Intro to Recommendation Systems

Nikhil Garg

Announcements

Data collection: Possible Errors

- Finite sampling error: I got some small number of responses compared to if I had asked a billion people (source of variance)
- Measurement error: Given someone responded, their response doesn't equal their actual opinion
- **Selection effect: true opinion among those who respond differs from population true opinion**
- Construct mismatch: Y isn't actually what I wanted!

Data collection: some approaches

- When collecting data, you can *stratify sample* groups: sample a fixed number of respondents per group, to counter selection effects and reduce variance
- After you already have the data, you can *reweight* the data to try to counter some of these effects
- These solutions are imperfect, and counter (some) selection effects but do not resolve construct mismatch, or measurement error

Data collection module summary

Never take opinion data at face value. Always ask:

1. What did I measure, versus what did I care to measure?
2. Who answered versus what's the population of interest
3. What am I going to *do* with the data, and how does that affect data collection?

Will show up in the rest of the course!

Recommendation systems

Module overview

Part 1 (today) – Prediction

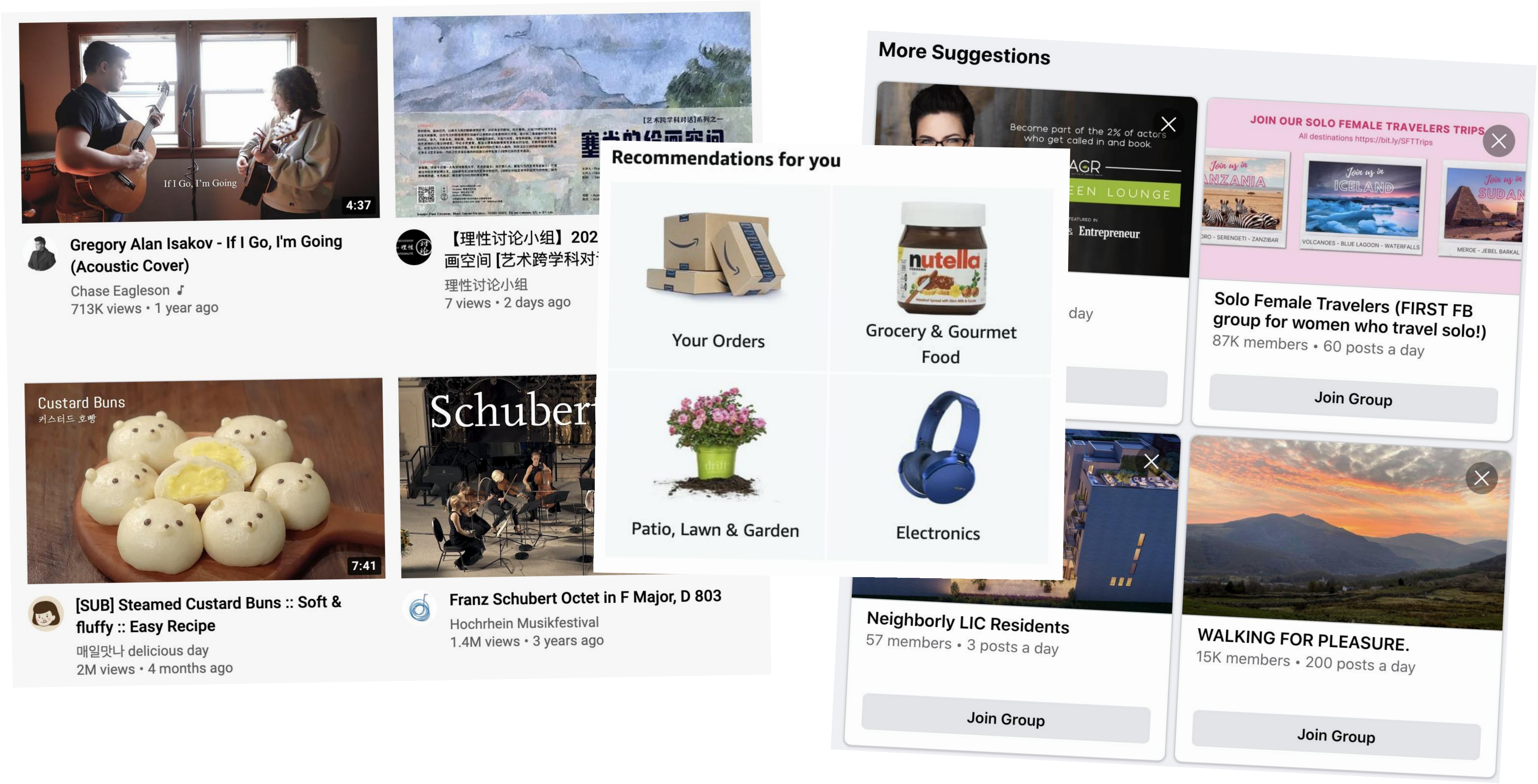
How much will a given user like an item?

- Problem formulation and some algorithms
- Data challenges

Part 2 (next time) – From predictions to decisions

How to use the predictions to recommend items in practice?

- Capacity constraints
- Recommendations in *2 sided* markets
- Feedback loops in recommendations



Slide credit: Amy Zhang, Cornell

Types of Recommendations

Editorial and hand curated

- List of favorites
- Lists of “essential” items

Simple aggregates

Top 10, Most Popular, Recent Uploads

Tailored to individual users (Personalized recommendations)

Amazon, Netflix, …

Personalized recommendations

- Motivation: filter the content to be more relevant for each individual
- Data Inferred from signals
  - Direct: ratings, feedback, etc
  - Indirect: purchase history, access patterns, etc
- Intermediate Goal: *predict* the relevance of each item for each user

Formal Model

- ***Users* will be indexed by *i***
- ***Items* will be indexed by j**

Predicted Utility: let r_{ij} denote our *prediction* for the utility that a user would get from item if they watched it. e.g., **0-5** stars, real number in **[0,1]**

(Remember distinction between construct and variable)

Ratings Matrix: suppose we have data $\hat{R}$

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

In reality, the vast majority of entries are missing

Goal: fill in the missing entries!

Example Metric: mean squared error on some heldout set of user/item pairs

Two Steps

Step 1: create a data matrix $\hat{R}$ from signals you have

Step 2: fill in the missing entries using some prediction model

Step 1: Using explicit data

Just ask people what they think

Challenges: all the opinion collection challenges already talked about!

- Answering rates
- Measurement error: does a scale reflect how much they like something?
- Are people consistent over time?

Step 1: Implicit data

- You have many implicit signals about people's opinions
  - Do they finish watching the show, or start watching the next episode?
  - Do they keep coming back and buying other things
  - Did they browse other items instead of putting something in their cart?
  - Do they re-hire the same freelancer/work with the same client again?
- These give *different* information than do explicit ratings
  - From a different population of users
  - Often more numerous, but harder to analyze
  - "revealed preference" – might be more predictive of future behavior
- Using such data
  - Train models to predict different future behavior, using various signals
  - Might take away "user agency" – what if they want to change their behavior? Predicting behavior is not the same as helping someone with their goals

Step 2: Filling in the missing entries

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

Possible strategies

- Content-based recommendations:
  Use existing data on items to group together similar items
- User-similarity-based recommendations
  Find similar users and use data from each other (e.g., demographics)
- Matrix factorization
  Automated way of finding the “dimensions” that matter

  (And more generally, many deep learning-based approaches)
- Language models
  - Can predict user preferences, perhaps better than existing approaches
  - Can also incorporate user requests in natural language

Content-based Recommendations

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

Example:

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

Filling in entries with content-based

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

Filling in entries with content-based

	Avatar	LOTR	Matrix	Pirates
Alice	1	1	0.2	
Bob	.5	0.5		0.3
Carol	0.2	.2	1	
David				0.4

Content-based Approach: Pros and Cons

+: No need for data on other users

Can recommend new items without ratings

+: Able to provide explanations

Can provide explanations of recommended items by listing content-features that caused an item to be recommended

–: Finding the appropriate features is hard

E.g., images, movies, music

–: Recommendations for new users

How to build a user profile?

–: Overspecialization

Can over-concentrate on past user behavior

User-similarity based recommendations

	Avatar	LOTR	Matrix	Pirates
Alice	1	.5	0.2	.3
Bob	1	0.5	.2	0.3
Carol	0.2		1	
David				0.4

Similar idea, now just clump together users

User-similarity based pros and cons

+ Works for any kind of item

- Using information about which users engage with an item

- Cold Start:

- Need enough users in the system to find a match

- First rater:

- Cannot recommend an item that has not been previously rated
- New items, Esoteric items

- Popularity bias:

- Tends to recommend popular items
- Performs poorly for users with unusual tastes

Matrix factorization – “Latent factor” models

- In previous approaches, we assumed we knew how items are related to each other, and how users are related to each other
  - Items are represented by a “vector” of characteristics like genre
  - Users by a “vector” of demographics, location, etc
- In reality, tastes may be complicated and based on subtle preferences unrelated to these things
- Idea: why not *learn* the vectors for each user and item from the history?
  - Learn vector $u_i \in R^d$ for each user, $\mathrm{v_j} \in R^d$ for each item
  - Such that $u_i \cdot v_j \approx \widehat{r_{ij}}$ (the rating user gave to the item in the past)

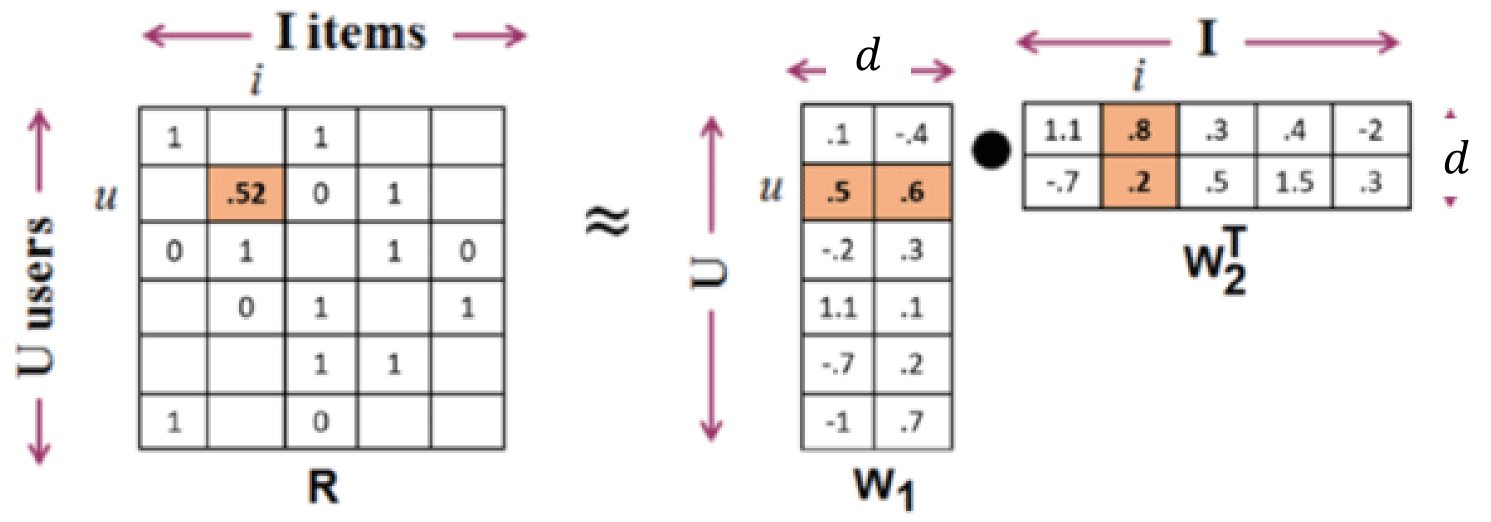
Matrix factorization – “Latent factor” models

Once we have $u_i \in R^d$ for each user, $\mathrm{v_j} \in R^d$ for each item

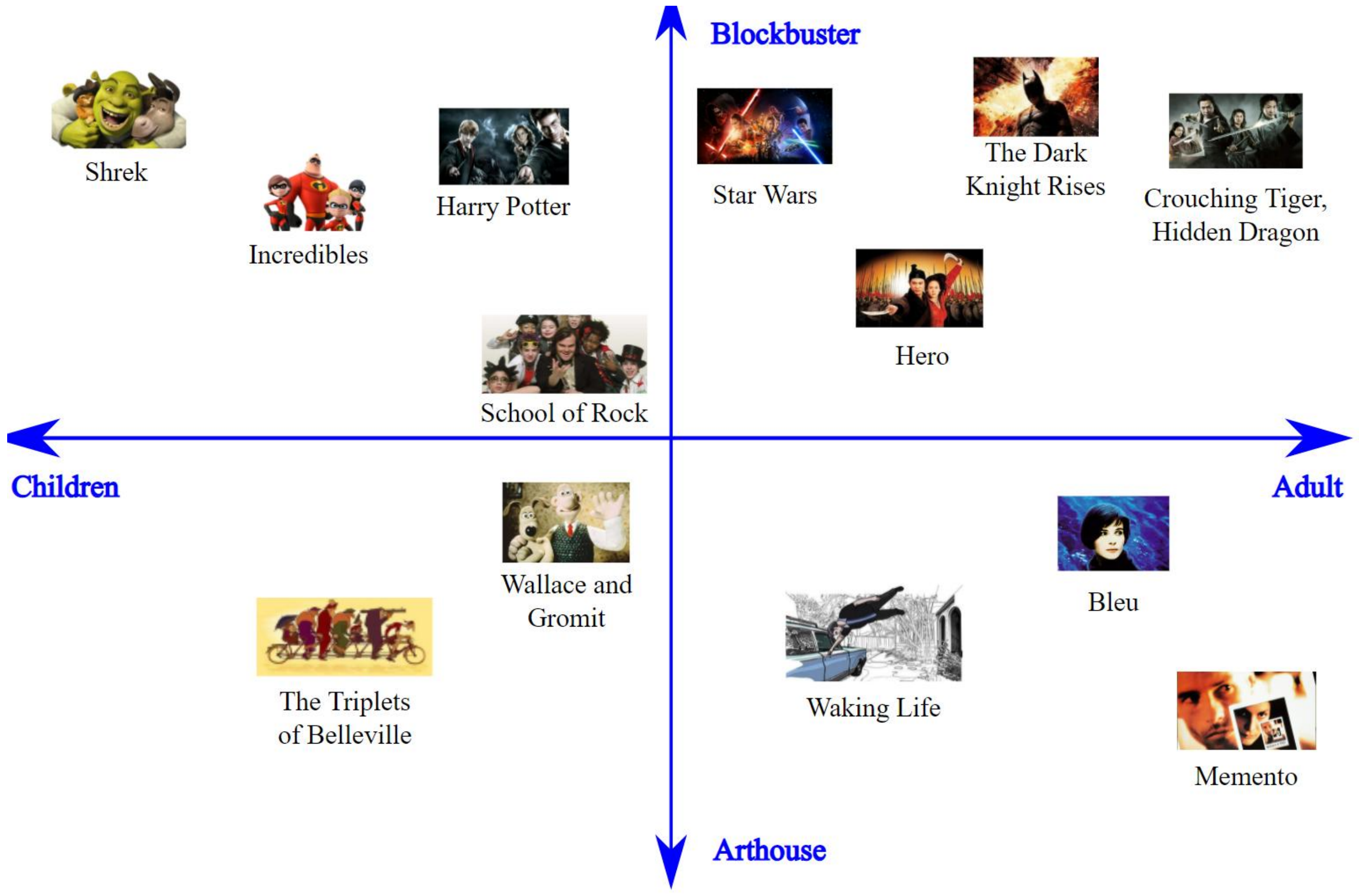
Such that $u_i \cdot v_j \approx \widehat{r_{ij}}$ (the rating user gave to the item in the past)

Then, for every pair of items and users that have not been rated:

Set predicted rating $\mathrm{r_{ij}} = u_i \cdot v_j$



Example vectors with d=2



[Embeddings | Machine Learning Crash Course | Google Developers]

Matrix factorization: Pros and Cons

+: Don’t need to guess at what features matter

–: Need historical data about each item and user

–: Hard to provide explanations

In practice, matrix-factorization-based methods (and modern deep learning successors) are used when you have enough data

“Cold start” with matrix factorization

- Chief challenge in many settings: you don’t have (a lot of) historical data on some new users or new items
    - How do you make recommendations for new users or items?
- Idea: Combine engagement-based approach with content- based approaches

  Step 1: Train matrix factorization model with dataset

  Step 2: For new users [items] find “nearby” users [items] to them and *initialize* their vector using the nearby users [items]

  Step 3: Over time, *update* their vectors using their own history
- Determining “nearby” users/items: must use data like genre and demographics
- Key idea in many settings: At first without individual data, pretend someone is like the “average” user. Then with more data, start doing personalized things
- With language models, perhaps we can also ask new users what they want

Step 2: Vectors from "nearby" users

Suppose we have a demographic vector for each new and old user:

[age, ethnicity, gender, income, …]

- Simple: K nearest neighbors
  - Define a distance function on the vector of demographics
  - For each new user, find the K closest old users and average their vectors
  - Challenge: defining the distance function!
- Also simple: train matrix factorization with known user vector
  - Instead of learning vector $u_i \in R^d$ for each user, $\mathrm{v_j} \in R^d$ for each item
  - Set u_i to the demographic vector, and just learn $\mathrm{v_j} \in R^d$ for each item
- Many other approaches:

  Train a model using the demographics to predict u_i^k , each dimension k of u_i, using all the old users

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 $\mathrm{r_{ij}} = u_i \cdot v_j$

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

Task: going from predictions → recommendations

Questions?