Lecture 16
Recommender Systems
(some slides inspired by Luis von Ahn)

Administration
- Review:
  - PA6 due tonight
  - PA7 goes out later today

2

Recommender Systems
- Tries to algorithmically predict what you'll like
  - Movies
  - Books
  - Music
  - Video games
  - Friends

3

Recommendation Background
- Amazon wants you to buy nice stuff
  - Partially makes up for difficult browsing
- Netflix wants you to like old movies
  - Netflix needs to have at least one copy of many titles, to keep selection up
  - If everyone wants new releases, Netflix needs to buy a lot of copies of one title
  - Much nicer for Netflix if you rent something no one else wants
  - Offered **Netflix Prize**; more later

5

Recommendation Challenges
- Recommendation is common, but surprisingly hard
  - Lots of recommendations to make
  - 10,000s of products
  - Users have very little tolerance for adding preference data; system knows almost nothing about you
  - Everyone is different (right?)

6

Algorithms

- How to predict what movies you like?
  - Features?
- One approach: do it like Web pages
  - Collect data on my movie likes

The Godfather	4
Ernest Goes to Camp	3
Casablanca	2
36 Hours	5
Love and Death	4

  - Collect features: genre, length, year, etc
  - Build score-predictor; recommend high-scorers
- Problems?

7

Collaborative Filtering

- Unfortunately, film-qualities may not be easily extractable
- How to recommend movies without knowing anything about movies?
  - Recommend movies enjoyed by people who are similar to you

8

Demo

	W.	Xanadu	Youngblood	Zorro
Alice	4	2	4	4
Bob	?	2	5	1
Chris	4	2	4	?
Donna	3	?	5	1

9

How can we estimate scores?

- Approach #1: average for film

10

Demo

	W.	Xanadu	Youngblood	Zorro
Alice	4	2	4	4
Bob	3.66	2	5	1
Chris	4	2	4	2
Donna	3	2	5	1

11

How can we estimate scores?

- Approach #1: average for film
- Approach #2: use rating of closest user
- One way to find user-closeness is with an approach similar to tf-idf
  - Consider u and v's vectors of ratings
    - Each movie is a dimension
    - Each score is a weight
  - Compute cosine(u, v), just as with tf-idf info-retrieval doc-ranking
  - What is the equivalent of "inverse doc frequency"?

12

User similarity cont'd

- Another method is *Pearson correlation*
  - S = set of movies
  - $R_{u,i}$ = rating of user u on movie i
  - S_{uv} = {i ∈ S | both *u* & *b* saw *i*}
  - S_u = {i ∈ S | if *u* saw *i*}
  - $r_u = \sum_{i \in S_u} r_{u,i} / |S_u|$

$$\frac{\sum_{i \in S_{uv}} (r_{u,i} - r_u)(r_{v,i} - r_v)}{\sqrt{\sum_{i \in S_{uv}} (r_{u,i} - r_u)^2 \sum_{i \in S_{uv}} (r_{v,i} - r_v)^2}}$$

13

Demo

	W.	Xanadu	Youngblood	Zorro
Alice	4	2	4	4
Bob	?	2	5	1
Chris	4	2	4	?
Donna	3	?	5	1

14

Demo

- Chris' closest match is Alice, so…

	W.	Xanadu	Youngblood	Zorro
Alice	4	2	4	4
Bob	?	2	5	1
Chris	4	2	4	?
Donna	3	?	5	1

15

Demo

- Chris' closest match is Alice, so…

	W.	Xanadu	Youngblood	Zorro
Alice	4	2	4	4
Bob	?	2	5	1
Chris	4	2	4	4
Donna	3	?	5	1

16

Can't Pick Just One

- Can also weight by similarity

$$r_{u,i} = k \sum_{v \in Top-Sim(u)} sim(u,v) r_{v,i}$$

- Where Top-Sim(u) is the n most-similar user neighbors to u
- k is a normalizer

$$k = 1 / \sum_{v \in Top-Sim(u)} |sim(u,v)|$$

17

Can't Pick Just One

- Can also re-add user average scores

$$r_{u,i} = r_u + k \sum_{v \in Top-Sim(u)} sim(u,v) r_{v,i} + (r_{v,i} - r_v)$$

18

Miscellaneous

- We've seen a collaborative filtering algorithm that is:
  - User-based (instead of item-base)
  - Neighborhood-based (top-N neighbors)
- What is the time-complexity of finding nearest-neighbor?
  - With locality-sensitive hashing, linear
  - Remember efficient shingling? The algorithm we saw is example of locality-sensitive hashing
- Many analogies between doc IR and collab. filtering
  - *What would doc d say about term t, even though it is silent on t now?*

Other Algorithms

- Problems so far?
  - What if data is sparse, I.e., a user can only rate a tiny number of products?
  - What if the # of users and products is millions each? Computationally difficult
- One solution: *item-based* filtering
  - Avoid nearest-neighbor operations on users
  - Recommend products similar to ones the user has liked in the past

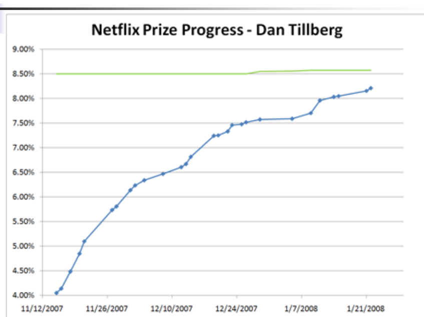
Item-Based Filtering

- For test item *i*, find *k* most-similar items the user has rated previously
  - i's score is a weighted combination of user's ratings on those *k* items
- How can we compute item similarity?
  - Can use cosine or correlation, as before
  - The vector for item i is the set of user-reviews associated with I
  - Previously, a user's profile was a vector of item scores
  - Why is this faster? Item-similarity is more static, can be precomputed. Also….

Model-Based Efficiency

- We can also skimp on the item-item model. New algorithm:
- For each item j, compute k nearest-items, where k << n
- When predicting u's opinion of i, retrieve the k nearest-items for i. Predict score based on the subset of k that i has rated
- Tradeoff between quality and model size!

Netflix Prize!



Netflix Prize!

- Announced October 2, 2006
- 1M$ to whoever could improve NF's own recommender by 10%
- Data from Netflix in the form:
  - <user, movie, date, grade>
  - Where grade is 1..5
  - That's it

Netflix Prize!

- Won on September 21, 2009!

25

Netflix Prize!

- How did they do it? Any ideas?
- Among many ideas:
  - Use IMDB for director, genre, etc
  - Some movies' grades change over time; they "age" well or poorly
    - Use "batch-grade" data to figure out how much time has elapsed since viewing
  - User grades depend on day of week

26