Outlines

- Netflix Challenge
- Introduction
  - terminology
  - long-tail property
  - content-based vs collaborative filtering
  - challenges
  - evaluation
- key recommendation techniques
  - content-based recommendation
  - neighborhood-based methods
  - latent factor models
  - more techniques
- Google’s Adwords problem
References

Recommended for you, Thomas

Literature & Fiction
62 ITEMS

Exercise & Fitness Equipment
6 ITEMS

Health, Fitness & Dieting Books
37 ITEMS

Tableware
12 ITEMS

Prime Video – Unlimited Streaming for Prime Members
19 ITEMS

Coffee, Tea & Espresso
98 ITEMS

Biographies & Memoirs
17 ITEMS

Engineering Books
7 ITEMS
Discover your movie in a few clicks.

Find a perfect match to your mood

Deep Learning Inside
We determine recommendations in real time using cutting-edge technologies. Wanna know how?
Netflix Challenge

Netflix, the world's largest internet-based movie rental company

- October 2, 2006, publicly released a set of data, and offered a Grand Prize of one million US dollars
- goal: produce a system for recommending movies to users based on predicting how much someone is going to like any particular movie

Data: ratings (from 1 star to 5 stars) that users have assigned to movies they have seen

- training set: \( \sim 100 \text{ million ratings}, 480,000 \text{ users}, 18,000 \text{ movies} \) (user-movie rating matrix is \( \sim 99\% \) sparse)
- quiz set: \( \sim 1.5 \text{ million ratings but withheld} \)
- test set: \( \sim 1.5 \text{ million ratings but withheld} \)

evaluation metric: root mean squared error (RMSE)
Netflix Challenge Performance

- overall average: 1.0528 for quiz, 1.0540 for test
- Cinematch: 0.9514 for quiz, 0.9525 for test (∼9.5% improvement)
- grand prize: RMSE 0.8572 (10%), or better, on the test set
- progress prize: 50k best with at least 1% better than previous year

July 26, 2009, the grand prize was won
Introduction

Recommendation systems:

- predict user response to item
- item examples: news article, produce/service ads, movie, ...

Examples:

- offer news articles to online newspaper readers, based on a prediction of reader interests
- offer customers of an online retailer suggestions (products/ads) about what they might like to buy, based on their past history of purchases and/or product searches
Long-tail Property

Long-tail graph shows the distribution of ratings or popularity among items or products in marketplace.

- On the x-column, items are ordered by their popularity or rating frequencies
- y-column shows the popularity in terms of ratings, demand etc

Three important facts:

- popularity
- diversity
- sparsity
Popularity

Products on left side (or in blue area) are called as “popular” because their popularity is higher than green or long-tail area.

- popular products are generally competitive products.

Products in green long-tail area are thought to be “unpopular” or “new products” in market.

The threshold which discriminates popular and unpopular items in market is an hyper-parameter.
Physical store vs Online store

Some researches show that even though popular products mean to be sold a lot, unpopular products or those in long-tail generally returns in better profit.

- physical institutions provide only the most popular items,
- while on-line institutions provide the entire range of items
Diversity

Recommender algorithms are generally designed to give recommendations for popular items because they are popular.

However, a good recommendation system should provide diversity.
  - Same and known items can make the customers bored.

Adjusting the threshold, starting point of long-tail, in recommendation system is an important research to take into account.
  - Moving it right in the graph can increase the diversity in recommendations made.
Sparsity (Missing Values)

Sparsity: Items in the right side of graph are less rated than the those in left side.

- There are much more sparsity or unobserved areas for unpopular items in ratings matrix.

Due to sparsity, a recommender system which relies on neighborhood algorithms may produce bad results.

- The more we move the threshold to right side, The worse recommendation system results.

Sparsity and long-tail are 2 important properties of a recommender system to take into account in design and process.
### Lecture 18: Recommender Systems

#### User-Item Rating Matrix ($R$)

<table>
<thead>
<tr>
<th>User 1</th>
<th>User 2</th>
<th>User 3</th>
<th>User 4</th>
<th>User 5</th>
<th>User 6</th>
<th>User 7</th>
<th>User 8</th>
<th>User 9</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

#### User-Feature Matrix ($U$)

<table>
<thead>
<tr>
<th>User 1</th>
<th>User 2</th>
<th>User 3</th>
<th>User 4</th>
<th>User 5</th>
<th>User 6</th>
<th>User 7</th>
<th>User 8</th>
</tr>
</thead>
<tbody>
<tr>
<td>UF1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UF2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Item-Feature Matrix ($V$)

<table>
<thead>
<tr>
<th>Item 1</th>
<th>Item 2</th>
<th>Item 3</th>
<th>Item 4</th>
<th>Item 5</th>
<th>Item 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IF2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Different Similarity Measures

- **content-based system**: measures similarity by looking for common features of items/users; e.g., if a Netflix user has watched many cowboy movies, then recommend a movie classified in the database as having the cowboy genre.

- **collaborative filtering**: measure similarity of users by their item preferences; measure similarity of items by the users who like them (user-item interaction)

- **hybrid system**: uses both types of information
Challenges

user-item interactions are extremely sparse

cold start problem
  - for content-based system, user has to dedicate an amount of effort using the system, so to construct their user profile, before the system can start providing any recommendation (fail for new user)
  - for collaborative filtering, it would fail to consider it would fail to consider items which no one has rated previously (fail for new item)
Evaluation of Recommender Systems

Accuracy of the predicted scores:
- root mean squared error (RMSE)
- mean absolute error

Accuracy of the recommended list:
- precision and recall:

\[
\text{precision}(L) = \frac{1}{N_{\text{user}}} \sum_u \frac{|L(u) \cap T(u)|}{|L(u)|}
\]

\[
\text{recall}(L) = \frac{1}{N_{\text{user}}} \sum_u \frac{|L(u) \cap T(u)|}{|T(u)|}
\]

- average precision for a given user \( u \):

\[
\sum_{k=1,\ldots,K} \text{precision}(k)
\]

where \( \text{precision}(k) \) is the precision at cut-off \( k \), i.e., the ratio of number of clicked items up to the position \( k \) over the number \( k \).
Key recommendation techniques

- content-based recommendation
- neighborhood-based methods
- latent factor models
- more techniques
**Content-based recommendation**

- Construction of feature profile:
  - Item profile: movie features (genre, release date, cast); news article features (topic, word frequencies); image tags
  - User profile: browsing history; demographical information

- Recommend items for a given user:
  - Recommend similar items based on item profiles
  - Build a decision / classification rule given item features as covariates
  - Would fail for new users
Neighborhood-based Recommendation

Key Idea:
- recommend similar items, or items of similar users
- simple, efficient, stable

Key components:
- similarity measure / weight: Pearson correlation or other measures; based on user and item profiles
- neighborhood selection: to address data sparsity, can cluster users and/or items into small groups with strong similarity first
Item-based recommendation

Predict user $u$’s rating of item $i$, $r_{ui}$, as

$$
\hat{r}_{ui} = \bar{r}_i + \frac{\sum_{j \in N_u(i)} w_{ij} (r_{uj} - \bar{r}_j)}{\sum_{j \in N_u(i)} w_{ij}},
$$

- for user $u$, the rating for item $i$ is the weighted average of the same user’s ratings on similar items $j \in N_u(i)$
- $\bar{r}_i$ is the average rating of all users have given to item $i$: mean-centering normalization.
User-based recommendation

Predict user $u$’s rating of item $i$, $r_{ui}$, as

$$
\hat{r}_{ui} = \bar{r}_u + \frac{\sum_{v \in N_i(u)} w_{uv} (r_{vi} - \bar{r}_v)}{\sum_{v \in N_i(u)} w_{uv}},
$$

- for item $i$, the rating by user $u$ is the weighted average of the ratings from similar users $v \in N_i(u)$
- $\bar{r}_u$ is the average rating of all items that user $i$ has given: mean-centering normalization.
Latent Factor Models

also known as matrix factorization

key idea: map both users and items to a joint latent factor space of dimensionality $k$, such that user-item interactions are modeled as inner products in that space

each item $i$ is associated with a vector $q_i \in \mathbb{R}^k$, and each user $u$ is associated with a vector $p_u \in \mathbb{R}^k$.

$q_i^T p_u$ captures the interaction between user $u$ and item $i$, i.e., the overall interest of the user in characteristics of the item

model only the observed ratings, avoid over-fitting via regularization.
Latent Factor Model Estimation

Model: for a known link function $g$,

$$g(E(r_{ui})) = b_0 + b_i + b_u + q_i^T p_u$$

Estimation: take $g =$ identity as an example, minimize

$$\sum_{\text{observed}} (r_{ui} - b_0 - b_i - b_u - q_i^T p_u)^2 + \lambda(b_i^2 + b_u^2 + \|q_i\|^2 + \|p_u\|^2).$$
Latent factor models incorporating user/item features:

Key Idea:
- model item/user latent factors based on item/user features
- use a Bayesian framework for computation

Model setup

\[ r_{ui} \sim \text{Normal}(\mu_{ui}, \sigma^2) \]
\[ r_{ui} \sim \text{Bernoulli}(\mu_{ui}) \]
\[ g(r_{ui}) = b_0 + b_i + b_u + q_i^T p_u \]
\[ b_i = \alpha^T z_i + \epsilon_{ib}, \quad \epsilon_{ib} \sim N(0, \sigma_i^2) \]
\[ b_u = \beta^T x_u + \epsilon_{ub}, \quad \epsilon_{ub} \sim N(0, \sigma_u^2) \]
\[ q_i = \phi z_i + \epsilon_{iq}, \quad \epsilon_{iq} \sim \text{MVN}(0, \Sigma_i) \]
\[ p_u = \psi x_u + \epsilon_{up}, \quad \epsilon_{up} \sim \text{MVN}(0, \Sigma_u) \]
Adwords Problem

Adwords Problem:

- a fundamental problem of search advertising (about 24 billions in 2012 for US market only)
- we term the “adwords problem,” because it was first encountered in the Google Adwords system
History of Search Advertising

Around 2000, a company called Overture (later bought by Yahoo!) introduced a new (revolutionary) kind of search - advertisers bid on keywords (words in a search query),

- when a user searched for that keyword, the links to all the advertisers who bid on that keyword are displayed in the order highest-bid-first;
- if the advertisers link was clicked on, they paid what they had bid

useful for when the search queryer was looking for advertisements, but rather useless if someone was just looking for information
Google’s Adwords System

Years later, Google adapted the idea in a system called “Adwords”

- by that time, the reliability of Google was well established, so people were willing to trust the ads they were shown
- Google kept the list of responses based on PageRank (or other criteria) separate from the list of ads, so the same system was useful for both types of users who wanted information or looked to buy something
Further Refinement/Complications

- show only a limited number of ads with each query - Google had to decide both which ads to show and in what order
- users of the Adwords system specified a budget: the amount they were willing to pay for all clicks on their ads in a month
- Google did not simply order ads by the amount of the bid, but by the amount they expect to receive for display of the ad - the value of an ad was taken to be the product of the bid and the click-through rate
- the decision regarding which ads to show must be made on-line
Adwords Problem: Input

- a set of bids by advertisers for search queries
- a click-through rate for each advertiser-query pair
- a budget for each advertiser (for a month, or other time length)
- a limit on the number of ads to be displayed with each search query
Adwords Problem: Output

respond to each search query with a set of advertisers s.t.

- the size of the set is no larger than the limit on the number of ads per query
- each advertiser has bid on the search query
- each advertiser has enough budget left to pay for the ad if it is clicked upon

greedy algorithm, balance algorithm, implementation