Recommending Products

Emily Fox & Carlos Guestrin
Machine Learning Specialization
University of Washington
Where we see recommender systems
Personalization is transforming our experience of the world

Information overload

Browsing is “history”
  - Need new ways to discover content

Personalization: Connects users & items

viewers  videos

100 Hours a Minute
What do I care about?
Movie recommendations

Connect users with movies they may want to watch
Product recommendations

Recommendations combine global & session interests
Music recommendations

Recommendations form coherent & diverse sequence
Friend recommendations

Users and “items” are of the same “type”
Drug-target interactions

What drug should we “repurpose” for some disease?

Cobanoglu et al. ’13
Building a recommender system
Solution 0: Popularity
Simplest approach: Popularity

- What are people viewing now?
  - Rank by global popularity

- Limitation:
  - No personalization
Solution 1: Classification model
What’s the probability I’ll buy this product?

User info
Purchase history
Product info
Other info

Classifier

Yes!
No

• Pros:
  - **Personalized:**
    Considers user info & purchase history
  - **Features can capture context:**
    Time of the day, what I just saw,...
  - **Even handles limited user history:** Age of user, ...
Limitations of classification approach

- Features may not be available
- Often doesn’t perform as well as collaborative filtering methods (next)
Solution 2: People who bought this also bought...
Co-occurrence matrix

• People who bought *diapers* also bought *baby wipes*

• **Matrix C:**
  store # users who bought both items *i* & *j*
  – (# items x # items) matrix

  – **Symmetric:** # purchasing *i* & *j* same as # for *j* & *i*  \( C_{ij} = C_{ji} \)
Making recommendations using co-occurences

• User purchased *diapers*

1. Look at *diapers* row of matrix

2. Recommend other items with largest counts
   - *baby wipes, milk, baby food,*...
Co-occurrence matrix must be normalized

• What if there are very popular items?
  – Popular baby item: *Pampers Swaddlers diapers*
  – For any baby item (e.g., $i=\text{Sophie giraffe}$)
  large count $C_{ij}$ for $j=\text{Pampers Swaddlers}$

• Result:
  – Drowns out other effects
  – Recommend based on popularity
Normalize co-occurrences: Similarity matrix

• **Jaccard similarity**: normalizes by popularity
  - Who purchased *i and j divided by* who purchased *i or j*

• Many other similarity metrics possible, e.g., cosine similarity
Limitations

• Only current page matters, no history
  – Recommend similar items to the one you bought

• What if you purchased many items?
  – Want recommendations based on purchase history
(Weighted) Average of purchased items

• User \( \text{bought items \{diapers, milk\}} \)
  – Compute user-specific score for each item \( j \) in inventory by combining similarities:

\[
\text{Score}(, \text{baby wipes}) = \frac{1}{2} (S_{\text{baby wipes, diapers}} + S_{\text{baby wipes, milk}})
\]

  – Could also weight recent purchases more

• Sort \( \text{Score}(, j) \) and find item \( j \) with highest similarity
Limitations

• Does not utilize:
  - context (e.g., time of day)
  - user features (e.g., age)
  - product features (e.g., baby vs. electronics)

• Cold start problem
  - What if a new user or product arrives?
Solution 3: Discovering hidden structure by matrix factorization
Movie recommendation
• Users watch movies and rate them

<table>
<thead>
<tr>
<th>User</th>
<th>Movie</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>⭐⭐⭐⭐⭐</td>
</tr>
<tr>
<td></td>
<td></td>
<td>⭐⭐⭐⭐⭐</td>
</tr>
<tr>
<td></td>
<td></td>
<td>⭐⭐⭐⭐⭐</td>
</tr>
<tr>
<td></td>
<td></td>
<td>⭐⭐⭐⭐⭐</td>
</tr>
<tr>
<td></td>
<td></td>
<td>⭐⭐⭐⭐⭐</td>
</tr>
<tr>
<td></td>
<td></td>
<td>⭐⭐⭐⭐⭐</td>
</tr>
<tr>
<td></td>
<td></td>
<td>⭐⭐⭐⭐⭐</td>
</tr>
<tr>
<td></td>
<td></td>
<td>⭐⭐⭐⭐⭐</td>
</tr>
</tbody>
</table>

Each user only watches a few of the available movies
Matrix completion problem

- **Data:** Users score some movies
  - $\text{Rating}(u,v)$ known for black cells
  - $\text{Rating}(u,v)$ unknown for white cells
- **Goal:** Filling missing data?
Suppose we had $d$ topics for each user and movie

- Describe movie $v$ with topics $R_v$
  - How much is it action, romance, drama, ...

\[ R_v = \begin{bmatrix} 0.3 & 0.6 & 1.5 & \ldots \end{bmatrix} \]

- Describe user $u$ with topics $L_u$
  - How much she likes action, romance, drama, ...

\[ L_u = \begin{bmatrix} 2.5 & 0 & 0.8 & \ldots \end{bmatrix} \]

- Rating($u, v$) is the product of the two vectors

\[ \text{Rating}(u, v) = R_v^T L_u = 0.3 \times 2.5 + 0 + 1.5 \times 0.8 + \ldots = 7.2 \]

\[ \text{Rating}(u, v) = L_u^T R_w = 2.5 \times 0.3 + 0 + 0.8 \times 0.1 + \ldots = 0.8 \]

- Recommendations: sort movies user hasn’t watched by Rating($u, v$)
Predictions in matrix form

\[
\text{Rating} = X_{ij}
\]

Rows index movies
Columns index users

But we don’t know topics of users and movies…
Matrix factorization model: Discovering topics from data

- Only use observed values to estimate “topic” vectors $\hat{L}_u$ and $\hat{R}_v$
- Use estimated $\hat{L}_u$ and $\hat{R}_v$ for recommendations

$\text{Rating} = \approx \begin{bmatrix} L \end{bmatrix} \begin{bmatrix} R' \end{bmatrix}$

$\text{RSS}(L, R) = \sum \left( \text{Rating}(u,v) - \langle \hat{L}_u, \hat{R}_v \rangle \right)^2$

- Many efficient algorithms for factorization
Limitations of matrix factorization

• Cold-start problem
  – This model still cannot handle a new user or movie
Bringing it all together:
Featurized matrix factorization
Combining features and discovered topics

• Features capture **context**
  – *Time of day, what I just saw, user info, past purchases,* …

• Discovered topics from matrix factorization capture **groups of users** who behave similarly
  – *Women from Seattle who teach and have a baby*

• **Combine** to mitigate cold-start problem
  – Ratings for a new user from **features** only
  – As more information about user is discovered, matrix factorization **topics** become more relevant
Blending models

• Squeezing last bit of accuracy by blending models

• Netflix Prize 2006-2009
  – 100M ratings
  – 17,770 movies
  – 480,189 users
  – Predict 3 million ratings to highest accuracy
  – Winning team blended over 100 models
A performance metric for recommender systems
The world of all baby products
User likes subset of items
Why not use classification accuracy?

• Classification accuracy = fraction of items correctly classified (liked vs. not liked)

• Here, not interested in what a person does not like

• Rather, how quickly can we discover the relatively few liked items?
  – (Partially) an imbalanced class problem
How many liked items were recommended?

Recall

\[ \frac{\# \text{ liked} \& \text{shown}}{\# \text{ liked}} = \frac{3}{5} \]
How many recommended items were liked?

Precision
\[
\frac{\text{# liked & shown}}{\text{# shown}} = \frac{3}{11}
\]
Maximize recall: Recommend everything
Resulting precision?
Optimal recommender

Recall = 1
Precision = 1
Precision-recall curve

- Input: A specific recommender system
- Output: Algorithm-specific precision-recall curve

To draw curve, vary threshold on # items recommended
- For each setting, calculate the precision and recall

©2015 Emily Fox & Carlos Guestrin
Which Algorithm is Best?

• For a given precision, want recall as large as possible (or vice versa)
• One metric: largest area under the curve (AUC)
• Another: set desired recall and maximize precision (precision at k)
Summary of recommender systems
What you can do now...

• Describe the goal of a recommender system
• Provide examples of applications where recommender systems are useful
• Implement a co-occurrence based recommender system
• Describe the input (observations, number of “topics”) and output (“topic” vectors, predicted values) of a matrix factorization model
• Exploit estimated “topic” vectors (algorithms to come...) to make recommendations
• Describe the cold-start problem and ways to handle it (e.g., incorporating features)
• Analyze performance of various recommender systems in terms of precision and recall
• Use AUC or precision-at-k to select amongst candidate algorithms