Recommender Systems

Sewoong Oh

CSE/STAT 416
University of Washington
Personalization is a successful use of learning from data

- Facebook advertisements from browsing history
- Amazon, YouTube, Netflix recommendations from user choices
- Input: user preferences (or activities)
- Goal: select (a small set of) items the user will like

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>User 1</td>
<td>5</td>
<td></td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>User 2</td>
<td></td>
<td>2</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>User 3</td>
<td></td>
<td></td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>User 4</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>User 5</td>
<td></td>
<td></td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>User 6</td>
<td></td>
<td>5</td>
<td></td>
<td>2</td>
</tr>
</tbody>
</table>

- Challenge: sparsity
- Key idea: collaborative filtering
  a user might like something, if similar users liked it
Challenges in recommender systems

• Some feedback are implicit
  • explicit feedback: rating, purchase history, ranking
  • Implicit feedback: browsing history, TV viewing pattern
    • Implicit feedback requires pre-processing of data such as time spent, clicked, interval, etc.

• We seek diversity which is not easy to impose because users are multifaceted
  • A person with Linear Algebra textbook does not need another one, but Top-k recommendation might stick to k linear algebra textbooks
  • We don’t want to recommend just Marvel movies

• Cold-start is hard
  • Recommendations for new user/movie with no data is hard
  • Need to use additional features/contexts (Netflix 20 questions)
Challenges in recommender systems

• Interests change over time, but dynamic models are hard to train
  • Users preferences change over time
  • Movies perception changes over time

• Given millions of users and hundreds of thousands of movies, we need a scalable (i.e. fast) algorithm
  • We need to exploit that data is sparse
Approach 0: popularity

- No personalization
- Netflix: trending now (average number of viewers)
- NY times: popular article (average views)

Approach 1: classifier

- Train a classifier on 
  \[ x = (\text{user features and movie features}) \]
  \[ y = \text{liked} (+1) \text{ or not} (-1) \]
- Output: +1 (recommend) or -1 (do not recommend)

Pros
- personalized
- flexible to include additional features like time

Cons
- Useful features are hard to get
- Empirical performance not as good as Collaborative Filtering
Approach 2: co-occurrences

• “People who bought X also bought …”

• Construct a normalized co-occurrence matrix $C$ where both rows and columns indicate items

\[
C_{ij} = \frac{\text{# of people who bought } i \text{ and } j}{\text{# of people who bought } i \text{ or } j}
\]

• This is a symmetric matrix: $C_{ij} = C_{ji}$

• For a user who bought \{milk, diapers\}

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

• If user bought similar items, then give more score for “baby wipes”
Approach 3: matrix factorization

- Movie recommendations
- Users watch movies and give ratings
- But each user only rates a few

### Input Data

<table>
<thead>
<tr>
<th>User</th>
<th>Movie</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>User 1</td>
<td>5</td>
<td>⭐⭐⭐⭐⭐</td>
</tr>
<tr>
<td>User 2</td>
<td>2</td>
<td>⭐⭐⭐⭐ ⭐</td>
</tr>
<tr>
<td>User 3</td>
<td>3</td>
<td>⭐⭐⭐⭐⭐</td>
</tr>
<tr>
<td>User 4</td>
<td>1</td>
<td>⭐⭐⭐⭐⭐</td>
</tr>
<tr>
<td>User 5</td>
<td>4</td>
<td>⭐⭐⭐⭐⭐</td>
</tr>
<tr>
<td>User 6</td>
<td>5</td>
<td>⭐⭐⭐⭐⭐</td>
</tr>
</tbody>
</table>

Input Data in a matrix form

<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>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>3</td>
<td>1</td>
<td>4</td>
<td>2</td>
</tr>
</tbody>
</table>

### Input Data

X

Rows index movies
Columns index users

\[ X_{ij} \]

known for black cells
unknown for white cells
Matrix completion problem

- Black cells indicate Rating(user,movie) known
- White cells indicate Rating(user,movie) unknown
- Each cell has values in {1,2,3,4,5}
- Goal: predict missing entries
Premise: Suppose we have $d$ types of movies

- We can describe each movie $v$ with feature vector $R_v$
  - How much is the movie action, romance, drama, …
  - $R_v = [0.3, \ 0.01, \ 1.5, \ … ]$
- We can describe a user $u$ with feature vector $L_u$
  - How much she likes action, romance, drama, …
  - $L_u = [2.3, \ 0 , \ 0.7, \ … ]$
- Perhaps we can find such features that the rating can be predicted as the **inner product** of those two vectors
  - Rating$(u,v) = 0.3*2.3 + 0.01*0 + 1.5*0.7 + …$
- This allows you to predict how a user will rate a movie, that she has not seen yet
Product recommendations

• Suppose the following features have been learned, which movie should we recommend to user #3?

<table>
<thead>
<tr>
<th>User ID</th>
<th>Feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(2, 0)</td>
</tr>
<tr>
<td>2</td>
<td>(1, 1)</td>
</tr>
<tr>
<td>3</td>
<td>(0, 1)</td>
</tr>
<tr>
<td>4</td>
<td>(2, 1)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Movie ID</th>
<th>Feature vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(3, 1)</td>
</tr>
<tr>
<td>2</td>
<td>(1, 2)</td>
</tr>
<tr>
<td>3</td>
<td>(2, 1)</td>
</tr>
</tbody>
</table>

Call this 4x2 matrix $L$

Call this 3x2 matrix $R$

• Such prediction can be computed for all (user,movie) pairs
• And be written in a matrix form:

$$ L = \begin{bmatrix} 6 & 2 & 4 \\ 4 & 3 & 3 \\ 1 & 2 & 1 \\ 7 & 4 & 5 \end{bmatrix} \quad \begin{bmatrix} 2 & 0 \\ 1 & 1 \\ 0 & 1 \\ 2 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 0 \\ 3 & 1 \\ 1 & 2 \\ 2 & 1 \end{bmatrix} = R^T $$
Predictions in a matrix form

- Ratings matrix is the product of L and R: user feature matrix and movie feature matrix
- How do we learn the feature matrices from data?
- When we have all the ratings, then it is easy
  - PCA gives optimal factorization $L$ and $R$ in terms of reconstruction error
- This automatically discovers the right topics from data
- But, if we have all the ratings, we don’t need to predict anything

\[ \text{Rating} = \begin{bmatrix} \text{Rows index movies} \\ \text{Columns index users} \end{bmatrix} \approx \begin{bmatrix} L \\ R' \end{bmatrix} \]
Matrix factorizations are not unique

- Let’s say we have an exact factorization $M = L^*R^T$

- There are infinitely many factorizations, which give the exactly same $M$
- For example, we can scale up the user features and scale down the movie ones, so that the ratings do not change

- Precisely, for any invertible matrix $Q$, $(LQ, RQ^T)$ give the same matrix as $(L, R)$ since
  \[ LQ^*(RQ^T)^T = LQ^*Q^{-1}R = LR = M \]
From factorization to Matrix completion

- In reality, we only have partial observations of the ratings matrix
- We fit the best $L$ and $R$, to the observed ratings
- There has been many efficient algorithms to find factorization based on partial observations, a.k.a. matrix completion problem

- We suppose there are $m$ movies and $n$ users, and $k$ topics, and the ground truth matrix $M$ is generated by $M = L_0 \cdot R_0^T$ for the form above

- No matter how many entries I observed, there are multiple choices of parameters $(L, R)$ that will match all the entries
  - because, if $(L, R)$ matches the entries, so does $(LQ, RQ^{-T})$
But when can we solve this problem?

That is how many entries do we need to see, in order for our prediction to be accurate?

One extreme: suppose we observe all entries, then

- Any factorization methods like singular value decomposition (SVD) will provide (one of the) correct factorizations
- And, this correct, i.e. resulting $M = LR^T$

Another extreme: suppose we observe one entry, then

- It is easy to match the entry observed
- But, most likely this is incorrect on the missing entries, i.e. $M \neq LR^T$
From factorization to Matrix completion

\[
\text{Rating} = \begin{array}{c}
\begin{array}{c}
\text{m} \\
\text{n}
\end{array}
\end{array} \approx \begin{array}{c}
\begin{array}{c}
L \\
k
\end{array}
\end{array} \begin{array}{c}
\begin{array}{c}
R' \\
k
\end{array}
\end{array}
\]

- If there are \(m\) movies and \(n\) users, and \(k\) topics, then how many parameters do we have in our factorization model \(L\) and \(R\)?
  
  degrees-of-freedom = \(k*m + k*n\)

- This is also sometimes called the degrees-of-freedom in the problem.

- How many entries do we observe if we have the full matrix?

- How many entries do you think we need, to accurately reconstruct the ground truth \(L\) and \(R\) that generated the data?
Algorithmic solution for matrix completion

- How do we write a program to find \((L, R)\) matching the observed entries?
- Machine learning approach:
  - Write a loss function and minimize

\[
\min_{L, R} \sum_{u,v : r_{uv} \neq ?} \left( (LR^T)_{uv} - r_{uv} \right)^2
\]

- Coordinate descent is a popular method in solving this optimization
Coordinate descent

- Consider a optimization problem (in 2-dimensions for illustration purposes)
  \[ \text{minimize}_{w_0, w_1} \ g(w_0, w_1) \]
  
- One method is called \textit{coordinate descent}
  
- Initialize \((w_0, w_1)\) to be random or smart initialization
  
- While not converged, repeat
  - Pick a coordinate \(j\) in \(\{0, 1\}\) (either random, round-robin, etc.)
    \[ w_j \leftarrow \arg \min_{w_j} \ g(w_0, w_1) \]

- Main idea:
  - Minimizing over 1 coordinate is much easier
  - No need to choose step-size
  - This is guaranteed to find optimal solution, under some constraints
  - When does it fail?
Coordinate descent

- Coordinate descent successfully finds the optimal solution if $g(.)$ is strongly convex and smooth
Coordinate descent for matrix completion

\[
\min_{L,R} \sum_{u,v: r_{uv} \neq ?} \left( (LR^T)_{uv} - r_{uv} \right)^2
\]

- Initialize \((L,R)\)
- Repeat
  - Fix \(R\) and optimize over \(L\)
  - Fix \(L\) and optimize over \(R\)

- First insight:

\[
\min_{L_1, \ldots, L_n} \sum_{(u,v): r_{uv} \neq ?} (L_u^T R_v - r_{uv})^2
\]

\[
= \min_{L_1, \ldots, L_n} \sum_{u=1}^n \left\{ \sum_{v: r_{uv} \neq ?} (L_u^T R_v - r_{uv})^2 \right\}
\]

\[
= \sum_{u=1}^n \left\{ \min_{L_u} \sum_{v: r_{uv} \neq ?} (L_u^T R_v - r_{uv})^2 \right\}
\]

This only involves each row of \(L\) independently, and can be solved as a separate optimization for each row.
Coordinate descent for matrix completion

• We broke down the problem into solving multiple inner optimizations of the form:

\[
\min_{L_u} \sum_{v: r_{uv} \neq ?} (L_u^T R_v - r_{uv})^2
\]

• Second insight:
  • And this is the standard linear regression with quadratic loss
  • Many efficient solvers exist + can be solved in a closed form too
Example: $2000 \times 2000$ rank-8 random matrix

- **low-rank matrix** $M$
- **sampled matrix** $M^E$
- **OptSpace output** $\hat{M}$
- **squared error** $(M - \hat{M})^2$

0.25% sampled
Example: $2000 \times 2000$ rank-8 random matrix

low-rank matrix $M$

sampled matrix $M^E$

$\text{OptSpace}$ output $\hat{M}$

squared error $(M - \hat{M})^2$

0.50% sampled
Example: $2000 \times 2000$ rank-8 random matrix

- low-rank matrix $M$
- sampled matrix $M^E$
- \texttt{OptSpace} output $\hat{M}$
- squared error $(M - \hat{M})^2$

0.75% sampled
Example: 2000 × 2000 rank-8 random matrix

- **low-rank matrix** $M$
- **sampled matrix** $M^E$
- **OptSpace output** $\hat{M}$
- **squared error** $(M - \hat{M})^2$

1.00% sampled
Example: $2000 \times 2000$ rank-8 random matrix

low-rank matrix $M$

sampled matrix $M^E$

\texttt{OptSpace} output $\hat{M}$

squared error $(M - \hat{M})^2$

1.25% sampled
Example: $2000 \times 2000$ rank-8 random matrix

low-rank matrix $M$

sampled matrix $M^E$

OptSpace output $\hat{M}$

squared error $(M - \hat{M})^2$

1.50% sampled
Example: $2000 \times 2000$ rank-8 random matrix

low-rank matrix $M$

sampled matrix $M^E$

$\textsf{OptSpace}$ output $\hat{M}$

squared error $(M - \hat{M})^2$

1.75% sampled
Application: localization

- Wireless sensors deployed in a region
- Each measure distance to the close-by sensors
- Goal: find the distances to all sensors
  - If we have all pairwise distances, then it is easy to find locations of all sensors simultaneously
Application: localization

- Why is this a **Matrix Completion problem**?
  - We have missing entries
  - The data is in a matrix form
  - But most importantly, the ground-truth is a low-rank matrix
    - The ambient dimension is 2 or 3, i.e. position is \((x_u, y_u)\)
    - \[D_{uv} = (x_u - x_v)^2 + (y_u - y_v)^2\]

\[
D = \begin{pmatrix}
1 & \sqrt{x_u^2 + y_u^2 - \sqrt{2}x_u - \sqrt{2}y_u} \\
\sqrt{2}x_v & 1 \\
\sqrt{2}y_v & \sqrt{2}x_v \\
\end{pmatrix}
\]

\[
R^T = \begin{pmatrix}
x_u^2 + y_u^2 \\
1 \\
\sqrt{2}x_v \\
\sqrt{2}y_v
\end{pmatrix}
\]
Application: recommendation systems

- Given partially observed ratings matrix
- Discover $k$ topics, and $k$-dimensional user features $L_u$
- Movie features $R_v$
- Predict how much a user will like a movie by $r_{uv} = L_u^T R_v$
- Make recommendations based on the prediction

- Applied to Wikipedia
Which is correct about matrix factorization based recommendation systems?

- a) provide personalization
- b) capture context (e.g. time of the day)

Another weakness of matrix factorization

- We need to know \( k \), in some sense
- If we set \( k = \min\{m,n\} \), what goes wrong?
  - overfitting

\[
\begin{bmatrix}
6 & 4 \\
4 & 3 \\
2 & \\
7 & 5 \\
\end{bmatrix}
\begin{bmatrix}
L \\
\end{bmatrix}
\begin{bmatrix}
R^T \\
\end{bmatrix}
\]

Solution: regularize

\[
\min_{L_u} \sum_{v: r_{uv} \neq ?} (L_u^T R_v - r_{uv})^2 + \lambda \|L_u\|^2
\]
**Featured** matrix factorization

- Limitations of matrix factorization
  - Cold-start problem
    - This model still cannot handle a new user or movie

\[
\text{Rating} = \begin{array}{c}
\begin{array}{c}
\text{Rows index movies} \\
\text{Columns index users}
\end{array}
\end{array}
\]

- As there is no observation for the entire row/column putting anything in that row has no penalty

\[
\begin{align*}
\text{minimize}_{L,R} & \sum_{u,v: r_{uv} \neq ?} \left( (L R^T)_{uv} - r_{uv} \right)^2 \\
& L^T_u R_v
\end{align*}
\]
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
Collaborative filtering with specified features

- Create feature vector for each movie (often have this even for new movies):
  \[ \phi(v) = (\text{genre}, \text{year}, \text{director}, \ldots) \]

- Define weights on these features for how much all users like each feature:
  \[ w = \text{vector of same length} \]

- Fit linear model:
  \[ \text{For all users, } r_{uv} \approx w \cdot \phi(v) \]
  \[ \text{standard regression model} \]

- Minimize:
  \[ \min_w \sum_{v \in \text{movies}} (w \cdot \phi(v) - r_{uv})^2 + \lambda \|w\|_1 \|w\| \text{ (LS, Lasso, Ridge)} \]
Building in personalization

- Of course, users do not have identical preferences
- Include a user-specific deviation from the global set of user weights:
  \[ r_{uv} = (\mathbf{w} + \mathbf{w}_u) \cdot \phi(v) \]
- If we don’t have any observations about a user, use wisdom of the crowd
  \[ \text{Initiate } \mathbf{w}_u = 0 \Rightarrow r_{uv} \approx \mathbf{w} \cdot \phi(v) \]
- As we gain more information about the user, forget the crowd
  \[ \text{\textbf{W}_u more informed (personalization)} \]
- Can add in user-specific features, and cross-features, too
  \[ \phi(u) = \text{age, gender, education} \]
  \[ \phi(u, v) = \text{... } \phi(u) \text{...}, \text{... } \phi(v) \text{...}, \text{cross features} \]
Featurized matrix factorization—
A combined approach

**Feature-based approach:**
- Feature representation of user and movies fixed
- Can address cold-start problem

**Matrix factorization approach:**
- Suffers from cold-start problem
- User & movie features are learned from data

**A unified model:**

\[ r_{uv} = Lu \cdot R_v + (w + w_u) \cdot \phi(v, v) \]

Solve via coord. desc., grad. desc., etc.