Recommendation as Search/Information Retrieval

Example: Recommending News Articles

- Dataset: https://www.kaggle.com/snapcrack/all-the-news/home
- Fields: id, title, publication name, author, date, year, month, url, content
- We’ll use first 1,000 articles from articles1.csv (which contains 50,000 articles)
Example: Recommending News Articles

- Remove stop words, use stemming
- Bag of words model
- Use 500 most frequent terms (to speed things up)
- Given an article, find $k$ nearest neighbours and Euclidean distance
- E.g. article “House Republicans Fret About Winning Their Health Care Suit’:
  - House Clears Path for Repeal of Health Law 0.82
  - Trump Follows Obamas Lead in Flexing Executive Muscle 0.83
  - Senate Republicans Open Fight Over Obama Health Law 0.85
  - G.O.P. Campaign to Repeal Obamacare Stalls on the Details 0.85
  - Turmoil Overshadows First Day of Republican-Controlled Congress 0.86
Example: Recommending Shopping

- Feature vector of length $N$, the number of different items stocked
- For "shopping basket" set feature vector entry = 1 if item in basket
- To make recommendations use $k$NN, then rank items by popularity
Collaborative Filtering Recommenders

- Search-based recommendations cheap and effective but have no personalisation, no surprises/exploration
- Main idea:
  - Store history (articles read, items bought, songs listened to etc) for each user, plus ratings if available
  - To make recommendations for a user find similar users and use their histories to form recommendations
- E.g. use $k$NN again to find nearest users, then recommend most popular items from the set of nearest users (weighted by user ratings of items, if available)
- Two major problems:
  - **Sparse Data**: User history data is usually very sparse (most users only rate a few items e.g. 10 items out of a possible 100k items) → hard to reliably find neighbouring users unless have a lot of data
  - **Cold Start**: New users have no history, new items are not in any users history
Collaborative Filtering As Matrix Completion

Example: users rate books they have read from 0-5.

<table>
<thead>
<tr>
<th>Book</th>
<th>Alice(1)</th>
<th>Bob(2)</th>
<th>Carol(3)</th>
<th>Dave(4)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Machine Learning for Dummies(1)</td>
<td>5</td>
<td>4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Hands-On Machine Learning(2)</td>
<td>?</td>
<td>5</td>
<td>?</td>
<td>0</td>
</tr>
<tr>
<td>Deep Learning(3)</td>
<td>5</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>A Kitten Called Holly(4)</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>?</td>
</tr>
<tr>
<td>Kittens 2018 Calendar(5)</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>4</td>
</tr>
</tbody>
</table>

Notation:

- \( n \) number of users, \( n = 4 \)
- \( m \) number of items, \( m = 5 \)
- \( d \) number of features
- \( R_{uv} \) rating given by user \( u \) to item \( v \), \( R_{11} = 5 \)
- \( \delta_{uv} = 1 \) if item \( v \) rated by user \( u \), 0 otherwise, \( \delta_{11} = 1, \delta_{12} = 0 \)
Matrix Completion

<table>
<thead>
<tr>
<th>Book</th>
<th>Alice(1)</th>
<th>Bob(2)</th>
<th>Carol(3)</th>
<th>Dave(4)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Machine Learning for Dummies(1)</td>
<td>5</td>
<td>4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Hands-On Machine Learning(2)</td>
<td>?</td>
<td>5</td>
<td>?</td>
<td>0</td>
</tr>
<tr>
<td>Deep Learning(3)</td>
<td>5</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>A Kitten Called Holly(4)</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>?</td>
</tr>
<tr>
<td>Kittens 2018 Calendar(5)</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>4</td>
</tr>
</tbody>
</table>

- Associate a feature vector \(x^{(v)}\) with \(v\)'th book, e.g. \(x^{(1)} = [1, 0]^T\), \(x^{(4)} = [0, 1]^T\) (number of features \(d = 2\))
- For each user \(u\) learn parameter vector \(\theta^{(u)}\), e.g. \(\theta^{(1)} = [5, 0]^T\), \(\theta^{(3)} = [0, 5]^T\)
- Predicted rating by user \(u\) of item \(v\) is \((\theta^{(u)})^T x\), e.g. rating by user 1 of item 1 is \([5, 0] \times [1, 0]^T = 5\)
Matrix Completion

- We are given a feature vector $x^{(v)}$ for $v$’th item/book
- Training data: a set of ratings $\{R_{uv}\}$ by users of a subset of the items (each user might only rate a few items)
- Hypothesis: predicted rating by user $u$ of item $v$ is:
  $$h_{\theta(u)}(x^{(v)}) = (\theta(u))^T x^{(v)}$$
- Parameters: $\theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(n)}$
- Cost function:
  $$J(\theta^{(1)}, \ldots, \theta^{(n)}) = \sum_{u=1}^{n} \sum_{v=1}^{m} \delta_{uv} \left( R_{uv} - (\theta(u))^T x^{(v)} \right)^2 + \lambda \sum_{u=1}^{n} (\theta(u))^T \theta(u)$$
- Select $\theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(n)}$ to minimise this cost function. This requires solving a least squares problem: use gradient descent or closed-form solution.
Matrix Completion

<table>
<thead>
<tr>
<th>Book</th>
<th>Alice(1)</th>
<th>Bob(2)</th>
<th>Carol(3)</th>
<th>Dave(4)</th>
<th>$x^{(1)}=[\text{ML}, \text{kittens}]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Machine Learning for Dummies(1)</td>
<td>5</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Hands-On Machine Learning(2)</td>
<td>?</td>
<td>5</td>
<td>?</td>
<td>0</td>
<td>?</td>
</tr>
<tr>
<td>Deep Learning(3)</td>
<td>5</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>A Kitten Called Holly(4)</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>Kittens 2018 Calendar(5)</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>4</td>
<td>0</td>
</tr>
</tbody>
</table>

- Associate a feature vector $x^{(v)}$ with $v$’th book. But what if we don’t know $x^{(v)}$?
- Suppose we know $\theta^{(1)} = [5, 0]^T$, $\theta^{(3)} = [0, 5]^T$, then

$$
[5, 0]^T x^{(1)} = 5, \quad [5, 0]^T x^{(3)} = 5, \quad [5, 0]^T x^{(4)} = 0
$$

$$
[0, 5]^T x^{(1)} = 0, \quad [0, 5]^T x^{(4)} = 5
$$

which is satisfied by $x^{(1)} = [1, 0]$, $x^{(3)} = [1, 0]$, $x^{(4)} = [0, 1]$
Matrix Completion

- Given \( \theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(n)} \), select \( x^{(1)}, x^{(2)}, \ldots, x^{(m)} \) to minimise

\[
\sum_{u=1}^{n} \sum_{v=1}^{m} \delta_{uv} \left( R_{uv} - (\theta^{(u)})^T x^{(v)} \right)^2 + \lambda \sum_{v=1}^{m} (x^{(v)})^T x^{(v)}
\]

- Define

\[
J(x^{(1)}, \ldots, x^{(m)}) = \sum_{u=1}^{n} \sum_{v=1}^{m} \delta_{uv} \left( R_{uv} - (\theta^{(u)})^T x^{(v)} \right)^2 \\
+ \lambda \sum_{v=1}^{m} (x^{(v)})^T x^{(v)} + \lambda \sum_{u=1}^{n} (\theta^{(u)})^T \theta^{(u)}
\]

- Repeat:
  - Given \( \theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(n)} \), select \( x^{(1)}, x^{(2)}, \ldots, x^{(m)} \) to minimise \( J \)
  - Given \( x^{(1)}, x^{(2)}, \ldots, x^{(m)} \), select \( \theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(n)} \) to minimise \( J \)

- Each update requires solving a least squares problem: use gradient descent or closed-form solution. This is called the alternating least-squares algorithm

- Recommendation: predicted rating by user \( u \) of item \( v \) is \( (\theta^{(u)})^T x^{(v)} \)
Matrix Factorisation

Another way to think about the same thing ...

- Observe ratings $R_{uv}$ by user $u$ for item $v$. Gather these into ratings matrix $R$. We want to predict the missing entries in $R$.

- To proceed, assume $R$ is low rank $d \ll n, m$ ...

$$R = \begin{bmatrix} 1 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \cdots & 1 \end{bmatrix} = \begin{bmatrix} d \end{bmatrix} = \begin{bmatrix} m \end{bmatrix} d$$

- Hypothesis: $R = U^T V$, but the elements of $U$ and $V$ are unknown.

- Cost Function: $\frac{1}{m} \sum_{u,v} (R_{uv} - (U^T V)_{uv})^2 + \lambda U^T U + \lambda V^T V$
Some Issues ...

- Cold-start (new user, new item)
- Popularity bias: hard to recommend to someone with unique tastes
  - Good quality data is always a key issue. Even with lots of data our model may not generalise well i.e. predict well for data outside the training set.

- What is the intrinsic noise when making predictions anyway? E.g. For Netflix data set the state of the art is RMSE of about 0.9. Ratings are concentrated between 3 and 5. So $4 \pm 0.9$ covers almost the whole range.
Issues

• Shilling attacks/adversarial data
  • Create costly barrier to keep bots etc out e.g. booking.com requires paying for a room in hotel before a review can be submitted.
  • Create barrier by building reputation over time e.g. stackoverflow

• And then there’s the question of privacy ...
  ... US, Europe and Asia have very different privacy regulations.
  • As access control (couched as “consent”) ...
  • Adding noise/perturbing the data ($k$-anonymity, differential privacy etc). Privacy comes at the cost of poorer performance.
  • Hiding in the crowd ?
Privacy by Design: Personalised Recommendations

Existing Recommender System
User accesses system via an account used only by them, so id and user strongly linked.

Conventional

OpenNym
User accesses system via an account shared with other users having similar interests.

OpenNym Identities

---

Privacy by Design: Personalised Recommendations

![MovieLens screenshot](https://movielens.org)

![Boxplot diagram](https://example.com)

- Nym size
- User rating size
- RMSE
- Type: separate accounts vs. OpenNym
Summary of the RMSE performance using validation sets from [1].

<table>
<thead>
<tr>
<th>Dataset</th>
<th>BMF</th>
<th>ALSWR</th>
<th>SVD++</th>
<th>SGD</th>
<th>Bias SVD</th>
<th>BLC local</th>
<th>(nyms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jester</td>
<td>4.33</td>
<td>5.64</td>
<td>5.54</td>
<td>5.72</td>
<td>5.82</td>
<td>4.30</td>
<td>4.20</td>
</tr>
<tr>
<td>Movielens</td>
<td>0.85</td>
<td>1.51</td>
<td>1.42</td>
<td>1.24</td>
<td>1.23</td>
<td>0.87</td>
<td>0.83</td>
</tr>
<tr>
<td>Dating</td>
<td>1.93</td>
<td>4.72</td>
<td>4.68</td>
<td>5.17</td>
<td>3.96</td>
<td>1.91</td>
<td>1.88</td>
</tr>
<tr>
<td>Books</td>
<td>1.94</td>
<td>4.71</td>
<td>4.73</td>
<td>5.18</td>
<td>3.95</td>
<td>1.96</td>
<td>1.87</td>
</tr>
<tr>
<td>Netflix</td>
<td>0.95</td>
<td>1.56</td>
<td>1.54</td>
<td>1.29</td>
<td>1.38</td>
<td>0.98</td>
<td>0.97</td>
</tr>
</tbody>
</table>

---