Overview

- Recommending = Matrices?
  - Matrix completion
  - Matrix Factorization
  - SLIM
  - BLC
- What do big companies use?
- Content of this lecture is optional – won’t be examined
Recommending = 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>

Define ratings matrix $R$ with element $R_{uv}$ equal to rating given by user $u$ to item $v$, e.g. $R_{12} = 4$

$$
R = \begin{bmatrix}
5 & 4 & 0 & 0 \\
? & 5 & ? & 0 \\
5 & ? & ? & ? \\
0 & 0 & 5 & ? \\
0 & 0 & 5 & 4
\end{bmatrix}
$$

Notation:

- $n$ number of users, $n = 4$
- $m$ number of items, $m = 5$
- $\delta_{uv} = 1$ if item $v$ rated by user $u$, 0 otherwise, $\delta_{11} = 1$, $\delta_{21} = 0$
Ratings matrix $R$, element $R_{uv}$ equal to rating given by user $u$ to item $v$, e.g.

\[
R = \begin{bmatrix}
5 & 4 & 0 & 0 \\
? & 5 & ? & 0 \\
5 & ? & ? & ? \\
0 & 0 & 5 & ? \\
0 & 0 & 5 & 4
\end{bmatrix}
\]

- The task of the recommender system is to predict the missing values in the ratings matrix (the ? elements) → matrix completion
- To make this task well-defined we need to add some constraints/restrictions on these predictions ...
Matrix Factorisation

Suppose we have following ratings data:

<table>
<thead>
<tr>
<th>Book</th>
<th>Alice(1)</th>
<th>Bob(2)</th>
<th>Carol(3)</th>
<th>Dave(4)</th>
<th>$x^{(1)}$ = [ML 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>

Rough idea:

- Each book has a topic e.g. machine learning or kittens. Use one-hot encoding to map topic to a feature vector of length $d = 2$ for an item. E.g.
  - Item 1 “Machine Learning for Dummies” has feature vector $x^{(1)} = [1, 0]$
  - Item 5 ‘Kittens 2018 Calendar” has feature vector $x^{(5)} = [0, 1]$

- Each user has a preference for the topics. So represent each user by a weight vector. E.g.
  - User 1 Alice has weight vector $\theta^{(1)} = [5, 0]$
  - User 3 Carol has weight vector $\theta^{(3)} = [0, 5]$

- Predict rating of item $v$ by user $u$ using:

$$\hat{R}_{uv} = (\theta^{(u)})^T x^{(v)} = \sum_{i=1}^{d} \theta^{(u)}_i x^{(v)}_i$$

E.g. Carol’s rating of “A Kitten Called Holly” is $0 \times 0 + 5 \times 1 = 5$ and predicted rating of “Deep Learning” is $0 \times 1 + 5 \times 0 = 0$
Matrix Factorisation

* **Idea:** Use linear model of items and user preferences
  * Each item $v$ is characterised by an abstract feature vector $x^{(v)}$ of length $d$. E.g. $d = 2$ and $x^{(1)} = [1, 4]$
  * Each user’s preferences are captured by the weight they place on each element of item feature vector i.e each user $u$ is characterised by a vector $\theta^{(u)}$ of $d$ weights. E.g. $\theta^{(1)} = [1, 0]$
  * Predicted rating of item $v$ by user $u$ is
    \[ \hat{R}_{uv} = (\theta^{(u)})^T x^{(v)} \]
    E.g. $\hat{R}_{11} = 1 \times 1 + 0 \times 4 = 1$.
  * **Cost function:** square error plus $L_2$ penalty
    \[
    \sum_{u=1}^{n} \sum_{v=1}^{m} \delta_{uv} (R_{uv} - \hat{R}_{uv})^2 + \sum_{u=1}^{n} (\theta^{(u)})^T \theta^{(u)}/C_1 + \sum_{v=1}^{m} (x^{(v)})^T x^{(v)}/C_2
    \]
  * Select $x^{(v)}$ and $\theta^{(u)}$ to minimise cost function e.g. using stochastic gradient descent
  * Note: cost is non-convex so training might be slow/tricky.
  * This approach won the Netflix prize in 2008. Matrix Factorization Techniques For Recommender Systems 2009
    https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5197422
Extra details:

* Calculate the mean rating $\mu$ over all items and users and write predicted rating as:

$$\hat{R}_{uv} = \mu + (\theta(u))^T x^{(v)}$$

i.e. $(\theta(u))^T x^{(v)}$ estimates deviations from mean rating.

* Some users tend to systematically give higher (or lower) ratings than other users $\rightarrow$ can mean that predictions based on their ratings are inaccurate. Add a bias parameter $b_u$ for user $u$, so predicted rating of item $v$ by user $u$ becomes:

$$\hat{R}_{uv} = \mu + b_u + (\theta(u))^T x^{(v)}$$

* Similarly for items. Add a bias parameter $b_v$ for item $v$:

$$\hat{R}_{uv} = \mu + b_u + b_v + (\theta(u))^T x^{(v)}$$

* Estimate parameters $b_u$, $b_v$, $\theta(u)$, $x^{(v)}$ by choosing values that minimise the cost function.
Example: Item-Based Movie Recommender

Python Lenskit package https://lkpy.readthedocs.io/en/stable/index.html implements many common recommender approaches, including matrix factorization collaborative filtering:

```python
from lenskit.algorithms.basic import Bias, Popular, TopN
from lenskit import topn
from lenskit.metrics.predict import rmse
import pandas as pd

algo_pop = Bias(); algo_ii = knn.ItemItem(20)
algo_als5 = als.BiasedMF(5)

def eval(aname, algo, train, test, all_preds):
    fittable = util.clone(algo)
    fittable = Recommender.adapt(fittable)
    fittable.fit(train)
    # predict ratings
    preds = batch.predict(fittable, test)
    preds["Algorithm"] = aname
    all_preds.append(preds)

def main():
    all_preds = []; test_data = []
    for train, test in xf.partition_users(ratings[['user', 'item', 'rating']], 5, xf.SampleFrac(0.2)):
        test_data.append(test)
        eval('BIAS', algo_pop, train, test, all_preds)
        eval('II', algo_ii, train, test, all_preds)
        eval('MF', algo_als5, train, test, all_preds)
    preds = pd.concat(all_preds, ignore_index=True)
    preds_ii = preds[preds['Algorithm'].str.match('II')]
    preds_bias = preds[preds['Algorithm'].str.match('BIAS')]
    preds_nf= preds[preds['Algorithm'].str.match('MF')]
    test_data = pd.concat(test_data, ignore_index=True)
    print('RMSE BIAS:', rmse(preds_bias['prediction'],preds_bias['rating']))
    print('RMSE II:', rmse(preds_ii['prediction'],preds_ii['rating']))
    print('RMSE MF:', rmse(preds_mf['prediction'],preds_mf['rating']))

if __name__ == '__main__':
    ml100k = ML100K('ml-100k')
ratings = ml100k.ratings; print(ratings.head())
main()
```
Example: Item-Based Movie Recommender

* Download data using:
  wget http://files.grouplens.org/datasets/movielens/ml-100k.zip
  unzip -f ml-100k.zip

* Typical output:

```
user item rating prediction Algorithm
3924 1 143 1.0 4.031989 II
3925 1 6 5.0 4.092580 II
3926 1 211 3.0 4.183012 II
3927 1 157 4.0 4.070335 II
3928 1 190 5.0 4.534220 II

user item rating prediction Algorithm
7848 1 143 1.0 2.989210 MF
7849 1 6 5.0 3.859588 MF
7850 1 211 3.0 4.437893 MF
7851 1 157 4.0 4.204632 MF
7852 1 190 5.0 4.026473 MF

user item rating prediction Algorithm
0 1 143 1.0 3.862483 POP
1 1 6 5.0 3.566096 POP
2 1 211 3.0 4.017835 POP
3 1 157 4.0 3.790314 POP
4 1 190 5.0 4.241681 POP

RMSE POP: 0.9403746168971867
RMSE II: 0.8997637525055545
RMSE MF: 0.909195791898838
```
Why “Matrix Factorisation”? Another way to think about the same thing ...

* Gather user parameter vectors into a matrix $U$:

$$U = [\theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(n)}]$$

where $[\theta^{(v)}$ is a column vector, so $U$ is a $d \times n$ matrix

* Gather item feature vectors into a $d \times m$ matrix $V$:

$$V = [x^{(1)}, x^{(2)}, \ldots, x^{(m)}]$$

* The transpose $U^T$ of matrix $U$ is

$$U^T = \begin{bmatrix}
(\theta^{(1)})^T \\
(\theta^{(2)})^T \\
\vdots \\
(\theta^{(n)})^T
\end{bmatrix}$$

and

$$U^T V = \begin{bmatrix}
(\theta^{(1)})^T x^{(1)} & (\theta^{(1)})^T x^{(2)} & \cdots & (\theta^{(1)})^T x^{(m)} \\
(\theta^{(2)})^T x^{(1)} & (\theta^{(2)})^T x^{(2)} & \cdots & (\theta^{(2)})^T x^{(m)} \\
\vdots & \vdots & \ddots & \vdots \\
(\theta^{(n)})^T x^{(1)} & (\theta^{(n)})^T x^{(2)} & \cdots & (\theta^{(n)})^T x^{(m)}
\end{bmatrix}$$

* So $\hat{R} = U^T V$ is prediction for rating matrix $R$. Expressing matrix $\hat{R}$ as the product of two factors $U$ and $V \rightarrow \text{matrix factorisation}$
* Idea: use a slightly more flexible/general matrix factorization model

* Rating matrix $R$. Let $R_v$ be column of $R$ corresponding to item $v \rightarrow$ a vector consisting of all the user ratings of item $v$.

* Predicted rating of item $v$ by user $u$:

$$\hat{R}_{uv} = R_v^T(\theta^{(v)}) = \sum_{i=1}^m \delta_{uv'} \theta_v^{(v)} R_{uv'}$$

where $\theta^{(v)}$ is a vector of weights and $\delta_{uv} = 1$ when user $u$ has rated item $v$ and 0 otherwise. So $R_{uv}$ is a weighted linear combination of the user’s previous item ratings.

* Cost function:

$$\sum_{u=1}^n \sum_{v=1}^m \delta_{uv}(R_{uv} - \hat{R}_{uv})^2 + \sum_{v=1}^m (\theta^{(v)})^T \theta^{(v)}/C_1 + \sum_{v=1}^m \sum_{v'=1}^m |\theta_v^{(u)}|/C_1$$

* Constraints: $\theta_v^{(v)} \geq 0, \theta_v^{(v)} = 0$. Note: $\theta_v^{(v)} \geq 0$ constraints sometimes dropped to simplify optimisation

* Performance of SLIM found to be good (often better than vanilla MF) in many independent tests.
Vanilla matrix factorization treats the user population as being homogeneous, then tries to patch things up by adding extra $b_u$ parameters.

But we expect that user’s can be grouped into rough clusters → e.g. advertisers have known this forever.

Idea: Have a set of user groups, suppose each user belongs to just one group. For users in same group use matrix factorization to predict ratings (so all users in same group have same predicted ratings).


Predicted rating for all users in group $g$ is

$$\hat{R}_{uv} = (\theta(g)))^T x(v)$$

and parameter $g^{(u)}$ is the group of user $u$. E.g. $g^{(1)} = 2$ when user 1 is in group 2

Select parameters $\theta(g)$, $x(v)$, $g^{(u)}$ to minimise the usual MF cost

Tests show state of the art performance with no further tuning.

Number of groups is a hyperparameter, but typically a small number is sufficient e.g. 32 groups → is that surprising or not?
Clustering Users: Privacy By Design

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

Conventional

Existing Recommender System → User Client App
(user id, item, rating) ▶️ recommendation

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

OpenNym

Existing Recommender System

User Client App

OpenNym Identities

(recommendation) ▶️ (group id, item, rating)
Summary

* Implicit feedback (clicks etc \(\rightarrow\) lack of negative feedback)
  * Plenty of pretty effective approaches for incorporating implicit feedback. E.g.
* Context e.g. time, price/discounts, review text
  * Typical approach is to extend the predicted rating to include extra context info:

\[
\hat{R}_{uv} = \mu + b_u + b_v + (\theta^{(u)})^T \chi^{(v)} + \text{context parameters}
\]
* Sessions
  * Not really, though idea of using abstract feature vector for users/items carries over.
* Cold-start
  * A thorny problem for all collaborative filtering approaches.
  * For new users can recommend most popular items until know more about user \(\rightarrow\) clustering of users into groups can help
  * For new items one approach is to randomly include them in recommendations to a few users to elicit initial feedback
What do big companies use?

**Amazon**

- Two Decades of Recommender Systems at Amazon.com 2017
  
  https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7927889
  
  - Item-based collaborative filtering, with tweaks to similarity measure.

**Netflix**

- ADMM SLIM: Sparse Recommendations for Many Users 2020
  
  
  - SLIM variant of matrix factorization

**Youtube**

- Deep Neural Networks for YouTube Recommendations 2016
  
  
  - Basically matrix factorization, with tweaks and implemented using neural net.

- Latent Cross: Making Use of Context in Recurrent Recommender Systems 2018
  
  
  - Use recurrent neural net for low-rank (i.e. matrix factorization-like) *session-based* approach. Softmax layer maps from RNN output to probability vector across all videos.
What do big companies use?

Facebook

- **Recommending Items To More Than A Billion People 2015**
  https://engineering.fb.com/core-data/recommending-items-to-more-than-a-billion-people/
  - Uses matrix factorization.
  - Article spends a lot of time discussing how to make things scalable.

- **Deep Learning Recommendation Model for Personalization and Recommendation Systems 2019**
  https://github.com/facebookresearch/dlrm
  - Map inputs to feature vector using a mix of approaches (incl. matrix factorization and neural nets).
  - Use MLP to map from features to a prediction of probability of a click.
  - Not many other concrete details though ...

Instagram

- **Powered by AI: Instagrams Explore Recommender System 2019**
  https://ai.facebook.com/blog/powered-by-ai-instagram-of-a-billion-people/
  - Map from account info to a feature vector using a neural net approach.
  - Keep track of accounts a user has interacted with previously. Use cosine distance between accounts to find similar accounts. Then use those to form list of candidate recommendations.
  - Rank candidates by predicting actions (like, save, negative action) that person is likely to take. Use a multi-pass MLP then ConvNet approach for this.
We haven’t talked about using ConvNets/Deep Learning for making recommendations ...

* Session-Based Recommendations With Recurrent Neural Networks 2016
  * Seems to work pretty well in independent evaluations
  * Recurrent neural nets are being used by youtube for making session-based recommendations
  * Recurrent neural net = treat interactions as a time-series, we leave time-series to another module!

* Are We Really Making Much Progress? A Worrying Analysis of Recent Neural Recommendation Approaches 2019
  https://dl.acm.org/doi/pdf/10.1145/3298689.3347058
  * Session-based approaches aside, serious concerns have been raised about claims of performance gains from use of ConvNets → item-based and matrix factorization approaches frequently observed to outperform ConvNets when fairly compared.
  * Jury is still out on this, but wise to be cautious about over-hyped claims and motivated reasoning re deep learning.
Some Known Issues

- **Shilling attacks/adversarial data.** E.g.
  - A hotel might post fake reviews (or pay someone else to post them) talking itself up, or talking down the opposition
  - Similarly with product reviews
  - Botnets and large-scale click fraud
  - Manipulation on facebook, twitter etc by state actors

- **Seems like no easy solutions ...**
  - Create costly barrier to keep bots etc out e.g. must pay for a room in hotel before a review can be submitted.
  - Create barrier by building reputation over time e.g. stackoverflow
  - Manual moderation/policing (automated statistical ML methods not so effective in adversarial situations)

- **Filter bubbles**
  - Feedback loop: recommendations → user actions → recommendations
  - Commercial pressure: maximising clicks etc
  - Manipulation: fake news?

- **Privacy/surveillance capitalism**
  - US, Europe and Asia have very different privacy regulations.
  - As access control (couched as “consent”) ...
  - Adding noise/perturbing the data (k-anonymity, differential privacy).
  - Hiding in the crowd/clustering
  - Emperors new clothes: maybe we can make equally bad recommendations with far less data?