Personalised recommendations are the application of machine learning most of us experience daily. Many examples:

- **Shopping, e.g. amazon:**

- **Videos, Music, e.g. youtube:**

- **News**

- **Apps**

- **Adverts**
On a scale of 1-5 (1=very good, 5=very bad), how useful do you find the recommendations you receive for:

- Videos e.g. youtube
- Movies e.g. netflix
- Music e.g. spotify
- Shopping e.g. amazon
- Adverts e.g. in google search

And:

- Do you use an ad blocker?
Recommender Systems

* Deciding whether an image contains a cat or not is an objective “technical” kind of problem – its fixed and well defined, fairly easy to agree when we’ve succeeded.

* In contrast making personalised recommendations are:
  * Subjective:
    * Personal. No ground truth, what I like you might dislike
    * Time-varying. What I like today I might be bored by in a month’s time
    * Feedback. Recommendations can change user behaviour/steer users, and recommender learns user behaviour → filter bubbles etc
  * Hard to evaluate performance → this is a whole research topic!

* Often conflicting objectives:
  * User wants useful suggestions
  * Operator of recommender system is providing the service for a reason. E.g. operator might prefer to recommend:
    * Most profitable movie rather than the one user would most like
    * News headline mostly likely be clicked rather than the one most likely to be informative
  * Content producers may want to manipulate recommendations to promote their content

* Often privacy concerns → surveillance capitalism

* Not such clear progress as in image processing → due to intrinsic noise or lack of data? Emperor’s new clothes?
Recommender Systems

- Repeat
  System displays info $y_t$ to user
  User takes action $x_t$

- *User actions* $x_t$
  - Clicks, ratings, time spent watching a video or reading news, purchase of an item, tweeting a link etc

- *Displayed info* $y_t$
  - Image of product, price, reviews, ratings by other users etc

- *Sequence* of displays to user and corresponding user actions 
  e.g. recommendations update as user explores displayed products by clicking on links
Recommender Systems

* Multiple users of system, so can learn from population of users

* Context e.g.
  * User’s current situation and short-term intents and interest.
  * Time → recent news, newer movies, videos of more interest
  * Price → are some items discounted/on sale?
  * Ordering → if buy camera may be interested in memory card
  * Recommend only unseen items or repeatedly recommend past ones → “Repeated recommendations as reminders”?

* Have less data on some users than others, especially:
  * Cold start: New users, new items e.g. news articles, videos
Let’s simplify:

- **User actions.** Map user action to a rating value e.g. between 0 and 1.
  - We know which items a user has rated and which they have not.
  - Have negative feedback: a rating of 0 tells us a user didn’t like an item, its *not* the same as not rating an item (which might occur because they don’t know an item exists, or for other reasons)

- **Display info.** Have a set $V$ of items

- **Sequence.** Ignore sequence information

- **Multiple users.** Have set $U$ of users.

- **Context.** Ignore.

- **Cold start.** Ignore.

So what we observe is a set of triplets (user, item, rating). This is a minimal setup, not realistic but hopefully captures the essence of the recommender task.
**Content-based Recommendation**

Task: predict the top $N$ items for user (the $N$ most highly rated items not already seen).

- **Idea 1:** recommend items to user that are similar to items previously rated highly by user
  1. How to measure similarity of items?
  2. How to predict rating for unseen items?

**Measuring similarity:**

- Map from details of item $v$ to feature vector $x(v)$ E.g.
  - Map item category/genre to feature using one-hot encoding
  - Map text description to feature vector using bag of words model and TF-IDF

- Calc cosine similarity of items $v$, $w$. For feature vectors $x(v)$ and $x(w)$ cosine similarity is $s(v, w) = \frac{\sum_{j=1}^{n} x_j(v)x_j(w)}{\sqrt{\sum_{j=1}^{n} x_j^2(v)}\sqrt{\sum_{j=1}^{n} x_j^2(w)}}$

**Predict user $u$ rating for unseen item $v$ using $k$NN approach**

- Find set $N_k$ of $k$ items seen/rated by user $u$ and most similar to unseen item $v$
- Predicted rating $\hat{R}_{uv} = \frac{\sum_{w\in N_k} s(v,w)R_{uw}}{\sum_{w\in N_k} s(v,w)}$
**Content-based Recommendation**

**Cosine similarity:**

- Suppose feature vector for item $v$ is $[1, 0, 0, 1]$ and for item $w$ is $[1, 0, 1, 0]$, then

\[
\sum_{j=1}^{n} x_j(v)x_j(w) = 1 \times 1 + 0 \times 0 + 0 \times 1 + 1 \times 0 = 1
\]

and

\[
\sqrt{\sum_{j=1}^{n} x_j^2(v)} = \sqrt{2}, \quad \sqrt{\sum_{j=1}^{n} x_j^2(w)} = \sqrt{2}
\]

so $s(v, w) = \frac{1}{2}$

- Suppose now feature vector for item $w$ is $[1, 0, 0, 1]$, then

\[
\sum_{j=1}^{n} x_j(v)x_j(w) = 1 \times 1 + 0 \times 0 + 0 \times 0 + 1 \times 1 = 2
\]

and $s(v, w) = 1$

- For the product $x_j(v)x_j(w)$ to be large we need *both* $x_j(v)$ and $x_j(w)$ large
import pandas as pd
import numpy as np
df = pd.read_csv("movie_dataset.csv")

def get_features(row):
    features = ''
    for c in ['keywords', 'cast', 'genres', 'director']:
        features = features + str(row[c]) + ''
    return features

df['features'] = df.apply(get_features, axis=1)

from sklearn.feature_extraction.text import TfidfVectorizer

tf = TfidfVectorizer(stop_words='english', max_df=0.2).fit_transform(df['features'])

# this does all the work - calculates the similarity between every pair of items,
# output is a matrix
from sklearn.metrics.pairwise import cosine_similarity
sims = cosine_similarity(tf)

movie_index = df[df.title == "Star Wars"]['index'].values[0]
predictions = np.argsort(-sims[movie_index])
for pred in predictions[1:6]:
    print(df[df.index==pred]['title'].values[0])

IMDb dataset columns:
'budget', 'genres', 'homepage', 'id', 'keywords', 'original_language', 'original_title', 'overview', 'popularity', 'production_companies',
'production_countries', 'release_date', 'revenue', 'runtime', 'spoken_languages', 'status', 'tagline', 'title', 'vote_average',
'vote_count', 'cast', 'crew', 'director'
Example 'features':
0 culture clash future space war space colony so...
1 ocean drug abuse exotic island east india trad...
2 spy based on novel secret agent sequel mi6 Dan...
3 dc comics crime fighter terrorist secret ident...
4 based on novel mars medallion space travel pri...

Example output:
The Empire Strikes Back
Return of the Jedi
Star Wars: Episode I - The Phantom Menace
Star Wars: Episode II - Attack of the Clones
Star Wars: Episode III - Revenge of the Sit
Item-based Collaborative Filtering

Task: predict the top $N$ items for user (the $N$ most highly rated items not already seen).

* Idea 2: recommend items that similar users previously rated highly \( \rightarrow \) collaborative filtering since using info from population of users

Measuring similarity of users:

* Let $M_u$ be set of items rated by user $u$ and $R_{uv}$ be rating of item $v \in M_u$ by user $u$. Then $M_{uu'} = M_u \cap M_{u'}$ is the set of items rated by both users. Cosine distance between users $u, u'$:

$$s(u, u') = \frac{\sum_{v \in M_{uu'}} R_{uv} R_{u'v}}{\sqrt{\sum_{v \in M_{uu'}} R_{uv}^2} \sqrt{\sum_{v \in M_{uu'}} R_{u'v}^2}}$$

i.e sum the product of the user ratings for items rated by both user $u$ and user $u'$. Will tend to be large when users both rate same items highly

* Problems: (i) usually each user rates only a small number of items so $M_u \cap M_{u'}$ might be small/empty, (ii) computationally expensive to calc for all user pairs $u, u'$ when have many users
Item-based Collaborative Filtering

Idea 3: Measure similarity of items collaboratively


* For two items $u$ and $w$ let $U_v$ be the set of users who have rated item $v$ and $U_w$ the set of users who have rated item $w$. Then $U_{vw} = U_v \cap U_w$ is the set of users who have rated both $u$ and $v$. Collaborative cosine similarity between items $v$ and $w$ is:

$$s(v, w) = \frac{\sum_{u \in U_{vw}} R_{uv} R_{uw}}{\sqrt{\sum_{u \in U_{vw}} R_{uv}^2} \sqrt{\sum_{u \in U_{vw}} R_{uw}^2}}$$

$s(v, w)$ large when users tend to rate items $v$ and $w$ similarly

Predict user $u$ rating for unseen item $v$ using $k$NN approach

* Find set $N_k$ of $k$ items seen/rated by user $u$ and most similar to unseen item $v$

* Predicted rating $\hat{R}_{uv} = \frac{\sum_{w \in N_k} s(v, w) R_{uw}}{\sum_{w \in N_k} s(v, w)}$

Note: (i) if enough users then $U_{vw}$ should be pretty large, (ii) #items usually much smaller than #users so calc for all item pairs $v, w$ not too expensive to compute
Collaborative cosine similarity:

* We can gather ratings data into ratings matrix $R$, element $R_{uv}$ is equal to rating given by user $u$ to item $v$, e.g. $R_{12} = 4$

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

* Users rating both items 1 and 2:

$$ R = \begin{bmatrix} 5 & 4 & 0 & 0 \\ ? & 5 & ? & 0 \\ 5 & ? & ? & ? \\ 1 & 0 & 5 & ? \\ 0 & 1 & 5 & 4 \end{bmatrix} $$

So $U_1 = \{1, 3, 4, 5\}$, $U_2 = \{1, 2, 4, 5\}$ and $U_{12} = U_1 \cap U_2 = \{1, 4, 5\}$ and

$$ \sum_{u \in U_{12}} R_{u1} R_{u2} = 5 \times 4 + 0 \times 5 + 0 \times 1, \quad \sqrt{\sum_{u \in U_{12}} R_{u1}^2} = \sqrt{5^2 + 1^2 + 0^2}, \quad \sqrt{\sum_{u \in U_{12}} R_{u2}^2} = \sqrt{4^2 + 0^2 + 1^2} $$

and $s(1, 2) = \frac{20}{\sqrt{26} \sqrt{16}} = 0.98 \rightarrow \text{items 1 and 2 are pretty similar wrt ratings.}$

Repeat calc for items 2 and 3, $s(2, 3) = \frac{5}{\sqrt{17} \sqrt{50}} = 0.17 \rightarrow \text{much less similar}$
Example: Item-Based Movie Recommender

Python Lenskit package https://lkpy.readthedocs.io/en/stable/index.html implements many common recommender approaches, including item-based 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)

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)
    preds = pd.concat(all_preds, ignore_index=True)
    preds_ii = preds[preds['Algorithm'].str.match('II')]
    print(preds_ii.head())
    preds_bias = preds[preds['Algorithm'].str.match('BIAS')]
    print(preds_bias.head())
    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']))

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:

<table>
<thead>
<tr>
<th>user</th>
<th>item</th>
<th>rating</th>
<th>prediction</th>
<th>Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
<td>354</td>
<td>3.0</td>
<td>3.371460</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>319</td>
<td>2.0</td>
<td>3.028429</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>345</td>
<td>3.0</td>
<td>3.250216</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>355</td>
<td>3.0</td>
<td>2.331747</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>294</td>
<td>2.0</td>
<td>2.898721</td>
</tr>
</tbody>
</table>

user item rating prediction Algorithm

<table>
<thead>
<tr>
<th>user</th>
<th>item</th>
<th>rating</th>
<th>prediction</th>
<th>BIAS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
<td>354</td>
<td>3.0</td>
<td>2.950680</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>319</td>
<td>2.0</td>
<td>2.729116</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>345</td>
<td>3.0</td>
<td>2.852062</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>355</td>
<td>3.0</td>
<td>2.389965</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>294</td>
<td>2.0</td>
<td>2.610576</td>
</tr>
</tbody>
</table>

RMSE BIAS: 0.9431657319235357
RMSE II: 0.9004012551052291
Session-based Recommendation

* Useful recommendations often depend on user’s current situation and short-term intents and interest
* Often users interact with a system in a “session” e.g. go to retailer web site to look for a coffee machine. Then come back and this time look for a jacket.

* Can use recent user interactions to improve recommendations
* Unsurprisingly, can give a significant boost in performance e.g. see:
  * Session-based Item Recommendation in E-Commerce 2017
  * Effective Nearest-Neighbor Music Recommendations 2018
* How to extend previous approach to use sessions?
Session-based Collaborative Filtering

How to extend previous approach to use sessions?

- Define a user session e.g. last $N$ interactions with a user
- Use nearest neighbours on sessions rather than single items:
  - Set $S$ of past sessions for all users. Each session $a \in S$ consists of a set $I_a$ of items and their ratings $r_a(v)$, $v \in I_a$.
  - Measure similarity between two sessions $a$ and $b$ using:
    \[
    s(a, b) = \frac{|I_a \cap I_b|}{\sqrt{|I_a||I_b|}}
    \]
    where $I_a \cap I_b$ is set of items that appear in both sessions $a$ and $b$.
  - Let $N_k$ be the set of $k$ sessions in $S$ closest to current session, e.g. use $k = 500$
  - Predicted rating of item $v$ by user $u$ is:
    \[
    \hat{R}_{uv} = \frac{\sum_{b \in N_k} s(a, b) 1_b(v)}{\sum_{w \in N_k} s(a, b)}
    \]
    where $1_b(v) = 1$ when session $b$ contains item $v$ and otherwise $1_b(v) = 0$
  - Many tweaks possible. E.g. replace $1_b(v) = 1$ by the rating $r_a(v)$ of item $v$ by the user in session $b$, in $s(a, b)$ give greater weight to more recent items in session
Session-based Recommendation

Session similarity;

* Suppose two previous sessions with items $I_1 = \{5, 4, 1, 10\}$ and $I_2 = \{1, 5, 10, 9, 20\}$. Then $I_1 \cap I_2 = \{1, 5, 10\}$ and session similarity is:

$$s(1, 2) = \frac{|I_1 \cap I_2|}{\sqrt{|I_1||I_2|}} = \frac{3}{\sqrt{4 \times 5}} = 0.67$$

* Suppose current session 3 has $I_3 = \{4, 5\}$ and set of $k$ nearest sessions is $N_k = \{1, 2\}$. Then $1_1(10) = 1$ and $1_2(10) = 1$, $s(3, 1) = \frac{2}{\sqrt{2 \times 4}} = 0.7$, $s(3, 2) = \frac{1}{\sqrt{2 \times 5}} = 0.31$. Predicted rating for item 10 is:

$$\hat{R}_{u10} = \frac{s(3, 1)1_1(10) + s(3, 2)1_2(10))}{s(3, 1) + s(3, 2)} = \frac{0.7 \times 1 + 0.31 \times 1}{0.7 + 0.31} = 1$$

and for item 7 is

$$\hat{R}_{u7} = \frac{0.7 \times 0 + 0.31 \times 0}{0.7 + 0.31} = 0$$
Collaborative Filtering With Implicit User Feedback

- So far we assumed that user’s rate an item and we know which items have been rated.
- What about clicks?
  - A click probably indicates some interest an item, but a single click is a weak signal as to whether a user likes an item or not
  - Main problem: lack of negative feedback. Absence of a click might mean two things. (i) user saw item but wasn’t interested in it, (ii) user doesn’t know item exists
- One way to measure similarity between items $v$ and $w$ is:

$$s(v, w) = \frac{|U_{vw}|}{\sqrt{|U_v||U_w|}}$$

where $U_{vw}$ is the set of users who have clicked on both items $v$, $w$, $U_v$ the set of users who have clicked item $v$ and $U_w$ the set of users who have clicked item $w$
- Observing repeated clicks is more informative, so keep can track of #clicks and use that as a surrogate rating
- Lack of negative feedback still a problem though
Summary

* With item-based approaches it's easy to incorporate context information by modifying similarity. E.g. to include:
  * Time between when two videos/news articles were posted
  * Difference in price
  * Review text sentiment

* Item-based approaches are easy to understand, easy to implement

* Widely used, a decent baseline
  * Two Decades of Recommender Systems at Amazon.com 2017
    https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7927889

* For collaborative filtering it's essential to have enough users so that $U_{vw}$ is small/empty $\rightarrow$ otherwise use a content-based approach e.g. use bag of words+TFIDF to find items with similar descriptions, type etc

* Thorny problems:
  * **Cold start.** (i) New user who hasn’t rated anything yet. Typically fall back to recommending most popular items until get more info about new user. (ii) Collaborative filtering also has item cold-start problem i.e. new item has no ratings yet
  * **Implicit feedback.** Clicks don’t provide negative feedback