CE888: Data Science and Decision Making
Lecture 4: Recommender Systems

Ana Matran-Fernandez
University of Essex

February 4, 2019

Recommender systems

- We will discuss one of the most popular applications of data science
- Every website does it
- Why? To improve relevancy of information in the age of information overload
  - Having lots of information is great, but if we can't filter it, it's useless!
  - The paradox of choice
- What? A computer program to automatically serve the right item to a user in a given context to optimise long-term business objectives
- Involves:
  - ML and statistics
  - Optimization
  - User modeling
  - (Some times) NLP to understand content

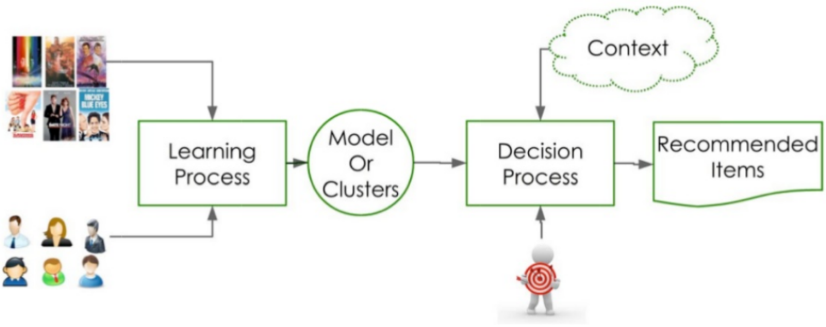
The value of recommendations

- 2/3 of the movies watched on Netflix are recommended
- Google news: recommendations generate 38% more clicks
- 35% of Amazon's sales come from recommendations
- People recommendation in LinkedIn, Twitter, Facebook…
- **Very hot topic!**

The Recommender Problem

- Estimate a **utility function** that **automatically predicts** how a user will **like** an item
- Based on:
  - past behaviour
  - relations to other users
  - item similarity
  - context (e.g., location, time)
  - ...

Two-step process



Approaches to recommender systems

- Collaborative filtering
  - Recommendation based on the user's past behaviour
  - No domain expertise needed
  - User-based approach: find similar users and recommend what they liked
  - Item-based approach: find similar items (based on past user behaviour, not on item features) to those I've previuosly liked
- Content-based filtering
  - Recommendation based on features from the item
- (there are more)

Not only ML

- User interface
- System requirements
- Serendipity: when you find something you didn't know you were looking for
  - Completely unexplored, not based on previous data

What works

- Depends on the domain and the particular problem
- Rule of thumb: collaborative filtering seems to work best
- What matters:
  - Data preprocessing
  - Dimensionality reduction using SVD/MF
  - Combining methods

Collaborative filtering

- Collaborative filtering is an effort to predict how products will be rated by a user, given previous ratings (from both the user and others)
- This prediction can help us recommend to the user only items that we think she will rate highly
- We have a list of n *users* and m *items*
- Each user has a list of items with an associated opinion
  - Explicit (e.g., a rating score)
  - Implicit (e.g., the user clicked/listened to a song) – **most of the time**
- Active user for whom we're recommending
- A metric to measure similarity between users
- A method for selecting a subset of neighbors
- A method for predicting a rating for items not currently rated by our user of interest

Collaborative filtering steps

1. Identify a set of ratings for the active user/target.
2. Identify a set of users most similar to the target according to a similarity function (neighborhood formation).
3. Identify products that the neighbours liked.
4. Generate a prediction (the rating that would be given by the target to the product) for each of those products.
5. Based on these predicted ratings, recommend a set of the top N products.

Additional division

- Neighborhood methods
  1. Items rated by the same user are clustered to try and establish the user's taste.
  2. Those ratings are compared to those from other users to find users with similar taste.
  3. Based on the similarities, the system recommends products that the similar users rated highly.
- Latent factor models
  1. Attempt to analyse the user-item interaction through factors inferred or gathered explicitly.
  2. These are called *latent factors*.

User-based CF

- Users $u_i, i = 1, ..., n$; products $p_j, j = 1, ..., m$
- V: $n \times m$ matrix of ratings, with $v_{i,j} = \text{NaN}$ if user i didn't rate product j
- Prediction for $v^*_{i,j}$:

$$v^*_{i,j} = K \sum_{v_{k,j} \neq NaN} (u_{i,k} \cdot v_{k,j})$$

or

$$v^*_{i,j} = v_i + K \sum_{v_{k,j} \neq NaN} (u_{i,k} \cdot (v_{k,j} - v_k))$$

where v_i is the average rating given by user i

- Similarity $u_{i,k}$ (weight for the neighbors) by Pearson correlation or cosine similarity
  - for getting the neighbors (e.g., the 10 closest ones)
  - for weighing the averages to calculate the predictions

Some sample data

	The Call of Cthulhu	Frankenstein	Dracula	Neuromancer	Dune
0	6	0	0	7	NaN
1	5	0	5	6	5
2	9	NaN	4	NaN	8
3	4	NaN	2	5	6
4	4	NaN	4	6	0
5	6	3	8	5	7
6	NaN	6	NaN	6	7
7	NaN	1	1	6	NaN
8	NaN	NaN	2	NaN	9
9	NaN	3	4	5	7

Challenges of memory-based CF methods

- Sparsity (<1% of ratings in the matrix)
  - 100M our of 850M in the Netflix Prize
  - Large sets of products, small percentage of items rated
  - and the probability of two users having one rating in common is very small!
- Scalability
  - Very costly to compute the similarity between all users or all items in the dataset
  - Clustering techniques such as k-means can help
- There could be a poor relationship among like-minded but sparse-rating users

Solution: use of latent factors to capture similarity between users and items in a reduced dimensionality space

Before moving on to latent factors

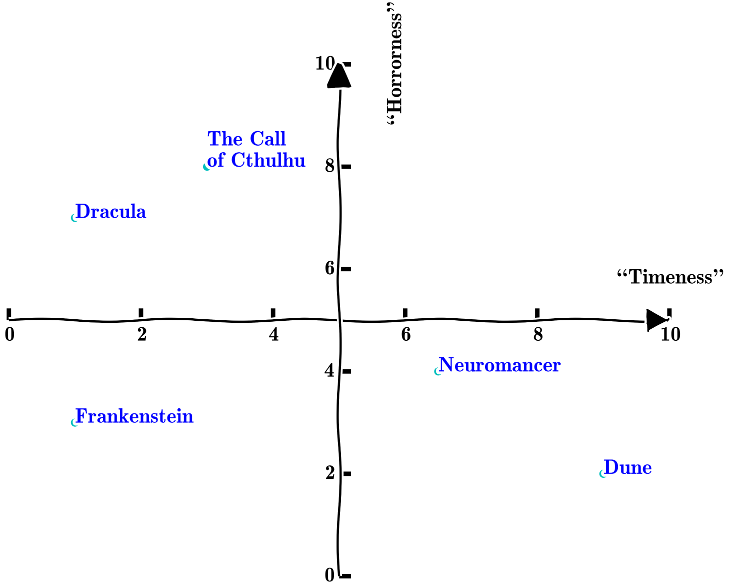
- **Item-based collaborative filtering**
  - Find similar items
  - If you liked A, you will like things similar to A
- We still have no information about the items
- We will define them as similar if the same subset of users liked them
- Using past ratings from the users
- Again, cosine similarity or Pearson correlation similarity

Factors

- Another partition for CF is Memory-based (use all the data I have) vs. **Model-based** (let's be smart)
- Instead of recommending items, I can recommend topics
- We are going to base our predictions on "hidden" qualities of the items and the users (**factors**)
- E.g., level of spiciness of food; bitterness of a drink
- A sensible way of describing items would be to see them as a collection of "factors"
  - But our data is just ratings!
- Thus, our factors are "latent" (i.e., hidden)
- "Assume that a user's rating of a movie is composed of a sum of preferences about the various aspects of that movie"[1]

[1] Simon Funk — `https://sifter.org/~simon/journal/20061211.html`

Example factors for our data



Factors and user preferences

- Let's assume f (unknown!) factors
- We can encode factors as a real valued vector $\boldsymbol{item\_factors_j} = [\phi_0, \phi_1, ..., \phi_{f-1}], i = 1, ..., m$
- For example "The Call of the Cthulhu" can be encoded as $\boldsymbol{item\_factors_0} = [3, 8]$
- Each user now can have preferences over factors, $\boldsymbol{user\_preferences_i} = [w_0, w_1, ..., w_{f-1}], j = 1, ..., n$
- The weight vector contains user preferences, e.g. $\boldsymbol{user\_preferences_0} = [0.5, 0.8]$
- But we don't have any user weights nor any item factors
    - Start by generating some random values

Some random data

- Each row in ***user_preferences*** represents the preferences of a user, while each row in ***item_factors*** represents the factors of an item

```
array([[ 0.092,  0.783],           array([[ 0.338,  0.519],
       [ 0.78 ,  0.488],                  [ 0.69 ,  0.256],
       [ 0.844,  0.062],                  [ 0.363,  0.93],
       [ 0.68 ,  0.549],                  [ 0.004,  0.112],
       [ 0.212,  0.43 ],                  [ 0.608,  0.104]])
       [ 0.961,  0.023],
       [ 0.659,  0.31 ],
       [ 0.92 ,  0.769],
       [ 0.817,  0.452],
       [ 0.834,  0.887]])
```

To obtain the rating we multiply:
$\boldsymbol{rating}[\boldsymbol{i}][\boldsymbol{j}] = \boldsymbol{user\_preferences}[\boldsymbol{i}] \cdot \boldsymbol{item\_factors}[\boldsymbol{j}]$:

$\boldsymbol{rating}[\mathbf{0}][\mathbf{0}] \leftarrow 0.437 = 0.092 * 0.338 + 0.783 * 0.519$

Far away from the observed rating of 6.

Predict ratings

- Example python code
- Still random values...

```
def predict_rating(user_row, item_row):
    """ Predict a rating """
    user_values = user_preferences[user_row]
    item_values = item_factors[item_row]
    return user_values.dot(item_values)
```

Training for a single example

- ***user_preferences*** and ***item_factors*** have random values
- Find the difference between the real and the predicted rating (“how far away am I from the goal ?”)
- Multiply by small learning rate $\alpha = 0.0001$ (“Don’t take my measurement so seriously”)
- Move ***user_preferences*** and ***item_factors*** towards the correct value, following the negative of the gradient (“Let’s move towards the direction of the most abrupt change”)

```python
def train(user_row, item_row, rating, alpha=0.0001):
    """ Adapt the values of user_preferences and item_factors
        to match the ones predicted by the users
    """
    err = alpha * (rating - predict_rating(user_row, item_row))
    user_preferences[user_row] += err * item_factors[item_row]
    item_factors[item_row] += err * user_preferences[user_row]
```

Training using all data

- Loop over all ***user_preferences*** and ***item_factors***
- Ignore cells with no value
- Repeat until some criterion

```python
def sgd_svd(iterations=30000):
    """ Iterate over all users and all items and train for
        a certain number of iterations
    """
    for i in range(iterations):
        for user_row in range(user_preferences.shape[0]):
            for item_row in range(item_factors.shape[0]):
                rating = user_ratings[user_row][item_row]
                if(not np.isnan(rating)):
                    train(user_row, item_row, rating)
```

Reconstructing data / predicting unseen ratings

- Run *sgd_svd*() and print the updated tables

```
array([[ 1.705,  0.486],            array([[ 2.713,  1.266],
       [ 1.857,  0.484],                   [-1.125,  4.09 ],
       [ 2.373,  1.311],                   [ 1.847,  0.514],
       [ 0.988,  1.495],                   [ 3.09 ,  0.807],
       [ 2.519, -2.088],                   [ 2.079,  2.522]])
       [ 1.868,  1.235],
       [ 1.367,  1.801],
       [ 1.405,  0.628],
       [ 0.133,  3.453],
       [ 1.562,  1.235]]
```

$rating[0][0] \leftarrow 1.705 * 2.713 + 0.486 * 1.266 \approx 5.2$, not 6, but close!

Visual comparison

- Calculate all predicted values
- $(data|prediction)$

	The Call of Cthulhu	Frankenstein	Dracula	Neuromancer	Dune
0	(6.000|5.209)	(0.000|0.029)	(0.000|3.397)	(7.000|5.699)	(nan|4.587)
1	(5.000|5.662)	(0.000|-0.046)	(5.000|3.699)	(6.000|6.208)	(5.000|4.956)
2	(9.000|8.112)	**(nan|2.705)**	(4.000|5.098)	**(nan|8.443)**	(8.000|8.198)
3	(4.000|4.564)	(nan|4.518)	(2.000|2.651)	(5.000|4.262)	(6.000|5.799)
4	(4.000|4.214)	(nan|-9.302)	(4.000|3.425)	(6.000|6.127)	(0.000|0.017)
5	(6.000|6.617)	(3.000|3.004)	(8.000|4.100)	(5.000|6.756)	(7.000|7.003)
6	(nan|5.960)	(6.000|5.735)	(nan|3.473)	(6.000|5.593)	(7.000|7.508)
7	(nan|4.577)	(1.000|0.950)	(1.000|2.918)	(6.000|4.858)	(nan|4.397)
8	(nan|4.494)	(nan|12.716)	(2.000|2.010)	(nan|2.851)	(9.000|8.985)
9	(nan|5.769)	(3.000|3.336)	(4.000|3.523)	(5.000|5.774)	(7.000|6.389)

- For user 2, recommend “Neuromancer” and ignore “Frankenstein”
- Reconstruction is not perfect – multiple reasons (e.g. data shuffling? mini-batches? more factors? more training?)

Other input signals

- We have used ratings
- But this is not the only possible input
- One can use other signals as well
  - User have searched for certain films
  - Users have clicked on certain films
- This is called **implicit feedback** from the user

Pros and cons of CF

- Pros:
  - Minimal knowledge required
  - No need to model the users/items
  - Good enough results in most cases
- Cons:
  - Requires a large number of user feedback data points to bootstrap
  - Requires products to be standardised (exactly the same product)
  - Assumes that prior behaviour determines current behaviour without taking into account contextual information (e.g., time)
  - Cold-start problem
  - Popularity bias

Content-based recommendations (I)

- Latent factors arise only when you actually have a good number of $< user, item, ratings >$ triplets for a user
  - What if the user just joined the website?
- Recommendations based on info on the content on items rather than on other users' opinions/interactions
- Uses a ML algorithm to induce a model of the users preferences from examples based on a featural description of content.
- The system tries to recommend items **similar** to those a given user has liked in the past
- A pure content-based recommender makes recommendations for a user based solely on the profile built up by **analyzing the content** of items which that user has rated in the past.

Content-based recommendations (I)

- Common for recommending text-based products (e.g., web pages)
- The items to recommend are *described* by their associated **features** (e.g., keywords)
- Lazy approach: **User model** structured in a similar way as the content: features/keywords more likely to occur in the preferred documents
  - Text documents recommended based on a comparison between their conten (words appearing) and user model (a set of preferred words)
- The user model can also be a **classifier** (e.g., neural network, linear regressor. . .)

What is content? Item descriptions

- It can be explicit **attributes** or characteristics of the item.
- Each film can have
  - Genre
  - Director
  - Age of intended audience
- It can also be **textual content** (e.g., title, description, table of content…)
  - Several techniques to compute the distance between two textual documents
  - Can use NLP techniques to extract content features
- Can be extracted from the signal itself (e.g., audio, image)

What is content? User descriptions

- But we might have data collected about a user as well
  - Age
  - Sex
  - Country of birth
  - Native language
- All kinds of data
- You can also ask the user questions explicitly
  - What kind of books do you like?
  - What is the maximum price you would pay for a book?
  - E.g,. new user on Netflix is asked which series they've previously liked.

Advantages of the content-based approach

- No need for data on other users
    - Avoiding the cold-start and sparsity problems
- Able to recommend to users with unique tastes
- Able to recommend new and unpopular items
    - No first-rater problem
- Can provide explanations of recommended items by listing content-features that caused an item to be recommended

Disadvantages of the content-based approach

- Requires content that can be encoded as meaningful features
- Some kind of items are not amenable to easy feature extraction methods (e.g,. movies, music)
- Even for text:
  - if you rate positively a page it may not be related to the presence of some keywords
  - not taking into account aesthetics, download time…
- Users' tastes must be represented as a learnable function of these content features
- Hard to exploit quality judgements of other users
- Difficult to implement serendipity
- Easy to overfit (e.g., for a user with few data points, we may "pigeon hole" her)

Content-based filtering

- One can use all features defined on the user and the item
- Create a classifier and do the predictions using the classifier
- We have very few samples
  - So we are going to use a linear classifier

Item feature matrix

Feature 0: First critic score of the book

Feature 1: Second critic score of the book

	Critic0	Critic1
0	0.3	0.9
1	0.9	0.3
2	0.6	0.4
3	0.2	0.1
4	0.7	0.8
5	0.9	0.1

User feature matrix

Feature 0: Male/Female

Feature 1: Over 60

	Gender	Over60
0	1.0	0.0
1	0.0	1.0
2	0.0	0.0
3	1.0	0.0
4	0.0	1.0
5	0.0	0.0
6	0.0	0.0
7	1.0	0.0
8	0.0	1.0
9	1.0	0.0

Combining the features

- We need to build a set of features for training for each person/item combo

	Sex	Over60	key	user_id	Critic0	Critic1	item_id	rating
0	1.0	0.0	0	0	0.3	0.9	0	8.0
1	1.0	0.0	0	0	0.9	0.3	1	2.0
3	1.0	0.0	0	0	0.2	0.1	3	5.0
4	1.0	0.0	0	0	0.7	0.8	4	4.0
0	0.0	1.0	0	1	0.3	0.9	0	3.0
1	0.0	1.0	0	1	0.9	0.3	1	2.0
3	0.0	1.0	0	1	0.2	0.1	3	7.0
4	0.0	1.0	0	1	0.7	0.8	4	7.0
0	0.0	0.0	0	2	0.3	0.9	0	9.0
2	0.0	0.0	0	2	0.6	0.4	2	7.0
3	0.0	0.0	0	2	0.2	0.1	3	8.0
4	0.0	0.0	0	2	0.7	0.8	4	5.0
2	1.0	0.0	0	3	0.6	0.4	2	7.0
3	1.0	0.0	0	3	0.2	0.1	3	8.0
4	1.0	0.0	0	3	0.7	0.8	4	9.0
1	0.0	1.0	0	4	0.9	0.3	1	1.0
2	0.0	1.0	0	4	0.6	0.4	2	8.0
3	0.0	1.0	0	4	0.2	0.1	3	3.0

Test set

- We are looking to predict this:

	Sex	Over60	key	user_id	Critic0	Critic1	item_id	rating
2	1.0	0.0	0	0	0.6	0.4	2	NaN
2	0.0	1.0	0	1	0.6	0.4	2	NaN
1	0.0	0.0	0	2	0.9	0.3	1	NaN
0	1.0	0.0	0	3	0.3	0.9	0	NaN
1	1.0	0.0	0	3	0.9	0.3	1	NaN
0	0.0	1.0	0	4	0.3	0.9	0	NaN
3	0.0	0.0	0	5	0.2	0.1	3	NaN
4	0.0	0.0	0	5	0.7	0.8	4	NaN
2	0.0	0.0	0	6	0.6	0.4	2	NaN
2	0.0	1.0	0	8	0.6	0.4	2	NaN
1	1.0	0.0	0	9	0.9	0.3	1	NaN

Code - Pandas magic!!!

```
user_ratings_df = pd.read_csv("user_ratings.csv")
user_features_df = pd.read_csv("user_features.csv")
item_features_df = pd.read_csv("item_features.csv")


user_features_df["key"] = 0
user_features_df["user_id"] = range(user_features_df.shape[0])
item_features_df["key"] = 0
item_features_df["item_id"] = range(item_features_df.shape[0])

merged_df = pd.merge(user_features_df, item_features_df,
                    left_index=True, on="key")

merged_df["rating"] = map(lambda ids: user_ratings_df.values[ids[1]][ids[2]],
                          merged[["user_id", "item_id"]].itertuples())

train = merged_df.dropna()

test = merged_df[merged.isnull().any(axis=1)]
```

Hybrid systems

- Can we merge the two approaches?
  - Of course we can - various ways of merging
- We will just augment collaborative filtering with standard features for now
- We can add the features as parts of the variables we are going to learn

Regularisation

- We will now add a penalty to features depending on what they are
- Penalty strong for latent features
- But could be the other way around
- Weight decay / l_2 regulariser
  - $w_i = w_i - \alpha(y_p - y)(\phi_i + \lambda w_i)$

Code - predict

```python
def predict_rating(user_id, item_id):
    """ Predict a rating given a user_id and an item_id.
    """
    user_preference = latent_user_preferences[user_id]
    item_preference = latent_item_features[item_id]

    user_score = user_features_weights[user_id].dot(user_features[user_id])
    item_score = item_features_weights[item_id].dot(item_features[item_id])

    return user_preference.dot(item_preference) + user_score + item_score
```

Code - train

```
def train(user_id, item_id, rating,alpha=0.0001, latent_feature_weight_decay=0.1,
          user_weight_decay=0.01, item_weight_decay=0.001):


    prediction_rating = predict_rating(user_id, item_id)
    err =  (prediction_rating - rating)

    user_pref_values = latent_user_preferences[user_id][:]
    latent_user_preferences[user_id] -= alpha *
                                        err * (latent_item_features[item_id] +
                                               latent_feature_weight_decay *
                                               latent_user_preferences[user_id])
    latent_item_features[item_id] -= alpha *
                                     err * (user_pref_values +
                                     latent_feature_weight_decay * latent_item_features[item_id])

    user_features_weights[user_id] -= alpha * err * (user_features[user_id] +
                                                     user_weight_decay * user_features_weights[user_id])
    item_features_weights[item_id] -= alpha * err * (item_features_weights[item_id] +
                                                     item_weight_decay * item_features_weights[item_id])

    return err
```

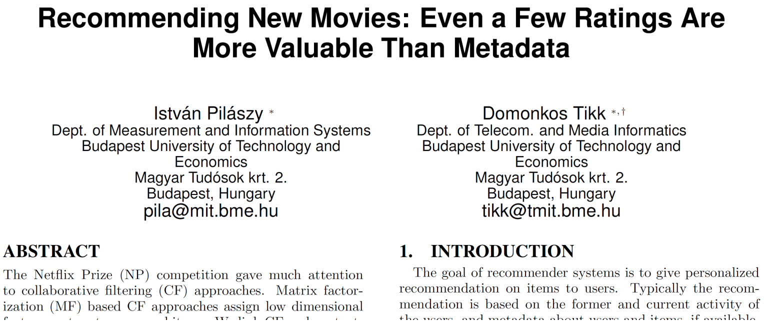
Removing latent factors

- What will happen to the example above if we remove all latent factors?
  - … and base our decisions only on known features
- Factors vs features

Evaluating

- Cross-validation again!
- We have used all the data in our examples
  - Is this correct?
  - We have also evaluated the system using MSE - this is not entirely correct either
- What would be the proper way of evaluating the system?

A word of caution



Recommending New Movies: Even a Few Ratings Are More Valuable Than Metadata

István Pilászy [*]
Dept. of Measurement and Information Systems
Budapest University of Technology and Economics
Magyar Tudósok krt. 2.
Budapest, Hungary
pila@mit.bme.hu

Domonkos Tikk [*,†]
Dept. of Telecom. and Media Informatics
Budapest University of Technology and Economics
Magyar Tudósok krt. 2.
Budapest, Hungary
tikk@tmit.bme.hu

ABSTRACT

The Netflix Prize (NP) competition gave much attention to collaborative filtering (CF) approaches. Matrix factorization (MF) based CF approaches assign low dimensional [illegible]

1. INTRODUCTION

The goal of recommender systems is to give personalized recommendation on items to users. Typically the recommendation is based on the former and current activity of the users, and metadata about users and items, if available

Implicit feedback

- When you are recommending books or food user preferences user likes/dislikes something
- How about if you are recommending
  - News?
  - Adverts?
- You are limited to clicks!
- But they are not proper feedback
  - User might not have seen something

No negative feedback regimes

- Not clicking is not negative feedback
- Create a preference matrix - 1 if user has clicked, 0 if she hasn't
- Weight the importance of each example by $c(i, j) = 1 + \alpha r(i, j)$
  - $r(i, j)$ is the number of clicks

Contextual bandits

- You can personalise the above scenario even further
- Send the user some random examples (e.g. news)
  - With let's say 0.2 probability
- Next week more on this…

Metrics

- Serendipity
  - Our predictions might not be surprising — you don't just need present the user with new items, but also with items that are sufficiently different than rated
- Coverage
  - Some items are not bought often — they should be recommended as well
- We have measured MSE, but ultimately this is a ranking problem
  - We can use ranking metrics, e.g. Mean Average Precision
- Diversity
  - Some items are way too similar to each other — should they be presented to the user?

Conclusion

- We have seen various instances of recommender systems
- Again, there are far more complex models
- But the examples here should have given you a good view about what is out there
- Also, pandas
  - Bring your data to a format that is usable by relevant libraries!