Recommender Systems: An Introduction

Our goals:

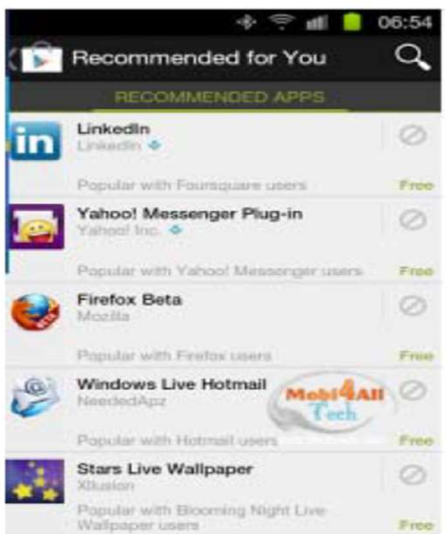
- What are recommender systems?
- User-based collaborative filtering?
- Item-based collaborative filtering?
- The KNN approach for collaborative filtering and its implementation using Python library surprise

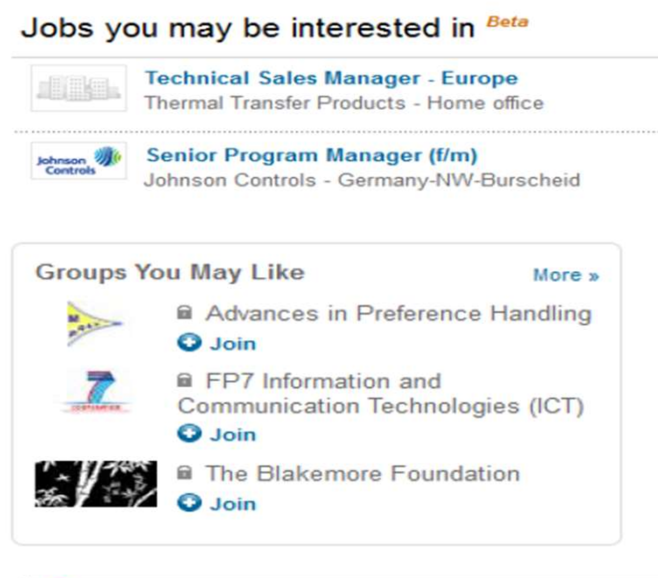
Movie Recommender System: MovieLens Dataset

- The MovieLens dataset is hosted by the GroupLens website. Several versions are available. We will use the MovieLens 100K dataset [Herlocker et al., 1999].

- This dataset is comprised of 100,000 ratings, ranging from 1 to 5 stars, from 943 users on 1682 movies. It has been cleaned up so that each user has rated at least 20 movies.

Recommendation system





Examples: E-commerce sites

- Amazon- People who buy this also buy this or who viewed this also viewed this
- Facebook- Friends recommendation
- Linkedin- Jobs that match you or network recommendation or who viewed this profile also viewed this profile
- Netflix- Movies recommendation
- Google- news recommendation, youtube videos recommendation

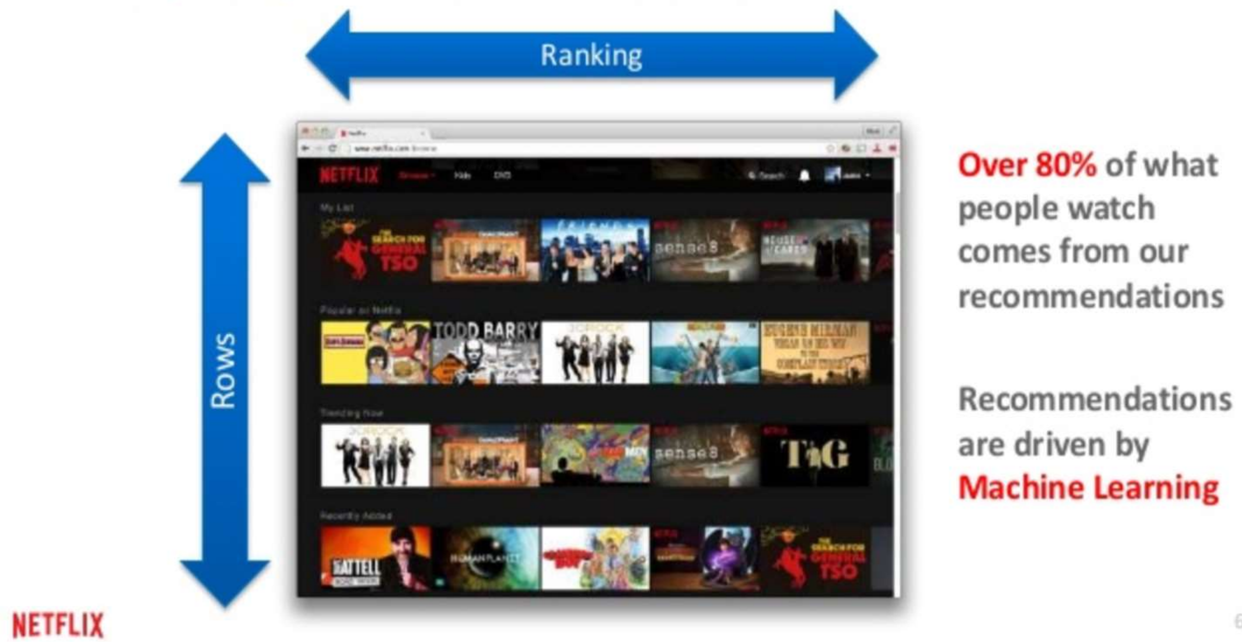

In E-store
Product Dicovery
Search
Recomm. Engines
Loyalty Programs
Bills and mailers
Email Campaigns
Recommendations for new products, up-sell, cross-sell suggestions and product discovery
Matches varying individual customers taste with a huge inventory
Order of search results on the store can be varied for each customer to aid product discovery
Recommendations pertaining to specific program, suggestions for redemption and program upgrades
Effectively targeting abandoned shopping carts with personalized recommendation
Whitespace can be used for cross-sell and up-sell. Usage can be analyzed for recommending plans.

- Recent Research from Monetate reveals that product recommendations can lead to a 70% increase in purchase rates, both in the initial session and in return sessions, and 33% higher average order values.

- A further study from Salesforce found that shoppers who click on product recommendations have 4.5x higher basket rates, make 4.8x more product views per visit, and have a 5x higher per-visit spend.

"More than 80 per cent of the TV shows and movies people watch on Netflix are discovered through the platform's recommendation system."

Read more from ***Josephina Blattmann, UX Planet***
https://uxplanet.org/netflix-binging-on-the-algorithm-a3a74a6c1f59



- Collaborative Filtering: Collaborative Filtering recommends items based on similarity measures between users and/or items. ...

- Content-Based Recommendation: It is supervised machine learning used to induce a classifier to discriminate between interesting and uninteresting items for the user.

1. **Collaborative Filtering** method finds a subset of users who have similar tastes and preferences to the target user and use this subset for offering recommendations.

Basic Assumptions :

- Users with similar interests have common preferences.
- Sufficiently large number of user preferences are available.

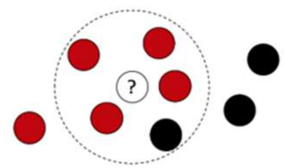
Main Approaches :

- User Based
- Item Based

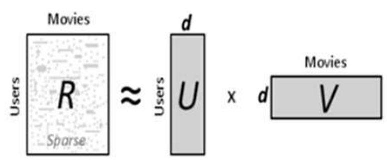
Collaborative Filtering
Techniques

Popular Techniques

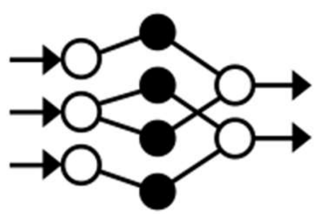
- Nearest Neighbor



- Matrix Factorization



- Deep Learning



Everyday Examples of User-based Collaborative Filtering...

- Bestseller lists
- Top 40 music lists
- The “recent returns” shelf at the library
- Unmarked but well-used paths thru the woods
- ....
- **Common insight**: personal tastes are *correlated*:
    - If Alice and Bob both like X and Alice likes Y then Bob is more likely to like Y
    - especially (perhaps) if Bob knows Alice

High-level Workflow of User-based Collaborative filtering

- A user rates items (e.g., movies, books) to express his or her preferences on the items
- The system treats the ratings as an approximate representation of the user's interest in items
- The system matches this user's ratings with other users' ratings and finds the people with the most similar ratings
- The system recommends items that the similar users have rated highly but not yet been rated by this user

- Example
  - A database of ratings of the current user, Alice, and some other users is given:

	Item1	Item2	Item3	Item4	Item5
Alice	5	3	4	4	?
User1	3	1	2	3	3
User2	4	3	4	3	5
User3	3	3	1	5	4
User4	1	5	5	2	1

  - Determine whether Alice will like or dislike *Item5*, which Alice has not yet rated or seen

Two-steps of User-based Collaborative filtering

- Step 1: Look for people who share the same rating patterns with the given user
- Step 2: Use the ratings from the people found in step 1 to calculate a prediction of a rating by the given user on a product

Collaborative filtering algorithm is processed in item-user rating matrix.

$$R_{mm} = \begin{bmatrix} r_{11} & r_{12} & \cdots & r_{1n} \\ r_{21} & r_{22} & \cdots & r_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ r_{m1} & r_{m2} & \cdots & r_{mn} \end{bmatrix}$$

User-item matrix usually is described as a $m \times n$ ratings matrix *Rmn*, shown as formula (1), where row represents *m* users and column represents *n* items. The element of matrix *rij* means the score rated to the user *i* on the item *j*, which commonly is acquired with the rate of users' interest

User-based correlation-based similarity

$$sim(u,v) = \frac{\sum_{i \in Iuv} (r_{ui} - \bar{r}_u)(r_{vi} - \bar{r}_v)}{\sqrt{\sum_{i \in Iuv} (r_{ui} - \bar{r}_u)^2} \sqrt{\sum_{i \in Iuv} (r_{vi} - \bar{r}_v)^2}}$$

Here $\bar{r}_u$ is the average rating of the u-th user. i.e.

$$\bar{r}_u = \frac{1}{|I_{uv}|} \sum_{i \in Iuv} r_{ui} \;,\quad \bar{r}_v = \frac{1}{|I_{uv}|} \sum_{i \in Iuv} r_{vi}$$

Measuring user similarity (2)

- A popular similarity measure in user-based CF: **Pearson correlation**

  a, b : users

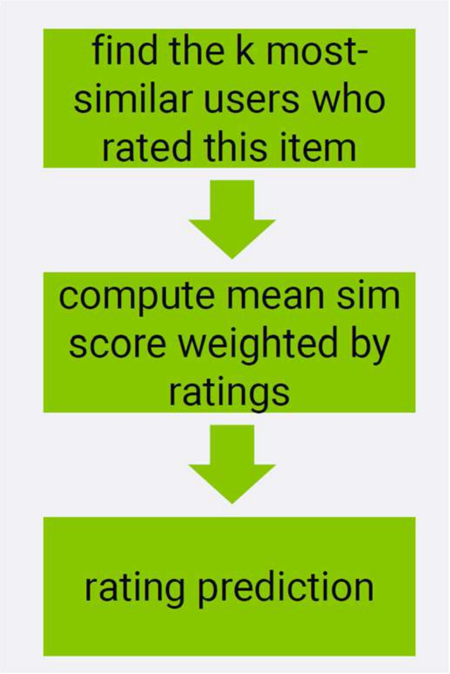
  $r_{a,p}$: rating of user a for item p

  P : set of items, rated both by a and b

  - Possible similarity values between -1 and 1

	Item1	Item2	Item3	Item4	Item5	
Alice	5	3	4	4	?	
User1	3	1	2	3	3	sim = 0,85
User2	4	3	4	3	5	sim = 0,00
User3	3	3	1	5	4	sim = 0,70
User4	1	5	5	2	1	sim = -0,79

User-based KNN



Neighborhood formation phase

- Let the record (or profile) of the target user be **u** (represented as a vector), and the record of another user be **v** ($\mathbf{v} \in T$).
- The similarity between the target user, **u**, and a neighbor, **v**, can be calculated using the **Pearson's correlation coefficient**:

$$sim(\mathbf{u}, \mathbf{v}) = \frac{\sum_{i \in C} (r_{\mathbf{u},i} - \bar{r}_{\mathbf{u}})(r_{\mathbf{v},i} - \bar{r}_{\mathbf{v}})}{\sqrt{\sum_{i \in C} (r_{\mathbf{u},i} - \bar{r}_{\mathbf{u}})^2} \sqrt{\sum_{i \in C} (r_{\mathbf{v},i} - \bar{r}_{\mathbf{v}})^2}},$$

Recommendation Phase (optional)

- Use the following formula to compute the rating prediction of item i for target user $\mathbf{u}$

$$p(\mathbf{u},i)=\bar{r}_{\mathbf{u}}+\frac{\sum_{\mathbf{v}\in V} sim(\mathbf{u},\mathbf{v})\times(r_{\mathbf{v},i}-\bar{r}_{\mathbf{v}})}{\sum_{\mathbf{v}\in V}\left|sim(\mathbf{u},\mathbf{v})\right|}$$

where V is the set of k similar users, p(u,i) is the prediction for the active user u for item i, $r_{\mathbf{v},i}$ is the rating of user $\mathbf{v}$ given to item i,

Recommendation Phase (cont'd)

- A common prediction function:

$$p(\mathbf{u},i) = \bar{r}_{\mathbf{u}} + \frac{\sum_{\mathbf{v}\in V} sim(\mathbf{u},\mathbf{v}) \times (r_{\mathbf{v},i} - \bar{r}_{\mathbf{v}})}{\sum_{\mathbf{v}\in V} |sim(\mathbf{u},\mathbf{v})|}$$

- Calculate, whether the neighbors' ratings for the unseen item i are higher or lower than their average
- Combine the rating differences – use the similarity as a weight
- Add/subtract the neighbors' bias from the active user's average and use this as a prediction

Improving the metrics / prediction function

- Not all neighbor ratings might be equally "valuable"
  - Agreement on commonly liked items is not so informative as agreement on controversial items
- Value of number of co-rated items
  - Use "significance weighting", by e.g., linearly reducing the weight when the number of co-rated items is low
- Case amplification
  - Intuition: Give more weight to "very similar" neighbors, i.e., where the similarity value is close to 1.
- Neighborhood selection
  - Use similarity threshold or fixed number of neighbors

Item-based CF

- The item-based approach works by comparing items based on their pattern of ratings across users. The similarity of items *i* and *j* is computed as follows:

$$sim(i,j) = \frac{\sum_{\mathbf{u}\in U}(r_{\mathbf{u},i} - \bar{r}_{\mathbf{u}})(r_{\mathbf{u},j} - \bar{r}_{\mathbf{u}})}{\sqrt{\sum_{\mathbf{u}\in U}(r_{\mathbf{u},i} - \bar{r}_{\mathbf{u}})^2}\sqrt{\sum_{\mathbf{u}\in U}(r_{\mathbf{u},j} - \bar{r}_{\mathbf{u}})^2}}$$

Another popular measure of similarity

- **Cosine Similarity**

Here, two items i_p and i_q are considered as two column vectors in the user ratings matrix R. The similarity between items is measured by computing the cosine of these two vectors.

$$sim(i_p, i_q) = \cos(i_p, i_q) = \frac{i_p \bullet i_q}{\sqrt{\| i_p \| * \| i_q \|}}$$

Here, "•" denotes the dot-product of two vectors.

Item-based KNN

Item-based CF: Recommendation phase

- After computing the similarity between items we select a set of k most similar items to the target item and generate a predicted value of user $\mathbf{u}$'s rating

$$p(\mathbf{u}, i) = \frac{\sum_{j \in J} r_{\mathbf{u},j} \times sim(i, j)}{\sum_{j \in J} sim(i, j)}$$

where J is the set of k similar items users

items	1	2	3	4	5	6	7	8	9	10	11	12
1	1		3		2.6	5			5		4	
2			5	4			4			2	1	3
3	2	4		1	2		3		4	3	5	
4		2	4		5			4			2	
5			4	3	4	2					2	5
6	1		3		3			2			4	

similarity

$s_{13} = 0.2$

$s_{16} = 0.3$

weighted average

$$\frac{0.2 \cdot 2 + 0.3 \cdot 3}{0.2 + 0.3} = 2.6$$

- unknown rating

- rating between 1 to 5

Item-based

Richard	5	1	4
Mary		2	5
Steve	4	3	2

$$? = \frac{\sum_{i=1}^{c} R_{ui} \times sim(t,i)}{\sum_{i=1}^{c} sim(t,i)} = \frac{2 \times -0.8 + 5 \times -0.3}{-0.8 + -0.3}$$

Pre-processing for item-based filtering

- Item-based filtering does not solve the scalability problem itself
- Pre-processing approach by Amazon.com (in 2003)
  - Calculate all pair-wise item similarities in advance
  - The neighborhood to be used at run-time is typically rather small, because only items are taken into account which the user has rated
  - Item similarities are supposed to be more stable than user similarities
- Memory requirements
  - Up to N^2 pair-wise similarities to be memorized (N = number of items) in theory
  - In practice, this is significantly lower (items with no co-ratings)
  - Further reductions possible
    - Minimum threshold for co-ratings
    - Limit the neighborhood size (might affect recommendation accuracy)

Data sparsity problems

- Cold start problem
  - How to recommend new items? What to recommend to new users?
- Straightforward approaches
  - Ask/force users to rate a set of items
  - Use another method (e.g., content-based, demographic or simply non-personalized) in the initial phase
  - Default voting: assign default values to items that only one of the two users to be compared has rated (Breese et al. 1998)
- Alternatives
  - Use better algorithms (beyond nearest-neighbor approaches)

Recommender System made easy with Scikit-Surprise

Installation

With pip (you'll need numpy, and a C compiler. Windows users might prefer using conda):

```
$ pip install numpy
$ pip install scikit-surprise
```

With conda:

```
$ conda install -c conda-forge scikit-surprise
```

Overview

Surprise is a Python scikit building and analyzing recommender systems that deal with explicit rating data.

Surprise **was designed with the following purposes in mind**:

- Give users perfect control over their experiments. To this end, a strong emphasis is laid on documentation, which we have tried to make as clear and precise as possible by pointing out every detail of the algorithms.
- Alleviate the pain of Dataset handling. Users can use both *built-in* datasets (Movielens, Jester), and their own *custom* datasets.
- Provide various ready-to-use prediction algorithms such as baseline algorithms, neighborhood methods, matrix factorization-based (SVD, PMF, SVD++, NMF), and many others. Also, various similarity measures (cosine, MSD, pearson...) are built-in.
- Make it easy to implement new algorithm ideas.
- Provide tools to evaluate, analyse and compare the algorithms performance. Cross-validation procedures can be run very easily using powerful CV iterators (inspired by scikit-learn excellent tools), as well as exhaustive search over a set of parameters.

MovieLens 100K Dataset

MovieLens 100K movie ratings. Stable benchmark dataset. 100,000 ratings from 1000 users on 1700 movies. Released 4/1998.

- README.txt
- ml-100k.zip (size: 5 MB, checksum)
- Index of unzipped files

Permalink: https://grouplens.org/datasets/movielens/100k/

- Neighborhood size is typically limited to a specific size
- An analysis of the MovieLens dataset indicates that "in most real-world situations, a neighborhood of 20 to 50 neighbors seems reasonable" (Herlocker et al. 2002)

Movie Recommender System: MovieLens Dataset

- The MovieLens dataset is hosted by the GroupLens website. Several versions are available. We will use the MovieLens 100K dataset [Herlocker et al., 1999].

- This dataset is comprised of 100,000 ratings, ranging from 1 to 5 stars, from 943 users on 1682 movies. It has been cleaned up so that each user has rated at least 20 movies.

KNNBasic in the Surprise library

class **surprise.prediction_algorithms.knns.KNNBasic**(*k=40, min_k=1, sim_options={}, verbose=True, **kwargs*)

Bases: `surprise.prediction_algorithms.knns.SymmetricAlgo`

A basic collaborative filtering algorithm.

The prediction $\hat{r}_{ui}$ is set as:

$$\hat{r}_{ui} = \frac{\sum\limits_{v \in N_i^k(u)} \text{sim}(u, v) \cdot r_{vi}}{\sum\limits_{v \in N_i^k(u)} \text{sim}(u, v)}$$

KNNWithMeans in the Surprise library

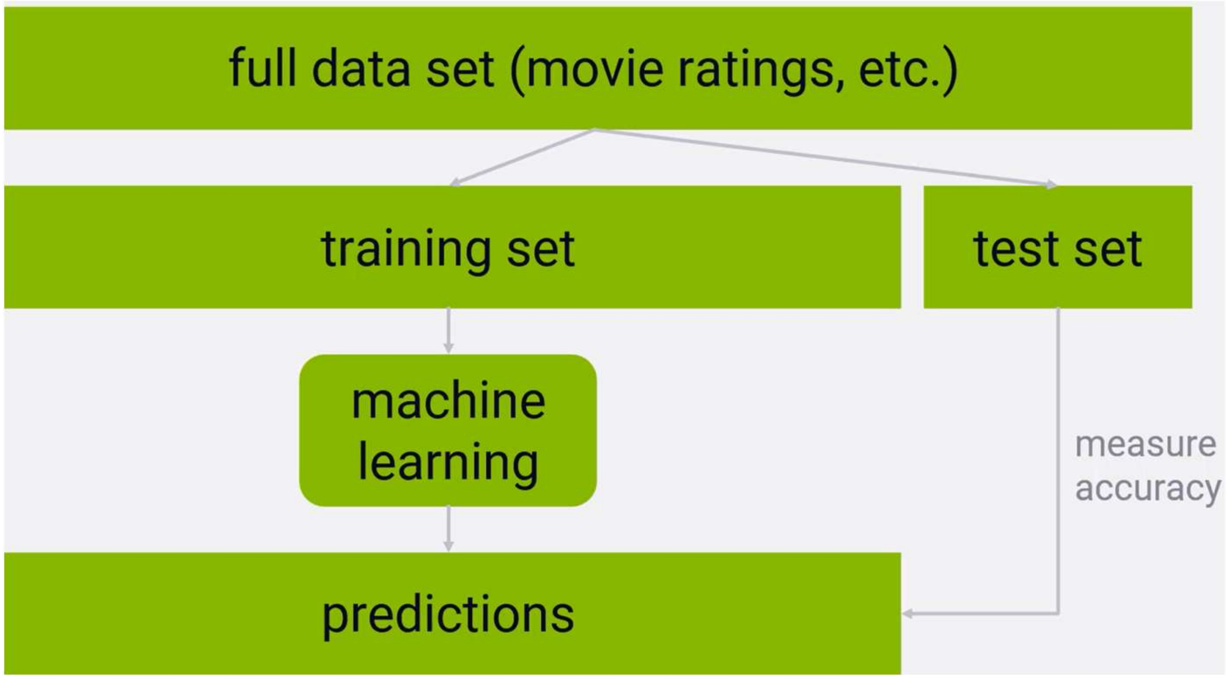
class **surprise.prediction_algorithms.knns.KNNWithMeans**(*k=40, min_k=1, sim_options={}, verbose=True, **kwargs*)

Bases: `surprise.prediction_algorithms.knns.SymmetricAlgo`

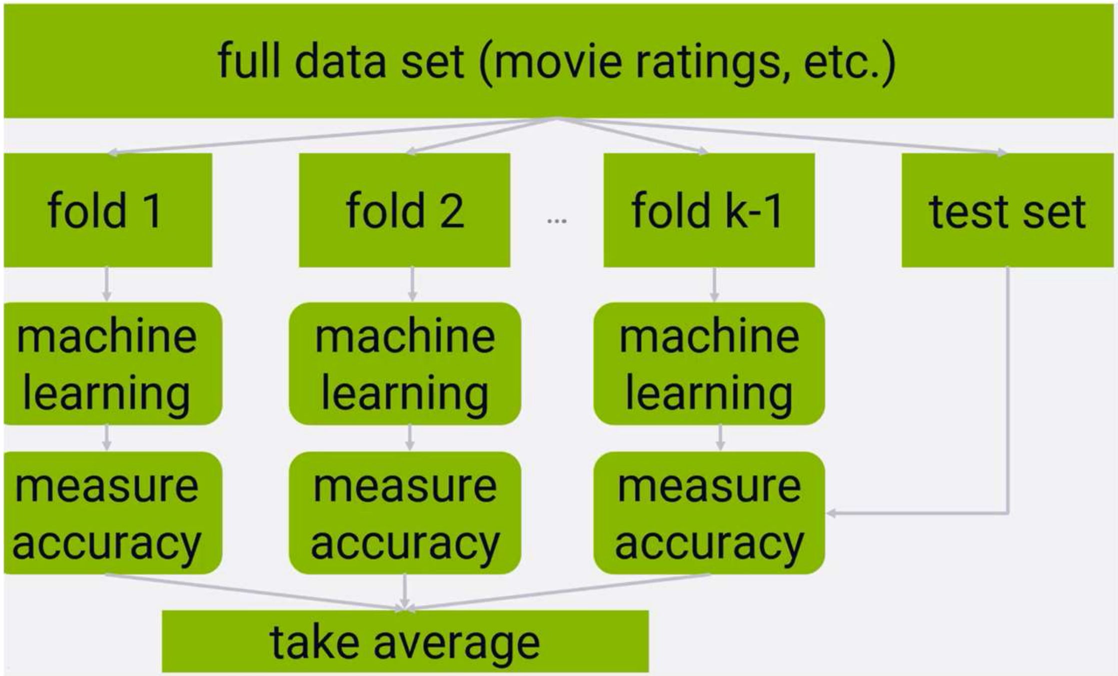
A basic collaborative filtering algorithm, taking into account the mean ratings of each user.

The prediction $\hat{r}_{ui}$ is set as:

$$\hat{r}_{ui} = \mu_u + \frac{\sum\limits_{v \in N_i^k(u)} \text{sim}(u, v) \cdot (r_{vi} - \mu_v)}{\sum\limits_{v \in N_i^k(u)} \text{sim}(u, v)}$$


full data set (movie ratings, etc.)
training set
test set
machine learning
predictions
measure accuracy

K-fold cross-validation

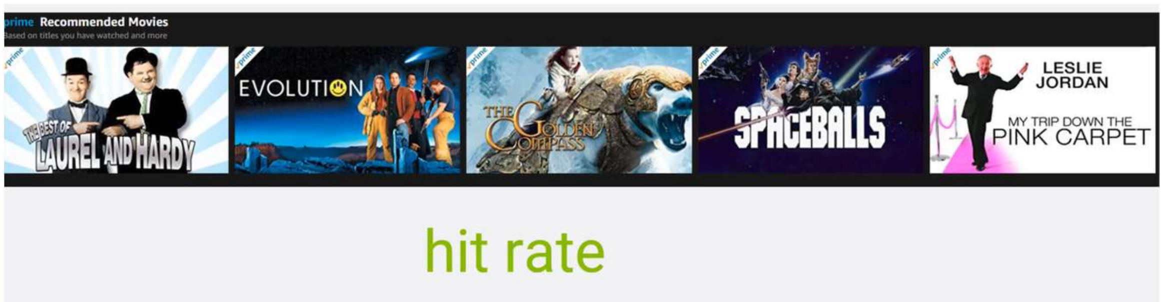


Evaluation

Real evaluation metrics:

- User satisfaction!
- Purchase (like) of recommended products
- Increment in sales?

Evaluating top-n recommenders



Average reciprocal hot rate (ARHR)

$$\frac{\sum_{i=1}^{n} \frac{1}{rank_i}}{Users}$$

rank	reciprocal rank
3	1/3
2	1/2
1	1

Other criteria

- Coverage

    % of <user, item> pairs that can be predicted

- Diversity

    $$(1 - S)$$

    S = avg similarity between recommendation pairs

- Novelty

Mean popularity of recommended items