Algorithms for Data Science
Item Recommendation

Silviu Maniu

October 7th, 2022

M2 Data Science

Table of contents

Recommendations

Content-based Recommendation

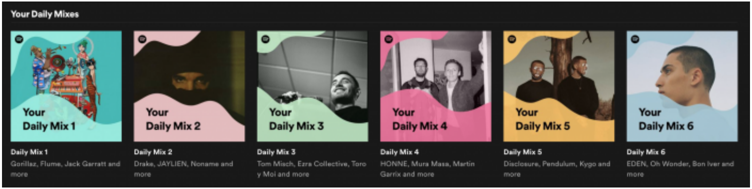
Collaborative Filtering

Recommendations in the Web age

Traditionally: Shelf space, newspaper ads, etc. (scarcity)

Now: Almost zero cost of information about products (user behaviour)

- abundance of choice
- more niche preferences → better filter → better recommendation engines



Types of Recommendations

Editorial / hand curated

- favourites, bookmarks
- curated lists

Simple aggregates

- top 10 lists
- “most popular”, “recent”

Personalized

- Netflix, Spotify, etc.

Formal Model

Set of **customers** X, set of **items** S

Utility function / matrix $u : X \times S \to R$

- R totally ordered set of **ratings** ($0 - 5$ stars, grades, percentages)

u	film1	film 2	film3	film4
cust1	0.9		0.3	
cust2		0.75		0.4
cust3	0.1		1	
cust4			0.4	

Challenges

Gathering ratings for the utility:

- explicit: ask people to rate
- implicit: learn from user actions (but issues with low ratings)

Extrapolate unknown ratings

- u is sparse (most people don't rate everything)
- cold start issues

Approaches

Three main approaches:

- **content-based**
- **collaborative filtering**
- **latent factors** (not covered)

Table of contents

Recommendations

Content-based Recommendation

Collaborative Filtering

Content-based Recommendations

Principle – recommend items to an user that are similar to other items highly rated by them

Applications

- *books, movies, music*: same actors/artists, same genre, etc.
- *products*: recommend other products that have the same characteristics

Main "workflow":

- aggregate item profiles $\rightarrow$ aggregate user profile $\rightarrow$ match other items

Item profiles

Item profile for each item – set or vector of features

- important words in document
- "one-hot" encoding of authors, titles, actors, ...
- embeddings

Similar to the information retrieval setting

- features that are present in fewer items are more important
- combine **feature frequency** with **inverse document frequency**

TF-IDF

Term Frequency – Inverse Document Frequency: heuristic from text mining

- in our case term is feature, document is item

Frequency of feature i in item j, f_{ij}

$$\mathrm{TF}_{ij} = \frac{f_{ij}}{\max_k f_{kj}}$$

Inverse frequency of feature i, n_i in total items N

$$\mathrm{IDF}_i = \log \frac{N}{n_i}$$

Item Profile

TF-IDF score for every pair of feature-item

$$w_{ij} = \mathrm{TF}_{ij} \times \mathrm{IDF}_i$$

Item profile: set of features having highest tf-idf scores

User Profile

User profile – aggregation of item profile attached to an user

- weighted average, difference from average, etc.

Recommendation

Prediction for user x and item i (cosine similarity)

$$s(x, i) = \cos \frac{\langle x, i \rangle}{\|x\| \; \|i\|}$$

- recommend top-k items by s scores
- recommend items above a similarity threshold

Pros and Cons

Pros

- not reliant on other users
- can predict to niche users, can predict new items
- can provide **explanations**

Cons

- finding the good features is hard
- hard to recommend to new users
- cannot recommend items outside user's content profile

Table of contents

Recommendations

Content-based Recommendation

Collaborative Filtering

Collaborative Filtering

Given an user x

- find a set of N other users having similar ratings
- "fill" x's rating based on the ratings of the other users

User-user collaborative filtering

Finding Similar Users

Vector r_x, r_y of user ratings

$$r_x = \begin{pmatrix} 1 & \_ & \_ & 1 & 3 \end{pmatrix} \quad r_y = \begin{pmatrix} 1 & \_ & 2 & 2 & \_ \end{pmatrix}$$

Jaccard similarity

- consider the vector as set of item rated; grades are ignored
- $\mathrm{sim}(x, y) = \frac{|\{1,4,5\} \cap \{1,3,4\}|}{|\{1,4,5\} \cup \{1,3,4\}|} = 0.5$

Cosine similarity

- measures the "angle" between vectors as similarity, 0 means complete de-correlation, -1 complete dissimilarity, 1 similarity
- assumes missing ratings are bad ratings
- $\mathrm{sim}(x, y) = \frac{\langle r_x, r_y \rangle}{\|r_x\| \cdot \|r_y\|} \approx 0.3$

Others: Pearson correlation coefficient, …

Predicting Ratings

r_x user x ratings, N set of k most similar users

Predicting missing ratings of an item i:

- $r_{xi} = \frac{\sum_{y \in N} r_y i}{k}$ (average)
- $r_{xi} = \frac{\sum_{y \in N} \text{sim}(x,y) \cdot r_{yi}}{\sum_{y \in N} \text{sim}(x,y)}$ (weighted average)
- not the only choices!

Item-Item Collaborative Filtering

Item-item view:

- for item i, find other similar items (similarity=same ratings by users)
- estimate rating for i based on ratings of similar items, $N(i;x)$ set of items rated by x similar to i
- can use same similarity metrics and prediction functions

$$r_{xi} = \frac{\sum_{j \in N(i;x)} \mathrm{sim}(i,j) \cdot r_{xj}}{\sum_{j \in N(i;x)} \mathrm{sim}(i,j)}$$

In Practice

The score is taken as compared to the **average scores in the data**

- μ overall item rating, b_x, b_i - average rating deviation from μ
- baseline estimator for $r_{xi} = \mu + b_x + b_y$

$$r_{xi} = b_{xi} + \frac{\sum_{j \in N(i;x)} \text{sim}(i,j)(r_{xj} - b_{xj})}{\sum_{j \in N(i;x)} \text{sim}(i,j)}$$

Hybrid methods

- combine different Recommenders
- combine content-based approach into item-item CF

Pros and Cons

Pros

- no feature selection needed; only ratings are sufficient

Cons

- cold start problem: needs enough users/items
- sparsity problem: hard to find users having rated same item
- popularity: unique tastes have sparsity problem; tens to recommend popular items

Evaluation

Train-test:

- remove a subset of ratings from a subset of users (same items)
- try to “guess” them

Measures:

- root mean square error $= \sqrt{\sum_{xi}(r_{xi} - r^*_{xi})^2}$
- precision at k (p@k): percent in top-k
- Spearman rank correlation between ideal and user’s rankings
- 0-1 model: number of items for which prediction can be made (coverage), predicting rating not too far from ideal (precision), can use concept of false positive/negative

Computation Issues

Number of items I, number of users U

Finding k most similar items $\mathcal{O}(kUI)$ – too expensive!

- need to pre-compute for all users if possible

How?

- locality sensitive hashing
- clustering
- dimensionality reduction

Acknowledgments

The contents follows Chapter 9 of [Leskovec et al., 2020].

References i

Leskovec, J., Rajaraman, A., and Ullman, J. (2020).
Mining of Massive Datasets.
Cambridge University Press.