Recommender Systems Challenge 2017

Tommaso Scarlatti
897651

Overview

- **Application domain:** music streaming service, where users listen to tracks and create playlists
- **Goal:** discover which track a user will likely add to a playlist
- **Evaluation:** MAP@5 (Mean Average Precision)

$$AP@5 = \sum_{k=1}^{5} \frac{P(k) \cdot \mathrm{rel}(k)}{\min(m, 5)}$$

$$MAP@5 = \frac{\sum_{u=1}^{N} AP@5_u}{N}$$

- **57.561 | 10.000** playlists | target
- **100.000 | 32.195** tracks | target
- **1.040.522** interactions

A first approach: Top-N recommender

- First naive attempt: a **non-personalized** recommender system
- Count each distinct track occurrence in *train_final.csv*
- Select the top 5 popular tracks
- Recommend these 5 tracks for all the target playlist

	playlist_id	track_ids
0	10024884	1563309 1363985 3705881 1595978 3166665
1	10624787	1563309 1363985 3705881 1595978 3166665
2	4891851	1563309 1363985 3705881 1595978 3166665
3	4267369	1563309 1363985 3705881 1595978 3166665
4	65078	1563309 1363985 3705881 1595978 3166665
5	10637124	1563309 1363985 3705881 1595978 3166665
6	3223162	1563309 1363985 3705881 1595978 3166665

MAP@5 = 0.001

Data Preprocessing

Pandas

- Read/write .csv
- Manage datasets efficiently
- Build up a validation set

Sklearn

The module *sklearn.preprocessing* was used to **binarize** the input data and then **normalize** the matrices.

train_final.csv

```
playlist_id track_id
3271849 2801526
5616275 727878
11267488	2805283
10103900	1515105
3836898 2945623
5270369 2821391
3794808 1166185
7908370 2498280
11460733	282687
886396	863177
5758965 676462
```

Grouped by playlist

```
playlist_id
7569		[2634448, 2693660, 222710
7614		[2285204, 1384962, 371143
7641		[3825235, 257968, 3711129
7692		[3036454, 1439032, 302730
7816		[2305305, 1039409, 267481
7912		[126424, 218197, 3767380,
```

CSR matrix

```
(45646, 50383)	1
(45646, 87184)	1
(45646, 94907)	1
(45647, 15126)	1
(45648, 9984) 1
(45648, 46177)	1
(45648, 38592)	1
(45648, 1632) 1
(45648, 69508)	1
(45648, 30378)	1
```

Global strategies

Indices

- known_indices ✔
- non_target_indices ✔
- owner_indices ✖

Recommendations

- One playlist per cycle to avoid computation of large dense matrices

KNN

- K-nearest-neighbours used in every similarity matrix

Matrices

- Sparse csr matrix to speed up the dot product

Attributes

Playlists

- Only *owner_id* considered with no success
- Tracks of URM used as attributes to compute the UCM

Tracks

- **Used:** *artist_id*, *album*, *tags*
- **Unused:** *duration*, *playcount*

Item-based recommender

- Item similarity matrix

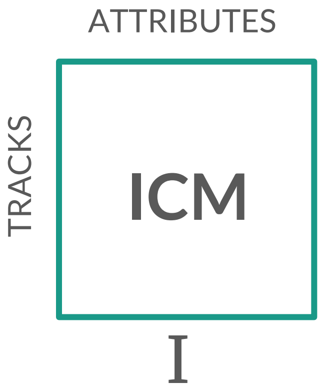
$$S = II^T$$

- Recommendation: top 5 for similarity

$$\tilde{R} = RS$$

- **MAP@5**

0.01122	0.05524	0.07695
NO TF-IDF	WITH TF-IDF	TF-IDF + L2-NORM



Collaborative filtering

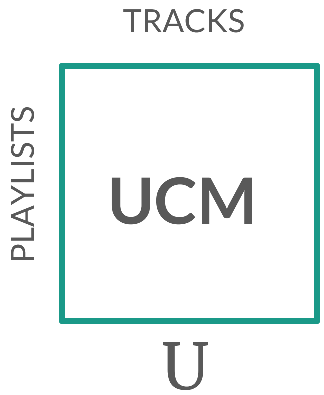
- User content matrix (build from URM)

$$U = tfidf(R^T)^T$$

- Similarity matrix

$$S = U^T U$$

- **MAP@5** = 0.06653



Average similarity recommender

- Added **l2 normalization** everywhere
- Weighted sum of S_ICM and S_UCM
- Much relevant S_ICM
- $\alpha \simeq 0.65$
- **MAP@5** = 0.09205

Combining predictions

$$\alpha \; \boxed{R_u \text{ ICM}} + (1-\alpha) \; \boxed{R_u \text{ UCM}}$$

SVD - Singular Value Decomposition

- New similarity matrix with ***k*** = 1000 latent factors and **knn** = 250
- Computationally expensive → *scipy.sparse.linalg.svds*
- Little improvements combining it with other recommenders
- **MAP@5** = 0.04553

$$\boxed{ICM} = \boxed{U}\,\boxed{S}\,\boxed{V^T}$$

Slim BPR - Bayesian Personalized Ranking

- Mainly based on the code on the jupyter notebook
- **lil_matrix** to incrementally build the similarity matrix
- Added positive and negative item **regularization** terms
- Added **knn**
- **Positive interactions** = number of zeros

- learning rate = 0.1
- epochs = 1
- pir = 1.0
- nir = 1.0

Matrix Factorization

- Code on the jupyter notebook
- Very poor results even with several epochs
- Added regularization terms with no success
- **MAP@5** like a top popular recommender

Round robin & Ranking average

- Combines recommendation of: item-based, collaborative filtering and SLIM
- Pick tracks according to their ranking

Round robin

- Tested in 3 different modes:
  “Standard”, “Jump”, “Mono”

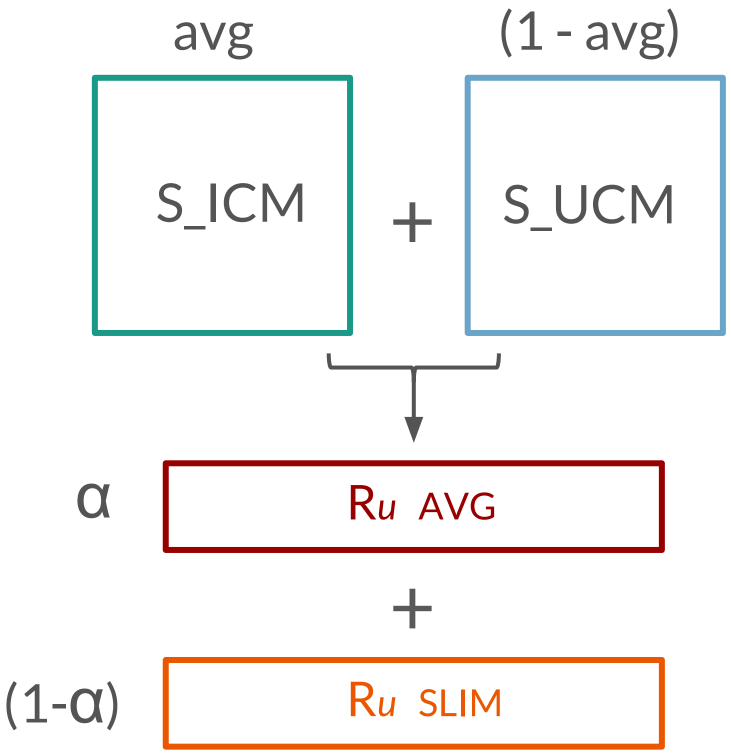
- **MAP@5**: no improvements

Ranking average

- Compute the ranking average for each track and pick the top 5 according to this value

Hybrid Recommender (best solution)

- Merging models + combining predictions
- Merge similarity matrices derived from the ICM and the UCM
- Combine prediction with Slim BPR
- avg = 0.74   α = 0.20
- **MAP@5** = 0.10205



Code organization

Recommenders

- *RandomRec*
- *TopPopRec*
- *ItemBasedRec*
- *CollabFilteringRec*
- *SVDRec*
- *SimilarityAvgRec*
- *SlimBPRRec*
- *RoundRobinRec*
- *HybridRec*

Evaluation

- *Evaluator*
    - *split (URM)*
    - *map5 (pred, test)*

Support

- *Builder*
- *Utils*
- *Test*
- *Run*
- *SlimBPR*
- *MFBPR*

Recommender System Challenge 2017

POLITECNICO
MILANO 1863

Thank you for your attention.

Tommaso Scarlatti