Artist-driven layering and user’s behaviour impact on recommendations in a playlist continuation scenario

Creamy Fireflies

RecSys Challenge Workshop 2018

The Creamy Fireflies Team

We are a team of six **MSc students** from Politecnico di Milano:

- Sebastiano Antenucci
- Simone Boglio
- Emanuele Chioso
- Ervin Dervishaj
- Shuwen Kang
- Tommaso Scarlatti

and one PhD candidate:

- Maurizio Ferrari Dacrema

Spotify RecSys Challenge 2018

Spotify® RecSys Challenge 2018

- Music recommendation, automatic playlist continuation
- Recommend 500 tracks for 10K playlists, divided in 10 categories

Tracks

- *Main:* only data provided by Spotify through the MPD
- *Creative*: external, public freely available data allowed

Metrics

- *R-precision*
- *NDCG*
- *Recommender Song Clicks*

The cold-start problem

For playlists with no interactions we built a feature space starting from **playlists titles**:

1. Removing spaces from titles made by only separated single letters

w o r k o u t —> workout

2. Elimination of uncommon characters
3. Extraction and reconciliation of dates
4. Apply Lancaster and Porter stemming to generate tokens

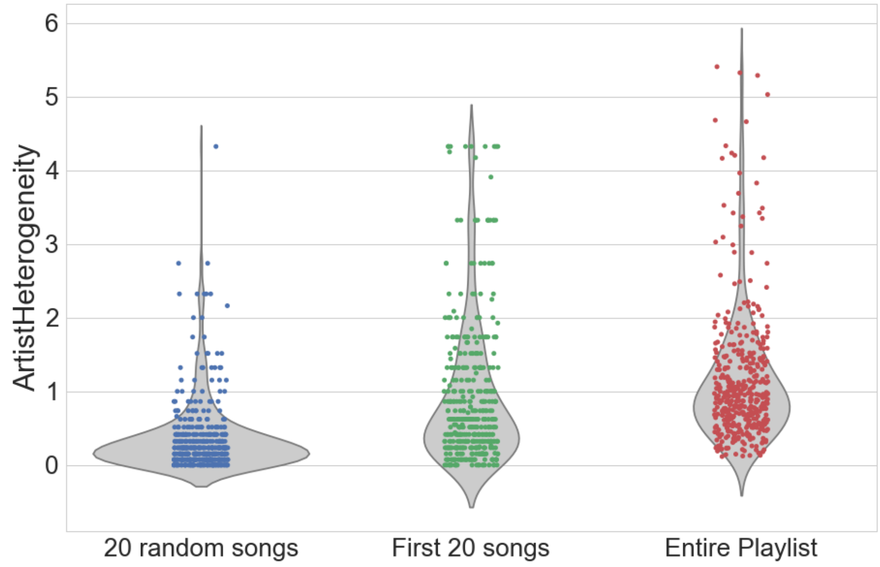
Preprocessing

Artist Heterogeneity

- Playlists sometimes exhibit a common underlying structure due to the way a user fills them:
  - Adding tracks from same album
  - Adding tracks from same artist (and featuring)
  - Creating a playlist with many different artists at first and add tracks of the same artists later on

Preprocessing

Artist Heterogeneity



$$ArH_p = \log_2\left(\frac{|\,uniqueTracks_p\,|}{|\,uniqueArtists_p\,|}\right)$$

Algorithms

- Personalised Top Popular
  - Track based
  - Album based

- Collaborative Filtering - Track based
- Collaborative Filtering - Playlist based

- Content Based Filtering - Track based
- Content Based Filtering - Playlist based
  - Track features
  - Playlist names

Personalized Top Popular

- For playlists with just **one track**, we applied a personalized top popular algorithm at two levels:

  - ***Track-based****:* compute top popular over all the playlists that contain that track
  - ***Album-based:*** given the album of the track, compute top popular over all the playlists that contain the tracks of the album

Collaborative Filtering

Track based

BM25 normalization

tracks similarity

$$s_{ij} = r_i * r_j$$

score prediction

$$r_{ui} = \sum_{j \epsilon I(u)}^{KNN} r_{uj} * (s_{ji})^p$$

Playlist based

playlists similarity (Tversky)

$$s_{ij} = \frac{r_i * r_j}{\alpha(|r_i| - r_i * r_j) + \beta(|r_j| - r_i * r_j) + r_i * r_j + h}$$

score prediction

$$r_{ui} = \sum_{j \epsilon I(u)}^{KNN} r_{uj} * (s_{ji})^p$$

Content Based Filtering - Track based

BM25 normalization

↓

tracks similarity

$$s_{ij} = f_i * f_j$$

↓

score prediction

$$r_{ui} = \sum_{j \epsilon I(u)}^{KNN} r_{uj} * (s_{ji})^p$$

FEATURES

TRACKS

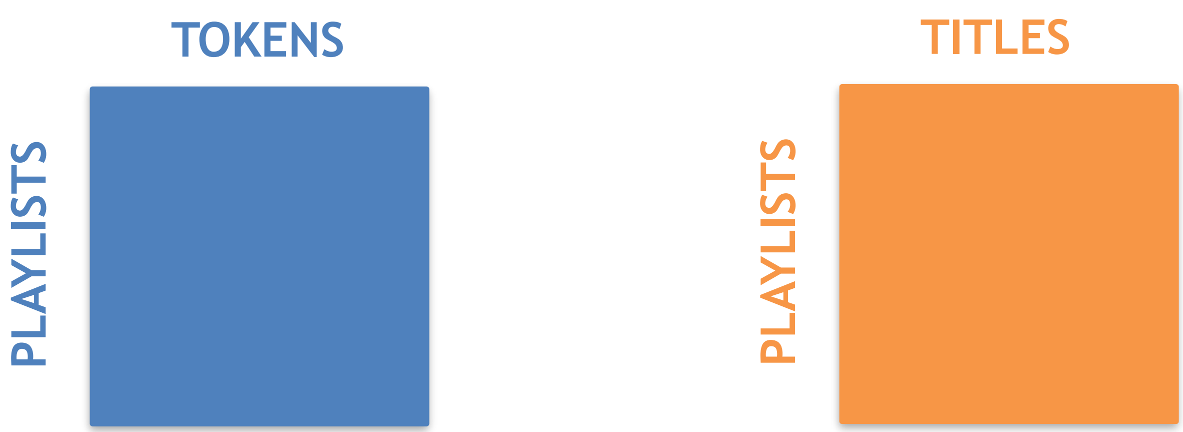
ICM

FEATURES = ALBUMS + ARTISTS

Content Based Filtering - Playlist based

- Two different approaches starting from the playlists title:

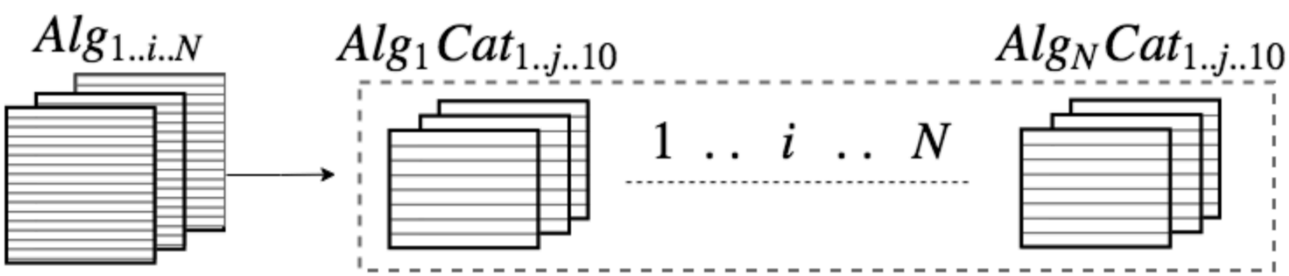
  1. CBF based on tokens extracted in preprocessing phase
  2. CBF based on an exact title match



Ensemble

- Different algorithms are better suited for subsets of playlists with specific characteristics
  - Content-based: short playlists with similar features
  - Collaborative filtering: long and heterogeneous playlists

- Weighted sum of the predictions of each algorithm for each category:

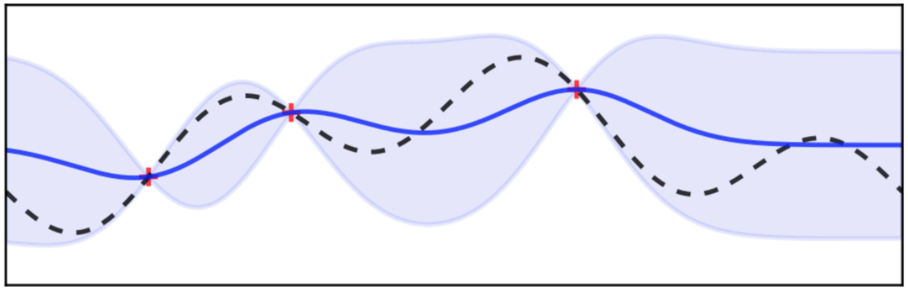


Parameters tuning

- For each algorithm and each category:
  - *k*-nearest neighbours
  - power *p* for similarity values
  - Tversky coefficients
  - shrink term *h*

Ensemble:

- Bayesian optimization
- NDCG



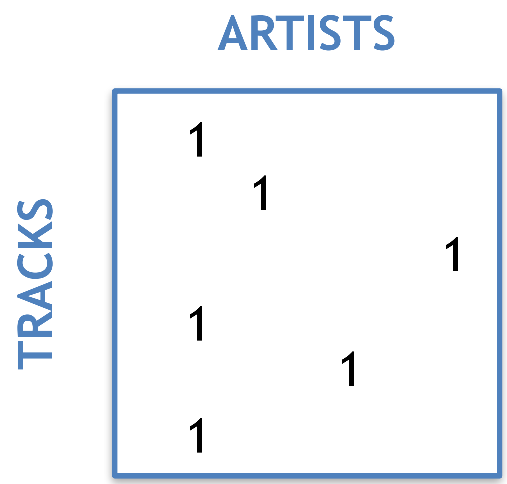
External datasets

- We tried several external datasets to enrich the *MPD*
- *We* used **Spotify API** to retrieve tracks popularity and audio features such as: loudness, danceability, energy, tempo…

Dataset Name	Data Type	Year
#nowplaying music [3]	Listening behavior	2018
#nowplaying playlists	Playlist	2015
MLHD [4]	Listening behavior	2017
FMA [5]	Audio Features	2017
MSD [6]	Audio Features	2011
Spotify API [7]	Audio Features, popularity	2018

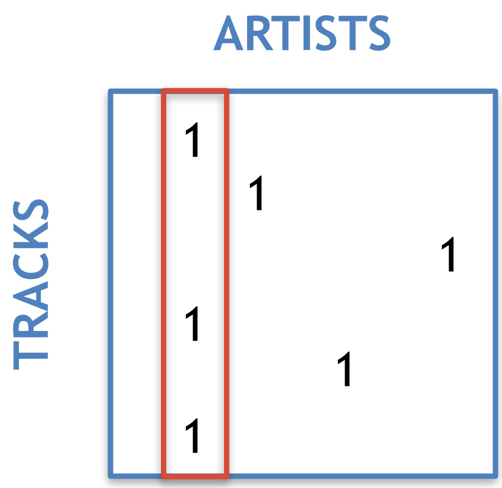
Creative track

- CBF which is able to adjust the artist-based track recommendation using 10 additional features
- Track-track similarity computed using only artists as features cannot distinguish tracks belonging to same artist



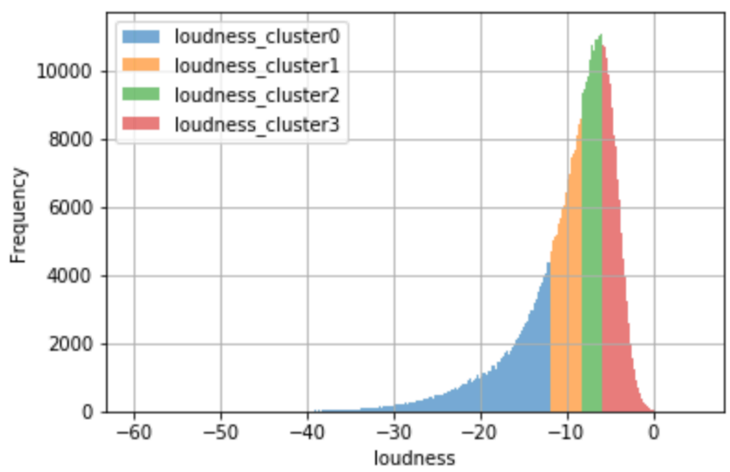
Creative track

- CBF which is able to adjust the artist-based track recommendation using 10 additional features
- Track-track similarity computed using only artists as features cannot distinguish tracks belonging to same artist



Creative track - Artist layering

1. Split tracks into 4 clusters with equal number of elements for each feature
2. Considering feature clusters as a 3rd dimension, split the dense ICM into 4 sparse layers
3. Concatenate 4 layers of sparse matrices horizontally in order to create a final sparse ICM and apply CBF



Postprocessing

- We improve our score leveraging on domain-specific patterns of the dataset
- Re-ranking with **boosts** that share a common workflow
  1. Start from a list of *K* predicted tracks for a playlist *p*
  2. Normalize the score
  3. Boost the precomputed score in this way:

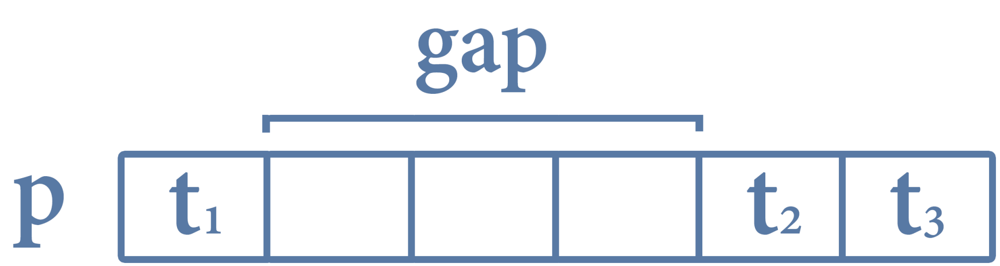
$$Score_{p_k} = Score_{p_k} + Boost_{p_k}$$

Postprocessing

Gap Boost

- Heuristic for playlists where known tracks are given not in order
- Re-rank the final prediction giving more weight to tracks which seems to better "fit" between all gaps

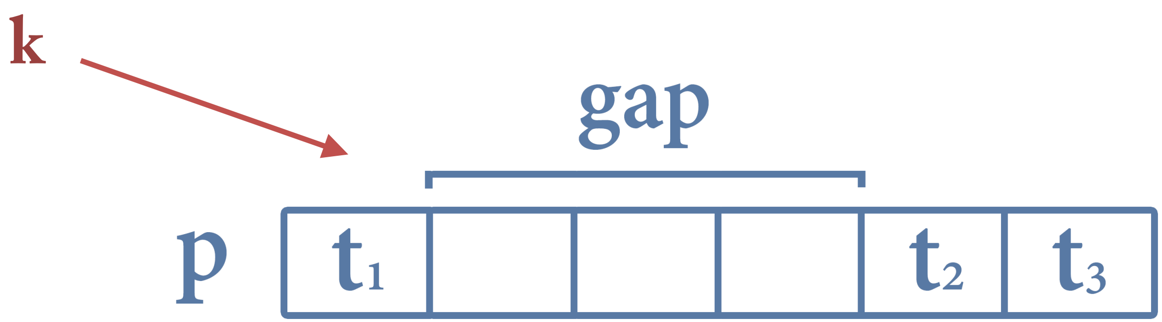
$$GapBoost_{p_k} = \gamma \sum_{g \in G} \frac{s_{k,g_l}\, s_{k,g_r}}{d_g} \quad \forall k \in K$$



Gap Boost

- Heuristic for playlists where known tracks are given not in order
- Re-rank the final prediction giving more weight to tracks which seems to better "fit" between all gaps

$$GapBoost_{p_k} = \gamma \sum_{g \in G} \frac{S_{k,g_l} \, S_{k,g_r}}{d_g} \quad \forall k \in K$$

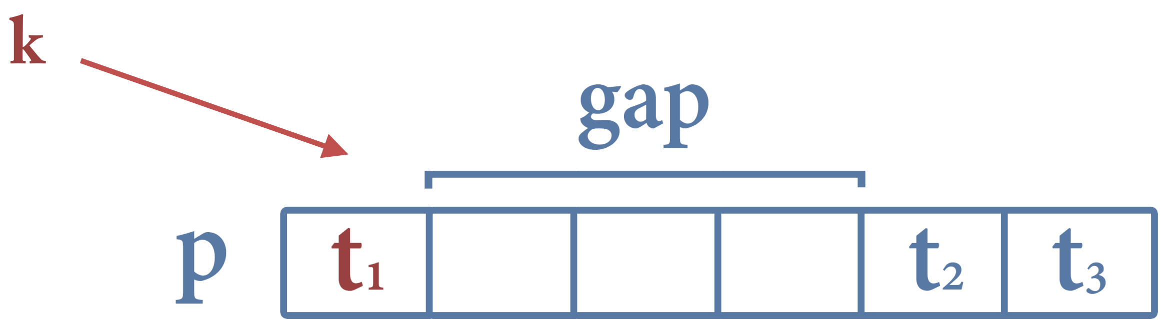


Postprocessing

Gap Boost

- Heuristic for playlists where known tracks are given not in order
- Re-rank the final prediction giving more weight to tracks which seems to better "fit" between all gaps

$$GapBoost_{p_k} = \gamma \sum_{g \in G} \frac{S_{k,g_l} \, S_{k,g_r}}{d_g} \quad \forall k \in K$$

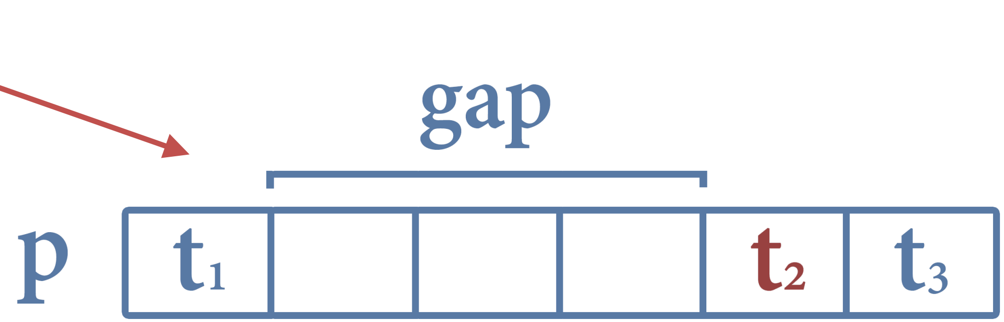


Postprocessing

Gap Boost

- Heuristic for playlists where known tracks are given not in order
- Re-rank the final prediction giving more weight to tracks which seems to better "fit" between all gaps

$$GapBoost_{p_k} = \gamma \sum_{g \in G} \frac{s_{k,g_l} \, s_{k,g_r}}{d_g} \qquad \forall k \in K$$

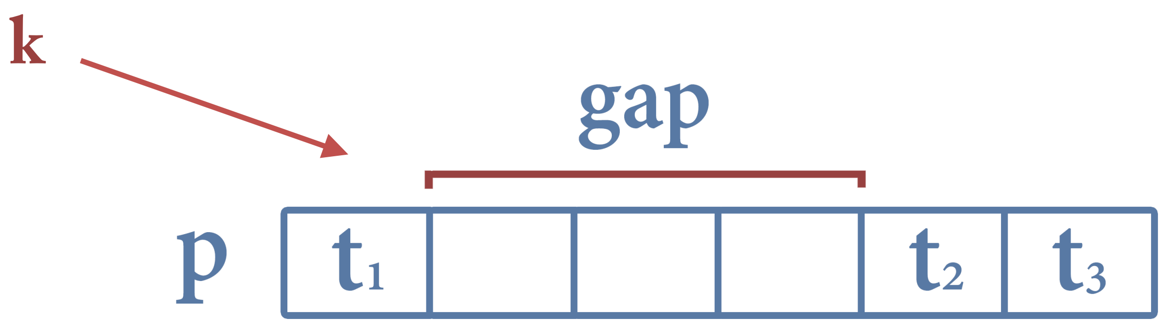


Postprocessing

Gap Boost

- Heuristic for playlists where known tracks are given not in order
- Re-rank the final prediction giving more weight to tracks which seems to better "fit" between all gaps

$$GapBoost_{p_k} = \gamma \sum_{g \in G} \frac{S_{k,g_l} \, S_{k,g_r}}{d_g} \quad \forall k \in K$$



Computational requirements

- To run our model we used a AWS memory optimized cr1.8xlarge VM with 32 vCPU and 244 GiB of RAM
- Parameters tuning for the ensemble takes up to 16h but only computed once

Step	**Time**	**RAM**
Model Creation	1.5h	80GB
Bayesian Optimization	16h	$\sim$15GB
Ensemble	5m	<8GB
Postprocessing	8m	<8GB

Results and conclusions

- Simple, modular architecture
- Extensible with no impact on the pre-existent workflow
- Implementation in **Cython** of the most computationally intensive tasks

Main track		
R-prec	0.2201	3rd
NDCG	0.3856	3rd
Clicks	1.9335	7th

Creative track		
R-prec	0.2197	2nd
NDCG	0.3845	2nd
Clicks	1.9252	4th

SimilariPy - Fast Python KNN-Similarity algorithms for Collaborative Filtering models

bogliosimone / similaripy

To install:

```
pip install similaripy
```

Basic usage:

```
import similaripy as sim
import scipy.sparse as sps

# create a random user-rating matrix (URM)
urm = sps.random(1000, 2000, density=0.025)

# train the model with 50 knn per item
model = sim.cosine(urm.T, k=50)

# recommend items for users 1, 14 and 8
user_recommendations = dot_product(urm, model, target_rows=[1,14,8], k=100)
```

POLITECNICO
MILANO 1863

Thank you!

Questions?

creamy.fireflies@gmail.com

github.com/maurizioFD/spotify-recsys-challenge

github.com/bogliosimone/similaripy