Efficient similarity based methods for the playlist continuation task

Efficient similarity based methods for the playlist continuation task

Guglielmo Faggioli
University of Padova
Padova, Italy
ggfaggioli@gmail.com

Mirko Polato
University of Padova
Padova, Italy
mpolato@math.unipd.it

Fabio Aiolli
University of Padova
Padova, Italy
aiolli@math.unipd.it

ABSTRACT

In this paper, the pipeline we used in the RecSys challenge 2018 is reported. In particular, we present content-based and collaborative-filtering approaches for the definition of the similarity matrices we used for the top-500 recommendation task we had in the challenge. The task consisted in recommending songs to add to partial playlists. Different methods have been proposed depending on the number of available songs in a playlist. We show how a hybrid approach which exploits both content-based and collaborative-filtering based information is effective on this particular task. Specifically, information derived by the playlist titles helped to tackle the cold-start issue.

CCS CONCEPTS

• **Information systems** → **Recommender systems**;

KEYWORDS

Collaborative Filtering, Top-N recommendation, Playlist continuation

ACM Reference Format:
Guglielmo Faggioli, Mirko Polato, and Fabio Aiolli. 2018. Efficient similarity based methods for the playlist continuation task. In *Proceedings of ACM RecSys conference (RecSys'18)*. ACM, New York, NY, USA, 6 pages. https://doi.org/10.475/123_4

1 INTRODUCTION

The task of the RecSys challenge 2018 was playlist continuation[7]. The playlist continuation task [2, 8, 9] can be described as in the following: given a playlist, i.e., a limited list of songs, recommending new songs that are likely to continue such playlist. In the challenge, the training set has been provided by Spotify®, and it contains 1M playlists (Million Playlists Dataset, MPD) with a total of more than 2M unique songs. The test set (a.k.a. challenge set) is composed by different kind of playlist scenarios: playlists with no songs but the title, playlists with few songs (e.g., 1 or 5), and playlists with "many" songs (¿ 10 up to 100). It is clear that, besides the playlist continuation task itself, many other challenges had to be faced, the cold start problem *in primis*. Our idea to tackle this challenge was to develop a framework that relies on the concept of *similarity*.

All the presented methods are based on a similarity matrix. Such similarity are defined over songs, playlists, and content (i.e., title) of the playlists.

The paper is structured as follows. Section 2 describes theoretical groundwork. First of all we describe CF-KOMD[3–5], a top-N recommendation technique based on the concept of margin maximization which aims to maximize the Area Under the ROC Curve (AUC). Then, the WMSD method [1] is presented in both its user-based and item-based versions. At the end of the section, the approach we used to handle the cold start is discussed. Section 3 contains the detailed description of how our pipeline to produce recommendations works. We describe how we validate our methods, and how we manage to build data structures and to wrangle data. Later on, we will describe how we actually combined all elements in order to produce recommendations using techniques described in the theoretical section. Afterwords, Section 4 briefly discusses some other attempts, like other kernels representations or recommendation techniques, which were promising but have failed in the challenge. Finally, Section 5 and 6 presents some of our results, focussing especially on performances, and then we draw some conclusions. Note that, since we are working on a Recommender system framework, in this paper the terms playlist and user will be used interchangeably, just as words song and item, in fact we can consider a playlist as a user and the songs belonging to said playlist as items rated by the user.

2 BACKGROUND

Our setting assumes there are n playlists, contained in the set $\mathcal{U}$, composed by songs taken from set $\mathcal{I}$, i.e., the set of all possible songs. We can define $\mathcal{U}_i$ the list of users who rated item i (or in other words, playlists containing the song i), and $\mathcal{I}_u$, or u^+ interchangeably, the list of songs in playlist u. Note that the challenge set is composed by 10000 playlists, divided in six buckets; each bucket contains playlists where we know a specific number of songs; we will call "seed" the number of songs that are known.

2.1 CF-KOMD

CF-KOMD [3–6] is a technique developed to explicitly maximize the AUC, inspired by preference learning and designed for top-N recommendation. CF-KOMD is a kernel-based method: this might be a strong limitation, since not all

RecSys'18, October 2018, Vancouver, Canada

ACM ISBN 123-4567-24-567/08/06.
https://doi.org/10.475/123_4

datasets can be represented through kernels because of the high amount of memory required. However, thanks to the fact that often datasets in recommender systems are sparse, this technique is often applicable to collaborative filtering problems. More so, it is an highly efficient method, with very short computational time, highly parallelizable, scalable and suited for datasets with few positives and many negative/unlabelled examples.

Let be $\mathbf{W} \in \mathbb{R}^{n\times k}$ the user embedding, and $\mathbf{X} \in \mathbb{R}^{k\times m}$ the item embedding in a k-dimensional latent factor space, where n is the number of users and m the number of items. We can define $\hat{\mathbf{R}} = \mathbf{WX}$ where $\hat{r}_{ui} = \mathbf{w}_u^\top \mathbf{x}_i$, with the constraint $\|\mathbf{x}_i\| = \|\mathbf{w}_u\| = 1$.
CF-KOMD learns implicitly the user representation w_u^* by solving the following optimization problem:

$$\boldsymbol{\alpha}_u^* = \arg\min_{\boldsymbol{\alpha}_{u^+}} \boldsymbol{\alpha}_{u^+}^\top \mathbf{K}_u \boldsymbol{\alpha}_{u^+} + \lambda \left\|\boldsymbol{\alpha}_{u^+}\right\|^2 - 2\boldsymbol{\alpha}_{u^+}^\top \mathbf{q}_u$$

where $\boldsymbol{\alpha}_{u^+}$ is a probability distribution over songs belonging to the playlist u (i.e., user), meanwhile $\mathbf{K}_u \in \mathbb{R}^{m\times m}$ is a kernel matrix between songs, induced by a kernel function κ, and $q \in \mathbb{R}^{|\mathcal{I}_u|}$ is the vector containing for each item the average kernel with all the other items:

$$q_{ui} = \frac{1}{m}\sum_{j\in\mathcal{I}} \kappa(\mathbf{x}_i, \mathbf{x}_j).$$

Finally, the recommendation for the user u is computed by following function:

$$\hat{\mathbf{r}}_u = \mathbf{x}^\top \mathbf{w}_u^* = \mathbf{K}_{u^+:}^\top \boldsymbol{\alpha}_{u^+} - \mathbf{q},$$

where $\mathbf{K}_{u^+:}^\top \in \mathbb{R}^{|\mathcal{I}_u|\times m}$ is the matrix obtained by taking only rows of $\mathbf{K}$ associated to positive items for u and $\mathbf{q} \in \mathbb{R}^m$ is like $\mathbf{q}_u$ but defined over the whole dataset, and considering only those elements associated to $\mathcal{I}_u$. The final recommendation is done by taking the items (i.e., songs) with the highest scores. For more details about CF-KOMD we refer the reader to the works [3, 4].

2.2 WMSD

One typical approach for making recommendation is collaborative filtering (CF). In particular we can cast the playlist continuation problem to a typical implicit feedback top-N recommendation task. In this section we briefly discuss the CF method winner of the Million Songs Dataset challenge [1]. We will refer to this method as WMSD.

One important observation about collaborative filtering is that, when calculating scoring function's value for a specific user-item pair, we can decompose contribution from user and items; what we are doing it is implicitly defining embeddings for users and items.

Thus, underling classical CF techniques, we have similarity measures, able to convey how much two items or two users are close. Among these similarity measures, one of the most used is cosine similarity. In its common form the cosine similarity is symmetric, however in [1] an asymmetric definition is provided: called $\mathbf{R}$ the binary rating matrix, given two users a and b, we can observe that:

$$P(a|b) = \frac{\mathbf{R}_a^\top \mathbf{R}_b}{\mathbf{R}_b^\top \mathbf{R}_b} \text{ and } P(a|b) = \frac{\mathbf{R}_a^\top \mathbf{R}_b}{\mathbf{R}_a^\top \mathbf{R}_a}$$

thus, called $\mathbf{a}$ and $\mathbf{b}$ vectors containing ratings for user a and b

$$P(a|b) = \frac{\mathbf{a}^\top \mathbf{b}}{\mathbf{b}^\top \mathbf{b}} = \frac{\mathbf{a}^\top \mathbf{b}}{\|\mathbf{b}\|^2}.$$

We can define cosine similarity as the square root of the two conditional probabilities:

$$cos(\mathbf{a}, \mathbf{b}) = P(a|b)^{\frac{1}{2}} P(b|a)^{\frac{1}{2}} = \frac{\mathbf{a}^\top \mathbf{b}}{\|\mathbf{a}\|\,\|\mathbf{b}\|}.$$

We can now generalize the concept of cosine similarity using the conditional probabilities in this way:

$$cos_\alpha(\mathbf{a}, \mathbf{b}) = P(a|b)^{\alpha} P(b|a)^{(1-\alpha)} = \frac{\mathbf{a}^\top \mathbf{b}}{\|\mathbf{a}\|^{2\alpha}\,\|\mathbf{b}\|^{2(1-\alpha)}}$$

with $0 \leq \alpha \leq 1$.
So, the asymmetric cosine similarity w_{uv} between the users u and v can be computed as:

$$w_{uv} = \frac{|\mathcal{I}_u \cap \mathcal{I}_v|}{|\mathcal{I}_u|^{\alpha} |\mathcal{I}_v|^{(1-\alpha)}}.$$

Similarly, we can define asymmetric cosine similarity w_{ij} between items i and j as:

$$w_{ij} = \frac{|\mathcal{U}_i \cap \mathcal{U}_j|}{|\mathcal{U}_i|^{\alpha} |\mathcal{U}_j|^{(1-\alpha)}}.$$

Note that when $\alpha = \frac{1}{2}$, w_{uv} is equal to the standard cosine similarity between u and w, while if $\alpha = 0$, the value for w_{uv} is $P(u|v)$. Same considerations can be done for the items. The final score is computed by:

$$\hat{r}_{ui} = \sum_{v\in\mathcal{U}} w_{uv} r_{vi}.$$

The scoring function can also be slightly modified in order to give more importance to the most similar users by exponentiating the weights as in the following:

$$\hat{r}_{ui} = \sum_{v\in\mathcal{U}} w_{uv}^q r_{vi},$$

with $q \geq 0$. If $q = 0$, all weights will be equal to 1, thus the score is the popularity of the song. On the other hand, when $q \gg 1$, the smallest weights will be brought to 0, ignoring the most different examples and considering only the closest neighbours.

2.3 Handling the cold start

One of the main issues with the proposed task, has been the handling of the cold start. The usual approach for facing the cold start problem is using the content of users and items and with such information a recommendation is produced. However, in this specific situation, using a content-based approach has been challenging because of the very few information at our disposal. The only information we can rely on are: the title of the playlist and the number of followers. After a statistical analysis, the latter information was discarded because of its very low significance. Hence, the only usable content was the title. Fortunately, titles represent a powerful tool to aggregate playlists: if two playlists share the same title, it is likely that they share at least a part of their songs. Yet, this is not always true: some titles don't held any information, e.g., "my playlist" or "music", are too generic and not very useful in narrow down the research area for possible recommendations. As better described later, titles have been used to implement a content-based strategy to infer relevant items associated to a playlist. By using the title, we can limit our search to those songs that appear in playlists having the same title.

3 PIPELINE

In order to build the final recommendation, we developed a pipeline composed by three main steps: preprocessing of raw data, building the needed structures, applying the recommendation techniques.

3.1 Step 0: defining the validation set

The following step is not necessary in order to produce a submissions, but it is necessary to validate the methods without doing it directly on the challenge set.

The validation set consists in a part of the training set of playlists used to validate the performance of the developed methods. We built a validation set which mimic the challenge set in the sense that it had the same number of playlists for each seed. In order to do so, we collected length of the shortest playlists for each seed; we then randomly shuffled playlists. Finally, for each seed, we run through this list and, if the length of the playlist is greater or equal to the minimum length for that seed, we include it in the validation set. This procedure is repeated until reaching the same amount of playlists in the challenge set for that specific seed.

3.2 Step 1: Playlists' titles pre-processing

As already pointed out in Section 2.3, the challenge set contains playlists with no useful information (0 seeds), except for their title. The documentation for the challenge specifically states that playlists have been chosen to share their title with many others in the dataset. Note that, although titles are not unique, we often have synonyms, variation of the same word (singular and plural) and punctuation confounding titles. Thus a preprocessing step has been applied to standardize the titles set. Specifically, the following steps have been performed:

- convertion to lower case;
- remotion of punctuation symbols, such as ?!,.-;()/ —:_#'ˆ. Often this punctuation doesn't have any relevant semantic, thus can be safely removed. Not all punctuation has been removed, special symbols like $ have been kept because we assumed having their importance in the titles.
- finally, stemming has been applied.

In the reminder we will refer to the titles set $\mathcal{T}$ as the set of titles produced after the application of the just mentioned procedure.

3.3 Step 2: Building Support Matrices

All the procedures described in this section can be ran in parallel; time, memory and disk space will be better detailed in Section 5.

3.3.1 CF-KOMD Kernel. As previously mentioned, CF-KOMD is a kernel-based collaborative filtering algorithm and thus it requires the construction of a kernel matrix between songs. We validated different kernels, however the best performing ones, in both validation and test (i.e., on the challenge set), have been the linear kernel and the monotone Disjunctive kernel (mD-kernel). This results confirm the considerations presented in [5].

Both the linear and the mD-kernel have been constructed assuming the songs represented in the space of the playlists. In particular, given a song i its representation is defined as the n-dimensional binary vector $\mathbf{x}$ where $\mathbf{x}_u = 1$ iff $i \in \mathcal{I}_u$. Hence, by arranging the songs in rows of a matrix we get the matrix $\mathbf{X} \in \{0,1\}^{m\times n}$.

Given $\mathbf{X}$, the linear kernel matrix can be easily computed by

$$\mathbf{K}_{LIN} = \mathbf{X}\mathbf{X}^\top.$$

In the experiments the normalized linear kernel is considered. For the construction of the mD-kernel matrix we refer the reader to the works [5, 6]. However, the final recommender has been built using the linear kernel.

3.3.2 WMSD playlists similarity matrix. In order to apply the WMSD algorithm in its user-based form, we need to build the playlists similarity matrix.

To save both memory and computational time, we limited the construction of this matrix only between challenge playlists and training playlists. Let be $\mathbf{R}^{ch}$ and $\mathbf{R}^{tr}$ the challenge set rating matrix and the training set rating matrix, respectively. Let us also define the following two matrices:

$$\hat{\mathbf{R}}^{ch}_{p,:} = \frac{\mathbf{R}^{ch}_{p,:}}{|\mathbf{R}^{ch}|^{\alpha}_{p,:}}, \quad \hat{\mathbf{R}}^{tr}_{p,:} = \frac{\mathbf{R}^{tr}_{p,:}}{|\mathbf{R}^{tr}|^{1-\alpha}_{p,:}},$$

where p is a playlist and α is defined as in Section 2.2. Then, the final similarity matrix is computed by:

$$\mathbf{P} = \left[\hat{\mathbf{R}}^{ch}(\hat{\mathbf{R}}^{tr})^\top\right]^q,$$

where $q \geq 0$ is the WMSD locality parameter which controls the size of the neighbourhood. In the extreme cases, if $q = \infty$, the neighbourhood is the playlist itself, while if $q = 0$ the neighbourhood is the whole set of playlists.

3.3.3 WMSD songs similarity matrix. Akin to the CF-KOMD case, the asymmetric cosine similarity between songs can be arranged in a similarity matrix. In this case however we cannot generally define it as a kernel because of its asymmetry. Recalling the asymmetric cosine defined in terms of conditional probabilities (Section 2.2), its corresponding similarity matrix can be computed using a procedure almost identical to the one described in the previous section. In this case however, no distinction has been done between challenge and training set. Simply all training songs has been considered. In the remainder we will refer to the songs similarity matrix with $\mathbf{M}$.

3.3.4 Titles similarity matrix. Instead of naïvely used titles simply as aggregators of playlists, we introduced the notion of titles similarity. Such similarity assumes that titles are represented in the space of songs. It is very similar to the user-based similarity previously defined. The only difference is that a title represents actually a group of playlists which share such title.

Let us define the matrix $\mathbf{A} \in \{0,1\}^{|\mathcal{T}|\times|\mathcal{I}|}$ such that $\mathbf{A}_{ti} = 1$ if song i appears in at least one playlist having title t. Finally we can define the titles similarity matrix $\mathbf{S}$ as

$$\mathbf{S}_{ij} = \mathbf{A}_{i,:}\mathbf{A}_{j,:}^{\top}.$$

Finally, the matrix $\mathbf{S}$ is row normalized. It is worth to notice that the similarity of a title with itself is equal to 1, but, in the dataset it might happen that the title doesn't appear (especially during validation/test phase). In such case it is necessary to force the value 1 in the diagonal.

3.4 Step 3: Building Recommendations

In order to build the recommendations, we applied different approaches on the basis of the number of available seeds:

- **0 seeds** requires a full content-based strategy, since we know the playlists titles and no information about the songs are available;
- **1 seed** we need something more user centred, since it can be seen as a cold start from the item based collaborative filtering point of view;
- **5, 10 and 25 seeds** represent the intermediate case, where enough information is available about items and thus item based strategies are effective;
- **100 seeds** are the highest available number of seeds and an hybrid content-based and user-based (CF-KOMD) approach have shown to be the most effective.

3.4.1 Case 0 seeds: full content-based strategy. As mentioned previously, we have at our disposal the title of the playlists. Note: in both the MPD and the challenge set all playlists with 0 seeds has a title. The rationale behind our strategy is the fact that playlists with similar titles contain, with high probability, similar songs. More so, it is likely that a popular song will be contained in a playlists, since, by definition, popular songs appear with high frequency in the dataset. Let us call t_u the title of the playlist u, and let $pop(t_u, i)$ be the number of times the song i appears in playlists with title t_u. Leveraging on the titles similarity matrix defined in Section 3.3.4, we define the following recommender: given a playlist u with title t_u, and a song i, the score for the playlist-song pair is computed by

$$\hat{r}_{ui} = \sum_{t=0}^{|\mathcal{T}|} pop(t,i)\mathbf{S}^{q}_{t_u,t}.$$

where q has the same role of the locality parameter of WMSD. The best performing q during the validation procedure has been 10. Note that we have to consider the case when the title wasn't in the training set, or the case where, after the preprocessing, the title remains empty: in that case we use the global popularity of songs, proposing songs that are the most populars in the entire dataset.

3.4.2 Case 1 seed: user-based WMSD. We recall that the similarity between challenge playlists and training playlists is stored in a previously built structure: $\mathbf{P}$. What we want to recommend are songs present in similar playlists to the one we are considering, weighted by the similarity between playlists. Hence, the score for a playlist-song pair is computed by:

$$\hat{r}_{ui} = \sum_{v\in\mathcal{U}_i} \mathbf{P}_{uv}.$$

In other words, the score for a song i w.r.t. the playlist u, is the sum of the similarities of u with all playlists v in which the song i appears. We can express this in matrix form: let $\mathbf{R}$ be the binary rating matrix, then

$$\hat{\mathbf{R}} = \mathbf{P}_{u,v}\mathbf{R}.$$

Inside the notation $\mathbf{P}$ there are two hidden hyper-parameters: q and $alpha$. In our experiments, the best performing setting has been $q = 1$ and $\alpha = 0.5$. In that case that the resulting $\hat{\mathbf{R}}$ contains less than 500 values greater than 0, the recommendation is filled with global popularity.

3.4.3 Case 5, 10 and 25 seeds: item-based WMSD. When lot of songs inside a playlist are available we can start relying in methods that better exploit such information. What we would like to recommend, are those songs that often appears in conjunction with the seeds. Additionally, we would like to suggest those songs that appears often with not only one of our seeds, but with all (many) of them, namely those songs i with the highest $\sum_{j\in\mathcal{I}_u} p(i|j)$ for a playlist u. Note that $p(i|j)$ is biased toward popular songs: if i is an highly popular song, it is likely that it appears associated with many songs j, but also with many others. Thus we want to weight $p(i|j)$ with $p(j|i)$: if i is a much more popular song than j, $p(j|i)$ will be low, and we will partially ignore information about $p(i|j)$. On the other hand, if i has more or less the same popularity as j and they often appears together, both $p(j|i)$ and $p(i|j)$ will be significant. Hence, the final score is calculated by:

$$\hat{r}_{ui} = \sum_{j\in\mathcal{I}_u} (p(i|j)^{\alpha} * p(j|i)^{1-\alpha})^{q} = \sum_{j\in\mathcal{I}_u} \mathbf{M}_{ij}.$$

We obtained the best results on all seeds (i.e., 5, 10 and 25) with $\alpha = 0.7$ and $q = 0.4$.

3.4.4 Case 100 seeds: Hybrid strategy with CF-KOMD and title-based filtering. As already said CF-KOMD is a very efficient and powerful recommendation technique which has shown to be effective in top-N recommendation scenarios. However, one of the big limitation about CF-KOMD is the metric it tries to maximize: by maximizing the AUC, we somehow ignore the real task of recommending (very) top-N elements. In order to tackle this problem we can consider applying CF-KOMD only to a smaller subset of songs that is likely to contain all the relevant songs. By reducing the research space, the AUC tends to be more similar to precision-oriented metric such as the *nDCG*. For each target playlist, we perform this filtering step by applying the technique used in the 0 seeds scenario and we pick the first 50000 songs with the highest score. Then, CF-KOMD is applied on the selected subset of songs. In our experiment the parameter λ has been set to 0.01.

4 LESS PERFORMING SOLUTIONS

4.1 Other Kernels

Since KOMD is a Kernel method, many kernels can be used. As already said, the one we decided to use is the linear kernel. Here we discuss some of the kernels that we tried to apply which have performed equally or worse than linear kernel. Note that one important limitation in applying kernel methods is the dimension of kernel matrices: having 2 millions of songs, a kernel between songs has a number of elements approximatively equal to $(2*10^6)^2 = 4*10^{12}$. The reason why we were able to use kernel methods is that linear kernel is highly sparse [4]: all kernels that are not highly sparse (like RBF)

have been immediately excluded.

4.1.1 Linear Kernel Among Popular songs.

In order to study the problem from a computational point of view, first experiments where based on a reduced version of the dataset. In this reduced version, only the 10^5 most popular songs have been considered. This test allowed us to study the sparsity of the kernel matrix for CF-KOMD (since we used the most popular songs, we are considering the most dense part of the kernel, other kernels based on a superset of these song will be at most as dense as this), and computational time needed by different algorithms. One important observation is that KOMD works better when a subset of songs is used instead of the entire dataset. Thanks to this, we developed the solution for seed 100.

4.1.2 Disjunctive kernel.

One generalization of the linear kernel is represented by the Disjunctive Boolean kernel; this kind of kernels consider the input vector as a set of Boolean variables and apply Boolean functions to them. If the features correspond to Boolean literals, we obtain the linear kernels, that simply counts how many true Boolean variables input vectors have in common. Behind the Disjunctive kernel[5] there is the intuition of using disjunction of Boolean variables as features. We consider a disjunction of d Boolean variables; given n input variables, there are $\binom{n}{d}$ possible combinations (i.e., disjunctions), that is the number of features in our resulting embedding. The k-th feature is 1, if at least one of d Boolean variables considered in the k-th combination is true. When $d = 1$, the corresponding disjunctive kernel is actually equal to the linear kernel. Note that disjunctive kernel produces a less expressive kernel than linear one: when $d \geq 2$ it is proven that the kernel is fully dense, thus unusable with our data. However, thanks to some particular properties of the disjunction we were able to build the kernel by precomputing many of the entries that has the same value. The method produced results that were slightly better than linear kernel, but with a contraction of the computational performances, thus we decided to keep using the linear kernel.

4.1.3 Words-based kernel.

As already said, one important information is contained in titles, thus one possible representation for songs, instead of using the playlists they belongs to, is using the words that form the playlist's title. This kind of representation is highly sparse, since the majority of titles are formed by one single word, and almost all of them have less than 4 words. On the basis of this representation a kernel, e.g., linear, can be computed and used inside the CF-KOMD framework. Unfortunately, results have been unsatisfying, thus the research path hasn't been fully explored, but we suppose it is possible to adapt this technique in order to combine kernels or exploit information held by titles.

4.2 Artists and albums

Intuitively, artists and albums should have a great power in shading data, making an higher abstraction of it. This is the rationale behind a plethora of experiments we tried where artists and albums were considered. Some examples of such methods are:

- using artists or albums to limit selection in content-based recommenders;
- using artists or albums as representation of titles in building similarity matrix;
- combining kernels for artists, albums and songs in CF-KOMD.

Unluckily, none of those experiments has been able to perform better than the version of the same algorithm where only songs have been used.

5 PERFORMANCE AND EFFICIENCY

In this section we provide some of the results obtained on our validation set constructed as described in Section 3.1. The metrics used are the same used in the challenge, namely r-precision, nDCG and clicks. For each of the methods we also report the average memory required and the average running time. All experiments have been performed in machine with the processor Intel®Xeon®CPU E5-2650 v3 @ 2.30GHz.

	Time (s)	Memory (GB)
S	224.29	10
K	2669.00	40
P	126.58	8
M	5388.56	100

Table 1: time in seconds and memory in gigabytes needed to build each matrix in phase 1

Note that the r-precision metric has been computed only at songs level and it does not include the artist level r-precision.

The python source code used to implement all the described method is available at https://github.com/guglielmof/recsys_

seed	method	R-prec	Ndcg	Click	Time	Memory
0	Content-based	0.1093334469	0.2451417298	7.706	290.59	16
1	User-based WMSD	0.1353051115	0.305051816	3.456	131.61	16
5	Item-based WMSD	0.124862255	0.307265287	3.653	13276.9508052	72
10	Item-based wMSD	0.177814445	0.381970242	0.5905	17417.31267	72
25	Item-based WMSD	0.200417129	0.387127462	0.3285	16785.9848182	72
100	Selected CF-KOMD	0.163324902	0.345422633	0.6905	6599.02536917	76

Table 2: method used, results, time in seconds and gigabytes needed for each seed

spt2018. Table 1 summarizes the computational time and the memory required to build and store the main similarity matrices. While, Table 2 shows the performance achieved by our model in the validation set. An observation that can be done about the results is that the best performing seed is 25 instead of 100. While, unsurprisingly, in general the more the seeds the better the results especially in the click metric. From an efficiency point of view, since these methods can be ran in parallel the proposed approach is very fast and it can produce a recommendation in less than 5 hours.

6 CONCLUSIONS

This paper describe the method used by the team IN3PD to tackle the playlist continuation task of the RecSys Challenge 2018. It has been shown how different approaches applied according to the type of task defined by different number of seeds. We showed how a content-based strategy based on playlist titles provided us good solutions but with wide margin of improvement when no songs were available. Instead, user-based solutions have provide best results on low seeds problems (only 1 seed). When more than 1 song were available, as in the case of 5, 10 and 25 seeds, item-based strategies have given very good results. Too many seeds, on the other hand, produced unstable results: longer playlists have often songs that tend to be less correlated than those in smaller playlists, thus it has been proven to be harder for pure item-based strategies to produce competitive results. The best solution was to combine a content-based technique that relies on playlist's title, with CF-KOMD, in order to first limit our research area and then refine our solution by gathering most relevant items.

In the future we aim to explore some of the research path we abandoned during the challenge, and also we will focus on injecting some real learning inside the recommendation techniques here described.

REFERENCES

[1] F. Aiolli. Efficient top-N recommendation for very large scale binary rated datasets. In *ACM Recommender Systems Conference*, pages 273–280, Hong Kong, China, 2013.
[2] G. Bonnin and D. Jannach. Automated generation of music playlists: Survey and experiments. *ACM Comput. Surv.*, 47(2):26:1–26:35, Nov. 2014.
[3] M. Polato and F. Aiolli. Kernel based collaborative filtering for very large scale top-n item recommendation. In *Proceedings of the European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN)*, pages 11–16, 2016.
[4] M. Polato and F. Aiolli. Exploiting sparsity to build efficient kernel based collaborative filtering for top-n item recommendation. 2017 forthcoming.
[5] M. Polato and F. Aiolli. Boolean kernels for collaborative filtering in top-n item recommendation. *Neurocomputing*, 2018.
[6] M. Polato, I. Lauriola, and F. Aiolli. A novel boolean kernels family for categorical data. *Entropy*, 20(6), 2018.
[7] M. Schedl, H. Zamani, C. Chen, Y. Deldjoo, and M. Elahi. Current challenges and visions in music recommender systems research. *CoRR*, abs/1710.03208, 2017.
[8] A. Vall, M. Dorfer, M. Schedl, and G. Widmer. A hybrid approach to music playlist continuation based on playlist-song membership. In *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*, SAC '18, pages 1374–1382. ACM, 2018.
[9] A. Vall, H. Eghbal-zadeh, M. Dorfer, M. Schedl, and G. Widmer. Music playlist continuation by learning from hand-curated examples and song features: Alleviating the cold-start problem for rare and out-of-set songs. In *Proceedings of the 2Nd Workshop on Deep Learning for Recommender Systems*, DLRS 2017, pages 46–54. ACM, 2017.