Large Scale Music Recommendation

Roberto Turrin
Moviri S.p.A.
ContentWise R&D
via Schiaffino, 11 - Milan, Italy
roberto.turrin@moviri.com

Andrea Condorelli
Moviri S.p.A.
ContentWise R&D
via Schiaffino, 11 - Milan, Italy
andrea.condorelli@moviri.com

Paolo Cremonesi
Politecnico Di Milano, DEIB
p.zza Leonardo da Vinci 32
Milan, Italy
paolo.cremonesi@polimi.it

Roberto Pagano
Politecnico Di Milano, DEIB
p.zza Leonardo da Vinci 32
Milan, Italy
roberto.pagano@polimi.it

Massimo Quadrana
Politecnico Di Milano, DEIB
p.zza Leonardo da Vinci 32
Milan, Italy
massimo.quadrana@polimi.it

ABSTRACT

Recommending tracks in the music domain is typically a demanding task, mainly due to the size of the catalog (e.g., last.fm claims a 12-million track catalog) and of the events per user (an active user can listen dozens of songs per day). Adding further dimensions, such as contextual information, usually makes the problem difficult to treat.

In this work we propose an innovative approach - namely the *implicit playlist recommender* (IPR) - and compare it with two context-aware, popularity-based solutions used as baselines. All algorithms are implemented using the Apache Spark programming model to face with the large-scale problem. The IPR model is trained using the user implicit listening sessions and proves to outperform the two baseline solutions when the user session is longer than 5 tracks.

Keywords

30Music dataset, large-scale, Apache Spark, Jaccard similarity, music, Idomaar, context, implicit playlist, listening session, play event

1. INTRODUCTION

Music recommender systems propose interesting music to a specific user. Differently from many other popular recommendation domains - such as video - items in the music domain are very short (few minutes) and are not consumed atomically, but always as a bundle of songs.

It is important noting that recommending a group of songs has very little to share with recommending the top-N relevant songs to a user. In accordance with the definition proposed by Cunningham et. al [6], we focused on *music playlist recommendations*, being a playlist a group of songs relevant with respect to a certain context, where the strict listening order does not necessarily matter. The user context strongly affects the music playlist consumption. Unfortunately, contextual information is usually missing and difficult to measure.

We overcome to such limitation by using the user current listening session as a proxy to the user context (in particular to his intent). In fact, regardless the user global preferences (e.g., preferred music genres), whether he starts listening to a certain kind of songs, he is indirectly expressing a strong, temporary preference. Consequently, the next song to suggest to such user has to be somehow aligned (e.g., similar genres, same theme, etc.) with the tracks listened in the current session. User listening sessions form indeed a rich and reliable set of implicit playlists that can be exploited to recommend new tracks relevant to the current listening session. The main steps of this solution consist of (i) identifying the user listening sessions from the atomic play events to create the implicit playlists and (ii) matching the user current listening session to find the relevant implicit playlists, in accordance to a defined similarity metric (see Section 5).

As described in Section 3, we processed all the collected play events of 45K users in the last year (31M events) on a 5.6M-item catalog, defining 2.7M sessions. Therefore, for each recommendation, the current session of the target user has to be compared with all the extracted implicit playlists. The best matching playlists are aggregated in order to identify the track(s) to recommend.

Given the problem size, the computation is particularly demanding and has been implemented using the Apache Spark[1] Python APIs (PySpark) for a scalable and distributed processing. Similarly, a set of experiments has been performed using Idomaar as framework for the evaluation of the recommendation quality, whose core functions are implemented according to the Spark programming model.

All experiments have been executed on Amazon AWS EC2 instances, creating a 9-node cluster with an aggregated power of 72 cores and 540GB RAM.

Outline.

The rest of the paper is structured as follows. An overview of existing approaches is summarized in Section 2. Section 3 presents the dataset, while Sections 4 and 5 provide the details about the implemented baselines and proposed solu-

WOODSTOCK '97 El Paso, Texas USA

[1] https://spark.apache.org/

tion, respectively. The experiments are presented and described in Section 6. Finally, the conclusions and future works are drawn in Section 7.

2. BACKGROUND

Music recommender systems are typically implemented to suggest to users either (i) top-n recommendations, i.e., a limited set of tracks, albums, or artists [12] that are believed to be appealing for them [9] or (ii) an ordered list of tracks or artist (i.e., playlist recommendations) [1, 2].

In this work we focus on presenting to a user a sequence of tracks, referred to as *playlist* . An updated list of solutions can be found in Bonnin and Jannach's survey [3]. Among the top performing algorithms, they mention Same Artist Greatest Hits (SAGH) and Collocated Artist Greatest Hits (CAGH), which we have used as baselines (see Section 4).

Ragno et al. [11] started from the consideration that the more frequently two tracks are played one after the other, the more similar they are. They proposed to use "editorial playlists" to build a graph where every node is a track and every weighted edge represents the similarity ratio among a couple of nodes: the recommendation is generated through a Random Walk in this graph.

A more recent work is presented by Dzuba and Bugaychenko [7]: they mined the user listening behavior to identify frequent patterns and build playlists, further optimized with respect to track diversity. Hariri et al. [8] tried a similar approach focusing on the next-artist recommendation problem: they built a graph and used a Markov Chain model to recommend the next artist. A completely different approach is described in [1], where playlists are generated throughout a 4-step Case-Based Reasoning problem. Finally, Chedrawy et al. [4] used collaborative filtering to build playlists.

3. DATASET

In our experiments we used the *30Music* dataset. This dataset contains 31M play events from 45K users over 5.6M tracks collected via Last.fm public API[2] in 1 year, from 20 Jan 2014 to 20 Jan 2015 (about 4GB of uncompressed CSV file).

Play events are grouped into *user play sessions*, where each session is an ordered list of play events that are assumed to be consequently listened without interruption. Two consecutive play events belong to the same session if the gap is not larger than 800 seconds. The dataset contains 2.7M user sessions, with an average of 11 sessions per user, each one, in turn, composed by 11 tracks, on average.

The *30Music* dataset also provides 4.1M user love preferences and 57K user created playlists. Despite being of undeniable interest for recommender systems, we have not considered explicit feedback from users in our experiments. We focused our attention on implicit feedbacks only - i.e., the user play events and play sessions - since they are the most challenging factors in terms of size.

4. BASELINE ALGORITHMS

In this section we describe SAGH and CAGH, two algorithms proposed in the literature in the settings of music recommendation [3] that we used as baselines. Both algorithms suggest popular tracks and, regardless the basic model behind, they proved to outperform more complex solutions such as Markov models, K-nearest-neighbors, frequent pattern mining.

[2] http://www.last.fm/api

4.1 Same Artist - Greatest Hits (SAGH)

This approach [10] considers only tracks of artists that the user has already listened to during the current listening session. The rank of the tracks is then based on their popularity. Thus, when a recommendation is requested, the algorithm searches for the top popular tracks of every listened artist in the current session. The success of this simple algorithm [3] proves the role of artist-awareness and popularity in driving music preferences.

The core of the algorithm consists in computing the top popular tracks (greatest hits) per artist. Using Apache Spark, we sum the play events for each song (*reduceByKey*) and we keep only sessions played more than once (*filter*). Finally, for each artist, we sort the songs in decreasing play event count in order to obtain the artist-track occurrences *map*.

4.2 Collocated Artists - Greatest Hits (CAGH)

This algorithm is an extension of SAGH [3]. Instead of only considering artists just listened in the user session, it also includes the greatest hits of the most "similar" artists. As a measure of artist similarity, we simply apply cosine similarity on the session-artist matrix, as typically used in collaborative filtering.

5. IMPLICIT PLAYLIST RECOMMENDER (IPR)

The proposed approach extracts recurrent listening patterns from the user sessions and use them to generate recommendations. These listening patterns will be referred to as *implicit playlists*. Given a user, we can identify two types of listening behaviors: "explore" and "exploit". The former occurs when the user is searching for new songs, the latter when a user is listening to music already known and appreciated. In this work we discard the exploratory behavior as we assume it adds noise to the system and it is not useful in understanding the user preferences. The following steps have been implemented to keep only the sequences of tracks that the user likes and to generate recommendations:

1. For each user, we extract all his listening sessions and compute the *shrunk Jaccard similarity* j as in (1).

2. We maintain only the user sessions if there exists at least another session with a Jaccard similarity over the threshold $j\_th \in [0, 1]$. These sessions form the *implicit playlists* of a user.

3. Using all the implicit playlists, we build a track-playlist binary matrix - denoted by $\mathbf{P}$ - where each element (m, n) is 1 *iff* track m belongs to the playlist n. The training makes an intense use of Apache Spark mapping functions such as *join* and *groupByKey*.

4. Given a user listening session, we define the vector p_u of size equals to the total number of tracks, where the m-th element is 1 *iff* the user played the track m in this session. In order to give more importance to the most recently listened tracks, the values of p_u were discounted exponentially in time with a factor w.

Finally, we multiply p_u by $\mathbf{P}$ to obtain a relevance score for each implicit playlist. The final recommendation list is generated by aggregating the implicit playlists by decreasing relevance score.

The shrunk Jaccard similarity is computed as follows:

$$j(S_1, S_2, j\_sh) = \frac{|S_1 \cap S_2|}{|S_1 \cup S_2| + j\_sh} \quad (1)$$

where S_1 and S_2 are the sets of the tracks of two sessions and j_sh is the shrinkage factor used to filter out sessions that are too short.

6. RESULTS AND DISCUSSION

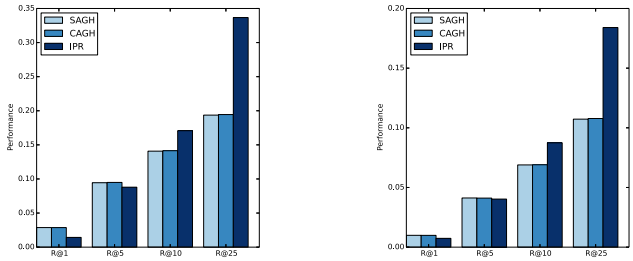
We compared the proposed IPR algorithm to the two baseline solutions (CAGH and SAGH) in terms of quality and computing times. We used the user listening sessions available in *30Music* dataset as input (Section 3), splitting them into two groups - training and test sets, in accordance to standard evaluation techniques for recommendation evaluation (e.g., [5]). We divided sessions on a time basis: sessions started before 21 Oct 2015 (9-month data) were included in the training set, the more recent ones (3-month data) in the test set. The training set has been used to bootstrap the recommendation model, either CAGH, SAGH, or IPR. The test set has been used to compute the recommendation quality. For each session S^i in the test set, we created the known session S^i_k and the hidden session S^i_h. The S^i_k set is composed by the first s play events of S^i and it is available to the recommendation algorithm to make all the required inference (e.g., to identify the user session). The S^i_h set is composed by all play events after the S-th event. In our experiments we used s - referred to as *session length* - equals to either 1, 2, 5, 10 or 20. All sessions with a number of play events lower than the required one were excluded from the related test.

For each session, we generate a recommendation list composed by N tracks, experimenting with $N = 1, 5, 10, 25$. Smaller values of N evaluate the capability of the algorithm to catch the immediate needs of the user, while larger ones better reflect the long-term ones (still within a single user session). The quality has been computed in terms of *recall*, as the percentage of recommended tracks that appear in the S^i_h set. The algorithm recall is obtained by averaging the recalls of all sessions.

During evaluation it is important to consider users may listen the same track several times within a session. To evaluate the impact of repetitions on the recommendation quality, we experimented both *with* and *without repetitions* in the test set only. No filtering over repetitions was performed for the training set, letting the algorithms free to exploit all the information during their training stages.

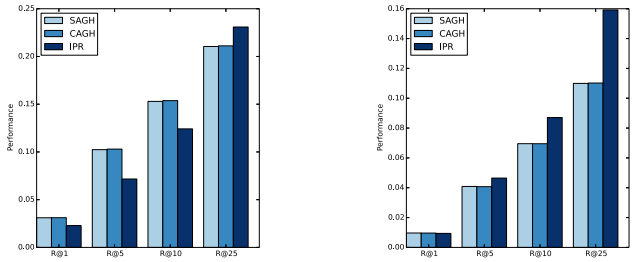
6.1 Experiment settings

In order to manage the massive amount of data, we used Idomaar (`http://rf.crowdrec.eu/`) as evaluator framework, whose core evaluation functionalities (dataset splitting into training, test, and ground-truth sets and quality evaluation) are implemented with distributed and scalable technologies such as Apache Kafka and Apache Spark. The main component of Idomaar is the "orchestrator", which is designed to manage the incoming streams of data and messages (e.g., user-item interactions/signals, requests for recommendations, etc.), distribute them to the "computing environment" where the recommendation models are bootstrapped and the recommendation requests are served; finally, the output of the computing environment is sent to the "evaluator" where they are evaluated both in terms of quality (e.g., precision, recall, NCDG, etc.) and response time.



(a) Session length: 1 (b) Session length: 10

Figure 1: With repetitions in the test set.



(a) Session length: 1 (b) Session length: 10

Figure 2: Without repetitions in the test set.

The experiments have been performed on an Amazon AWS EC2 cluster composed by 9 nodes. Each node was a *r3.2xlarge* instance equipped with 8 virtual CPUs, 61 GB RAM, and 160 GB SSD, running Apache Spark 1.4.0. One of the nodes was elected to cluster master and was provisioned as on-demand instance, the others worked as cluster slaves and were provisioned as spot instances. Practically, there were 8 nodes dedicated to computation and 1 to orchestrating the process, for a total of 72 virtual CPUs, 540GB RAM, and 1.4TB SSD.

We tuned algorithms' hyper-parameters on an independent 10% subset of the users in the dataset. In the end, we configured SAGH and CAGH so that the number of greatest hits per author was equal to 100 and 50, respectively. In addition the author similarity threshold used by CAGH was set to 0.4. As for the IPR, we configured $j\_sh = 5$, $j\_th = 0.1$, and $w = 0.7$.

6.2 Recommendation quality

Figure 1 shows the performance of each algorithm for session lengths 1 and 10, i.e., recommendations are provided to users when they have listened to only 1 and 10 tracks respectively. Repeated tracks were not removed from the test sessions in these experiments. The performance of SAGH and CAGH are almost equal, with a small advantage for CAGH for large N values. IPR always significantly outperforms the baselines for $N \geq 5$, with differences ranging from

20% to 70%. With lower values of N, the baselines still provide better results for small user session lengths (s). The difference w.r.t. the baselines is even clearer when repeated tracks are filtered out from the test set, as reported in Figure 2. In the extreme case the user session is composed by one track, the baselines clearly outperform IPR. However, when more information is available (i.e., $s > 1$), IPR is the clear winner.

IPR suffers from extreme cold-start situations, i.e., when only few information is available. In these conditions, IPR does not have the necessary support to correctly infer the implicit playlist for the user. Popularity-based algorithms - like SAGH and CAGH - are not affected by this issue and have better performance. However, as the user interacts with the system and provides additional feedback, the "trivial" popularity criterion fails. In such condition the more sophisticated criterion at the ground of IPR clearly better matches user tastes and provides higher quality recommendations.

Moreover, IPR has been proven to be superior to baselines for longer recommendation lists. This shows that IPR is able to provide higher quality recommendation with a lower number of recommendation requests w.r.t. baselines. We believe this is due to the intrinsic capability of IPR to capture track co-consumption patterns, which are highlighted only in longer recommendation lists. SAGH and CAGH are still too biased by popularity for being able to capture such patterns effectively.

6.3 Computing time

We extracted the computing times in training and generation of the recommendation lists for each tested algorithm by analyzing the execution logs of Apache Spark. No significant differences in computing times were noticed among the different experimental conditions.

SAGH and CAGH run respectively in about 24 and 35 minutes. Their execution times where mainly dominated by the stage in which artists' popularity is counted along sessions. The additional overhead of CAGH w.r.t. SAGH is due to the quadratic nature of the artist-artist similarity computation.

Surprisingly, IPR generated recommendations in 18 minutes, with a 1.3x and 2x improvement w.r.t. SAGH and CAGH. We believe these improvements, especially against CAGH, are due to the intrinsic flexibility of IPR, which is able to generate a model through a single pass over the input data and with no need of external metadata, such as artist identifiers.

Finally, it is important to point out that configurations of the tuning parameters, especially the similarity thresholds for CAGH and IPR, may lead to significantly different computing times and response qualities. Algorithm configurations were chosen according to the best speed/quality trade-off we found during the hyper-parameter tuning phase.

7. CONCLUSIONS

In this work we developed IPR, a new playlist recommendation algorithm, and compared it against two of the best state-of-the-art recommendation algorithms. All algorithms have been efficiently implemented in the Apache Spark programming framework. IPR outperforms the state-of-the-art baselines with longer user sessions. This can be explained by the fact that IPR better recognizes longer and recurrent patterns in user listening behavior. In addition, we improved the running times w.r.t. the baselines, making it feasible to be deployed in larger production environments.

Acknoledgements.

The research leading to these results was performed in the CrowdRec project, which has received funding from the European Union Seventh Framework Programme FP7/2007-2013 under grant agreement n.610594.

8. REFERENCES

[1] C. Baccigalupo and E. Plaza. Case-based sequential ordering of songs for playlist recommendation. In *Advances in Case-Based Reasoning*, pages 286–300. Springer, 2006.

[2] G. Bonnin and D. Jannach. Evaluating the quality of playlists based on hand-crafted samples. *14th Int. Society for Music Information Retrieval*, 2013.

[3] G. Bonnin and D. Jannach. Automated generation of music playlists: Survey and experiments. *ACM Computing Surveys (CSUR)*, 47(2):26, 2014.

[4] Z. Chedrawy and S. S. R. Abidi. *A web recommender system for recommending, predicting and personalizing music playlists*. Springer, 2009.

[5] P. Cremonesi, Y. Koren, and R. Turrin. Performance of recommender algorithms on top-n recommendation tasks. In *Proceedings of the Fourth ACM Conference on Recommender Systems*, RecSys '10, pages 39–46, New York, NY, USA, 2010. ACM.

[6] S. J. Cunningham, D. Bainbridge, and A. Falconer. More of an art than a science: Supporting the creation of playlists and mixes. In *Proceedings of 7th International Conference on Music Information Retrieval*, pages 240–245, Victoria, Canada, 2006.

[7] A. Dzuba and D. Bugaychenko. Mining users playbacks history for music recommendations. In *Machine Learning and Data Mining in Pattern Recognition*, pages 422–430. Springer, 2014.

[8] N. Hariri, B. Mobasher, and R. Burke. Context-aware music recommendation based on latenttopic sequential patterns. In *Proceedings of the sixth ACM conference on Recommender systems*, pages 131–138. ACM, 2012.

[9] N. Liu and S. Hsieh. Intelligent music playlist recommendation based on user daily behavior and music content. *Advances in Multimedia Information Processing-PCM âĂę*, pages 671–683, 2009. From Duplicate 1 (.

[10] B. McFee, L. Barrington, and G. Lanckriet. Learning content similarity for music recommendation. *Audio, Speech, and Language Processing, IEEE Transactions on*, 20(8):2207–2218, 2012.

[11] R. Ragno, C. J. Burges, and C. Herley. Inferring similarity between music objects with application to playlist generation. In *Proceedings of the 7th ACM SIGMM international workshop on Multimedia information retrieval*, pages 73–80. ACM, 2005.

[12] M. Schedl and D. Schnitzer. Location-aware music artist recommendation. *MultiMedia Modeling*, pages 1–9, 2014.