A hybrid two-stage recommender system for automatic playlist continuation

Vasiliy Rubtsov
National Research University Higher School of Economics
& Avito.ru
vrubcov@hse.ru
vnrubtsov@avito.ru

Mikhail Kamenshchikov
Avito.ru
makamenshchikov@avito.ru

Ilya Valyaev
Avito.ru
iavalyaev@avito.ru

Vasiliy Leksin
Avito.ru
vleksin@avito.ru

Dmitry I. Ignatov
National Research University Higher School of Economics
dignatov@hse.ru

ABSTRACT

In this paper, we provide the solution for RecSys Challenge 2018 by our Avito team, which obtained the 3rd place in main track. The goal of the competition was to recommend music tracks for automatic playlist continuation. As a part of this challenge, Spotify released a large public dataset, which allowed us to train a rather complex algorithm. Our approach consists of two stages: collaborative filtering for candidate selection and gradient boosting for final prediction. The combination of these two models performed well with the playlist and track metadata given.

CCS CONCEPTS

• Information systems → Recommender systems; **• Computing methodologies →** *Boosting*; *Factorization methods*;

KEYWORDS

music recommendations, recsys challenge, collaborative filtering, candidate selection, hybrid recommender systems

ACM Reference Format:
Vasiliy Rubtsov, Mikhail Kamenshchikov, Ilya Valyaev, Vasiliy Leksin, and Dmitry I. Ignatov. 2018. A hybrid two-stage recommender system for automatic playlist continuation. In *Proceedings of the ACM Recommender Systems Challenge 2018 (RecSys Challenge '18), October 2, 2018, Vancouver, BC, Canada.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3267471.3267488

1 INTRODUCTION

An automated music playlist continuation is one of the key problems in music recommender system [13, 14]. High-quality recommendations for playlist continuation could significantly improve user experience and engagement. Cunningham et al. [5] identified that the playlist curation process is a complex problem and it is influenced by a variety of factors like mood, theme or purpose.

Several groups of methods fit to tackle this problem. One of them are content-based approaches [1], which extract genre, semantic features, labels, tags and audio features from the songs user listened to and recommends the most similar songs to the user's favorites. These approaches depend heavily on music metadata quality and carefully crafted features. Different approaches for content models were explored: for example, linear [16], tree-based [12], and neural-based [14].

Another promising approach is collaborative filtering based on curated music playlists. The neighborhood-based approach is popular in the industry for its speed and simplicity [10] [4], however, SVD-based models tend to show the better perfomance [11]. The collaborative approach has several drawbacks [2]. Firstly, an item with no ratings cannot be recommended, which is known as the cold-start problem. Secondly, the music tracks' distribution is highly biased towards popular ones, while many tracks may lack collaborative information.

In our paper, we combine these two approaches into a hybrid method similar to [14]; however, sometimes even simple aggregation schemes like linear combination are suitable [6]. We propose a two-stage algorithm. The first stage is candidate selection based on matrix factorization. The second stage is gradient boosting on decision trees with features extracted both from matrix factorization and music metadata. Our approach performs better than collaborative filtering alone and scales easily to large datasets.

2 OVERVIEW

As part of RecSys Challenge 2018, Spotify provided the Million Playlist Dataset. The goal of the challenge was to build track-recommender system for playlist continuation. On RecSys 2018 two parallel challenge tracks were held: main and creative. The main track allowed no usage of external data sources in the solution. Public and freely accessible by all participants data sources were admitted in the creative track. Our paper provides only the main track's solution. The source code is available on GitHub[1].

RecSys Challenge '18, October 2, 2018, Vancouver, BC, Canada
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-6586-4/18/10…$15.00
https://doi.org/10.1145/3267471.3267488

[1] https://github.com/VasiliyRubtsov/recsys2018

The remainder of this paper is organized as follows. Section 3 describes the Million Playlist Dataset (MPD). Section 4 reviews evaluation metrics for the challenge. Section 5 presents our hybrid approach along with relevant subsections on the training procedure, candidate selection model, feature generation, final recommender, and technical settings. Section 6 outlines our prospective ideas. Finally, conclusions are drawn in Section 7.

3 DATA

As stated in the Challenge Readme[2], the dataset contains 1,000,000 playlists that have been created by Spotify users. Each playlist includes title, ordered track listings, number of missing tracks, and total duration. Track listings on average have 65 tracks per playlist. The track data consists of track's title, duration, album, and artist names. The test dataset was prepared as follows: 10,000 playlists were made up of 10 different challenge categories, with 1,000 playlists in each category. These categories differ in what data is used to predict tracks for a playlist: **(1)** title only **(2)** title and the first track **(3)** title and the first five tracks **(4)** the first five tracks (no title) **(5)** title and the first 10 tracks **(6)** the first 10 tracks (no title) **(7)** title and the first 25 tracks **(8)** title and 25 random tracks **(9)** title and the first 100 tracks **(10)** title and 100 random tracks. A final submission for this challenge should contain 500 tracks for each of the test playlists, ordered by relevance.

4 EVALUATION METRICS

There were three different metrics in the competition: two of them are evaluated on exact track matches and one of them (R-precision) is evaluated both on track and artist matches. In what follows, we denote the ground truth set of tracks by G, and the ordered list of recommended tracks by R. The size of a set or list is denoted by $|\cdot|$, and we use from:to-subscripts to index a list.

4.1 R-precision

R-precision is the number of retrieved relevant tracks divided by the number of known relevant tracks (i.e., the number of withheld tracks):

$$R\text{-precision} = \frac{\left|G \cap R_{1:|G|}\right|}{|G|}. \tag{1}$$

The metric is averaged across all the playlists in the challenge set. This metric rewards the total number of retrieved relevant tracks (regardless of their order).

4.2 Normalized Discounted Cumulative Gain (NDCG)

Discounted Cumulative Gain (DCG) measures the ranking quality of the recommended tracks, increasing when relevant tracks are placed higher in the list. The Normalized DCG (NDCG) is determined by calculating the DCG and dividing it by the ideal DCG in which the recommended tracks are perfectly ranked:

$$DCG = rel_1 + \sum_{i=2}^{|R|} \frac{rel_i}{\log_2(i+1)}. \tag{2}$$

[2]https://recsys-challenge.spotify.com/challenge_readme

In our case, the ideal DCG or IDCG is equal to:

$$IDCG = 1 + \sum_{i=2}^{|G|} \frac{1}{\log_2(i+1)}. \tag{3}$$

If the set intersection of G and R is empty, then the DCG is equal to 0. The NDCG metric is calculated as:

$$NDCG = \frac{DCG}{IDCG}. \tag{4}$$

4.3 Recommended Songs clicks

Recommended Songs is a Spotify feature such that given a set of tracks in a playlist, it recommends 10 tracks to add to the playlist. The list can be refreshed to produce 10 more tracks. Recommended Songs clicks is the number of refreshes needed before a relevant track is encountered. It is calculated as follows:

$$\text{clicks} = \left\lfloor \frac{\arg\min_i \{R_i : R_i \in G|\} - 1}{10} \right\rfloor \tag{5}$$

If the metric is undefined (i.e., there is no a relevant track in R), a value of 51 is picked (which is 1 + the maximum number of clicks possible).

4.4 Rank aggregation

Final rankings were computed by using the Borda Count election strategy. For each of the rankings of p participants according to R-precision, NDCG, and Recommended Songs clicks, the top-ranked system received p points, the second system received $p-1$ points, and so on. The participant with the highest total points would win.

5 OUR APPROACH

5.1 Training procedure

In the attempt to reproduce train/test distribution locally, we split the Million Playlist Dataset into three parts (Fig. 1). Part I contains 980,000 full playlists from the MPD. Other parts (II, III) contain 10,000 playlists each, which are also split into the seed and holdout parts according to the test set. Thus, in II_{seed} and III_{seed} we have 1,000 playlists with only titles, 1,000 playlists with one track and 2,000 playlists with 5, 10, 25, 100 tracks, respectively, and the rest tracks are placed in $II_{holdout}$ and $III_{holdout}$.

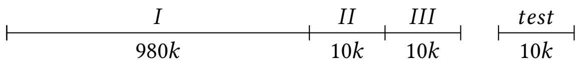

Figure 1: Playlists partition.

In the candidate selection (CS) model (the detailed approach is described in Section 5.2), we use $I \cup II_{seed} \cup III_{seed} \cup test$ for training and $II_{holdout}$ for validation. After training the candidate selection model, we generate candidates for II, III, and $test$ parts named $II_{candidates}$, $III_{candidates}$ and $test_{candidates}$, respectively.

For the selected candidates we generate features (described in Section 5.3) and train ranking model (described in Section 5.4). The ranking model's hyperparametes are tuned using $III_{candidates}$ and the final metrics are calculated on $III_{holdouts}$.

5.2 Candidate selection model

For our candidate selection model (*CS*) we used the LightFM library [7]. It is a Python implementation of a matrix factorization model with the support of user/item metadata. The score is generated as:

$$score(p,t) = b_p + b_t + < q_p, q_t >,$$

where $score(p,t)$ is the score given by the playlist p for the track t, $b_p, b_t \in \mathbb{R}$ are the biases (or the baseline predictors), and $q_p, q_t \in \mathbb{R}^n$ are the latent vectors for the playlist and track, respectively.

There are two types of loss function in LightFM for implicit feedback learning-to-rank: WARP (Weighted Approximate-Rank Pairwise) loss [15] and BPR (Bayesian Personalised Ranking) loss. WARP significantly outperformed BPR on this data. So we use WARP loss, choosing hyperparameters by the scores obtained on validation. Also, we use our enhanced version of LightFM[3] which reduces the fitting time and inference time by about 20 times compared with the original version.

We trained two different models *CS* and CS_{text}. The first *CS* model uses only collaborative information: occurence of *track_id* in *playlist_id* in form of binary sparse matrix with each row representing playlist and each column representing track. This *CS* model is used for playlists with non-empty seeds.

The second model uses words from given playlist names as features (the top-2000 most frequent words). In this case, the bias and the latent vector of the playlist is the sum of the biases and latent vectors of the features (words) included in playlist name:

$$b_p = \sum_{i \in f_p} b_i, \quad q_p = \sum_{i \in f_p} q_i,$$

where f_p consists of the features of a given playlist p.

For each playlist, we know how many tracks are in the holdouts. If for a playlist p there are only k tracks in the holdouts, we take top-max$(k \cdot 15, k + 700)$ tracks according to the model's prediction excluding the tracks which are already present in the seed. Parameters 15 and 700 were chosen based on the following assumptions: we wanted to generate enough candidates for playlists with a few tracks in holdout set (therefore $k + 700$) and $k \cdot 15$ is chosen in order to keep final training set size about 10 000 000 (due to RAM and training time restrictions).

5.3 Feature engineering

Here, we describe features that are generated for each candidate, a track t, to extend a certain playlist p.

LightFM features. The features are produced with trained LightFM models. We use five features: $score(p,t), b_p, b_t, < q_p, q_t >$ and the score rank over all the candidates for a given playlist. These five features are produced from each of the two models: *CS* and CS_{text}.

Co-occurrence features. Let $n_{i,j}$ denote the number of playlists containing both tracks i and j. Also, let n_i be the number of playlists with track i. These numbers are calculated on the consolidated playlist set $I \cup II_{seed} \cup III_{seed} \cup test$.

We compute features for each playlist p and candidate t as follows. Let p be a playlist with the seed consisting of tracks $t_1, t_2, \ldots, t_n$. We compute mean, min, max, and median statistics over the tracks in the given playlist for $n_{t,t_1}, n_{t,t_2}, \ldots, n_{t,t_n}$. Then we compute the same statistics for $\frac{n_{t,t_1}}{n_{t_1}}, \frac{n_{t,t_2}}{n_{t_2}}, \ldots, \frac{n_{t,t_n}}{n_{t_n}}$. The statistics are used as features for the final ranking model.

[3]https://github.com/dmitryhd/lightfm

Additional features. For a pair (p,t):

- The number of unique artists and albums in p
- The number and share of tracks in p with the same artist and album as t
- The global frequency of t
- The global frequency of t's artist and album
- The number of tracks in p's seed and holdouts

5.4 Recommendation model

As a result of candidate selection we have three sets of candidates: $II_{candidates}$, $III_{candidates}$, and $test_{candidates}$, which consist of pairs (p,t), where the track t is a candidate to extend the playlist p. For each pair we have features described in section 5.3. Next, for the sets $II_{candidates}$ and $III_{candidates}$ we generate labels: 0, if there is no pair in the holdouts, or 1, otherwise.

As a final recommendation model we use XGBoost [3] trained on $II_{candidates}$ with binary logistic loss; the hyper-parameters are tuned on the validation set $III_{candidates}$ maximizing ROC AUC value.

Next, we use the predictions of this model on $test_{candidates}$, while ranking the tracks for each playlist.

Table 1 shows the top-10 features according to the XGBoost [3] feature importance with the gain importance type.

Table 1: Top 10 features

Feature	Gain
Co-occurrence normalized mean	1049
CS_{text} model, dot product $< q_p, q_t >$	101
Playlist count	100
Co-occurrence normalized max	74
Tracks holdout count	63
Co-occurrence median	33
Track count	29
CS model, dot product $< q_p, q_t >$	28
CS_{text} model, score rank	26
Co-occurrence mean	20

As we can see from Table 1, the most contributing feature is the co-occurrence normalized mean. Perhaps, it could be beneficial to use that feature as a candidate selection model.

5.5 Technical details

All the experiments were conducted on a Debian machine with Intel(R) Xeon(R) CPU E5-2697 v3 @ 2.60GHz (28 cores, 56 threads) and 256 GB of RAM memory. Our method was implemented in Python3. The entire pipeline execution takes about 100 hours, including 85 hours spent on fitting *CS* model and consumes up to 200GB RAM.

6 FUTURE RESEARCH

Our paper covers the basic approach for two-staged recommendation system. In our opinion, it can be improved by altering candidate selection and adding new features to ranking. An easy step to improve and diversify candidate selection could be candidate sampling with high co-occurrence normalized mean. As can be seen in Table 1, this feature itself is a strong predictor of playlist-track match. New ranking features can be of three types:

- Co-occurrence features;
- Text and text-based models;
- Features using track and playlist duration, albums and artists.

Firstly, the co-occurence features might be seen as a continuation of the current idea: $\frac{n_{t,t_i}}{n_{t_i}}$ can be seen as a conditional probability estimation of candidate t given track t_i. Averaging them implies independance of different tracks t_i in playlist. This assumption can be weakened by estimating $\frac{n_{t,t_i,t_j}}{n_{t_i,t_j}}$ for each pair of tracks t_i and t_j in the input playlist.

Secondly, the text-based features (words) were used in CS_{text}. Word embeddings can be used to determine playlist and track proximity.

Thirdly, some simple features can be used to consider the relative duration of certain track and playlist. Simple heuristics like a match in certain playlist title and artist or album can give precise recommendation for a fraction of playlists.

7 CONCLUSION

Our experience of participation in the last three RecSys Challenges [8, 9] shows that the most effective solutions exploit a two-stage scheme: first, candidate selection and then ranking model. We have used this approach this year too.

Another important condition of the success in the competition is the fruitfully designed scheme of local validation. We have paid much attention to the local validation sets; it helps to avoid overfitting and obtain high scores on the test set.

Once more, we have confirmed that the models based on matrix factorization (LightFM, in particular) are capable to play the role of a base model for candidate selection. Also, the candidate selection with LightFM trained on text features helped to significantly improve Clicks metric by handling cold-start problem with only playlist titles known.

ACKNOWLEDGEMTS

The work of the last author was implemented within the framework of the Basic Research Program at the National Research University Higher School of Economics (HSE) and supported within the framework of a subsidy by the Russian Academic Excellence Project '5-100'.

REFERENCES

[1] Pedro Cano, Markus Koppenberger, and Nicolas Wack. 2005. Content-based Music Audio Recommendation. In *Proceedings of the 13th Annual ACM International Conference on Multimedia (MULTIMEDIA '05)*. ACM, New York, NY, USA, 211–212. https://doi.org/10.1145/1101149.1101181

[2] Òscar Celma. 2010. Conclusions and Further Research. In *Music Recommendation and Discovery: The Long Tail, Long Fail, and Long Play in the Digital Music Space*. Springer Berlin Heidelberg, Berlin, Heidelberg, 185–191. https://doi.org/10.1007/978-3-642-13287-2_9

[3] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*. 785–794. https://doi.org/10.1145/2939672.2939785

[4] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep Neural Networks for YouTube Recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems (RecSys '16)*. ACM, New York, NY, USA, 191–198. https://doi.org/10.1145/2959100.2959190

[5] Sally Jo Cunningham, David Bainbridge, and Annette Falconer. 2006. More of an Art than a Science: Supporting the Creation of Playlists and Mixes.. In *Proceedings of 7th International Conference on Music Information Retrieval* (2007-01-05). Victoria, Canada, 240–245. http://dblp.uni-trier.de/db/conf/ismir/ismir2006.html#CunninghamBF06

[6] Dmitry I. Ignatov, Sergey I. Nikolenko, Taimuraz Abaev, and Jonas Poelmans. 2016. Online recommender system for radio station hosting based on information fusion and adaptive tag-aware profiling. *Expert Syst. Appl.* 55 (2016), 546–558. https://doi.org/10.1016/j.eswa.2016.02.020

[7] Maciej Kula. 2015. Metadata Embeddings for User and Item Cold-start Recommendations. In *Proceedings of the 2nd Workshop on New Trends on Content-Based Recommender Systems co-located with 9th ACM Conference on Recommender Systems (RecSys 2015), Vienna, Austria, September 16-20, 2015. (CEUR Workshop Proceedings)*, Toine Bogers and Marijn Koolen (Eds.), Vol. 1448. CEUR-WS.org, 14–21. http://ceur-ws.org/Vol-1448/paper4.pdf

[8] Vasily Leksin, Andrey Ostapets, Mikhail Kamenshikov, Dmitry Khodakov, and Vasily Rubtsov. 2017. *Combination of Content-Based User Profiling and Local Collective Embeddings for Job Recommendation.* MPRA Paper/ Published in CEUR Workshop Proceeding Experimental Economics and Machine Learning. vol.1968 (2017): pp. 9-17. University Library of Munich, Germany. https://EconPapers.repec.org/RePEc:pra:mprapa:82808

[9] Vasily A. Leksin and Andrey Ostapets. 2016. Job recommendation based on factorization machine and topic modelling. In *Proceedings of the 2016 Recommender Systems Challenge, RecSys Challenge 2016, Boston, Massachusetts, USA, September 15, 2016*. 6:1–6:4. https://doi.org/10.1145/2987538.2987542

[10] G. Linden, B. Smith, and J. York. 2003. Amazon.com recommendations: item-to-item collaborative filtering. *IEEE Internet Computing* 7, 1 (Jan 2003), 76–80. https://doi.org/10.1109/MIC.2003.1167344

[11] Francesco Ricci, Lior Rokach, and Bracha Shapira. 2015. *Recommender Systems Handbook* (2nd ed.). Springer Publishing Company, Incorporated.

[12] Peter Romov and Evgeny Sokolov. 2015. RecSys Challenge 2015: Ensemble Learning with Categorical Features. In *Proceedings of the 2015 International ACM Recommender Systems Challenge (RecSys '15 Challenge)*. ACM, New York, NY, USA, Article 1, 4 pages. https://doi.org/10.1145/2813448.2813510

[13] Markus Schedl, Hamed Zamani, Ching-Wei Chen, Yashar Deldjoo, and Mehdi Elahi. 2018. Current challenges and visions in music recommender systems research. *IJMIR* 7, 2 (2018), 95–116. https://doi.org/10.1007/s13735-018-0154-2

[14] Andreu Vall, Matthias Dorfer, Markus Schedl, and Gerhard Widmer. 2018. 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)*. ACM, New York, NY, USA, 1374–1382. https://doi.org/10.1145/3167132.3167280

[15] Jason Weston, Samy Bengio, and Nicolas Usunier. 2011. WSABIE: Scaling Up to Large Vocabulary Image Annotation. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume Three (IJCAI'11)*. AAAI Press, 2764–2770. https://doi.org/10.5591/978-1-57735-516-8/IJCAI11-460

[16] Tong Zhang and Vijay S. Iyengar. 2002. Recommender Systems Using Linear Classifiers. *J. Mach. Learn. Res.* 2 (March 2002), 313–334. https://doi.org/10.1162/153244302760200641