Efficient K-NN for Playlist Continuation*

Domokos M. Kelen
MTA SZTAKI†
kdomokos@sztaki.hu

Dániel Berecz
MTA SZTAKI†
berecz.daniel@sztaki.hu

Ferenc Béres
MTA SZTAKI†
beres@sztaki.hu

András A. Benczúr
MTA SZTAKI†
benczur@sztaki.hu

ABSTRACT

We present our solution for the RecSys Challenge 2018, which reached 9th place on the main track leaderboard of the competition. We developed a light-weight playlist-based nearest neighbor method to complete music playlists by using the playlist-track matrix along with track and playlist metadata. Our solution uses a number of domain specific heuristics for improving recommendation quality. One major advantage of our approach is its low computational resource use: our final solution can be computed on a traditional desktop computer within an hour.

KEYWORDS

Music Recommendation, Playlist Continuation

ACM Reference Format:
Domokos M. Kelen, Dániel Berecz, Ferenc Béres, and András A. Benczúr. 2018. Efficient K-NN for 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/10.1145/3267471.3267477

1 INTRODUCTION

In this paper, we present the solution of the team Definitive Turtles for ACM RecSys Challenge 2018 [4]. The task of the challenge was automatic playlist continuation. Given a user playlist containing some seed tracks, a list of tracks should be recommended for continuing the playlist. The challenge task is based on the Million Playlist Dataset [7] released by Spotify.

The core of our approach is a playlist-based nearest neighbor method, which recommends items from similar playlists. Our solution uses a modified cosine similarity metric that takes into account the popularity of the tracks. Similarity calculation is improved by accounting for the position of tracks on the playlist. We further improve the score of the algorithm by applying transforming functions such as amplification [3] over the similarity scores.

We computed separate models for each subtask, applying different techniques and hyperparameter values. Despite its simplicity, our solution reached 9th place overall on the main track of the competition. Out of the three different evaluation metrics for the challenge, we optimized our solution for NDCG. Consequently, we reached best result for NDCG: our submission ranks 13 for RPREC, 7 for NDCG and 11 for Clicks.

Compared to better performing solutions of the challenge, our approach is relatively light-weight and simple: it requires no access to supercomputing or GPUs, and can be implemented in a few hundred lines of Python code. Our final solution can be computed within an hour on a standard desktop processor and 16 GB memory.

We describe the Main Track of the competition along with its subtasks. We present the variants and parameters of our proposed nearest neighbor method, and the results of our experiments.

2 CHALLENGE TASK

The task of the main track of the 2018 ACM RecSys Challenge was *playlist continuation*. Challenge participants had to provide a recommendation toplist of length 500 for each of 10,000 incomplete playlists, given some combination of (i) the first n tracks (ii) random n tracks (iii) the name of the playlist. In contrast to the Creative Track of the challenge, it was forbidden to use external datasets or pre-trained models, thus the only information available to the challenge participants was information available in the Million Playlist Dataset, which was provided by Spotify.

The incomplete 10,000 playlists were published in the *Challenge set* of the competition. For each playlist, some *holdout tracks* were hidden from the participants for the purpose of evaluating submissions. Note that the number of holdout tracks were given for each playlist. The challenge task was divided into 10 *subtasks*, with different information available on playlists in each category,

(1) the name of the playlist,
(2) the name and the first track,
(3) the name and the first 5 tracks,
(4) the first 5 tracks,
(5) the name and the first 10 tracks,
(6) the first 10 tracks,
(7) the name and the first 25 tracks,
(8) the name and 25 random tracks,
(9) the name and the first 100 tracks,
(10) the name and 100 random tracks.

Each subtask contained 1,000 playlists. Note that for subtasks 1 and 2 the track information is missing, or just limitedly available. Hence these are significantly different from the others, and resulted in less reliable recommendations. In contrast, subtasks yielding the highest performance scores were the ones where tracks at random positions were provided, i.e. subtasks 8 and 10.

*Support from H2020 project Streamline No 688191 and 2018-1.2.1-NKP-00008: Exploring the Mathematical Foundations of Artificial Intelligence of the Hungarian Government.
†Institute for Computer Science and Control, Hungarian Academy of Sciences

Publication rights licensed to ACM. ACM acknowledges that this contribution was authored or co-authored by an employee, contractor or affiliate of a national government. As such, the Government retains a nonexclusive, royalty-free right to publish or reproduce this article, or to allow others to do so, for Government purposes only.
RecSys Challenge '18, October 2, 2018, Vancouver, BC, Canada
© 2018 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-6586-4/18/10.
https://doi.org/10.1145/3267471.3267477

2.1 Data

The training data for the challenge was provided by Spotify in the form of 1 million playlists. For each playlist, both the tracks and the name of the playlist were given. Additionally, for each song, the name of the artist and the album were also available in the dataset. The lengths of playlists in the dataset ranged from 5 to 250 and contained 2,262,292 tracks from 571,628 albums of 287,740 artist overall.

To preprocess the names of playlists, we used a name normalization function, which worked by converting all characters to lower case, and stripping out special characters. A few playlist names consist of only emojis: in these cases, we normalized the name by stripping out duplicate characters and sorting the remaining ones. With these normalization steps, the original number of 92,941 playlist names was reduced to 16,752, leaving six unmatched names in the challenge set, only one of which was from subtask 1.

2.2 Evaluation

The following three evaluation metrics were used in the Challenge, which we briefly describe next:

RPREC: R-precision is the number of retrieved relevant tracks (regardless of order) divided by the number of known relevant tracks, averaged across all playlists in the challenge set. The version used in the challenge only considered the first n recommended tracks from the toplist, where n is the length of the ground-truth playlist.

NDCG: Discounted cumulative gain (DCG) measures the ranking quality of the recommended tracks, increasing when relevant tracks are placed higher in the list. Normalized DCG (NDCG) is determined by calculating the DCG and dividing it by the ideal DCG in which the recommended tracks are the actual ground-truth tracks, perfectly ranked [1].

Clicks: Recommended Songs Clicks is a metric based on a Spotify feature that, given a set of tracks in a playlist, recommends 10 tracks to add to the playlist. The list can be refreshed to produce 10 more tracks. Clicks is the number of refreshes needed before a relevant track is encountered. If no relevant track exists in the recommendation, the value 51 is picked.

The competition additionally included *artist scoring*, which awards partial scores when a recommended track is not relevant, but its artist has tracks that appear in the ground-truth set. This scoring was applied only in case of the R-precision metric, the following way: a partial hit is represented by adding 0.25 instead of 1 to the nominator of the formula (i.e. the number of retrieved relevant tracks), with the restriction that an artist can only count as a partial hit once per playlist.

3 PREDICTION METHODS

Several nearest-neighbor based approaches for playlist recommendation are described in the literature [2]. Our main solution is collaborative filtering based. We use the user k-nearest-neighbor (*kNN*) method for computing the similarity of playlists and to recommend tracks from similar ones. For similarity computation, generally we used the playlist-track co-occurrence matrix. Furthermore, we also defined similarity scores based on the track, artist and album metadata, and the name of the playlists. We performed training and parameter optimization for each subtask separately.

Our kNN based method first finds similar playlists to the one in question, and then recommends tracks based on the ones in these lists [5]. Given the pairwise similarities s_{uv} of all playlist pairs u and v, we can define a score for track i belonging to playlist u by

$$\hat{r}_{ui} = \frac{\sum_{v \in N_k(u)} s_{uv} \cdot r_{vi}}{\sum_{v \in N_k(u)} s_{uv}}, \tag{1}$$

where $\hat{r}_{ui}$ is the predicted relevance of track i for playlist u, r_{ui} is the known relevance of track i for playlist u, and $N_k(u)$ is the set of k playlists most similar to u. A toplist of length n is then computed by taking the items with the n highest scores.

For similarity computation we used the cosine similarity measure

$$s_{uv} = \sum_{i \in I} \frac{r_{ui} r_{vi}}{\|R_u\|_2 \|R_v\|_2}, \tag{2}$$

where $R_u \in \mathbb{R}^{|I|}$ is the vector of relevance values r_{ui} for $i \in I$. The relevance value is either 1 or 0 depending on whether the track is present in the playlist. Some playlists contain certain tracks multiple times; in these cases, it is possible to use some function of the number of occurrences to define the relevance.

3.1 Amplification and normalization

We experimented with a modified version of Equation (1), proposed in [3]: we define an exponent α, and use s_{uv}^{α} instead of s_{uv}. This method is called *amplification*, as values of $\alpha > 1$ have the effect of amplifying the importance of more similar playlists relative to less similar ones, this way improving recommendation accuracy.

Since the set of similarities $S_u = \{s_{uv} \mid v \in N_k(u)\}$ may be of different magnitude for different playlists u, we found that it is useful to normalize the scores to the interval $[0, 1]$ before applying amplification. We define

$$\tilde{s}_{uv} = \frac{s_{uv} - \min S_u}{\max S_u - \min S_u}, \tag{3}$$

and use the value $\tilde{s}_{uv}^{\alpha}$ instead of s_{uv} in Equation (1).

3.2 Weighting by Inverse Item Frequency

The use of Inverse Document Frequency in information retrieval [6] is justified by the fact that rare items define similarity better than common items. We utilize this idea to define Inverse Item Frequency: Two playlists are more likely similar if they include the same low popularity tracks than if they share tracks that a very large number of other lists also include.

We apply Inverse Item Frequency to modify the definition of similarity between playlists, counting shared tracks between playlist in inverse proportion to their frequency. By experimentation, we found $((f_i - 1)^{\rho} + 1)^{-1}$ to be a well-performing track weight coefficient, where f_i denotes the number of playlists containing track i. Using this formula, we replace Equation (2) by

$$s_{uv} = \sum_{i \in I} ((f_i - 1)^{\rho} + 1)^{-1} \frac{r_{ui} r_{vi}}{\|R_u\|_2 \|R_v\|_2}, \tag{4}$$

Table 1: NDCG scores on different subtasks using different techniques. *Score A* is the base score of the user-kNN algorithm, *score B* includes amplification and normalization, *score C* additionally includes inverse frequency scaling. *Final score* also includes name information for subtask 2 and positional weighting for subtask 9.

subtask	sample tracks	score A	score B	score C	final score
1	0	-	-	-	0.192
2	first 1	0.292	-	-	0.296
3-4	first 5	0.321	0.326	0.334	0.334
5-6	first 10	0.342	0.348	0.358	0.358
7	first 25	0.336	0.342	0.352	0.352
8	random 25	0.435	0.447	0.482	0.482
9	first 100	0.278	0.293	0.293	0.302
10	random 100	0.445	0.451	0.474	0.474

Table 2: Optimal values for k and ρ for different subtasks.

Subtask	sample tracks	k	ρ
2	first 1	1,500	-
3-4	first 5	8,000	0.35
5-6	first 10	12,000	0.38
7	first 25	40,000	0.43
8	random 25	9,000	0.4
9	first 100	40,000	0.44
10	random 100	4,000	0.39

with optimal values of ρ around 0.4. We also experimented with linear and logarithmic scaling of the track frequency

$$((f_i - 1)\lambda + 1)^{-1} \text{ and} \tag{5}$$

$$\left(\log_b(f_i) + a\right)^{-1}, \tag{6}$$

both of which produced comparable, but slightly worse results.

3.3 Weighting by position

When the first k tracks of a playlist are given in a query playlist and the task is to predict the continuation, tracks with positions above k may be more relevant for defining similarity than the first few ones.

Given a query playlist u and a neighboring playlist v, in the original similarity formula of Equation (2), the nonzero elements of the sum—the tracks shared between the playlists—are counted with equal weight. We modify the relevance of item i of the query playlist based on its position in the playlist. We define a weight coefficient for position p, and modify the relevance r_{ui} of item i on playlist u as

$$\tilde{r}_{ui} = r_{ui}\left(1 + \frac{\max(l,\, p_u(i))}{d}\right), \tag{7}$$

where $p_u(i)$ denotes the position of item i on u, and variables l and d are treated as hyperparameters of the formula. The rationale behind our formula is to assign higher weight to tracks closer to the end of query playlist u and use the same value $1 + l/d$ for the first l tracks. With this, Equation (2) becomes

$$s_{uv} = \sum_{i \in I} \frac{\tilde{r}_{ui} r_{vi}}{\|R_u\|_2 \|R_v\|_2}. \tag{8}$$

Note that with this modification, the calculation of similarity becomes asymmetric: the position of items in the query playlist is relevant, because the items can be either closer or farther away from the end of the playlist, which is the place we are recommending items for. In contrast, the position of items in the neighboring playlist is irrelevant in our definition.

We also experimented with weighting the r_{vi} relevance of tracks of the neighboring playlists in Equation (1) based on their proximity to shared items. However this approach failed to improve recommendation quality.

3.4 Metadata based similarity

Metadata about playlists or tracks can be utilized for calculating similarities between playlists as well. Artists or albums can be used instead of tracks for calculating playlist similarities. Furthermore, two playlists may be similar if their name is the same after certain text normalization steps. Track, artist, album, or name based similarities can be combined either by using the k most similar playlists for each type separately, or by taking a weighted average over the similarity values. Overall, our experience was that metadata based similarities were much less reliable than collaborative filtering i.e. track based ones, and were only useful in cases where track information was missing or very limited (subtasks 1 and 2).

4 RESULTS

For our experiments, we used random train-test splits of the Million Playlist Dataset, with the testing sets only containing playlists with length appropriate for the subtasks. Although the scores in our internal measurements were lower than on the leaderboard, their relative order was consistent with the leaderboard. When comparing models or hyperparameters, we relied on comparing NDCG score, as in our experience, it was the most indicative of overall performance of the three scores combined.

We experimented with several variations and combinations of the techniques described in Section 3, measuring for each subtask separately. The basic algorithm accounts for about 95% of the final score of our method and the described techniques for the remaining 5%, when averaging over all subtasks. However, different techniques improve the score on certain subtasks more: the largest improvement over the score of the basic algorithm was over 10%.

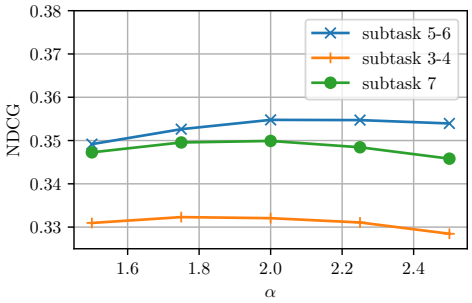

Figure 1: Change in NDCG with different values of α.

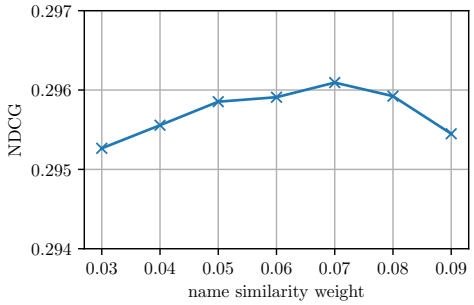

Figure 2: Change in NDCG with different weights for name similarity in subtask 2.

4.1 Effect of similarity normalization

The techniques described in Sections 3.1- 3.3 were used to improve recommendations in all subtasks except for 1 and 2, and contributed most in improving the score of the base algorithm. Table 1 summarizes our results in NDCG for the 10 different subtasks. We highlight here the effect of our methods for improving the original nearest neighbor model. *Score A* is the base score of the user-kNN algorithm, *score B* includes amplification and normalization, *score C* additionally includes inverse frequency scaling. *Final score* also includes name information for subtask 2 and positional weighting for subtask 9.

Amplification, normalization, and inverse item frequency methods (see Sections 3.1 and 3.2) worked best on subtasks 8 and 10, where random tracks from the playlist were given. For amplification rate, the value $\alpha = 2$ proved to be the best choice overall, with optimal values at or near 2 for all subtasks (see examples on Figure 1). Optimal values for k in Equation (1) and ρ in Equation (4) were found using grid search, and are presented in Table 2.

Weighting by position, as described in Section 3.3, was the most useful for improving the scores for subtask 9. Note that subtask 9 has the most provided sample tracks from the beginning of the playlist. The hyperparameters $l = 30$ and $d = 2$ were used in our final solution, making the last track of the query playlists more than 3 times as important as the first track. We were also able to slightly improve upon the score of the basic algorithm on subtask 7 using this method. However with other techniques applied as well, this improvement became negligible. Furthermore, we found that subtask 2 was the only one where using the number of occurrences for item relevance instead of a constant 1 resulted in improvement.

4.2 Text and meta information

For subtask 1, we experimented with multiple approaches based on matching normalized playlist names. This includes normalizing track relevance for a playlist by either the length of the playlists or $\|R_u\|_2$, and taking all track relevance values in playlists with matching names into account. The approach described in Section 3 worked best: for each challenge playlist, we collected playlists with the same normalized name from the training set, and ordered items by the number of playlists with matching names they appeared on.

We were also able to utilize name-based similarity in subtask 2, where only one track from the playlist was given. Even in this case, track information was much more useful than name similarity: our combined similarity measure used track information with weight 93%, and name similarity with weight only 7%, see Figure 2. In the other subtasks, name similarity could not significantly improve the recommendation quality. We also experimented with multiple approaches for using artist and album information, but were unable to improve the scores of our model.

4.3 Implementation and running time

Our models were implemented in Python 3.5, using only built-in functions and the packages *NumPy*, *SciPy* and *Pandas*. The code computes our final submission for the challenge in 58 minutes on a desktop computer with a 7th gen Intel Core i5 processor and 16 GB of memory. This runtime could possibly be further improved by utilizing nearest-neighbor search data structures: our current solution uses the naive approach of computing the full matrix-vector product of the sparse playlist-track matrix and the query playlist vector to collect the most similar playlists.

5 CONCLUSIONS

We described the solution of team Definitive Turtles for the ACM RecSys Challenge 2018, which reached 9th place on the main track leaderboard of the competition. We optimized the standard nearest neighbor method using a number of domain-specific heuristics, which gave good accuracy with very low computational requirements. The source code for producing our final submission for the challenge is available online at https://github.com/proto-n/recsys-challenge-2018.

REFERENCES

[1] Azzah Al-Maskari, Mark Sanderson, and Paul Clough. 2007. The relationship between IR effectiveness measures and user satisfaction. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, 773–774.
[2] Geoffray Bonnin and Dietmar Jannach. 2014. Automated Generation of Music Playlists: Survey and Experiments. *ACM Comput. Surv.* 47, 2, Article 26 (Nov. 2014), 35 pages. https://doi.org/10.1145/2652481
[3] John S Breese, David Heckerman, and Carl Kadie. 1998. Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence*. Morgan Kaufmann Publishers Inc., 43–52.
[4] Ching-Wei Chen, Paul Lamere, Markus Schedl, and Hamed Zamani. 2018. RecSys Challenge 2018: Automatic Music Playlist Continuation. In *Proceedings of the 12th ACM Conference on Recommender Systems (RecSys '18)*. ACM, New York, NY, USA.
[5] Paul Resnick and Hal R Varian. 1997. Recommender systems. *Commun. ACM* 40, 3 (1997), 56–58.
[6] Stephen Robertson. 2004. Understanding inverse document frequency: on theoretical arguments for IDF. *Journal of documentation* 60, 5 (2004), 503–520.
[7] Spotify. 2018. Million Playlist Dataset. https://recsys-challenge.spotify.com/dataset