TrailMix: An Ensemble Recommender System for Playlist Curation and Continuation

Xing Zhao, Qingquan Song, James Caverlee, and Xia Hu
Texas A&M University

ABSTRACT

This paper describes TrailMix, an ensemble model designed to tackle the RecSys Challenge 2018 for automatic music playlist continuation. TrailMix combines three different models designed to exploit complementary aspects of playlist recommendation: (i) CC-Title, a cluster-based approach for playlist titles; (ii) DNCF, an extension of Neural Collaborative Filtering for taking advantage of the flat interaction among tracks; and (iii) C-Tree, a hierarchical approach akin to Phylogenetic trees for finding relationships between tracks.

KEYWORDS

Recommender System, Playlist Continuation, Constructed Tree Comparison, Neural Network, Collaborative Filtering

ACM Reference Format:

Xing Zhao, Qingquan Song, James Caverlee, and Xia Hu. 2018. TrailMix: An Ensemble Recommender System for Playlist Curation and Continuation. In *Proceedings of the ACM Recommender Systems Challenge 2018 (RecSys Challenge '18), October 2, 2018, Vancouver, BC, Canada.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3267471.3267479

1 INTRODUCTION

With the popularity of online music streaming service, e.g. Spotify and Pandora, recommender systems have been widely used to automatically recommend specific tracks, often personalizing per user or per playlist, e.g., [5, 6, 10, 13, 15, 16]. This year's RecSys Challenge builds on this work by focusing on *playlist continuation* so that users can create and extend their own playlists. The main resource is a rich collection of 1 million playlists, including seed songs, playlist titles, playlist length, among many other features.

In this paper, we present the overarching design of **TrailMix**, our team's ensemble approach to the 2018 RecSys Challenge. TrailMix combines three different models designed to exploit complementary aspects of playlist recommendation:

- The first model – CC-Title – exploits and clusters the context information provided by a playlist title alone, with no knowledge of the component tracks in each playlist;
- The second model – Decorated Neural Collaborative Filtering (DNCF) – takes advantage of the flat interaction among given tracks by extending the recently introduced NCF [8]; and
- Finally, the third model – C-Tree – explores the hierarchical structures of a playlist (e.g., playlist-artist-album-track) akin to Phylogenetic trees for finding relationships between tracks.

Together, Trailmix ensembles these three models toward tackling the RecSys Challenge. In the following, we briefly describe the challenge setting and then dive into the details of each approach[1].

Table 1: Dataset Statistics

Items	Quantity	Proportion
Playlists	1,000,000	
unique tracks	2,262,292	100%
unique tracks (freq ≥ 5)	599,341	96.05%
unique tracks (freq ≥ 100)	70,229	80.67%
unique albums	734,684	
unique artists	295,860	

2 DATASET AND EVALUATION

We adopt the large-scale dataset provided by RecSys Challenge 2018. This dataset contains 1 million music playlists created by Spotify users; each playlist has passed a series of quality filters[2].

Table 1 shows the basic dataset statistics. There are over 2.2 million unique tracks and 0.29 million unique artists in this dataset. Considering the sparsity of the playlist-track matrix, we count the tracks which have a frequency more than a specific threshold and their related proportions in this dataset. Specifically, as shown in Figure 1, fewer than 27% of all tracks which appear equal to or more than 5 times take up more than 96% of all playlist-track pair samples; furthermore, around 3% of all tracks which appear 100 times or more take up more than 80% of all samples. To avoid challenges of memory and compute time for the long-tail, we focus in the following on models built over thresholded versions of the original dataset.

For evaluating recommendation quality, we adopt the standard metrics, $R-precision$ and $NDCG$; and the number of refresh actions needed before a relevant track is encountered, $Clicks$, defined by this challenge. More details of the definition and equations can be found at the workshop overview paper [3].

3 TRAILMIX

In this section, we introduce our three major approaches to this challenge, plus how we ensemble the results. Since the challenge is divided into tasks, our hope was to identify models that were well-suited for particular tasks. For TASK 1 (where only playlist

RecSys Challenge '18, October 2, 2018, Vancouver, BC, Canada

ACM ISBN 978-1-4503-6586-4/18/10.
https://doi.org/10.1145/3267471.3267479

[1]All data, annotated samples, code, and experiments are available at https://github.com/xing-zhao/RecSys-Challenge-2018-Trailmix
[2]More details are shown at https://recsys-challenge.spotify.com/

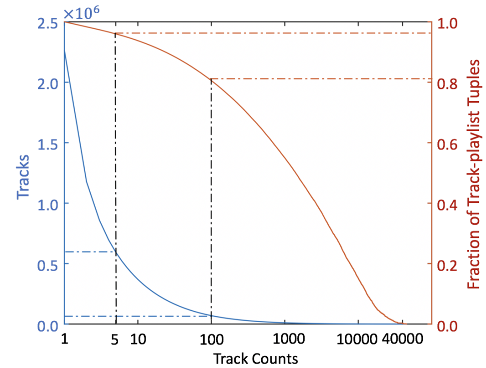


Figure 1: Track Statistics of 1 Million Playlist Dataset.

titles are available), we build a model called CC-Title to exploit the information provided by the given titles. For TASK 2 to 10 (where playlists tracks are given), we build a pair of models, each designed to mine the information given by the existing tracks of playlists. The first – DNCF – takes advantages of the flat interactions of the given tracks; and the second – C-Tree – seeks to explore the hierarchical structures of a playlist. In addition, we build an ensemble model for combining the results of the aforementioned three models and give the final recommendation.

3.1 CC-Title: Context Clustering using Playlist Title

The main idea of the model for TASK 1, which provides playlist title as the only information, is using the context clustering based on a word-track matrix to recommend the relevant tracks to a specific word. The non-processed titles consist of words (including many stop words), punctuation, emoji, etc. First, we use the stemmer provided by NLTK [2] to pre-process the playlist title, and delete all punctuation and emojis. Second, to delete stopwords in the playlist title, we use NLTK stopwords list [2] with some hand-curated music-related stopwords, such as *playlist*, *music*, *songs*, etc. After this pre-processing, we have 17,381 unique playlist titles which contain 9,817 unique normalized words. Since there are many playlist titles consisting of only punctuation, emoji, or stopwords, these titles will be blank after processing. The total number of playlists with blank titles is 22,921 out of 1 million.

From a word-track perspective, we have a matrix of size $9,817 \times 2,262,292$ to recommend relevant tracks to a specific word. The number in each cell, $C(w_i, t_j)$, of this matrix is the frequency of a track t_j in a playlist whose title contains the word w_i. We test many machine learning models on this word-track matrix to find the latent relationship between each word and track, such as matrix factorization and content-based collaborative filtering. However, the performance and time and space complexity were not ideal. Eventually, we adopt contextual clusters to find the latent relationship between words. After grouping the words based on their contained tracks in the word-track matrix, we recommend the most popular tracks, with the highest frequency, from each contextual cluster to the playlist. The main contribution is this simplified model can handle the size of the word-track matrix ($9,817 \times 2,262,292$ in our case), resulting in better performance than our other tested methods. The pseudo code of this model is shown in Algorithm 1.

Algorithm 1 Context Clustering using Playlist Title

P: 1 Million Playlist
$Title$: Title of each playlist in P
$Track$: A list of tracks of each playlist in P
M: Number of unique words
N: Number of unique tracks
k: Number of Context Clusters (hyper-parameter)
β: A bias power for frequency
IDF: The inverse document frequency of all tracks
function BUILD WORD-TRACK MATRIX(P, $Title$, $Track$, IDF, β)
    $C = zeros_{M \times N}$
    for playlist p in P **do**
        for word w in $Title[p]$ **do**
            for track t in $Track[p]$ **do**
                $C(w, t)$ += 1
            $C(w, t) = C(w, t)^{\beta} \times IDF[t]$
    return C
function CONTEXTUAL CLUSTERING(k, C)
    $Cluster = K - Means - Clustering(k, C)$ ▷ objects are rows in matrix C
    for $sub\_cluster \in Cluster$ **do**
        for word w in $sub\_cluster$ **do**
            $sub\_cluster$ += $C[w]$
    return $Cluster$
function RECOMMENDATION($Cluster$, $GivenTitle_p$)
    $R_p = zeros_{1 \times N}$
    for word w in $GivenTitle_p$ **do**
        for all $sub\_cluster_w$ contained word w **do**
            R_p += $sub\_cluster_w$
    Sort R_p by decreasing order
    return $R_p[:500]$ ▷ Top 500 most frequent tracks in R_p

Further note that we adopt an inverse document frequency (IDF) weighting approach to normalize the number in each cell of the word-track matrix by using the track's inverse document frequency over the entire 1 million list. Since in most cases a track only exists one time in a playlist, there is no term frequency (TF) component as in many TF-IDF variations.

3.2 DNCF: Decorated Neural Collaborative Filtering

Since the remaining tasks provide seed tracks for each playlist, we could adopt factor-based models which have achieved great success in multiple recommendation tasks [12]. Most of these approaches follow the matrix factorization setting, in which users' preference and items' features are modeled as latent factors; and their interactions are constructed as the linear combinations of these factors. However, existing works have shown that simple linear combinations are often insufficient to model complex user-item interactions [8]. This problem could be alleviated by inducing deep architectures, which raises considerable discussions in recent literature [17].

To incorporate these state-of-the-art advances of deep learning techniques, we propose to adopt the Neural Collaborative Filtering method (NCF) [8] and modify it with two decorations towards higher effectiveness and efficiency. There are two reasons that we chose NCF as our ground model. First, the model structure is simple and easy to generalize into large-scale settings. Second, it can leverage implicit negative feedback of users' preference. Although NCF requires low memory and space complexity comparing with other advanced deep frameworks, it is still not practical to directly apply on the target problem due to the large recommendation (item) scope and the matrix sparsity. Thus, we propose two modifications in the training and testing phase respectively to address this issue: (i) First, we adopt a constrained negative sampling during the training phase for more targeted training; and (ii) Second, we constrain the recommendation space and reorder the final 500 predictions with Word2Vec model during the testing phase towards better prediction. Details are introduced as follows.

3.2.1 Constrained Negative Sampling. Since there are more than two million tracks in the whole dataset while each playlist only contains no more than 250 tracks, the sparsity of the whole playlist-track matrix is lower than 0.05%. It is challenging for factor-based models to effectively extract useful implicit negative feedback during the training phase since the negative sampling space for each playlist is too huge to handle with limited time and computational power. We intuitively constrain the negative sampling space to the space of the tracks appearing equal to or more than 100 times in the training data. This constrained negative sampling is equivalent to enlarging the sampling probability of popular samples and lowering the probability of rarely appearing samples, which has been shown to be effective in the literature [4, 9]. It is worth noting that only the negative samples are constrained while the positive samples remain the whole dataset during training, which protects the feasible embedding and prediction of all the testing data except for TASK 1.

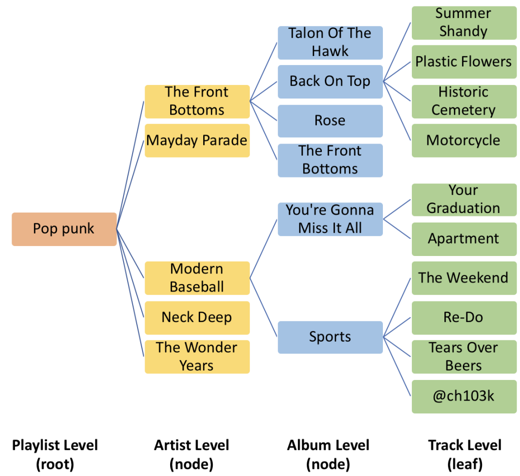
3.2.2 Constrained Recommendation with Reordering. As described in Section 2, the long-tail property of track frequency illustrates that fewer than 4% of tracks which appear equal to or more than 100 times take up more than 80% of the positive training samples. It is reasonable to constrain the recommendation space by only recommending the popular tracks during testing phase towards a more targeted prediction. Coupled with the selection of our negative sampling space, we only recommend tracks appearing 100 times or more in the training data, which is always shown to be effective during our evaluation. For each playlist, we rank the prediction scores of all the tracks appearing equal to or more than 100 times, which do not appear in its positive training samples, and then recommend the top 500 tracks denoted as set ϕ_1^p for playlist p.

To partially encode the rest of the information of the other tracks, after acquiring the top 500 tracks for each playlist, we adopt an ensemble trick to reorder these 500 tracks for better performance. Specifically, we apply Word2Vec model [11] on the training data and get the embedding vector of each track, and then adopt a three step construction for each playlist p as follows. Firstly, find the most similar 50 tracks of each track, merge these similar tracks together based on the training tracks of playlist p, and denote this merged set as ϕ_2^p. Secondly, directly find the most similar 500 tracks playlist p and denote it as set ϕ_3^p. These operations can be easily implemented with the *gensim* python package [14]. Finally, we reorder ϕ_1^p as follows:

$$\phi_{final} = [L_1, L_2, L_3, L_4],$$

where $L_1 = \phi_1^p \cap \phi_2^p \cap \phi_3^p$, $L_2 = \phi_1^p \cap \phi_3^p \setminus L_1$, $L_3 = \phi_1^p \cap \phi_2^p \setminus L_1$, and $L_4 = \phi_1^p \setminus (L_1 \cup L_2 \cup L_3)$. The intra-order of tracks in $L_i (i = 1, 2, 3, 4)$ is consistent with their orders in ϕ_1^p, i.e., if track a and b are both in L_i, a ranks higher than b in L_i if a ranks higher than b in ϕ_1^p.

Figure 2: An example playlist in the structure of tree



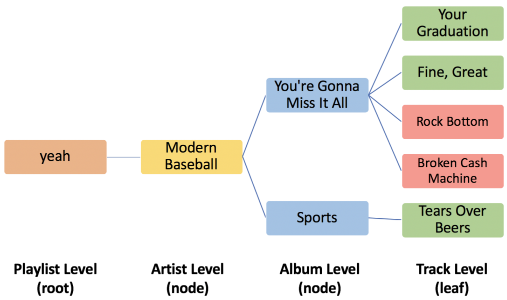
3.3 C-Tree: Constructed Tree

While DNCF utilizes the flat structure of a playlist to surface the latent relationships between tracks, can we also treat each playlist as a hierarchical structure? To complement DNCF, we propose an alternative model, C-Tree, inspired by the use of phylogenetic trees widely used in bioinformatics for indicating the ties of consanguinity among taxa[7] [1]. In general, two leaves in the same node are closer than leaves outside the node, in terms of their latent internal similarity or connection. Here, we adopt the similar tree structure to present a playlist because of:

(1) A playlist consists of different tracks, and these tracks always belong to a specific album of an artist, which indicates a playlist has a natural tree-structure;
(2) A list of tracks in a specific playlist always have latent similarity, such as genres, style, listening sense, etc., which indicates a playlist is a specific cluster of certain aforementioned features.

Figure 2 (T_{train}) shows a real example of playlist (pid:11548) in the 1 million train dataset. This playlist, whose title is 'pop punk', includes 48 pop punk and rock tracks, which belong to 12 albums by 5 artists. In terms of genres and styling, it is obvious that tracks coming from the same album (or artist), such as *Re-Do* and *Tears Over Beers* from album *Sports* are closer to each other than

Figure 3: An incomplete playlist contain 5 seed tracks



tracks belonging to different albums (or artists), such as *Re-Do* (rock music) from album *Sports* and *Gold Steps* (pop punk music) from album *Life's Not Out To Get You*. Another problem we are facing is that it is hard to determine the distance of two songs crossing branches in a single tree. For instance, the distance between tracks of artist *Modern Baseball* should be closer to tracks of artist *Mayday Parade* (both of them are rock bands), than tracks of artist *Neck Deep* (a pop punk band). It is impossible to determine this difference using a single tree. Fortunately, we have 1 million playlists which means a huge forest can be utilized for determining the relationship between each pair of artists, albums or tracks. Through comparing one playlist with other 1 million playlists, similar tracks (leaves in the tree) will be clustered into a big branch based on their latent features, e.g. genres. In this way, when all latent feature surface, the rock tracks will be closer to each other than to pop punk tracks in this case, since not every playlist combines pop punk tracks with rock tracks together.

Next, in the recommending process, we use the seed tracks of a playlist as a sub-tree (incomplete tree since the playlist is not complete) to compare to 1 million trees in the training forest. Figure 3 (T_{test}) shows a sub-tree of a playlist which is selected from the challenge dataset (pid:1000320). In this incomplete playlist, there are 5 seed tracks, and 3 of them exist in the training complete tree shown in Figure 2 and the other two (colored as pink) do not. Intuitively, since we know T_{test} prefers band *Modern Baseball*, based on the given training tree T_{train}, we should recommend tracks from album *You're Gonna Miss It All* in T_{train} with a high confidence, since there are 4 out of 5 tracks in the incomplete playlist coming from this album; and recommend tracks from album *Sports* in T_{train} with a lower confidence since only one track in the T_{test} comes from the same album. Furthermore, we would also recommend other artists' tracks, such as *Mayday Parade* and *The Front Bottoms* since they exist in T_{train} and are all rock bands. In addition, during such recommendation, tracks from other pop punk bands in T_{train} will also be recommended but with a very low level of confidence since there is no evidence that playlist T_{test} prefers pop punk music. In summary, we compare an incomplete playlist (sub-tree T_{test}) with a complete train playlist (tree T_{train}) once per time, and give different scores of confidence for each leaf (track) in T_{train} based on the comparison of their structures.

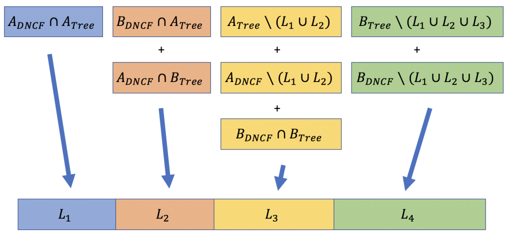
Another challenge is to determine the similarity/distance between incomplete playlist and the train complete playlist. In the research area of bioinformatics, there are many measures to present the similarity/distance between two trees, e.g. Robinson-Foulds (RF) distance. However, such widely used comparison measures could not be used in our case because: 1) our constructed tree has fixed depth for all branches; 2) each node in our tree has strict meaning; 3) in most cases, a leaf can only belong to one specific parent node, in another word, one track only belongs to a specific album by a certain artist. Thus, we design a measure to calculate the distance between one incomplete tree and another complete tree. All recommendation processes of this model are shown on Algorithm 2.

It should be noticed that the Θ_{level}, which is the multiplier for each level of constructed trees, and β, a bias power of similarity score, will be different for each task. These hyper-parameters depend on the number of seed tracks in the incomplete playlist. Again, we use the same strategy of normalization shown in 3.1.

Algorithm 2 Constructed Tree Comparison

MPD: The 1 million complete playlist dataset
T_{train}: A complete playlist in *MPD*
T_{test}: A incomplete playlist in *C*
Θ_{level}: A multiplier for each level of constructed trees
β: A bias power for calculated similarity
ALL_TRACK: All tracks existed in *MPD*
IDF: The inverse document frequency of all tracks existed in *MPD* for normalization
function COMPARISON($T_{train}, T_{test}, IDF, \Theta_{level}, \beta$)
    Using DFS retrieve T_{train} and T_{test}
    $S[root] = 0$ ▷ S stores the similarity score for each node in T_{train} compare with T_{test}
    for *level* from artist to track **do**
        $common\_nodes = T_{train\_level} \cap T_{test\_level}$
        for node n in *common_nodes* **do**
            if n is leaf **then**
                $i = n$'s parent in T_{train}
                $S[n] = |common\_nodes| \times \Theta_{level} + S[i]$
            else
                $i = n$'s parent in T_{train}
                $j = n$'s children in T_{train}
                $k = n$'s children in T_{test}
                $S[n] = min(|j|, |k|) \times \Theta_{level} + S[i]$
    for leaf l in S **do**
        $Normalized\_S[l] = S[l]^{\beta} \times IDF[l]$
    return *Normalized_S*
function RECOMMENDATION(T_{test}, MPD)
    for track t in $ALL\_TRACK \setminus T_{test}$ **do**
        Score[t] = 0 ▷ Initial score for every track not in T_{test}
    for playlist T_{train} in *MPD* **do**
        $Normalized\_S = Comparison(T_{train}, T_{test})$
        for track t in T_{train} **do**
            $Score[t] \mathrel{+}= Normalized\_S[t]$
    Sort *Score* by decreasing order
    return $Score[:500]$ ▷ Top 500 tracks in *Score*

Figure 4: TrailMix For TASK 2



3.4 Ensemble Model: TrailMix

Finally, we ensemble the three models together into our final TrailMix recommender. Based on the individual performance of each model (shown in Section 4), we employ some sub-models for TASK 2 to 10, respectively.

3.4.1 For TASK 2. We find that C-Tree performs much better for TASK 2 in terms of all three measures than DNCF. To extract the most accurate predictions from DNCF model, here we employ a feature *num_holdouts*, the number of tracks that have been omitted from the playlist, as an important information given by the challenge dataset. The *num_holdouts* is essential for evaluating measure R-precision since R-precision will only consider the first *num_holdouts* tracks of 500 recommended tracks. We take the advantage of both models and design an ensemble model shown on Figure 4. Let R_{DNCF} be the recommended 500 tracks by DNCF and R_{C-Tree} be the recommended 500 tracks by C-Tree. And we define the A_{DNCF} as the first *num_holdouts* tracks of R_{DNCF} and B_{DNCF} as the rest set of tracks; similarly, define A_{C-Tree} as the first *num_holdouts* tracks of R_{C-Tree} and B_{C-Tree} as the rest set of tracks. Finally, we recommend the list ϕ_{final} as follows:

$$\phi_{final} = [L_1, L_2, L_3, L_4].$$

In terms of the internal ordering of tracks in each set (L_1 to L_4), we combined the order of tracks from R_{DNCF} and R_{C-Tree} by giving weights (based on model's individual performance) for ranked tracks in each model. Details of this ordering part will be shown in Section 3.4.2.

3.4.2 For TASK 3 to 10. For TASK 3 to 10, which are given between 5 to 100 seed tracks in each playlist, we combine the recommended 500 tracks in R_{NueCF} and R_{C-Tree} using their individual ordering. For $t \in R_{DNCF}, Score_{t\_DNCF} = 500 - Index_t$ where $Index_{t\_DNCF}$ is the ranked index from 1 to 500 in R_{NueCF}. Similarly, we calculate the $Score_{t\_C-Tree}$ for $t \in R_{C-Tree}$. Next, based on checking the individual performance for each model, we give a pre-tuned multiplier as a weight for each model. Finally, we sum the weighted score for t in S^t_{DNCF} and S^t_{C-Tree}, then keep the top 500 tracks with highest scores as our final recommendation.

3.4.3 For TASK 1 and other meaningful playlist title. . For TASK 1, we purely use the CC-Title model for the recommendation. Surprisingly, although the overall performance of this model is not good, we found when playlist title contains some certain words, e.g. *Christmas*, *Christian*, *Disney*, etc. the performance in terms of all three evaluation measures overcomes results from DNCF model and C-Tree model. Therefore, when we encounter such words in a playlist title, we combine the recommended tracks from the CC-Title model, $R_{CC-Title}$, together with R_{DNCF} and R_{C-Tree} using the combination method mentioned in Section 3.4.2.

4 EXPERIMENT RESULTS AND ANALYSIS

In the experiments, we split the 1 million train playlists into five subsets and use cross-validation for hyper-parameter tuning. In terms of the pre-processing the train/test dataset for different tasks, we strictly follow the rules shown on the RecSys Challenge website. We use 0.8 million complete playlists as the train data and the other 0.2 million playlists as the test data. We process the test data for each task by keeping a number of seed tracks sequentially or randomly and using the rest tracks as ground truth.

Since the results shown on the RecSys Challenge Leaderboards[2] are only the average results for all 10 tasks, we report how our models perform in different tasks. Table 2 shows the results for TASK 1, which provides the playlist title only as the information. In summary, the overall performance is much worse compared with other models since the only available information could be used is the title of the playlist, and there is no doubt TASK 1 becomes the bottleneck of our overall performance. However, when we check the details of the results, we found that in some cases, specifically when playlist title contains some words like *Christmas*, *Disney*, or *Christian*, the results using CC-Title will perform much better than the other two models. Based on this finding, we decide to combine the results from CC-Title with other results when these words exist in the playlist title in TASK 2 to 10.

Table 2: Results for CC-Title

	#Seeds	R-precision	NDCG	Clicks
TASK 1	0	0.0639	0.1473	11.77

Table 3: Results for DNCF

	#Seeds	R-precision	NDCG	Clicks
TASK 2	1	0.1001	0.1817	9.663
TASK 3	5	0.2057	0.3189	2.774
TASK 5	10	0.2095	0.3383	1.767
TASK 7	25^S	0.2137	0.3442	1.552
TASK 8	25^R	0.2251	0.3642	0.981
TASK 9	100^S	0.2027	0.3187	1.412
TASK 10	100^R	0.2135	0.3384	1.142

S: sequential ordering; R: random ordering. Same definitions will be used in Table 4 and Table 5.

Table 3 and Table 4 show the results for TASK 2 to TASK 10. We have omitted the results for TASK 4 and 6, because TASK 3 and 5 contain the same sequential seed tracks as TASK 4 and 6, respectively. Since we have not considered the use of the playlist title, applied recommendation model is the same. During checking the individual result for DNCF and C-Tree, we found that the performances of the former overcome the latter in every task, especially for TASK 2 which provides one seed track. Therefore, when we combine these two models, we employ the TrailMix shown in 3.4.1 which takes a greater weight from C-Tree than DNCF for TASK 2. And for TASK 3 to 10, we give a higher weight for the individual recommended tracks coming from C-Tree when we combined it with DNCF. With the growth of the number of provided seeds for the ten tasks, we have noticed that the results steadily increase with maximum performance at seed 25. Then they dropped down when the number of seed tracks continually growing to 100. We think this is because the average *num_holdouts* decreased for the playlist with more seed tracks, although more seed tracks provide more information to predict and recommend, these two models still face a greater challenge to predict accurate tracks in the ground truth with limited chances. An interesting insight was observed when comparing the results for TASK 7 & 8, and TASK 9 & 10. Our models perform better for playlists with random seeding tracks, which inspires us to employ the giving orders of seeds into our models in the future works.

Table 4: Results for C-Tree

	#Seeds	R-precision	NDCG	Clicks
TASK 2	1	0.1554	0.2750	3.6256
TASK 3	5	0.2106	0.3618	1.3147
TASK 5	10	0.2220	0.3793	1.4374
TASK 7	25^S	0.2317	0.3938	1.2532
TASK 8	25^R	0.2322	0.3974	1.0826
TASK 9	100^S	0.2173	0.3797	1.3031
TASK 10	100^R	0.2168	0.3837	1.1926

Table 5 shows the result for the ensemble model, TrailMix, we designed in Section 3.4. Most scores are improved compared with the individual performance of each model. Specifically, the average R-precision is 14.9% better than DNCF and 6.9% better than C-Tree; the average NDCG is 17.8% better than DNCF and 1.8% better than C-Tree; and the average Clicks is 48.3% better than DNCF and 11.7% better than C-Tree. These results verify that our ensemble models perform effectively and steadily.

Table 5: Results for TrailMix

	#Seeds	R-precision	NDCG	Clicks
TASK 2	1	0.1664	0.2746	3.6075
TASK 3	5	0.2301	0.3750	1.0753
TASK 5	10	0.2377	0.3890	1.3275
TASK 7	25^S	0.2470	0.4008	1.0847
TASK 8	25^R	0.2481	0.4104	0.7286
TASK 9	100^S	0.2270	0.3792	1.1921
TASK 10	100^R	0.2281	0.3832	0.9100

Table 6 shows the official results shown on the challenge leaderboards. It should be emphasized that we cannot submit the pure results from DNCF nor C-Tree since TASK 1 must be finished before submission. Therefore, Table 6 shows the results from CC-Title (TASK 1) + DNCF (TASK 2-10), CC-Title (TASK 1) + C-Tree (TASK 2-10), and TrailMix. The results coming from final ensemble model are used as our final submission of this challenge[3].

[3]These scores are shown on the RecSys Challenge leaderboards at https://recsys-challenge.spotify.com/leaderboard

Table 6: Results for TrailMix on Leaderboards

	R-precision	NDCG	Clicks
DNCF + CC-Title	0.1724	0.3292	2.8152
C-Tree + CC-Title	0.1981	0.3567	2.4756
TrailMix	0.2057	0.3711	2.2710

5 CONCLUSION & FUTURE WORK

In this paper, we presented our methods for the RecSys Challenge 2018. While providing some encouraging results, we believe there is still much room for improvement. In our future work, one of the directions is to further explore and employ different distance methods to compare the training playlist with the seeded incomplete playlist. We can consider using more information into the models, such as the orders and names of tracks. In addition, we can work on the model using only playlist title, e.g. employ different Natural Language Processing methods to embed the titles and discover the latent relations between words and phrases, to significantly overcome our bottlenecks.

REFERENCES

[1] Bernard R Baum and Mark A Ragan. 2004. The MRP method. In *Phylogenetic supertrees*. Springer, 17–34.
[2] Steven Bird, Ewan Klein, and Edward Loper. 2009. *Natural language processing with Python: analyzing text with the natural language toolkit.* " O'Reilly Media, Inc.".
[3] 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.
[4] Ting Chen, Yizhou Sun, Yue Shi, and Liangjie Hong. 2017. On sampling strategies for neural network-based collaborative filtering. In *KDD*.
[5] Bruce Ferwerda, Markus Schedl, and Marko Tkalcic. 2015. Personality & emotional states: Understanding users' music listening needs. CEUR-WS. org.
[6] Michael Gillhofer and Markus Schedl. 2015. Iron Maiden While Jogging, Debussy for Dinner?. In *MultiMedia Modeling*, Xiangjian He, Suhuai Luo, Dacheng Tao, Changsheng Xu, Jie Yang, and Muhammad Abul Hasan (Eds.). Springer International Publishing, Cham, 380–391.
[7] Allan D Gordon. 1986. Consensus supertrees: the synthesis of rooted trees containing overlapping sets of labeled leaves. *Journal of classification* 3, 2 (1986), 335–348.
[8] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural collaborative filtering. In *WWW*.
[9] Xiangnan He, Hanwang Zhang, Min-Yen Kan, and Tat-Seng Chua. 2016. Fast matrix factorization for online recommendation with implicit feedback. In *SIGIR*.
[10] Beth Logan. 2002. Content-Based Playlist Generation: Exploratory Experiments.. In *ISMIR*.
[11] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *NIPS*.
[12] Andriy Mnih and Ruslan R Salakhutdinov. 2008. Probabilistic matrix factorization. In *NIPS*.
[13] Martin Pichl, Eva Zangerle, and Günther Specht. 2015. Towards a context-aware music recommendation approach: What is hidden in the playlist name?. In *Data Mining Workshop (ICDMW), 2015 IEEE International Conference on*. IEEE, 1360–1365.
[14] Radim Rehurek and Petr Sojka. 2010. Software framework for topic modelling with large corpora. In *In Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*.
[15] Peter J Rentfrow and Samuel D Gosling. 2003. The do re mi's of everyday life: the structure and personality correlates of music preferences. *Journal of personality and social psychology* 84, 6 (2003), 1236.
[16] Yi-Hsuan Yang and Homer H Chen. 2011. *Music emotion recognition.* CRC Press.
[17] Shuai Zhang, Lina Yao, and Aixin Sun. 2017. Deep learning based recommender system: A survey and new perspectives. *arXiv preprint arXiv:1707.07435* (2017).