Two-stage Model for Automatic Playlist Continuation at Scale

Maksims Volkovs
Layer6 AI
maks@layer6.ai

Himanshu Rai
Layer6 AI
himanshu@layer6.ai

Zhaoyue Cheng
Layer6 AI
joey@layer6.ai

Ga Wu
Vector Institute
wuga@mie.utoronto.ca

Yichao Lu
University of Toronto
yichao@cs.toronto.edu

Scott Sanner
Vector Institute
ssanner@mie.utoronto.ca

ABSTRACT

Automatic playlist continuation is a prominent problem in music recommendation. Significant portion of music consumption is now done online through playlists and playlist-like online radio stations. Manually compiling playlists for consumers is a highly time consuming task that is difficult to do at scale given the diversity of tastes and the large amount of musical content available. Consequently, automated playlist continuation has received increasing attention recently [1, 7, 11]. The 2018 ACM RecSys Challenge [14] is dedicated to evaluating and advancing current state-of-the-art in automated playlist continuation using a large scale dataset released by Spotify. In this paper we present our approach to this challenge. We use a two-stage model where the first stage is optimized for fast retrieval, and the second stage re-ranks retrieved candidates maximizing the accuracy at the top of the recommended list. Our team vl6 achieved 1'st place in both main and creative tracks out of over 100 teams.

KEYWORDS

Playlists Continuation, Collaborative Filtering, Convolutional Neural Network, Gradient Boosting

ACM Reference format:
Maksims Volkovs, Himanshu Rai, Zhaoyue Cheng, Ga Wu, Yichao Lu, and Scott Sanner. 2018. Two-stage Model for Automatic Playlist Continuation at Scale. In *Proceedings of Proceedings of the ACM Recommender Systems Challenge 2018, Vancouver, Canada, 2018 (RecSys Challenge'18),* 6 pages.
https://doi.org/10.1145/3267471.3267480

1 INTRODUCTION

As music consumption shifts towards online playlists, automated playlist continuation is becoming an increasingly more important problem in music recommendation. While significant progress has been made in recent years [1, 7, 11], majority of published approaches are evaluated on proprietary datasets making open collaboration and benchmarking difficult. The 2018 ACM RecSys Challenge aims to bridge this gap by conducting standardized evaluation of playlist continuation models. At the core of this challenge is the Million Playlist Dataset (MPD) released by Spotify [14]. MPD is the largest publicly available dataset of it's kind with 1M playlists and over 2.2M songs. The challenge task is to create a playlist continuation model that is able to accurately recommend next songs for each playlist.

The models are evaluated using a separate set of 10K test playlists for which a subset of songs are withheld. Notably, test playlists vary significantly in length from 0 songs (cold start) to 100 songs. This simulates the production scenario where recommendation model has to perform well during all phases of playlist creation. To avoid over fitting, test held out songs are not released, and teams are required to submit their recommendations to the evaluation server. Over 100 teams participated in the main track of this challenge and our team vl6 achieved 1'st place in both main and creative tracks.

Our approach is based on a two-stage architecture. The first stage focuses on reducing the large 2.2M song search space to a much smaller set of candidates for each playlist. By using a combination of collaborative filtering (CF) and deep learning models we are able to retrieve 20K candidates for each playlist with over 90% recall. Moreover, top-1K songs in these candidate sets already cover close to 60% recall. High recall ensures that most relevant songs are captured in the retrieved set, allowing the subsequent models to only focus on this set without significant loss in accuracy. This, in turn, enables us to apply more sophisticated models in the second stage with minimal impact on run time. In the second stage we develop a pairwise model that directly maps each (playlist, song) pair to a relevance score. By jointly incorporating both playlist and song features into the input, this model can capture pairwise relationships that are difficult to express with traditional CF methods. The aim for the second stage is to re-rank the candidate songs maximizing the accuracy at the top of the recommended list.

In the following sections we describe both stages in detail as well as data partitioning, training and inference procedures.

2 APPROACH

The model architecture diagram of our approach is shown in Figure 1. In the first stage, latent CF model Weighted Regularized Matrix Factorization (WRMF) [8] is used to quickly retrieve 20K candidate songs for each playlist. For each retrieved song we compute additional model scores using embedding convolutional neural network (CNN) as well as User-User and Item-Item [13] neighbor-based models. All model scores together with their linear weighted combination are then concatenated with extracted playlist-song features and used as input to the second stage. In the second stage,

RecSys Challenge'18, 2018, Vancouver, Canada

ACM ISBN 978-1-4503-6586-4/18/10…$15.00
https://doi.org/10.1145/3267471.3267480

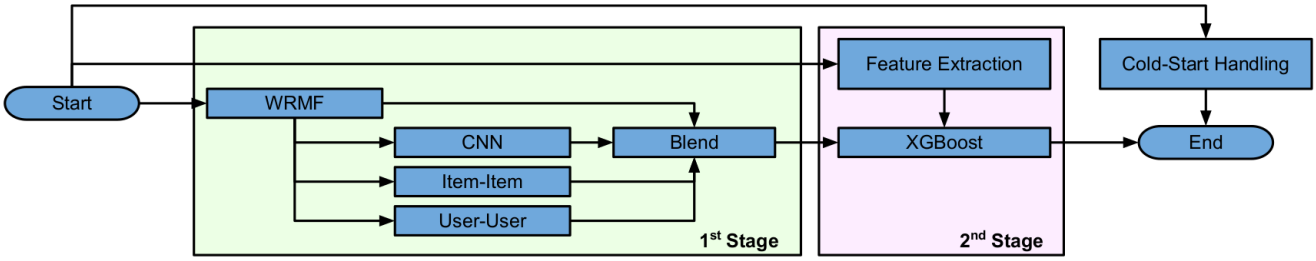

Figure 1: Two-stage model architecture. In the first stage, WRMF is used to retrieve 20K songs for each playlist. CNN, Item-Item and User-User models are then applied to each retrieved song. All model scores together with their linear weighted combination (Blend) are concatenated with extracted playlist-song features and used as input to the second stage. In the second stage, gradient boosting model re-ranks all retrieved songs and outputs the final ranking. Cold start playlists are handled separately (see Section 2.3).

gradient boosting model re-ranks all candidate songs and outputs the final ranking. Note that aside from cold start, a *single* two-stage model is used for all playlists regardless of their length. This is done deliberately to reduce complexity and speed up training time. In this section we describe each stage in detail, the following notation is used throughout:

- $\mathbf{R}$: playlist-song matrix where $\mathbf{R}_{ij} = 1$ if song i is in playlist j and $\mathbf{R}_{ij} = 0$ otherwise. $\mathcal{V}(i)$ deotes the set of all songs that appear in playlist i, and $\mathcal{U}(j)$ denotes the set of playlists that contain song j.
- $\mathbf{U}$, $\mathbf{V}$: playlist and song latent representations. $\mathbf{U}_i$, $\mathbf{V}_j$ denote latent representation for playlist i and song j respectively. To avoid confusion we use superscript to indicate the latent model, for example $\mathbf{U}_i^{wrmf}$, $\mathbf{U}_i^{cnn}$ etc.
- $\mathbf{S}$: predicted relevance scores where $\mathbf{S}_{ij}$ denotes relevance score for playlist i and song j.

2.1 First Stage

The main goal of the first stage is to quickly retrieve candidate song set with sufficiently high recall. Latent CF models have been shown to perform well at scale [3, 5] with efficient inference and high accuracy. Empirically, we found that WRMF [8] produced the best performance and use it to do the initial retrieval. However, WRMF ignores the song order within the playlist which was shown to be important for playlist continuation [7]. To incorporate temporal information, we develop a CNN-based latent model that uses convolutions over song embeddings to produce order-dependent playlist embedding. Moreover, latent models have also been shown to focus on the global patterns, and tend to be poor at detecting strong localized correlations among small sets of items [3, 10]. So we additionally incorporate neighbor-based CF models in the first stage with the aim of capturing patterns missed by the latent models. To avoid repeating full retrieval multiple times we only apply these models to candidate songs retrieved by the WRMF thus significantly reducing the computational overhead. Below we describe each model in detail.

WRMF. WRMF [8] is one of the most popular latent models for binary/implicit CF. Despite the fact that it was published over 10 years ago, we found that with sufficient tuning it can still achieve highly competitive accuracy. WRMF applies least squares to iteratively optimize the following objective:

$$\arg\min_{\mathbf{U},\mathbf{V}} \sum_{i,j} c_{ij}(\mathbf{R}_{i,j} - \mathbf{U}_i\mathbf{V}_j)^2 + \lambda_U \|\mathbf{U}_i\|_2^2 + \lambda_V \left\|\mathbf{V}_j\right\|_2^2)$$

where $c_{ij} = 1 + \alpha\mathbf{R}_{i,j}$. The α constant in c_{ij} controls how much weight is given to the observed playlist-song pairs. After parameter sweeps we found that using rank 200 with $\alpha = 100$ and $\lambda_U = \lambda_V = 0.001$ produced good performance.

CNN. Existing work on modeling temporal patterns in CF has primarily focused on recurrent neural networks (RNNs) [7, 16]. However, RNNs are inherently sequential and difficult to parallelize making training and inference slow. Recent work in language modeling and other domains has shown that CNNs can be readily applied to sequence tasks with comparable or better performance to RNNs [2, 6]. Unlike RNNs, CNNs are fully parallelizable leading to significant speed ups on modern architectures such as GPUs. Inspired by these results we develop a temporal CNN model to generate playlist embeddings that take into account song order within the playlist.

Noting the parallels between language modeling and playlist continuation with documents=playlists and words=songs, we adapt the recently proposed language model by Dauphin et al. [6] to this task. The model architecture is shown in Figure 2. In this model concatenated song embeddings are passed through multiple layers of gated linear unit (GLU) convolutional blocks (see [6] for more details on the GLU architecture). Activations from the last GLU block are then aggregated together using max pooling and passed to fully connected layers that output the playlist embedding.

Formally, input for each playlist i consists of concatenated embeddings for songs in i:

$$\mathbf{\Phi}_i^{1:k} = [\mathbf{V}_1^{cnn}, \mathbf{V}_2^{cnn}, ..., \mathbf{V}_k^{cnn}]$$

Note that since any song subsequence within the playlist also forms a valid playlist we have the following range for k: $1 \leq k \leq |\mathcal{V}(i)|$. $\mathbf{\Phi}_i$ is thus a $p \times k$ matrix where p is the size of the input embedding. Convolutions in GLU blocks are applied from left to right to $\mathbf{\Phi}_i^{1:k}$, and each successive layer captures increasingly longer range structure within the song sequence. To deal with variable length

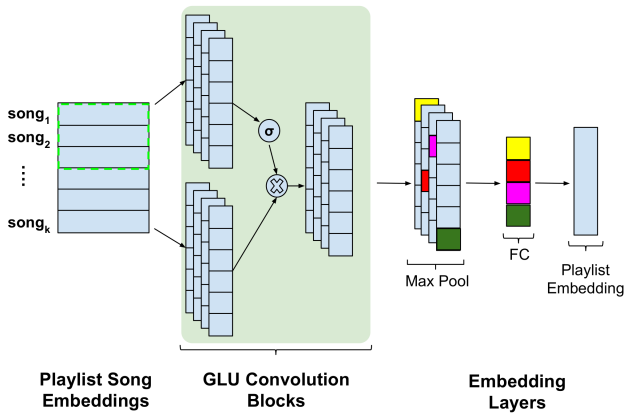

Figure 2: CNN playlist embedding model architecture.

input we use max pooling after last convolutional layer. Max pooling retains the largest activation for each convolutional kernel, for example if last convolutional layer has 500 kernels then the output of max pooling will be a vector of length 500. The output length thus only depends on the number of kernels and not on the input size.

Given the input song sequence passing it through the CNN produces playlist embedding:

$$\mathbf{U}_i^{cnn} = f(\mathbf{\Phi}_i^{1:k}, \theta)$$

where θ is the set of free parameters to be learned. Since our goal is retrieval, $\mathbf{U}_i^{cnn}$ should be an accurate predictor of songs that follow $\mathbf{V}_k^{cnn}$. This forms the basis of our objective function. By appropriately sizing the fully connected layers we make playlist embeddings $\mathbf{U}^{cnn}$ to be the same size as input song embeddings $\mathbf{V}^{cnn}$. The probability that a given song j is contained in playlist i is then defined as:

$$P(\mathbf{V}_j^{cnn} | \mathbf{U}_i^{cnn}) = \frac{1}{1 + e^{-\mathbf{U}_i^{cnn} \mathbf{V}_j^{cnn}}} \tag{1}$$

During training, given a prediction point k, we aim to raise the probability of songs that follow $\mathbf{V}_k^{cnn}$ and lower it for all other songs:

$$\mathcal{L}(i, k, \theta) = \\ - \sum_{\substack{j \in \mathcal{V}(i) \\ j > k}} \log \left(\frac{1}{1 + e^{-\mathbf{U}_i^{cnn} \mathbf{V}_j^{cnn}}} \right) - \sum_{j' \notin \mathcal{V}(i)} \log \left(1 - \frac{1}{1 + e^{-\mathbf{U}_i^{cnn} \mathbf{V}_{j'}^{cnn}}} \right) \tag{2}$$

Adopting a stochastic optimization approach we repeatedly sample prediction point $k \in [1, |\mathcal{V}(i)|]$ for playlists in each mini batch. Given k, we sample songs $j \in \mathcal{V}(i)$ that appear after position k, and songs $j' \notin \mathcal{V}(i)$. Gradients from $\mathcal{L}$ are then used to update the model. Note that unlike language modeling where only the next word is predicted, we instead train the model to predict arbitrary far into the future. Similar to [7], we found that there is strong correlation between consecutive songs. Consequently, only predicting the next song makes the model heavily focus on the last song in the input, significantly hurting performance.

Once training is completed, we make forward passes through the model using all songs in each playlist to obtain embeddings $\mathbf{U}^{cnn}$. Retrieval is then done by computing dot products in the embedding space. For this challenge our CNN model consists of 7 GLU blocks where each block consists of convolutional layer with 900 kernels, followed by batch normalization [9] and ReLU activation. Last GLU block is followed by top-3 max pooling retaining largest three values per kernel, and fully connected layer. Input and output embedding dimension is set to 200 and we use WRMF song representations to initialize input song embeddings $\mathbf{V}^{cnn}$. These embeddings are then updated during model training together with model weights.

Neighbor-based Models. Two popular neighbor-based CF models are User-User and Item-Item [13], we use both models here. The User-User approach estimates relevance by computing similarity between rows of $\mathbf{R}$. Formally, for a given playlist-song pair (i, j) User-User compares all playlists where j appears to i:

$$\mathbf{S}_{ij}^{user} = \sum_{i' \in \mathcal{U}(j)} \frac{\mathbf{R}_{i:} \mathbf{R}_{i':}}{\|\mathbf{R}_{i:}\| \, \|\mathbf{R}_{i':}\|} \tag{3}$$

where $\mathbf{R}_{i:}$ is the i'th row of $\mathbf{R}$. The intuition here is that if song j appears in many playlists that are similar to i then $\mathbf{S}_{ij}^{user}$ is high and j should be included in the recommended list.

Similarly, Item-Item approach estimate relevance by computing similarity between columns of $\mathbf{R}$. This method compares j with all songs in playlist i:

$$\mathbf{S}_{ij}^{item} = \sum_{j' \in \mathcal{V}(i)} \frac{\mathbf{R}_{:j} \mathbf{R}_{:j'}}{\left\|\mathbf{R}_{:j}\right\| \left\|\mathbf{R}_{:j'}\right\|} \tag{4}$$

where $\mathbf{R}_{:j}$ is the j'th column of $\mathbf{R}$. As with User-User, if song j is similar to songs already added in i then $\mathbf{S}_{ij}^{item}$ is high and j is recommended.

Numerous extensions and generalizations have been proposed to improve performance and/or runtime of neighbor-based models [15]. One of the prominent problems that have been identified is the strong bias towards popular items. Popularity translates to denser rows/columns that inflate similarity scores resulting in recommendations that are heavily skewed towards popular items. We found this to be especially prevalent in music recommendation where popularity distribution is highly skewed towards the most popular songs. To address this problem Verstrepen et al. [15] propose to rescale similarity scores by the inverse popularity. We adopt this approach here and multiply both User-User and Item-Item scores by $popularity_j^{-(1-\beta)}$. Here, β is an empirically chosen constant and $popularity_j$ is a normalized popularity score. Setting $\beta < 1$ down weights relevance scores inverse proportionally to song popularity. We found this approach to work well, significantly improving the accuracy of both User-User and Item-Item models. Empirically, we set $\beta = 0.6$ for User-User and $\beta = 0.9$ for Item-Item.

Model Blend. In previous sections we described four different CF models: WRMF, CNN, Item-Item and User-User. Here, we outline how these models are combined. A simple solution would be to select a single best model. However, selecting one particular model may lead to overconfident predictions and higher variance as it ignores model uncertainty in favor of specific model assumptions. Consequently, combining multiple models is desirable [12].

To avoid overfiting we resort to linear weighted ensemble method. Here all models scores are linearly combined with model specific weights:

$$S^{blend} = w_1 S^{wrmf} + w_2 S^{cnn} + w_3 S^{user} + w_4 S^{item} \quad (5)$$

The scores for each model are standardized before the blend by subtracting the mean and dividing by standard deviation. Standardization re-scales the scores into the same range making them comparable across models. The weights are chosen greedily from the set $\{0, 0.1, 0.2, 0.3, 0.4, 0.5\}$, we deliberately keep this set small to further reduce overfitting. After a round of greedy optimization we select the following weight combination: $w_1 = 0.1, w_2 = 0.4, w_3 = 0.3, w_4 = 0.3$. This combination achieved the highest validation accuracy and is used in all subsequent experiments. The five scores $\{S^{wrmf}, S^{cnn}, S^{user}, S^{item}, S^{blend}\}$ are used as input to the second stage for each candidate song.

2.2 Second Stage

Given the candidates retrieved by the first stage, the goal of the second stage is to accurately re-rank these candidates maximizing the accuracy at the top of the recommended list. Since the candidate set is small, second stage model can be more expensive and trade off efficiency for accuracy. We thus focus on pairwise interactions here and develop a model that jointly maps (playlist, song) pairs to relevance scores. This approach is complementary to the first stage where none of the four models are considering pairwise interactions. The main components of the second stage are input features and model architecture, we describe both in detail in the sections below.

Feature Extraction. We conducted extensive feature engineering with the aim of capturing all important aspects of playlist-song relevance. The final set of features that we used can be partitioned into five groups:

- **Input From First Stage**. We use five scores from the first stage directly as input features. This allows the second stage model to quickly recover the performance of the first stage and then focus on improving it.
- **Playlist Features**. Playlist features summarize playlist content information and the types of songs that are in the playlist . We use features such as: playlist name and length, average song/artist/album popularity, song homogeneity and others. Homogeneity is estimated by comparing playlist latent representation (WRMF, CNN etc.) with all the songs that appear in the playlist: $\frac{1}{\|\mathcal{V}(i)\|}\sum_{j\in\mathcal{V}(i)} \mathbf{U}_i\mathbf{V}_j$. We found that some playlists mostly contain songs of specific genre/type making the recommendation task easier, while others are much more diverse. The homogeneity score correlates well with song diversity and is one of the most important features in this group.
- **Song Features**. Similarly to playlist features, we focus on summarizing song content information and types of playlists where the song appears. We use song artist/album/title features, song duration and playlist statistics. Here, we also compute the homogeneity score by comparing song representation to all playlists that contain it: $\frac{1}{\|\mathcal{U}(j)\|}\sum_{i\in\mathcal{U}(j)} \mathbf{U}_i\mathbf{V}_j$. This feature estimates whether the song typically appears in playlists of similar type, and has also proven to be effective.

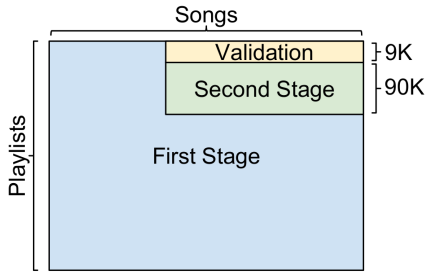

Figure 3: Data partitioning diagram. We sample $9K$ playlists from the MPD with same length distribution as in challenge test set ($1K$ per group excluding cold start). We then sample additional $90K$ playlists with same length distribution for the second stage training ($10K$ per group excluding cold start). For the sampled playlists the training portion is used to train the first stage and the validation portion is used for model evaluation in the $9K$ set, and for second stage training in the $90K$ set.

- **Playlist-Song Features**. This is the most important feature group that directly describes the pairwise similarity between each playlist-song pair. We compute similarity features between target song and songs that are in the playlist, as well as between target playlist and playlists that contain the target song. These features are analogous to item-item and user-user similarities but include additional content information. For example, we compute statistics such as artist/album overlap, average latent score similarity, difference in duration and length etc. Adding this feature group provided the biggest improvement over the first stage, so this is where we believe future effort should focus on.
- **Creative Track**. For the creative track, we extracted 12 additional features from the Spotify Audio API including: acousticness, danceability, energy, instrumentalness, key, liveness, loudness, mode, speechiness, tempo, time signature and valence. Adding these feature provided a small but consistent gain.

Model Architecture. We opted to use a tree-based gradient boosting model (GBM) in the second stage using the excellent XGBoost [4] library. To create the training set we use the 20K song candidates returned by the first stage and randomly sample (up to) 20 relevant songs and 20 not relevant songs for each playlist. These samples form the training set with binary relevant/not relevant targets. We then train a GBM model with pairwise ranking loss as the training objective. Empirically, ranking loss performed better than binary cross entropy objective for this task. In all submissions, we use a GBM model with 150 trees of depth 10.

2.3 Cold Start

Cold start playlists is the only group that is handled separately from the two-stage model since it can't be applied to cold start. Dealing with cold start is challenging in this dataset since only two playlist content features are available - name and length. We found name to be a good predictor of playlist content and focus our cold start

model on this feature. To deal with large scale retrieval in over 2.2M song space we again use a latent approach and aim to create a latent representation that only depends on playlist name.

We achieve this by using matrix factorization as commonly done in latent CF models. We create a new matrix $\mathbf{R}^{name}$ where rows represent songs and playlists concatenated together, and columns correspond to names. Each playlist row of $\mathbf{R}^{name}$ is a one-hot encoding of playlist name. Similarly, each song row of $\mathbf{R}^{name}$ corresponds to name counts from all playlists that contain this song. We then factorize this matrix with truncated SVD:

$$\mathbf{R}^{name} \approx \mathbf{U}\Sigma\mathbf{V} = \mathbf{U}^{name}\mathbf{V}^{name} \tag{6}$$

where $\mathbf{U}^{name} = \mathbf{U}\Sigma$. Given $\mathbf{U}^{name}$ and $\mathbf{V}^{name}$, we treat this as a latent CF model and do retrieval by computing dot products between the SVD vectors. Note that no additional information besides name is required for playlists, so this model can be directly applied to playlist cold start.

The SVD approach has several advantages, first, with a single factorization we jointly embed both playlists and songs into the same latent space. Second, the resulting representations are compact and support efficient retrieval. Finally, there are many efficient libraries available for truncated SVD so even though $\mathbf{R}^{name}$ has over 3.2M rows the factorization can be computed in minutes.

2.4 Training and Inference

The test set for this challenge is comprised of ten different playlist groups with 1K playlists per group. Each group represents a different playlist completion task:

(1) Predict tracks for a playlist given its title only
(2) Predict tracks for a playlist given its title and the first track
(3) Predict tracks for a playlist given its title and the first 5 tracks
(4) Predict tracks for a playlist given its first 5 tracks (no title)
(5) Predict tracks for a playlist given its title and the first 10 tracks
(6) Predict tracks for a playlist given its first 10 tracks (no title)
(7) Predict tracks for a playlist given its title and the first 25 tracks
(8) Predict tracks for a playlist given its title and 25 random tracks
(9) Predict tracks for a playlist given its title and the first 100 tracks
(10) Predict tracks for a playlist given its title and 100 random tracks

This partitioning is done to simulate the production application of the playlist continuation model. The model has to successfully handle all stages of playlist creation from cold start to first few songs, to long playlists with lots of songs. As discussed above, to promote simplicity and improve generalization we train a *single* model for all warm start playlists (groups 2 to 10). To achieve this, we create a validation set with nearly identical playlist length distribution to the test set. For each warm start test group we randomly sample playlists from the MPD that have the same length and partition them as per test test. So for group 2 only first track is used for training and the rest for validation, for group 3 first five tracks are used for training and so on. This provides a validation set with 9K playlists that closely mimics the test set, and we use this set to validate all models before submission. Empirically, this validation scheme provided consistent generalization where improvements on the validation set directly translated to the test set leaderboard.

Given that the second stage model uses predictions form first stage as input, it can't be trained on the same training set. We thus create an additional split for second stage training. To ensure generalization we again follow the same procedure and sample 10K playlists for each of the warm start groups 2 to 10. This provides additional 90K playlist split where training portion is combined with other playlists in MPD to train the first stage, and validation portion is used to train the second stage. Using the 20-20 sampling scheme outlined in Section 2.2 the 90K set produces over 3.5M training examples for the second stage which is sufficient to obtain good results. The full data partitioning diagram is shown in Figure 3.

To train the model we first train first stage on the training portion of MPD (see Figure 3), and then train second stage on the 90K playlist subset using first stage as input. Both stages are validated using the 9K validation set.

	RPREC	NDCG	Clicks
WRMF	0.1641	0.3350	2.1230
CNN	0.1594	0.3230	2.1157
Item-Item	0.1772	0.3534	2.1649
User-User	0.1768	0.3550	1.6332
Blend	0.1866	0.3728	1.4064
2^{nd} Stage	**0.1985**	**0.3846**	**1.1998**
Cold Start			
Popular	0.0395	0.0815	17.2662
Name SVD	0.1106	0.2083	7.9609

Table 1: Validation set results.

3 EXPERIMENTS

All experiments are conducted on a single Ubuntu Linux server with two Intel Xeon(R) E5-2620 CPUs, 256GB RAM, and Titan V GPU. With this architecture end-to-end inference in our model is can be done in under 50ms for each playlist. Following the challenge rules, we use three metrics to evaluate model performance: R-Precision (RPREC), NDCG and Clicks [1]. For all metrics the truncation is set to 500, so the model has to retrieve 500 songs for each playlist.

Table 1 shows the validation performance for both first and second stage models. From the table we see that neighbor-based models produce strong performance either matching or outperforming the latent models. This indicates that despite their simplicity, with appropriate tuning neighbor-based models can still provide highly accurate recommendations. We also see that blending the models together improves performance with Blend significantly outperforming the best individual model on all three metrics. Strong blend performance further supports the conclusion that combining diverse models is beneficial for this task. We also see that CNN has the worst performance out of the four first stage models. However, in the blend this model has the highest weight of 0.4 (see Section 2.1), and thus contributes the most to the combined model. As none of the other models are temporal, this indicates that the modeling temporal structure is important for this task and should be explored further.

The second stage model further improves performance even over the highly competitive blend model. We see over a point gain in both RPREC and NDCG after re-ranking the candidates in the second stage. Empirically, we observed that after the first few trees the GBM would already recover the blend performance and start to improve on it. This is in part because all five first stage model

[1] See https://recsys-challenge.spotify.com/rules for definition of each metric.

Playlist 1	**Playlist 2**	**Playlist 3**
1. Drake - Sneakin	1. Stone & Van Linden - Summerbreeze	1. London Philharmonic Orchestra - Symphony No. 40 in G Minor, K. 550: Allegro molto
2. Logic - The Incredible True Story	2. Havana Brown - We Run The Night	2. London Philharmonic Orchestra - Requiem, K. 626: Lacrimosa dies illa
3. Travis Scott - Birds In The Trap Sing McKnight	3. Calvin Harris - Sweet Nothing	3. Elisabeth Ganter - Concerto in A Major for Clarinet and Orchestra, K. 622: II. Adagio
4. Big Sean - I Decided	4. David Guetta - Titanium	4. Tbilisi Symphony Orchestra - Symphony No.25 In G Minor, K. 183 I. Allegro Con Brio
5. The Weeknd - Starboy	5. Calvin Harris - Feel So Close	5. Tbilisi Symphony Orchestra - Requiem Mass In D Minor, K. 626 : II. Dies Irae
Model Recommendations		
Drake - Fake Love	Rihanna - We Found Love	London Philharmonic Orchestra - String Quintet No.4 in G Minor, K.516: I. Allegro
Migos - Bad and Boujee	Swedish House Mafia - Don't You Worry Child	Takács Quartet- String Quintet No.3 in C Minor, K.515: III. Andante
Post Malone - Congratulations	Calvin Harris - Summer	Columbia Symphony Orchestra - Serenade in G major, K. 525 i. Allegro

Table 2: Playlist continuation examples. We show first five songs contained in each playlist, and top-3 recommendations produced by our model.

	RPREC	NDCG	Clicks	Borda
vl6	**0.2241**	**0.3946**	1.7839	**329**
hello world!	0.2233	0.3932	1.8952	323
Avito	0.2153	0.3845	**1.7818**	322
Creamy Fireflies	0.2201	0.3856	1.9335	320
MIPT_MSU	0.2167	0.3823	1.8754	320

Table 3: Main track test set leaderboard results for the top-5 teams.

scores are explicitly given as inputs to the second stage. These results suggest that the second stage provides an effective way to further improve accuracy by modeling pairwise relationships in a scalable way.

Table 1 also shows validation cold start results. We compare our name-based SVD approach to a simple popularity baseline. To conduct this evaluation we remove all training songs from the validation playlists and treat them as cold start. Here we see that the SVD model significantly outperforms the popularity baseline, indicating that playlist name is useful for the continuation task. However, the cold start model still performs nearly 2x worse than the warm start one. We believe that with additional information such as details on playlist creator, date/time statistics, device etc., the gap in performance between the two models can be significantly reduced. Since no such information is available in the MPD we leave this investigation for future work.

Table 3 shows the main track test set leaderboard results for the top-5 teams. The final ranking is computed using the Borda count aggregation of the individual rankings for each of the three objectives. From the table we see that our team vl6 outperforms all other teams on RPREC and NDCG, and has second best result on Clicks thus achieving the highest overall Borda score of 329. Over 100 teams participated in this challenge applying a wide range of CF approaches, so these results demonstrate that our two-stage model provides an effective way to achieve leading accuracy while remaining efficient during inference.

Examples of test playlists continuations are shown in Table 2. Here, we show first five songs contained in each playlist and top-3 suggestions produced by the model. We see that the model is able to accurately capture and continue playlist genre – from Hip-Hop/Pop in the first playlist, to EDM in the second playlist and classical pieces in last one. The recommendations are also diverse and feature songs by different artists. We consistently observed artist diversity in recommended song lists even though the model was not directly optimized for this during training.

4 CONCLUSION

In this paper, we described our two-stage approach for the 2018 ACM RecSys Challenge. This challenged focused on automated playlist continuation and was organized by Spotify. In the first stage we use a combination of latent, temporal and neighbor-based models to retrieve a set of candidates with high recall. Then in the second stage the candidates are re-ranked using extensive pairwise information. This approach allows to apply complex feature engineering in the second stage without sacrificing run-time performance, and we are able to provide end-to-end recommendations in under 50ms for each palylist . Our model achieved highly competitive performance placing first out of over 100 teams.

REFERENCES

[1] Andreja Andric and Goffredo Haus. 2006. Automatic playlist generation based on tracking user's listening habits. *Multimedia Tools and Applications* (2006).

[2] Shaojie Bai, J Zico Kolter, and Vladlen Koltun. 2018. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271* (2018).

[3] Robert M Bell and Yehuda Koren. 2007. Lessons from the Netflix prize challenge. *ACM SIGKDD Explorations Newsletter* (2007).

[4] Tianqi Chen and Carlos Guestrin. 2016. Xgboost: A scalable tree boosting system. In *International Conference On Knowledge Discovery and Data Mining*.

[5] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep neural networks for YouTube recommendations. In *ACM Recommender Systems Conference*.

[6] Yann N Dauphin, Angela Fan, Michael Auli, and David Grangier. 2017. Language modeling with gated convolutional networks. In *International Conference on Machine Learning*.

[7] Cedric De Boom, Rohan Agrawal, Samantha Hansen, Esh Kumar, Romain Yon, Ching-Wei Chen, Thomas Demeester, and Bart Dhoedt. 2017. Large-scale user modeling with recurrent neural networks for music discovery on multiple time scales. *Multimedia Tools and Applications* (2017).

[8] Yifan Hu, Yehuda Koren, and Chris Volinsky. 2008. Collaborative filtering for implicit feedback datasets. In *IEEE International Conference on Data Mining*.

[9] Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*.

[10] Yehuda Koren. 2008. Factorization meets the neighborhood: A multifaceted collaborative filtering model. In *International Conference on Knowledge Discovery and Data Mining*.

[11] François Maillet, Douglas Eck, Guillaume Desjardins, and Paul Lamere. 2009. Steerable Playlist Generation by Learning Song Similarity from Radio Station Playlists. In *International Society for Music Information Retrieval Conference*.

[12] David Opitz and Richard Maclin. 1999. Popular ensemble methods: An empirical study. *Journal of artificial intelligence research* 11 (1999).

[13] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. 2001. Item-based collaborative filtering recommendation algorithms. In *International Conference on World Wide Web*.

[14] Markus Schedl, Hamed Zamani, Ching-Wei Chen, Yashar Deldjoo, and Mehdi Elahi. [n. d.]. RecSys Challenge 2018: Automatic Playlist Continuation.

[15] Koen Verstrepen, Kanishka Bhaduriy, Boris Cule, and Bart Goethals. 2017. Collaborative filtering for binary, positiveonly data. *ACM SIGKDD Explorations Newsletter* 19, 1 (2017).

[16] Chao-Yuan Wu, Amr Ahmed, Alex Beutel, Alexander J Smola, and How Jing. 2017. Recurrent recommender networks. In *International Conference on Web Search and Data Mining*.