12-2019

SeER: An Explainable Deep Learning MIDI-based Hybrid Song Recommender System

Khalil Damak
University of Louisville

Olfa Nasraoui
University of Louisville, olfa.nasraoui@louisville.edu

SeER: An Explainable Deep Learning MIDI-based Hybrid Song Recommender System

Khalil Damak
Knowledge Discovery and Web Mining Lab, CECS Department, University of Louisville
Louisville, USA
khalil.damak@louisville.edu

Olfa Nasraoui
Knowledge Discovery and Web Mining Lab, CECS Department, University of Louisville
Louisville, USA
olfa.nasraoui@louisville.edu

ABSTRACT

State of the art music recommender systems mainly rely on either matrix factorization-based collaborative filtering approaches or deep learning architectures. Deep learning models usually use metadata for content-based filtering or predict the next user interaction by learning from temporal sequences of user actions. Despite advances in deep learning for song recommendation, none has taken advantage of the sequential nature of songs by learning sequence models that are based on content. Aside from the importance of prediction accuracy, other significant aspects are important, such as explainability and solving the cold start problem. In this work, we propose a hybrid deep learning model, called "SeER", that uses collaborative filtering (CF) and deep learning sequence models on the MIDI content of songs for recommendation in order to provide more accurate personalized recommendations; solve the item cold start problem; and generate a relevant explanation for a song recommendation. Our evaluation experiments show promising results compared to state of the art baseline and hybrid song recommender systems in terms of ranking evaluation. Moreover, based on proposed tests for offline validation, we show that our personalized explanations capture properties that are in accordance with the user's preferences.

KEYWORDS

hybrid recommender system, deep learning, recurrent neural networks, matrix factorization, music recommender system, explainability, user cold start problem, explainable AI

1 INTRODUCTION

Recommendation is becoming a prevalent component of our daily lives that has attracted increasing interest from the Machine Learning research community in recent years. Among the fields in which recommendation is most decisive is music. Music streaming platforms are indeed numerous: Spotify [38], Pandora [31], YouTube Music [48] and many others. However, what makes the success of a platform is its capacity to predict which song the user wants to listen to at the moment given their previous interactions. The most accurate recommender systems rely on complex black box machine learning models that do not explain why they output the predicted recommendation. In fact, one main challenge is designing a recommender system that mitigates the trade-off between explainability and prediction accuracy [3]. Today, the most widely used techniques in music recommendation are matrix factorization (MF)-based collaborative filtering approaches [27] and deep learning architectures [49]. MF is based on similarities between users and items in a latent space obtained by factorizing the rating matrix into user and item latent factor matrices [19]. For state of the art deep learning recommender systems, there are mainly two approaches. The first approach relies on content based filtering [42] using metadata to recommend items. The second approach uses sequence models [26] [8] [16] to predict the next interaction (played song) given the previous interactions [14][39][47]. Despite the advances in deep learning for song recommendation and despite the sequential nature of songs that makes them naturally adapted to sequence models, no work has used sequence models with the **content** of songs for recommendation. Aside from accuracy and explainability, the cold start problem is a significant issue for collaborative filtering recommender systems [2]. In fact, most recommender systems need an initial history of interactions (ratings, clicks, plays, etc.) to recommend items. In music streaming platforms, new users and songs are constantly added making solving this issue crucial. In this work, we take advantage of the sequential nature of the songs, the prediction power of MF and the superior capabilities of deep learning sequence models to achieve the following objectives:

- Propose a method to transform the Musical Instrument Digital Interface (MIDI) format [1] of songs into multidimentional time series to be used as input to deep learning sequence models and keep a large amount of information about the song;
- Integrate content based filtering using deep learning sequence models into collaborative filtering MF to build a novel hybrid model that provides accurate predictions compared to baseline recommender systems, solves the item cold start problem, and provides explanations to the recommendations; and
- Propose a new type of explanation to song recommendation that consists of presenting to the user a short personalized MIDI segment of the song that characterizes the portion that the user is predicted to like the most.

2 RELATED WORK

Various recommender systems rely on sequence models. However, not all of them use them for recommendation with user preferences. In fact, some are session-based CF models [14][39][47] that predict the next interaction in a sequence of interactions regardless of the user's personal preferences. Other methods introduce content to session-based recommendation [15][37] and prove that side information enhances the recommendation quality [49]. Other recommender systems using sequence models take into consideration

Table 1: Notation used in Section 3.

Symbol/Notation	Definition
U	User embedding (latent) matrix
U_u	User u's latent vector
S	Song lookup matrix
S_s	Song s's flattened array
x^s	Song s's array (multidimensional time series)
x_t^s	Song s's array at time step t
$x^{s,k}$	Array of segment k from song s
$x_u^{s,exp}$	Explainability segment array of song recommendation s to user u
R	Training data
r_{us}	Actual rating of user u to song s
$\hat{r}_{us}$	Predicted rating of user u to song s
$\hat{r}_{us}^k$	Predicted rating of user u to segment k of song s
$h_t^{<m>,s}$	Hidden state of sequence model layer m at time step t on input song s
T	Normalized number of time steps
$\|.\|$	Cardinality
$\cdot$	Dot product
$\nabla_w J$	Gradient of J with respect to w

user identification [46][45]. These engines model temporal dependencies for both users and movies [46][45] and generate reviews [45]. The main objective of these models is to predict ratings of users to items using seasonal evolutions of items and user preferences in addition to user and item latent vectors. Alternate models aimed to generate review tips [25], predict the returning time of users and predict items [17] or produce next item recommendations for a user by proposing a novel Gated Recurrent Unit [8] (GRU) structure [10]. Finally, some recommender systems also use sequence models as a feature representation learning tool [49]. [5] creates a latent representation of items and uses it as input to a CF model with a user embedding to predict ratings. On the other hand, song recommendation received contributions from few hybrid models that often diverge in terms of input data and features created. In fact, music items can be represented by features derived from audio signals, social tags or web content [40]. Among the most noticeable hybrid song recommender systems, [44] learns latent factors of users and items using matrix factorization and sums their product with the product obtained with created user and song features. [6] combines non-negative MF and graph regularization to predict the inclusion of a song in a playlist. [30] learns artist embeddings from biographies and track embeddings from audio spectrograms, then aggregates and multiplies them by user latent factors obtained by weighted MF to predict ratings. [41] trains a Convolutional Neural Network [22] on spectrograms of song samples to predict latent features obtained with an MF approach for songs with no ratings. Finally, [4] positions the users in a mood space, given their favorite artists, and recommends new artists using similarity measures.

3 METHODS

In this section, we start by describing the data that we used along with its preparation procedure. Then, we present our model called "Sequence-based Explainable Recommender system" (SeER). This being done, we describe our explainability process called "Segment Forward propagation". But first, we show the list of variables that are used in this section in Table 1 to ease the reading of the remainder of this article.

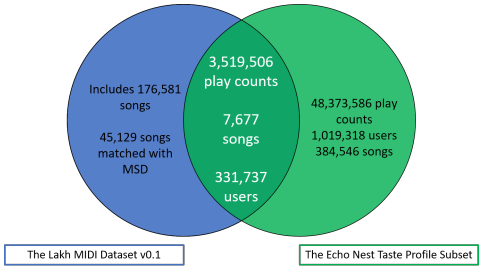


Figure 1: Our dataset resulting from the intersection between "The Lakh MIDI Dataset v0.1" and "The Echo Nest Taste Profile Subset".

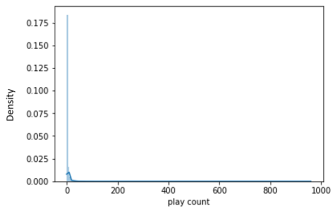
3.1 Data Preparation

We needed a dataset that includes both user to item interactions and song content data. Thus, we used two datasets from the Million Song Dataset (MSD) [7]. The Echo Nest Taste Profile Subset [28] includes 48,373,586 play counts of 1,019,318 users to 384,546 songs collected from The Echo Nest's undisclosed partners. The Lakh MIDI Dataset includes 45,129 unique MIDI files matched to MSD songs [33] [34]. We combined both datasets by taking the intersection in terms of songs as presented in Fig. 1. Then, we followed the same methodology used in [13] to reduce the sparsity of the data. We filtered out users that interacted with less than 20 unique songs. We obtained a dataset with 32,180 users, 6,442 songs with available MIDI files, and 941,044 play counts. Our dataset has a sparsity of 99.54%.

We preprocessed our dataset by first mapping the play counts to ratings in order to remove outliers. To prove the necessity of this step, we show the distribution and statistics of the play counts in Fig. 2. The play counts follow a power law distribution with a median of 1. Also, there are users that listened to the same song hundreds and thousands of times; and the maximum play count is 3,532. These high play counts are outliers that may bias the training of the model. While it is true that the more a song is listened to by a user, the more likely the user likes it, whether a user listens to a song 10 or 3,000 times, it is clear that they like it. Hence, both cases should be considered the same. Therefore, we used the statistics of the play counts to map them to ratings as shown in Fig. 3. Next, we created the input to train sequence models by transforming each MIDI file into a multidimensional time series. MIDI files are polyphonic digital instrumental audios that are usually used to create music. They are constituted of event messages that are consecutive in time [1]. Each message includes a type (such as a note), notation (the note played), time (the time it is played) and velocity (how rapidly and forcefully it is played) [43]. These events are distributed over 16 available channels of information, which are independent paths over which messages travel [1]. Each channel can be programmed to play one instrument. Thus, a MIDI file can play up to 16 instruments simultaneously. We first used "MIDICSV" [43] to translate the MIDI files into sheets of the event messages. We only considered the "Note on C" events to focus our interest on the sequences of notes played throughout time. Thus, we extracted the notes that are played within the 16 channels with their velocities. As a result, each transformed multidimensional time series is constituted of a

Mean	3.08
Median	1
1st quartile	1
3rd quartile	3
Highest datum	6
Minimum	1
Maximum	3,532

(a)



(b)

Figure 2: Play count statistics: (a) represents the statistics of the play count and (b) represents the density plot of the play count (filtered play counts < 1000 for better visualization).

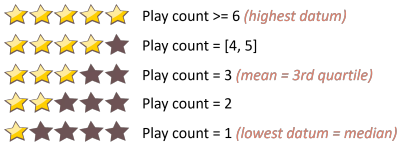


Figure 3: Play count normalization into 5-star ratings.

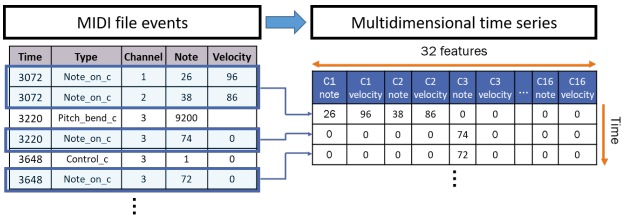


Figure 4: MIDI to multidimensional time series transformation process.

certain number of rows representing the number of "Note on C" events and 32 features representing the notes and velocities played within the 16 channels. The transformation process is summarized in Fig. 4. We then normalized the number of time steps to the median number of timesteps of the songs in our dataset (2,600) to be able to train with mini-batches [24]. At least 50% of the songs kept all their notes and 75% of the songs kept at least half of their notes. Finally, in order to avoid duplicates of the same song in the input and ensure memory efficiency, we created a song lookup matrix by flattening each multidimensional time series into one row in the matrix.

3.2 SeER

Three main observations motivated the design of our model. **First**, the sequential nature of songs, particularly represented by MIDI files, can be best modeled using sequential models. **Second**, the hidden state (output) of a sequence model is both learnable and of chosen size, both being basic properties of an embedding matrix. Thus, we opted to assimilate it to a user embedding. **Third**, sequence models can propagate instances with varying time steps. This inspired us to try to explain the recommendations using song

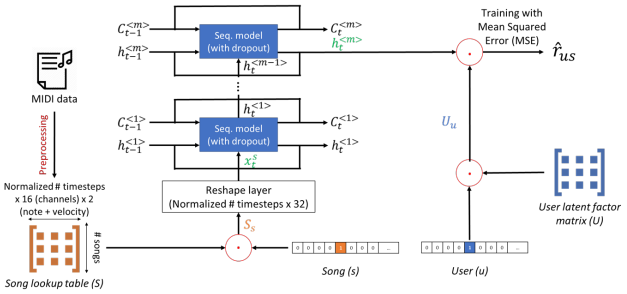


Figure 5: Structure of SeER: For every training tuple (user, song, rating), the model extracts the corresponding user latent vector and flat song array from the user latent matrix and the song lookup matrix respectively. The song array is reshaped to its original 2-dimensional format and input to a sequence model. The resulting song hidden state vector is multiplied with the user latent vector to predict the rating.

segments. These motivations led us to design our model, called "SeER": a sequence-based explainable hybrid song recommender system with the structure presented in Fig. 5. SeER takes as input the song lookup matrix and a user embedding matrix. For each user, song, and rating triplet (u, s, r_{us}) in the training data R, we extract the corresponding latent factor vector U_u of the user and the flattened song array S_s. The latter process is illustrated in Fig. 5 with multiplications of the user embedding and song lookup matrices with one hot vectors of u and s respectively. The song array is next reshaped into its two-dimensional original shape (2600 time steps by 32 features). The resulting array x_s is input to a sequence model and, finally, the hidden state of the last layer m at the last time step $(T = 2,600)$ $h_T^{<m>,s}$ is multiplied with the song latent vector U_u to predict a rating of the user to the song $\hat{r}_{us} = U_u \cdot h_T^{<m>,s}$. To be consistent, we chose the size of the sequential hidden state to be the same as the number of user latent features. This enables computing the scalar product of the two latent vectors to yield a predicted rating. The model is trained using the Mean Squared Error (MSE) [23] as a loss function by comparing the actual rating r_{us} to the predicted rating $\hat{r}_{us}$. Thus, our objective function is:

$$J = \frac{1}{|R|} \sum_{(u,s,r_{us})\epsilon \mathrm{R}} (\hat{r}_{us} - r_{us})^2 = \frac{1}{|R|} \sum_{(u,s,r_{us})\epsilon \mathrm{R}} (U_u \cdot h_T^{<m>,s} - r_{us})^2 \tag{1}$$

Note that in Fig. 5, the cell states can be ignored when using Recurrent Neural Networks [26] (RNNs) or GRUs.

The training process of SeER for each epoch is described in Alg. 1.

3.3 Recommendation and Segment Forward Propagation Explainabiliy

The recommendation consists of feeding user and unrated song inputs to the SeER model in Fig. 5 which results in a predicted rating for each input song. The highest predicted ratings yield a list of recommended songs. After generating a song recommendation s to a user u, we explain it by presenting a 10-second MIDI

Algorithm 1 SeER training algorithm (for each epoch) with step size α, using mini-batch gradient descent for simplicity

procedure SeER_training_epoch (song lookup matrix S, user latent factor matrix U, set of mini-batches B, learning rate α, number of sequence model layers m, number of timesteps T)
  for b *in* B **do**
    for (u, s, r_{us}) *in* b **do**
      $U_u \leftarrow One\_hot(u) \cdot U$ ▷ extract latent vector of u
      $S_s \leftarrow One\_hot(s) \cdot S$ ▷ extract flat song array of s
      $x^s \leftarrow Reshape(S_s)$ ▷ reshape S_s to $(T \times 32)$
      $h_T^{<m>,s} \leftarrow Sequence\_model(x^s)$
      $\hat{r}_{us} \leftarrow U_u \cdot h_T^{<m>,s}$ ▷ predict rating of u to s
    end for
    $J = \frac{1}{|b|} \sum_{(u,s,r_{us}) \epsilon b} (U_u h_T^{<m>,s} - r_{us})^2$ ▷ compute prediction loss
    $w \leftarrow w - \alpha \cdot \nabla_w J$ ▷ back propagate (w refers to the parameters of U and the sequence model)
  end for
end procedure

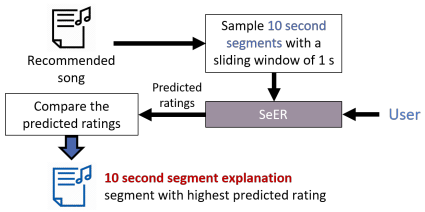


Figure 6: Segment Forward Propagation Explainability.

segment $x_u^{s,exp}$ of the song that tries to capture the most important portion of the recommended song for the input user. First, we sample segments of the MIDI file by using a 10-second sliding window of one-second stride. This means that the first segment is the first 10 seconds of the audio, the second segment is from second 2 to second 11, and so forth, until we reach the end of the song. To do this, we start by creating absolute time segments that we match to MIDI times in the song to determine the range of time steps of each segment. In fact, the time in a MIDI file is in pulses and can be converted to absolute time such that $time[\mu s] = \frac{MIDI\ time[pulses]}{Division[pulses/QR.\ note]} Tempo[\mu s/QR.\ note]$. The division is the number of pulses per quarter note and the tempo is a measure of speed [43]. Then, we create a multidimensional time series $x^{s,k}$ for each segment k by truncating the time series of the recommended song x^s. Finally, we feed each segment's time series $x^{s,k}$ along with the user latent vector U_u as input to the SeER model to estimate a rating $\hat{r}_{us}^k$ of that user to the segment. The segment that obtains the highest predicted rating $\hat{r}_{us}^k$ is presented to the user as an explanation for the song recommendation. We call this explainability process "Segment Forward Propagation Explainability" because it relies on forward propagation of segments to explain the prediction. The aforementioned explanation process is presented in Fig. 6 and is summarized in Alg. 2.

Algorithm 2 Segment Forward Propagation Explainability

procedure Segment_Forward_Propagation (recommended song s, length of s in seconds L, song array x^s, user latent vector U_u, number of timesteps T, trained model $SeER$)
  $abs\_time\_x^s \leftarrow \left[\frac{MIDI\ time(x_t^s)}{Division(x_t^s)} \cdot Tempo(x_t^s) \mid t = 1..T \right]$ ▷ match timesteps to absolute times in x^s
  $abs\_time\_seg \leftarrow [(i, i+9) \mid i = 1..L-9]$ ▷ create absolute time segments
  $song\_segments \leftarrow [x^{s,k} = x^s[i : j] \mid (abs\_time\_x^s[i], abs\_time\_x^s[j])\ in\ abs\_time\_seg]$ ▷ create 10 second segments of x^s
  $seg\_ratings \leftarrow [\hat{r}_{us}^k = SeER(x^{s,k}, U_u) \mid x^{s,k}\ in\ song\_segments]$ ▷ predict ratings for each segment
  $x_u^{s,exp} \leftarrow song\_segments[argmax_k(seg\_ratings)]$ ▷ determine explainability segment
end procedure

Table 2: Top 5 normalized ratings of User 1000 in the dataset.

Artist name	Title	Genres	Rating
Dido	White Flag	Pop, Hip-hop	5
Travie McCoy	Billionaire [ft. Bruno Mars]	Pop	5
Dido	Thank You	Pop	4
Alicia Keys ft. Adam Levine	Wild Horses	Neo-soul	4
Michael Bublé	Put Your Head On My Shoulder	Easy listening	2

Table 3: Example of top 5 recommendation predicted by SeER to user 1000 (partially listed in Table 2). The explanations are represented by the start and end times of the 10-second samples in μs.

Artist name	Title	Genres	Predicted rating	Explanation
Andreas Johnson	Glorious	Alternative/Indie, Pop	5.360812	(130074061.0, 139999986.0)
The Knack	My Sharona	Rock	5.346163	(11172411.0, 20937925.8)
Cat Stevens	Trouble	Singer-songwriter	5.330237	(24230213.1, 33972849.8)
CoCo Lee	Before I Fall In Love	Contemporary R&B	5.314626	(126034512.0, 135942920.0)
Red Hot Chili Peppers	Blood Sugar Sex Magik	Alternative/Indie	5.290801	(248107860.0, 257837580.0)

In order to illustrate the SeER recommendation and explainability processes, we compute the top 5 recommendations for User number 1000 whose top 5 rated songs are listed in Table 2. The top 5 recommendations with explanations for this user are shown in Table 3. The explanations are presented with the start and end times of the 10-second samples in μs. We provide a link to a video[1] demo where these explainability segments can be heard.

4 OFFLINE EXPERIMENTAL EVALUATION

In this section, we describe an offline evaluation pipeline aiming to assess the recommendation performance and capabilities of our model.

[1] https://drive.google.com/file/d/1imlh4nPFhXetE1jCzPkGPm8XRxUxcRJR/view?usp=sharing

Table 4: Hyperparameter tuning results: MAP@10 on the test data after 20 epochs. Best results (in bold) obtained, first, with 150 latent features, then, with GRU.

Hyperparameter	Value	MAP@10
# of latent features	50	0.1236
	100	0.1424
	150	**0.1433**
	200	0.1425
Sequence model type	LSTM	0.1433
	RNN	0.0973
	GRU	**0.1437**

4.1 Experimental Setting

We used the same 80/20% train/test split for all the experiments in order to be consistent when comparing two models or when reproducing an experiment. Due to computational and time constraints, we trained all the models in 20 epochs and evaluated the results in terms of recommendation ranking using Mean Average Precision at cutoff K (MAP@K). Furthermore, in order to assess the statistical significance when comparing two models, we replicated each experiment 5 times and applied statistical tests.

4.2 Hyperparameter Tuning

We fixed the number of sequence model layers to 1 and the batch size to 500 because of memory constraints. Also, we relied on the Adaptive Moment Estimation (Adam) [18] optimizer because it yields a relatively fast convergence and adapts the learning rate for each parameter [13]. Finally, we tuned the number of latent features from 50 to 200 with increments of 50 and the sequence model type by trying RNN, GRU and Long Short-Term Memory [16] (LSTM) networks. We relied on a greedy approach, that consists of varying the hyperparameters one by one independently from each other. We started by initializing the sequence model type to LSTM and tuned the number of latent features. Then, we varied the sequence model type. The results are presented in Table 4. We obtained the best performance with 150 latent features and GRU.

4.3 Research Questions

To evaluate the prediction ability of our model, we made both wide and narrow comparisons. For the wide comparison, we matched our model to baseline recommender systems regardless of their types and data nature. On the other hand, the narrow comparison consists of comparing our model to its closest competitors which are state of the art hybrid song recommender systems. This leads us to formulate our first two research questions: **RQ1:** How does our model compare to baseline recommender systems? and **RQ2:** How does our model compare to state of the art (SOTA) hybrid song recommender systems? Also, SeER can be seen as an updated version of MF with the item embedding matrix being replaced with the output of a sequence model that takes as input our preprocessed song content data. Thus, we assess the importance of the way we use the content data by comparison to MF in the third research question: **RQ3:** What is the importance of our use of the content data? Finally, we assess whether our explanations share similar characteristics.

Table 5: Comparison of SeER with baseline models: MAP@10 results after 20 epochs for 5 replicates.

Replicate	SeER	MF	NeuMF	ItemPop
1	**0.1436**	0.1289	0.1314	0.0778
2	**0.1481**	0.1292	0.1303	0.0778
3	**0.1399**	0.1285	0.1366	0.0778
4	**0.1453**	0.1266	0.1376	0.0778
5	**0.1414**	0.1288	0.1378	0.0778
Average	**0.1437**	0.1284	0.1347	0.0778

The intuition behind this is that the shared characteristics may be interpreted as user preferences captured and incorporated in the explanations. This would indicate that the explanations represent the preferences of the user and are not an artificial product of the model. This is translated in **RQ4:** Do the personalized explanations share similar characteristics?

4.4 RQ1: How does our model compare to baseline recommender systems?

The baseline recommender systems we used for comparison are:

- **Matrix Factorization [27]:** One of the most used collaborative filtering techniques and basis of a large number of recommender systems including ours. We used the same number of latent factors as our model which is 150.
- **NeuMF [13]:** State of the art collaborative filtering technique that combines Generalized Matrix Factorization [13] (GMF) and Multi-Layer Perceptron [9] (MLP). We replaced its output layer with a dot product and used MSE as a loss function because we are working with ratings. We used three hidden layers for MLP and 150 latent features for all embedding matrices.
- **ItemPop [35]:** Most popular item recommendation. Used to benchmark the recommendation performance.

We present the results obtained with each model in Table 5. Our model yields an average MAP@10 of 0.1437 which is higher than all the other methods. It also has the benefit of being explainable. Furthermore, we validated our results with ANOVA [11] and Tukey [12] tests. All the p-values were lower than 0.01 meaning that our model performs significantly better than all the other models. **Note that comparing our model to MF can be considered as an ablation study that aims to prove the importance of the sequence model layers and the use of the content data. In fact, replacing the sequence model layers with an embedding layer reduces our model to MF.**

4.5 RQ2: How does our model compare to SOTA hybrid song recommender systems?

The most related hybrid song recommender system we found is [30]. It applies MF-, Convolutional Neural Network [21] (CNN)- and MLP-based [9] models on play counts, audio spectrograms and artist biographies to generate recommendations. The dataset used is a subset of the MSD that overlaps with ours. Their dataset includes around 1M users and 328,821 songs. We compared our model

Table 6: Comparison of SeER with MM-LF-LIN [30] on an overlapping dataset. Our model's performance is assessed with MAP@500 after 20 epochs with 5 replicates.

Replicate	SeER	MM-LF-LIN
1	0.1438	0.0036
2	0.1483	-
3	0.1400	-
4	0.1455	-
5	0.1415	-
Average	**0.1438**	0.0036

directly to the results in [30] using the same evaluation process that they used. Although comparing two models on overlapping datasets is unconventional, the results can give us an idea about the ranges in which the ranking performances of the two models are. The best performing configuration, MM-LF-LIN [30], presents a MAP@500 of 0.0036, which is significantly lower (ANOVA p-value < 0.01) than our average performance of 0.1438 as presented in Table 6.

4.6 RQ3: What is the importance of our use of the sequential content data?

We can assess the importance of using the sequential song content data as follows:

- First, as we already proved in RQ1, the content data helped improve the recommendation performance since our model performs significantly better than pure rating-based MF.
- Second, the content data allowed us to solve the item cold start problem because the item data comes from both the song's MIDI content data and the user ratings. Thus, items with no ratings can be recommended by relying solely on their content.
- Finally, the sequential nature of our content data, in addition to the structure of our model, allowed us to generate 10-second instrumental explanations making recommendation more transparent. The explanations are evaluated in RQ4 below.

4.7 RQ4: Do the personalized explanations share similar characteristics that capture user-preferences?

In order to validate the 10-second segment explanations offline, we tried to determine, for every user, whether their personalized explanations share common characteristics. Explanations that share common properties are likely to be generated based on captured hidden preferences of the user. Hence, these explanations may represent the most important sections of the recommended songs. In that case, the explanations may not be just artefacts. To study the latter property, we propose two approaches based on analysis of the song content similarities and tags respectively.

4.7.1 Content-based validation. Given that we have the content of the explanations, we first relied on similarity measures to prove that they share similar characteristics. We randomly selected 100 users as our test sample. For every test user, we use our model to generate the top 5 recommendations with explanations and compute the average Dynamic Time Warping (DTW) [36] distance between the explanations (DTWe). DTW is a powerful distance measure between multidimensional time series that do not necessarily have the same size. To generate the average DTW distance between two lists of multidimensional time series, we compute the DTW distance matrix between them and take the average of all the values in the matrix. In the case of DTW distances between explanations, both lists are similar and include the song arrays of the generated 5 explanations. To compare, we selected a random 10-second segment from every recommended song and computed the average DTW distance between these 5 segments (DTWr) for every user. We compare to average DTW distances between 10-second segments instead of between the whole recommended songs to avoid any bias coming from the different song lengths. Finally, we considered the problem as a Randomized Complete Block Design (RCBD) [29] and applied a Tukey test [12] for pairwise comparison. The null hypothesis is whether the averages over all the users of the average DTW distances between the explanations and between the random segments are similar. For simplicity, we will call these two entities "Avg. DTW between explanations" (or DTWe) and "Avg. DTW between random segments" (or DTWr). We show these average values with the 95% Confidence Intervals (CIs) of the difference (DTWe - DTWr) and the statistical test results in Table 7. We notice that Avg. DTW between explanations is significantly smaller than Avg DTW between random segments (p-value<0.05 and 0 is not in the Confidence Interval). This means that for each user, we can assert with 95% confidence that the explanations are significantly close to each other compared to the random segments from the recommendations. **Thus, we can assert that our generated 10-second segment explanations share common characteristics which are likely to represent the preferences of the user.**

Table 7: Significance testing with 95% confidence of the difference between Avg. DTW between explanation and Avg. DTW between random segments: The explanations are significantly close to each other compared to the random segments. This means that the explanations capture and share common characteristics that are likely to represent the user's preferences.

Avg. DTW between explanations (DTWe)	Avg. DTW between random segments (DTWr)	95% CI of the difference (DTWe - DTWr)	Adjusted p-value
22,844.1	24,820.9	(182, 3,771)	**0.031**

4.7.2 Tag-based validation. Tags can capture an item's properties. In the case of songs, they can include genres, the era, the name of the artist or subjective emotional descriptions. We used the tags from the "Last.fm" dataset [7] provided with the MSD. These tags are available for almost every song in the MSD and amount to 522,366 tags [7]. In our dataset, we selected the songs that intersect with the "Last.fm" dataset and filtered the tags that occur in at least 100 songs in order to remove noisy tags. We obtained 4,659 songs with 227 tags. From the users that interacted with these songs, we filtered the ones that have at least 10 liked songs. In fact, we made the assumption that a rating strictly higher than 3 means that the

user likes the song. Next, we randomly selected 100 users as our test sample. For every user, we determined the Top 1, 2 and 3 preferred tags, based on the tags of their liked songs, and generated the top 5 recommendations with explanations. Our objective is to determine how much the personalized recommendations and explanations match the preferred tags of every user. Thus, we need to determine the tags of both the recommendations and the explanations, which are not necessarily in the tags dataset. To cope with this issue, we trained a multi-label classification model on our tags dataset to predict the tags of the recommendations and explanations. The classification model is basically a sequence model layer with 20% dropout followed by Multilayer Perceptron (MLP) [32] layers with ReLU activation functions and an output layer with 227 nodes, corresponding to the 227 classes, each with a Sigmoid activation function. The model is trained to optimize the Binary Cross-entropy loss to predict the probability of each tag individually in every node [20]. To tune our model's hyperparameters, we started with an LSTM layer followed by the output layer. We tuned the size of the hidden state from 100 to 500 with an increment of 100. Then, we tuned the number of MLP hidden layers from 1 to 5. We chose the number of nodes in the hidden layers to be the optimal size of the hidden state, which is 300. Finally, we tuned the sequence model type of the first layer by additionally trying RNN and GRU. The best model has one LSTM layer with a hidden state size of 300 followed by 4 MLP layers of the same size and, finally, the output layer. We reached a performance of 93.4% accuracy and respectively 51.8%, 61.9% and 67.7% top-1, top-2 and top-3 categorical accuracy with 5-fold cross validation. We used top-k categorical accuracy [20] because we are interested in correctly predicting the existing tags in a sparse target. We used our trained classifier to predict the tags of all the recommendations and explanations for all the users. Then, we calculated the Average Percentage Matching of the recommendations and explanations with the top 1, 2 and 3 user preferred tags. We define the Percentage Matching of a list of songs S with the top k preferred tags $T_k(u)$ of a user $u \epsilon U$ as the percentage of songs from S including at least one of the top k preferred tags $T_k(u)$, as follows:

$$\%\,Matching(S, T_k(u)) = \frac{100}{|S|}|\{s \epsilon S | Tags(s) \cap T_k(u) \neq \emptyset\}| \quad (2)$$

$Tags(s)$ is the set of tags of the song s. In our case, the set of tags of a recommendation or an explanation is predicted using the multi-label classification model. The Average Percentage matching is the average of the Percentage Matching over all the test users:

$$Avg\,\%\,Matching(S, U, k) = \frac{100}{|U|}\sum_{u=1}^{|U|} \%\,Matching(S(u), T_k(u)) \quad (3)$$

$S(u)$ in our case is either the set of recommendations or explanations of user u. We varied k and considered every problem as a RCBD [29]. We applied Tukey tests [12] for pairwise comparison. The null hypothesis for every test is whether the average percentage matchings of the recommendations and of the explanations with the top k liked songs (Avg%Matching(rec., U, k) and Avg%Matching(exp., U, k) respectively) are equal. We show the two average percentage matching values with the corresponding 90% CIs of the differences (Avg%Matching(rec., U, k) - Avg%Matching(exp., U, k)) and adjusted p-values of the Tukey tests in Table 8. We notice that for all k, the explanations match the preferred tags of the users more than the recommendations. The difference is significant for k = 1 (CI of the difference does not include 0 and p-value<0.1). However, starting from k = 2, the difference becomes insignificant as both the recommendations and explanations start matching the top k preferred tags comparably well, but still with a slight advantage for the explanations. **This means that the explanations share similar properties which are more in accordance with the preferred tags of the users than even the overall recommendations.** Assuming that the tags represent the genres, if the user's preferred genre is, for instance, "Rock" and a "Pop" song gets recommended, the explanation of that song is likely to be a "Rock" segment of the song. We show an illustrative example of a user from our test sample in Table 9.

Table 8: Significance testing with 90% confidence of the difference between the Avg % Matching of recommendations and explanations with user top k preferred tags.

k	Avg%Matching (rec., U, k)	Avg%Matching (exp., U, k)	90% CI of the difference	Adjusted p-value
1	84.24%	**84.85**%	(-0.01181, -0.00031)	**0.083**
2	90.71%	**90.91**%	(-0.00537, 0.00133)	0.320
3	94.75%	**94.95**%	(-0.00537, 0.00133)	0.320

Table 9: Example of a test user (#26647) where the explanations match the favorite tags more than the recommendations: The first recommended song is a "pop" song. However, the explainability segment is both "pop" and "rock" which matches the favorite tags of the user better than the recommendation itself.

Recommendation	Recommendation tags	Explanation tags
1	**pop**	pop, rock
2	pop, rock	pop, rock
3	pop, rock	pop, rock
4	pop, rock	pop, rock
5	pop, rock	pop, rock
User top 3 tags (sorted)	**rock**, pop, favorites	

k	1	2	3
$\%\,Matching(rec., T_k(u))$	**80%**	100%	100%
$\%\,Matching(exp., T_k(u))$	100%	100%	100%

5 CONCLUSION

We proposed a hybrid song recommender system that uses both ratings and song content to generate personalized recommendations accompanied with short MIDI segments as explanations. We made recommendation more transparent while relying on powerful deep learrning models. Our experiments demonstrated that our architecture performs significantly better than both baseline and SOTA hybrid song recommender systems. Moreover, we validated the effectiveness of the way we integrate the content data and solved the item cold start problem which is a notorious limitation of Collaborative Filtering techniques. Finally, we validated our explainability

approach by showing that the personalized explanations are able to capture properties that are in accordance with the preferences of the user. Our approach has limitations such as the slow training time and the user cold start problem. In the future, we plan to extend our methods to more complex and challenging modalities such as images and videos.

REFERENCES

[1] [n. d.]. Introduction to MIDI and Computer Music: The MIDI Standard. http://www.indiana.edu/~emusic/361/midi.htm. Accessed: 2019-03-11.

[2] Behnoush Abdollahi. 2017. *Accurate and justifiable : new algorithms for explainable recommendations.* Ph.D. Dissertation. University of Louisville.

[3] Behnoush Abdollahi and Olfa Nasraoui. 2017. Using Explainability for Constrained Matrix Factorization. In *Proceedings of the Eleventh ACM Conference on Recommender Systems (RecSys '17)*. ACM, New York, NY, USA, 79–83. https://doi.org/10.1145/3109859.3109913

[4] Ivana Andjelkovic, Denis Parra, and John O'Donovan. 2018. Moodplay: Interactive Music Recommendation based on Artists' Mood Similarity. *International Journal of Human-Computer Studies* (2018), –. https://doi.org/10.1016/j.ijhcs.2018.04.004

[5] Trapit Bansal, David Belanger, and Andrew McCallum. 2016. Ask the GRU: Multi-Task Learning for Deep Text Recommendations. https://doi.org/10.1145/2959100.2959180 cite arxiv:1609.02116Comment: 8 pages.

[6] Kirell Benzi, Vassilis Kalofolias, Xavier Bresson, and Pierre Vandergheynst. 2016. Song recommendation with non-negative matrix factorization and graph total variation. *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (2016), 2439–2443.

[7] Thierry Bertin-Mahieux, Daniel P.W. Ellis, Brian Whitman, and Paul Lamere. 2011. The Million Song Dataset. In *Proceedings of the 12th International Conference on Music Information Retrieval (ISMIR 2011)*.

[8] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, 1724–1734. https://doi.org/10.3115/v1/D14-1179

[9] Yann Le Cun. 1988. A Theoretical Framework for Back-Propagation.

[10] Tim Donkers, Benedikt Loepp, and Jürgen Ziegler. 2017. Sequential User-based Recurrent Neural Network Recommendations. In *Proceedings of the Eleventh ACM Conference on Recommender Systems (RecSys '17)*. ACM, New York, NY, USA, 152–160. https://doi.org/10.1145/3109859.3109877

[11] R.A. Fisher. 1925. *Statistical methods for research workers.* Edinburgh Oliver & Boyd.

[12] Winston Haynes. 2013. *Tukey's Test.* Springer New York, New York, NY, 2303–2304. https://doi.org/10.1007/978-1-4419-9863-7_1212

[13] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural Collaborative Filtering. In *Proceedings of the 26th International Conference on World Wide Web (WWW '17)*. International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, Switzerland, 173–182. https://doi.org/10.1145/3038912.3052569

[14] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. 2015. Session-based recommendations with recurrent neural networks. *arXiv preprint arXiv:1511.06939* (2015).

[15] Balázs Hidasi, Massimo Quadrana, Alexandros Karatzoglou, and Domonkos Tikk. 2016. Parallel Recurrent Neural Network Architectures for Feature-rich Session-based Recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems (RecSys '16)*. ACM, New York, NY, USA, 241–248. https://doi.org/10.1145/2959100.2959167

[16] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Comput.* 9, 8 (Nov. 1997), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

[17] How Jing and Alexander J. Smola. 2017. Neural Survival Recommender. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining (WSDM '17)*. ACM, New York, NY, USA, 515–524. https://doi.org/10.1145/3018661.3018719

[18] Diederik P. Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. *CoRR* abs/1412.6980 (2014). arXiv:1412.6980 http://arxiv.org/abs/1412.6980

[19] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix Factorization Techniques for Recommender Systems. *Computer* 42, 8 (Aug. 2009), 30–37. https://doi.org/10.1109/MC.2009.263

[20] Maksim Lapin, Matthias Hein, and Bernt Schiele. 2017. Analysis and optimization of loss functions for multiclass, top-k, and multilabel classification. *IEEE transactions on pattern analysis and machine intelligence* 40, 7 (2017), 1533–1554.

[21] Yann Lecun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*. 2278–2324.

[22] Yann LeCun, Patrick Haffner, Léon Bottou, and Yoshua Bengio. 1999. Object recognition with gradient-based learning. In *Shape, contour and grouping in computer vision*. Springer, 319–345.

[23] E.L. Lehmann and G. Casella. 1998. *Theory of Point Estimation.* Springer Verlag.

[24] Mu Li, Tong Zhang, Yuqiang Chen, and Alexander J. Smola. 2014. Efficient Mini-batch Training for Stochastic Optimization. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '14)*. ACM, New York, NY, USA, 661–670. https://doi.org/10.1145/2623330.2623612

[25] Piji Li, Zihao Wang, Zhaochun Ren, Lidong Bing, and Wai Lam. 2017. Neural Rating Regression with Abstractive Tips Generation for Recommendation. *CoRR* abs/1708.00154 (2017). arXiv:1708.00154 http://arxiv.org/abs/1708.00154

[26] Zachary Chase Lipton. 2015. A Critical Review of Recurrent Neural Networks for Sequence Learning. *CoRR* abs/1506.00019 (2015).

[27] R. Mehta and K. Rana. 2017. A review on matrix factorization techniques in recommender systems. In *2017 2nd International Conference on Communication Systems, Computing and IT Applications (CSCITA)*. 269–274. https://doi.org/10.1109/CSCITA.2017.8066567

[28] The Echo Nest. [n. d.]. The Echo Nest Taste Profile Subset. https://labrosa.ee.columbia.edu/millionsong/tasteprofile.

[29] Donald M Olsson. 1978. Randomized Complete Block Design. *Journal of Quality Technology* 10, 1 (1978), 40–41.

[30] Sergio Oramas, Oriol Nieto, Mohamed Sordo, and Xavier Serra. 2017. A Deep Multimodal Approach for Cold-start Music Recommendation. *CoRR* abs/1706.09739 (2017). arXiv:1706.09739 http://arxiv.org/abs/1706.09739

[31] pandora. [n. d.]. pandora. https://www.pandora.com/.

[32] Marius-Constantin Popescu, Valentina E. Balas, Liliana Perescu-Popescu, and Nikos Mastorakis. 2009. Multilayer Perceptron and Neural Networks. *WSEAS Trans. Cir. and Sys.* 8, 7 (July 2009), 579–588. http://dl.acm.org/citation.cfm?id=1639537.1639542

[33] Colin Raffel. [n. d.]. The Lakh MIDI Dataset v0.1. https://colinraffel.com/projects/lmd/.

[34] Colin Raffel. 2016. *Learning-Based Methods for Comparing Sequences, with Applications to Audio-to-MIDI Alignment and Matching.* Ph.D. Dissertation. COLUMBIA UNIVERSITY.

[35] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2009. BPR: Bayesian Personalized Ranking from Implicit Feedback. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence (UAI '09)*. AUAI Press, Arlington, Virginia, United States, 452–461. http://dl.acm.org/citation.cfm?id=1795114.1795167

[36] Stan Salvador and Philip Chan. 2007. Toward accurate dynamic time warping in linear time and space. *Intelligent Data Analysis* 11, 5 (2007), 561–580.

[37] Elena Smirnova and Flavian Vasile. 2017. Contextual Sequence Modeling for Recommendation with Recurrent Neural Networks. *CoRR* abs/1706.07684 (2017). arXiv:1706.07684 http://arxiv.org/abs/1706.07684

[38] Spotify. [n. d.]. Spotify. https://www.spotify.com/us/.

[39] Yong Kiam Tan, Xinxing Xu, and Yong Liu. 2016. Improved Recurrent Neural Networks for Session-based Recommendations. *CoRR* abs/1606.08117 (2016). arXiv:1606.08117 http://arxiv.org/abs/1606.08117

[40] Andreu Vall and Gerhard Widmer. 2018. Machine Learning Approaches to Hybrid Music Recommender Systems. *CoRR* abs/1807.05858 (2018). arXiv:1807.05858 http://arxiv.org/abs/1807.05858

[41] Aaron Van den Oord, Sander Dieleman, and Benjamin Schrauwen. 2013. Deep content-based music recommendation. In *Advances in neural information processing systems*. 2643–2651.

[42] Robin van Meteren. 2000. Using Content-Based Filtering for Recommendation.

[43] John Walker. 2004. MIDICSV. https://colinraffel.com/projects/lmd/.

[44] Xinxi Wang and Ye Wang. 2014. Improving Content-based and Hybrid Music Recommendation Using Deep Learning. In *Proceedings of the 22Nd ACM International Conference on Multimedia (MM '14)*. ACM, New York, NY, USA, 627–636. https://doi.org/10.1145/2647868.2654940

[45] Chao-Yuan Wu, Amr Ahmed, Alex Beutel, and Alexander J. Smola. 2017. Joint Training of Ratings and Reviews with Recurrent Recommender Networks *(ICLR 2017)*.

[46] Chao-Yuan Wu, Amr Ahmed, Alex Beutel, Alexander J. Smola, and How Jing. 2017. Recurrent Recommender Networks. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining (WSDM '17)*. ACM, New York, NY, USA, 495–503. https://doi.org/10.1145/3018661.3018689

[47] Sai Wu, Weichao Ren, Chengchao Yu, Gang Chen, Dongxiang Zhang, and Jingbo Zhu. 2016. Personal recommendation using deep recurrent neural networks in NetEase. *2016 IEEE 32nd International Conference on Data Engineering (ICDE)* (2016), 1218–1229.

[48] YouTube. [n. d.]. YouTube Music. https://music.youtube.com/.

[49] Shuai Zhang, Lina Yao, and Aixin Sun. 2017. Deep Learning based Recommender System: A Survey and New Perspectives. *CoRR* abs/1707.07435 (2017). arXiv:1707.07435 http://arxiv.org/abs/1707.07435