Chapter 3

MSA-SRec: Improved Self-Attentive MIDI Content-based Music Recommendation System

3.1 Introduction

As discussed in the two previous chapters, older approaches to recommendation generation, such as those based on Collaborative Filtering face the issues of cold start and data sparsity [121, 122]. The first can be when a new song is introduced and is not rated by any user, and another when a new user logs in to the system and does not have any listening history [123]. Other music recommendation generation systems rely more on content-based recommendation algorithms due to the problems associated with CF-based recommendations. Content-based music recommendation techniques can be considered part of the more comprehensive field of music information retrieval (MIR). MIR aims to retrieve the music data and extract their semantic information, by which applications can estimate the usefulness of music [124].

To recommend suitable music for a user, we need to think based on a user's perspective, and we need to focus on these research areas.

- What is the most suitable music from a user's perspective?
- How can a model learn a user's preferences without using his/her so much personalized information?
- How can we model the dynamic preferences of users, especially in music recommendation systems?

Considering these issues, we tried to develop a new approach for music recommendation, including music content information with user interaction history for recommendation generation. The content-based recommendation has used the song's metadata information in the last few years to get a state-of-the-art recommendation model, e.g., audio and video data. Instead of such data, we use MIDI (Musical Instrument Digital Interface) data, a protocol that allows electronic instruments and digital music tools to communicate with each other [125]. We then feed MIDI information and the user's listening sequences into a deep sequential model with a self-head attention mechanism.

Researchers have used MIDI data for various music applications in different forms in music information retrieval problems like genre classification, music generation, etc. The inclusion of MIDI content in music recommendation is relatively recent and more efficient than audio and lyrics, as it is structured data. Most articles on content-based music recommendation systems include audio data, and some of the studies include lyrics data for resolving cold-start problems.

In our proposed model, we use MIDI data as a piece of additional content information which is a primary source for resolving the cold-start problem. We use MIDI's extracted data in sequential details and use it as input information for a deep learning model. The benefit of using MIDI data in music recommendation is computational cost and lower maintenance. For pre-processing of songs MIDI data, we use Python library *midicsv*[1]. The use of content information in this model resolves the cold start problem of basic collaborative filtering. Instead of using only a sequence model, we use the

[1] `https://pypi.org/project/py-midicsv/`

self-head attention mechanism, now a ubiquitous method in advanced Deep Learning models. In this work, we take advantage of the song's sequential characteristics (habits of listening in terms of sequential patterns) with the self-head attention concept, along with the matrix factorization technique for predicting the next song to be recommended for the user. We also validate our results by reporting on various evaluation metrics that are commonly used for recommendation systems evaluation.

The chapter is organized as follows. Next, we demonstrate our proposed music recommendation framework (MSA-SRec) is designed and implemented in Section 3.2, and the results and the performance evaluation are discussed in Sect. 3.3. The conclusion and future scope are illustrated in Section 3.4.

3.2 Problem Formulation And the Proposed Algorithm

This section describes the approach we have used for MIDI content-based music recommendation.

3.2.1 Data-Preprocessing

The methodology we are proposing requires a dataset which contains user-item interaction history. We describe a music recommendation system that uses content information, which means we also need music content data, so we use MIDI data for content feature extraction of music. A MIDI file is a piece of musical information in a digital format, while MP3 data is an encoded form of digital audio. Another reason for using MIDI data is that it represents the essential musical (score) information, not the audio, leading to more information in less data, easily machine-readable. As our MIDI dataset, we use the Million Song dataset [126], which also contains The Lakh MIDI dataset. The Lakh MIDI dataset has 176,581 different MIDI files, 45,129 of which have been matched and aligned to entries in the Million Song Dataset. We used the Lakh MIDI-matched dataset having 45129 songs matched to entries in the Million Song Dataset.

Apart from the MIDI dataset, we also needed user and song interaction data, for which we use The Echo Nest Taste Profile dataset [127]. The Echo Nest Taste Profile Subset consists of many different datasets. We employed two of them, namely train_triplets.txt and unique_tracks.txt. The Taste Profile dataset has a total of 1019318 unique users and 384546 unique songs, and the user-song play-counts is 48373586. The unique_tracks.txt file retrieves the unique track_id of the song in the data train_triplets.txt. This dataset is very sparse, so we need to preprocess the dataset, where we delete users who have listened to less than 20 songs from the train_triplets dataset. We follow the preprocessing of these listen counts of mapping to rating using the methodology used in [128]. Table 3.1 shows the play count conversion to the rating of the dataset. Figure 3.1 shows the rating distribution plot of the dataset. The rating distribution graph shows that it follows the power-law distribution of rating and considers the original listening count to lead to outliers in model training.

Table 3.1: Mapping of play_counts to a Rating Table

Play_Counts	**Rating**
1	1
2	2
3	3
4,5	4
more than 6	5

Table 3.2 shows a description of the dataset after preprocessing.

Table 3.2: Dataset Description after Preprocessing

Count	**Mean**	**Std**	**Min**	**25%**	**50%**	**Max**
4.3407	1.944	1.381	1.0	1.0	3.0	5.0

3.2.2 MIDICSV Conversion

Researchers in the field of music information retrieval extract the content data of music from various resources. During the early stages, they used meta-data such as the

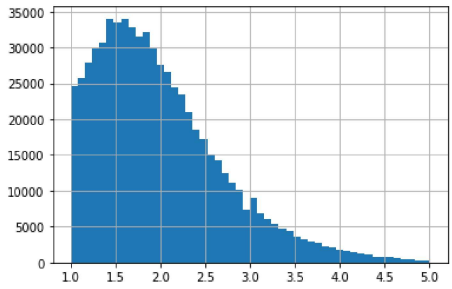

Figure 3.1: Rating Distribution Graph

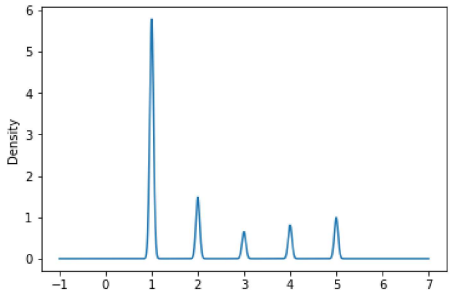

Figure 3.2: Kernel Density Graph of Rating Data

artist name, year of release, genre, etc., for content-based recommendations. In recent years, the extraction of content from audio data like spectrogram information, text information, etc., has also been used for content-based recommendation. We extract the MIDI data into the CSV format and use it as part of the content information in our recommendation system. It is used as an input sequence for training the deep sequence model by turning it into a multidimensional time series. In our proposed methodology, we use a tool called MIDICSV[2] that was initially developed by Fourmilab to convert MIDI files of songs to their corresponding CSV formats. We used the Python implementation of these methods.

As described earlier, MIDI is a technical standard that is about a collection of event messages according to a communications protocol, and it connects various electronic musical instruments, computers, and related audio devices for playing, editing, and recording music. A MIDI file contains sixteen channel information that can be routed

[2]`https://www.fourmilab.ch/webtools/midicsv/`

to separate individual instruments. MIDI carries 16 channel information with notes on and off notations, notes themselves, velocity values and clock signals.

In the current work, we have only used information about the 'Note on C' event message sequence according to the timestamps. A total of 16 channels are present in the event messages, and each channel has a note and a velocity value. So a total of 32 features representing their note and velocity played through 16 channels. The complete conversion process of MIDI to CSV is shown in Figure 3.3. The next step is to normalize the timestamp information because we need to input the sequence information into the model for training. We calculated the median value for the timestamp.

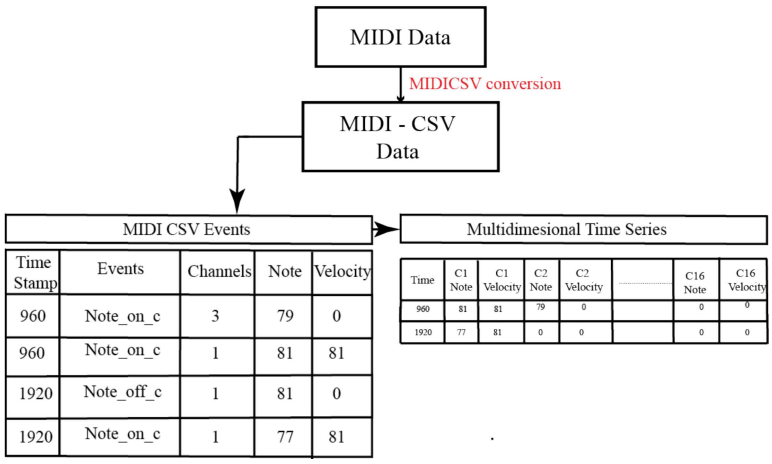

Figure 3.3: MIDI Data to Multidimensional Time Series Transformation

3.2.3 Matrix Factorization

The Echo Nest Taste Profile data contains implicit feedback from the user in the form of a play count for each song for each user. The play count information shows how many times the user listens to a particular song. There are some songs that are not heard by many users; maybe the users might not be aware of these songs, or they might not enjoy them. Our ultimate goal is to predict this unknown rating of a song for a particular user. For rating prediction, we use this implicit feedback of the user as a user-song interaction matrix and apply matrix factorization. Let there be M users and

N songs present, then the complete user-song matrix is of size $M \times N$. The matrix factorization is performed on the user-song interaction matrix. Matrix factorization can be considered as finding two matrices whose product is the original matrix. Here, the purpose of using matrix factorization is to obtain user U_u and item latent factor S_s information. SVD of an $M \times N$ real matrix M is a factorization of the form UEV^T, where U is an $M \times k$ real matrix, E is a $k \times k$ diagonal matrix with non-negative real numbers on the diagonal, and V is an $N \times k$ real matrix.

$$\hat{r}_{us} = U_u^T \times S_s \tag{3.1}$$

The objective function with respect to latent factor U_u and S_s, which are (M,k) and (k,N) matrices, is:

$$min_{p^* q^*} \sum_{u,s \epsilon k} (r_{us} - U_u^T S_s)^2 + \lambda(\|U_u\|^2 + \|S_s\|^2) \tag{3.2}$$

The latent factors of users U_u and songs S_s obtained from this are going to be used in our proposed sequential Deep Learning model (MSA-SRec).

3.2.4 MIDI Based Self Attentive Sequential Music Recommendation (MSA-SRec)

Here in this section, we discuss our proposed model architecture and how our model uses the MIDI features as content information. The main observation in MIDI data is its sequential structure which can be best modelled by the deep sequential models. In our proposed model MSA-SRec we mainly used MIDI files as input embedding matrix with self-attention blocks for recommendation generation. The benefit of using a sequence model is that it will propagate instances with varying time steps. Figure 3.4 represents the architecture of our proposed model. For training our model, we include user, song and rating tuples for obtaining latent user vectors, and flat midi information

of a corresponding array is embedded with the song lookup matrix. Further, these song embeddings are transformed into a two-dimensional format and used as input information for the sequential model. These embeddings of the songs' lookup matrix are further passed with the self-attention blocks for efficient representation. Then the resulting song vector is multiplied by the user latent vector to predict the final rating for the target user.

Over the last few years, attention mechanisms have been implemented in various Deep Learning applications and have found extensive applications in many domains. We applied self-attention in the sequential music recommendation problem. In sequential music recommendation, we have a sequence of songs listened to by the user, and we train our model on these users' actions and predict the next song for the user to hear. In this section, we discuss how to build a sequential model using additional music information. In a content-based music recommendation model, music can be recommended based on the available metadata; here, we use MIDI information in our approach, a significantly less explored area in the music recommendation system. Our key contribution here is to use MIDI data and its different event information like Note_on_c, Note_off_c, Pitch_bend, and Control_on_c information which is not used before in music recommendation generation.

Most of the reported models use artists, audio data, and other available metadata. This can often lead to predictable recommendations. For example, recommending the same artist's songs repeatedly is less useful. We Include MIDI data, along with the sequential recommendation and self-attention blocks. Our model's first step is to prepare an input embedding layer for MIDI data. For input information, we extracted MIDI embedding using a Python library to extract features from MIDI data, which is discussed in section 3.2.2. There is a lot of feature information extracted from MIDICSV for each MIDI file, such as Note_on_C, Note_off_C, Pitch_bend, Control_on_C etc. We use only Note on C information for an embedding generation. Note on C event in

MIDI consists of note and velocity values. We created a matrix using these Note_on_C pieces of information. The matrix is shown in Figure 3.3. In MIDICSV conversion, we processed this information for each timestamp, followed by note and velocity values. This processed information is now transformed into multi-dimensional time series information and will be used as a piece of input information for the sequential model. The mathematical representation of the input is the sequence information of each track's MIDI features $S_M = S_1^m, S_2^m, ...S_n^m$.

Further, we transform the training sequence into a fixed length of size 2600 (median value) for each track, and we consider only the current timestamp values for tracks, which creates an input embedding matrix $E = R^{n \times d}$. Further, we use these matrices as multi-dimensional time-series information as we process it with the one-hot vector of each track for which we have MIDI files available. This embedding layer is used in the sequence model as a recurrent layer. The output from these consecutive recurrent layers is further passed as a piece of input information for the self-attention block.

Self-attention is also called intra-attention. The parameters for attention calculation are Query, Key, and Value. An attention function is described as the mapping of a query to a set of key-value pairs to output. It is a mechanism relating to different positions of a single sequence to compute a representation of the sequence. It is frequently used in various tasks such as text summarization, machine translation, and learning task-independent sentence representations. The calculation of self-attention is done in three steps. First is to take the query and each key then calculate the similarity between them to obtain the weight information. In the second step, use the softmax function to normalize these weights, and the final step is to conjugate these weights to achieve final attention weights. The input consists of queries and keys of dimension d_k and values of dimension of d_v. We compute the dot products of the query with all the keys, divided by $\sqrt{d_k}$. The equation shown in 3.4 represents the attention function, which takes a set of queries into a matrix Q. The keys and values are also packed into matrices K and

V, respectively.

$$Attention(Q, K, V) = Softmax\frac{QK^T}{\sqrt{d_k}}V \tag{3.3}$$

In the context of a self-attentive sequential music recommendation system, we need a user's listening sequence history $S_u = S_1^u, S_2^u, \ldots S_{|s_u|-1}^u$. The sequence here is the MIDI extracted data sequence of multidimensional time-series information as shown in Figure 3.3. In our approach, we fixed the sequence length to the median value of the timestamp, which is 2600. If the sequence length is below this median value, we pad 0 on the left until it reaches 2600. Item embedding matrix is created using song lookup matrix. The song look-up matrix is created from the MIDICSV conversion, and for each song, it stores Note on C information as discussed in section 3.2.2.

Apart from using the Note_on_C event information, we also used Pitch_bend, Control_on_C, and Note_off_C event to ensure that adding extra content information will lead to good performance, as emphasized in the section 3.3. As far as we know, extracting that much information from MIDI and using it in the content-based music recommendation system is not yet discussed. In our proposed model, the first layer is used as an embedding layer for item feature information. The next layer is used as a feed-forward neural network that relies on a large number of users with their interaction history. To make the model computationally efficient, we use users' latent factors to combine with the item information obtained from the self-attention blocks. In self-attention blocks, the input sequence is the output received from the recurrent layer.

Figure 3.5 shows the proposed model neural architecture, including complete input and output description of our model MSA-SRec. The proposed algorithm needs a user-song triplet dataset that contains users, songs, and their listening history. We transform this dataset into a user-song interaction matrix (U, S, r_{us}), where U represents the total number of users and S represents the total number of songs present in the dataset. The

value of rows and columns is the play count information, which we already normalized into rating (r_{us}) information. This matrix is used as an input for matrix factorization, which is used for extraction of the user's latent factors U_u and song latent factor S_s information. Figure 3.4 shows the structure of our proposed model. It takes input from the song lookup matrix, which is a multidimensional time series sequence data used with a one-hot vector of the song and the user to learn about the position. The multiplication of these matrices is fed as an input to the sequence model, which is a song array x_s. This song array is reshaped into a median timestamp value with the dimension of 2600×32. Instead of using a single recurrent layer, we stacked two recurrent layers in our model representation to make our model deeper. The additional hidden layers are interpreted to recombine the learned representation of weights from prior layers and create new illustrations at high levels of abstraction. Adding more layers means that it adds levels of abstraction to input observations over time.

In the next step, we process these hidden layer outputs to the self-attention blocks. The self-attention block takes n inputs and outputs n values. The self-attention mechanism allows input to interact with each other and obtain which values should be paid more attention than other values; after calculating input drive key query and value aggregates of these interaction values into attention scores. Finally, we apply softmax to gain softmaxed attention score for each input calculated using equation 3.4, and at the end, these scores for input get multiplied by their corresponding value.

$$a_i = Softmax(f(Q, K)) = \frac{exp(f(Q, K_i))}{\sum_i exp(f(Q, K_j))} \tag{3.4}$$

Our proposed model uses self-attention blocks to take the input from the last recurrent layer output and process these outputs to the multiple dense layers. After applying self-attention, we get the softmax attention score matrix for each output of the last hidden layer of the recurrent layer ($h_T^{<n,s>}$), where n is the last layer ($n = 2600$). Then, we multiply these with the user's latent vector U_u and one-hot encoding of the user and

pass it to multiple dense layers and train our model with the mean squared error. We fix the dense layer size with the latent factor size so that their multiplication generates a rating score for each user and song using equation 3.5.

$$\hat{r}_{us} = U_u.h_T^{<n,s>} \tag{3.5}$$

In this equation (3.5), U_u, is the user latent vector and h_T is the output of the last dense layer. The flowchart of our proposed approach is shown in the Figure 3.6. The MSA-SRec training procedure is illustrated in the algorithm 1.

Algorithm 1: Epoch-wise training algorithm for MSA-SRec

Data: Song-lookup matrix (M_s), user-rating matrix R_{us}, user latent factor U_u, One-hot vector of song S_s, batch_size b, sequence layer size = n, time steps = T

Result: Rating prediction $\hat{r}_{us}$

initialization;
for *each batch_size b in B training* **do**
  for (U, S, r_{us}) **do**
    $U_u \leftarrow One\_hot(u).U$; ▷ `Extract user latent factor`
    $S_s \leftarrow One\_hot(s).M_s$; ▷ `Extract song array as a sequence for input`
    $X_s \leftarrow Re\_Shape(S_s)$
    $SA(\hat{X}_s) \leftarrow Attention((\hat{X}_s.W^Q, \hat{X}_s.W^K, \hat{X}_s.W^V))$; ▷ `Self-Attention`
    $F_i(X_{si}) = ReLu(X_{si}.W^i + b_i)W^{i+1} + b^{i+1}$ $X_{s_{i+1}} = SA(F_i(X_{si}))$; ▷ `Stacking Self-Attention Block`
    $h_{T_{<n,s>}} = X_{s_{i+1}} \leftarrow FFN(X_s^i)$; ▷ `Feed-forward layer`
    $\hat{r}_{us} = h_T^{<n>,s}.U_u$; ▷ `rating prediction`
  $TrainingLoss(J) = \frac{1}{|b|} \sum_{u,s,r_{us} \epsilon b} (U_u.h_T^{<n,s>} - r_{us})^2$
  Backpropagation

3.2.5 Objective Function

In the proposed approach, we used latent factor vectors, which are real-valued, so the objective function is meant to reduce the mean squared error (MSE) of the rating predictions. The model is trained with MSE as a loss function, which compares the

actual rating r_{us} and predicted rating value $\hat{r}_{us}$.

$$MSE = \frac{1}{|R|} \sum_{u,s,r_{us} \epsilon R} (\hat{r}_{us} - r_{us})^2 \tag{3.6}$$

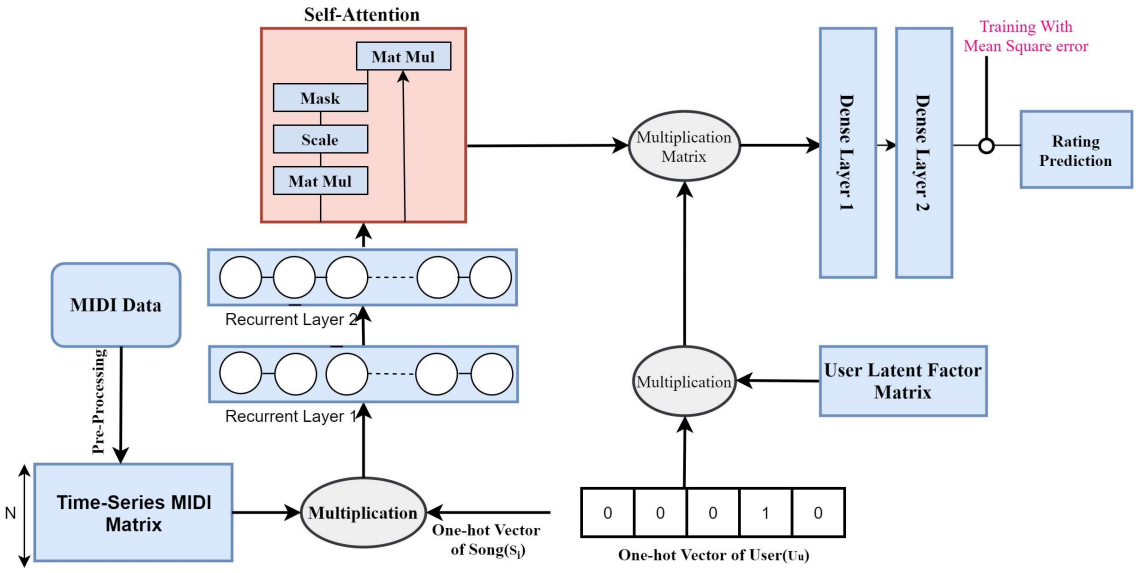

Figure 3.4: Structure of MIDI Based Self-Attentive Sequential Music Recommendation (MSA-SRec)

3.3 Analysis and Results

In this section, we describe an evaluation method aiming to estimate our model's recommendation performance and capabilities. We compare our model with the baseline approaches based on sequential models.

3.3.1 Dataset

A music recommender system is based on various user factors, but in a basic recommender system, whether it is for movies or for e-commerce, the recommendation is based on user and item interaction history. So considering these requirements, we utilized freely available datasets for our music recommendation system. First, we used The

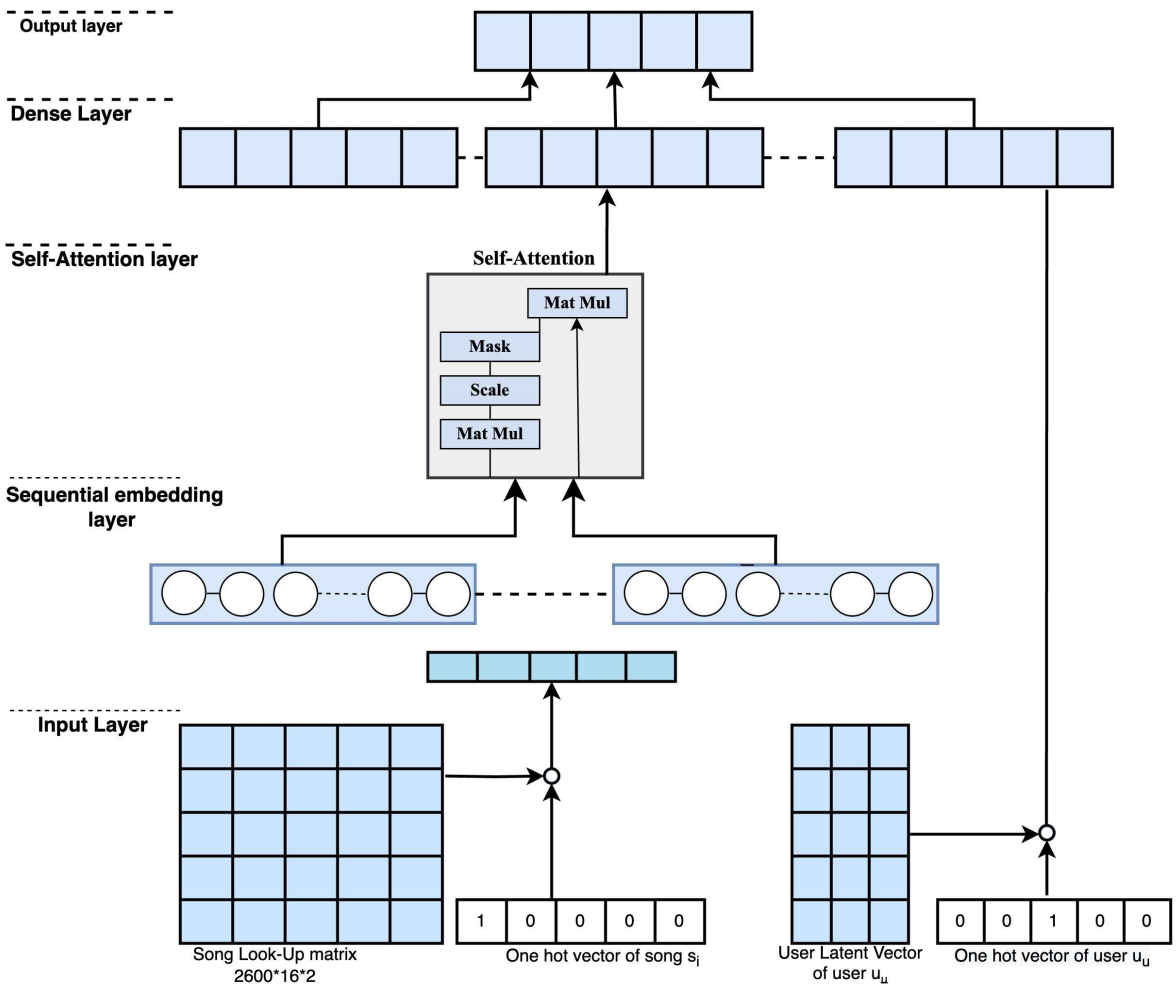

Figure 3.5: Neural Architecture of MIDI Based Self-Attentive Sequential Music Recommendation (MSA-SRec)

Echo Nest Taste Profile dataset, which is an official user dataset of the Million Song Dataset. This dataset is used as an interactive history of users and songs. Statistics of this dataset are described in Table 3.3.

In our proposed model, we extracted content from MIDI files of music for which we use the IMD_Matched Data set of The Lakh MIDI dataset. The IMD_matched dataset is a subset of The Lakh MIDI dataset, where the song_id of the MIDI file is coordinated with The Echo Nest dataset. We used the IMD-matched dataset having 45129 songs matched to entries in MSD Dataset. We use the dataset obtained from preprocessing discussed in section 3.2.1. The statistics of the preprocessed datasets are

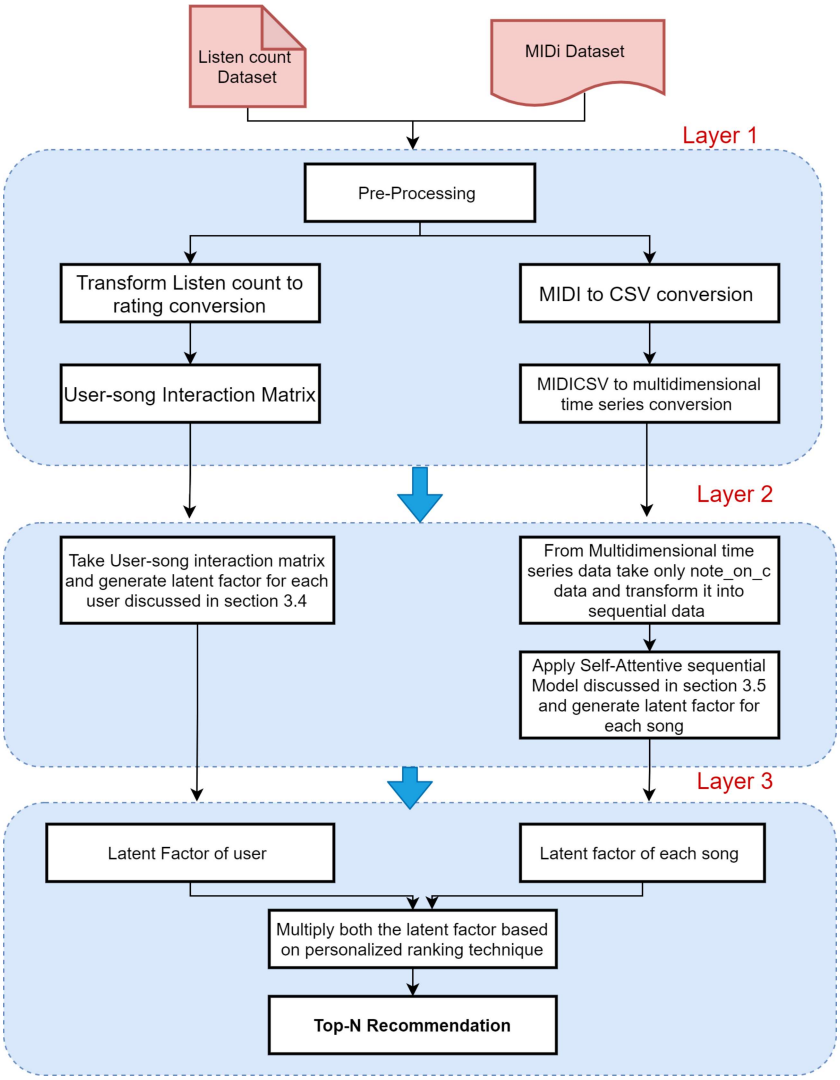

Figure 3.6: Architecture of our proposed model MSA-SRec)

shown in Table 3.3.

3.3.2 Experimental Settings

Our proposed model consists of two recurrent layers and two dense layers. The first recurrent layer has 150 neurons and returns the entire sequence of outputs for each sample (one vector per time step per sample), which is used as input for the second recurrent layer. The second recurrent layer has 50 (L=50) neurons and returns output corresponding to the last time-step, containing information about the entire input

Table 3.3: Statistics of the echo-nest taste profile Dataset

Data	**Users**	**Songs**	**Interactions**
Data Statistics	1019318	384546	48373586
Pre-processed Data	64258	10274	2043926

sequence. Finally, the last output layer (timestamp = 2600) is multiplied by the user latent vector. We choose latent factor dimensionality for the matrix factorization model as $k = 20, 50, 100$ and $l2$ regularization is used with the value of 0.001. After the second recurrent layer, we use a single self-attention block which learns item embedding from these. Now, this value is passed to a dense layer consisting of 5 neurons. The dense layer's output is fed to the last layer, i.e., a dense layer consisting of a single neuron predicting the rating a user would give to a particular song. The loss between the actual and predicted ratings is calculated using the mean-square-error loss function. Finally, the weights of the dense layer and all the recurrent layers are updated using back-propagation. We use a batch size of 256 for model training, and for testing, the batch size is 32. In this experiment, it is visible that the dataset is biased towards rating 1 after applying preprocessing of the dataset. So we use $l2$ regularization on the matrix factorization technique to obtain user and item latent factors. We trained our model for five epochs, used ADAM (Adaptive Moment) as an optimizer for our sequence model, and trained on 5076 unique users, which contains 6255 individual songs and interactions of 161959 user-song entries. We used 70% data for training and 30% for testing our model.

3.3.3 Recommendation Performance

For the architecture of our proposed MSA-SRec, we have used two self-attention blocks to learn the positional embedding. The learned item embedding (MIDI) and user embedding (latent vector) of the user are shared for the prediction layer. We have used TensorFlow for implementation with Adam optimizer with a learning rate = 0.001.

The maximum sequence length n is set to 2600 with various latent vector sizes. The performance of our model (MSA-SRec) is evaluated using various evaluation metrics defined in section 2.4. For our model's performance measure, we calculated precision, recall, and mean average precision for top-N recommendations, where N = 5, 10, 15, 20. We also calculated the error analysis in terms of the RMSE of our model with other sequential baseline models. Our proposed model (MSA-SRec) outperforms the baseline model with better accuracy. The change in accuracy with different k (latent factor) value is shown in the Table 3.5, 3.6, 3.7. To better understand the proposed model's result, we summarize the best model for the top-5 recommendations, shown in Table 3.4. Another two metrics, which are essential measures for the recommender system's assessment, are precision and recall, which show the item accuracy and relevance in the top-N recommendation, respectively. For our proposed model (MSA-SRec), in all variants of the sequence model, we got better results than the other baseline models. All the comparative results for various latent factors are shown in Table 3.5, 3.6 and 3.7. In terms of accuracy, we got the best result for SA-LSTM and SA-GRU with a slight difference, but in terms of MAP, precision and recall results for the SA-GRU are the best. We got almost 7% improvement in the precision and 6% improvement in MAP with the SA-GRU and the second-best performing model for top-5 recommendations is SA-LSTM. The comparative results for precision and MAP for top-n recommendations are shown using the line graph in Figure 3.7, 3.8 respectively. We further evaluated our model's performance for various hyperparameters we adopted in our model. We check our model's performance for different latent vector sizes, detailed results are discussed in the section 3.3.3.1 below. The best result we obtain is for latent factor 100 and mostly for the self-attentive GRU model shown in Table 3.4. We also tried to use the item latent factor with this proposed model, which makes this computation a bit complex, and it also increases the space and time complexity, and there is not much difference in the results.

3.3.3.1 Influence of vector dimension

We experimented with various latent vector sizes to test our model for the optimal value of the latent vector. We conducted an experiment where we tested for k values where we set (k = 20,50,100, 150, 200). All the results obtained from various latent dimensions are reported in Table 3.5, 3.6, 3.7. We used precision, recall, and MAP for top-n recommendations performance measures. From Table 3.7 we concluded that results are best for latent factor value k= 100 and for model SA-GRU. We also tested our model with latent dimensions 150 and 200, where results are improving with slight differences, but the computational time and space are more for our model.

Table 3.4: Hyperparameter Tuning Results: MAP@5, Precision@5, Recall@5 on the test data after 5 epochs. Best results (in bold) were obtained, first, with 100 latent features, and the best models for MAP, precision, and recall are SA-GRU, SA-RNN, and SA-GRU, respectively

Hyperparameter	**Value**	**MAP@5**	**Precision@5**	**Recall@5**
# of latent factors	20	0.1322	0.1998	0.1304
	50	0.1749	0.1509	0.1893
	100	**0.2036**	**0.3439**	**0.2303**
Sequence Model Type	SA-RNN	0.1520	0.2598	0.1545
	SA-LSTM	0.1890	0.3327	0.3327
	SA-GRU	**0.2036**	**0.3439**	**0.2303**

3.3.4 Ablation Study

Since there are many components in our method, we analyze their respective impact via a rough ablation study. Table 3.5,3.6,3.7 shows our default method's performance and its variants on dataset (with latent dimension = 20, 50, 100). We introduce the variants and analyze their effect, respectively:

- **Replacing content information**: In MSA-SRec, we replace content information from MIDI with lyrics information. Lyrics of the same track_id are extracted from the MusicXmatch dataset of the million song database. Lyrics information for the tracks is available in a bag of word format to avoid copyright. We use

Table 3.5: Various Evaluation Measure Metrics for Latent Factor 20

	RNN	LSTM	GRU	SA-RNN	SA-LSTM	SA-GRU
RMSE	1.3695	1.3569	1.3557	1.3485	1.3332	1.3414
Precision@5	0.1261	0.1230	0.1787	0.1998	0.2229	0.2174
Recall@5	0.0605	0.0596	0.1779	0.1105	0.1263	0.1304
MAP@5	0.0686	0.0677	0.1778	0.1185	0.1267	0.1322
Precision@10	0.1257	0.1228	0.1778	0.1983	0.2213	0.2161
Recall@10	0.0693	0.0702	0.0897	0.1206	0.1367	0.1421
MAP@10	0.0532	0.0544	0.0977	0.0843	0.0894	0.0944
Precision@15	0.1257	0.1226	0.1003	0.1982	0.2212	0.2157
Recall@15	0.0718	0.0737	0.1014	0.1235	0.1399	0.1448
MAP@15	0.0483	0.0503	0.0921	0.0757	0.0809	0.0851
Precision@20	0.1256	0.1225	0.0658	0.1981	0.2212	0.2157
Recall@20	0.0727	0.0752	0.0595	0.1244	0.1411	0.1459
MAP@20	0.0465	0.0491	0.0577	0.0727	0.0781	0.0821

Table 3.6: Various Evaluation Measure Metrics for Latent Factor 50

	RNN	LSTM	GRU	SA-RNN	SA-LSTM	SA-GRU
RMSE	1.3517	1.3493	1.3424	1.3493	1.3142	1.3239
Precision@5	0.2002	0.1509	0.2191	0.1509	0.3020	0.3038
Recall@5	0.1024	0.0732	0.1124	0.0732	0.1856	0.1893
MAP@5	0.1083	0.0831	0.1111	0.0831	0.1655	0.1749
Precision@10	0.1996	0.1498	0.2182	0.1498	0.2999	0.3014
Recall@10	0.1111	0.0836	0.1219	0.0836	0.1982	0.2031
MAP@10	0.0764	0.0633	0.0781	0.0633	0.1166	0.1237
Precision@15	0.1995	0.1496	0.2181	0.1496	0.2995	0.3012
Recall@15	0.1127	0.0871	0.1246	0.0871	0.2007	0.2059
MAP@15	0.0681	0.0586	0.0708	0.0586	0.1047	0.1110
Precision@20	0.1995	0.1496	0.218	0.1496	0.2995	0.3011
Recall@20	0.1132	0.0883	0.1253	0.0883	0.2015	0.2066
MAP@20	0.0658	0.0574	0.0686	0.0574	0.1014	0.1075

Table 3.7: Various Evaluation Measure Metrics for Latent Factor 100

	RNN	**LSTM**	**GRU**	**SA-RNN**	**SA-LSTM**	**SA-GRU**
RMSE	1.3659	1.3370	1.3478	1.3441	1.3207	**1.3186**
Precision@5	0.1901	0.2333	0.2668	0.2598	0.3327	**0.3439**
Recall@5	0.0998	0.1292	0.1467	0.1545	0.2158	**0.2303**
MAP@5	0.1072	0.1256	0.1358	0.1520	0.1890	**0.2036**
Precision@10	0.1888	0.2325	0.2666	0.2588	0.3302	**0.3407**
Recall@10	0.1099	0.1449	0.1575	0.1656	0.2281	**0.2429**
MAP@10	0.0777	0.0965	0.0963	0.1054	0.1302	**0.2088**
Precision@15	0.1886	0.2321	0.2665	0.2586	0.3310	**0.3405**
Recall@15	0.1122	0.1490	0.1602	0.1678	0.2306	**0.2451**
MAP@15	0.0706	0.0883	0.0872	0.0947	0.1179	**0.2254**
Precision@20	0.1887	0.2322	0.2666	0.2585	0.3299	**0.3405**
Recall@20	0.1129	0.1504	0.1611	0.1684	0.2311	**0.2458**
MAP@20	0.0687	0.0865	0.0851	0.0917	0.1145	**0.2221**

Table 3.8: Evaluation of baseline algorithms on the echo nest music dataset, lower value for RMSE, and higher value for precision. recall and MAP are better.

	MF [129]	**NeuMF [130]**	**BPR+MF [131]**	**ItemPop [132]**	**MSA-SRec**
RMSE	1.5523	1.3406	1.3298	1.3354	**1.3186**
Precision@5	0.0363	0.1452	0.1682	0.1357	**0.3439**
Recall@5	0.0171	0.0924	0.1247	0.0426	**0.2303**
MAP@5	0.1011	0.1364	0.1429	0.0657	**0.2036**

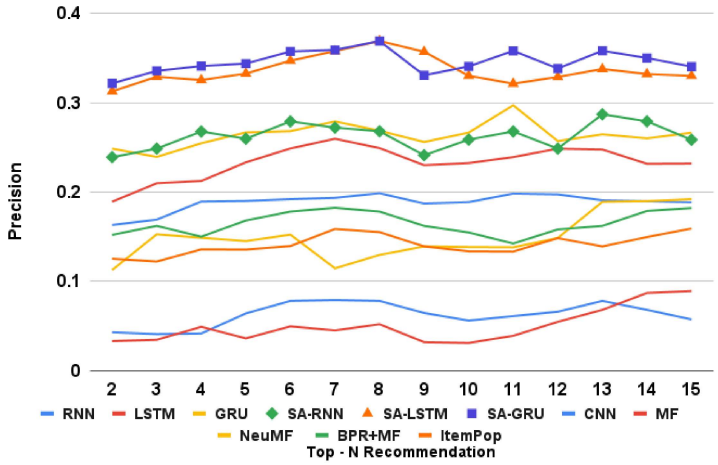

Figure 3.7: MSA-SRec result in precision@k

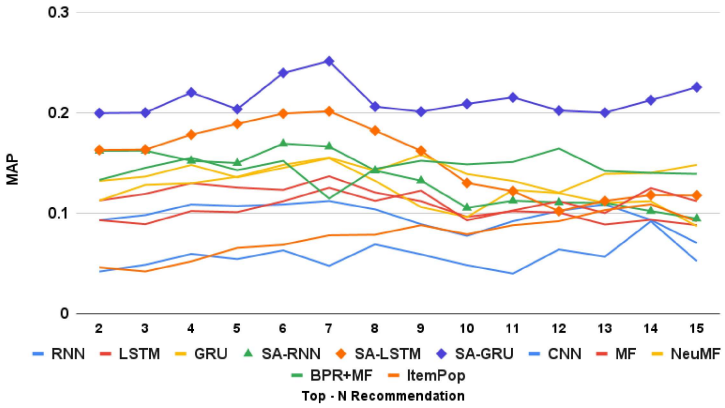

Figure 3.8: MSA-SRec result in MAP@k

the Word2Vec technique for embedding generation and then generate recommendations using our proposed model. The brief results are presented in Table 3.9, which clearly shows that our model performs better with MIDI.

- **Adding more content information**: In MSA-SRec, we add more content information with more event information extracted from MIDI, ex. Note_off_c, pitch_bend and control_on_c event information, and then we use our MSA-SRec algorithm, results are shown in Table 3.10. Adding more content information in the model leads to improvement of approximately 0.1% and 1.6% in precision.
- **Removing content information**: To test a model without any content infor-

mation, we choose the neural collaborative filtering (NeuMF) algorithm which only considers the listening history of the user as input and generates a recommendation based on the users' past interaction history. Comparative results for various baseline algorithms are shown in Table 3.11.

- **MSA-SRec performance for Transformer Model (T-MSA-SRec)**: Self-attention and the Transformer model are closely related because the self-attention model is a crucial component in transformer architecture. The Transformer model is a neural network architecture that uses self-attention as a fundamental building block [133]. The transformer model includes encoder-decoder architecture. Each multi-layer component of the encoder and decoder architecture includes self-attention blocks and feed-forward neural network sub-layers. We apply transformer architecture in our proposed model MSA-SRec to capture the long dependencies and contextual relationships between midi sequences. Adding a transformer to our model (T-MSA-SRec) increases the performance. The experimental results are shown in Table 3.12.

Table 3.9: Evaluation of MSA-SRec model on using Lyrics information as content Information

	RNN	**LSTM**	**GRU**	**SA-RNN**	**SA-LSTM**	**SA-GRU**
RMSE	1.3623	1.4090	1.3190	1.3450	1.3210	**1.3006**
Precision@5	0.0924	0.0550	0.1121	0.1598	0.1027	**0.1249**
Recall@5	0.0910	0.0595	0.1104	0.1221	0.1051	**0.1254**
MAP@5	0.0826	0.0545	0.1001	0.0951	0.0982	**0.1126**

Table 3.10: Evaluation of MSA-SRec on adding extra content Information (Note_off_c, Pitch_bend, Control_on_C)

	RNN	**LSTM**	**GRU**	**SA-RNN**	**SA-LSTM**	**SA-GRU**
RMSE	1.3241	1.3201	1.3190	1.3105	**1.3009**	1.3092
Precision@5	0.2103	0.2301	0.2898	0.2706	0.3542	**0.3599**
Recall@5	0.0901	0.1202	0.1559	0.1657	**0.2220**	0.2103
MAP@5	0.2201	0.1108	0.1549	0.2482	0.3201	**0.3441**

Table 3.11: Evaluation of baseline algorithms on the echo nest music dataset, lower value for RMSE, and higher value for precision. recall and MAP are better.

	MF [129]	**NeuMF [130]**	**BPR+MF [131]**	**ItemPop [132]**	**MSA-SRec**
RMSE	1.5523	1.3406	1.3298	1.3354	**1.3186**
Precision@5	0.0363	0.1452	0.1682	0.1357	**0.3439**
Recall@5	0.0171	0.0924	0.1247	0.0426	**0.2303**
MAP@5	0.1011	0.1364	0.1429	0.0657	**0.2036**

Table 3.12: Evaluation of MSA-SRec using Transformer model)

	RNN	**LSTM**	**GRU**	**SA-RNN**	**SA-LSTM**	**SA-GRU**	**T-MSA-SRec**
RMSE	1.3659	1.3370	1.3478	1.3441	1.3207	1.3186	**1.2632**
Precision@5	0.1901	0.2333	0.2668	0.2598	0.3327	0.3439	**0.3667**
Recall@5	0.0998	0.1292	0.1467	0.1545	0.2158	0.2303	**0.3001**
MAP@5	0.1072	0.1256	0.1358	0.152	0.189	0.2036	**0.2556**
Precision@10	0.1888	0.2325	0.2666	0.2588	0.3302	0.3407	**0.3991**
Recall@10	0.1099	0.1449	0.1575	0.1656	0.2281	0.2429	**0.2877**
MAP@10	0.0777	0.0965	0.0963	0.1054	0.1302	0.2088	**0.2553**
Precision@15	0.1886	0.2321	0.2665	0.2586	0.3310	0.3405	**0.4112**
Recall@15	0.1122	0.1490	0.1602	0.1678	0.2306	0.2451	**0.2988**
MAP@15	0.0706	0.0883	0.0872	0.0947	0.1179	0.2254	**0.2445**
Precision@20	0.1887	0.2322	0.2666	0.2585	0.3299	0.3405	**0.4112**
Recall@20	0.1129	0.1504	0.1611	0.1684	0.2311	0.2458	**0.3221**
MAP@20	0.0687	0.0865	0.0851	0.0917	0.1145	0.2221	**0.3109**

3.3.5 Statistical Significance Test

We perform a statistical significance test for performance comparison of our proposed approach MSA-SRec and the second-best performing approach. A statistical significance test is based on a few simple ideas: hypothesis testing, normal distribution, and p-values. We performed a significance test for analysis of results obtained from MSA-SRec to check whether they are significant or not. We did a paired t-test as a piece of statistical evidence, which shows that the mean difference between different paired results on a distinct outcome is significantly different from zero. The parametric t-test is defined as:

$$t = \frac{m}{\frac{s}{\sqrt{n}}} \tag{3.7}$$

where m is the mean, s is the standard deviation of the difference between all pairs and n is the total number of samples we have taken for testing. We test our hypothesis using a one-sided t-test where our hypothesis says that the difference between the mean of two different samples is greater than zero. For this test, we considered precision@k and MAP@k values shown in Table 3.13, 3.14, respectively, where values in bold represent the best and values with superscript with a plus sign indicate the second-highest value. We also use the actual and predicted rating mean for the MSA-SRec for the same hypothesis to check the stability and prove the significance of the proposed MSA-SRec against other state-of-the-art techniques. The statistic must follow the t-distribution with n-1 degrees of freedom for all samples in order to demonstrate the null hypothesis. The p-value should be less than the x (x = 0.01, 0.05). We reject the null hypothesis for precision@k value paired samples with the first group $mean = 0.256720$ and $SD(standard\ deviation) = 0.017221$ and for the second group $mean = 0.34080$ and $SD = 0.001972$. We reject the null hypothesis with a 95% confidence interval of this difference from -0.107333 to -0.060827.

Table 3.13: Precision@k of MSA-SRec and other baselines methods

Methods	P@5	P@10	P@15	P@12	P@20
RNN	0.1901	0.1888	0.1886	0.1974	0.1887
LSTM	0.2333	0.2325	0.232	0.2487	0.2322
GRU	0.2668^{+}	0.2666^{+}	0.2665^{+}	0.2571^{+}	0.2666^{+}
CNN	0.0642	0.0562	0.0575	0.0661	0.0594
MF	0.0363	0.0312	0.0893	0.0548	0.0915
NeuMF	0.1452	0.1385	0.1921	0.1487	0.2017
BPR+MF	0.1682	0.1548	0.1821	0.1582	0.1933
ItemPop	0.1357	0.1338	0.1592	0.1485	0.1854
MSA-SRec	**0.3439**	**0.3407**	**0.3405**	**0.3384**	**0.3405**

Best results are highlighted in bold, and second best are in (+)

Table 3.14: MAP@k of MSA-SRec and other baselines methods

Methods	MAP@5	MAP@10	MAP@15	MAP@12	MAP@20
RNN	0.1072	0.0777	0.0706	0.1023	0.1887
LSTM	0.1256	0.0965	0.0883	0.1007	0.2320
GRU	0.1358	0.0963	0.0872	0.1201	0.2666^{+}
CNN	0.0545	0.0484	0.0525	0.0641	0.0594
MF	0.1011	0.0931	0.1121	0.1129	0.1284
NeuMF	0.1364	0.1391	0.1482^{+}	0.1205	0.1658
BPR+MF	0.1429^{+}	0.1487^{+}	0.1392	0.1644^{+}	0.1548
ItemPop	0.0657	0.0792	0.0921	0.0922	0.1092
MSA-SRec	**0.2036**	**0.2088**	**0.2254**	**0.2022**	**0.3405**

Best results are highlighted in bold, and the second best is in (+)

3.4 Summary

The primary objective of the proposed MSA-SRec model is to provide recommendations for the cold-start user and items. We adopted a sequential model for recommendation generation, which will include users' dynamic interests in their model. This model is a deep hybrid approach that uses content information of music extracted from MIDI files, along with user listening history. The inclusion of content data in music recommendations will improve our model's performance, and it also restricts the use of users' personalized information. As a part of our future work, we plan to extend the model by using more content, such as from the MIDI data itself, rather than relying on personalization.