Optimizing Music Playlists in Real-Time

Hadi Askari
Department of Computer Science
University of California, Davis
Davis, CA 95616
haskari@ucdavis.edu

Muhammad Haroon
Department of Computer Science
University of California, Davis
Davis, CA 95616
mharoon@ucdavis.edu

1 Introduction

The surgence of online shorts platforms like TikTok, Instagram's Reels, and YouTube's shorts demonstrate the effectiveness of algorithmically curated content consumption. In contrast to traditional recommendation systems that recommend several items for the users to choose from, these short platforms make the choice for the user themselves by recommending only one item at a time. The user then simply has to either like or dislike the content, a sentiment that can be captured either *implicitly* (user skips, partially views, completely views the content, etc.) or *explicitly* (user likes/dislikes, clicks do not recommend, etc.) As a result, these platforms have seen a marked increase in user engagement levels as compared to other platforms. For example, in just about 5 years, TikTok has amassed about a billion users online [1], was the most downloaded application of 2022 [2], and that the average TikTok user spends approximately 1.5 hours daily on the app [2].

The sophisticated algorithm used by these platforms is likely what leads to their addicting nature. Relying on immediate implicit feedback from users, the platform's algorithm learns the user's preferences in real-time. Our goal for this project is to implement a similar system albeit at a much smaller scale and understand how reinforcement learning can be used to that end. We consider the problem of music playlist recommendations similar to how popular apps such as Spotify and Apple Music implement stations and radios. For simplicity and in accordance with the readily available dataset, we also consider a simplified version where the platform has already decided the next track and our system merely decides whether to play it or skip it.

2 Related Work

Music recommendation is a multi-billion dollar business. Companies like Apple, Amazon, Google, Spotify, Soundcloud spend an exorbitant amount of money trying to perfect their algorithms. These companies try to model the users on their platform. Researchers have explored the connection between personality and individuals' musical preferences. [3, 4, 5]. Another research avenue, which can be seen as both distinct and complementary, focuses on comprehending and modeling users' interactions with the underlying platform. This particular problem related to online streaming services has received limited attention in the research community despite its longstanding nature [6]. One example of such interactions is the act of skipping between songs. Proper modeling and comprehension of these skips during music listening sessions are pivotal in gaining insight into users' behavior. Since skips often represent the only available information for the underlying Music Recommendation System (MRS), they are frequently utilized as a proxy for inferring music preferences [7, 8].

Previous studies have already employed the skipping signal in various ways, such as incorporating it as a metric in heuristic-based playlist generation systems, evaluating user satisfaction and relevance, or utilizing it as a counterfactual estimator [9, 10, 11, 12]. Skipping a track does not always indicate a negative preference. There are multiple hypotheses that can be proposed to explain why users skip songs. Recent research indicates that individuals exhibit a universal behavior when it comes to skipping songs, which can be influenced by factors such as time, geography, and reactions to audio fluctuations or musical events [13, 14, 15].

In a recent study by Meggetto et al. [7], a clustering-based method was introduced to effectively categorize users into four types based on their skipping activity during music listening sessions. These types include the listener, listen-then-skip, skip-then-listen, and skipper, and their classification is influenced by factors such as the duration of the listening session, time of day, and the type of playlist being used. However, a key limitation of these previous works is that they primarily investigate the relationship between the listening context, content, and the skipping behavior. The previous studies do not delve into the impact of user interactions with the platform on skip detection. This limitation is specifically addressed in this work. This work primarily deals with Spotify's Skip prediction challenge and its' accompanying dataset. A majority of the submissions to the challenge were Deep Learning based models using RNNs or Supervised Learning based models that used traditional Machine Learning methods such as Random Forests/XGBoost [16, 17, 18, 19].

The utilization of reinforcement learning (RL) can support classifiers in acquiring beneficial features [20, 21], as well as selecting high-quality instances from noisy data [22]. Wiering et al. [23] provide evidence that RL is indeed well-suited for classification tasks. Specifically, the studies conducted by the authors in [21, 24] demonstrate that a DQN [25] approach outperforms and exhibits greater resilience compared to state-of-the-art algorithms. In this study, we undertake a novel investigation into the viability of Deep Reinforcement Learning (DRL) for the sequential prediction of users' music skipping behavior. Our motivation stems from the shortcomings of existing methods and the inherent benefits of DRL. By conducting a thorough analysis of users' historical data, we examine the effectiveness and impact of DRL in our approach for this task. This research represents the initial stride towards comprehending the underlying reasons behind music skipping among individuals.

3 Methodology

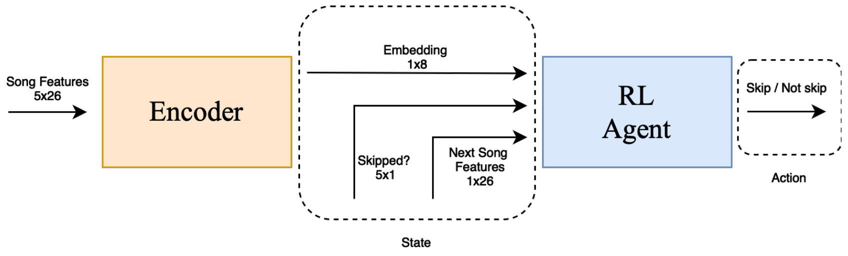


Figure 1: Overview of the pipeline used for training the RL agent.

Figure 1 provides an overview of the pipeline that we built for training the reinforcement learning model for predicting user skip preferences. In the rest of this section, we provide a description of the dataset used and additional details on what the encoder and RL agent in the figure represent.

3.1 Dataset

The dataset in use was the real-world Music Streaming Sessions Dataset (MSSD) released by Spotify [6]. The training set used in this study, which is publicly available, comprises around 150 million logged streaming sessions. These sessions were collected over a span of 66 days, specifically from July 15th to September 18th, 2018. Each day consists of ten logs, with each log containing streaming listening sessions that are uniformly sampled at random throughout the entire day. The sessions themselves are sequences of songs or tracks that users have listened to, with each session containing anywhere from 10 to a maximum of 20 records (one record per song). Each record encompasses diverse contextual information about the user's streaming experience, including details such as the playlist type and the individual's interaction history with the platform. While the specific track titles are unavailable, descriptive audio features and metadata are provided for each track. These include attributes such as acousticness, valence, and year of release. Figures 1 and 2 depict the dataset format and the song features available.

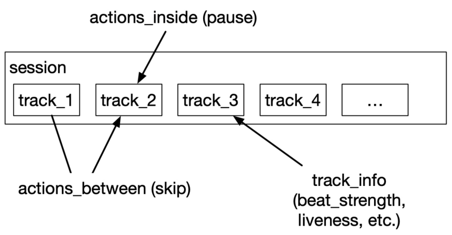
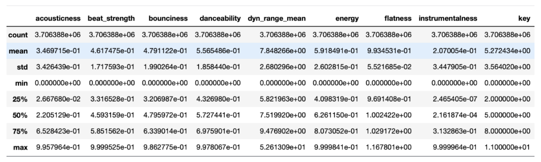


Figure 2: Dataset Format



	acousticness	beat_strength	bounciness	danceability	dyn_range_mean	energy	flatness	instrumentalness	key
count	3.706388e+06	3.706388e+06	3.706388e+06	3.706388e+06	3.706388e+06	3.706388e+06	3.706388e+06	3.706388e+06	3.706388e+06
mean	3.469715e-01	4.617475e-01	4.791122e-01	5.565486e-01	7.848266e+00	5.918491e-01	9.934531e-01	2.070054e-01	5.272434e+00
std	3.426439e-01	1.717593e-01	1.990264e-01	1.858440e-01	2.680296e+00	2.602815e-01	5.521685e-02	3.447905e-01	3.564020e+00
min	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
25%	2.667680e-02	3.316528e-01	3.206987e-01	4.326980e-01	5.821963e+00	4.098319e-01	9.691408e-01	2.465405e-07	2.000000e+00
50%	2.205129e-01	4.593159e-01	4.795972e-01	5.727441e-01	7.519920e+00	6.261150e-01	1.002422e+00	2.161874e-04	5.000000e+00
75%	6.528423e-01	5.851562e-01	6.339014e-01	6.975901e-01	9.476902e+00	8.073052e-01	1.029172e+00	3.132863e-01	8.000000e+00
max	9.957964e-01	9.999525e-01	9.862775e-01	9.978067e-01	5.261309e+01	9.999841e-01	1.167801e+00	9.999964e-01	1.100000e+01

Figure 3: Song Features

We used the training set of the dataset to create our own custon training and test sets. Logs from 4 different days were chosen as our training set and 1 day was kept as our test set to keep generalizability and to remove any temporal connections. All the features were kept in the state representation and the categorical features were one-hot encoded. All the audio features are standardised to have a distribution with a mean value of 0 and a standard deviation of 1.

3.2 Encoder

To trace the user's running music preferences over time, we design an embedding space that maintains the user's listening preferences over time. Specifically, we are trying to keep track of the songs recommended to the user and what action was performed by the user in response to that recommendation. Because our end goal is to maintain a long-running, variable-length sequence of user preferences, we seek to devise a way in which we can encode this variable-length sequence into a fixed-length representation that has high fidelity. Encoders are usually trained as part of a joint, end-to-end encoder-decoder architecture colloquially known as an autoencoder. The encoder encodes the sequential data into a fixed-length "latent" embedding, and the decoder tries to decode this embedding back to the original sequence. Gradient descent can then be performed on the loss function between the original and the decoded sequence to train the model in an end-to-end fashion. Once done, we can discard the decoder and only keep the encoder to generate the latent embedding for any given sequence.

To that end, we played around with two well-known approaches that have been used in prior work on sequential data.

LSTM Autoencoders. Long short-term memory units have been used for a long time in neural networks to capture patterns over long-running, sequential data. They are known for their ability to *forget* unneeded information and hold onto information that may be pertinent to the task as the sequence moves forward. Thus, we speculate that such a model will be able to track what track features are more important and predictive of the users' preferences.

Transformers. Transformers are a relatively newer architecture for dealing with sequential data. Instead of recurrent units, transformers use the notion of *self-attention* to identify parts of the sequence that are more predictive. Through gradient descent, the attention units inside the transformer architecture can identify the more important parts of the feature space. Transformers have demonstrated unmatched performance gains in sequential tasks in NLP and vision tasks over LSTM encoder units and we expect them to outperform the LSTM autoencoder.

3.3 Reinforcement Learning Agent

Reinforcement learning is a reward-based machine learning approach where desired behavior by the model is rewarded and undesired behavior is punished, and the model learns a policy that maximizes reward. We now define our RL agent.

3.3.1 State Space

As can be seen in Figure 1, the state space for the RL agent consists of three components.

- **Latent embedding:** The latent embedding from the encoder represents the features of the user's prior listening history.
- **Skipped:** This vector provides the user's preferences for the tracks inside the latent embedding. We keep this separate from the latent embedding so as to keep the RL agent aware of it as a separate entity
- **Next track features:** These are the features of the next track for which the RL agent has to make a decision about.

Since our state is comprised of a combination of latent embedding and track features, it is **continuous** in nature.

3.3.2 Action Space

The action space for the RL agent is binary, i.e., to skip or not to skip. Thus, it is **discrete** in nature.

3.3.3 Reward

The RL agent is rewarded +1 for every correct prediction that it makes. Since we have the ground-truth (i.e., what action did the user actually perform), we can compare the agent's prediction directly and reward the behavior accordingly.

3.3.4 Policy

The RL agent has Q-values for the two actions for which it needs to learn the optimal values.

3.3.5 Structure of RL Agent

We implement the RL agent as a deep-learning DQN and DDQN model.

DQN Model:

This was developed to help in cases where the state space was too large for tabular methods. It combines the Q learning algorithm with Deep Learning by using Neural Networks as function approximators for the Q table. Since we have million of possible songs and a user could have listened to them in a number of different combinations, we are left with an exorbitant amount of states which implies a need for a model such as DQN. In a DQN, current states and actions are estimated by the main neural network and the next states and actions are estimated by the target neural network. They overcome instability via Experience Replay, having a Target Network, using Clipping Rewards and using Skipping Frames.

In our implementation of DQN, The experience replay memory was 10000, the batch size and frequency of updates were 256, the learning rate was set to 0.001, and the discount factor was 0.9. The policy network consists of three fully-connected layers (of size 128) and a final action-value linear output layer of size 2. This final layer computes the Q-values for each action.

Dueling DQN Model:

An issue encountered with the DQN algorithm is the tendency to overestimate the actual rewards. The Q-values within the algorithm can lead the agent to anticipate a higher return than what it will actually achieve in reality. This led to researchers developing improvements on these algorithms such as the Double DQN [26] and Dueling DQN [27]. This algorithm splits the Q-values in two different parts, the value function V(s) and the advantage function A(s, a). The value function V(s) tells us how much reward we will collect from state s. And the advantage function A(s, a) tells us how much better one action is compared to the other actions. The same neural network splits its last layer in two parts, to estimate the state value function s (V(s)) and to estimate the advantage function for each action a (A(s, a)), combining at the end for the Q values.

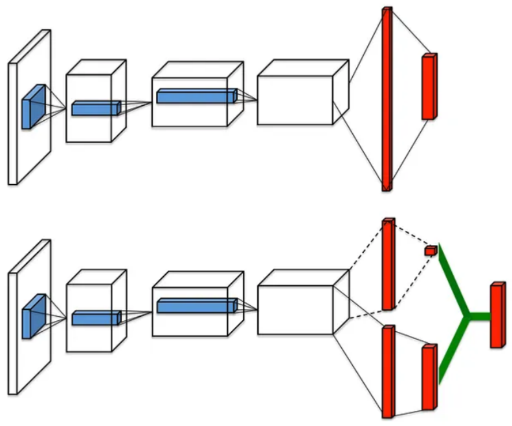


Figure 4: Architecture of DQN comapred to the architecture of Dueling DQN

We used the implementation of Dueling DQN provided by the Tensorforce library [28]. Other parameters/hyperparameters were kept consistent with the DQN implementation.

3.4 Evaluation Metrics

Every second half of a session is the test set is used for prediction. Odd numbered sessions are rounded-up. In the dataset schema, prediction is based on the skip_2 feature. It indicates a threshold on whether the user played the track only briefly (no precise threshold is provided) before skipping to the next song in their session. Which we count as a skip. Mean Average Accuracy and First Prediction Accuracy were the metrics we used.

Mean Average Accuracy:

$$MAA = \frac{\sum_{i=1}^{T} A(i)L(i)}{T}$$

In the given context, T represents the number of tracks to be predicted within the session. A(i) denotes the accuracy of the predictions up to position i in the sequence, while L(i) indicates whether the i-th prediction is correct or not.

First Prediction Accuracy:

This is the accuracy at predicting the first interaction for the second half of each session

4 Results

4.1 Encoder

Our initial attempt at a variable-length encoder input did not achieve good results. The error did not decrease at all so instead, owing to time constraints with the project, we decided to use a sliding window approach where we only considered the last n tracks in the user's history. We arbitrarily set $n = 5$ though a more systematic approach of an increasing n until the loss is undesirable could've been employed to find the optimal value. Figure 5 shows the mean square error loss for the original and the decoded sequence over several epochs for the autoencoder where $n = 5$. We see that the error drops < 1 after training for a few epochs for normalized values. Eye-balling the original and decoded sequences, we could tell that the features were approximately close enough to the original values even without any normalization. Thus, we believe that the encoded embeddings are an accurate enough approximation of the original sequence.

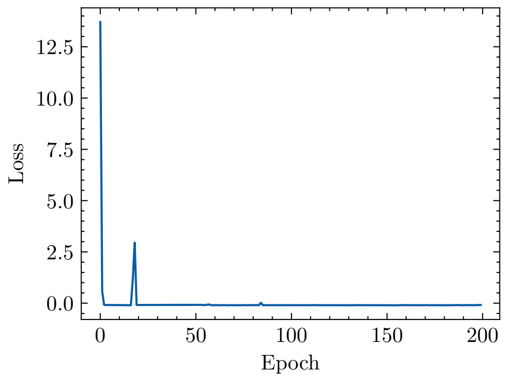


Figure 5: MSE Loss between the original and the decoded sequence used for training the autoencoder.

4.2 Reinforcement Learning

We trained our models for 100,000 episodes over 5 iterations on 2 x NVIDIA RTX A6000 - 48GB GDDR6 GPU's. The training time took approximately 14 hours. The testing time was 20 minutes.

Model	**Mean Average Accuracy**	**First Prediction Accuracy**
DQN	0.818	0.879
Dueling DQN	0.821	0.881

Table 1: Evaluation Metrics on our different Models

5 Conclusion

5.1 Limitations

One limitation of our study is that we are using a sliding window approach to our users' song history. We select the last 5 songs played and calculate the watch history state embedding from that. Now while the recent playing history is the most indicative of whether the user would play or skip the next song, it doesn't provide the entire picture. Another limitation is that the dataset is highly skewed towards skipping behaviours (66%) which is not indicative of realistic human behaviour. Most people search for their songs that they want to listen to.

5.2 Future Improvements

A major improvement that we can perform is that we can train our model end-to-end rather than training the encoder block separately and the RL module separately. This would help the RL agent learn better state representations and make better decisions on what the user actually interested in. Another improvement that we tried was to have a transformer based encoding block. We eventually settled on LSTM based autoencoders due to time constraints but believe that theoretically, transformer based models will provide us with more accurate encodings. We can also address the limitation of using the sliding window approach. We can replace the last 5 videos with the entire watch history of the user. We could also try to experiment with different lengths of watch histories to see which one would best model the skipping behaviour of the user. Finally, we could also create a human feedback element similar to RLHF [29] where we can have users dynamically rate whether they would have skipped the songs the model predicted them to skip to create a better performing model.

References

[1] Sabri Ben-Achour and Erika Soderstrom. Social media platforms are adapting to compete with TikTok. Are their efforts working? `https://www.marketplace.org/2022/09/26/social-media-platforms-are-adapting-to-compete-with-tiktok-are-their-efforts-working/`, 2022.

[2] Demand Sage. 36 TikTok Statistics 2023: How Many Users Are There! `https://www.demandsage.com/tiktok-user-statistics/`, 2023.

[3] Alexandra Langmeyer, Angelika Guglhör-Rudan, and Christian Tarnai. What do music preferences reveal about personality? *Journal of individual differences*, 2012.

[4] Peter J Rentfrow and Samuel D Gosling. The do re mi's of everyday life: the structure and personality correlates of music preferences. *Journal of personality and social psychology*, 84(6):1236, 2003.

[5] Peter J Rentfrow and Samuel D Gosling. Message in a ballad: The role of music preferences in interpersonal perception. *Psychological science*, 17(3):236–242, 2006.

[6] Brian Brost, Rishabh Mehrotra, and Tristan Jehan. The music streaming sessions dataset. In *The World Wide Web Conference*, pages 2594–2600, 2019.

[7] Francesco Meggetto, Crawford Revie, John Levine, and Yashar Moshfeghi. On skipping behaviour types in music streaming sessions. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pages 3333–3337, 2021.

[8] Markus Schedl, Peter Knees, Brian McFee, and Dmitry Bogdanov. Music recommendation systems: Techniques, use cases, and challenges. In *Recommender Systems Handbook*, pages 927–971. Springer, 2021.

[9] Klaas Bosteels, Elias Pampalk, and Etienne E Kerre. Evaluating and analysing dynamic playlist generation heuristics using radio logs and fuzzy set theory. In *ISMIR*, volume 9, pages 351–356, 2009.

[10] Elias Pampalk, Tim Pohle, and Gerhard Widmer. Dynamic playlist generation based on skipping behavior. In *ISMIR*, volume 5, pages 634–637, 2005.

[11] Casper Hansen, Christian Hansen, Lucas Maystre, Rishabh Mehrotra, Brian Brost, Federico Tomasi, and Mounia Lalmas. Contextual and sequential user embeddings for large-scale music recommendation. In *Proceedings of the 14th ACM Conference on Recommender Systems*, pages 53–62, 2020.

[12] Hongyi Wen, Longqi Yang, and Deborah Estrin. Leveraging post-click feedback for content recommendations. In *Proceedings of the 13th ACM Conference on Recommender Systems*, pages 278–286, 2019.

[13] Jonathan Donier. The universality of skipping behaviours on music streaming platforms. *arXiv preprint arXiv:2005.06987*, 2020.

[14] Nicola Montecchio, Pierre Roy, and François Pachet. The skipping behavior of users of music streaming services and its relation to musical structure. *Plos one*, 15(9):e0239418, 2020.

[15] Aaron Ng and Rishabh Mehrotra. Investigating the impact of audio states & transitions for track sequencing in music streaming sessions. In *Proceedings of the 14th ACM Conference on Recommender Systems*, pages 697–702, 2020.

[16] Sainath Adapa. Sequential modeling of sessions using recurrent neural networks for skip prediction. *arXiv preprint arXiv:1904.10273*, 2019.

[17] Ferenc Béres, Domokos Miklós Kelen, András Benczúr, et al. Sequential skip prediction using deep learning and ensembles. 2019.

[18] Sungkyun Chang, Seungjin Lee, and Kyogu Lee. Sequential skip prediction with few-shot in streamed music contents. *arXiv preprint arXiv:1901.08203*, 2019.

[19] Andres Ferraro, Dmitry Bogdanov, and Xavier Serra. Skip prediction using boosting trees based on acoustic features of tracks in sessions. *arXiv preprint arXiv:1903.11833*, 2019.

[20] Gabriel Dulac-Arnold, Ludovic Denoyer, Philippe Preux, and Patrick Gallinari. Datum-wise classification: a sequential approach to sparsity. In *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2011, Athens, Greece, September 5-9, 2011. Proceedings, Part I 11*, pages 375–390. Springer, 2011.

[21] Jaromír Janisch, Tomáš Pevný, and Viliam Lisý. Classification with costly features using deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 3959–3966, 2019.

[22] Jun Feng, Minlie Huang, Li Zhao, Yang Yang, and Xiaoyan Zhu. Reinforcement learning for relation classification from noisy data. In *Proceedings of the aaai conference on artificial intelligence*, volume 32, 2018.

[23] Marco A Wiering, Hado Van Hasselt, Auke-Dirk Pietersma, and Lambert Schomaker. Reinforcement learning algorithms for solving classification problems. In *2011 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*, pages 91–96. IEEE, 2011.

[24] Jaromír Janisch, Tomáš Pevný, and Viliam Lisý. Classification with costly features as a sequential decision-making problem. *Machine Learning*, 109:1587–1615, 2020.

[25] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[26] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

[27] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. Dueling network architectures for deep reinforcement learning. In *International conference on machine learning*, pages 1995–2003. PMLR, 2016.

[28] Tensorforce Authors. TensorForce Github. `https://github.com/tensorforce/tensorforce`, 2017.

[29] Shane Griffith, Kaushik Subramanian, Jonathan Scholz, Charles L Isbell, and Andrea L Thomaz. Policy shaping: Integrating human feedback with reinforcement learning. *Advances in neural information processing systems*, 26, 2013.