Song Recommendation using Reinforcement Learning

Abhay Kumar Singh
Dept. of Computer Science & Engg.
Texas A&M University
College Station, Texas, USA
abhay@tamu.edu
UIN - 130004782

Mukund Srinath Heragu
Dept. of Computer Science & Engg.
Texas A&M University
College Station, Texas, USA
msh_tam@tamu.edu
UIN - 128003513

***Abstract*—A recommender system, or a recommendation system (sometimes replacing 'system' with a synonym such as platform or engine), is a subclass of information filtering system that seeks to predict the "rating" or "preference" a user would give to an item. [1]. They are primarily used in commercial applications.**
Recommendation systems are an important aspect in our day to day lives. Everything, ranging from the food we eat to the movies we watch, to the clothes we wear, are all recommended to us in some or the other way. Granted, we could hear about a lot of these different categories from people, but we cannot deny that recommendation systems also play a major role in what music we listen to or what shows we watch etc.
One of the ways that recommendation systems, that we interact with regularly, recommend things to us is by using supervised learning and memory based content-filtering from our history to recommend our next step. Although this works quite well, it is a static approach to recommendation systems and it completely disregards how these recommendation systems are impacted over each time interval.
Reinforcement Learning based recommendation systems work better as they learn ceaselessly and without break and so if and when the user's preferences change, so do the recommendations. [2]

A. Github URL: Click Here!

B. Youtube Video URL: Click Here!

.

I. Objective

The objective of this project is to design a model using Reinforcement Learning techniques to overcome the challenges faced by current recommendation systems and provide recommendations of higher quality. In this project, we work on the MSSD (Music Streaming Sessions Dataset) dataset. Recommender systems that are built using Reinforcement Learning approaches work better than traditional approaches as they can constantly update their values and converge to an optimal value.

II. Literature Survey

There is some work done in field of song recommendation system. Using reinforcement learning for recommendation system is relatively new field and we will be talking about few papers which solved this problem by either song recommendation or playlist recommendation.

A. DJ-MC: A Reinforcement-Learning Agent for Music Playlist Recommendation [3]

To formulate the problem into an MDP of (S,A,P,R,T) tuple, they considered every list of songs to be the list of state, set of actions are the next songs to be played, P and R functions are determined using the data provided and finally, the terminal state is the playlist of k length. In their case, the data was more scarce than computationally heavy, therefore they went ahead with a model-based method of reinforcement learning. For a model based approach, you need to know the reward function (R) as well as the Transition probability function (P). P was evaluated trivially by the data. The reward function R corresponding to a listener can be factored as the sum of two distinct components: 1) the listener's preference over songs in isolation, $R_s : A \rightarrow R$ and 2) his preference over transitions from past songs to a new song, $R_t : S \times A \rightarrow R$. That is, $R(s, a) = R_s(a) + R_t(s, a)$. Each song is modelled using spectral auditory descriptors and viewed as a vector with 34 features. The DJ-MC agent architecture contains two major components: learning of the listener parameters and planning a sequence of songs. The learning part is in itself divided into two parts - initialization and learning on the fly. Learning the listener parameters helped in avoiding cold start of a recommendation models. Due to large song space, DJ-MC exploits the structure of the song space by clustering songs according to song types and does planning on the abstract song type rather than concrete songs and thus reducing the tree search complexity.

B. Automatic, Personalized, and Flexible Playlist Generation using Reinforcement Learning [4]

Shih and Chi used Policy Gradient and Recurrent Neural Network to generate a playlist. Their system consists of three main steps. First, using Chen et al.'s work [5], they generated baseline playlist based on user preference about song. Using the bipartite graph between user and songs, they calculated the embedding feature of songs and thus obtained

the baseline playlist for each song using K-nearest neughbours. Second, by formulating baseline playlists as sequences of words, they pretrained RNN language model(RNN-LM) to obtain better initial parameters for the following optimization, using policy gradient reinforcement learning. Combination of neural network and reinforcement learning turned out to usedful to achieve better performance. Finally, given preferences from user profiles and the pretrained parameters, personalized playlists are generated by exploiting techniques of policy gradient reinforcement learning with corresponding reward functions. For policy gradient, they formulated the problem into MDP in the similar to Liebman et al.'s work [3] with action being the song id to recommend next and state being the already recommended songs till now. Reward is the linear combination 4 aspects of song rewards and Policy is the parameters of RNN-LM and the policy is optimised using Policy gradient method.

C. Generating Music Playlists with Hierarchical Clustering and Q-Learning [6]

In this paper, authors used Content Based (CB) method for Automatic Playlist Generation (APG) to inform an unsupervised machine learning algorithm (Q-learning) which can over time learn from implicit user feedback to generate personalized playlist for every user. Instead of using song's metadata, they extracted features for each song from the song audio data itself. To reduce the space and time complexity of the learning method, they proposed a solution that uses hierarchical clustering to reduce the song space size. So now, instead of transitioning from song to song, they will transition from cluster to cluster. To formulate the problem into an MDP, the set of states S corresponds to the set of clusters and an action is choosing the next cluster. Since they used Q-learning which is a model-free learning method, they don't need a transition probability and reward function. And while agent was learning, they considered reward as 0 or 1 based on the behaviour of the user if he skips the song or listen to it. They also introduced a playlist reward for small length playlist.

D. Playlist Recommendation based on Reinforcement Learning [7]

In this paper, recommendation is considered to be based on information obtained from content as well as the relationship between correlation within the playlist. The recommendation system is modeled on an MDP or Markov Decision process. The authors also talk about a compression method that solves the problem of an extremely large state space. Their recommendation strategy is based on the RLWRec model which is in turn based on the Q-learning model with an ε-greedy strategy. Initially, the states of the model are considered as a sliding window of size k, where k is the number of songs that the user has listened to so far. Each subsequent song is regarded as an action. Once a new song has been recommended, the new song is then sent to the state space as a new state s- and the first song that was in the window is removed. Therefore, the state space remains the same with only the latest k number of songs listened to. This still means that the state space is very large. To overcome this problem, the authors have used a collaborative filter that replaces songs with song clusters based on some similarity features. Listen, collect, download are the three different types of feedback information that are used as reward functions. The learning algorithm employed is the Q-learning algorithm and also apply the ε-greedy approach to solving the problem of picking the next best states and actions. According to the authors, this also increased the overall coverage space that can be used for recommendation. Instead of just using the Q value to determine the next song for recommendation, they have also used probabilities of selecting clusters and songs to improve the efficiency and accuracy of their system.

E. Enhancing Collaborative filtering music recommendation by balancing exploration and exploitation [8]

The authors of this paper talk about how the general idea of recommendation is a greedy approach that only considers, the exploitative strategies and ignore the explorative strategies, especially in collaborative filtering methods. The main idea proposed in this paper involves using Bayesian graphical models that takes into account the latent factors of the Collaborative filtering model and the novelty of the music in question. The next step is to apply the Bayesian inference algorithm that can predict the posterior distributions of the ratings. The Bayesian inference algorithm is a substitute for the MCMC (Markov Chain Monte Carlo) algorithm which does not work as effectively as the Bayesian algorithm for this problem statement. Matrix factorization is done to characterize the latent factors of users and the songs. The Reinforcement learning approach used in this paper is the n-armed Bandit. A user's rating of a song is affected by not only the latent factors of the song from the CF method, but also the novelty of the song. The authors use different formulae to determine the value of these scores and use them together to finally determine how a user would rate a particular song. Since, the greedy approach is not a favorable one over the long term, it is necessary to extract other factors that affect the preferences of a user and since this is done by exploration, the authors have used Bayesian Upper Confidence Bounds [9]. The song with the highest value amongst the posterior distribution will then be recommended to the user.

III. Dataset

As mentioned earlier, we have used the MSSD dataset (Music Streaming Sessions Dataset) which was originally published by Spotify during one of their competitions. [10]. This dataset mainly consists of 2 data files. The first data file is the session data file. This file contains all the information about each session. A session is comprised of twenty songs together, one after the other and the tracks are the songs that people listen to.

Within each session, we not only have information about the different tracks in that session, but also a user's preferences in that session. We have additional information that talks about the transition between tracks and also the actions within

the tracks themselves. Actions between the tracks include information like: if the track was skipped, how much of the track did the user listen to and if the track wasn't skipped. Actions within the tracks talk about how the user played or paused the song (how long did the user wait before playing or pausing) and if the user moved forward or backward within the song. Figure1 helps us visualize the structure of a session.

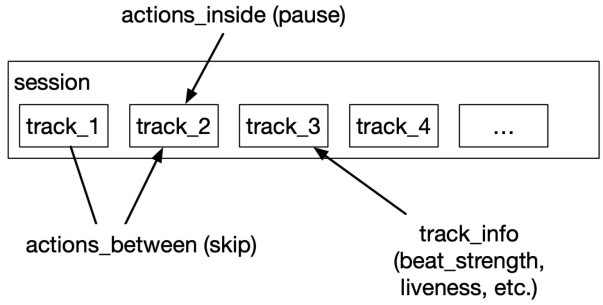

Fig. 1. Session Representation

All this information signifies the user's mannerisms and ways while harkening to music. This information that provides a user's choices plays a significant role in deciding the reward functions.

In the second data file, each track also has it's own features like danceability, acousticness, seven different acoustic vectors, loudness, beat_strength etc; There are about 20 different song features for each of the songs. As we will see later, these features also play an important role in deciding the transition between songs. URL [11]: https://www.aicrowd.com/challenges/spotify-sequential-skip-prediction-challenge/dataset_files

IV. Formulating an MDP

In order to apply Reinforcement Learning Algorithms to our data, we need to transform our data to a MDP or Markov Decision Process. A Markov decision process (MDP) is a discrete time stochastic control process. It provides a mathematical framework for modeling decision making in situations where outcomes are partly random and partly under the control of a decision maker. [12]. To fabricate our MDP, let's define some mathematical terms:

- t : refers to a particular instant in time.
- X_t : is the decision of the user(has the song been skipped or not). These values can either be 0 or 1. 0 refers to the song being skipped and 1 indicates that the user listened to the song.
- f_t : is the feature vector of the song playing at time t.
- S_t : refers to each step in the session which has the last 5 songs listened to as the state.
- a_t : refers to the action at time t.
- r_a: denotes the reward obtained by taking action 'a'.
- Q : refers to the Q-value.

$$S_t = [f_{t-6:t-1}, X_{t-6:t-1}]$$

The above equation represents the equation for a set of 5 states or songs where $f_{t-6:t-1}$ is the feature vector for each of the 5 songs.

V. Approach

The MSSD dataset is extremely large and contains about 20 million songs and about 17 million sessions. The entire datasaet is about 600 GB of data. For our experimental purposes, we have considered about 10% of the data. We have in our dataset, about two hundred thousand songs (200,000 +) or tracks, and about a hundred thousand sessions.

In order to get this smaller size of data, the data had to be sampled. To perform sampling of data, we first sampled the session data and from the session data by choosing the session with length 20, we obtained the track information for those sessions.

The data itself has already been divided into 10 zipped file with about 60 GB of data in each of the files. Within each of the zipped files are a certain number of data files. Let us call the number of data files as 'X'. But the data files do not have an equal number of sessions. Since they are different, we had to sample about sessions

$$\frac{10,000}{X} \tag{1}$$

from each file with a probability. If we consider the value S as the number of sessions for a file, the probability of sampling data from the file can be given as

$$\frac{10,000}{X * S} \tag{2}$$

Now that we have all the sessions we need, we need to extract the tracks that are in each of the sessions. We created a dictionary of the session information and from each of the sessions, we retrieved the track ids of all the tracks and used the ids to obtain the track information. The overall sampling was done on about three hundred different data files. Figure 2 shows the entire workflow of the project.

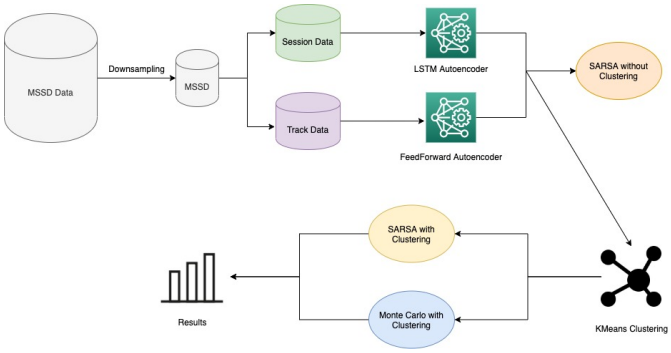

Fig. 2. Entire Work Flow

A. Data Processing

Now that we have the data, we can decide what approach to use and what algorithms to apply. Before implementing a model, we need to process the data to ensure it works well with any algorithm that we choose to use.

Our tracks dataset contains songs with than 29 features. Out of those 29, we filtered 20 features which defines the song type and genres. Table V-A shows few of the important track features that defines a song. The 20 features for every song

Important Track Features		
acousticness	beat_strength	bounciness
danceability	energy	flatness
instrumentalness	liveness	mechanism
organism	speechiness	valence

is large enough to increase the complexity of the problem. We can reduce the size of song features using autoencoders. Autoencoders are neural networks that where the input and output is same. It tries to learn the data itself and in the process, converts the data into a short latent form in the middle of the network. Figure 3 shows the encoder-decoder architecture. To reduce the dimension of track features, we

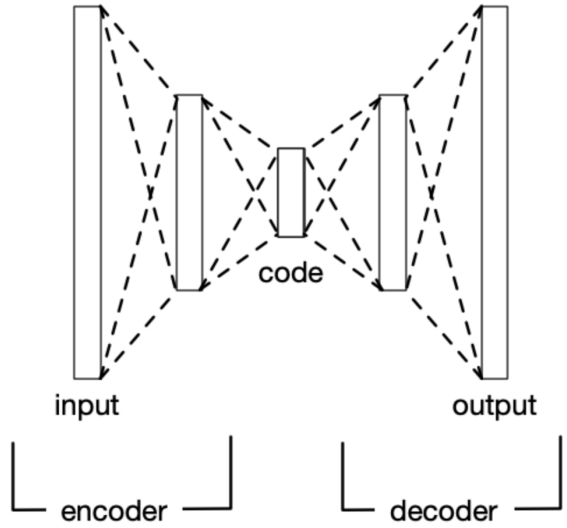

Fig. 3. The Encoder-Decoder architecture

trained a simple feed forward autoencoder that takes the 1x20 feature vector and encode it into a 1x8 latent representation of the features. Figure 5 shows the model summary of the autoencoder architecture we designed to reduce the dimension of track features.

Once, we have the latent representation of the track features, next step is to encode each state itself to reduce the dimension. As shown in Figure 1, a session is represented by number of tracks along with the action performed between those songs. As part of this project, we are only considering skip/no_skip feature of track transition. Therefore, if we

```
Model: "model_14"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_9 (InputLayer)         (None, 20)                0
_________________________________________________________________
dense_28 (Dense)             (None, 14)                294
_________________________________________________________________
dense_29 (Dense)             (None, 8)                 120
_________________________________________________________________
dense_30 (Dense)             (None, 14)                126
_________________________________________________________________
dense_31 (Dense)             (None, 20)                300
=================================================================
Total params: 840
Trainable params: 840
Non-trainable params: 0
_________________________________________________________________
```

Fig. 4. Track Feature Autoencoder

take 5 songs and its transition as a single state, it will be represented in a 5x21 matrix where 20 are the song features and 1 is the skip/no_skip action. Since we have already compressed the song feature vector to 8 from 20, therefore the state matrix size will become 5x9. Once extracting all 5x9 state from the dataset, we created an LSTM Autoencoder to reduce the dimension of state from 5x9 to 1x9. LSTM autoencoders are used when we have a sequence in the data and from our state representation, we can clearly see that our state contains sequence of 5 songs and actions over it and if we create a new permutation of those 5 sequences, it will be considered as a new state. Figure **??** shows the model summary of the autoencoder architecture we designed to reduce the dimension of state features.

```
Model: "sequential_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_17 (LSTM)               (None, 5, 18)             2016
_________________________________________________________________
lstm_18 (LSTM)               (None, 9)                 1008
_________________________________________________________________
repeat_vector_7 (RepeatVecto (None, 5, 9)              0
_________________________________________________________________
lstm_19 (LSTM)               (None, 5, 9)              684
_________________________________________________________________
lstm_20 (LSTM)               (None, 5, 18)             2016
_________________________________________________________________
time_distributed_7 (TimeDist (None, 5, 9)              171
=================================================================
Total params: 5,895
Trainable params: 5,895
Non-trainable params: 0
_________________________________________________________________
```

Fig. 5. Session State Feature Autoencoder

We trained both autoencoder for 50 epochs and 128 batch size. Table V-A shows the MSE loss of both of the autoencoders.

Autoencoder	Results
Track Features Autoencoder	MSE Loss: 0.041
Session State Features Autoencoder	MSE Loss: 0.0037

Method of dimensionality reduction (autoencoder) will reduce the number of dimensions in which track and state is represented. But it will not reduce the number of states we

have or apply some approximation over it. To reduce the state and action (track) space, we have performed clustering on the songs and the sessions to have songs with similar feature vectors and sessions with similar transitions and songs in the same cluster. [13] We decided to use an unsupervised approach and used the K-Means Algorithm to implement clustering and divided the 200K+ songs into 100 and 1000 clusters and compared their results and also clustered the sessions into 1000 and 10,000 clusters. Since K-Means algorithm on high values (like 1000 and 10,000) consume a lot of resources, we implemented the algorithms using Spark Programming and trained the model on EMR (Amazon Elastic MapReduce) clusters with 5 m5.xlarge nodes to get results faster. Figure 6 shows an example of how the data looks after clustering.

	track_id	0	1	2	3	4	5	6	7	cluster
0	t_f9b9a072-8dae-4816-bfd6-d91fc75a5744	0.555277	0.543247	0.685906	0.274507	0.674009	0.417390	0.446871	0.799099	756
1	t_7f435320-5d3b-497d-9346-0c97429af2b9	0.618304	0.414371	0.455408	0.635635	0.762511	0.544788	0.388787	0.666174	525
2	t_949441c4-4920-4eed-b929-2d67d37b7bd9	0.527924	0.451953	0.628986	0.332040	0.862317	0.380717	0.386505	0.734776	283
3	t_63eae5ac-744a-46ee-b719-0db8e70d6f7f	0.587948	0.457448	0.577631	0.343994	0.840386	0.437686	0.434940	0.714831	280
4	t_3c2c29b5-c738-42c0-83e1-ee0e88ac834d	0.545369	0.607428	0.880114	0.396198	0.756682	0.493495	0.310475	0.583829	396

Fig. 6. How the data looks after clustering

B. Method

A typical Reinforcement Learning model has states, actions, reward functions, an agent and an environment. Since we are working on a recommender system, we have an imaginary user that is listening to the songs and the agent our model that recommends the songs to this imaginary user. This imaginary user is the environment that responds to the recommendations by choosing to listen to or skip the song. Choosing to listen or skip is what we have considered to be the reward in this case. The states are the last 5 songs that the user has listened to and the actions are what song to recommend next. The state acts as a queue. The actions taken are added to the queue and the song at the front of the queue is taken out. This way, we always maintain 5 songs as the state. Figure 7 shows how a basic model of the agent, environment, action and reward looks.

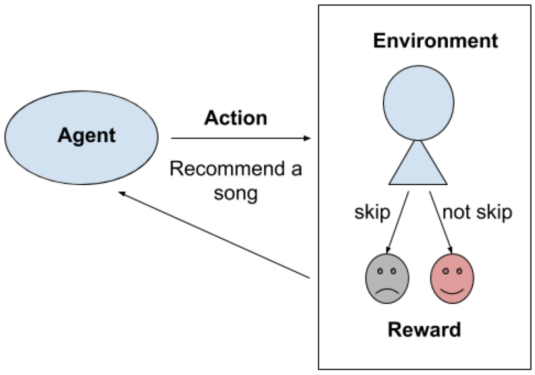

Fig. 7. Simple representation of the model

To start the learning, we considered each session as an episode, maintained the policy using Q matrix. When calling a policy for a state, it will return the action with maximum q-value and if the state is not present in the Q matrix, it will return a random action. We tried to create the Q matrix or learning using 2 reinforcement learning algorithms - SARSA and Monte Carlo. We ran the both of these algorithms but will different approaches -

- **SARSA without Clustering:** We ran SARSA algorithm without performing clustering over the state and action space. In this case, when calling a policy for a state, it returned the action with max q-value if state was present in the Q matrix. If current state was not present, it iterated through the entire Q matrix to find the state with minimum euclidean distance to the current state. We made an assumption that if we encountered a very new state while learning, then we can approximate it to the closest state explored till now (nearest neighbour). To evaluate whether the action returned by the policy and the actual action are same or similar, we evaluated the euclidean distance between them and if that values is less than a threshold, we assumed it to be a correct/good recommendation.
- **SARSA with Clustering:**: We ran SARSA algorithm with clustering over the state and action space. It reduced the state space for the problem and thus, made the algorithm faster. In this case, when calling a policy for a state, it returned the action with max q-value if state cluster was present in the Q matrix. If current state cluster was not present, it returned a random action. To evaluate whether the action returned by the policy and the actual action are same or similar, we compared the cluster id of both the songs and if songs recommended and actual songs lies in same cluster, then it is a good recommendation. We made an assumption here that songs with very similar features (same genres) will lie in same cluster and thus the recommendation will be good.
- **Monte Carlo with Clustering:**: We ran Monte Carlo algorithm with clustering over the state and action space. It followed the exact same approach as of SARSA with clustering.

VI. Experiments

We implemented two Reinforcement Learning algorithms; SARSA and Monte Carlo and used the clustered data with multiple values of K and compared the results. We used moving average error as the metric to evaluate the learning process.

1) SARSA without Clustering: SARSA without clustering took a lot of time of complete and therefore we killed the process in between. It took around 10 hours to complete 3000 episodes. This inspired us to apply clustering over the state and action space to introduce some approximation and increase the learning speed.

2) SARSA with Clustering: We ran SARSA with clustering on multiple value of cluster size k.

- **State k = 10000 and action k = 1000:** The high cluster k value didn't gave us good results as most of the predictions came out to be wrong. Out of 99660 episodes, only 437 episodes gave error below 0.5. Figure 8 shows the error plot for all episodes where x-axis is the episode count and y-axis is error in each episode.

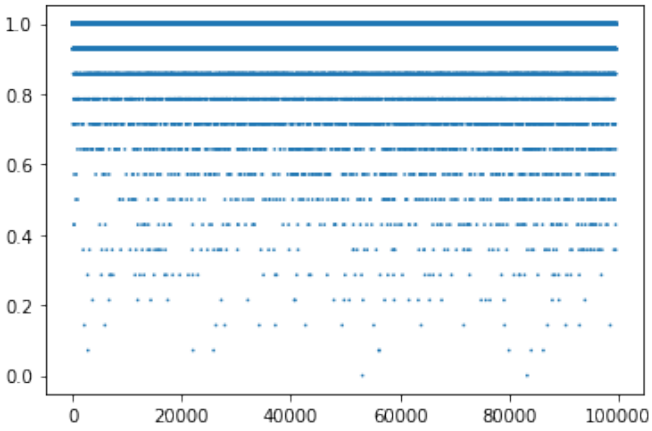

Fig. 8. SARSA with Clustering error plot: State k = 10000 and action k = 1000

- **State k = 1000 and action k = 100:** We experimented on reduced cluster size to see the behaviour of the algorithm. Out of 99660 episodes, 893 episodes gave error below 0.5, a slight improve over the previous one. Figure 9 shows the error plot for all episodes where x-axis is the episode count and y-axis is error in each episode.

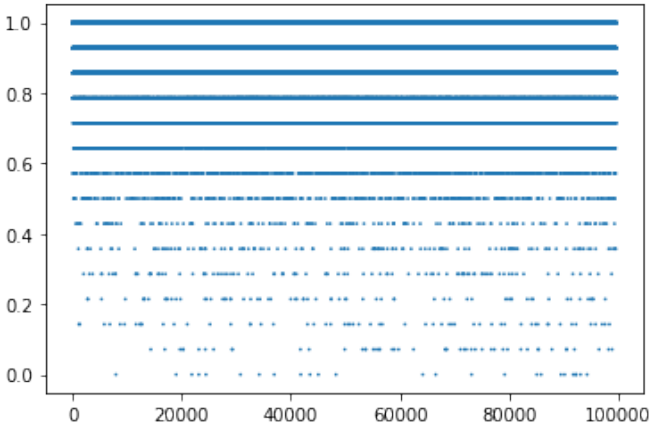

Fig. 9. SARSA with Clustering error plot: State k = 1000 and action k = 100

3) Monte Carlo with Clustering: We ran Monte Carlo with clustering on multiple value of cluster size k.

- **State k = 10000 and action k = 1000:** We got lower results with respect to SARSA. Out of 99660 episodes, only 25 episodes gave error below 0.5. Figure 10 shows the error plot for all episodes where x-axis is the episode count and y-axis is error in each episode.

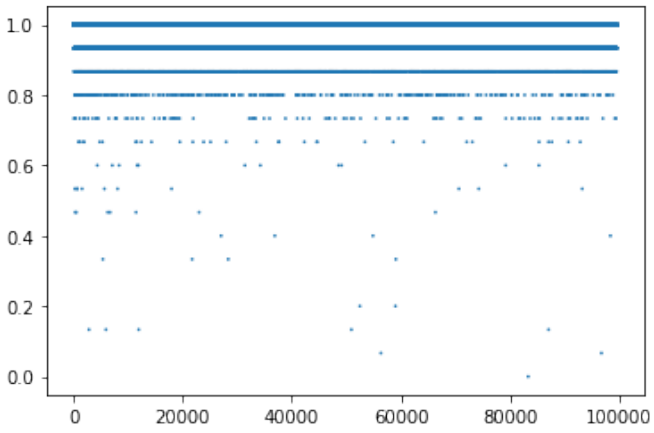

Fig. 10. Monte Carlo with Clustering error plot: State k = 10000 and action k = 1000

- **State k = 1000 and action k = 100:** Reduced cluster size gave good results with respect to higher cluster size for monte carlo. Out of 99660 episodes, 395 episodes gave error below 0.5. Overall, Monte Carlo didn't performed better than SARSA. Figure 11 shows the error plot for all episodes where x-axis is the episode count and y-axis is error in each episode.

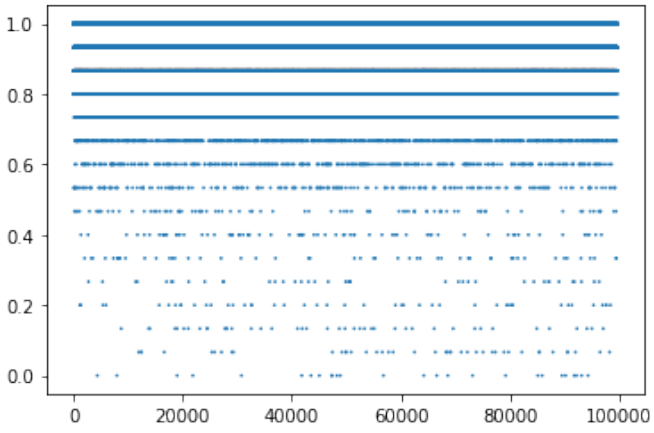

Fig. 11. Monte Carlo with Clustering error plot: State k = 1000 and action k = 100

Comparing above 2 algorithms with clustering, SARSA performed better than Monte Carlo for all the cluster sizes. Decreasing the size of cluster improved the results a bit but decreasing more will lead to more generalization of state and action space and thus will underfit learning. The learning is not much improved with the increase in the episode and this gives the opportunity to experiment more and try out a lot of things to understand how this problem can be solved using reinforcement learning.

VII. Conclusion

From the results we can see that although we weren't always able to recommend good songs, we were able to recommend really good music in some of the episodes. We implemented two traditional Reinforcement Learning Algorithms to help compare the results and realize a good approach to solving the problem.

In Future we plan to improve the efficiency of the Autoencoders, perform clustering in different ways, run hyperparameter tuning to find good values of cluster size and also implement other algorithms like the Deep-Learning based Reinforcement Learning algorithm that we think would give us better results.

References

[1] Recommender systems definition as given in wikipedia. 2019.

[2] Aishwarya Srinivasan. Recommendation system using reinforcement learning. 2019.

[3] Elad Liebman and Peter Stone. Dj-mc: A reinforcement-learning agent for music playlist recommendation. 01 2014.

[4] Shun-Yao Shih and Heng-Yu Chi. Automatic, personalized, and flexible playlist generation using reinforcement learning, 2018.

[5] Chih-Ming Chen, Ming-Feng Tsai, Yu-Ching Lin, and Yi-Hsuan Yang. Query-based music recommendations via preference embedding. In *RecSys '16*, 2016.

[6] James King and Vaiva Imbrasaitė. Generating music playlists with hierarchical clustering and q-learning. In Allan Hanbury, Gabriella Kazai, Andreas Rauber, and Norbert Fuhr, editors, *Advances in Information Retrieval*, pages 315–326, Cham, 2015. Springer International Publishing.

[7] Binbin Hu, Chuan Shi, and Jian Liu. Playlist recommendation based on reinforcement learning. pages 172–182, 09 2017.

[8] James McInerney, Benjamin Lacker, Samantha Hansen, Karl Higley, Hugues Bouchard, Alois Gruson, and Rishabh Mehrotra. Explore, exploit, and explain: Personalizing explainable recommendations with bandits. In *Proceedings of the 12th ACM Conference on Recommender Systems*, RecSys '18, page 31–39, New York, NY, USA, 2018. Association for Computing Machinery.

[9] Emilie Kaufmann, Olivier Cappe, and Aurelien Garivier. On bayesian upper confidence bounds for bandit problems. In Neil D. Lawrence and Mark Girolami, editors, *Proceedings of the Fifteenth International Conference on Artificial Intelligence and Statistics*, volume 22 of *Proceedings of Machine Learning Research*, pages 592–600, La Palma, Canary Islands, 21–23 Apr 2012. PMLR.

[10] Sophie Zhao, Yizhou Wang, and Feng Qian. Recommendation system with reinforcement learning. 2019.

[11] Spotify. Spotify sequential skip prediction challenge. 2019.

[12] Wikipedia. Markov decision process.

[13] Emre Rençberoğlu. Clustering the most listened to songs of the 2010s using spotify data. 2018.