SpotifyDJ

A Multi-Armed Bandit Virtual DJ

By

Michael Ross
SID: [redacted]

A DISSERTATION

Submitted to

The University of Liverpool

in partial fulfillment of the requirements
for the degree of

MASTER OF SCIENCE
DATA SCIENCE & ARTIFICIAL INTELLIGENCE

23 SEPTEMBER 2022

Supervisor: [redacted]
Second Marker: [redacted]

ABSTRACT

Recommendation systems are implemented more than ever across nearly all entertainment and e-commerce platforms. These systems are designed to increase user engagement, giving a sense of personalization that companies hope will result in long-term satisfaction and engagement. However, the majority of these recommendations are usually presented to the user in slate form – a list of products chosen from a candidate pool, displayed to the user normally on the platform's homepage.

While slate recommendations can be useful in many contexts, it does present several issues – most notably the issue of sparsity. The underlying algorithms behind these recommendation systems are designed to adapt to the user's preference profile, but if the user simply chooses not to interact with any of the recommendations made in the current slate, it is difficult to discern whether the lack of interaction was due to poor recommendations or if there was an external factor inhibiting the user from providing useful feedback for what could have been great recommendations. For this reason, it is necessary to create a recommendation system that almost guarantees feedback from the user, allowing for real-time adaptation for improved recommendations. A great avenue to test this approach is with music recommendations.

This work introduces SpotifyDJ, a reinforcement learning-based music recommender that, in essence, performs for the user by selecting songs and plays them one at a time, using user feedback for each song to update its trajectory of song sequences. SpotifyDJ acts as a feature that is activated only if the user selects it and the feedback the user provides (skip, like, dislike, save, etc.) are mapped with a set of rewards that accumulate throughout the listening session where SpotifyDJ learns to maximize. This is accomplished by using a multi-armed bandit, a classic reinforcement learning framework while comparing the performances of a random versus greedy policy in accumulating rewards. Simulation results show that SpotifyDJ learns the user's music preferences based on limited starting information by balancing music exploration and knowledge exploitation.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

Entertainment mediums available to society today all utilize some form of a recommender system, providing a personalized touch for the user's experience on the platform with surveys conducted in 2017 showing that 80% of respondents would do business with a company if it offers personalized experiences (Epsilon, 2018). Some examples of these recommendation systems can be seen daily while shopping online on Amazon or deciding what movie to stream on Netflix. A company's ability to accurately predict what movie, news article, advertisement, social media connection, or even songs its users will positively interact with provides a significant business advantage with users viewing the service as enjoyable, resulting in long-term use through customer loyalty.

These platforms use a combination of two primary types of recommendation systems: content-based and collaborative-based (Madhukar, 2014). With content-based systems, users are presented with products or services that fall in the same or similar category that the user most interacts with. Collaborative filtering systems, however, use a community-based approach in which users are recommended products or services based on a similarity measure of other users. Both content and collaborative-based systems require a profiling of the user's preferences in order to provide useful recommendations and are largely presented in slate form (Madhukar, 2014). An example of a slate recommendation is Spotify's "Discover Weekly" playlist unique for each user consisting of songs that the user may like.

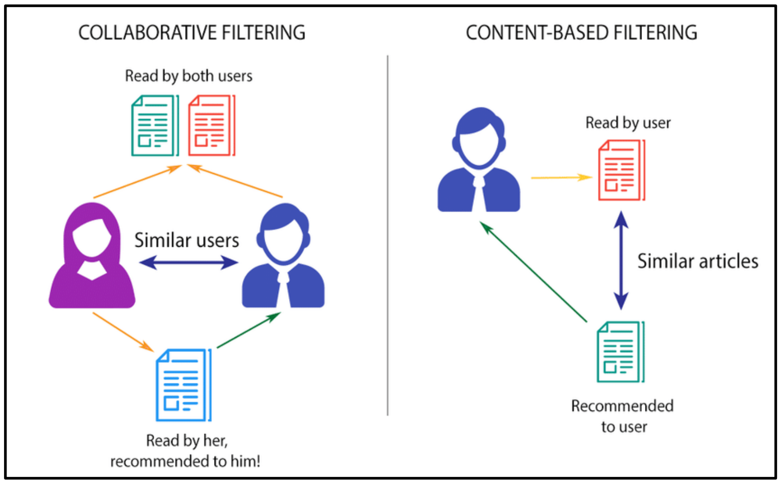


Figure 1. Collaborative vs. content-based filtering in recommendation systems (Tondji, n.d.)

Though slate recommendations may be an effective approach for providing a sense of personalization (Ie *et al*., 2019), it also has the potential to become irrelevant altogether. There are several limitations to the current slate recommendation formats: 1. Inability to overcome the issue of a *cold start* (Milankovich, 2015), 2. No guarantee of an adaptive recommendation algorithm due to sparse feedback from the user (Al-Bakri and Hashim, 2018), 3. Slate recommendations are not designed to satisfy a specific intent a user may have for music listening sessions, 4. Recommendations may become repetitive as a result of relying on the user's preferences and as a result, recommendations may become predictable (Amer-Yahia *et al.,* 2009).

The first obstacle facing slate recommendations is that it cannot overcome the issue of a cold start – a problem when a lack of information on the user leads to the inability to provide useful recommendations. This is a recurring issue across all platforms when there is a new user or a very inactive user. On the Spotify platform, there would not be an issue with a cold start amongst active users that regularly interact with songs, providing regular feedback that creates a preference profile boundary for the recommendation system to reference (Milankovich, 2015).

Additionally, even if cold start isn't an issue and there is enough information on the user's preferences to provide useful recommendations, the underlying algorithms behind providing the recommendations itself are designed to be adaptive to allow for changing preferences. For Spotify, the "Discover Weekly" playlist is displayed on the homepage and updated weekly. However, it is completely up to the user to decide whether to interact with the playlist itself. If that is the case, there is no way to interpret the performance of the recommendation algorithm because feedback provided is sparse (Al-Bakri and Hashim, 2018). If the user does not interact with the recommendations provided, is this interpreted as a poorly constructed playlist of recommended songs or could there have been an external factor that prevented the listener from interacting with the playlist altogether?

It is also entirely possible that the playlist created for the user did not satisfy a specific intent. The purpose of music listening sessions vary drastically. Does the listener want recommendations for new music discovery purposes or to compliment and capture a specific theme? Traditional recommender systems cannot satisfy the intent a user may have for each user's specific music listening session (Lalmas and Mehrotra, 2021). Furthermore, users may avoid the recommendations provided altogether after realizing that the recommendations start to repeat itself either through the specific songs recommended or with the same artists week after week.

To combat the limitations of the current state of music recommendation systems, this project presents SpotifyDJ, an adaptive reinforcement learning-based music recommender under the guise of a personalized virtual DJ. SpotifyDJ is a proposed feature that would only be activated if the user chooses to do so. The user selects an initial starting song and using the features of that song, a candidate pool is created of similar songs; however, the candidate songs are pulled from user-created playlists making this a community-based approach. As of October 2021, Spotify consists of over 4 billion user-created playlists (Dean, 2021) that all capture some unique theme or song sequence. For example, the same song can appear in a pop-genre playlist as well as a 1990's themed playlist. Taking advantage of the diversity of playlists provided by Spotify, a diverse set of candidates are created for SpotifyDJ to use. From there, we formulate the song selection process as a Markov Decision Process covered in section 2.

1.1 AIMS & OBJECTIVES

- Establish an understanding of the current state of recommendation systems
- Explore current research on implementations of reinforcement learning for recommendation systems
- Design a recommender system that can overcome a cold start, requiring very little information to provide useful song recommendations for users
- Construct candidate songs using user-created playlists made available by Spotify
- Design a recommender system that is adaptable to changing music preferences yielding longer user engagement
- Implement a music streaming feature that can satisfy a user's specific music listening intent
- Outperform a randomized recommendation algorithm associated with slate recommendations

1.2 RELATED WORKS

The use of reinforcement learning in recommendation systems has been researched and slowly experimented with in recent years. For example, multi-armed bandits are being used to determine relevant advertisements for a user to see on any given platform. With music, however, there has not been an extensive application of reinforcement learning with very few proposals for the applications of reinforcement learning for music recommendations overall. It is also entirely possible that Spotify, a goliath in the music industry with billions in resources, are already researching an effective method to use reinforcement learning for their platform if deemed feasible.

The most notable research published on the use of reinforcement learning for music recommendations was in 2014 by the University of Texas, Austin when "DJ-MC" was presented as a holistic playlist generator that emphasized learning user preferences to guide song sequences in playlist creation (Liebman, Saar-Tsechansky and Stone, 2015). As with standard reinforcement learning tools, DJ-MC also formulates their song selection process utilizing a Markov Decision Process where a finite set of *n* musical tracks $M = \{a_1, a_2, \ldots, a_n\}$ is defined by the entire catalogue of songs and a predetermined playlist generation of length *k*. If DJ-MC were to use Spotify, *M* would be equivalent to an environment of approximately 82 million songs, according to Wikipedia (Wikipedia, 2022). From a finite set of approximately 82 million songs, DJ-MC constructs a recommendation playlist of k songs – in essence, this is a slate recommendation. Despite the slate recommendation format used in DJ-MC, this project will take inspiration from the underlying components to construct a comparison report.

1.3 ETHICAL USE OF DATA

This project utilizes the Spotify Million Playlist Dataset that was made publicly available as part of the RecSys Challenge 2018 hosted by Spotify directly (AIcrowd, 2018). The dataset contains one million playlists that were sampled from all user-created playlists on Spotify at the time of the competition and does not contain any personally identifiable information nor does it contain any information in which a user can be identified. The playlists contain all relevant information about itself to include name of playlist, number of tracks, track title, track artist, duration of song, and the track's unique ID. Furthermore, evaluation of the model itself is performed in a simulation to overcome the challenges of the time and constraints required to test enough people for long enough sessions.

2. DESIGN AND FRAMEWORK

2.1 DATA

As stated in Section 1.3, this project utilizes the Million Playlist Dataset made public by Spotify directly as part of a recsys challenge in 2018. For the purposes of reducing memory load, this project utilizes a subset of 20,000 playlists that consists of over 1.3 million songs. The mean playlist length is 67 songs, and the median length is 49 songs.

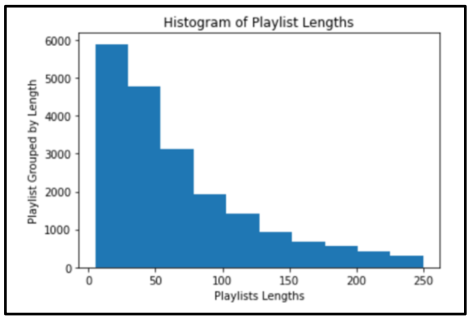


Figure 2. Playlist lengths shown in 10 bins

```
"playlists": [ { "name": "Throwbacks", "collaborative": "false", "pid": 0, "modified_at": 1493424000, "num_tracks": 52, "num_albums": 47,
"num_followers": 1, "tracks": [ { "pos": 0, "artist_name": "Missy Elliott", "track_uri": "spotify:track:0UaMYEvWZi0ZqiDOoHU3YI",
"artist_uri": "spotify:artist:2wIVse2owClT7go1WT98tk", "track_name": "Lose Control (feat. Ciara & Fat Man Scoop)", "album_uri":
"spotify:album:6vV5UrXcfyQD1wu4Qo2I9K", "duration_ms": 226863, "album_name": "The Cookbook" }, { "pos": 1, "artist_name": "Britney
Spears", "track_uri": "spotify:track:6I9VzXrHxO9rA9A5euc8Ak", "artist_uri": "spotify:artist:26dSoYclwsYLMAKD3tpOr4",
"track_name": "Toxic", "album_uri": "spotify:album:0z7pVBGOD7HCIB7S8eLkLI", "duration_ms": 198800,
"album_name": "In The Zone" }
```

Figure 3. An example playlist

2.2 ANALYZING TRACK FEATURES

Utilizing the Spotify API, we can use each track's URI provided by the Spotify Million Playlist Dataset to extract its specific track features. The features are used in order to generate candidate songs for the SpotifyDJ feature. For this project, a candidate pool of 40 songs is generated prior to formulating our song selection process detailed in Section 2.4.

```
[{'danceability': 0.581, 'energy': 0.963, 'key': 11, 'loudness': -4.087, 'mode': 1, 'speechiness': 0.0981, 'acousticness': 0.0295, 'instrumentalness': 0,
 'liveness': 0.139, 'valence': 0.788, 'tempo': 129.992, 'type': 'audio_features', 'id': '3cHyrEgdyYRjgJKSOiOtcS',
 'uri': 'spotify:track:3cHyrEgdyYRjgJKSOiOtcS', 'track_href': 'https://api.spotify.com/v1/tracks/3cHyrEgdyYRjgJKSOiOtcS',
 'analysis_url': 'https://api.spotify.com/v1/audio-analysis/3cHyrEgdyYRjgJKSOiOtcS', 'duration_ms': 204160, 'time_signature': 4}]
```

Figure 4. Example features returned by Spotify audio_features function for random track

```
extracted features: [0.581, 0.963, 11, -4.087, 1, 0.0981, 0.0295, 0, 0.139, 0.788, 129.992]
```

Figure 5. Numerical features extracted

As stated, SpotifyDJ would only work when activated by the user after providing an initial starting song. While the initial song is playing, a search function parses through every user-created playlist (20,000 playlists in this project) to find every playlist that contains the initial starting song. This project generalizes the community-based filtering method used in common recommendation systems to take advantage of the natural song sequences selected by real users directly and increase chances of music diversity. This decision was made intentionally in order to avoid predictability (Amer-Yahia *et al.,* 2009), as discussed in Section 1.

The search function will return a temporary list of varying lengths containing every song in each playlist identified. We then extract the features of each song to conduct a clustering algorithm to group the candidate songs into quartiles, in which each quartile will essentially act as the "arm" in our multi-armed bandit problem. However, unlike other unsupervised learning algorithms such as k-means or k-medians which use Euclidean distance and Manhattan distance respectively to group points into clusters, this is not ideal for our features due to high dimensionality as every feature shown in Figure 4 acts as a dimension and would therefore result in poor distance distinction (Aggarwal, Hinneburg and Keim, 2001).

Instead, L2 normalization is performed on the differences of candidate songs' features and initial song's features to return a single variable length given that:

$$x = \text{candidate\_song} - \text{starting\_song},$$

$$\text{where } |x| = \sqrt{\textstyle\sum_{k=1}^{n} |x_k|^2} \text{ or } |x| = \sqrt{x_1^2 + x_2^2 + x_3^2 + \cdots x_{11}^2}, \text{ for all candidate songs.}$$

Figure 6. L2 normalization

2.3 MULTI-ARMED BANDIT FRAMEWORK

As opposed to the model-based approach used by DJ-MC, this project uses a model-free approach using the Q-Learning algorithm. Using this approach has the benefit of being significantly less computationally demanding than a model-based approach due to model-based approaches simulating or planning ahead for best trajectories (Liebman, Saar-Tsechansky and Stone, 2015). With q-learning, and the ability to reference q-values, our action selection process becomes deterministic and does not require planning ahead, making this a much quicker approach.

We first formulate the song selection process as a classic Markov Decision Process (MDP) where a single episode of an MDP consists of a state in a set of states $s \in S$, an action from a set of actions $a \in A$, and a reward from a set of rewards $r \in R$. Using this method, the agent learns through a system of trial and error, taking record of the state-action pair *Q(S,A)* and its corresponding reward given for that action. These state-action pairs along with the reward, make up our Q-values that provide guidance for which action is best.

By selecting an action $a \in A$, the agent transitions to a new state, in which it makes another action. Each action taken yields some negative or positive reward that is the driving force behind finding an optimal policy denoted as π^*, which defines which action an agent should take (Slivkins, 2022). At each time *t,* the agent is presented with a set of four songs, in which the action it can take is defined as choosing the index of a song. The song is played, giving the listener an opportunity to provide some feedback that is synonymous with the reward the agent receives for that song selection.

$$candidates = \begin{Bmatrix} s_{1,1} & s_{2,1} & & s_{10,1} \\ s_{1,2} & s_{2,2} & & s_{10,2} \\ s_{1,3} & s_{2,3} & \cdots & s_{10,3} \\ s_{1,4} & s_{2,4} & & s_{10,4} \end{Bmatrix}$$

Figure 7. Candidate songs constructed in 4x10 structure

To identify an optimal policy π*, SpotifyDJ tactically balances an exploration vs exploitation action selection strategy by deploying an epsilon ϵ-greedy strategy. In the earlier stages, when there is relatively unknown rewards and user behavior, SpotifyDJ will explore its environment by selecting nearly all possible actions to observe its associated rewards, recorded in the q-values for the specific user. After this process is conducted several times, it will eventually trigger an exploitation condition in which it will choose the action that yields the greatest reward references in the q-values. The rate at which SpotifyDJ will explore is determined by each individual user's epsilon value assigned ϵ. This is a value that is learned through simulation trials to create a sample population for evaluation purposes and detailed in Section 3. The epsilon value is defined as the rate at which the agent will explore rather than exploit its current knowledge of the user. This project also tests the idea of decay, which reduces the exploration rate after certain iterations to reach exploitation faster and at a higher rate.

2.4 Q-VALUES AND REWARD FUNCTION

To model real behavior associated with music streaming services, we associate common actions taken by users with a reward. While listening to music on any platform, users have the ability to like, dislike, skip, or add the song being played. For this reason, we have a set of reward probabilities for each arm shown in Section 3.

Action	"Downloaded"	"Add to Playlist"	"Liked"	"No Response"	"Skipped"	"Disliked"
Reward	+10	+5	+3	0	-2	-5

Table 1. User feedback and rewards

For every song selected by the agent, we update a set of q-values which stores the actions taken with the rewards received. This is especially helpful during the exploration phase, to establish the actions that will likely yield the greatest results. The q-values updates are given that for each arm, the q-value of that arm is the arm selection count divided by the total accumulated rewards for that arm.

	t = 1	t = 2	t = 3	t = 4	t = 5	t = 6
Arm Selection	1	3	2	1	4	1
User Feedback	"Like" = +3	"Dislike" = -5	"Skipped" = -2	"Downloaded" = +10	"Add to Playlist" = +5	"Like" = +3
Song Count	(1, 0, 0, 0)	(1, 0, 1, 0)	(1, 1, 1, 0)	(2, 1, 1, 0)	(2, 1, 1, 1)	(3, 1, 1, 1)
Q-Values	(3, 0, 0, 0)	(3, 0, -5, 0)	(3, -2, -5, 0)	(6.5, -2, -5, 0)	(6.5, -2, -5, 5)	(5.33, -2, -5, 5)
Accumulated Rewards	3	-2	-4	6	11	14

Table 2. Example of q-values update

For the ϵ-greedy policy, the q-values are ignored for action selection when epsilon determines the agent must explore over exploit. Otherwise, it selects the argmax of the q-values. For the example given in Table 2, if the epsilon indicates the agent should exploit, the agent will select the first arm and continue updating the q-values.

2.5 PROJECT OVERVIEW

In addition to the ϵ-greedy strategy for song selection, we also implement a randomized agent that mimics the same behavior expected from slate-based recommendations. While ϵ-greedy strategically balances exploring song choices and exploiting known user preferences (Barto and Sutton, 2018), the randomized agent takes no regard for recording q-values or the reward function and instead chooses a random arm.

We test both strategies to determine which yields greater accumulated rewards:

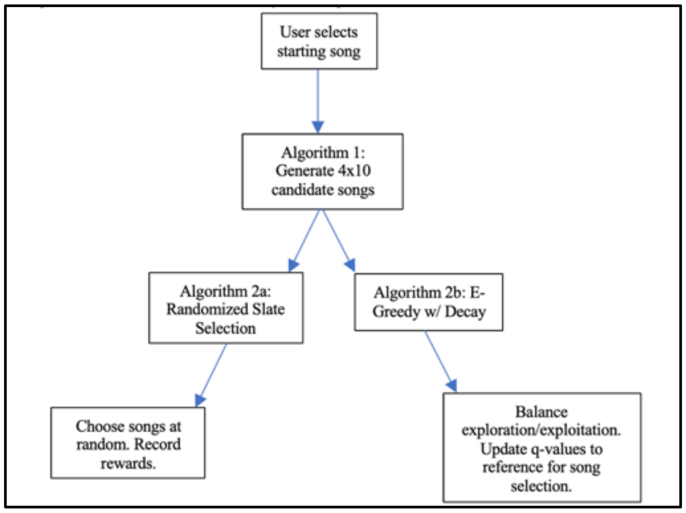

User selects starting song
Algorithm 1: Generate 4x10 candidate songs
Algorithm 2a: Randomized Slate Selection
Algorithm 2b: E-Greedy w/ Decay
Choose songs at random. Record rewards.
Balance exploration/exploitation. Update q-values to reference for song selection.

Figure 8. SpotifyDJ flowchart

3. SIMULATION DESIGN – CREATING USERS

In order to effectively evaluate a music recommender such as SpotifyDJ, it would require a large sample of listeners and also require a significant amount of time invested. Testing the performance of SpotifyDJ cannot be done with a handful of songs or over a few listening sessions. To overcome these challenges, this project uses two separate python classes: the SpotifyDJ class and a users class. The users class contains sample objects of simulated user behavior to provide feedback based on their music feedback tendencies.

For each user created, a learning function is executed which presents the user with different songs to record its feedback. To clarify, this function does not learn the user's specific music preferences, but rather its reward probability distribution per arm. After the reward probabilities are learned, an initial epsilon value is assigned which will get adjusted by the ϵ-greedy agent.

Feedback	**Arm 1**	**Arm 2**	**Arm 3**	**Arm 4**
"Downloaded" = + 10	0.2833	0	0	0.095
"Added" = +5	0.1583	0	0	0.1833
"Liked" = +3	0.4750	0.0333	0	0.245
"No Response" = 0	0.0833	0.3667	0.3067	0.2766
"Skipped" = -2	0	0.4167	0.3033	0
"Disliked" = -5	0	0.1833	0.3900	0

Table 3. Example of learned reward probabilities of simulated user

Table 3 shows an example of a single user's reward probability distribution per arm selection. Based on the probabilities showing some favoritism towards arms 1 and 4, its starting epsilon value is 0.5. In a given simulation, SpotifyDJ would select an arm and the corresponding arm's reward distribution is referenced to provide some reward. In this example, if arm 1 is selected, the most likely feedback given is that the user "liked" the song at a rate of approximately 47%. However, to include randomness, the algorithm implements a small chance that user will provide a negative feedback despite choosing an arm with high positive reward probabilities.

For this project, we start with an initial sample population of 50 users each with their own unique reward probability distribution and classified into one of three epsilon starting classes. For reference, epsilon values' equivalence are as follows: 0.25 – user has a very low tendency of music exploration, reacts poorly to music presented outside of their music preference boundaries, 0.5 – user sometimes reacts positively to music exploration, 0.75 – user has a very diverse taste in music and has the best chance of reacting positively to music exploration.

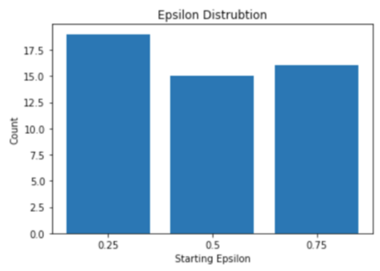


Figure 9. Epsilon distribution of created sample population

4. EVALUATION

As stated in Section 2.3, each SpotifyDJ session is constructed into a 4x10 set of candidate songs resulting in 10 songs played total. For the simulation, we execute a total of 1,000 songs or 100 total sessions to observe the accumulation of rewards using both the random agent as well as the ϵ-greedy agent underlying the decision-making behavior of SpotifyDJ. First, we observe the performance of single users prior to running the simulation on the entire sample used in this project.

4.1 EVALUATION ON LOW EPSILON USER

First, we observe the performances of the random agent vs ϵ-greedy agent's performances on the first 10 songs for a user with epsilon scores of 0.25, 0.5, 0.75.

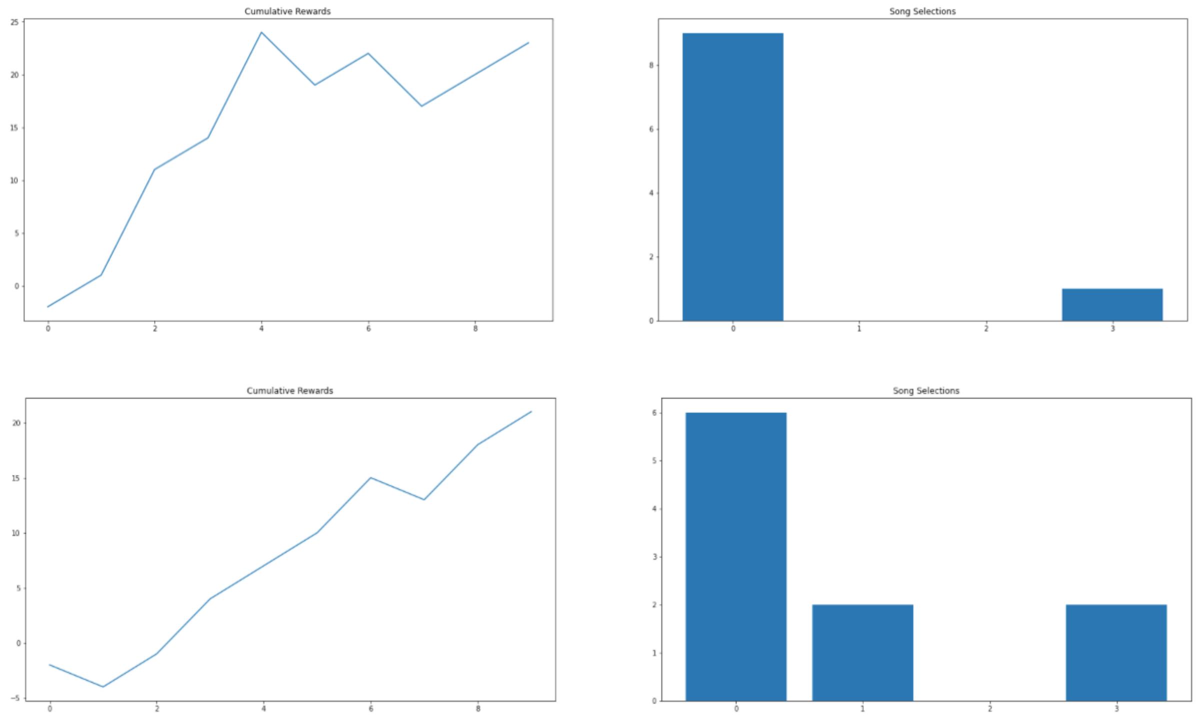
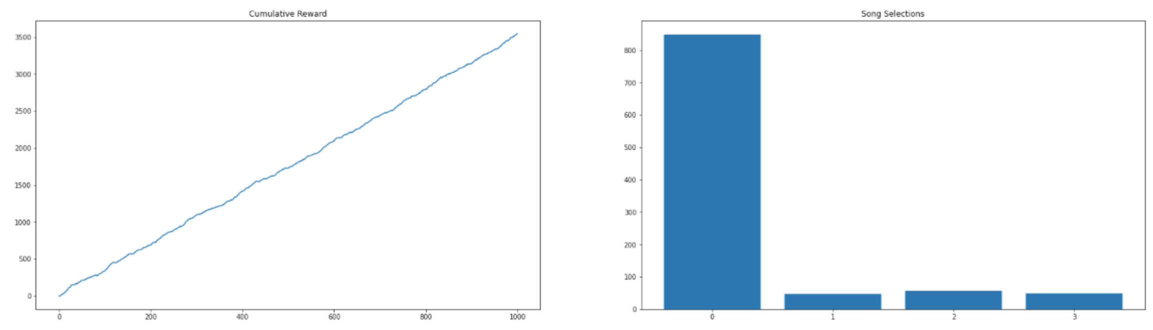
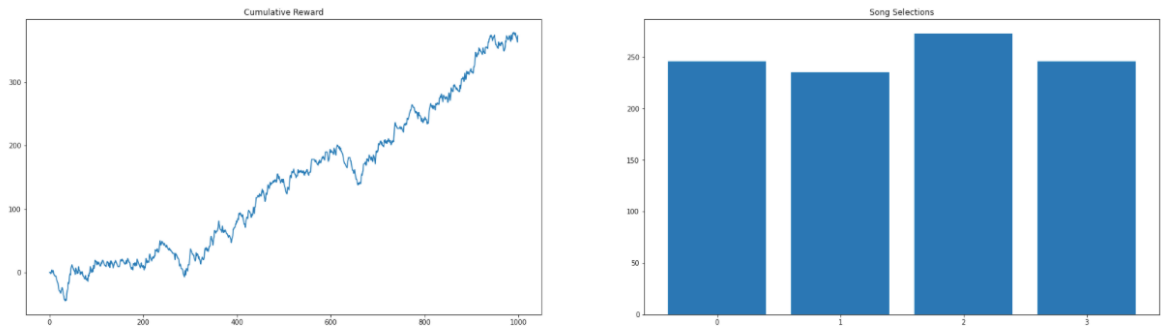


Figure 10. ϵ-greedy agent (top) vs. random agent (bottom) for user with epsilon 0.25 after 10 songs

Upon executing both a random agent and an ϵ-greedy agent for an epsilon score of 0.25, initial observations are that both algorithms seemed to accumulate similar amounts of rewards, but this comes at a cost of a lack of exploration. Assuming this trajectory continues, an ϵ-greedy agent may prematurely conclude a suboptimal policy. To evaluate this model, we observe its performance for a much longer simulation.



```
Total rewards after 1000 songs: 3545.0
```



```
Total rewards after 1000 songs: 373.0
```

Figure 11. ϵ-greedy agent (top) vs. random agent (bottom) after 1000 songs simulated

On a much larger scale, we observe that the ϵ-greedy agent outperforms a random agent, continuously accumulating positive rewards. This can be seen in its song selection counts which clearly shows that after a period of exploration and exploitation, it does converge on a single arm that has the highest probability of resulting in a positive reward. This makes sense as the user was originally classified as possessing a low epsilon music preference.

To put this result into perspective, we introduce the concept of regret ρ, and in particular the zero-regret strategy. In reinforcement learning, regret is defined as the difference between how many total points could have been accumulated having known the optimal policy minus the actual points accumulated (Barto and Sutton, 2018). We calculate regret by subtracting the accumulated rewards from the total maximum rewards that could have been accumulated if the optimal selection was made for every time *t*. For the ϵ-greedy agent, the regret we calculate is (10 x 1000 songs) – (3545) resulting in 6455. The random agent's regret would be significantly higher. The idea of the zero-regret strategy states that as $t \to \infty$, $\frac{\rho}{t} \to 0$ which indicates that a good policy that continuously accumulates rewards resulting in continuously increasing cumulative rewards will eventually result in a lower regret (Vermorel and Mohri, 2005). In music streaming terms, this can be viewed as being synonymous with long-term user satisfaction.

4.2 EVALUATION ON HIGH EPSILON USER

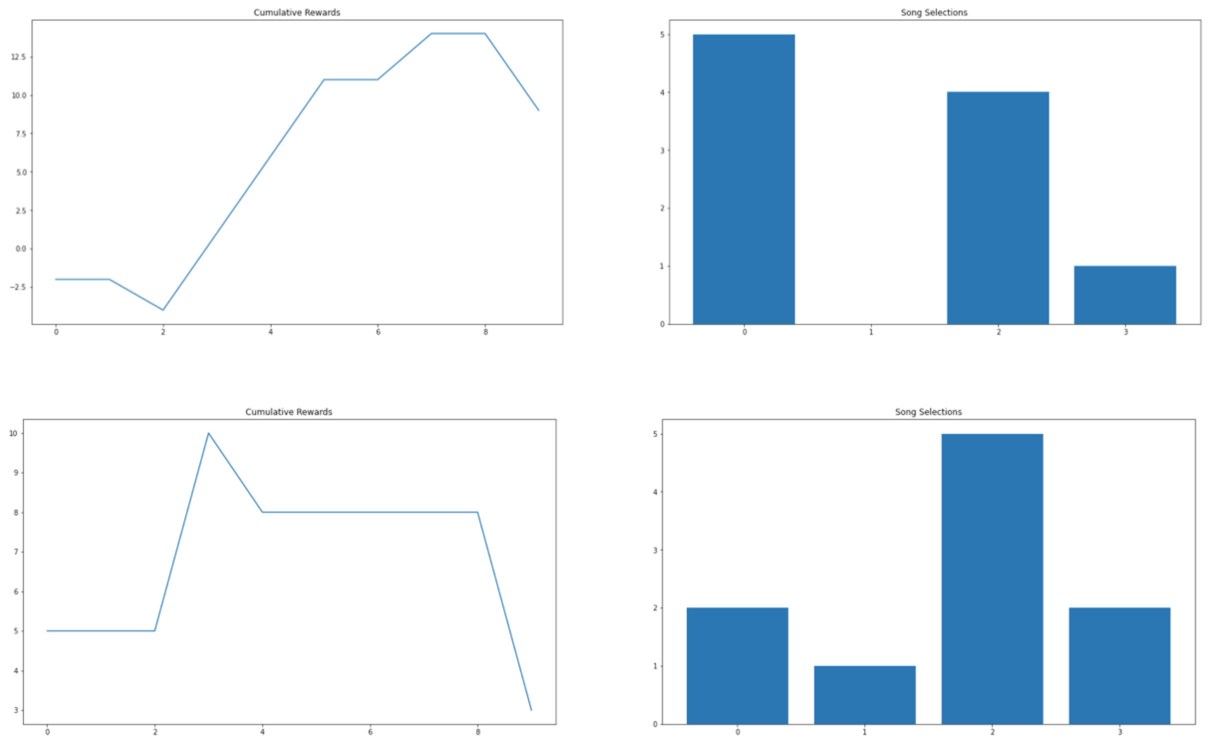
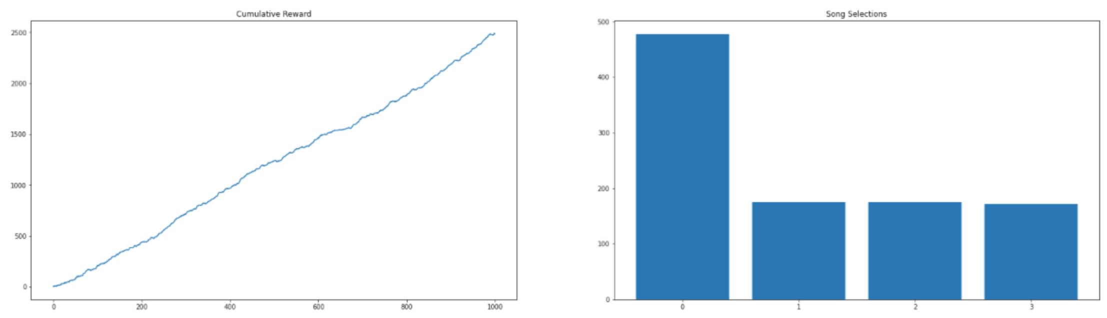
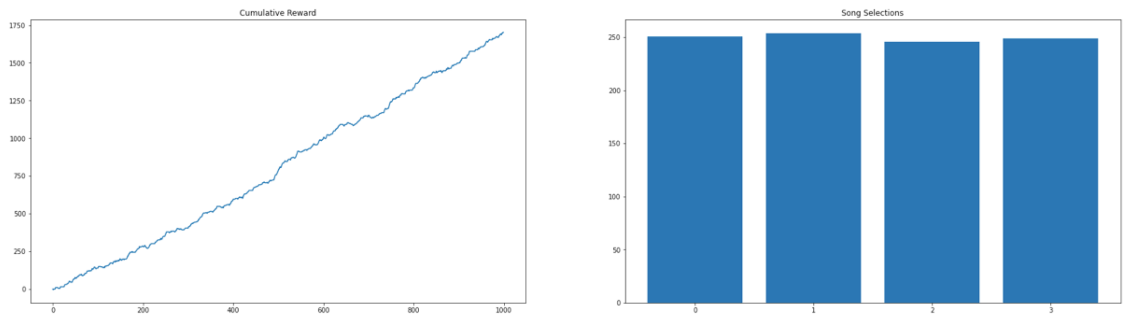


Figure 12. ϵ-greedy agent (top) vs. random agent (bottom) for high epsilon user after 10 songs

As with a user with a low epsilon score, the initial simulations after 10 songs do not indicate much a difference in performance for a high epsilon user. However, we do observe early indications that the ϵ – greedy agent knows the user has a diverse music preference as seen in its song selections. To get a much clearer performance, we execute the simulations for 1000 songs.



```
Total rewards after 1000 songs: 2486.0
```



```
Total rewards after 1000 songs: 1702.0
```

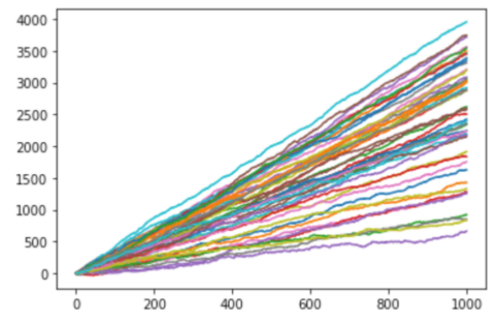
Figure 13. ϵ-greedy agent (top) vs. random agent (bottom) after 1000 songs simulated

After simulating both algorithms for 1000 songs, we still see that the ϵ – greedy agent does outperform the random agent as with the previous simulation with a low epsilon user. However, in this case, the difference in total rewards is not as substantial. This is due to the fact that users with a more diverse taste in music will likely give positive rewards at a higher rate. Random selections are likely to result in positive rewards more often than with a low epsilon user.

4.3 SIMULATION ON ALL USERS

Upon simulating both the random song selection algorithm and the ϵ-greedy algorithm for all 50 users across 1000 songs, we have successfully concluded that the ϵ-greedy algorithm significantly outperforms the random agent. This proves that a great list of recommendation songs (slate recommendations) is insufficient for music recommendations as interaction is key to receiving feedback for algorithm adaptation. Moreover, random displays of recommended songs may actually deter users from continued use of the platform altogether.

```
The average total rewards for all users with an e-greedy agent: 2505.84
```



```
The average total rewards for all users with a random agent: 1007.08
```

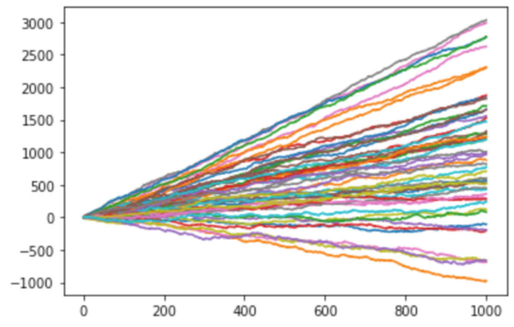


Figure 14. Average total rewards on all users using ϵ – greedy agent (top) vs. random agent (bottom) after 1000 songs

5. FINAL THOUGHTS

Slate-based recommendations commonly found in many entertainment and e-commerce platforms do so fully understanding that interactions with the recommendations themselves are optional. What they accomplish is a sense of personalization that may result in higher rates of consumption whether it may result in higher spending rates or time on the platform indirectly. Music recommendations have to be viewed in a different manner because in the 21st century, music isn't widely purchased anymore, but instead access is granted to an entire catalogue for a subscription.

For this reason, music streaming services should encourage music discovery by introducing highly experimental features. Slate-based music recommendations do not encourage music discovery as they cannot satisfy a specific intent of a user. This project showed that we can identify a more effective way in receiving user feedback on recommendations by enabling a feature that forces feedback.

SpotifyDJ acts as a feature available on any song that is playing on the platform. Once activated, the underlying algorithm takes the current song playing in order to create a candidate list of songs and chooses songs to play, constantly adjusting in trajectory of song choices based on the feedback received by the user. Not only does this feature enable better preference learning, but also enables a secondary bonus feature of being able to let Spotify perform for its users.

This project demonstrated that choosing songs randomly may result in poor performance as a result of a majority of received negative feedback from the user. In reality, this may result in the user discontinuing using the feature or platform altogether. Instead, we introduced the $\boldsymbol{\epsilon}$ – greedy policy, in which SpotifyDJ chooses songs in a strategic manner by balancing exploration and exploitation to continually learning the user's adjusting preferences. Furthermore, by introducing the concept of the zero-regret strategy, we find that the $\boldsymbol{\epsilon}$ – greedy agent continuously improves its performance and may indicate that this feature will result in long-term use from most users.

6. LEARNING POINTS AND FUTURE WORKS

This project was an incredible opportunity to learn a very complex subject such as reinforcement learning. The potential for this subset of machine learning has a significant amount of potential in almost all domains and not just in recommender systems as in this project. Though this project successfully implemented a multi-armed bandit using the $\boldsymbol{\epsilon}$ – greedy policy for song selections, in reality, this is just one of many algorithms available in the realm of reinforcement learning.

Originally, the aim of this project was to implement additional algorithms to use for comparison purposes, but at the moment, to the best of my knowledge, the majority of the available support in reinforcement learning is geared towards training an agent to solve a virtual game of some kind. For this reason, the available python libraries for reinforcement learning are structured to use the OpenAI gym that was not suitable for implementing a recommender system. As a workaround, I instead decided to design an agent and a recommender environment completely from scratch using object-oriented programming.

For the future, as a continuation of this project, I'd like to explore alternative options to implement more complex reinforcement learning strategies to include deep q-learning which implements artificial neural networks. Unfortunately, this project could not do so as the available data was insufficient for executing neural networks. The idea was to use the input layer of the network for raw track features to identify more suitable candidates, but the track features given by Spotify are not in raw form and instead is the result of some analysis performed by Spotify themselves that cannot be unpacked. Still, presumably with better access to technology and music data, it is possible to execute more complex algorithms of reinforcement learning for a feature such as SpotifyDJ.

BIBLIOGRAPHY

Aggarwal, C., Hinneburg, A. and Keim, D., 2001. On the Surprising Behavior of Distance Metrics in High Dimensional Space. *IBM Watson Research Center*. Available at: <https://bib.dbvis.de/uploadedFiles/155.pdf>.

Al-Bakri, N. and Hashim, S., 2018. Reducing Data Sparsity in Recommender Systems. *Journal of Al-Nahrain University*, [online] 21(2). Available at: <https://www.iasj.net/iasj/download/ca46a3f16fea9999>.

AIcrowd | Spotify Million Playlist Dataset Challenge | Challenges. 2022. *AIcrowd | Spotify Million Playlist Dataset Challenge | Challenges*. [online] Available at: <https://www.aicrowd.com/challenges/spotify-million-playlist-dataset-challenge>.

Amer-Yahia, S., Lakshmanan, L., Vassilvitskii, S. and Yu, C., 2009. *Battling Predictability and Overconcentration in Recommender Systems*. [online] Available at: <https://www.researchgate.net/publication/220283328_Battling_Predictability_and_Overconcentration_in_Recommender_Systems> .

Barto, A. and Sutton, R., 2018. *Reinforcement Learning: An Introduction*. MIT Press.

Dean, B., 2022. *Spotify User Stats (Updated Oct 2021)*. [online] Backlinko. Available at: <https://backlinko.com/spotify-users>.

Epsilon.com. 2022. *New Epsilon research indicates 80% of consumers are more likely to make a purchase when brands offer personalized experiences*. [online] Available at: <https://www.epsilon.com/us/about-us/pressroom/new-epsilon-research-indicates-80-of-consumers-are-more-likely-to-make-a-purchase-when-brands-offer-personalized-experiences>.

Ie, E., Jain, V., Wang, J., Narvekar, S., Agarwal, R., Wu, R., Cheng, H.T., Lustman, M., Gatto, V., Covington, P. and McFadden, J., 2019. Reinforcement learning for slate-based recommender systems: A tractable decomposition and practical methodology. *arXiv preprint arXiv:1905.12767*.

Lalmas, M. and Mehrotra, R., 2022. *User Intents and Satisfaction with Slate Recommendations*. [online] Spotify Research. Available at: <https://research.atspotify.com/user-intents-and-satisfaction-with-slate-recommendations/>.

Liebman, E., Saar-Tsechansky, M. and Stone, P., 2015. *DJ-MC: A Reinforcement-Learning Agent for Music PlaylistRecommendation*. [online] AAMAS. Available at: <https://www.researchgate.net/publication/279180021_DJ-MC_A_Reinforcement-Learning_Agent_for_Music_Playlist_Recommendation>.

Madhukar, M., 2014. Challenges & Limitation in Recommender Systems. *International Journal of Latest Trends in Engineering and Technology (IJLTET)*, [online] 4(3). Available at: <https://www.ijltet.org/wp-content/uploads/2014/10/21.pdf>.

Milankovich, M., 2022. *The Cold Start Problem for Recommender Systems*. [online] Medium. Available at: <https://medium.com/@markmilankovich/the-cold-start-problem-for-recommender-systems-89a76505a7>.

Slivkins, A., 2022. Introduction to Multi-Armed Bandits. *Microsoft Research NYC*, [online] Available at: <https://arxiv.org/pdf/1904.07272.pdf>.

Tondji, L., n.d. *Content based filtering vs Collaborative filtering*. [image] Available at: <https://www.themarketingtechnologist.co/building-a-recommendation-engine-for-geeksetting-up-the-prerequisites-13/> .

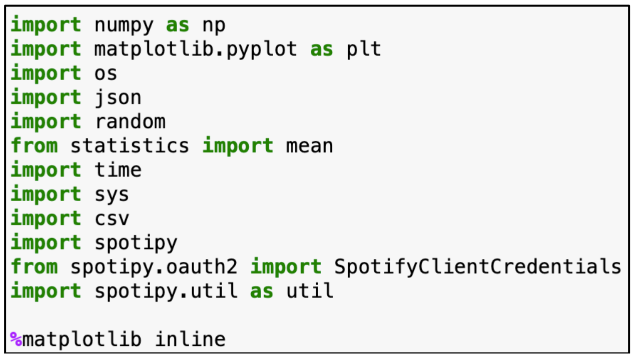
Vermorel, J. and Mohri, M., 2005. Multi-Armed Bandit Algorithms and Empirical Evaluation. *Courant Institute of Mathematical Sciences*, [online]. Available at: <https://cs.nyu.edu/~mohri/pub/bandit.pdf>

Wikipedia. 2022. Spotify. Available at: <https://en.wikipedia.org/wiki/Spotify>

APPENDICES

Appendix A. Python Dependencies Used

This project used python version 3.9 using the Jupyter Notebook IDE.



```
import numpy as np
import matplotlib.pyplot as plt
import os
import json
import random
from statistics import mean
import time
import sys
import csv
import spotipy
from spotipy.oauth2 import SpotifyClientCredentials
import spotipy.util as util

%matplotlib inline
```

Figure 15. Dependencies used

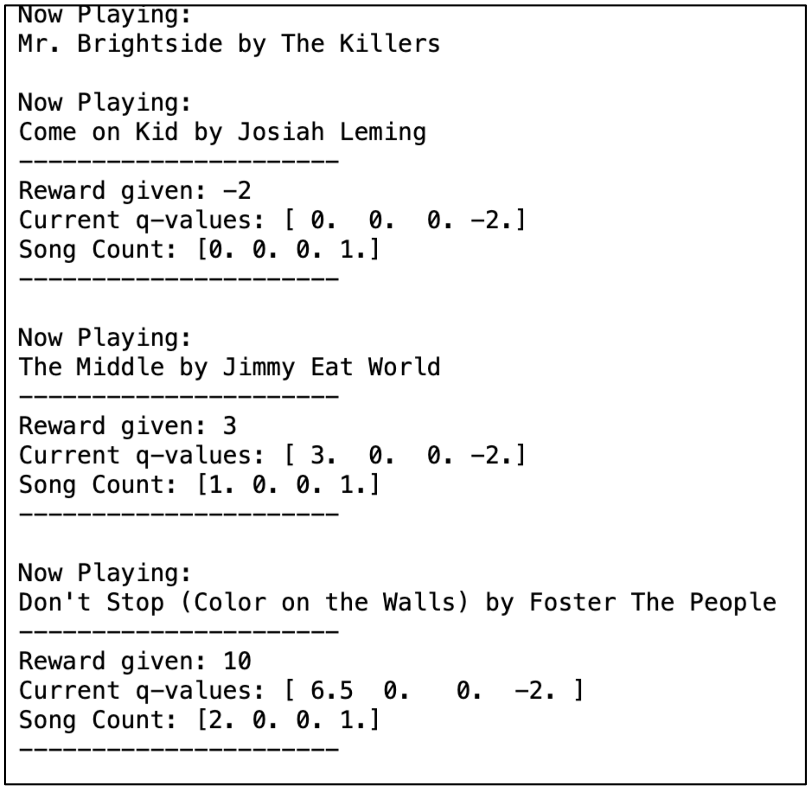
Appendix B. Spotify API

Using the Spotipy library, Spotify provides a range of functions that can be used to search for tracks, artists, albums and also to include track add functions for developers who wish to integrate Spotify into their applications. The official Spotipy documentation can be found at: https://spotipy.readthedocs.io/en/master/

Appendix C. Markov Decision Process

There is a very good reason a significant portion of reinforcement learning research is demonstrated through videos games. This is because in every video game, there is a character or in RL terms, an agent that is placed in some environment. For example, if we place a squirrel inside of a maze, the squirrel itself is the agent and the maze is the environment. For each time *t*, the squirrel observes a certain representation of the environment which is represented by the state it's in and depending on the state, it has a list of possible actions it can take. Through trial and error, the agent learns how to achieve some reward by remembering bad actions and choosing good ones instead.

In this project, SpotifyDJ itself is the agent and for each time *t*, it is shown four possible songs to choose from and upon choosing the song, it receives a reward from the user that it displayed the song to. It doesn't move physically as a squirrel inside of a maze, but instead observes a constantly changing set of songs strategically placed in an indexed order. It learns to choose the best song based on a specific policy that it must adjust continuously. This policy is influenced by q-values, which are expected returns for each possible action.


```
Now Playing:
Mr. Brightside by The Killers

Now Playing:
Come on Kid by Josiah Leming
----------------------
Reward given: -2
Current q-values: [ 0.  0.  0. -2.]
Song Count: [0. 0. 0. 1.]
----------------------

Now Playing:
The Middle by Jimmy Eat World
----------------------
Reward given: 3
Current q-values: [ 3.  0.  0. -2.]
Song Count: [1. 0. 0. 1.]
----------------------

Now Playing:
Don't Stop (Color on the Walls) by Foster The People
----------------------
Reward given: 10
Current q-values: [ 6.5  0.   0.  -2. ]
Song Count: [2. 0. 0. 1.]
----------------------
```

Figure 16. Song selection demonstration with updated q-values based on reward

Appendix D. Evaluation Methodology

To effectively evaluate whether this is a feasible recommender/music playing feature would be almost impossible. There is no way to justify a reasoning for volunteers to give up hours of their time to test a feature, especially without some monetary compensation. For this reason, we create "bots" that mimic a diverse set of behaviors and music preferences.