Automatic Playlist Continuation: Strategies and Insights from the Spotify Dataset

Abstract

The Spotify Million Playlist Dataset Challenge provides a unique opportunity for research in music recommendation systems through the task of automatic playlist continuation. The dataset, comprising 1,000,000 playlists created between 2010 and 2017, includes titles, track metadata, and user-generated content, enabling researchers to explore methods for predicting subsequent tracks in playlists based on given seed information. Our project focuses on leveraging this dataset to implement and evaluate various playlist continuation strategies. Initially, we employed a baseline approach, which recommended the 500 most frequently occurring tracks in the training set for all playlists, achieving an R-Precision score of 0.01306. Then attempting tokenization approaches with Word2Vec, and finally a Deep Sequential Recommendation model focused around learning a latent representation of songs from interaction data. Through this work, we aim to contribute to the broader understanding of playlist recommendation systems, exploring the relationship between user preferences and playlist structures while addressing the challenge of personalizing music discovery at scale.

1 Dataset

The Million Playlist Dataset (MPD) comprises 1,000,000 user-generated playlists from Spotify, collected between January 2010 and October 2017. This dataset serves as a resource for research aimed at enhancing the music listening experience. Each playlist within the dataset is structured as a JSON dictionary of individual tracks and their attributes. The attributes for each playlist included its name, number of followers, number of edits, the last time it was edited, a description (most playlists do not have one), and an array of each track in the playlist. Each individual track entry in the playlist contained information about the track's name, album name, artist name, duration, position in the playlist, and additional metadata such as Spotify URIs.

The dataset provides a detailed look at a large collection of playlists, tracks, albums, and artists. It contains 2,262,292 unique tracks, 734,684 unique albums, and 295,860 unique artists. Notably, there are 92,944 unique track titles and 17,381 unique normalized titles, showcasing significant diversity in song names. Most playlists contain an average of 66 tracks, with the distribution skewed to the right, as shown in Figure 3. Popular playlist titles often reflect broad genres and themes like "chill," "country," and "rap" as seen in Figure 1. Top tracks include "HUMBLE." by Kendrick Lamar, "One Dance" by Drake, and "Broccoli" by DRAM, highlighting the prominence of hip-hop and pop music. Leading artists include Drake, Kanye West, and Kendrick Lamar, with Drake appearing most frequently (847,160 times). Playlist edits vary, with most showing between 2 and 21 edits, and most playlists have a small following, averaging 1 to 3 followers, indicating limited engagement despite the high volume of playlists. Additionally, playlists with longer total durations tend to have more edits on average, as shown in Figure 2, suggesting that longer playlists are interacted with more frequently.

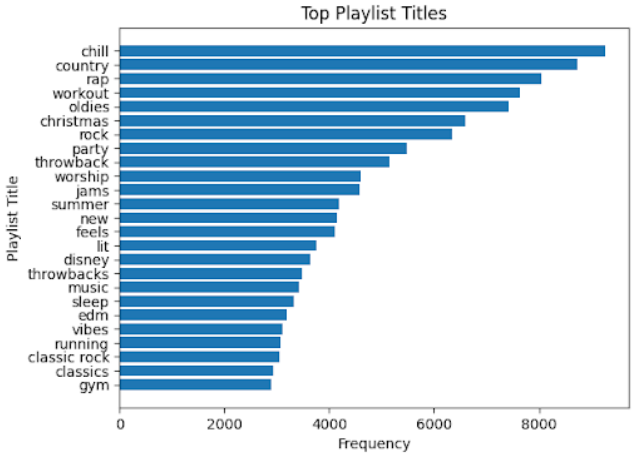

Figure 1: Top playlist titles by frequency

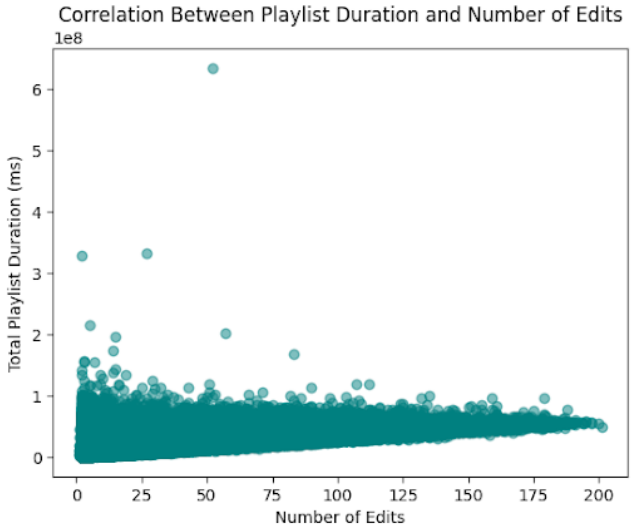

Figure 2: Playlist Duration vs Edit Count

An interesting breakdown of playlist data by follower count reveals that playlists with fewer than 100 followers make up the vast majority, totaling 999,178 playlists, and they contain 66,276,984 tracks, with an average length of 66 tracks per playlist. Only 1.87% of these playlists have descriptions. In contrast, playlists with 100 or more followers are fewer (822 playlists) but are longer on average, with 84 tracks each. These playlists also show higher engagement, with 14.23% including descriptions. This data highlights that while smaller playlists dominate the dataset, those with higher follower

counts tend to be more detailed and curated, potentially indicating higher user interest and interaction.

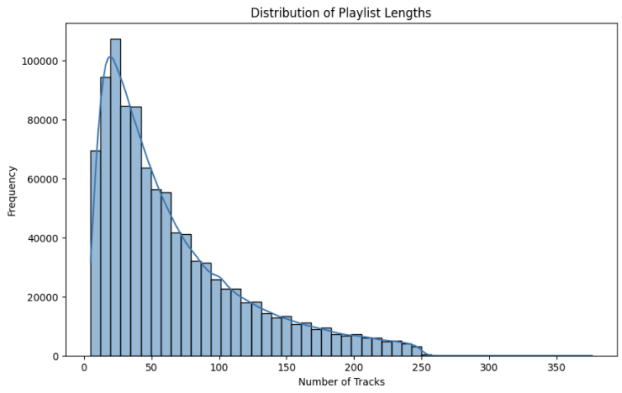

Figure 3: Playlist Length vs Frequency

A separate challenge dataset was used to evaluate the effectiveness of the developed algorithms. This dataset comprised 10,000 incomplete playlists and divided into 10 scenarios, each containing 1,000 playlists: (1) title only, no tracks, (2) title and first track, (3) title and first 5 tracks, (4) no title and first 5 tracks, (5) title and first 10 tracks, (6) no title and first 10 tracks, (7) title and first 25 tracks, (8) title and 25 random tracks, (9) title and first 100 tracks, and (10) title and 100 random tracks. The goal of the challenge dataset is to predict the missing tracks for each playlist and submit lists of 500 ordered predictions.

2 Predictive Task

The predictive task we studied was to predict 500 tracks for playlists in the validation set through playlist continuation models and determine whether there was a relationship between general playlist information and tracks included in playlists. To provide a starting point for evaluating the effectiveness of such models, a straightforward baseline approach was implemented. This method involved identifying the 500 tracks that appeared most frequently in the training set and recommending these same tracks uniformly for all playlists in the challenge set, regardless of playlist-specific features such as title or seed tracks. Our main metric of interest is R-Precision, specified as:

$$\text{R-precision} = \frac{|S_T \cap G_T| + 0.25 \cdot |S_A \cap G_A|}{|G_T|} \tag{1}$$

Where G_T and G_A are the set of unique track IDs and artist IDs in the ground truth, and S_T is the subset of tracks IDs in the top-$|G_T|$ tracks recommended in the submitted playlist, and S_A be the set of unique artist IDs in the same set.[2]

While simplistic and devoid of any personalization or contextual understanding, this baseline offered a useful reference for assessing improvements made by more advanced algorithms. The results of this approach yielded an R-Precision score of 0.01306, indicating that, on average, only 1.31% of the top recommended tracks matched the actual withheld tracks from the test playlists. This low score underscores the limitations of the baseline model and emphasizes the need for more sophisticated techniques to improve recommendation accuracy. We utilized this baseline R-Precision score as a measurement of validity regarding the techniques we employed, with a higher value implying a better approach was taken.

3 Model Specifications

Our first model attempt to use was a Word2Vec model that recommended tracks based on the similarity between playlist titles. This approach involved training the model on the titles of playlists to capture the semantic relationships between the words within them. By leveraging these relationships, we aimed to find playlists with similar titles and suggest tracks from these playlists as recommendations.

The process began by training the Word2Vec model on 80,000 randomly selected playlists from our dataset. The model generated vector representations for each word in the titles, allowing us to compute similarities between different playlist titles. For example, our model determined that rap was semantically close to hop and 90's. When given a title from the validation set, the model identified other titles in the training set that were most similar in meaning or context. We used these similar titles to compile a list of recommended tracks. After making the recommendations, we evaluated the performance using the R-Precision metric. We attempted to extend the Word2Vec approach by augmenting the text corpus of the title by mapping song ids to integer indices in the corpus, and training W2V on the sequence of text tokens + song tokens, however we were not able to complete this in time.

In addition to looking to leverage a more classical Natural Language Processing approach with the Word2Vec model, we also aimed to leverage some of the more recent deep learning advancements. One inspiration that we drew from recent class lectures, was that of learning an embedding metric and some heuristic on the embedded metrics space. Similarly to the paper, we modeled the embedding space by only using the interactions between songs, as well as the sequence that they appeared in each playlist.[3]

Our approach to learning the embedding space however is an Auto Encoder approach. There is a similar idea in a related paper called AutoSeqRec, however our model focuses on item only interactions.[5] In our setup, we created an asymmetric item-to-item interaction matrix, where we logged the interactions between every song along with the top 20, 000 songs from the entire dataset. We do this for two reasons, computability, and efficiency. By seeing how songs interact with the most popular songs we can learn overarching playlist behavior.

The approach is specifically done by first counting how many playlists each song is on, to find the most prevalent songs in the catalog. Then we create a $C \times K$ dimensional binary matrix, $\mathcal{M}$ that keeps track of the playlist interactions between songs.

Here, C is the size of the song catalog, and K is the number of most popular songs that we will model relationships with. Here we set $K = 20000$, so the song-song interaction matrix is defined as:

$$m_{i,j} = \begin{cases} 0 & \nexists P \text{ such that } s_i \in \mathcal{P} \text{ and } s_j \in \mathcal{P} \\ 1 & \exists \mathcal{P} \text{ such that } s_i \in P \text{ and } s_j \in \mathcal{P} \end{cases} \tag{2}$$

Where s_i is a song $s \in \mathcal{S}$ and $\mathcal{S}$ is the catalog of songs, and $P \in \mathcal{P}$ is a playlist in the set of playlists.

Each row corresponds to a song in the catalog, and each column to a song in the top K songs (in our case, 20000). The (i,j) element is 1 when song i is on the same playlist as song j. In this way, we can succinctly encode the global relationships between songs. Through a graph theory lens, we have created an adjacency matrix which defines the graph of relationships between songs. The autoencoder aims to reconstruct the song-song interaction vector for each row of the matrix, such that the latent space organizes into a semantic space that encodes the adjacency matrix.

This approach might struggle for any playlists that have very obscure music that don't have any relationships with top songs. The Auto Encoder was trained with an Mean Squared Error (MSE) reconstruction loss, since we want the reconstructed interactions to be similar to each other.

In addition to our Auto Encoder, we built a sequence deep learning model that aimed to model the sequence of songs in the playlist using the embeddings. This model is a Convolutional Neural Network. The setup here was since we created a 100 dimensional latent space for each song, so we can model this sequence as a signal with 100 channels. We also introduced padding and truncation, and looked at a sequence of the last 100 songs of each playlist.

The training objective for this model was to output a vector prediction of the next song given the sequence. This was trained with the Cosine Embedding loss, so that we could optimize the model to output a point as close as possible. It is shown that this loss works particularly well for CNNs.[1]

Cosine Embedding Loss is defined as:

$$\text{loss}(\mathbf{x}, y) = \begin{cases} 1 - \cos(x_1, x_2) & \text{if } y = 1 \\ \max(0, \cos(x1, x2) - \text{margin}) & \text{if } y = -1 \end{cases} \tag{3}$$

Where y is used to indicate whether the cosine similarity should be maximized or minimized, and the margin is a number in the range $[-1, 1]$ and is defaulted to 0.5.

With this, we have a model that can predict a point in space given an input sequence of songs. From there, we found the top 500 nearest neighbors with FAISS[4] and mapped it back to the points for us to calculate our R-Precision on the withheld songs from the dataset.

4 Related Work

The dataset we chose was originally ran as an AI competition by Spotify, as mentioned in [Chen], where there were two sets of model submissions, one with external data, and one using only the playlist and interaction data. The best team in the challenge achieved an R-precision of 0.2241. There doesn't seem to be any information or descriptions on what model approach the top teams took however.

The playlist continuation problem in general is a very interesting problem with many approaches at the core of recommender systems as a whole. In class we went over the Metric Embedding approach, which inspired our autoencoder model, but additionally, new sequential recommendation techniques like SASRec and Bert4Rec are poised to provide additional performance here. In a survey of the problem by Bonnin and Janach, the state of the art has been leveraging "background knowledge" that can be learned from things like audio signal (the music itself), ratings, tags, and social graphs.

In our case, because we did not use external data, we couldn't leverage this technique, so we opted for Deep Sequential models that we haven't yet seen applied to this problem yet. The approach in Exploiting Sequential Music Preferences via Optimisation-Based Sequencing is a particularly interesting state of the art model, where they optimize a sequence model from interaction data by maximizing "total consumption".

Given the amount of time we had to implement our models, we went with a Sequential Model that would be simpler to implement and train than a full Transformer, or a complex algorithm like the metric embeddings.

Table 1: Comparison of R-Precisions

Model	R-Precision
Naive Popularity	0.01306
Word2Vec	0.000543
AutoEnc + CNN	0.1324
Best Model	0.2241

5 Results

Our W2V approach over the text of our titles, showed an R-Precision score of 0.000543, indicating that only about 0.0543% of the top recommended tracks matched the actual withheld tracks in the test playlists. This score was actually worse than the baseline, highlighting significant limitations in using only playlist titles as the basis for recommendations. The poor R-Precision score indicated that while Word2Vec was able to capture some semantic similarities between words, it was not sufficient for accurately predicting relevant tracks. This underscores the need for more advanced approaches that integrate additional data, such as track features, user behavior, or contextual information, to improve the quality and accuracy of the recommendations. In addition to

The Auto Encoder + CNN model performed decently with an R-Precision of 0.1324, which beat out our Word2Vec approach, and the baseline, however not outperforming the best model from the original Spotify challenge. Despite the lesser performance, we feel that there are many approaches to improve our model given the time. One important note is the approach to training the Auto Encoder. One issue is that our Auto Encoder is pretty basic compared to the Auto Encoders that are dominating in the State of the art models, like Variational Auto Encoders, or Vector Quantized Variational Auto Encoders. These techniques would have benefited us with better reconstruction, as well as a better organized latent space with the potential for more semantic meanings. One thing we would have liked to do was also compare our learned representations to representations created from the Metric Embedding paper. An aggregated table is shown below:

Additionally, the CNN that used the Latent Representations as input features, could have been much deeper. Most successful CNN's are deep with many repeated blocks of convolutions. Ours is only one Convolutional Module. This model may be the most problematic aspect of our results. Given more time, we would have wanted

to implement a Self-Attention based model like that from SASRec or Bert4Rec. We feel that this would have done a better job of explicitly learning about the sequence, since these models learn Sequence Embeddings. When comparing our somewhat lukewarm results with the excellent results from AutoSeqRec, it seems that without a good representation of sequence in addition to our item-interactions, that we might be losing out on useful features.

Acknowledgments

We would like to express our gratitude to the researchers at Spotify who developed the Million Playlist Dataset, which has been invaluable for advancing research in music recommendation systems. Special thanks to the following individuals for their contributions: Cedric De Boom, Paul Lamere, Ching-Wei Chen, Ben Carterette, Christophe Charbuillet, Jean Garcia-Gathright, James Kirk, James McInerney, Vidhya Murali, Hugh Rawlinson, Sravana Reddy, Marc Romejin, Romain Yon, and Yu Zhao.

Additionally, we extend our appreciation to the organizers of the RecSys Challenge 2018 for their efforts in fostering collaboration and innovation in the field: Ching-Wei Chen (Spotify, New York, USA), Markus Schedl (Johannes Kepler University, Linz, Austria), Hamed Zamani (University of Massachusetts Amherst, MA, USA), and Paul Lamere (Spotify, USA). Their work has significantly contributed to the development of cutting-edge music recommendation technologies.

Finally, we are deeply thankful to our professor, Julian McAuley, for guiding and inspiring us throughout the Web Mining and Recommender Systems course. His expertise and mentorship have been instrumental in shaping our understanding of Recommender Systems and their practical applications.

References

[1] Björn Barz and Joachim Denzler. 2019. Deep Learning on Small Datasets without Pre-Training using Cosine Loss. *CoRR* abs/1901.09054 (2019). arXiv:1901.09054 http://arxiv.org/abs/1901.09054

[2] Ching-Wei Chen, Paul Lamere, Markus Schedl, and Hamed Zamani. 2018. Recsys challenge 2018: automatic music playlist continuation. In *Proceedings of the 12th ACM Conference on Recommender Systems* (Vancouver, British Columbia, Canada) *(RecSys '18)*. Association for Computing Machinery, New York, NY, USA, 527–528. https://doi.org/10.1145/3240323.3240342

[3] Shuo Chen, Josh L. Moore, Douglas Turnbull, and Thorsten Joachims. 2012. Playlist prediction via metric embedding. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (Beijing, China) *(KDD '12)*. Association for Computing Machinery, New York, NY, USA, 714–722. https://doi.org/10.1145/2339530.2339643

[4] Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2019. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data* 7, 3 (2019), 535–547.

[5] Sijia Liu, Jiahao Liu, Hansu Gu, Dongsheng Li, Tun Lu, Peng Zhang, and Ning Gu. 2023. AutoSeqRec: Autoencoder for Efficient Sequential Recommendation. arXiv:2308.06878 [cs.IR] https://arxiv.org/abs/2308.06878