Song Recommender System Using Graph Neural Network on Spotify Million Playlist Dataset

Robel Tirfe
Department of Computer Science, George Mason University
rtirfe@gmu.edu

December 11, 2025

1 Introduction

Nowadays most people uses music streaming services like Apple Music and Spotify, but that was not always the case – radio and paid download services (e.g., iTunes) once dominated the industry. Given the huge music libraries within these streaming services, they utilizes different recommender systems to tailor contents to end users. In this project, I will build graph based playlist-track recommender.

2 Problem Definition

The task addressed in this project is building a model for automatic playlist continuation. In other words, given a partially observed Spotify playlist contains a set of seed tracks, the objective is to predict what additional tracks should be recommended to the user in a way that these tracks keep the playlist's musical mood intact. This problem is central to music recommendation systems, where users expect suggestions that naturally extend their curated music collections.

The playlist continuation problem can be easily formulated as a link prediction task on a bipartite graph. This graph will consist of two sets of disjoint nodes: playlist and track nodes. An edge will exist between a playlist and the track node if and only if the given track is part of the playlist. Once the bipartite graph is generated, the next objective will be to produce low-dimensional embeddings for both the playlist and track nodes. The learning goal here is that these embeddings capture the graphical relationship between playlists and tracks so that a playlist and its corresponding tracks will be positioned nearby in the learned embedding space.

Once we learn playlists and tracks embeddings, then the playlist continuation problem can be expressed as: for a target playlist, compute similarity scores between the playlist embedding and the embeddings of all candidate tracks. Then next, rank tracks by their corresponding scores, and the top-k-ranked tracks will be a recommendation for the given playlist.

The inspiration for this project comes from the work of Eva Batelaan et al., who model the data as a bipartite playlist–track graph and experiment with several GNN architectures (LightGCN, GraphSAGE, and GAT) under a transductive link prediction split. We adopt their bipartite graph construction and data-splitting strategy as our baseline [1].

Moreover, the dataset we will use for this project is the Spotify Millioin Playslist Dataset (MPD), which was released by Spotify in 2018 for AIcrowd challenge for music recommendation research [2].

3 Methodology

Our proposed method is based on a Graph Neural Network (GNN) architecture, specifically GraphSAGE, to learn playlist and track embeddings from a bipartite graph. GNNs provide a general framework for defining deep neural networks on graph data, where node representations depend explicitly on the graph structure as well as available node features. In out setting, this means playlist and track embeddings are shaped both by co-occurrence patterns and by tracks-level audio features. Additionally, the inductive nature of GraphSAGE also allows us to generate representations for new playlists at test time using their seed tracks, making it an optimal fit for our playlist continuation problem.

Track nodes are initialized with 11-dimensional audio feature vectors(e.g., energy, danceability, temp.) For playlist nodes, we construct a feature vector by taking a position-weighted average of auto features of its constituent tracks, where earlier tracks receive higher weights via a linearly decaying coefficient (from 1.0 to 0.3.) These playlist and track features are then used as inputs to GraphSAGE together with the bipartite graph structure.

The core of our model consists of GraphSAGE convolutional layers that iteratively refine node embeddings via neighborhood aggregation and neural message passing, where vector messages are exchanged between nodes and updated using neural networks [5]. Taken together, these GraphSAGE layers act as an encoder that produces low-dimensional embeddings for playlist and track nodes. Given the final node representations from this encoder, a lightweight decoder—implemented as an inner product—is used as the prediction head of our model to compute link scores between playlist–track pairs.

Algorithm 1: Compute playlist feature vector from track audio features

```
Input: Playlist tracks $\{t_1, \ldots, t_n\}$
for $i = 1$ to $n$ do
    $\mathbf{f}_i \leftarrow$ get_track_features($t_i$) ;      // 11-dim audio features
Generate linearly decaying weights $w_1, \ldots, w_n$ from $1.0$ to $0.3$;
Normalize weights: $w_i \leftarrow w_i / \sum_j w_j$;
Compute playlist feature:
    $$\mathbf{x}_{\text{playlist}} \leftarrow \sum_{i=1}^{n} w_i \mathbf{f}_i$$
```

4 Experiments

4.1 Baseline Experiment

Our baseline experiment uses a 3-layer GraphSAGE model with 64-dimensional embeddings, trained using Bayesian Personalized Ranking (BPR) loss with negative sampling. We use Recall@500 as our primary evaluation metric, measuring how many of a playlist's held-out tracks are recovered within its top-K recommended tracks (with $K = 500$). The model is trained for 600 epochs with the Adam optimizer, using a learning rate of 0.01 and weight decay of 10^{-5}. We do not perform k-fold cross-validation; instead, we rely on a single fixed train/validation/test split as described below.

Following Batelaan et al. [1], we adopt a transductive link prediction setup in which the full graph structure is visible during training, but a subset of edges is held out for prediction at each stage. Edges that remain available to the GNN for neighborhood aggregation are referred to as *message passing edges*, while edges that are masked and used only for loss computation and evaluation are referred to as *supervision edges*. We split the supervision edges into 70%, 15%, and 15% for training, validation, and test sets, respectively. Table 1 summarizes the resulting edge counts for each split.

Table 1: Dataset split statistics

Split	**Supervision Edges**	**Message Passing Edges**
Train	553,709	1,107,418
Validation	118,651	1,107,418
Test	118,651	1,344,720

	Baseline	**Exp 1**	**Exp 2**	**Exp 3**
Train Loss	0.162	0.248	0.151	0.138
Val Loss	0.179	0.243	0.133	0.139
Val Recall	0.621	0.504	0.686	**0.737**
Test Loss	0.177	0.253	0.131	0.137
Test Recall	0.626	0.512	0.689	**0.741**

Table 2: Experiment Results

As shown in Table 2 above, the GraphSAGE baseline converges to a low training loss (0.162) and a similar validation loss (0.179), suggesting limited overfitting. The validation Recall@500 of 0.621 indicates that the model recovers roughly 62% of the held-out tracks within the top-500 recommendations. Test performance is very close (loss 0.177, Recall@500 = 0.626), showing that the model generalizes well to unseen supervision edges. Together, these results provide a strong reference point for evaluating our subsequent experimental variants.

4.2 Experiment 1: Adding Node Features

ID embeddings: In the baseline model, each playlist and each track node is assigned a unique integer ID. We then use a learnable embedding layer, `Embedding(num_nodes, embedding_dim)`, that maps each node ID to a dense d-dimensional vector. These ID embeddings are randomly initialized and optimized during training, allowing the model to capture collaborative information about playlist–track co-occurrence even in the absence of explicit features. In the baseline, these ID embeddings serve as the sole node representations.

In Experiment 1, we keep all training and architectural settings fixed but replace these ID embeddings with audio-level node features for playlists and tracks. This variant performs notably worse than the baseline, with Val Recall@500 dropping to 0.504 and Test Recall@500 to 0.512 (vs. 0.621/0.626 for the baseline), and a higher test BPR loss of 0.254. A plausible explanation is that the audio features introduce additional noise or misalignment with the collaborative signal, and that removing the powerful ID embeddings reduces the model's capacity to capture fine-grained playlist–track interactions.

We also observed that configuring the first GraphSAGE layer with an input dimension of 11 (to directly match the raw audio feature size) and an output dimension of 64 led to the worst performance among our variants. To mitigate this, we introduced a linear projection layer that maps the 11-dimensional audio features into a 64-dimensional space before feeding them into GraphSAGE. This modification yielded somewhat improved performance but still lagged behind the baseline, reinforcing the conclusion that audio features alone are not sufficient to match the strength of collaboratively learned node embeddings.

4.3 Experiment 2: Hybrid ID + Feature Embeddings

In this experiment, we address the limitations of Experiment 1 by reintroducing the learnable ID embeddings defined above while still incorporating playlist and track audio-level features. Concretely, each node representation now combines its ID embedding with its audio-based feature vector to form a richer input for GraphSAGE.

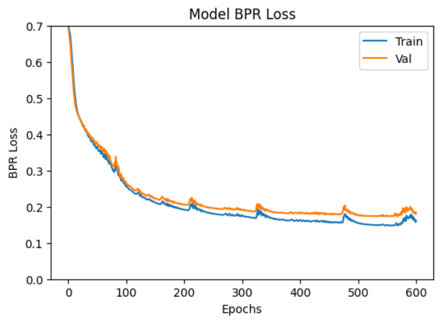
The hybrid model achieves a Val Recall@500 of 0.686 and a Test Recall@500 of 0.690, clearly outperforming both the baseline (0.621 / 0.626) and the feature-only variant. It also attains a lower test BPR loss of 0.132, indicating a better fit to the pairwise ranking objective. These results suggest that audio features are most effective when used to *complement*, rather than replace, learnable ID embeddings.

4.4 Experiment 3 : Multiple Negative Sampling

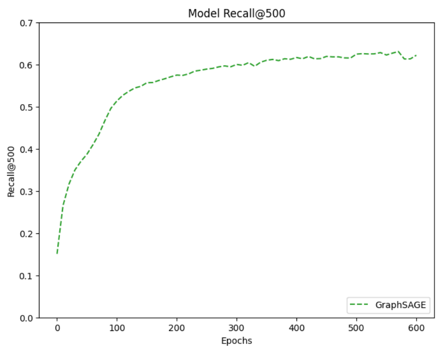
The Bayesian Personalized Ranking (BPR) loss we use provides a notion of what constitutes good and bad recommendations. Rather than predicting absolute ratings, BPR is a pairwise ranking objective where it assumes that, for a given playlist, positive edges should be ranked above negative edges. To compute this loss, we therefore need scores for pairs of positive and negative edges, where a positive edge corresponds to an existing playlist-track combination we want to recover, and a negative edge is a sampled playlist-track pair that does not appear in the graph. The loss then encourages the model to assign higher scores to positive edges than to their negative counterparts.

Sampling multiple negatives per positive edge sharpens this effect: each playlist-track interaction is contrasted against several alternative tracks, exposing the model to a more diverse set of "negative" candidates and yielding a stronger ranking than comparing against a single negative.

We focus on the impact of the number of negative samples used in the BPR loss in our last experiment. We adopt $n = 4$ negative samples per positive playlist-track edge which yields the best performance across all models we tested, achieving a Val Recall@500 of 0.737 and a Test Recall@500 of 0.742. The corresponding test BPR loss of 0.138 indicates a strong fit to the pairwise ranking objective, confirming that using multiple negative samples per positive edge provides a more informative and effective training signal for our GraphSAGE-based recommender.

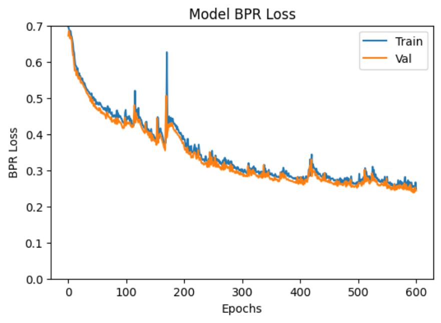


Train/Validation Loss

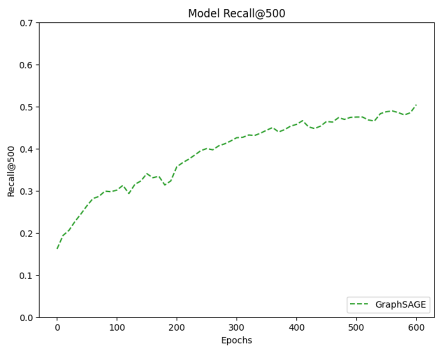


Recall@500

Figure 1: Baseline model loss and recall curves.

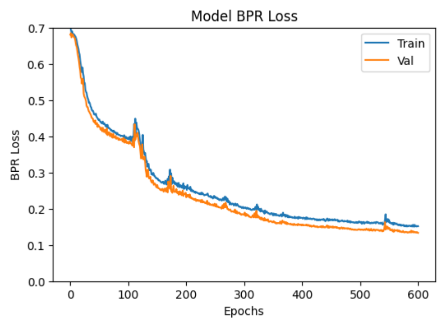


Train/Validation Loss

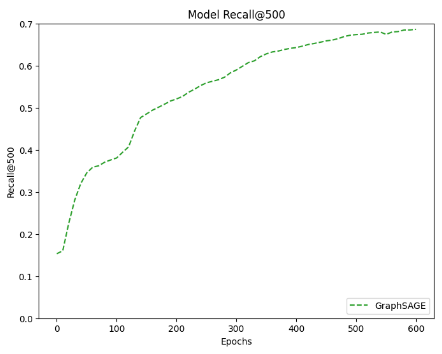


Recall@500

Figure 2: Experiment 1 model loss and recall curves.

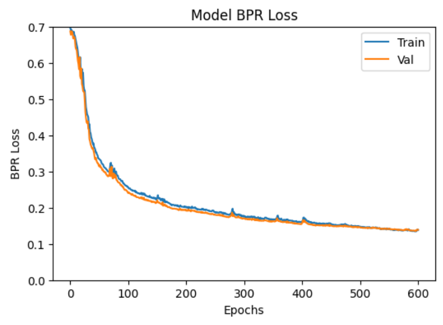


Train/Validation Loss

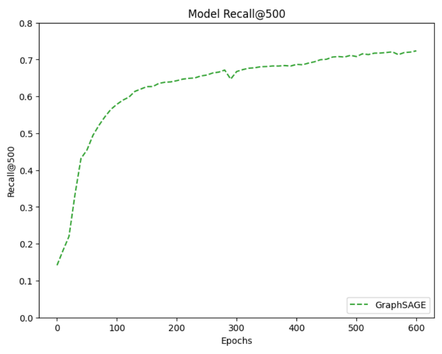


Recall@500

Figure 3: Experiment 2 model loss and recall curves.



Train/Validation Loss



Recall@500

Figure 4: Experiment 3 model loss and recall curves.

References

[1] Batelaan, E., Brink, T.,& Wittenbrink, B. (2023, May 16). Spotify track neural recommender system. Medium, Stanford CS224W: Machine Learning with Graphs. https://medium.com/stanford-cs224w/spotify-track-neural-recommender-system-51d266e31e16

[2] C.W. Chen, P. Lamere, M. Schedl, and H. Zamani. Recsys Challenge 2018: Automatic Music Playlist Continuation. In Proceedings of the 12th ACM Conference on Recommender Systems (RecSys ’18), 2018.

[3] “Web API Reference: Spotify for Developers.” Home, https://developer.spotify.com/documentation/web-api/reference/.

[4] W. Hamilton, R. Ying, and J. Leskovec. (2018). Inductive Representation Learning on Large Graphs. Advances in neural information processing systems, 30.

[5] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. Neural message passing for quantum chemistry. *Proceedings of the 34th International Conference on Machine Learning (ICML)*, 2017.