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 10, 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 62of 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

4.3 Experiment 2

4.4 Experiment 3

5 Dataset: Spotify Million Playlist (MPD)

The dataset I 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]. The dataset contains 1,000,000 playlists, including playlist titles and track titles, created by users on the Spotify platform between January 2010 and October 2017.
The challenge's original evaluation task was automatic playlist continuation: given a seed playlist title and/or initial set of tracks in a playlist, to predict the subsequent tracks in that playlist. My project will also addresses the same task.
For this project, I will model the dataset as a bipartite playlist-track graph: playlists and tracks are nodes, and undirected edge indicates that a track appears in the playlist. To do so, I will only keep playlist title and track name, dropping artist and album information. An example JSON record is shown below.

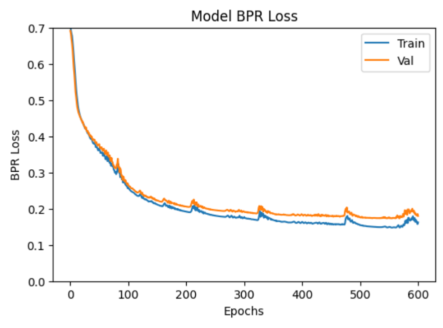
```
{
  "playlist_title": "musical",
  "tracks": [
      { "track_name": "Finalement" },
      { "track_name": "Betty" },
      { "track_name": "Some Beat in My Head" }
    }
  ]
}
```

6 Node Features

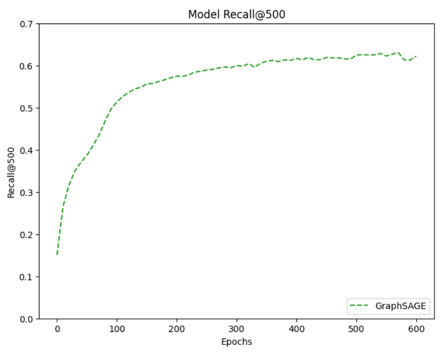
For each track node in the dataset, I will attach a feature vector. Using Spotify's public Web API [3], I will pull per-track attributes to serve as model input:

- Danceability – [float] describes how suitable a track is for dancing
- Acousticness – [float] whether the track is acoustic (0 to 1)
- Energy – [float] a perceptual measure of intensity and activity
- Instrumentalness – [float] Predicts whether a track contains no vocals
- Key – [integer] The key the track is in
- Liveness – [float] Detects the presence of an audience in the recording
- Loudness – [float] The overall loudness of a track in decibels (dB).
- Mode – [integer] indicates the modality (major/1 or minor/0) of a track
- Speechiness – [float] detects the presence of spoken words in a track
- Tempo – [float] the overall estimated tempo of a track in beats/minute
- Valence – [float] the musical positiveness conveyed by a track

Given each playlist is a set of tracks, I define the playlist-node feature vector as the element-wise mean (permutation invariant) of its tracks vector.

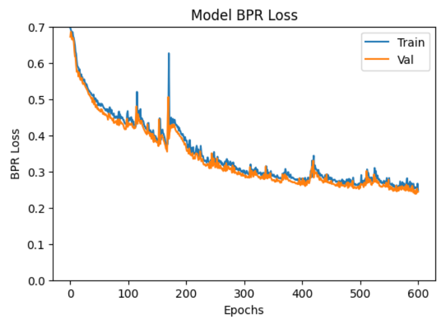


Train/Validation Loss

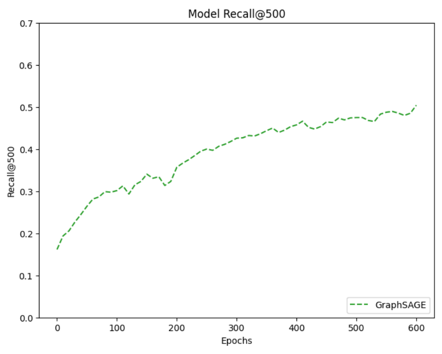


Recall@500

Figure 1: Baseline model loss and recall curves.



Train/Validation Loss

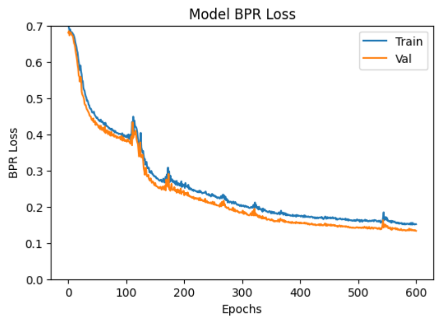


Recall@500

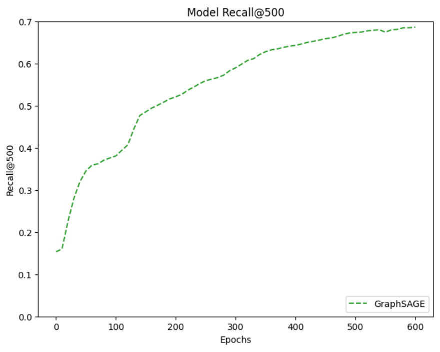
Figure 2: Experiment 1 model loss and recall curves.

7 GraphSAGE

In this project, I will use GraphSAGE, a framework for inductive representation learning on large graphs, to generate low-dimensional embeddings for nodes [3]. GraphSAGE learns an embedding function that aggregates a node's features with sampled neighbors, so it can produce embeddings for new playlists or tracks as the catalog evolves. Which in-return mitigate the cold-start problem with many recommender systems, by producing reasonable embeddings for newly created or sparsely connected playlists-tracks from their features and any available neighbors.

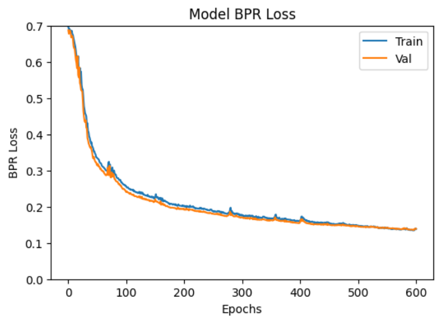


Train/Validation Loss

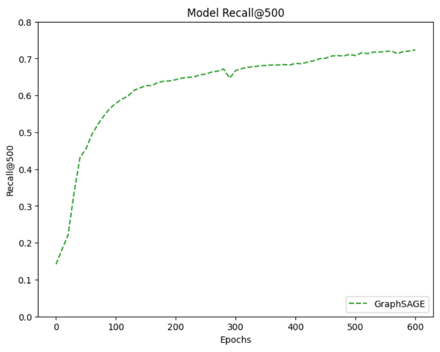


Recall@500

Figure 3: Experiment 2 model loss and recall curves.



Train/Validation Loss



Recall@500

Figure 4: Experiment 3 model loss and recall curves.

8 Link Prediction Head

For the link prediction head, I will use simple dot-product scorer between playlist and track embeddings. After training the embeddings, then given P ,matrix of playlist embeddings and T ,matrix of track embeddings; compute $S = PT^{\intercal}$, which is the score of the track T for the playlist P. These scores are then used to rank candidate tracks and recommend the top-K for each playlist.

9 Related Works

I’m building on the work of Eva Batelaan et al., I adopt their bipartite playlist–track graph as the baseline [4]. My primary modification is to integrate track node features retrieved from the Spotify Web API (e.g., energy, tempo, valence), and to derive playlist features by averaging the features of their constituent tracks. This content-aware augmentation is intended to strengthen recommendations, especially under cold-start conditions, while keeping the overall approach simple and reproducible.

10 Conclusion

To provide personalized recommendations, I assume manually curated playlists are internally coherent: where users add songs to a playlist that share characteristics. This assumption underpins my bipartite graph formulation and feature design, enabling the model to identify tracks that best git a given playlist.

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.