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

Robel Tirfe , 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 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[1]. 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" }
    }
  ]
}
```

3 Node Features

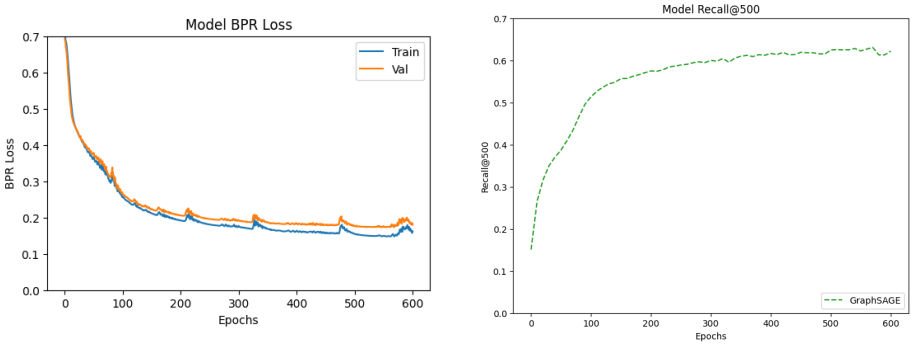
For each track node in the dataset, I will attach a feature vector. Using Spotify's public Web API [2], I will pull per-track attributes to serve as model input:

- 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.

	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 1: Experiment Results



Train/Validation Loss

Recall@500

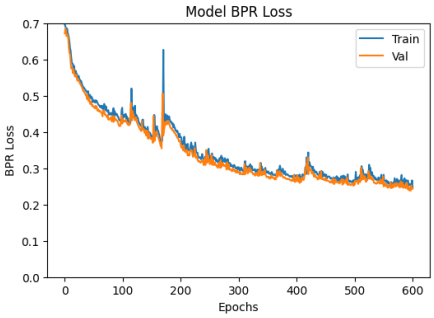
Figure 1: Baseline model loss and recall curves.

4 GraphSAGE

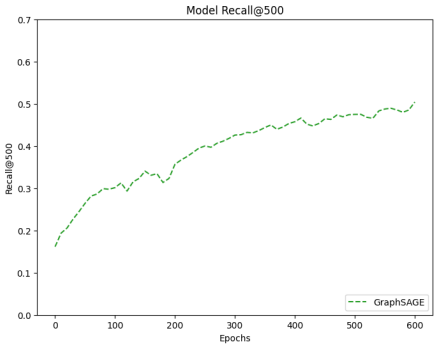
In this project, I will use GraphSAGE, a framework for inductive representation learning on large graphs, to generate low-dimensional embeddings for nodes [3]. GraphSAGE learns an embedding function that aggregates a node's features with sampled neighbors, so it can produce embeddings for new playlists 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.

5 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 $S = PT^{\mathsf{T}}$, 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.

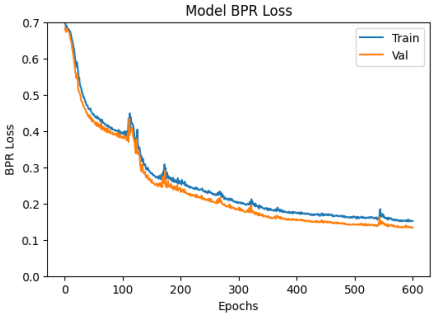


Train/Validation Loss

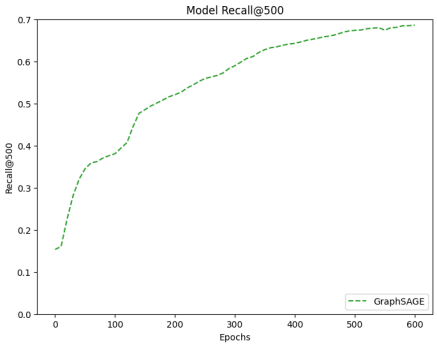


Recall@500

Figure 2: Experiment 1 model loss and recall curves.



Train/Validation Loss

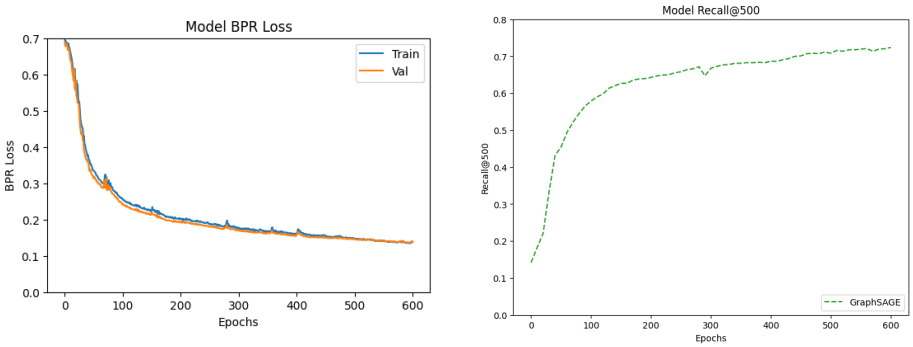


Recall@500

Figure 3: Experiment 2 model loss and recall curves.

6 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.



Train/Validation Loss

Recall@500

Figure 4: Experiment 3 model loss and recall curves.

7 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] 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.

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

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

[4] 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