



CSE 6240 - Web Search & Text Mining
Course Final Project

Track Recommendation System for Spotify Playlists

Aryan Vats
Megha Sharma
Reshma Anugundanahalli Ramachandra
Rynaa Grover





Introduction

Problem Statement:

Given a Spotify playlist, recommend tracks for automatic playlist continuation.

Importance:

- We propose a graph-based approach that leverages graph structure and playlist/track features to provide better recommendations.
- This can improve user experience on the Spotify platform and increase the discoverability of new songs.



Data

Original dataset:

- We use the Million Playlist Dataset from the RecSys Challenge 2018 (Chen et al., 2018).
- The dataset consists of 1 million playlists created by Spotify users, each containing between 5 to 250 tracks.

Property	Value
Number of playlists	1, 000, 000
Number of unique tracks	2, 262, 292
Number of unique albums	734, 684
Number of unique artists	295, 860
Number of unique playlist titles	17, 381

Dataset we used:

- Due to limited computing resources, we couldn't use the entire dataset.
- We sampled 1000 playlists at random and generated a new dataset to run the baseline and our approach.

Property	Value
Number of playlists	1, 000
Number of unique tracks	34, 443
Number of unique albums	17, 437
Number of unique artists	9, 754
Number of unique playlist titles	789

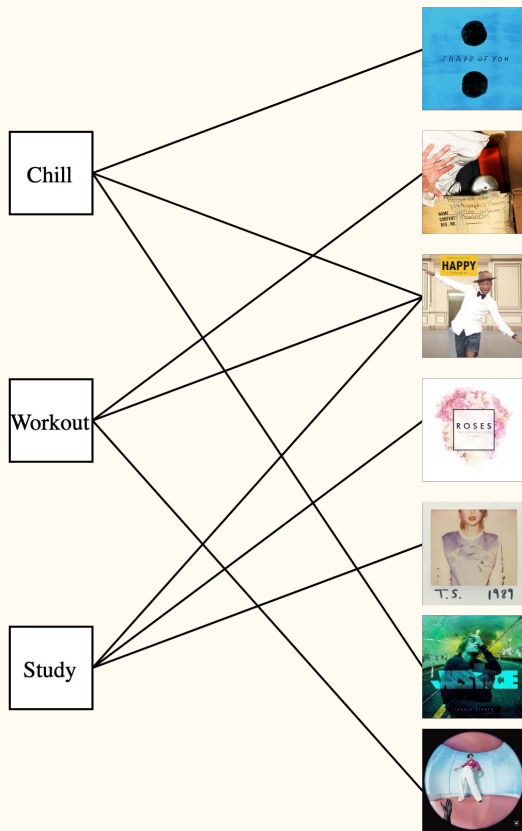


Approach - Algorithms and Visualizations

(Option 2)

PLAYLISTS

TRACKS



Baseline Approach: Graph Creation

- The playlist-track interactions are modeled as an undirected bipartite graph.
- A track t belongs to a playlist p if a link (p,t) exists in the graph.
- Thus, the problem is now a link prediction task.



Baseline Approach: Details

Algorithm: Random Walks with Restarts

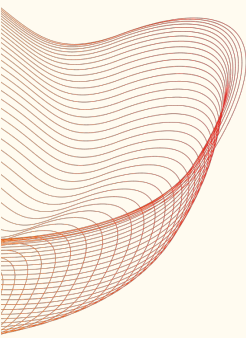
- Based on Multiple Random Walk approach on the Pixie algorithm for Pinterest.

How does it work:

- Tracks occurring in the same playlists are more likely to be similar.
- Use random walks with restarts to get node visit counts.
- Tracks with highest visit counts will be recommended to continue the playlist.

Drawback:

- Does not use playlist/track features and graph structure other than edges.
- No non-linearity
- No negative sampling



Baseline Approach: Algorithm

Algorithm 4.1 Single Random Walk with Restarts (based on [5])

```
1: function SINGLERANDOMWALK( $q$ : Query Node,  $G = (T, P, E)$ : Playlist  
   Graph,  $\alpha$ : Restart Probability,  $N_q$ )  
2:    $n \leftarrow 0$   
3:    $V \leftarrow \{v: 0\} \forall v \in T$   
4:   repeat  
5:      $t_{curr} \leftarrow q$   
6:      $n_{curr} \leftarrow 0$   
7:      $N_{curr} \leftarrow \text{RANDOMGEOMETRIC}(\alpha)$   
8:     repeat  
9:        $p_{curr} \leftarrow \text{RANDOMNEIGHBOR}(G, t_{curr})$   
10:       $t_{curr} \leftarrow \text{RANDOMNEIGHBOR}(G, p_{curr})$   
11:       $V[t_{curr}] \leftarrow V[t_{curr}] + 1$   
12:       $n_{curr} \leftarrow n_{curr} + 1$   
13:    until  $(n_{curr} \geq N_{curr}) \vee (n + n_{curr} \geq N_q)$   
14:     $n \leftarrow n + n_{curr}$   
15:  until  $n \geq N_q$   
16:  return  $V$   
17: end function
```

Algorithm 4.2 Multiple Random Walks with Restarts (based on [5])

```
1: function MULTIPLERANDOMWALK( $\mathbf{q}$ : Query Tracks,  $G = (T, P, E)$ : Playlist  
   Graph,  $\alpha$ : Restart Probability,  $N$ )  
2:    $N_q \leftarrow N/|\mathbf{q}|$   
3:    $V \leftarrow \{v: 0\} \forall v \in T$   
4:   for all  $q \in \mathbf{q}$  do  
5:      $V_q \leftarrow \text{SINGLERANDOMWALK}(q, G, \alpha, N_q)$   
6:      $V \leftarrow V + V_q$   
7:   end for  
8:   return  $V$   
9: end function
```



Proposed Approach : Graph Creation

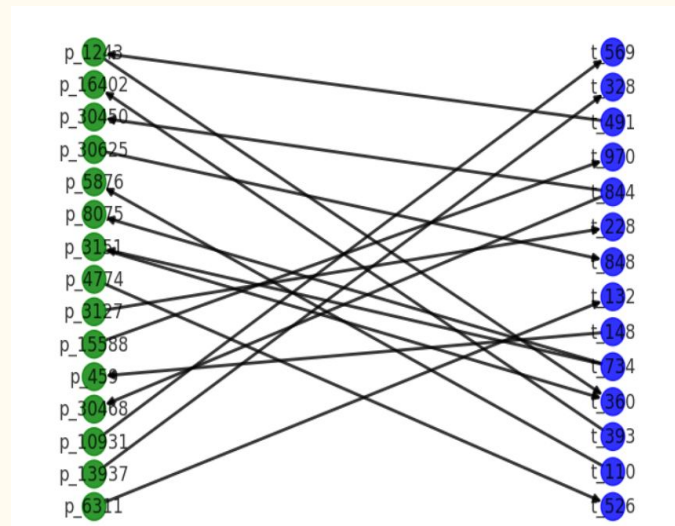
- Modeled similar to the baseline approach but with directed edges.
- Utilized the StellarGraph library to generate a heterogenous bipartite graph.
- StellarGraph enables inclusion of playlist and track features.

Playlist Features

Playlist name
Number of tracks
Number of albums
Number of followers

Track Features

Track name
Artist name
Album name
Duration
Position in playlist



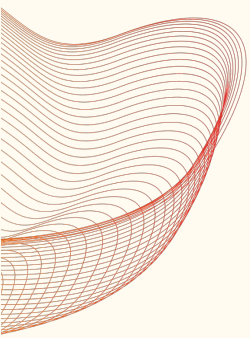
Graph generated for a sample of playlists and tracks



Proposed Approach : Data Preprocessing

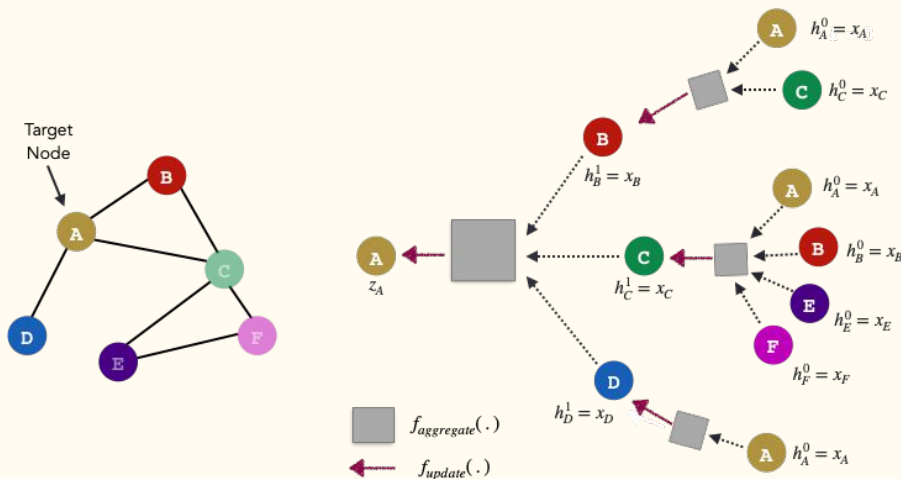
We explored three techniques to encode the textual features as numerics:

1. **One-hot encoding:** The generated embeddings were very sparse and large. Since the graph algorithm could process only float columns, we had to add separate columns for each entry of the encoding.
2. **Word2vec:** Training Word2Vec on proper nouns wasn't very efficient. Similar to one-hot encoding, the embeddings required a separate column for each entry.
3. **Count Vectorization:** Since we wanted to encode the entire name of the playlist, track, and album instead of individual words, we decided to use count vectorization to get unique identifiers for each name.





Proposed Approach : HinSAGE



$$\mathbf{h}_{\mathcal{N}(v)}^k \leftarrow \text{AGGREGATE}_k(\{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\});$$
$$\mathbf{h}_v^k \leftarrow \sigma \left(\mathbf{W}^k \cdot \text{CONCAT}(\mathbf{h}_v^{k-1}, \mathbf{h}_{\mathcal{N}(v)}^k) \right)$$

Recap - GraphSAGE:

- GraphSAGE is a generalized inductive Graph Convolution Network (GCN) that leverages node feature information to efficiently generate node embeddings for previously unseen data.
- The embeddings are generated by aggregating features from the node's neighborhood.
- However, GraphSAGE is capable of handling only homogenous node and edge types.



Proposed Approach: Mean HinSAGE

Node Embeddings: The Stellar library created two types of node embeddings (playlists and tracks) using the Deep Graph Infomax technique (Velickovic et al., 2019). This technique generates embeddings in an unsupervised manner using Random Walk.

Edge Embeddings: The edge embeddings are generated by concatenating the source and destination embeddings.

Negative Sampling: We created our own set of negative samples by randomly selecting a playlist and a track that doesn't already exist in the playlist. To avoid bias during training, we ensured that the positive and negative link samples were of equal size. Thus, total number of edges were equal to $(66k + 66k = 132k)$

Aggregation: Similar to GraphSAGE, the algorithm averages the embeddings of the neighbors using mean aggregator and concatenates with the node's embedding from previous iteration.





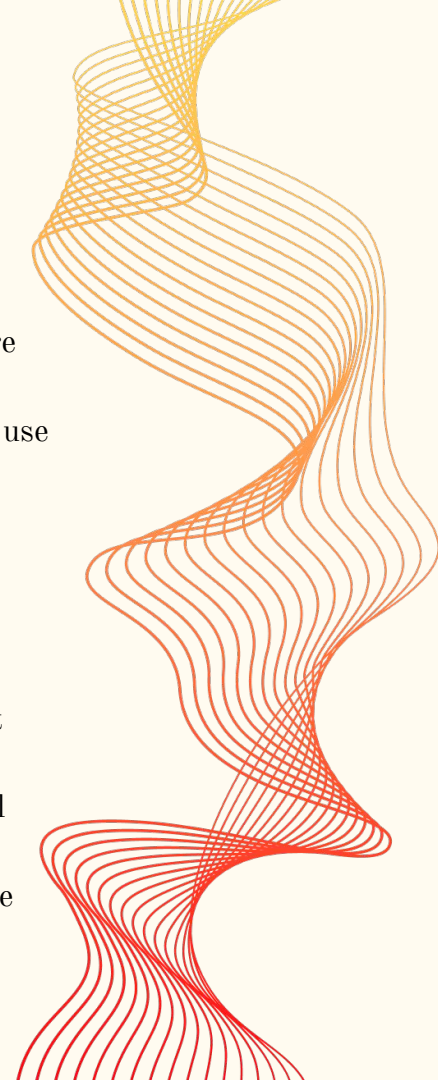
Proposed Approach: Discussion

Novelty:

- Prior automatic playlist continuation approaches did not leverage the graph structure and node features together.
- The proposed approach utilizes HinSAGE to generate holistic node embeddings and use them for link prediction.

Why does it solve drawbacks of baseline approach?

- Incorporating playlist and node features improves the performance considerably and yielded accurate link predictions.
- Using playlist features may also help in Cold Start cases where we just have playlist name.
- Inclusion of negative sampling ensured that we model the relationship and the model learns when a track does not belong in a playlist.
- Non-Linearity introduced through HinSAGE convolutions helped us better model the relationship between track and playlist.



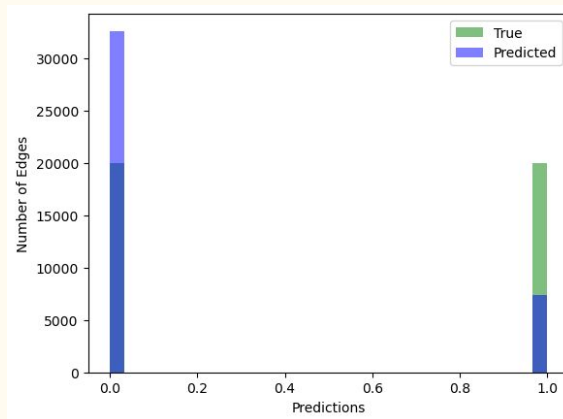


Experimental Setup & Results



Experimental Setup

- We formulated the task of track recommendation for playlist continuation as link prediction.
- We preprocessed the data, created a graph using StellarGraph library, and trained the HinSAGE algorithm on the PACE clusters with 2 GPU units.
- We trained the model to predict links of the type (playlist, track).
- For each pair of playlist and track the model outputs a confidence score between 0 and 1.
- For this task, we use the threshold of 0.7 to predict a link as we want high probability recommendations.

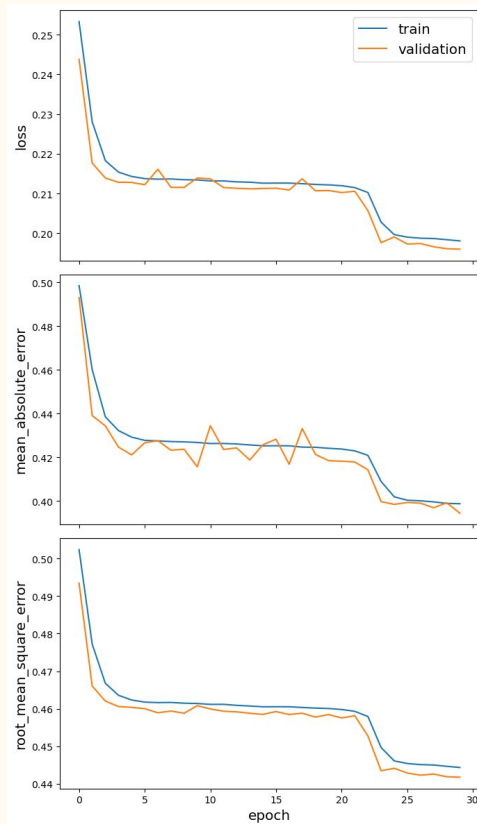



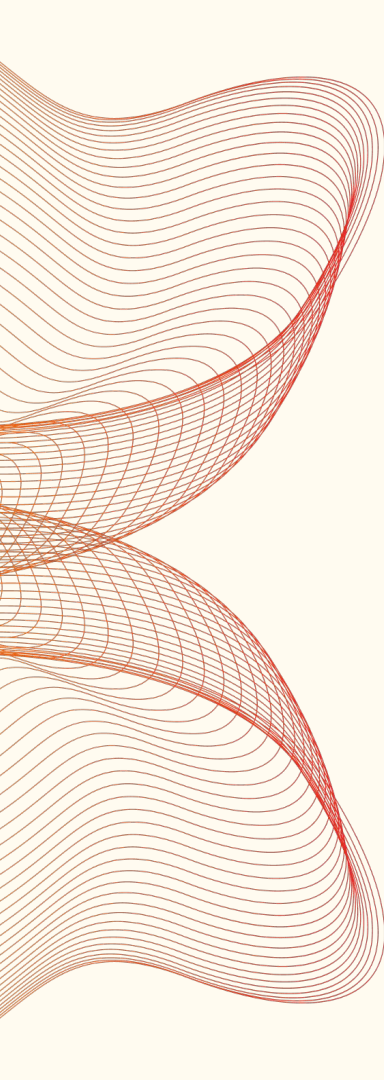


Experimental Setup & Results

We trained the HinSAGE model with the following hyperparameters to get the best predictions:

- Epochs = 30
- Learning rate = $5e-4$
- Batch size = 64
- Sizes of 2 hidden layers = 32, 16
- Dropout = 0.0
- Optimizer = Adam
- Loss = Cross entropy
- Number of GPUs = 2
- Number of workers = 4



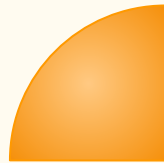


Evaluation & Results: Comparison with Baseline

We used two evaluation metrics to compare our proposed method with the baseline.

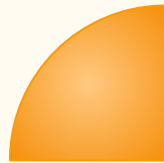
1. **R-Precision:** Measured by the number of retrieved relevant tracks divided by the number of known relevant tracks.
2. **Normalized Discounted Cumulative Gain (NDCG):** Measured by dividing the Discounted Cumulative Gain (DCG) by the ideal DCG (for the case with perfectly ranked recommendation track list). Rewards relevant tracks ranks higher in the list by measuring the ranking quality.

	Baseline	HinSAGE
R-Precision	0.0624	0.2756
NDCG	0.1785	0.3251





Limitations and Future Improvements

- By training on more data we can make the model more robust. Currently we have only trained our model on one slice given compute limitations.
 - Our link prediction task is unable to give a ranked order of recommended tracks. As a future improvement we can extend the link prediction model to generate ranked recommendations for the playlist.
 - Count Vectorization method of converting textual data can be improved since it is still a naive way of representing information.
- 

References

(Chen et al., 2018) Chen, C. W., Lamere, P., Schedl, M., & Zamani, H. (2018, September). Recsys challenge 2018: Automatic music playlist continuation. In Proceedings of the 12th ACM Conference on Recommender Systems (pp. 527-528).

(van Nidek & de Vries, 2018) van Nidek, T., & de Vries, A. P. (2018). Random walk with restart for automatic playlist continuation and query-specific adaptations. In Proceedings of the ACM Recommender Systems Challenge 2018 (pp. 1-6).

(Velickovic et al., 2019) Velickovic, P., Fedus, W., Hamilton, W. L., Liò, P., Bengio, Y., & Hjelm, R. D. (2019). Deep graph infomax. ICLR (Poster), 2(3), 4.



Thank you for your time 😊