Track Recommendation for Spotify Playlists using Graph Neural Networks

Course project for CSE 6240: Web Search and Text Mining, Spring 2023

Aryan Vats
Georgia Institute of Technology
avats31@gatech.edu

Megha Sharma
Georgia Institute of Technology
meghasharma@gatech.edu

Reshma Anugundanahalli Ramachandra
Georgia Institute of Technology
reshmaram@gatech.edu

Rynaa Grover
Georgia Institute of Technology
rgrover30@gatech.edu

ABSTRACT

Recommender systems are essential for music streaming services like Spotify, to provide personalized recommendations for users' playlists. However, many existing methods rely solely on content-based and collaborative filtering approaches, without considering the graph structure of the relationships between playlists and songs. A graph-based recommender system, such as GraphSAGE, can leverage the playlist-songs-user interactions and graph relationships to generate more accurate and robust embeddings for track recommendation. This paper proposes a graph-based recommendation system that utilizes Heterogeneous GraphSAGE (HinSAGE), an extension of GraphSAGE to heterogeneous graphs, for track recommendation on the Spotify dataset. The playlist-songs-user interactions in the dataset are converted into a bipartite graph representation using the Stellargraph library. Track recommendation is then modeled as a link prediction problem where given a playlist and track, the HinSAGE model can predict if a link exists between them. For this task, the proposed approach outperforms the baseline approach for a sampled version of the dataset. Thus, this approach can improve user engagement and ultimately enhance the overall user experience on the Spotify platform.

1 INTRODUCTION

Spotify is an immensely popular online music streaming service with over 140 million active users and a library of over 30 million tracks. The app enables users to create playlists, which presents the challenge of recommending new tracks to them. In this project, we aim to tackle the problem of automatic playlist continuation, which involves suggesting new tracks to users based on their existing playlists. This task was hosted at the ACM RecSys Challenge 2018[1], where most submissions relied on traditional recommender systems like collaborative filtering. However, these models have limitations, as they cannot effectively use the relationship between playlists and tracks to make recommendations.

To overcome this issue, one team adopted a graph-based random walk approach [14], which performed the best and won the challenge. However, we identified a few drawbacks in the approach, such as the reliance on task-specific feature engineering instead of generating node embeddings, and the computational resources and time required to train the model. To address these limitations, we proposed the use of the GraphSAGE algorithm [6], which generates node embeddings based on the graph's structure and has shown state-of-the-art performance in several benchmark datasets, especially those involving unseen data.

The dataset provided in the challenge consisted of 1 million playlists created by Spotify users, along with information about the tracks contained in these playlists. Due to limiting computing resource, we could not use the entire dataset and selected a subset of 1000 playlists to run the Multiple Random Walk with Restart and GraphSAGE approaches.

We created a heterogeneous bipartite graph with edges from a playlist to a track if the playlist contained the track and the modeled the problem of track recommendation as that of link prediction. We then used the HinSAGE algorithm to map the GraphSAGE technique to our heterogeneous graph. Finally, we compared the performance of our proposed approach against the Multiple Random Walk with Restart approach of the winning team using two popular evaluation metrics, R-Precision and Normalized Discounted Cumulative Gain. Our results show that HinSAGE shows significant improvement in both metrics when compared with the baseline for the same data and therefore is promising for the task of track recommendation.

Our work would be of benefit to the stakeholders at Spotify and other music streaming platforms as providing good song recommendations can greatly enhance user experience and increase user engagement and the discoverability of new songs.

2 LITERATURE SURVEY

2.1 Automatic Playlist Continuation

Automatic Playlist Continuation (APC) is defined in Bonnin and Jannah [2] as any task that utilizes *background knowledge* and *target characteristics* of playlist to generate a sequence of tracks. The authors approach the definition from a human perspective. Early works [1] define target characteristics of the playlist in terms of their musical attributes or metadata such as artist, style. Other works have defined target characteristics based either on the context of the listener by making use of sensor data and spatiotemporal data [12]. Background knowledge is extracted from the implicit actions of the users through the user activity. For instance, the study by Pichl. et. al [11] extracts and integrates contextual information from playlist names into the recommendation process. In this context, our goal of track recommendation can be thought of as a sub-task of APC. Here, Bonnin [2] proposes an alternative approach for

[1] https://www.recsyschallenge.com/2018/

applying collaborative filtering techniques to this task, wherein playlists are treated as users and tracks as items. Although this approach is discussed in the context of matrix-based techniques, we intend to adopt this definition for our graph-based solution.

2.2 Graph based Approaches

The two prominent approaches to collaborative filtering are (1) Latent factor methods that characterizes based on rating patterns and (2) Neighborhood methods that focus on the relationship between the commodity and the user. Matrix factorization is a popular latent factor method [8] that is employed in [7] in the context of collaborative filtering. A shortcoming of this method is that it has the new user problem or in this case, a playlist with less number of songs. Niedek et al. [14] employs a graph based approach to this problem statement by tracking recommendations as a result of multiple random walks over a playlist graph which is an undirected bipartite graph connecting tracks to playlists. The random walks exploit the concept that tracks belonging to the same playlists must be similar and playlists that have overlapping tracks must be similar. The results show that the systems suffer from strong popularity bias and that longer random walks tend to digress from target characteristics of the playlists. The experiment setup also ignored ordering information and included multiple edges for the same track. We intend to remove such redundancies in graph creation to see if it determine whether it yields a noteworthy enhancement.

2.3 Diversity Inclusion

Past studies [9] reveal the shortcomings of content-based recommendations where users criticized the uniformity and repetitiveness in consecutive tracks. Kaya and Bridge [7] introduces a novel idea for introducing diversity in recommendations. For long playlists, it performs greedy re-ranking of the baseline recommendations that minimizes their objective function that incorporates diversity. It employs a form of intent-aware diversification by identifying subprofiles of tracks- a subset of items that is intended to capture the user's interests- within each playlist. But such a diversification is not defined for short playlists and the paper falls back to baseline recommendations for short and medium sized playlists. Employing graph neural networks to represent data enables the solution to grasp intricate relationships and information from the graph's structure, we expect a higher likelihood of generating recommendations that are diverse yet still aligned with the user's preferences.

3 DATASET DESCRIPTION

The Million Playlist Dataset (MPD) used is the dataset from the Spotify Million Playlist Dataset Challenge [3]. The MPD randomly samples from 4 billion playlists created by users in the United States from 2010 to 2017. A playlist is an ordered list of tracks with playlist metadata. Each track has an unique URI by which it can be retrieved. The related artists and albums are also included. The MPD consists of 1 million playlists with over 2 million unique tracks by nearly 300,000 artists, forming the largest public dataset of music playlists in the world. For model evaluation, a set of ground truth tracks ("reference set") is also provided for each playlist to compare with predicted tracks.

Our basic data preprocessing involved dropping rows with missing values, converting playlist and track titles to lowercase, stripping punctuations, representing emojis occurring in playlist titles as separate tokens and removing stop words. Furthermore, we propose additional methods to extract more meaningful information from categorical fields in the dataset which is detailed in Section 5.2.2. We downloaded the data from the Spotify API [13].

3.1 Data Preparation

For the purpose of our project we sampled 1000 random playlists to form the dataset that we used for training both the baseline and our proposed method for comparable results. The basic statistics of the data is provided in Table 1.

Table 1: Basic statistics for original MPD dataset and our sampled dataset

Property	Original	Sample
Number of playlists	1,000,000	1,000
Number of unique tracks	2,262,292	34,443
Number of unique albums	734,684	17,437
Number of unique artists	295,860	9,754
Number of unique playlist titles	17,381	789

4 EXPERIMENTAL SETUP

The prediction task takes a given playlist with its metadata and a list of tracks in the playlist as input and predicts a list of tracks that would continue the given playlist. The evaluation metrics considered by the Spotify Million Playlist Dataset Challenge will also be used in this project in order to evaluate where the proposed model stands with respect to previous submissions to this challenge.

The evaluation metrics considered are:

(1) R-Precision: Rewards total number of retrieved tracks irrespective of order. Measured by the number of retrieved relevant tracks divided by the number of known relevant tracks (obtained from ground truth set of tracks).
(2) Normalized Discounted Cumulative Gain (NDCG): Rewards relevant tracks ranks higher in the list by measuring the ranking quality. Measured by dividing the Discounted Cumulative Gain (DCG) by the ideal DCG (for the case with perfectly ranked recommendation track list). This is calculated as:

$$NDCG = \frac{rel_1 + \sum_{i=2}^{|R|} \frac{rel_i}{\log_2 i}}{1 + \sum_{i=2}^{|G \cap R|} \frac{1}{\log_2 i}} \quad (1)$$

We conducted experiments on the sampled dataset to train both the baseline and proposed approach. The experimental setup for both approaches was identical. The baseline implementation was obtained from a publicly available GitHub repository linked in the next section. On the other hand, we employed the StellarGraph library[4] to create the graph and train the HinSAGE algorithm for our proposed approach. We faced some difficulties in the installation of StellarGraph library as it is compatible with only Python versions< 3.8.0. Thus we used Python 3.7 environment for this project.

The training process for the HinSAGE algorithm was carried out on the PACE clusters, utilizing 2 GPU units. To ensure the reliability of the results, we partitioned the sampled data into train and test sets, allocating 70% and 30% of the data respectively for each set. We employed the test set to validate the performance of the models during training.

5 METHODS

Traditional recommender systems perform recommendations based on content or user similarity, in the case of collaborative filtering. However, they do not leverage the information from the graph of relationships between nodes. Graph-based recommender systems take into account the structure of the graph and are thus able to recommend better. Additionally, graph-based recommender systems provide more interpretability and result in diverse recommendations. In this project, we use a graph-based random-walk algorithm as baseline and create a new HinSAGE model.

5.1 Baseline

For the purposes of defining a baseline, we chose to look at the implementation of Random Walk on the bipartite Playlist-Song graph as adapted in the thesis by Van Niedek et al. [14]. We used the code available on their GitHub repository[2] for this project.

5.1.1 Graph Creation. They created a heterogeneous bipartite graph as $G = (P, T, E)$ where P is the set of all playlists, T is the set of all tracks, and E is the set of edges defined as (p, t) for every track $t \in T$ that is present in playlist $p \in P$. An idea of the graph is shown in Figure 1.

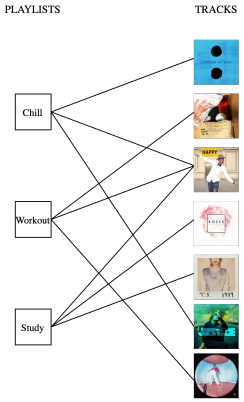

Figure 1: Depiction of the idea behind graph creation.

5.1.2 Multiple Random Walk with Restart Algorithm. The algorithm employed in this study is based on the Pixie Random Walk method described by Eksombatchai et al. [5]. It consists of two main parts: Single Random Walk with Restart and Multiple Random Walk with Restart.

The Single Random Walk with Restart algorithm takes a query track node as input and performs multiple random walks of varying lengths. The lengths of the random walks are randomly sampled from a geometric distribution over the restart probability parameter, α. During each random walk, the walker traverses from the query track node to a neighboring playlist node and then to a new neighboring track node, repeating the process until the walk ends. The track nodes are then ranked in the order of their visit counts, with nodes having higher visit counts being considered more similar to the query node.

For a playlist, the Multiple Random Walk with Restart algorithm is applied to run Single Random Walk for every track node in the playlist and then aggregate the results. This approach aims to capture the relationship between playlists and tracks by considering each track in the playlist separately and then combining the results to generate a playlist recommendation.

This approach helps the model use the data as a graph rather than perform rote memorization using learning techniques. We see a marked improvement from non-graph based methods to graph based methods detailed in the results in the thesis [14].

For our implementation, we used the following hyperparameters:

Table 2: Hyperparameters for baseline model

Hyperparameter	Value
Restart Probability	0.96
Number of steps per Random Walk	100000
Smoothing for transition probabilities	0.25

5.2 Proposed Method

The proposed method aimed to improve upon the limitations of the previous graph-based approach of random walk. The random walk approach utilized task-specific feature engineering but did not represent nodes (playlists and tracks) as general embeddings, and thus, could not capture higher-order relationships between nodes in a graph that traditional feature engineering methods miss. Additionally, the random walk approach took a long time to train. To address these limitations, the proposed method suggested using the GraphSAGE algorithm. However, since GraphSAGE is limited to homogeneous graphs, for the MPD dataset, we used HinSAGE instead. HinSAGE is a generalization of GraphSAGE that can be applied to heterogenous graphs.

5.2.1 Reasoning behind Proposed Approach. Node embedding methods are advantageous as they possess a better understanding of the graph structure and can generalize to unseen nodes. Deep learning techniques such as Graph Convolution Networks (GCN) and GraphSAGE are able to learn information about the graph's structure by taking into account the nodes' neighbors. As a result, these algorithms can generate recommendations for new nodes. Since the goal of our task is to recommend tracks for unseen playlists, GraphSAGE is a suitable option. It is an inductive approach that generates node embeddings by sampling neighboring nodes and aggregating their features. Hamilton et al. [5] demonstrated that GraphSAGE achieves state-of-the-art performance on various benchmark datasets, suggesting its ability to generate precise recommendations.

5.2.2 Data Preprocessing. Our dataset comprised two distinct types of entities: playlists and tracks. For each entity, a set of different features was available, as outlined in Table 3. Prior to utilizing

[2]https://github.com/TimovNiedek/recsys-random-walk

these node features to construct a graph, we performed necessary preprocessing procedures.

In particular, we examined several techniques for encoding textual features such as "Playlist Name," "Artist Name," "Track Name," and "Album Name" into numerical representations, enabling their use in the graph construction process.

(1) One-hot Encoding : We converted each textual column into unique columns with a value of either 0 or 1, depending on the text in the original column. Each unique occurrence, such as a track being from a particular artist, was represented by a single column. However, this approach resulted in data sparsity and was overall an inefficient way of representing textual columns.
(2) Word2Vec Encoding : Word2Vec [10] is a technique used to encode words into numerical word embeddings. We utilized the Word2Vec framework to convert each textual column into a 100-dimensional word embedding. The embeddings were high dimensional and required 100 columns for each features. Additionally, the embeddings were not very meaningful for proper nouns like the name of the artist.
(3) Count Vectorization : We assigned a numerical identifier to each unique entry of the column and represented each item in the feature column as it's numerical identifier. This approach helped us model the relationship of tracks by the same artist, or in the same album. It did not suffer from sparsity like the other approaches and was the approach we chose to proceed with.

Table 3: Playlist and Track node features

Playlist Features	Track Features
Playlist name	Track name
Number of tracks	Artist name
Number of albums	Album name
Number of followers	Duration
Collaborative	Position in playlist

5.2.3 Graph Creation. A popular approach for building graph-based recommender systems is to model the dataset as a bipartite graph, with two types of nodes representing playlists and tracks. Edges between two nodes indicate the presence of that particular track in that specific playlist. However, a significant challenge in creating this graph is representing the two distinct types of nodes, that have different features associated with them. To address this, we utilized the Stellargraph library which provides a unified API for constructing, analyzing, and visualizing heterogeneous graphs. We used the library to generate a directed, heterogeneous, bipartite graph for the 1000 playlist dataset that we created. We have $66k$ edges in our graph and Figure 2 shows the visualization of our bipartite representation on a sample of 25 nodes. In addition, we were also able to include playlists and track features using Stellargraph. Therefore, our graph creation approach resulted in a more information-rich representation of the playlist-track interactions compared to the baseline [14].

5.2.4 Details about HinSAGE.

- *Node and Edge Embeddings:* Since our graph is bipartite and contains different types of nodes, we generated different node embeddings for "Playlist" nodes and "Track" nodes. The StellarGraph library [4] assigns Node embeddings to each node by using the Deep Graph Infomax technique [15] that learns the embeddings in an unsupervised manner using random walks. The embeddings for the edges connecting a playlist and a track node were generated by concatenating the source and destination node embeddings. The model maintains separate self-feature matrices for every node type, ensuring that the Playlist nodes and the Track nodes learn embeddings and weights relevant to their own features.

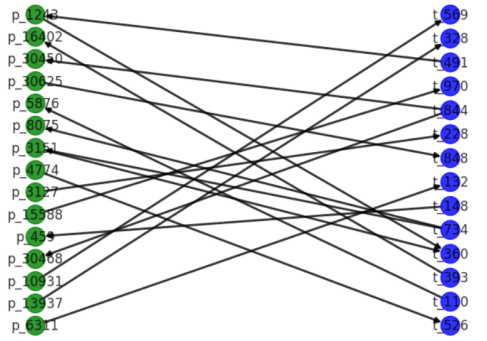

Figure 2: A subset of playlist and track nodes visualized using Networkx. The playlist nodes are shown in green with p_ prefix and the track nodes are in blue with t_ prefix.

- *Negative Sampling:* A key innovation from the baseline method was the incorporation of negative sampling to help our model better understand the relationship between tracks and playlists, including when a track should not be recommended as an APC candidate. We sampled negative edges through randomly selecting a playlist and a track and that does not exist in the playlist. The number of negative edge samples added were equal to the positive edge samples to avoid bias. Thus, the model was finally trained on $132k$ edges.

- *Aggregation:* HinSAGE employs similar aggregation techniques to GraphSAGE by aggregating the embeddings of the neighbours and concatenating the resultant embedding with the node's embedding from the previous iteration to yield the new embedding. We applied the mean aggregator to obtain the neighbourhood information over 2 layers, sampling 8 neighbours in the first hop and 4 neighbours in the second hop to aggregate the embeddings. The equation for the mean aggregation over the neighbors is shown in Equation 2, where $N_r(v)$ represents the neighborhood of node v having r link between them.

$$h^k{}_{N_r(v)} = \frac{1}{|N_r(v)|} \sum_{u \in N_r(v)} [h_u{}^{k-1}] \tag{2}$$

One forward pass through layer k is shown in Equation 3, where $W^k{}_{t_v,\text{self}}$ is the weight matrix for self-edges for node

type t_v, r denotes the edge type, ${W^k}_{r,\text{neigh}}$ is the weight matrix for edges of type r, and b^k is the bias term.

$$ {h_v}^k = \sigma\left(\text{concat}[{W^k}_{t_v,\text{self}}[{h_v}^{k-1}], {W^k}_{r,\text{neigh}}{h^k}_{N_r(v)}] + b^k\right) \quad (3) $$

- *Model Implementation:* We implemented the HinSAGE model on the PACE cluster using the StellarGraph library and a Python 3.7.0 kernel. We also utilized 2 GPU units to enable parallel processing and speed up the training. Additionally, we used the Adam Optimizer for backpropogating through the layers and measured loss using Binary Cross-Entropy function. We experimented with several hyperparameter combinations to obtain the best model performance. The final hyperparameters chosen for model comparison are mentioned in Table 4

6 EXPERIMENTS AND RESULTS

6.1 Model Performance

In this section, we outline the experimental methodology and provide a summary of the results obtained. Firstly, we conducted a rigorous hyperparameter tuning process to optimize the model's performance. The specific hyperparameters used are presented in Table 4. To evaluate the performance of the model, we analyzed the loss and error curves, as shown in Figure 3. From the curves, we observe that the model exhibits good fitting without overfitting or underfitting. The primary objective of the model is to predict links between a playlist and a track. The output of the model is a value between 0 and 1. We selected a threshold of 0.7 for this task, as we were interested in obtaining high-probability recommendations. Overall, the experimental results indicate that our proposed model is effective in predicting links between playlists and tracks, and the selected hyperparameters are suitable for optimizing its performance.

Table 4: Hyperparameters for best model fit

Hyperparameter	Value
Epochs	30
Learning Rate	0.0005
Batch Size	64
Hidden Layer Sizes	32, 16
Optimizer	Adam
Loss	Binary cross-entropy
Number of GPUs	2
Number of workers	4

6.2 Comparison with Baseline Approach

In this section, we present a comparative analysis of our proposed HinSAGE model with the baseline approach using two evaluation metrics, namely R-Precision and NDCG, as mentioned in Section 4.

We conducted experiments for both the approaches on the same dataset and under identical experimental settings. The experimental results (shown in Table 5) indicate that our HinSAGE model outperforms the Multiple Random Walk with Restart model by a significant margin. We attribute this success to the following factors:

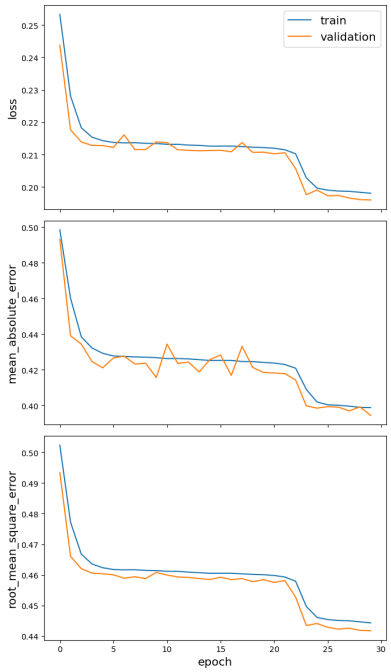

Figure 3: Loss and error curves for HinSAGE training

(1) The incorporation of playlist and track features improved the model's performance and facilitated more accurate link predictions.
(2) The utilization of playlist features proved beneficial in Cold Start cases where only playlist names were available.
(3) The inclusion of negative sampling enabled the model to learn when a track does not belong in a playlist.
(4) The introduction of non-linearity through HinSAGE convolutions allowed us to better model the relationship between tracks and playlists, thus contributing to improved performance.

Our proposed HinSAGE model exhibited faster training time (approximately 5 minutes) in comparison to the Random Walk algorithm, which took significantly longer (approximately 30 minutes) to run.

Table 5: Comparison between baseline and proposed method

	R-Precision	NDCG
Baseline	0.0624	0.1785
HinSAGE	**0.2756**	**0.3251**

In summary, the experimental results demonstrate that our proposed HinSAGE model overcomes several drawbacks of the baseline approach and yields superior performance in predicting links between playlists and tracks.

7 CONCLUSION

7.1 Limitations

While our model shows promise, there are specific areas where our model is limited by several constraints and can be enhanced to offer more relevant recommendations. Due to constraints on computation time and resources, the current training was limited to just 1000 randomly sampled playlists rather than the entirety of the data-set. Another limitation lies in the use of Count Vectorization for the representation of textual data (such as Playlist Name, Track Name, Artist Name) into trainable features. A final limitation that we observed was the inability to provide a rank order to the recommendations made, since our model is trained on a link prediction task.

7.2 Future Work

To improve the model's robustness and generalizability, an extension would be to train our model on the entire dataset, leveraging all of the data available to make intuitive recommendations. By extending our link prediction task to generate rankings for our predictions, we can prioritize and recommend more likely tracks as recommendations first. Another possible avenue for enhancing our approach is to investigate more sophisticated techniques for encoding textual data, such as playlist names and associated tags. These elements contain valuable information about the content of a given node type, and can be better utilized along with intuitive metadata features such as Acoustics, Valence and Tempo to better inform our model.

8 CONTRIBUTIONS

All team members contributed equally in completing the project.

REFERENCES

[1] M. Al ghoniemy and A.H. Tewfik. 2001. A network flow model for playlist generation. In *IEEE International Conference on Multimedia and Expo, 2001. ICME 2001.* 329–332. https://doi.org/10.1109/ICME.2001.1237723

[2] Geoffray Bonnin and D. Jannach. 2014. Automated Generation of Music Playlists: Survey and Experiments. *ACM Computing Surveys (CSUR)* 47 (2014), 1 – 35.

[3] 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*. 527–528.

[4] CSIRO's Data61. 2018. StellarGraph Machine Learning Library. https://github.com/stellargraph/stellargraph.

[5] Chantat Eksombatchai, Pranav Jindal, Jerry J. Liu, Yuchen Liu, Rahul Sharma, Charles Sugnet, Mark Ulrich, and Jure Leskovec. 2017. Pixie: A System for Recommending 3+ Billion Items to 200+ Million Users in Real-Time. *arXiv (Cornell University)* (11 2017). https://arxiv.org/pdf/1711.07601.pdf

[6] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. *Advances in neural information processing systems* 30 (2017).

[7] Mesut Kaya and Derek Bridge. 2018. Automatic Playlist Continuation Using Subprofile-Aware Diversification. In *Proceedings of the ACM Recommender Systems Challenge 2018* (Vancouver, BC, Canada) *(RecSys Challenge '18)*. Association for Computing Machinery, New York, NY, USA, Article 1, 6 pages. https://doi.org/10.1145/3267471.3267472

[8] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix Factorization Techniques for Recommender Systems. *Computer* 42, 8 (aug 2009), 30–37. https://doi.org/10.1109/MC.2009.263

[9] Jin Ha Lee. 2011. How Similar Is Too Similar?: Exploring Users' Perceptions of Similarity in Playlist Evaluation. In *International Society for Music Information Retrieval Conference.*

[10] Tomas Mikolov. 2013. Efficient Estimation of Word Representations in Vector Space. https://arxiv.org/abs/1301.3781

[11] Martin Pichl, Eva Zangerle, and Günther Specht. 2015. Towards a Context-Aware Music Recommendation Approach: What is Hidden in the Playlist Name?. In *2015 IEEE International Conference on Data Mining Workshop (ICDMW)*. 1360–1365. https://doi.org/10.1109/ICDMW.2015.145

[12] Markus Schedl, Georg Breitschopf, and Bogdan Ionescu. 2014. Mobile Music Genius: Reggae at the Beach, Metal on a Friday Night?. In *International Conference on Multimedia Retrieval, ICMR '14, Glasgow, United Kingdom - April 01 - 04, 2014*, Mohan S. Kankanhalli, Stefan M. Rüger, R. Manmatha, Joemon M. Jose, and Keith van Rijsbergen (Eds.). ACM, 507. https://doi.org/10.1145/2578726.2582612

[13] Spotify. 2023. Spotify for Developers. https://beta.developer.spotify.com/documentation/web-api/reference/

[14] Timo Van Niedek and Arjen P. De Vries. 2018. Random Walk with Restart for Automatic Playlist Continuation and Query-Specific Adaptations. *Conference on Recommender Systems* (10 2018). https://doi.org/10.1145/3267471.3267483

[15] Petar Veličković. 2018. Deep Graph Infomax. https://arxiv.org/abs/1809.10341