The Sound of Links

Maja Kolar[a], **Blaž Pridgar**[a,1], **and Jure Savnik**[a]

[a]University of Ljubljana, Faculty of Computer and Information Science, Večna pot 113, SI-1000 Ljubljana, Slovenia

The manuscript was compiled on May 31, 2024

Spotify's recommender system is a key feature that enhances user experience by suggesting tracks to expand and improve new or existing playlists. This study explores the effectiveness of network analysis techniques for predicting links in Spotify's playlist-track data. We first reduce the dataset by sub-sampling it and using a k-core subgraph method to manage computational complexity. We then compare the performance of traditional Machine Learning methods (KNN) applied to handmade network features with advanced Graph Neural Networks (GNNs) such as LightGCN, GAT, and SAGE. Additionally, we incorporate Spotify's audio features as initial embeddings to improve the performance of our GNN models. Our findings reveal that while GNNs with audio features show improved predictions, the simpler KNN model using handmade network features surprisingly outperforms GNNs. This suggests that we don't need complex models to get good link predictions, but can rely on the very powerful yet simple structural features of the network.

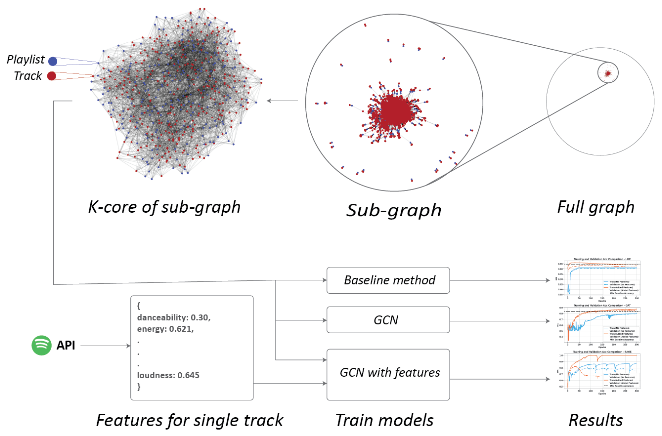

Fig. 1. The Visualisation of our study's pipeline. We construct a bipartite subgraph from the data, consisting of track and playlist nodes. We then create its k-core with k=30, split the data, and proceed to train and evaluate our chosen models. Additionally, we incorporate Spotify track features collected via API.

Spotify is a leading platform in the audio-streaming business, with over 600 million users. One of the features that attracts many customers to Spotify is their recommender system. Although not all the details about their algorithms are known, Spotify is open about some of the methods they use (Collaborative and Content-Based filtering), and encourage the public to come up with solutions of their own. One such example was the RecSys Spotify 1 Million Playlist Challenge in 2018, where teams from around the world competed to create the best Playlist completion algorithm. Spotify provided the dataset of 1 million playlist created by users between 2010 and 2017, the metrics and a challenge test dataset with a hidden ground truth. The competitors had to create 500 track predictions for each playlist in 10 different scenarios, with varying amount of metadata and number of tracks already included in the playlist.

After the official RecSys challenge was concluded, a continuation of the challenge was opened on AICrowd and is still running. The challenge is by itself an intriguing problem for any computer scientist, but our interests were spiked by the lack of Network Analysis based solutions in the top contenders of the official challenge. Since the music data falls so naturally into a graph shape and this is after all the Introduction to Network Analysis course, we set out to test how the network based approaches would fare in the challenge.

In the first part of the problem, we needed to find a way to sub-sample and reduce the very large dataset so that it could fit on our local machines. Once the smaller dataset was defined, we performed link prediction, first with a simple baseline method, using handmade network features. Next we wanted to find out if a deep learning approach could outperform the simple method. We used the LightGCN architecture and evaluated the performance of more complex convolutional layers, such as GAT and SAGE. Additionally, we attempted to boost performance by incorporating Spotify music features as initial embeddings.

Related work

The official RecSys challenge (1) was concluded 6 years ago, and since then, the continuation of the challenge on AICrowd (2) has been constantly active. Therefore many solutions have accumulated over the years, so we had a lot of literature to lean upon. The first piece of literature was the overview of the top results and methods from the official RecSys challenge (3). The top teams mostly used a two-stage architecture, by retrieving high recall tracks in the first stage and re-rank those in the second stage for a better prediction. In the first stage, methods like Matrix Factorization, Feed-Forward Networks, Recurrent Neural Networks and Autoencoders were popular. In the second stage, competitors used algorithms like XGBoost, GBDT, and LambdaMART. The teams in the creative track of the competitions also utilized the Spotify audio features that anyone can procure with the Spotify API (4).

We distanced ourselves from the methods used in the official challenge and focused solely on using Network Analysis tools. For this we used the architecture and algorithms for GNNs already implemented in the PyTorch library (5). For the baseline approach we used algorithms for community detection and other network structural features from the CDLib library (6). We found articles that used a similar deep learning techniques as us, like the *Spotify Track Neural Recommender System* Medium article (7). However, we found no articles that would make a comparison of deep learning techniques with the simpler machine learning algorithms applied to handmade network features. We also didn't find articles that would

All authors contributed equally to this work.

[1]To whom correspondence should be addressed. E-mail: bp58607@student.uni-lj.si.

describe the influence of using Spotify audio features as initial embeddings on the performance of GNN models.

Results

Each model's performance was assessed using ROC AUC and accuracy, with detailed visual representations of training and validation phases of our GNN models presented in Figures 2 and 3. Our results indicate that initializing embeddings with track features significantly improved the models' performance metrics with LGC and GAT convolutional layers. However, the SAGE model exhibits signs of overfitting, especially when augmented with additional track features. Overfitting can be seen in the ROC AUC and accuracy on train set reach nearly perfect scores, while the scores on validation set drop significantly.

Evaluation scores of test set are shown in Table 1 for all our models. Our baseline KNN algorithm which utilized hand-made network features performed surprisingly well given its computational simplicity. It outperformed all our GNN models in terms of both ROC AUC and accuracy. The performance metrics of GNN models highlight the improved performance when the models incorporate track features.

Model	Spotify Features	ROC AUC	Accuracy
LGC	No	0.8807	0.6455
	Yes	0.8985	0.6842
GAT	No	0.7385	0.6603
	Yes	0.8923	0.8178
SAGE	No	0.7070	0.6625
	Yes	0.6876	0.6408
Baseline	No	**0.9207**	**0.8402**

Table 1. Performance metrics for different models with and without added features

Discussion

We provide several insights into the use of network analysis for link prediction on Spotify playlist-track data. Firstly, when building a graph, the application of a k-core reduction strategy effectively reduced the graph size, while still maintaining a high percentage of network links. This approach was fundamental for our further analysis and management of computational demands of GNNs.

Next, augmenting original data with Spotify audio features and using them as initial embeddings in GNNs significantly boosted their predictive performance. These results confirmed our hypothesis that domain-specific information can fundamentally improve recommendation systems' performance. However, the incorporation led to overfitting in our GNN SAGE pipeline, which calls for caution in complex GNN architectures.

The robust performance of our baseline KNN model was a surprising key finding of our study, which undermines the common assumption that more complex models yield better results. Our results suggest that for certain types of networks and network-based predictions, simpler models not only provide lower computational demands, but can also be preferable in terms of final predictions. However, we are also aware that our training was done on a severely reduced subgraph. Especially in deep learning, data size has proven to be of utmost importance for improving the models' predictive performance. Therefore, running the same pipeline on hpc or with another service with more system and GPU RAM could possibly demonstrate superior performance of GNNs over simpler methods.

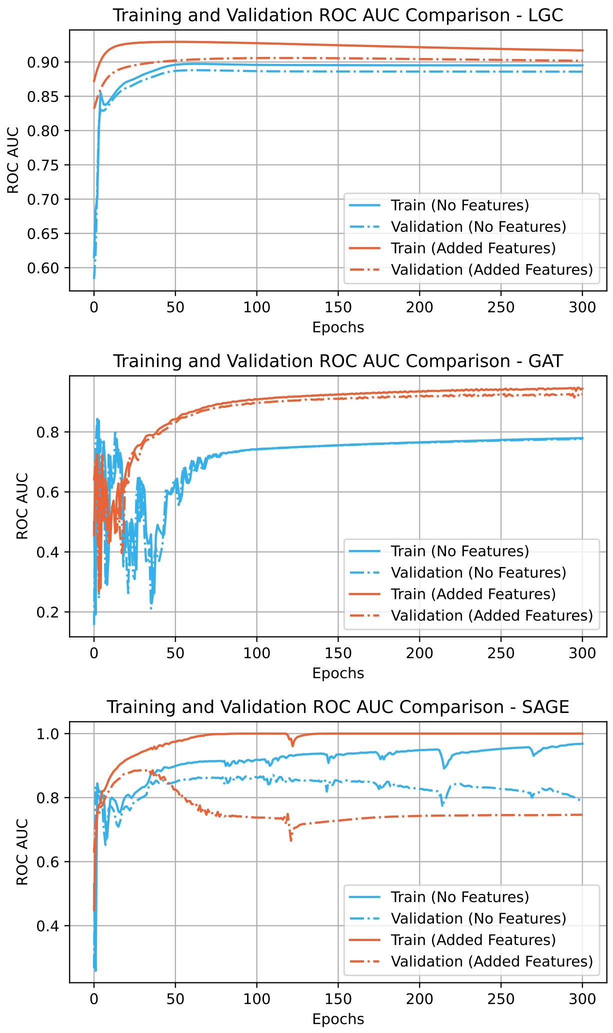

Fig. 2. ROC AUC Comparison. ROC AUC metric during training of all three GNN models with Spotify audio features (red) and without them (blue).

We are also aware that predicting additional tracks for playlists is a dynamic process, with new playlists being added in bulk to Spotify and other music streaming providers. Therefore, the network we dealt with is dynamic in nature and in production, we would need to re-evaluate our scores in regular time intervals.

Future work could explore several additional aspects, including regularization techniques to control overfitting in complex models like SAGE, further tuning of features integration and experimenting with different types of embedding initializations for GNN models. Since the simple models performed well with extracted network features, further exploration of incorporating the same network features with GNNs could yield positive results. Model stacking of simpler models such as KNN with complex GNN models could also potentially enhance model performances even further.

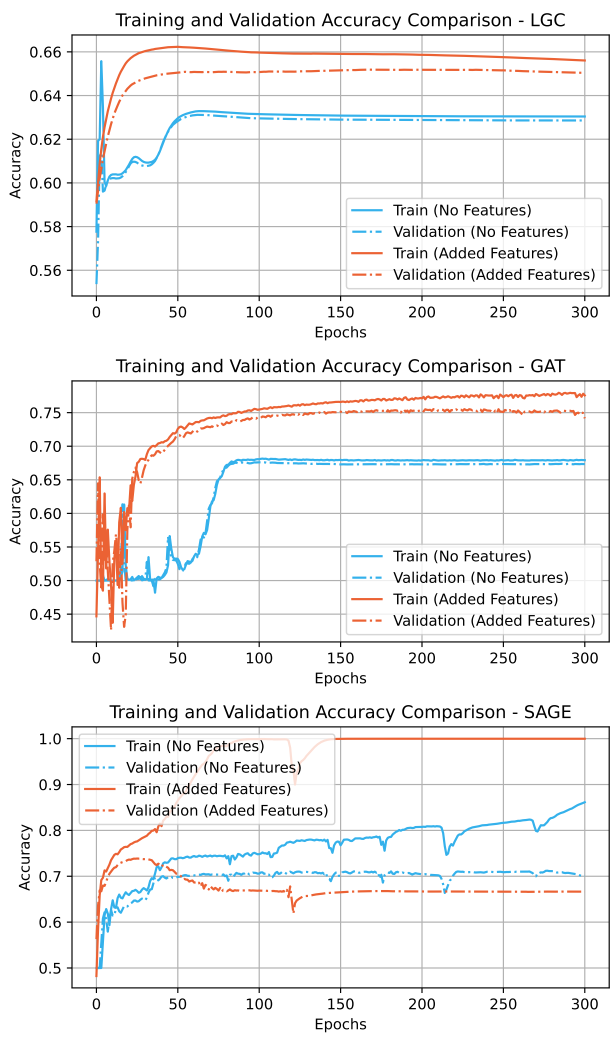

Fig. 3. Accuracy Comparison. Accuracy metric during training of all three GNN models with Spotify audio features (red) and without them (blue).

Methods

Data Our project uses the Spotify Million Playlist Dataset (MPD), which originally supported the RecSys Challenge 2018 (1) and was re-released on AIcrowd in 2020 (2), making it accessible for the development of music recommendation systems. The dataset includes one million real user-generated playlists created from 2010 to 2017 The collection consists of over two million unique tracks, which vary in terms of genre and other thematic groups. Each playlist includes the title, track list, and other meta information. Each track in a playlist consists of the unique track URI among other track information. Due to computational constraints, our analysis focuses on a subset of the MPD. We selected the first 50 JSON files from the dataset, representing 50,000 playlists, which is a manageable yet sufficiently diverse sample for our model training and testing in a standard Google Colab notebook with GPU RAM.

Building a graph For the purpose of our network analysis, we imagined our data as a bipartite graph where one set of nodes represents playlists and the other set represents tracks. An edge between two nodes from different sets indicates the inclusion of the track in the playlist. Firstly, we defined classes for loading the data, which were Track, Playlist and JSONFile. After loading the data, we built a bipartite graph, which contained 511,880 nodes and 3,300,244 edges. The data was too big to process in our pipeline, so we decided to make a reasonable subgraph of the data, which would reduce the set of nodes while maintaining a dense subgraph structure. We decided on making a k-core subgraph, where we chose k=30. The subgraph contained **34,753 nodes** (22527 playlists, 12226 tracks) and **1,575,269 edges**. The k-core reduction strategy effectively reduced the dataset's size, while still maintaining a large percentage of network edges. Since our analysis also focused on using Spotify audio features as initial embeddings, we added track features using **Spotify API** to each track. Each track has a numerical value between 0 and 1 (those that weren't were normalized) for the next features: danceability, energy, key, loudness, mode, speechiness, acousticness, instrumentalness, liveness, valence, tempo, and time signature. We additionally applied standardization to the 12 track features with mean 0 and standard deviation 1. Afterwards, we created an expanded features matrix, which contained all pairwise interactions of features, including the each feature with itself. We computed it through element-wise multiplication of two feature column vectors. We concatenated standardized track features with all pairwise interaction features to an expanded features matrix. We created a PyTorch Data object for our graph, which included the edge indices and the expanded features matrix for the node feature matrix.

Train-val-test split We had to pick an appropriate graph split for transductive link prediction. We employed the PyTorch Geometric's *RandomLinkSplit* function, designed to effectively perform splits that respect the graph's topology without any data leakage. This split returns two types of edges: message passing edges and supervision edges. The message passing edges are used to propagate information across the network, while the supervision edges are the edges we aim to predict. The method randomizes the removal of edges to create training, validation and test splits and ensures the following two conditions:

- the training set doesn't contain edges present in the validation and test sets,
- the validation set doesn't contain edges present in the test set.

Models For this project we used 4 different models. For the baseline we chose the **KNN** algorithms and ran it on the extracted network features which included: ID of the first node, ID of the second node, a train/test flag for splitting, the **preferential attachment** index, **Jaccard** index, **Adamic-Adar** index and two community flags indicating if the two nodes were part of the same community according to the **Louvain** and **Infomap** algorithms.

Next we employed 3 versions of the **LightGCN** algorithm with 3 different convolutional layers: **LGN** (default for LightGCN), **GAT** and **SAGE** as seen in (7). All models were trained for **300 epochs** with **3 convolutional layers**, a learning rate of **0.01** and a weight decay of **1e-5**. We use Bayesian Personalized Ranking (BPR) loss (8). All models were first trained with no initial embeddings with 78 dimensions as well as with the 78 Spotify Audio features that we created. To predict potential playlist-track pairs, we calculate the dot product between the two embeddings, which serves as a measure of similarity.

Evaluation We used ROC AUC and accuracy of predictions to evaluate the models' performance. We evaluated the GNN on training supervision edges, validation edges, and test edges, as described in the data split section.

Code The code we used in this project and the results it yielded can be found at https://github.com/BP4769/INA-Project-Spotify-Challenge.

1. ACM RecSys challenge 2018, . URL https://www.recsyschallenge.com/2018/.
2. AIcrowd | spotify million playlist dataset challenge | challenges, . URL https://www.aicrowd.com/challenges/spotify-million-playlist-dataset-challenge.
3. Hamed Zamani, Markus Schedl, Paul Lamere, and Ching-Wei Chen. An analysis of approaches taken in the ACM RecSys challenge 2018 for automatic music playlist continuation. URL http://arxiv.org/abs/1810.01520.
4. Web API reference | spotify for developers, . URL https://developer.spotify.com/documentation/web-api/reference/get-audio-features.
5. PyTorch documentation — PyTorch 2.3 documentation, . URL https://pytorch.org/docs/stable/index.html.
6. Giulio Rossetti, Letizia Milli, and Rémy Cazabet. CDLIB: a python library to extract, compare and evaluate communities from complex networks. 4(1):52. ISSN 2364-8228. . URL https://appliednetsci.springeropen.com/articles/10.1007/s41109-019-0165-9.
7. Benjamin Wittenbrink. Spotify track neural recommender system. URL https://medium.com/stanford-cs224w/spotify-track-neural-recommender-system-51d266e31e16.
8. Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, UAI '09, page 452–461, Arlington, Virginia, USA, 2009. AUAI Press. ISBN 9780974903958.

et al. Project report | **May 31, 2024** | **3**