Graph Neural Network Based Recommender Systems for Spotify Playlists

Shone Patil, Jiayun Wang, Benjamin Becze
GitHub: https://github.com/shonepatil/GNN-Spotify-Recommender-Project

Introduction

With the rise of music streaming services on the internet in the 2010's, many have moved away from radio stations to streaming services like Spotify and Apple Music. This shift offers more specificity and personalization to users' listening experiences, especially with the ability to create playlists of whatever songs that they wish. Oftentimes user playlists have a similar genre or theme between each song, and some streaming services like Spotify offer recommendations to expand a user's existing playlist based on the songs in it. Using Node2vec and GraphSAGE graph neural network methods, we set out to create a recommender system for songs to add to an existing playlist by drawing information from a vast graph of songs we built from playlist co-occurrences. The result is a personalized song recommender based not only on Spotify's community of playlist creators, but also the specific features within a song.

Data

Our song song recommendation system will work with just any music dataset that contains a community of users with playlists that they have created. The most popular of these would likely come from Apple Music, Spotify, Youtube, or Amazon as they are by far the most used music streaming services in America (that support playlist creation) as of January 2021 [1]. In all the markets Spotify and Youtube contend for the most used, but Youtube is not solely a music streaming service, and they do not release data for public use as readily as Spotify does, so we chose to go with Spotify as our dataset.

In January 2018, Spotify released a vast dataset containing 1 million playlists created by users between January 2010, and October 2017 for the purpose of an online data competition to try to predict subsequent tracks within a playlist [2]. Though the competition is over, we used this dataset of user's playlists to try to create personalized recommendations for a user's playlist. Currently we have taken the first ten thousand playlists from this dataset to train our model on, though scaling up to include more playlists (and subsequently songs) is possible, but currently not necessary for us to demonstrate the efficacy of this recommender.

Features

From these ten thousand playlists, we extracted all of the unique songs, which comes out to around 170,000 unique songs. We then utilized the Spotify developer public API to query information about each of these songs and obtain features for our model. These features include Spotify's own extracted numerical data from each song, of which we kept the following [3]:

- Danceability | Numerical - How suitable a track is for dancing.
- Energy | Numerical - Intensity and activity.
- Loudness | Numerical - Overall loudness of a track in decibels.
- Speechiness | Numerical - Presence of spoken words in a track.
- Acousticness | Numerical - How acoustic the track is.
- Instrumentalness | Numerical - How instrumental the track is.
- Liveness | Numerical - The presence of an audience in the recording.
- Valence | Numerical - The musical positiveness conveyed by a track.
- Tempo | Numerical - Estimated tempo in beats per minute.
- Duration | Numerical - Duration of the song in milliseconds.
- Key | Categorical - The key that the track is in.
- Mode | Categorical - Major or minor modality of a track.
- Time Signature | Categorical - Estimate of time signature.

For our recommender system to successfully provide personalized recommendations, we work under the assumption that when users create playlists manually, they generally will add songs that are similar to each other in some ways. A playlist could be comprised of songs pertaining to a specific genre like dance music or r&b, but it could also reflect a specific mood like happy songs that make you want to dance, or quiet sad songs. So within a playlist, we would expect the measures of the features above to be quite close to each other. To ensure this we examine the distribution of variances of the features from a random sample of songs, versus the distribution of variances from a random sample of playlists.

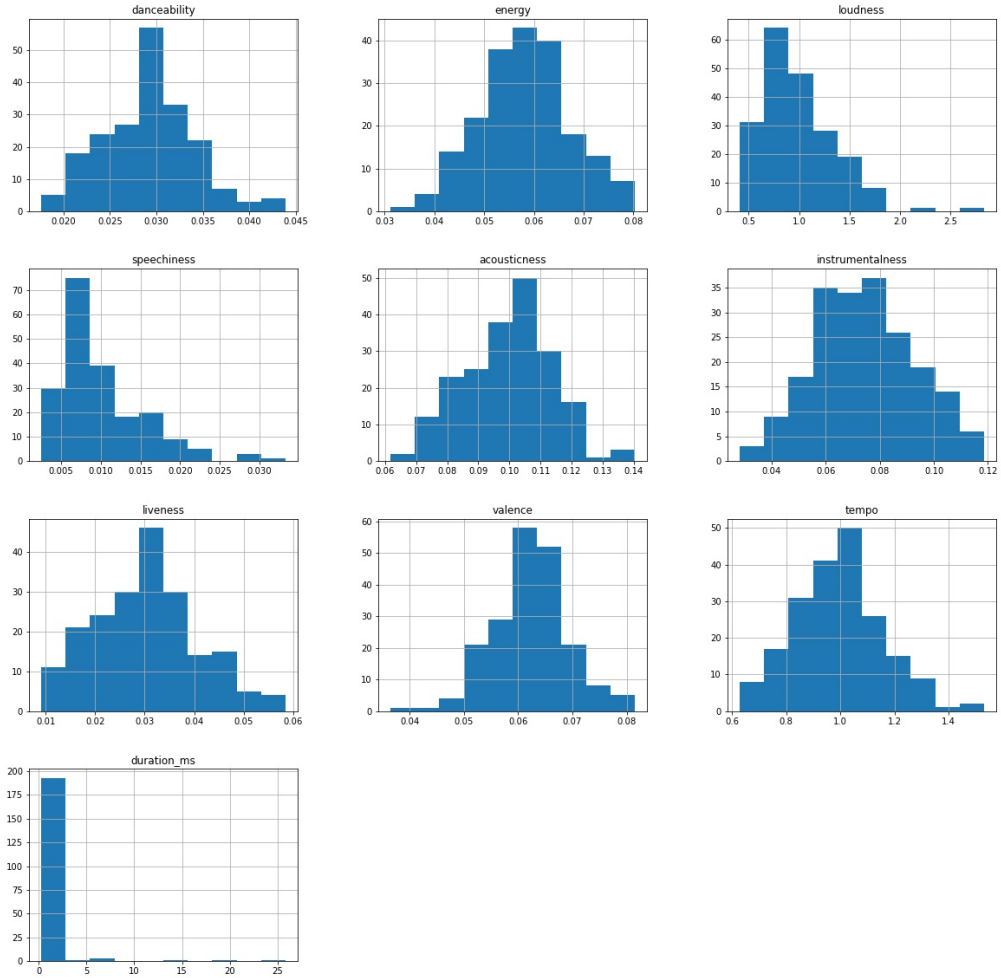


Figure 1: Variance distributions over 200 random samples of 70 songs each.

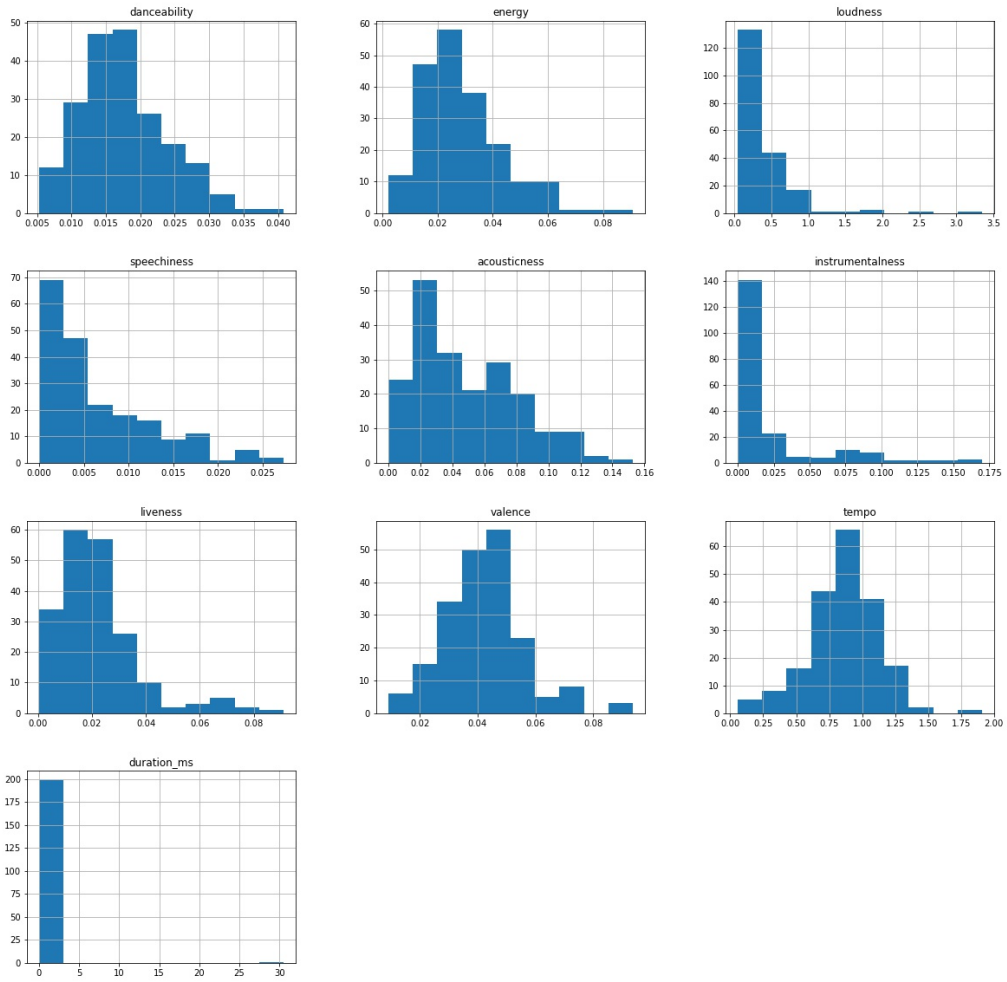


Figure 2: Variance Distributions of 200 randomly sampled playlists.

For comparing the variances of song features, we sampled 200 random playlists and compared the distribution of variances to 200 random samples of songs from a population of all songs. Each random sample was 70 songs, the average length of the playlists that we had sampled. When we compare the variance distributions from Figure 1 to Figure 2, we can see that there is a tendency for many of the features to be right skewed, indicating an overall smaller variance for playlists. This supports our hypothesis that songs within playlists change much less than randomly selected songs. There are additional measures and figures included in the EDA notebook in src/analysis.

Graph

The graph we created consists of about 170,000 nodes corresponding to each unique song, and a vast set of edges connecting the songs that appear in a playlist from the first 10,000 playlists we selected. To create an effective recommender, we needed a way to rank the closeness of two songs, so as our aggregate we decided on co-occurrence of songs within playlists as the edges between them with a weight on each edge representing the amount of co-occurrences across all playlists. It should be noted that this method of connecting songs through co-occurrence can be considered as a hyperparameter for the entire pipeline. There are some other possible ways of determining edges and edge weights in the graph such as connecting nodes based on how close they appear to each other within a playlist, or how many times two songs that appear together in a playlist were not skipped. We chose simply co-occurrence for our graph because we want to capture node neighborhoods of songs that are alike for our recommender, and we assume that people will create playlists of songs that are at least somewhat alike. We believe this is sufficient for this purpose, but with future optimizations and time to re-create graph structure, trying different methods for graph creation could yield potentially beneficial results. Each node also contains a feature set of the features that are described above. With weighted edges and node features, we would have enough data to create a personalized link prediction problem. Our result was a weighted adjacency matrix with the following measures:

Nodes	461880
Edges	106486690
Average Degree	461.1011
Features	13

Though the graph is a network created from the inputs of various users, it holds a few advantages over a typical collaborative filtering network recommender. The weighted edges were created from information about the co-occurence of songs between the many playlists that we have sampled, but the nodes themselves are unique songs that contain descriptive audio features about themselves. By incorporating these features as well as the weights on the edges, we can begin to define more general groups of songs (playlists) that are based on more than just user input, but about the nature of the songs themselves.

Using a graph for our recommender also adds a solution for cold start issues that many existing recommenders have. GraphSAGE, an inductive algorithm, aids with this issue in that it will be able to create a node embedding for an unseen node using the information that was gathered from node neighborhoods from the training data. Where a traditional recommender may fail in these type of problems, GraphSAGE can readily add new data and create edge predictions for them because of the embedding that can be generated from its features as well as the

neighborhood that it falls in, which in this case would be what other songs appear in the playlist(s) that the new song has been added to. This inductivity also makes re-training of the dataset unnecessary whenever new nodes are introduced, which for Spotify-a music streaming service that constantly hosts new music from creators-is a scalable and realistic approach.

Methods and Literature

Node2Vec

One of the earlier graph-based methods is the node2vec which uses biased random walks to create low dimensional space representations for nodes [4]. This algorithm aims to preserve node neighborhood networks for the node embeddings and it allows for more accurate classification on nodes because of these neighborhoods. The algorithm utilizes biased 2nd order random walks at the core of its algorithm with p and q tunable parameters to determine the probability of each node subsequent from the original of being visited. Tuning of these parameters allows for the user's choice of having a more local walk emulating bread-first sampling, or a more explorative walk emulating depth-first sampling. The p parameter determines the probability of a node being revisited right after a step, where a high value makes it less likely that the node is revisited, promoting a depth-first random walk. The q parameter controls the probabilities of stepping to an unvisited node, where a higher value is biased towards local nodes and a smaller value promotes visiting nodes farther from the original.

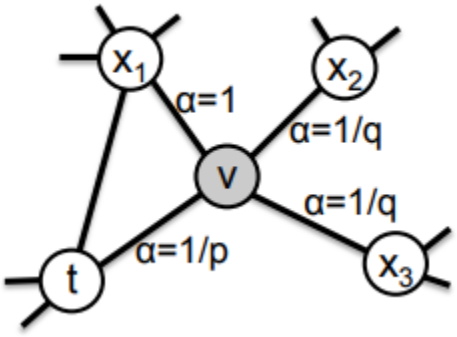


Figure 3: Grover and Leskovic visualization of the node2vec random walk [4]. "t" is the starting node, "v" is the current node in the random walk.

In our edge prediction model, we experiment with node2vec to create node embeddings which we then input into a KNN model to generate our predictions. We compare that with the following GraphSAGE method.

GraphSAGE

GraphSAGE is a framework for node embeddings that separates itself from existing transductive methods that require every node to be present in the training process by proposing an inductive approach to create node embeddings. This is a much more scalable approach that is ideal for large networks that don't all fit into memory, and that can be continuously updated.

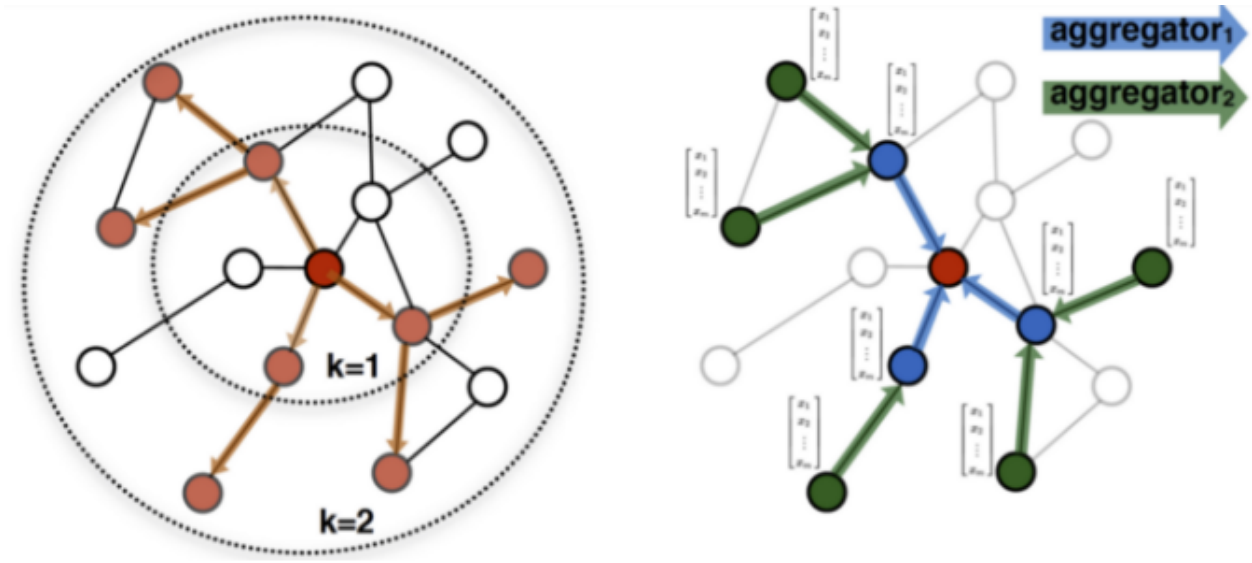


Figure 4: Hamilton, Ying, and Leskovic's visualization of feature aggregation from a node's neighborhood [5].

The graphSAGE algorithm's inductive ability comes from its use of neighborhood feature aggregation for a node to create an embedding for that node that will capture information about its neighborhood. Aggregation can be done in a number of ways, the most common being LSTM, mean, pooling aggregator functions. Depending on the aggregator chosen, graphSAGE will capture different information from its neighboring nodes. It is the neighborhood aggregation that makes it so that when a new node is added to the graph, its embedding can be generated from its features and neighboring nodes, rather than having to create new embeddings using the entire graph structure over again.

$$J_{\mathcal{G}}(\mathbf{z}_u) = -\log\left(\sigma(\mathbf{z}_u^\top \mathbf{z}_v)\right) - Q \cdot \mathbb{E}_{v_n \sim P_n(v)} \log\left(\sigma(-\mathbf{z}_u^\top \mathbf{z}_{v_n})\right)$$

GraphSAGE loss function [5]

The above loss function an unsupervised loss function used when generating embeddings, where u and v are two neighboring nodes, Q is the number of negative samples, v_n is a negative sample, σ is the sigmoid function, P_n is the negative sample distribution, and z_u are the

representations. For an edge prediction problem, our negative samples are generated from the non-existing edges between each song node.

Model

Node2Vec

As a baseline model, we built Node2Vec embeddings for songs in the graph based on the weighted connections of co-occurrences in playlists that existed. Using these node embeddings, we fed them into a K-Nearest Neighbor model to find similar embeddings to serve as recommended songs for each song. We explored this model because this is a common method used to encode graph data into a single feature set alongside song features and doesn't suffer from scalability issues like Graph Convolutional Networks do. It also was able to account for weighted edges that we created in our song graph. After finding similar embeddings for each song, we treated those as predicted edges to evaluate against ground truth to compute precision and recall metrics. We also aggregated the closest embeddings to be incorporated into a recommender. Some drawbacks we expected were performance and accuracy of Node2Vec embeddings in representing neighborhoods as well as runtime issues when computing K-Nearest Neighbor on heavily connected songs with high degrees.

GraphSAGE

Based on the graph of all songs with links weighted by co-occurrences in the playlists, we compute GraphSAGE embeddings for each song with a mean aggregator. Specifically, given the original node features, each step of GraphSAGE will sample from the neighbors of each song and average the information of those samples.

With those new representations of each song on hand, we then feed those embeddings into a multilayer perceptron predictor. For each pair of nodes in the graph, the predictor concatenates their corresponding embeddings as the edge feature, runs those embeddings through fully-connected layers and outputs a scalar score for each edge of the given graph.

We then use binary cross entropy as the loss function, which compares the predicted probabilities of the edge existence and the ground truth and penalizes the probabilities based on the difference.

We employed the GraphSAGE model since it provides a general and inductive framework which leverages node features to efficiently generate embeddings for large graphs with rich node attribute information, which matches our case of Spotify million playlists. For the downstream link prediction task, we experimented with dot product predictor and multilayer perceptron predictor and observed better loss update with the multilayer perceptron.

Recommender

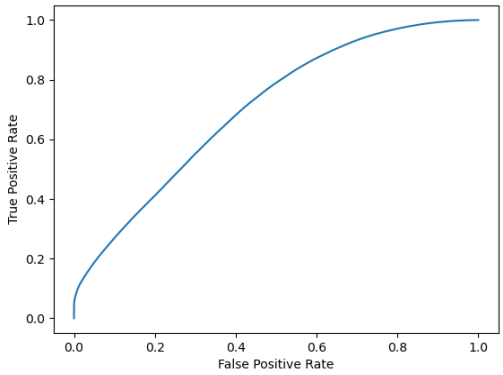
With the trained embeddings of songs in hand, given a playlist with k seed tracks, for each seed track s and each the candidate track c for recommendation, we will feed the embeddings of s and c into the predictor to compute the predicted scores, which indicates the probability of a edge existing in between. After computing the scores for each seed song, we rank the scores in descending order and prioritize the recommendation of those songs that have the highest probability of a link with the seed tracks.

Results

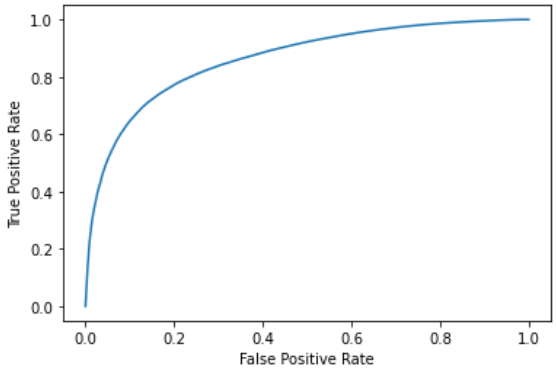
Link/Edge Prediction Metrics

	AUC	Precision	Recall
Node2vec - KNN (K = 50)	**0.70303**	**0.54797**	**0.23445**
GraphSAGE - MLP Train	**0.87212**	**0.97296**	**0.75723**
GraphSAGE - MLP Test	**0.86339**	**0.83424**	**0.75982**

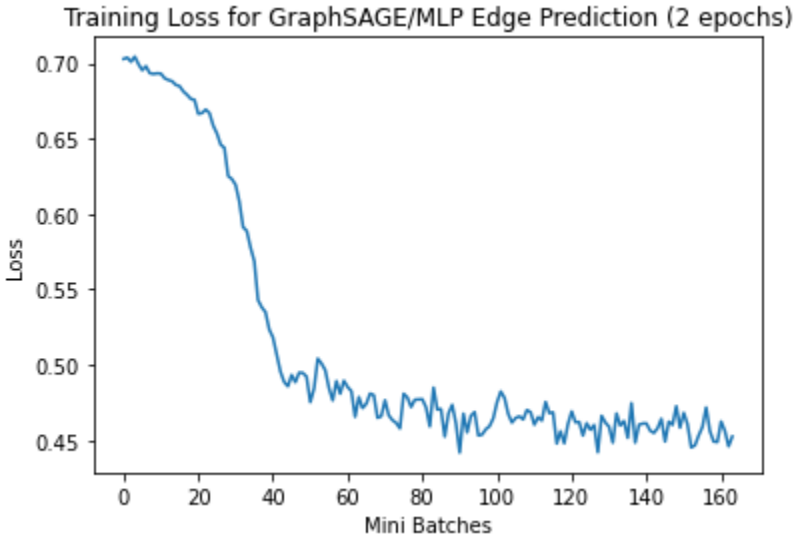
After running these two models and their respective embeddings, we found that the GraphSAGE embeddings with the MLP predictor gave the highest accuracy. This is as expected, as Node2vec embeddings are quite a bit less complex than the convolutional learning layers that are used in the GraphSAGE embeddings. The high precision entailed that the model was good at predicting edges between songs that should exist for a given song and limited false positives overall. We were more interested in recall as it told us the ratio of predictions out of all edges that do exist between songs. This was important as we believed the strongest recommender would first pull from existing edges that come from a song but aren't necessarily in a given playlist yet. We saw improvement in both categories and also saw higher AUC scores in GraphSAGE - MLP. The AUC score increase showed better performance in predicting true positives and true negatives over all the data. We can also see this in the ROC curves below that show GraphSAGE - MLP to have a curve closer to the top left than the Node2vec - KNN curve.



Node2vec/KNN ROC



GraphSAGE/MLP ROC



GraphSAGE Loss Curve

The above graph is the record of training loss for the edge predictions when training the MLP model, where we trained for 2 epochs, with 82 mini batches for each epoch. The reason for stopping at 2 epochs is that we saw the loss begin to stagnate and oscillate around 0.48. To avoid overfitting the model, we ended training at this point. We had also noticed the validation AUC still increasing in spite of no training loss decrease so this gave us another reason to stop early to prevent overfitting on the validation set. To help with training speed, we added options to use the CUDA framework and utilize GPUs during mini-batch training.

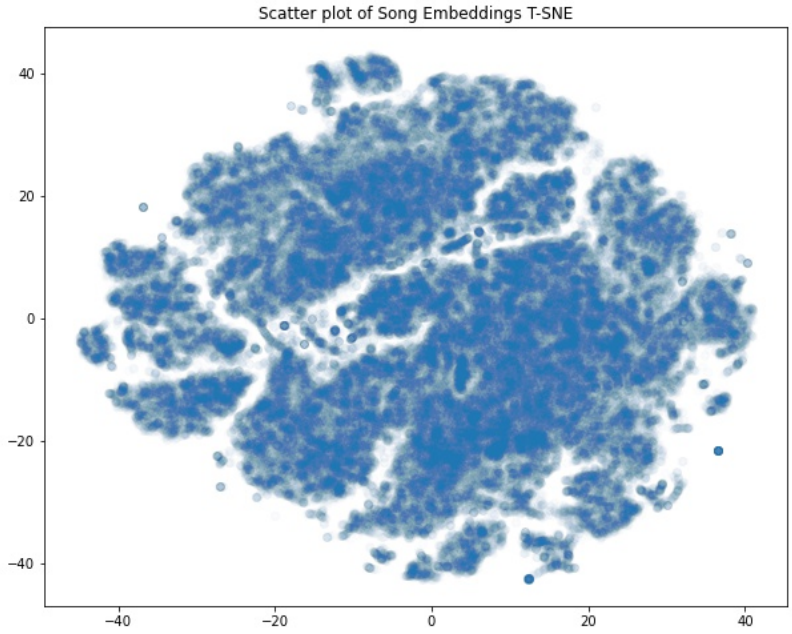


Figure n: GraphSAGE embeddings T-SNE plot (170k Songs)

The above T-SNE plot shows the different embeddings of node neighborhoods. We used embeddings for a subset of 170,000 nodes for better readability, but the model and the process are the same.

Playlist Recommendation Results

All of the following are done on recommendation sets for 200 random playlists. The recommendations are done as sets of the top recommended songs for each unique song in the playlist.

File Location: data/mpd.slice.9000-9999.json
Name: ‘Happy :)’
Pid: 9360

Playlist:
Lisztomania---Phoenix
Rabid Animal---Lake Street Dive
Dancing On Quicksand---Bad Suns
The Sweet Escape---Gwen Stefani
Ants Marching---Dave Matthews Band
Rock the Casbah - Remastered
Stare Into The Sun---Graffiti6
Feel It Still---Portugal. The Man

anywayican---WALK THE MOON

Recommendations:
John Cougar, John Deere, John 3:16---Keith Urban
The Tiki, Tiki, Tiki Room — The Mellomen
Crazy In Love (feat. Jay-Z) — Beyonce
Woman — Harry Styles
Sweet Caroline — Neil Diamond
Before You Start Your Day — Twenty One Pilots
X (feat. Future) — 21 Savage
Rock and Roll All Nite — KISS
T-Shirt — Migos
HUMBLE. — Kendrick lamar

Evaluate the Recommender: R-Precision

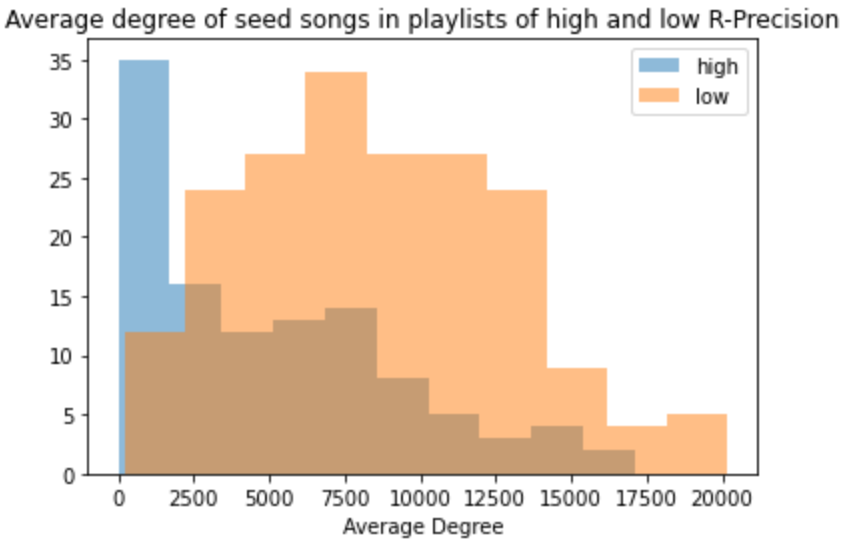
Given a playlist, we will partition the list with half of the tracks as seeds and the rest as masked relevant tracks (i.e our recommendation goals). The R-Precision will be calculated as the number of songs that belongs to the intersection of our recommended candidate tracks and the masked relevant tracks, divided by the number of masked relevant tracks.

The metric is averaged over all the playlists to test.

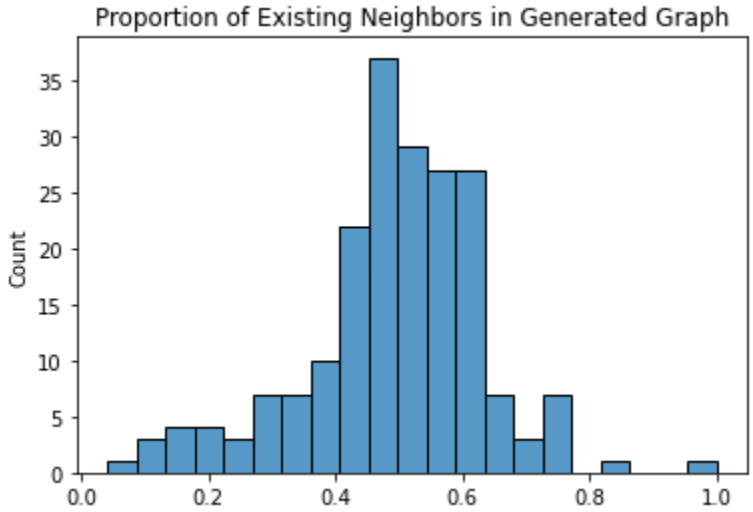
$$\text{R-precision} = \frac{|G \cap R_{1:|G|}|}{|G|}$$

We randomly sampled 300 playlists and ran recommender on each. Each playlist will get same number of recommended candidate tracks as the number of seed tracks given. The averaged R-precision of these 300 playlists is about 0.0635. This indicates that the recommender will be able to throw out about ~6% of the songs that are exactly the ones already in the existing tracks in the target playlists.

To further understand the performance, we individually looked at playlists with high R-precision (i.e >=5%) with the lower ones. We observe that for those playlists with higher R-precision, the average degree of the songs (i.e average number of songs that appear together with this song in the playlists) is significantly lower than those with lower R-precision.



Song connectivity in Song recommendations:



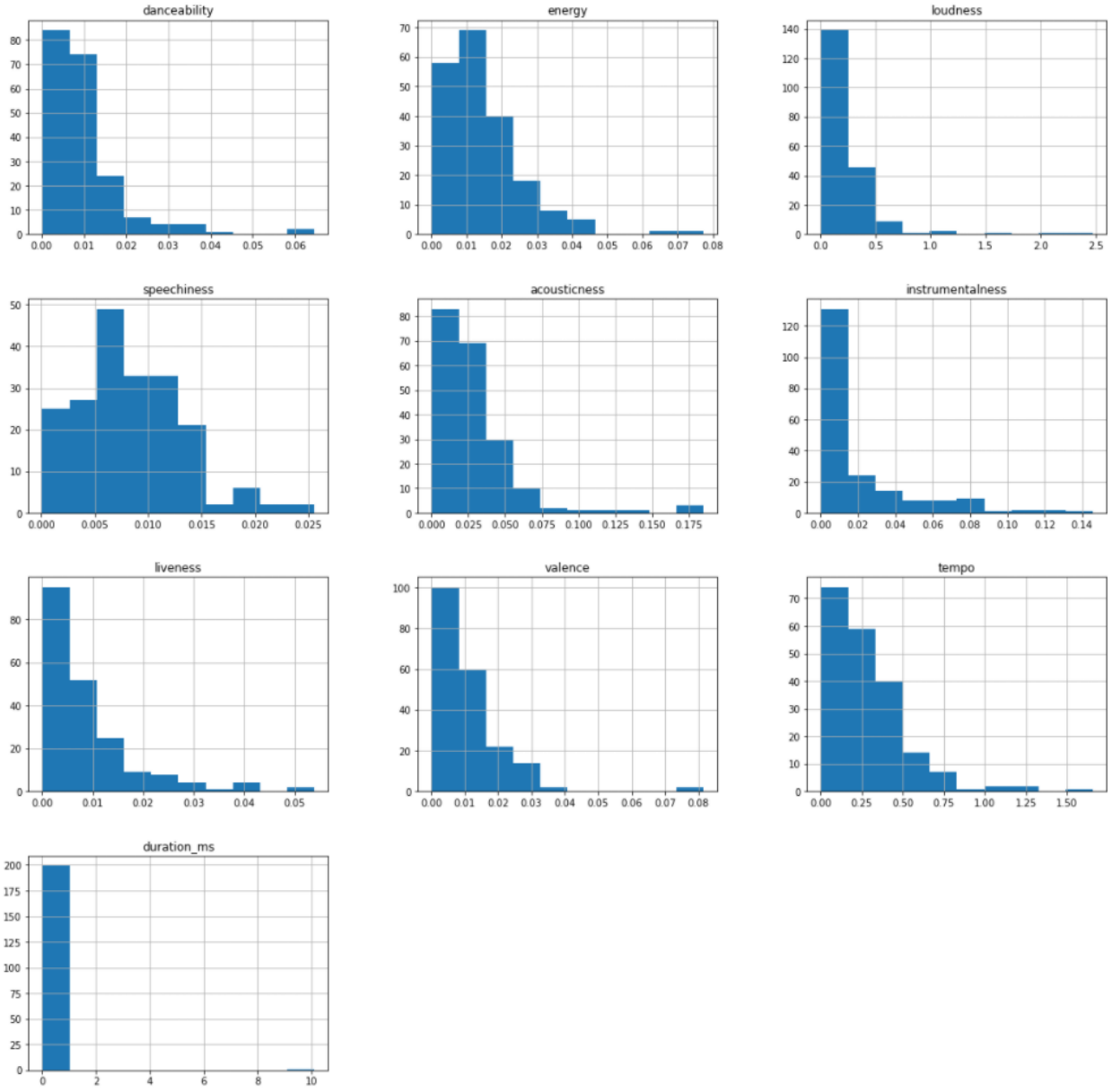
Checking all pairs of nodes in the recommended set, and this is the proportion of those n^2 pairs that are actually connected in the main graph (where n is the number of songs in a playlist). We can see that there is a seemingly normal distribution, where most of the playlist recommendations have about a 0.5 rate of neighbors existing together. Though there is some higher connectivity for some playlists, there are more that seem to be less connected, as indicated by the very slight right skew. Having this metric is important, because it serves as a metric for how well the recommended playlist was able to emulate the relationships between songs in the original playlist. Ideally, we would have higher values than 0.5, but we suspect that some of this is due to our subset of only about 460,000 songs. Though they have not released a specific number, in reality Spotify hosts more than a few million songs on their platform, which allows for many more complex neighborhoods to form. Were we to scale up to using every song and

playlist co-occurrence connection between them, we suspect there would be a much higher connectivity rate in the recommendations. Unfortunately, with our resources and the scope of this project, we are unable to access all of the songs Spotify has and we don't have computing resources for such a dataset.

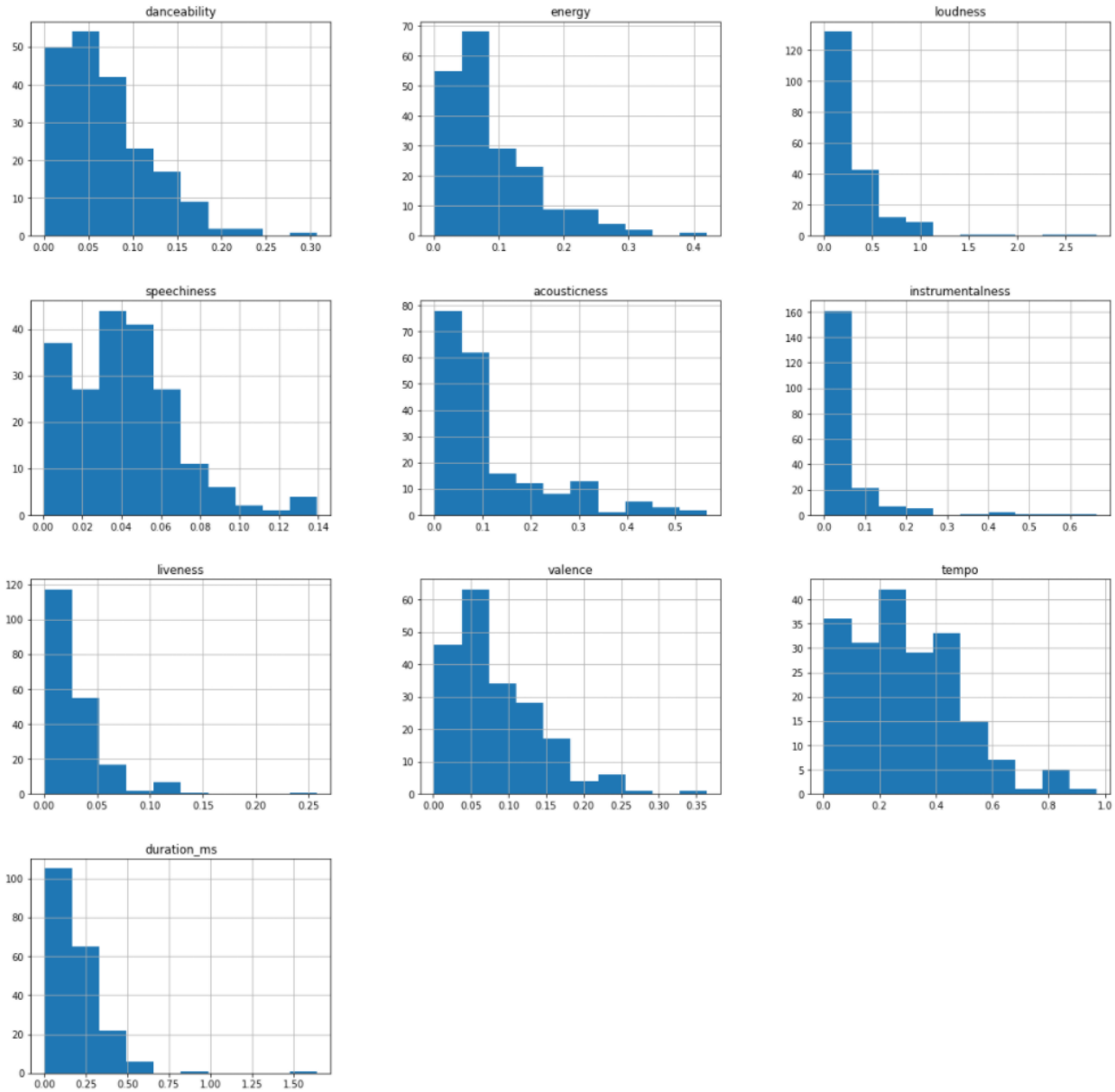
Feature Distribution in Recommendations:

The following tables and histograms are drawn from taking the variance, average, average difference of numerical features in consecutive song pairs, and difference in range from the original playlists and from the recommended sets. The differences between these measures are then taken and the resulting distribution is plotted, indicating the difference in measures for each audio feature. The table shows the average of the distribution of differences for each measure as well.

	Avg. Difference in Variances	Avg. Difference in Averages	Avg. Difference Consecutive Difference Avg.	Avg. Difference in Ranges
danceability	0.009638	0.069470	0.037568	0.152791
energy	0.014631	0.088511	0.044256	0.174784
loudness	0.241333	0.290248	0.173112	0.921693
speechiness	0.008321	0.041542	0.048691	0.185236
acousticness	0.028738	0.113215	0.074247	0.197555
instrumentalness	0.018474	0.047722	0.055689	0.288574
liveness	0.008634	0.029628	0.037779	0.149290
valence	0.010406	0.083076	0.041302	0.076147
tempo	0.287721	0.296381	0.242702	0.748015
duration_ms	0.182516	0.196597	0.142668	0.887408



Distribution of Differences in Variance



Distribution of Differences in Average

In general, we can see that there is little change in the features between the recommended song sets. For most of the variance and average differences the majority of the distribution is below 0.1, especially for the variance of danceability, energy, acousticness, valence, and liveness. There are, however, a few features that do seem to change more between the recommended song sets, such as tempo and duration. These two features are somewhat expected to have a more spread out distribution, as one would not typically expect a playlist to have songs of all very similar length and tempo. Speechiness is the least changing feature, which is a good sign, because oftentimes very speechy playlists consist of rap-heavy songs, and so the recommended songs seem to reflect that.

For future iterations of the model, it may be beneficial to try training with some of the less relevant features like duration and tempo removed. At the moment we keep them in to see if they can capture more playlist information.

Discussion

We set out to use graph based methods to create Spotify song recommendations for playlists and found promising results overall. We first looked at link prediction metrics to determine how our model performed at finding edges that should already exist between songs as co-occurrence in playlists. The significant boost in AUC score, precision, and recall in GraphSAGE over Node2Vec gave us confidence that more of the graph structure was being learned and potential recommendations as a result could be more useful going forward. The intent was to better encode the graph structure while also retaining song specific features that could traditionally help in song recommendation as we assumed similar songs had less variance in features such as acousticness, beats per minute, or danceability.

When it came to seeing the playlist recommendation results and the actual tracks that were recommended based on the top candidate tracks for each given song in a sample playlist, we had mixed outcomes. In many cases, the model was able to recommend songs in similar genres, by similar artists, and even specific albums. In other cases, however, the recommendations tended towards largely popular songs and artists without much genre overlap as we wanted. We think this could be due to the highly connected nature of the graph and the heavier weights being put on edges connected to popular songs. In the GraphSAGE neighborhood sampling, songs with few edges may be close in proximity to popular songs through our heavily connected graph structure and thus could have similar neighborhoods even though genres and style might be vastly different.

One thing we definitely would want to experiment with is different graph structures based on not only playlist co-occurrence, but also album or artist co-occurrence where edges exist between songs if they are in the same album or by the same artist. This would result in a heterogeneous graph that encodes varying weights of information in each edge. We also think that lowering the connectivity by creating edges in a more restrictive way could help with the GraphSAGE neighborhood sampling for less popular songs. Connectivity likely leads to higher importance on popular songs, especially for recommendations and we think the model could personalize better if it functioned on a more representative underlying graph.

For future improvements on obtaining node neighborhoods based on song features, we would like to incorporate a categorical feature that is representative of a song's genre. This could however be somewhat problematic, as there has always been some debate around certain songs' classification to certain genres already, and Spotify does not provide features about a song's genre. Incorporation of such a feature could yield much better results, as many themed playlists seem to be centered around one or two similar genres. Future work could make use of a

classification model for basic song genres, and utilize it in the preprocessing for playlist recommendation to add another categorical feature.

Conclusion

The business case for this approach to playlist recommendation, is that once the graph is built, it is able to add new data concerning both new nodes and new playlists. GraphSAGE is an inductive approach; re-training of the entire model would not be required. Because Spotify is ever growing, obtaining new users and new songs from artists each day, an approach like this that can easily evolve with growth is needed. Our project provides somewhat of a proof of concept that playlist recommendation with inductive graph learning approaches works, though there is room for improvement. We feel like the idea works in certain instances very well and could help music distribution services find more powerful ways to personalize for users. We tried to put emphasis on not only naively recommending songs from the same artists but also allowing discovery and different styled songs to surface as new avenues to explore.

References

[1] Nguyen Hoang, The most popular music streaming platforms in key markets globally, https://yougov.co.uk/topics/media/articles-reports/2021/03/18/services-used-stream-music-poll

[2] Spotify Million Playlist Dataset
https://www.aicrowd.com/challenges/spotify-million-playlist-dataset-challenge

[3] Spotify Web API
https://developer.spotify.com/documentation/web-api/reference/#/operations/get-audio-features

[4] A. Grover and J. Leskovec. node2vec: Scalable feature learning for networks. In KDD, 2016. https://arxiv.org/pdf/1607.00653.pdf

[5] W. L. Hamilton, R. Ying, and J. Leskovec. 2017. Inductive Representation Learning on Large Graphs. In NIPShttps://arxiv.org/pdf/1706.02216.pdf

[6] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. In ICLR, 2016. https://arxiv.org/abs/1609.02907