ARTIST SIMILARITY WITH GRAPH NEURAL NETWORKS

Filip Korzeniowski **Sergio Oramas** **Fabien Gouyon**
Pandora Media LLC., Oakland, California, USA
fkorzeniowski@pandora.com

ABSTRACT

Artist similarity plays an important role in organizing, understanding, and subsequently, facilitating discovery in large collections of music. In this paper, we present a hybrid approach to computing similarity between artists using graph neural networks trained with triplet loss. The novelty of using a graph neural network architecture is to combine the topology of a graph of artist connections with content features to embed artists into a vector space that encodes similarity. To evaluate the proposed method, we compile the new OLGA dataset, which contains artist similarities from AllMusic, together with content features from AcousticBrainz. With 17,673 artists, this is the largest academic artist similarity dataset that includes content-based features to date. Moreover, we also showcase the scalability of our approach by experimenting with a much larger proprietary dataset. Results show the superiority of the proposed approach over current state-of-the-art methods for music similarity. Finally, we hope that the OLGA dataset will facilitate research on data-driven models for artist similarity.

1. INTRODUCTION

Music similarity has sparked interest early in the Music Information Retrieval community [1,2], and has since then become a central concept for music discovery and recommendation in commercial music streaming services.

There is however no consensual notion of *ground-truth* for music similarity, as several viewpoints are relevant [2]. For instance, music similarity can be considered at several levels of granularity; musical items of interest can be musical phrases, tracks, artists, genres, to name a few. Furthermore, the perception of similarity between two musical items can focus either on (1) comparing *descriptive* (or *content-based*) aspects, such as the melody, harmony, timbre (in acoustic or symbolic form), or (2) *relational* (sometimes called *cultural*) aspects, such as listening patterns in user-item data, frequent co-occurrences of items in playlists, web pages, et cetera.

In this paper, we focus on *artist*-level similarity, and formulate the problem as a *retrieval* task: given an artist, we want to retrieve the most similar artists, where the ground-truth for similarity is *cultural*. More specifically, artist similarity is defined by music experts in some experiments, and by the "wisdom of the crowd" in other experiments.

A variety of methods have been devised for computing artist similarity, from the use of audio descriptors to measure similarity [3], to leveraging text sources by measuring artist similarity as a document similarity task [4]. A significant effort has been dedicated to the study of graphs that interconnect musical entities with semantic relations as a proxy to compute artist similarity. For instance, in [5], user profiles, music descriptions and audio features are combined in a domain specific ontology to compute artist similarity, whereas in [6], semantic graphs of artists are extracted from artist biographies.

Other approaches use deep neural networks to learn artist embeddings from heterogeneous data sources and then compute similarity in the resulting embedding space [7]. More recently, metric learning approaches trained with triplet loss have been applied to learn the embedding space where similarity is computed [8–13].

In this work, we propose a novel artist similarity model that combines graph approaches and embedding approaches using graph neural networks. Our proposed model, described in details in Sec. 2, uses content-based features (audio descriptors, or musicological attributes) together with explicit similarity relations between artists made by human experts (or extracted from listener feedback). These relations are represented in a graph of artists; the topology of this graph thus reflects the contextual aspects of artist similarity.

Our graph neural network is trained using triplet loss to learn a function that embeds artists using content features and graph connections. In this embedding space, similar artists are close to each other, while dissimilar ones are further apart.

To evaluate our approach (see Sec. 4), we compile a new dataset from publicly available sources, with similarity information and audio-based features for 17,673 artists, which we describe in Sec. 3. In addition, we evaluate the scalability of our method using a larger, proprietary dataset with more than 136,731 artists.

2. MODELLING

The goal of an artist similarity model is to define a function $s(a, b)$ that estimates the similarity of two artists—i.e., yields a large number if artist a is considered similar to artist b, and small number if not.

Many content-based methods for similarity estimation have been developed in the last decades of MIR research. The field has closely followed the state-of-the-art in machine learning research, with general improvements coming from the latter translating well into improvements in the former. Acknowledging this fact, we select our baselines based on the most recent developments: Siamese neural networks trained with variants of the triplet loss [9–13]. Building and training this type of models falls under the umbrella of *metric learning*.

2.1 Metric Learning

The fundamental idea of metric learning is to learn a projection $\mathbf{y}_v = f(\mathbf{x}_v)$ of the input features $\mathbf{x}_v$ of an item v into a new vector space; this vector space should be structured in a way such that the distances between points reflect the task at hand. In our case, we want similar artists to be close together in this space, and dissimilar artists far away.

There is an abundance of methods that embed items into a vector space, many rooted in statistics, that have been applied to music similarity [14]. In this paper, we use a neural network for this purpose. The idea of using neural networks to embed similar items close to each other in an embedding space was pioneered by [15], with several improvements developed in the following decades. Most notably, the contrastive learning objective—where two items are compared to each other as a training signal—was replaced by the *triplet loss* [16, 17]. Here, we observe three items simultaneously: the *anchor* item $\mathbf{x}_a$ is compared to a *positive* sample $\mathbf{x}_p$ and a *negative* sample $\mathbf{x}_n$. With the following loss formulation, the network is trained to pull the positive close to the anchor, while pushing the negative further away from it:

$$\mathcal{L}(t) = \left[d(\mathbf{y}_a, \mathbf{y}_n) - d(\mathbf{y}_a, \mathbf{y}_p) + \Delta\right]^+,$$

where t denotes the triplet $(\mathbf{y}_a, \mathbf{y}_p, \mathbf{y}_n)$, $d(\cdot)$ is a distance function (usually Euclidean or cosine), Δ is the maximum margin enforced by the loss, and $[\cdot]^+$ is the ramp function.

As mentioned before, state-of-the-art music similarity models are almost exclusively based on learning deep neural networks using the triplet loss. We thus adopt this method as our baseline model, which will serve as a comparison point to the graph neural network we propose in the following sections.

2.2 Graph Neural Networks

A set of artists and their known similarity relations can be seen as a graph, where the artists represent the nodes, and the similarity relations their (undirected) connections. Graph methods thus naturally lend themselves to model the artist similarity problem [6]. A particular set of graph-based models that has been gaining traction recently are *graph neural networks* (GNNs), specifically *convolutional* GNNs. Pioneered by [18], convolutional GNNs have become increasingly popular for modelling different tasks that can be interpreted as graphs. We refer the interested reader to [19] for a comprehensive and historical overview of GNNs. For brevity, we will focus on the one specific model our work is based on—the GraphSAGE model introduced by [20] and refined by [21]—and use the term GNNs for convolutional GNNs.

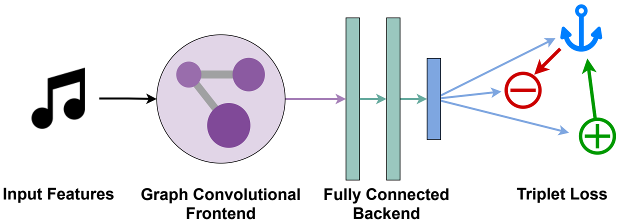

Figure 1: Overview of the graph neural network we use in this paper. First, the input features $\mathbf{x}_v$ are first passed through a *front-end* of graph convolution layers (see Sec. 2.2.2 for details); then, the output of the front-end is passed through a traditional deep neural network *back-end* to compute the final embeddings $\mathbf{y}_v$ of artist nodes. Based on these embeddings, we use the triplet loss to train the network to project similar artists (positive, green) closer to the anchor, and dissimilar ones (negative, red) further away.

2.2.1 *Model Overview*

The GNN we use in this paper comprises two parts: first, a block of *graph convolutions* (GC) processes each node's features and combines them with the features of adjacent nodes; then, another block of fully connected layers project the resulting feature representation into the target embedding space. See Fig. 1 for an overview.

We train the model using the triplet loss, in an identical setup as the baseline model. Viewing the proposed GNN from this angle, the only difference of the GNN from a standard embedding network is the additional *Graph Convolutional Frontend*. In other words, if we remove all graph convolutional layers, we arrive at our baseline model, a fully connected Deep Neural Network (DNN).

2.2.2 *Graph Convolutions*

The graph convolution algorithm, as defined in [20, 21], features two operations which are not found in classic neural networks: a *neighborhood function* $\mathcal{N}(\cdot)$, which yields the set of neighbors of a given node; and an *aggregation function*, which computes a vector-valued aggregation of a set of input vectors.

As a neighborhood function, most models use guided or uniform sub-sampling of the graph structure [20–22]. This limits the number of neighbors to be processed for each node, and is often necessary to adhere to computational limits. As aggregation functions, models commonly apply pooling operators, LSTM networks, or (weighted) pointwise averages [20].

In this work, we take a simple approach, and use pointwise weighted averaging to aggregate neighbor representations, and select the strongest 25 connections as neighbors.

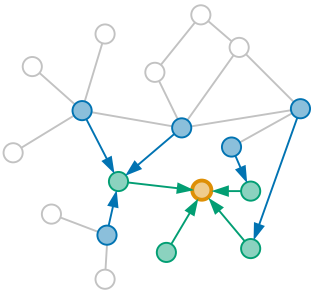

Figure 2: Tracing the graph to find the necessary input nodes for embedding the target node (orange). Each graph convolution layer requires tracing one step in the graph. Here, we show the trace for a stack of two such layers. To compute the embedding of the target node in the last layer, we need the representations from the previous layer of itself and its neighbors (green). In turn, to compute these representations, we need to expand the neighborhood by one additional step in the preceding GC layer (blue). Thus, the features of all colored nodes must be fed to the first graph convolution layer.

If weights are not available, we use the simple average of random 25 connections. This enables us to use a single sparse dot-product with an adjacency matrix to select and aggregate neighborhood embeddings. Note that this is not the full adjacency matrix of the complete graph, as we select only the parts of the graph which are necessary for computing embeddings for the nodes in a mini-batch.

Algorithm 1 describes the inner workings of the graph convolution block of our model. Here, the matrix $\mathbf{X} \in \mathbb{R}^{D \times V}$ stores the D-dimensional features of all V nodes, the symmetric sparse matrix $\mathbf{A} \in \mathbb{R}^{V \times V}$ defines the connectivity of the graph, and $\mathcal{N}(v)$ is a neighborhood function which returns all connected nodes of a given node v (here, all non-zero elements in the v^{th} row of $\mathbf{A}$).

To compute the output of a graph convolution layer for a node, we need to know its neighbors. Therefore, to compute the embeddings for a mini-batch of nodes $\mathcal{V}$, we need to know which nodes are in their joint neighborhood. Thus, before the actual processing, we first need to trace the graph to find the node features necessary to compute the embeddings of the nodes in the mini-batch. This is shown in Fig. 2, and formalized in lines 1–4 of Alg. 1.

At the core of each graph convolution layer $k \in [1 \ldots K]$ there are two non-linear projections, parameterized by projection matrices $\mathbf{Q}_k \in \mathbb{R}^{H_{Q_k} \times D}$ and $\mathbf{W}_k \in \mathbb{R}^{H_{W_k} \times (H_Q + D)}$, and a point-wise non-linear activation function σ, in our case, the Exponential Linear Unit function (ELU). Here, H_{Q_k} and H_{W_k} are the output dimensions of the respective projections. The last output, $\mathbf{X}_K \in \mathbb{R}^{H_{W_K} \times V}$, holds the l_2-normalized representations of each node in the mini-batch in its columns. It is fed into the following fully connected layers, which then compute the output embedding $\mathbf{y}_v$ of a node. Finally, these embeddings are used to compute the triplet loss and back-propagate it through the GNN.

Algorithm 1: GRAPH CONVOLUTION BLOCK

```
Input  : Node input features $\mathbf{X}$.
         Sparse connectivity matrix $\mathbf{A}$.
         Nodes in mini-batch $\mathcal{V} \subset [1 \ldots V]$.
Output: Node output representation $\mathbf{X}_K$

   ▷ Trace back input nodes for each layer.
1  $\mathcal{V}_K \leftarrow \mathcal{V}$ ;
2  for $k = K - 1 \ldots 0$ do
3      $\mathcal{V}_k \leftarrow \bigcup_{v \in \mathcal{V}_{k+1}} \mathcal{N}(v)$ ;
4  end
   ▷ Select input features for first layer. We use $\mathbf{M}[r, c]$ to
     denote selecting $r$ rows and $c$ columns from a matrix
     $\mathbf{M}$.
5  $\mathbf{X}_0 = \mathbf{X}[\cdot, \mathcal{V}_0]$ ;
6  for $k = 1 \ldots K$ do
7      $\mathbf{A}_k = \mathbf{A}[\mathcal{V}_{k-1}, \mathcal{V}_k]$ ;
8      $\mathbf{N}_k = \sigma(\mathbf{Q}_k \cdot \mathbf{X}_{k-1}) \cdot \mathbf{A}_k$ ;
9      $\mathbf{X}_k \leftarrow \sigma\left(\mathbf{W}_k \cdot \begin{bmatrix} \mathbf{N}_k \\ \mathbf{X}_{k-1}[\cdot, \mathcal{V}_k] \end{bmatrix}\right)$ ;
       ▷ $l_2$-normalize embeddings of each output node.
10     $\mathbf{X}_k \leftarrow \left[ \frac{\mathbf{x}_v}{\|\mathbf{x}_v\|_2} \mid v \in \mathcal{V}_k \right]$ ;
11 end
12 return $\mathbf{X}_K$
```

3. DATASETS

Many published studies on the topic of artist similarity are limited by data: datasets including artists, their similarity relations, and their features comprise at most hundreds to a few thousand artists. In addition, the quality of the ground truth provided is often based on 3^{rd} party APIs with obscure similarity methods like the last.fm API, rather than based on data curated by human experts.

For instance, in [6], two datasets are provided, one with ~2k artists and similarity based on last.fm relations, and another with only 268 artists, but based on relations curated by human experts. In [4], a dataset of 1,677 artists based on last.fm similarity relations is used for evaluation. Also, the dataset used in the Audio Music Similarity and Retrieval (AMS) MIREX task, which was manually curated, contains data about only 602 artists. Other works, like [8], use tag data shared among tracks or artists as a proxy for similarity estimation—which can be considered as a weak signal of similarity—and use a small set of 879 human-labeled triplets for evaluation.

For all these issues regarding existing datasets, we compiled a new dataset, the OLGA Dataset, which we describe in the following.

3.1 The OLGA Dataset

For the OLGA ("**O**h, what a **L**arge **G**raph of **A**rtists") dataset, we bring together content-based low-level features from AcousticBrainz [23], and similarity relations from AllMusic. Assembling the data works as follows:

1. Select a common pool of artists based on the unique artists in the Million Song Dataset [24].

2. Map the available MusicBrainz IDs of the artists to AllMusic IDs using mapping available from MusicBrainz.

3. For each artist, obtain the list of "related" artists from AllMusic; this data can be licensed and accessed on their website. Use only related artists which can be mapped back to MusicBrainz.

4. Using MusicBrainz, select up to 25 tracks for each artist using their API, and collect the low-level features of the tracks from AcousticBrainz.

5. Compute the track feature centroid of each artist.

In total, the dataset comprises 17,673 artists connected by 101,029 similarity relations. On average, each artist is connected to 11.43 other artists. The quartiles are at 3, 7, and 16 connections per artist. The lower 10% of artists have only one connection, the top 10% have at least 27.

While the dataset size is still small compared to industrial catalog sizes, it is significantly bigger than other datasets available for this task. Its size and available features will allow us to apply more data-driven machine learning methods to the problem of artist similarity. [1]

For our experiments, we partition the artists following an 80/10/10 split into 14,139 training, 1767 validation, and 1767 test artists.

3.2 Proprietary Dataset

We also use a larger proprietary dataset to demonstrate the scalability of our approach. Here, explicit feedback from listeners of a music streaming service is used to define whether two artists are similar or not.

For artist features, we use the centroid of an artist's track features. These track features are *musicological* attributes annotated by experts, and comprise hundreds of content-based characteristics such as "amount of electric guitar", or "prevalence of groove".

In total, this dataset consists of 136,731 artists connected by 3,277,677 similarity relations. The number of connections per artists is a top-heavy distribution with few artists sharing most of the connections: the top 10% are each connected to more than 134 others, while the bottom 10% to only one. The quartiles are at 2, 5, and 48 connections per artist.

We follow the same partition strategy as for the OLGA dataset, which results in 109,383 training, 13,674 validation, and 13,674 test artists.

4. EXPERIMENTS

Our experiments aim to evaluate how well the embeddings produced by our model capture artist similarity. To this end, we set up a ranking scenario: given an artist, we collect its K nearest neighbors sorted by ascending distance, and evaluate the quality of this ranking. To quantify this, we use normalized discounted cumulative gain [25] with a high cut-off at $K = 200$ ("ndcg@200"). We prefer this metric over others, because it was shown that at high cut-off values, it provides better discriminative power, as well as robustness to sparsity bias (and, to moderate degree, popularity bias) [26]. Formally, given an artist a with an ideal list of similar artists s (sorted by relevance), the nDCG_K of a predicted list of similar artists $\hat{\mathrm{s}}$ is defined as:

$$\mathrm{nDCG_K}(a, \hat{\mathbf{s}}, \mathbf{s}) = \frac{\sum_{k=1}^{K} g(\hat{s}_k, a) d(k)}{\sum_{k=1}^{K} g(s_k, a) d(k)},$$

where $g(\cdot, a)$, the *gain*, is 1 if an artist is indeed similar to a, and 0 otherwise, and $d(k) = \log_2^{-1}(k+1)$ the *discounting* factor, weights top rankings higher than the tail of the list.

In the following, we first explain the models, their training details, the features, and the evaluation data used in our experiments. Then, we show, compare and analyze the results.

4.1 Models

As explained in Sec. 2.2.1, a GNN with no graph convolutional layers is identical to our baseline model (i.e. a DNN trained using triplet loss). This allows us to fixate hyper-parameters between baseline and the proposed GNN, and isolate the effect of adding graph convolutions to the model. For each dataset, we thus train and evaluate four models with 0 to 3 graph convolutional layers.

The other hyper-parameters remain fixed: each layers in the graph convolutional front-end consists of 256 ELUs [27]; the back-end comprises two layers of 256 ELUs each, and one linear output layer with a 100 dimensions; we train the networks using the ADAM optimizer [28] with a linear learning-rate warm-up [29] for the first epoch, and following a cosine learning rate decay [30] for the remaining 49 epochs (in contrast to [30], we do not use warm-restarts); for selecting triplets, we apply distance-weighted sampling [31], and use a margin of $\Delta = 0.2$ in the loss; finally, as distance measure, we use Euclidean distance between l_2-normalized embeddings.

We are able to train the largest model with 3 graph convolutional layers within 2 hours on the proprietary dataset, and under 5 minutes on OLGA, using a Tesla P100 GPU and 8 CPU threads for data loading.

4.2 Features

We build artist-level features by averaging track-level features of the artist's tracks. Depending on the dataset, we have different types of features at hand.

In the OLGA dataset, we have low-level audio features as extracted by the Essentia library. [2] These features represent track-level statistics about the loudness, dynamics and spectral shape of the signal, but they also include more abstract descriptors of rhythm and tonal information, such as bpm and the average pitch class profile. We select all numeric features and pre-process them as follows: we apply

[1] The procedure to assemble the dataset, including relevant metadata, is available on https://gitlab.com/fdlm/olga/.

[2] See https://essentia.upf.edu/streaming_extractor_music.html#music-descriptors

element-wise standardization, discard features with missing values, and flatten all numbers into a single vector of 2613 elements.

In the proprietary dataset, we use numeric musicological descriptors annotated by experts (for example, "the nasality of the singing voice"). We apply the same preprocessing for these, resulting in a total of 170 values.

Using two different types of content features gives us the opportunity to evaluate the utility of our graph model under different circumstances, or more precisely, features of different quality and signal-to-noise ratio. The low-level audio-based features available in the OLGA dataset are undoubtedly noisier and less specific than the high-level musical descriptors manually annotated by experts, which are available in the proprietary dataset. Experimenting with both permits us to gauge the effect of using the graph topology for different data representations.

In addition, we also train models with *random vectors* as features. For each artist, we uniformly sample a random vector of the same dimension as the real features, and and keep it constant throughout training and testing. This way, we can differentiate between the performance of the real features and the performance of using the graph topology in the model: the results of a model with no graph convolutions is only due to the features, while the results of a model with graph convolutions but random features is only due to the usage of the graph topology.

4.3 Evaluation Data

As described in Section 3, we partition artists into a training, validation and test set. When evaluating on the validation or test sets, we only consider artists from these sets as candidates and potential true positives. Specifically, let $\mathcal{V}_{\text{eval}}$ be the set of evaluation artists, we only compute embeddings for those, and retrieve nearest neighbors from this set, and only consider ground truth similarity connections within $\mathcal{V}_{\text{eval}}$.

This notion is more nuanced in the case of GNNs. Here, we want to exploit the *known artist graph topology* (i.e., which artists are connected to each other) when computing the embeddings. To this end, we use all connections between artists in $\mathcal{V}_{\text{train}}$ (the training set) *and* connections between artists in $\mathcal{V}_{\text{train}}$ and $\mathcal{V}_{\text{eval}}$. This process is outlined in Fig. 3.

Note that this does not leak information between train and evaluation sets; the features of evaluation artists have not been seen during training, and connections within the evaluation set—these are the ones we want to predict—remain hidden.

4.4 Results

Table 1 compares the baseline model with the proposed GNN. We can see that the GNN easily out-performs the DNN. It achieves an NDCG@200 of 0.55 vs. 0.24 on the OLGA dataset, and 0.57 vs. 0.44 on the proprietary dataset. The table also demonstrates that the graph topology is more predictive of artist similarity than content-based features: the GNN, using random features, achieves better results than a DNN using informative features for both datasets (0.45 vs. 0.24 on OLGA, and 0.52 vs 0.44 on the proprietary dataset).

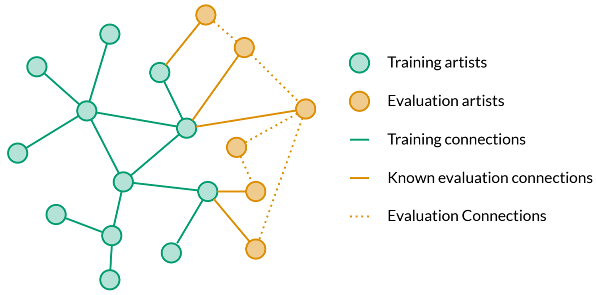

Figure 3: Artist nodes and their connections used for training (green) and evaluation (orange). During training, only green nodes and connections are used. When evaluating, we extend the graph with the orange nodes, but only add connections between validation and training artists. Connections among evaluation artists (dotted orange) remain hidden. We then compute the embeddings of all evaluation artists, and evaluate based on the hidden evaluation connections.

Dataset	Features	DNN	GNN
OLGA	Random	0.02	0.45
	AcousticBrainz	0.24	0.55
Proprietary	Random	0.00	0.52
	Musicological	0.44	0.57

Table 1: NDCG@200 for the baseline (DNN) and the proposed model with 3 graph convolution layers (GNN), using features or random vectors as input. The GNN with real features as input gives the best results. Most strikingly, the GNN with random features—using only the known graph topology—out-performs the baseline DNN with informative features.

Additionally, the results indicate—perhaps to little surprise—that low-level audio features in the OLGA dataset are less informative than manually annotated high-level features in the proprietary dataset. Although the proprietary dataset poses a more difficult challenge due to the much larger number of candidates (14k vs. 1.8k), the DNN—which can only use the features—improves more over the random baseline in the proprietary dataset (+0.44), compared to the improvement (+0.22) on OLGA. These are only indications; for a definitive analysis, we would need to use the exact same features in both datasets.

Similarly, we could argue that the topology in the proprietary dataset seems more coherent than in the OLGA dataset. We can judge this by observing the performance gain obtained by a GNN with random feature—which can only leverage the graph topology to find similar artists—compared to a completely random baseline (random features without GC layers). In the proprietary dataset, this performance gain is +0.52, while in the OLGA dataset,

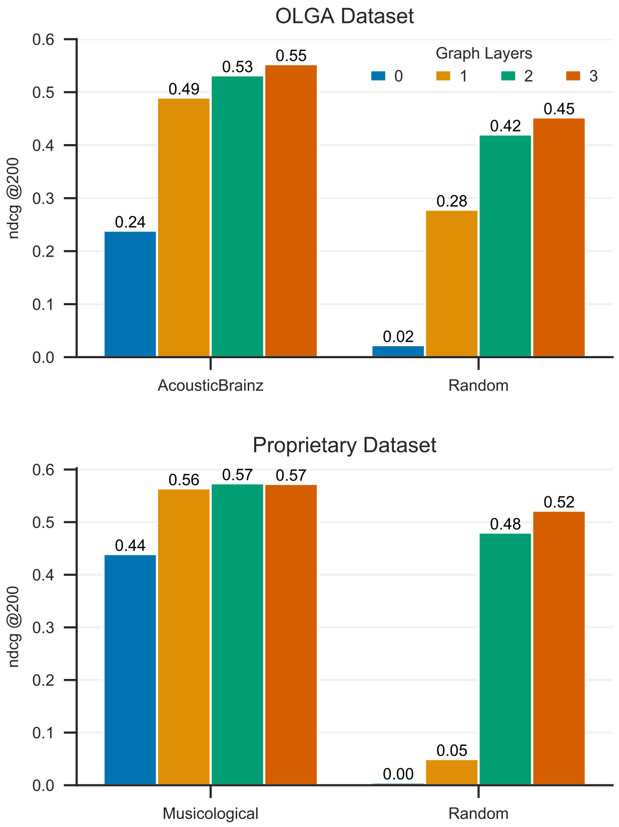

Figure 4: Results on the OLGA (top) and the proprietary dataset (bottom) with different numbers of graph convolution layers, using either the given features (left) or random vectors as features (right).

only +0.43. Again, while this is not a definitive analysis (other factors may play a role), it indicates that the large amounts of user feedback used to generate ground truth in the proprietary dataset give stable and high-quality similarity connections.

Figure 4 depicts the results for each model and feature set depending on the number of graph convolutional layers used. (Recall that a GNN with 0 graph convolutions corresponds to the baseline DNN.) In the OLGA dataset, we see the scores increase with every added layer. This effect is less pronounced in the proprietary dataset, where adding graph convolutions does help significantly, but results plateau after the first graph convolutional layer. We believe this is due to the quality and informativeness of the features: the low-level features in the OLGA dataset provide less information about artist similarity than high-level expertly annotated musicological attributes in the proprietary dataset. Therefore, exploiting contextual information through graph convolutions results in more uplift in the OLGA dataset than in the proprietary one.

Looking at the scores obtained using random features (where the model depends solely on exploiting the graph topology), we observe two remarkable results. First, whereas one graph convolutional layer suffices to outperform the feature-based baseline in the OLGA dataset (0.28 vs. 0.24), using only one GC layer does not produce meaningful results (0.05) in the proprietary dataset. We believe this is due to the different sizes of the respective test sets: 14k in the proprietary dataset, while only 1.8k in OLGA. Using only a very local context seems to be enough to meaningfully organize the artists in a smaller dataset.

Second, most performance gains are obtained with two GC layers, while adding the third GC layer pushes the results to a much lesser degree. Our explanation for this effect is that most similar artists are connected through at least one other, common artist. In other words, most artists form similarity cliques with at least two other artists. Within these cliques, in which every artist is connected to all others, missing connections are easily retrieved by no more than 2 graph convolutions.

In fact, in the OLGA dataset, ~71% of all cliques fulfill this requirement. This means that, for any hidden similarity link in the data, in 71% of cases, the true similar artist is within 2 steps in the graph—which corresponds to using two GC layers.

5. SUMMARY AND FUTURE WORK

In this paper, we described a hybrid approach to computing artist similarity, which uses graph neural networks to combine content-based features with explicit relations between artists. To evaluate our approach, we assembled a novel academic dataset with 17,673 artists, their features, and their similarity relations. Additionally, we used a much larger proprietary dataset to show the scalability of our method. The results showed that leveraging known similarity relations between artists can be more effective for understanding their similarity than high-quality features, and that combining both gives the best results.

Our work is a first step towards models that directly use known relations between musical entities—like tracks, artists, or even genres—or even across these modalities. Multi-modal connections could also help predicting artist similarity; we could add collaborations, or band membership connections to the graph. Finally, it would be interesting to analyze the effect of our approach on long-tail recommendations and/or the cold-start problem.

6. REFERENCES

[1] J.-J. Aucouturier and F. Pachet, “Music Similarity Measures: What’s The Use?” in *Proc. of the 3rd International Conference on Music Information Retrieval (ISMIR)*, Paris, France, Oct. 2002.

[2] D. P. W. Ellis, B. Whitman, A. Berenzweig, and S. Lawrence, “The Quest for Ground Truth in Musical Artist Similarity,” in *Proc. of the International Symposium on Music Information Retrieval (ISMIR)*, Paris, France, Oct. 2002.

[3] T. Pohle, D. Schnitzer, M. Schedl, P. Knees, and G. Widmer, “On Rhythm and General Music Similarity,” in *Proc. of the 10th International Society for Music Information Retrieval Conference (ISMIR)*, Kobe, Japan, Oct. 2009.

[4] M. Schedl, D. Hauger, and J. Urbano, “Harvesting microblogs for contextual music similarity estimation: A co-occurrence-based framework,” *Multimedia Systems*, vol. 20, no. 6, pp. 693–705, Nov. 2014.

[5] Ò. Celma and X. Serra, “FOAFing the music: Bridging the semantic gap in music recommendation,” *Journal of Web Semantics*, vol. 6, no. 4, pp. 250–256, Nov. 2008.

[6] S. Oramas, M. Sordo, L. Espinosa-Anke, and X. Serra, “A Semantic-Based Approach for Artist Similarity,” in *Proc. of the 16th International Society for Music Information Retrieval Conference (ISMIR)*, Málaga, Spain, Oct. 2015.

[7] B. McFee and G. R. G. Lanckriet, “Heterogeneous Embedding For Subjective Artist Similarity,” in *Proc. of the 10th International Society for Music Information Retrieval Conference (ISMIR)*, Kobe, Japan, Oct. 2009.

[8] J. Lee, N. J. Bryan, J. Salamon, Z. Jin, and J. Nam, “Disentangled Multidimensional Metric Learning for Music Similarity,” in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Barcelona, Spain, May 2020.

[9] G. Doras, F. Yesiler, J. Serrà, E. Gómez, and G. Peeters, “Combining Musical Features for Cover Detection,” in *Proc. of the 21st International Society for Music Information Retrieval Conference (ISMIR)*, Montréal, Canada, Oct. 2020.

[10] J. Lee, N. J. Bryan, J. Salamon, Z. Jin, and J. Nam, “Metric Learning vs Classification for Disentangled Music Representation Learning,” in *Proc. of the 21st International Society for Music Information Retrieval Conference (ISMIR)*, Montréal, Canada, Aug. 2020.

[11] J. Park, J. Lee, J. Park, J.-W. Ha, and J. Nam, “Representation Learning of Music Using Artist Labels,” in *Proc. of the 19th International Society for Music Information Retrieval Conference (ISMIR)*, Paris, France, Jun. 2018.

[12] F. Yesiler, J. Serrà, and E. Gómez, “Accurate and Scalable Version Identification Using Musically-Motivated Embeddings,” in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Barcelona, Spain, May 2020.

[13] M. Dorfer, A. Arzt, and G. Widmer, “Learning Audio-Sheet Music Correspondences for Score Identification and Offline Alignment,” in *Proc. of the 18th International Society for Music Information Retrieval Conference (ISMIR)*, Suzhou, China, Jul. 2017.

[14] M. Slaney, K. Q. Weinberger, and W. White, “Learning a Metric for Music Similarity,” in *Proc. of the 9th International Conference on Music Information Retrieval (ISMIR)*, Philadelphia, USA, Sep. 2008.

[15] J. Bromley, I. Guyon, Y. LeCun, E. Säckinger, and R. Shah, “Signature verification using a "Siamese" time delay neural network,” in *Proc. of the 6th International Conference on Neural Information Processing Systems (NIPS)*, San Francisco, USA, Nov. 1993.

[16] E. Hoffer and N. Ailon, “Deep Metric Learning Using Triplet Network,” in *Similarity-Based Pattern Recognition (SIMBAD)*, A. Feragen, M. Pelillo, and M. Loog, Eds., Copenhagen, Denmark, Oct. 2015.

[17] J. Wang, Y. Song, T. Leung, C. Rosenberg, J. Wang, J. Philbin, B. Chen, and Y. Wu, “Learning Fine-Grained Image Similarity with Deep Ranking,” in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Columbus, USA, Jun. 2014.

[18] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, “Spectral Networks and Locally Connected Networks on Graphs,” in *Proc. of the International Conference on Learning Representations (ICLR)*, Banff, Canada, Apr. 2014.

[19] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, “A Comprehensive Survey on Graph Neural Networks,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 1, pp. 4–24, Jan. 2021.

[20] W. L. Hamilton, R. Ying, and J. Leskovec, “Inductive Representation Learning on Large Graphs,” in *Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS)*, Long Beach, USA, Dec. 2017.

[21] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec, “Graph Convolutional Neural Networks for Web-Scale Recommender Systems,” in *Proc. of the 24th International Conference on Knowledge Discovery & Data Mining (SIGKDD)*, Jul. 2018.

[22] J. Oh, K. Cho, and J. Bruna, “Advancing GraphSAGE with A Data-Driven Node Sampling,” in *Proc. of the ICLR Workshop on Representation Learning on Graphs and Manifolds*, New Orleans, USA, May 2019.

[23] A. Porter, D. Bogdanov, R. Kaye, R. Tsukanov, and X. Serrà, “AcousticBrainz: A Community Platform for Gathering Music Information Obtained from Audio,” in *Proc. of the 16th Conference of the International Society for Music Information Retrieval (ISMIR)*, Málaga, Spain, Oct. 2015.

[24] T. Bertin-Mahieux, D. P. W. Ellis, B. Whitman, and P. Lamere, “The Million Song Dataset,” in *Proc. of the 12th International Society for Music Information Retrieval Conference (ISMIR)*, A. Klapuri and C. Leider, Eds., Miami, USA, Oct. 2011.

[25] K. Järvelin and J. Kekäläinen, “Cumulated gain-based evaluation of IR techniques,” *ACM Transactions on Information Systems*, vol. 20, no. 4, pp. 422–446, Oct. 2002.

[26] D. Valcarce, A. Bellogín, J. Parapar, and P. Castells, “On the robustness and discriminative power of information retrieval metrics for top-N recommendation,” in *Proc. of the 12th ACM Conference on Recommender Systems (RECSYS)*, Vancouver, Canada, Oct. 2018.

[27] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, “Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs),” in *Proc. of the International Conference on Learning Representations (ICLR)*, San Juan, Puerto Rico, May 2016.

[28] D. P. Kingma and J. Ba, “Adam: A Method for Stochastic Optimization,” in *Proc. of the International Conference on Learning Representations (ICLR)*, San Diego, USA, May 2015.

[29] J. Ma and D. Yarats, “On the adequacy of untuned warmup for adaptive optimization,” in *Proc. of the 35th Conference on Artificial Intelligence (AAAI)*, Virtual conference, Feb. 2021.

[30] I. Loshchilov and F. Hutter, “SGDR: Stochastic Gradient Descent with Warm Restarts,” in *Proc. of the International Conference on Learning Representations (ICLR)*, Toulon, France, May 2017.

[31] C.-Y. Wu, R. Manmatha, A. J. Smola, and P. Krähenbühl, “Sampling Matters in Deep Embedding Learning,” in *Proc. of the International Conference on Computer Vision (ICCV)*, Venice, Italy, Oct. 2017.