GRAPH CONVOLUTIONAL NETWORK FOR MUSIC RECOMMENDER

Longshen Ou
School of Computing, National University of Singapore
oulongshen@u.nus.edu.sg

1. MOTIVATION AND DESCRIPTION

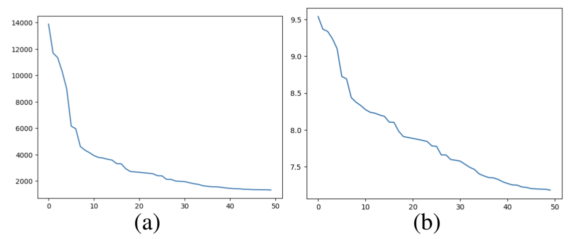
In recent years, music recommendation systems have emerged as essential tools to help users navigate the vast landscape of digital music content. Graph Convolutional Networks (GCNs) [1] offer a promising approach for addressing this challenge by leveraging the inherent graph structure present in user-artist listening event data. In this project, we employ GCNs to develop a music artist recommendation system that considers the relationships between users and artists in order to model their identities. By predicting potential affinity scores between users and unseen artists, our system effectively recommends artists that users are most likely to enjoy.

2. DATA PREPARATION

2.1 Data Acquisition

We utilize the LastFM dataset [2] for our project. This dataset comprises social networking, tagging, and music artist listening information for a set of 2,000 users from the Last.fm online music system. We focus exclusively on the user-artist listening event information within the dataset, which contains 1,892 users, 17,632 artists, and 92,834 user-artist pairs with diverse listening event counts.

2.2 Data Exploration and Preparation



(a) (b)

Figure 1: Playing event count of a randomly selected user with all artists he/she listen to. (a) raw count; (b) after log operation.

Since all edges are between user-artist pairs, it is not necessary to differentiate between the user and artist node types. As a result, we convert the artist-user listening event data into an undirected homogeneous graph with a normalized *liking score* serving as the edge weight. Normalization is essential here because the playing event counts for each user-artist relationship vary across a wide range. Moreover, the raw user-artist playing counts are typically not evenly distributed and follow a long-tail distribution, as shwon in Fig. 1a. If we were to use even the scaled version of the raw counts directly as the liking score, most artists would receive a low score. To reduce the influence of the most liked artists to each user, we apply a logarithmic function to the listening count, as in Fig. 1b, and then normalize it by dividing with the maximum weight of all its edges of that user node.

3. GML SOLUTION

3.1 Problem formulation

We approach this problem as an edge regression task. Given the undirected homogeneous graph g, we would like our model to compute a hidden representation h of each node from the graph information, and predict the weight of edges e_{ij} that are unseen in training.

3.2 Baseline

We use GCN without edge embeddings as the baseline model, with the node update equation as follows:

$$h_i^{l+1} = \eta(\frac{1}{d_i}\sum_{j\in N_i} A_{ij}W^l h_j^l) \tag{1}$$

where A represents the adjacency matrix, W denotes the model weight for convolution, h is node embedding, d stands for node degree, and η is the learning rate.

3.3 Improvement: Edge as Aggregation Weight

The edge weight in our problem, the liking score, naturally conveys the level of connection strength between users and artists. This can serve as an inductive bias in our GCN model. We incorporate the edge weight into the summation of update equations to guide the message passing, resulting in the following equation:

$$h_i^{l+1} = \eta(\frac{1}{d_i}\sum_{j\in N_i} A_{ij}w_{ij}W^l h_j^l) \tag{2}$$

where w_{ij} represents the edge weight.

3.4 Improvement: Augment Weight with Attention

Are edges with the same score equal? Not necessarily. Imagine two users u1, u2 and two rock stars a1, a2. Both the liking scores of u1-a1 and u2-a2 are 0.9. However, user u1 only likes rock and never listens to music from other genres, while u2 has a broad taste—he/she listens to all types of music equally often. In this case, the impact of a1 on a2 is probably different: a2 stands out from various artists, meaning he/she tends to have a broader impact and greater audience; a1 may be less popular but should be distinctive, likely having a small but passionate fan base. Overall, even the same edge value (liking score) may convey different information.

The original message passing of GCN in Eq. 2 does not consider this: it treats all edges with the same score exactly the same way, using that score as the weight for aggregation. To calibrate the edge weight with node similarity (attention), the update equation is modified as follows:

$$\mathbf{a}^j = Softmax(U(Vh_j||Vh_k)) \tag{3}$$

$$h_i^{l+1} = \eta(\frac{1}{d_i}\sum_{j\in N_i} A_{ij}(w_{ij} + \mathbf{a}_j^j)W^l h_j^l) \tag{4}$$

where $||$ is the concatenation operator, U and V are additional weight matrices, a_j is the attention weight vector with length equal to the number of nodes N, and $\mathbf{a}^j$ is the attention weight between node j and all its adjacent nodes.

3.5 Improvement: Data Augmentation

In the preliminary experiment, we noticed that it was difficult for the model to overfit the training data to reach a very small training loss, even when using a large hidden dimension. We suspect that the scale of samples in the training set is insufficient to support the complexity of this task, but there is no easy way to simulate new user-artist pairs as additional training samples.

In real world scenario, there might be situations where a user's number of play events does not exactly correspond to their degree of liking. Perhaps they accidentally played a song by a disliked artist for some reason, or maybe their most liked artist has fewer play counts because the user tends to watch the artist on YouTube instead of listening on Last.fm. Therefore, augmenting the training data by adding additional randomness is a feasible approach that increase the diversity of the training set while maintaining the underlying structure of data.

Our strategy is to add noise to the edge weights in the graph. We achieve this by adding a noise factor in the update equation, so Eq. 4 becomes:

$$h_i^{l+1} = \eta(\frac{1}{d_i}\sum_{j\in N_i} A_{ij}(w_{ij} + \mathbf{a}_j^j + n)W^l h_j^l) \tag{5}$$

where n is a noise factor sampled from a normal distribution $n \sim \mathcal{N}(\mu = 0, \sigma = 0.1)$.

4. EXPERIMENTS AND RESULTS

4.1 Data Preprocessing

The dataset is split with a ratio of 8:1:1 for training, validation, and testing. The size of the graph makes full batch training difficult. To overcome this issue, we adopt a stochastic subgraph sampler [1] for training. In each training step, the algorithm first samples a desired number of edges and includes all neighboring nodes of both vertices of the edge to the subgraph. Additionally, a bipartite graph is created for each layer of GCN, including nodes and edges necessary for message-passing computation.

[1] https://docs.dgl.ai/generated/dgl.dataloading.MultiLayerFullNeighborSampler.html

Models	Test MSE	Test MAE
Naive GCN	0.0271	0.1254
Edge as weight	0.0210	0.1094
+ Attention	0.0185	0.1033
+ Augmentation	0.0163	0.0957

Table 1: Performance comparison of different GCNs.

4.2 Hyperparameter setting

For all experiments, the number of convolutional layers is set to 2, batch size (number of edges in the sampled subgraph) to 4096, learning rate 0.01, hidden dimension 64, and trained for 20 epochs. We compute mean square error as loss and use mean absolute error to evaluate performance. The learning rate is scheduled with exponential decay for each epoch, with the decay ratio for each model being the best one among [0.7, 0.8, 0.9, 1.0].

4.3 Results

The results are shown in Table 1. The baseline model has a decent performance with 0.1254 MAE. Moreover, each of our improvement in Section 3 contributes to the performance, with both lower testing loss and lower mean absolute error. The edge weight, additional attention, and data augmentation achieve 0.0160, 0.0061, and 0.0076 reductions in mean absolute error, respectively. Surprisingly, the augmentation method, despite its simplicity, is very effective in achieving performance gains.

5. FUTURE DEVELOPMENT

In future developments, we plan to explore more advanced GNN architectures, such as GraphSAGE and Gated GCN, to capture more complex interactions between users and artists. Additionally, we will investigate the potential of incorporating additional information provided in the dataset, such as artist metadata and user-user connections, which were not utilized in this study, to enhance recommendation quality. Furthermore, we are also interested in exploring whether introducing edge embeddings and edge updates can lead to performance gains.

6. REFERENCES

[1] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *International Conference on Learning Representations*.

[2] I. Cantador, P. Brusilovsky, and T. Kuflik, "2nd workshop on information heterogeneity and fusion in recommender systems (hetrec 2011)," in *Proceedings of*

the 5th ACM conference on Recommender systems, ser. RecSys 2011. New York, NY, USA: ACM, 2011.