Gender Classification of Deezer Europe Users with Graph Neural Networks

Xinyu Liu
EPFL
xinyu.liu@epfl.ch

Weijiang Xiong
EPFL
weijiang.xiong@epfl.ch

***Abstract*—In this project, we investigated the Deezer Europe social network dataset. We first explored the network, analyzed its various properties and established that it is a scale-free network. We then performed node embedding to transform the nodes to Euclidean space for downstream machine learning tasks. Furthermore, we implemented various combinations of node embedding methods with downstream neural network classifiers for a node classification task. Our best model and embedding method reached an ROC AUC score of 0.725. Our codes are publicly available at https://github.com/Weijiang-Xiong/NML23-Project.**

***Index Terms*—Gender Classification, Graph Neral Networks, Deep Learning, Feature Embedding**

I. Network Exploration

A. Basic Network Properties

The Deezer Europe social network dataset [1] is a social network of Deezer users from European countries and edges are mutual follower relationships between them. The vertex features are extracted based on the artists liked by the users. The task related to the graph is binary node classification - one has to predict the gender of users. The target features was derived from the name field for each user.

In this part, we first performed an explorative analysis of the dataset to better understand the general structure of the graph. We calculated some important network features which are listed in Table I. As can be seen from the table, the network is relatively large with more than 30,000 nodes but also sparse with a density of 0.0002, similar to most real networks. Importantly, the network does not exhibit small-world behaviors since it has a long diameter and low clustering coefficient compared to most real networks. A visualization of the network is shown in Figure 1.

TABLE I: Basic properties of the network.

Nodes	Edges	Density	Clustering Coefficient	Diameter	Features	Classes
28281	92752	0.00023	0.141	21	31240	2

B. Degree Distribution

We plotted the degree distribution of the network in Figure 2. Based on observation, the degree distribution can be fitted by a power-law distribution:

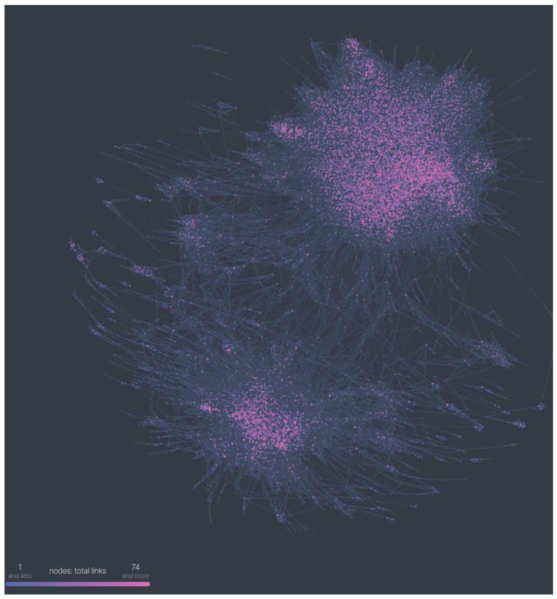

Fig. 1: Visualization of the Deezer Europe social network created using Cosmograph (https://cosmograph.app/), beset viewed in color.

$$Y = kX^{\alpha}$$

where X and Y are variables of interest, k is a constant, and α is the law's exponent. Based on experiments, we chose $k = 0.5$ and $\alpha = -1$ for our network. The fitted power-law distribution is also plotted in Figure 2. As can be seen from the figure, power-law distribution can provide a good fit to the degree distribution of the original network.

C. Network Modelling

To better understand the behavior of the network, we can use a network model to capture its degree distribution. Since the degree distribution exhibit a scale-free pattern, we choose the Barabási–Albert model [2]. The Barabási–Albert (BA) model is random network model used for generating scale-free networks using a preferential attachment mechanism, where a graph of n nodes is grown by attaching new nodes each with

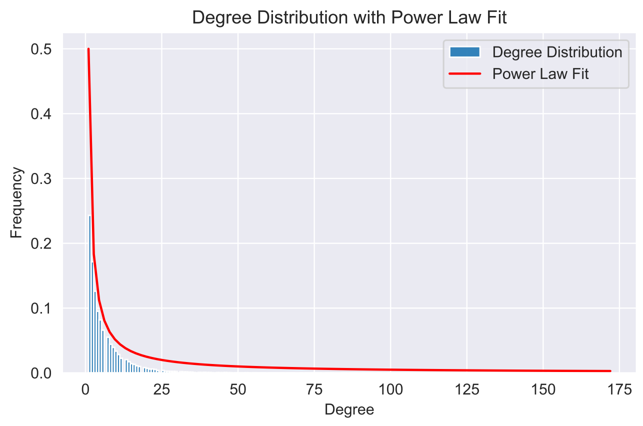

Fig. 2: Degree distribution of the Deezer Europe social network and power-law distribution fitting.

m edges that are preferentially attached to existing nodes with high degree.

In our experiment, the parameter m is chosen based on the average number of edges in the original network so that the generated BA graph has approximately the same number of edges as in the graph we want to simulate. Therefore, m is chosen to be half of the mean degree of the original network. Simulated degree distribution is plotted in Figure 3. As can be seen from the figure, the BA model realized an ideal fit for the degree distribution, but has more highly-connected nodes compared to the original network. This is due to the original network does not exactly follow the scale-free distribution with the absence of highly-connected nodes.

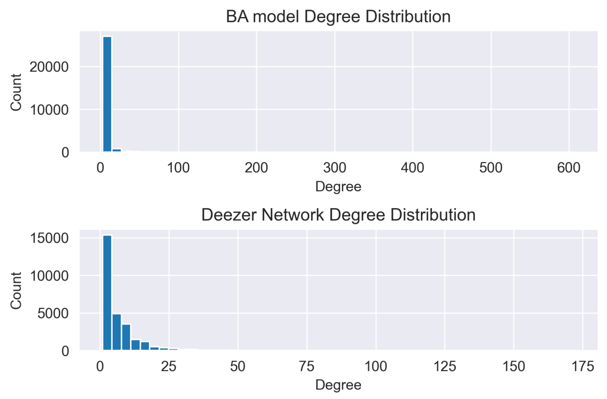

Fig. 3: Degree distribution of the Deezer Europe social network and a random network generated using BA model.

D. Gender Ratio

The task related to this dataset is to predict users' genders based on their mutual followership and artists followed. Based on a label analysis, the proportion of genders in users is $0.56/0.44$, thus a balanced scnerio for our binary-classification task.

E. Popular Artists

Figure 4 shows the distribution of follower counts, where a vast majority of artists only have a small group of followers. Still, there are a few popular artists, which are not obvious in the histogram. Table II shows the IDs of the six most popular artists, each of whom has more than three thousand followers.

TABLE II: Artists with more than 3k followers

Artist ID	505	12	251	675	21342	87
#Follower	4693	3777	3401	3376	3311	3013

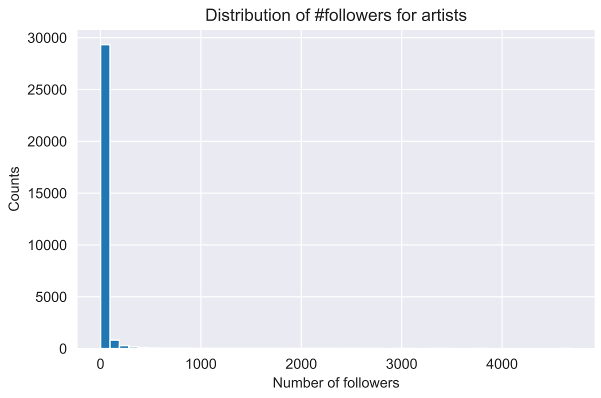

Fig. 4: The distribution of artists' follower counts

II. Gender Prediction

In this section, we evaluate the quality of various node embedding methods using the corresponding gender prediction performance of two basic graph neural network models.

A. Node Feature Embeddings

The Deezer Europe Dataset [1] only contains a single graph, where the nodes are users and edges are mutual following relationship. For the gender prediction task, we can utilize both the structure of the graph, and the node features, i.e., the liked artists of users. While the graph structure is clear enough, its raw features doesn't suit a neural network directly because each user may like different number of artists, making the feature vectors vary in lengths. On the other hand, the features (liked artists) can reveal the users' preference and personalities, and contain valuable information for gender prediction. Therefore, providing adequate feature embedding is essential in for the success of gender prediction task.

Concretely, we consider Binary and Feather [1] methods. The **Binary** embedding uses a vector with binary element values to represent the liked artists of a user. Formally, the Binary feature embedding of the i-th user is defined by:

$$\mathbf{x}_i = [b_i^0, b_i^1, ..., b_i^M], \quad b_i^k = 1 \quad \text{if } k \in \mathcal{A}_i, \tag{1}$$

where M is the number of artists ($\sim$31k). $\mathcal{A}_i$ represents the set of liked artists of the i-th user, and b_i^k will be 1 if artist k is in $\mathcal{A}_i$ and 0 otherwise.

The **Feather** [1] method uses a set of characteristic functions to compute node embeddings, and it will make use of both the graph structure and node features. We refer to the original paper for a complete tour of the mathematical derivation, and summarize the main idea as follows. Firstly, take the Binary embedding (matrix) of all nodes, and use Singular Value Decomposition (SVD) to reduce the feature dimension to 128. Secondly, for the embedding of each node, separately multiply the embedding vector with the elements from a arithmetic sequence, which starts from 0.01 to 2.5 and contains 25 elements in total. Thridly, concatenate the 25 different 128D vectors into a 3200D vector. After that, compute the element-wise sin and cos for the 3200D vector and concatenate them, resulting in a 6400D vector. Finally, recurrently multiply the 6400D vector with the normalized adjacency matrix $D^{-1}A$ for 5 times, record all intermediate outputs and concatenate them all, ending up with a 32,000D node embedding vector.

Both of the two methods produce high-dimensional node embeddings, which may not be applicable in some cases. As such, we can consider making them *lighter* by reducing their dimensions with SVD. Besides, Feather embedding may have an advantage over Binary, because it utilizes both graph structure and node features, while Binary embedding only uses node features. Therefore, we can combine Node2Vec [3] embeddings and Binary embeddings to compose a joint embedding with information from both graph structure and node features. We use an opensource implementation of Node2Vec [4], the Feather implementation from Karate Club [5] and the SVD implementation from Sklearn [6].

B. Model Structure

As shown in Figure 5, we use a feed-forward architecture with two blocks of *conv-relu-dropout* and a trailing *conv* layer to output predictions. Our implementation is based on Pytorch Geometric [7], and we create two variants of this architecture by implmenenting the graph convolution block with Graph Convolution (GCN) [8] and Transformer Convolution (TFC) [9] respectively.

Generally, GCN and TFC belong to message passing layers, which aggregates and process information from the neighborhood of a node (including itself). Both of them and have one or more learnable parameter matrix $\mathbf{W}$. Formally, GCN calculates the node embedding of the i-th node as follows:

$$\mathbf{x}_i' = \mathbf{W}^\top \sum_{j \in \mathcal{N}(i) \cup \{i\}} \frac{e_{j,i}}{\sqrt{d_j d_i}} \mathbf{x}_j, \qquad (2)$$

where $e_{i,j}$ denotes the edge weights between node i and j and $d_i = 1 + \sum_{j \in \mathcal{N}(i) \cup \{i\}} e_{i,j}$.

While GCN weight the neighborhood embeddings by normalized edge weights, TFC computes a similarity (or attention) score between the node of concern and each of its neighbors, and then weight the neighborhood embeddings with the obtained scores. Formally, the embedding of the i-th node is given by:

$$\mathbf{x}_i' = \mathbf{W}_1 \mathbf{x}_i + \sum_{j \in \mathcal{N}(i)} \alpha_{i,j} \mathbf{W}_2 \mathbf{x}_j, \qquad (3)$$

where $\alpha_{i,j}$ denotes the attention score of node i and node j

$$\alpha_{i,j} = \operatorname{softmax}\left(\frac{(\mathbf{W}_3 \mathbf{x}_i)^\top (\mathbf{W}_4 \mathbf{x}_j)}{\sqrt{d}} \right). \qquad (4)$$

Notably, TFC also introduced more parameter matrices compared to GCN.

C. Metrics and Evaluations

Following [1], we evaluate the performance of the classifiers with the Area Under Curve (AUC) score of the Receiver Operating Characteristic (ROC) curve. Given the prediction scores of a binary classifier, the ROC curve plots the true positive rate (TPR, TP/P) as a function of false positive rate (FPR, FP/N), as the classification threshold varies from 0 to 1. As special cases, a perfect classifier will have 1.0 ROC AUC score, while a random-guessing classifier will have 0.5.

Table III shows the ROC AUC scores of the two models when provided with different node embeddings. The embeddings vary as we combine the two feature embedding methods with SVD and structural embedding from Node2Vec. For a fair comparison, the dimensions of the embeddings are set to $\sim$30K for unreduced ones (with $\times$ in SVD column) and 128 for the reduced ones. For example, No.3 combines a 64D SVD-reduced binary embedding with 64D Node2Vec embeddings, thus the final embeddings have 128 dimensions.

All the experiments are launched with the same random seed, and we randomly split the nodes into train, validation and test sets, which contains 70%, 10% and 20% of the nodes respectively. We train the models for 500 epochs[1], and evaluate the model with best validation performance on the test set.

TABLE III: ROC AUC Scores of different feature embeddings.

No.	Feature	SVD	N2V	Dim.	Model	
					GCN	TFC
1	Binary	×	×	31241	0.640	0.725
2	Binary	✓	×	128	0.642	0.713
3	Binary	✓	✓	128	0.637	0.692
4	Binary	×	✓	31369	0.640	0.717
5	Feather	×	×	32000	0.594	0.559
6	Feather	✓	×	128	0.634	0.637
7	Preset	-	-	128	0.640	0.698
8	None	-	✓	128	0.532	0.529

As shown in Equation 1, although the Binary method (No.1) very is straightforward, the TFC model obtains best score

[1]This is equal to 500 iterations since the dataset contains only one graph

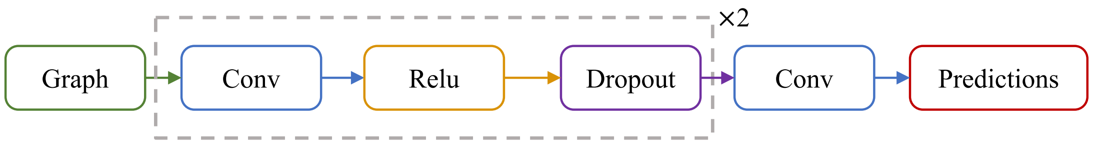

Fig. 5: Model structure and data flow

0.725 with binary embedding and GCN scores 0.640 with it. The dimension of the binary embedding can be too high for many applications, still, we find reducing the dimension down to 128 doesn't sacrifice too much performance (No.2). Contrary to our expectations, combining Node2Vec embeddings with binary embedding (No.4) doesn't improve the performance of GCN, and TFC can even degrade to 0.717 with the added embeddings. We suppose the reason is that, these two models will natrually take the graph structure into account in their compuation. If the feature dimension is further reduced (No.3), the score of TFC decreases to 0.692.

In [1], Feather embeddings are tested with a logistic regression classifier, and the authors obtained 0.673 ROC AUC score. However, GCN and TFC models scores 0.594 and 0.559 respectively (No.5), which are much lower compared to using Binary embeddings (No.1). Interestingly, reducing the embedding dimension (No.6) even improved the performance of both models. We notice the construction progress of Feather embedding (Section 2.1) may produce many high-frequency components. It concatenates the sin and cos of scaled initial embeddings, and multiply the embeddings by matrix power of the normalized adjacency matrix. While the former can make the embedding vector into a high-frequency signal, the latter can create high-frequency signal across the graph.

The preset embedding is provided by Pytorch Geometric, but the documents does not clearly state how the pre-processing is done. Its performance is similar to No.2 and No.3. As a baseline, we evaluated the performane of the two models on Node2Vec embeddings (No.8). In this case, the node features (liked artists) is completely discarded, and both model have very bad scores close to 0.5, which means they are almost randomly guessing. Therefore, the node features are very important for gender classification.

Finally, we observe the TFC model generally have better performance compared to the GCN model, which shows that attention mechanism can improve the capability of a graph neural network.

D. Training progress

Figure 6 shows the progress of training loss and validation ROC AUC Score of the TFC model on Binary node embeddings. The training loss converges roughly after 200 epochs, but the highest validation score is observed before 100 epochs. After that, the validation score decrease and converge to around 0.700, which suggests model is overfitting.

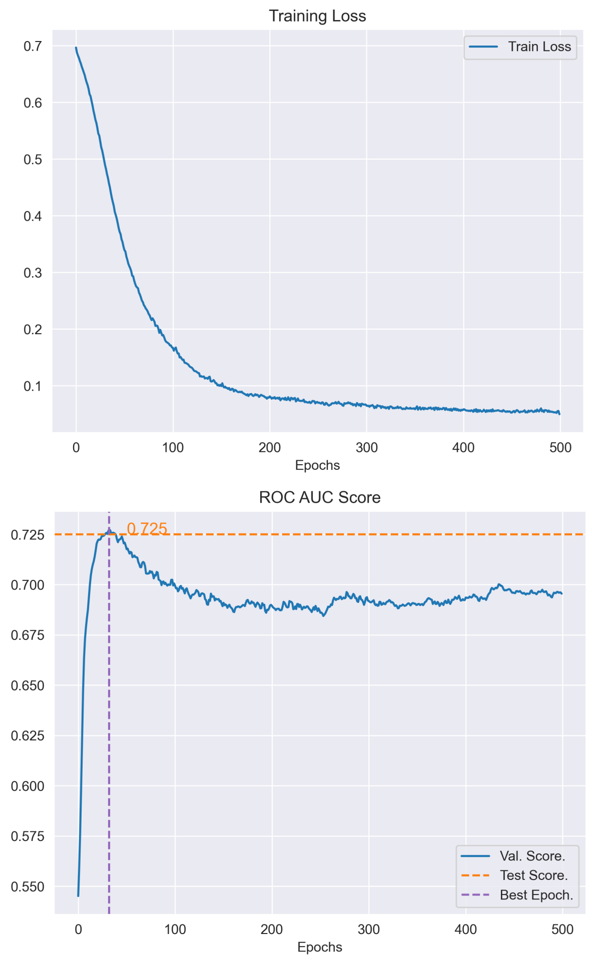

Fig. 6: Training log of TFC model on binary node embeddings. Please refer to our github repo for all training logs

Our GitHub repo provides the training log of all experiments in Table III, and many of them look similar to Figure 6. However, we notice the training progress with Feather embedding is very unstable, and the SVD-reduced Feather embedding gives more stable training compare to the unreduced Feather.

III. Conclusion

In this project, we explored the user graph of Deezer Europe dataset, and evaluated various feature embedding methods by using their embeddings to train a gender classification network. The importance of node features in classifying the gender of users has been demonstrated by the random-guessing behavior

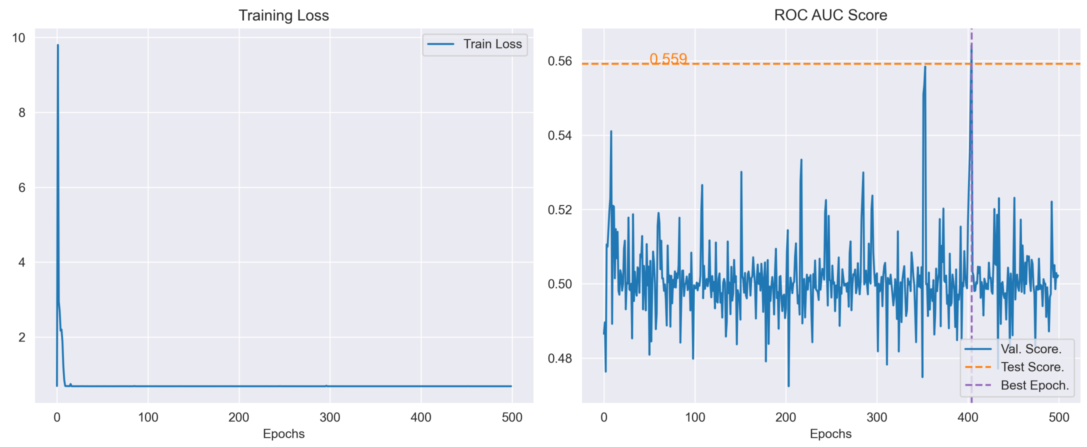

Fig. 7: Training log of TFC model on Feather embeddings. Training is not stable.

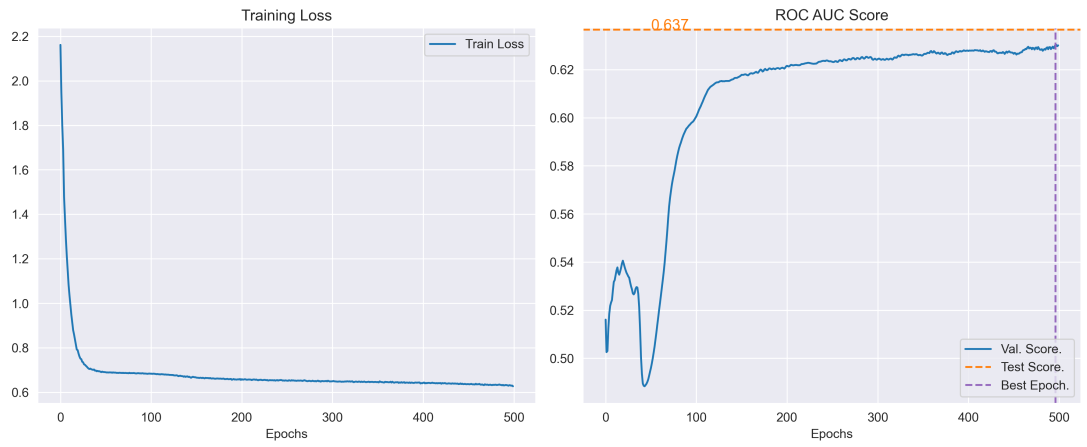

Fig. 8: Training log of TFC model on SVD-reduced Feather embeddings.

of the classifier without access to node feature embeddings. The experiments also show that a straightforward binary embedding is a strong baseline, and dimension reduction with SVD can make a reasonable trade-off between performance and computation. Furtermore, the unstable training progress of Feather-based models has raised questions on the practicability of this method, and SVD shows the ability to stablize the training.

References

[1] B. Rozemberczki and R. Sarkar, "Characteristic functions on graphs: Birds of a feather, from statistical descriptors to parametric models," in Proceedings of the 29th ACM international conference on information & knowledge management, 2020, pp. 1325–1334.

[2] A.-L. Barabási and R. Albert, "Emergence of scaling in random networks," science, vol. 286, no. 5439, pp. 509–512, 1999.

[3] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining, 2016, pp. 855–864.

[4] "Node2Vec python3 implementation," https://github.com/eliorc/node2vec, accessed: 2023 June 6.

[5] B. Rozemberczki, O. Kiss, and R. Sarkar, "Karate Club: An API Oriented Open-source Python Framework for Unsupervised Learning on Graphs," in Proceedings of the 29th ACM International Conference on Information and Knowledge Management (CIKM '20). ACM, 2020, p. 3125–3132.

[6] "Sklearn Decomposition Truncated SVD," https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.TruncatedSVD.html, accessed: 2023 June 6.

[7] M. Fey and J. E. Lenssen, "Fast graph representation learning with PyTorch Geometric," in ICLR Workshop on Representation Learning on Graphs and Manifolds, 2019.

[8] C. Morris, M. Ritzert, M. Fey, W. L. Hamilton, J. E. Lenssen, G. Rattan, and M. Grohe, "Weisfeiler and leman go neural: Higher-order graph neural networks," in Proceedings of the AAAI conference on artificial intelligence, vol. 33, no. 01, 2019, pp. 4602–4609.

[9] Y. Shi, Z. Huang, S. Feng, H. Zhong, W. Wang, and Y. Sun, "Masked label prediction: Unified message passing model for semi-supervised classification," arXiv preprint arXiv:2009.03509, 2020.