GNN Model Based On Node Classification Forecasting in Social Network

1st A.K. Awasthi
dept. of Mathematics,
Lovely Professional University,
Phagwara, Punjab, India
dramitawasthi@gmail.com

2nd Arun Kumar Garov
dept. of Mathematics,
Lovely Professional University,
Phagwara, Punjab, India
arunkumar170@yahoo.com

3rd Minakshi Sharma
dept. of Mathematics,
Lovely Professional University,
Phagwara, Punjab, India
minakshisohareya@gmail.com

4th Mrigank Sinha
dept. of Mathematics,
Lovely Professional University,
Phagwara, Punjab, India
mrigank.sinhal1311@gmail.com

2023 International Conference on Artificial Intelligence and Smart Communication (AISC) | 979-8-3503-2230-9/23/$31.00 ©2023 IEEE | DOI: 10.1109/AISC56616.2023.10085118

***Abstract*—In the time of ever-growing technology, engineering, and deep learning methods, one thing that has caught the attention of people is the invention of Neural Networks, also known as Artificial Neural Networks [1]. These are the subset of machine learning and are at the core of deep learning. Their structure and nomenclature are modeled after the human brain, mimicking the communication between biological neurons [2].**

This work is presented and explained by ANN, Graphical Neural Network [3], which is a type of NN that works on graphs. In today's world, one can see various real-life applications of GNNs like those in various social networks, prediction of molecules, and drug preparation in medical sciences, road traffic, etc.

The article deals with the application of the GNN showing how can a GNN helps in forecasting information about a person in a social network based on various given datasets. In the end, one can easily forecast the information of a person using various tools like Pytorch, etc.

Keywords*— *ANN, GNNs, Forecasting, Social Networks, Machine Learning.

I. Introduction

A social network is a collection of people and their mutual interactions with each other in a community. Collection of people and their mutual interactions with each other in a community is known as a social network. Such a kind of community can commonly be seen these days on any social media like Facebook, Instagram, Twitter, etc.

Actually, a social network is very much analogous to a Graphical Neural network in real life. For example; In a graph there are various entities called nodes with pre-defined personal features and various mutual relationships between these nodes which connect them as edges [4]. Similarly in a social network, the people in a community can be seen as nodes where every person has its own personal traits and the various relationships between the people such as various social interactions, number of mutual friends, number of followers, etc. can be seen as edges.

So, a social network becomes an ideal subject to implement and base a GNN on. By doing this one can perform various tasks on a social network using a GNN. Such tasks include prediction, regression, etc. There are various levels of such tasks [5]. [6] CNN regression for crop field prediction, and [10] forecast of the stock index by QBFTS and statistical weights on the relationship of QBFLRGs.

II. Graph Convolution

A. General Graph Convolution formula

Some insight will be given to the particular graph-based NN model f(X, A) that will be used in this research. Multiple-layer Graph Convolutional [9] with the propagation rule (layer-wise) is described below:

$$H^{(l+1)} = \sigma\,(\tilde{D}^{-1/2}\,\tilde{A}\tilde{D}^{-1/2}\,H^{(l)}\,W^{(l)}) \tag{1}$$

The adjacency matrix of G (undirected graph) with additional self-connections $\tilde{A}$ is shown here as

$$A + I_N = \tilde{A}$$

For graph G, A is an adjacency matrix, (I_N) an identity matrix, and a layer-specific trainable weight matrix is $W^{(l)}$, σ (.) denote the activation function. The hidden feature matrix for lth layer is $H^{(l)}$. Here, proposed how the uses of this propagation rule may be motivated by a 1st-order approximation of localised spectrum filters on graphs [17, [13].

B. Spectral Graph Convolution

Used spectral convolution as x, a signal as belongs to $\mathbb{R}^n$ and multiplied with filter, $g_\theta =$ diag(Θ), parameterized by $\theta \in \mathbb{R}^N$ for Fourier domain, the result is a spectral convolution on the graph [13].

$$g_\theta(L) * x = U g_\theta(\wedge) U^T x \tag{2}$$

in the above equation (2), the unitary matrix U of the eigenvectors of the symmetrically normalized Laplacian matrix,

$$L = I_N - D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$$

Here, D is a diagonal degree matrix, into eigen decomposed $U\Lambda U^T$. Diagonal matrix Λ consists of eigenvalues of L and U^Tx represents the graph's Fourier transform of x, and known as a function of L's eigenvalues, g_Θ

or $g_{\Theta}(\Lambda)$. The concept of Chebyshev polynomials [16] can also be used.

A brief description of $T_k(x)$ (Chebyshev polynomials) up to Kth order can accurately approximate $g_{\Theta}(\Lambda)$[17].

$$g_{\Theta}(\Lambda) \approx \sum_{k=0}^{K-1} \Theta_k T_k(\tilde{\Lambda}) \tag{3}$$

where $\tilde{\Lambda}$ is equal to $\frac{2}{\lambda_{max}}\Lambda - I_N$. λ_{max} represents largest eigen-value of Laplacian Matrix L. Vector (θ) of Chebyshev polynomial.

Now on inserting the obtained value of $g_{\Theta}(\Lambda)$ in "(2)",

$$g_{\Theta} * x \approx \sum_{k=0}^{K} \Theta_k T_k(\tilde{\mathcal{L}})x \tag{4}$$

with $\tilde{\mathcal{L}}$ is equal to $2L/\lambda_{max} - I_N$ as polynomial for Kth-order in the Laplacian, this equation is now K-localized [16], meaning that it only depends on nodes that are at most K steps from the centre node (Kth-order neighbourhood).

Can easily prepare a linear function by putting K=1 and thus restricting the message passing to one hop away from the central node (direct neighbours) [19]. So, by stacking convolutional layers in the form of "(4)", where every layer is followed by non-linearity (point-wise), may construct a neural network model based on graph convolutions.

In this way, the equation becomes independent of Chebyshev polynomials parameters and yet still rich class of convolutional filter functions can be recovered by stacking several such layers. For graphs with extremely wide node degree distributions, such as knowledge graphs, citation networks, social networks, and other real-world graph datasets, it is intuitively expected that such a model can help mitigate the issue of overfitting on local neighborhood structures.

Additionally, on further approximating $\lambda_{max} \approx 2$ in equation (4), get

$$g_{\theta} * x \approx \theta_0 x + \theta_1 (L - I_N) x = \theta_0\, x - \theta_1 D^{-1/2} A D^{-1/2} x \tag{5}$$

Free parameters are θ_0 and θ_1 in equation (5). In real-world applications, it may be preferable to further limit the number of parameters in order to combat overfitting and reduce the number of operations at per layer (matrix multiplications). The following phrase as a result;

$$g_{\theta} * x \approx \theta(I_N + D^{-1/2} A D^{-1/2})\, x \tag{6}$$

θ_0 and θ_1 have been replaced with a single parameter θ, and $I_N + D^{-1/2}AD^{-1/2}$ has eigenvalues between 0 to 2. Therefore, when applied in a deep neural network model, repeated use of this operator may result in numerical instabilities [20]. The renormalization approach that can be used to solve this issue is;

$$I_N + D^{-1/2} A D^{-1/2} \rightarrow \tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}$$

with $D_{ii} = \sum_j \tilde{A}_{ij}$ and $A + I_N = \tilde{A}$.

It can be applied generally to X signal with a feature vector (C–dimensional) for each node and filters F as

$$Z = D^{-1/2} A D^{-1/2} X \Theta \tag{7}$$

In equation (7), the matrix of the filter represents by Θ and convoluted signal matrix by Z.

C. Double Layered Graph Convolution network

The final structure of the GCN model will take the form as;

$$Z = f(X, A) = softmax(\hat{A} ReLU\,(\hat{A} X W^{(0)}) W^{(L)}) \tag{8}$$

$\tilde{A}$ is an adjacency matrix (binary or weighted) with self-connections included, $\hat{A} = \tilde{D}^{1/2}\tilde{A}\tilde{D}^{-1/2}$ and $W^{(L)}$ as the weight matrix of the graph at each layer L, feature matrix (X) and $\tilde{D}$ represent the degree matrix of the graph with self-connections included. Activation function (ReLU) for the first layer and Softmax [21] for second layer.

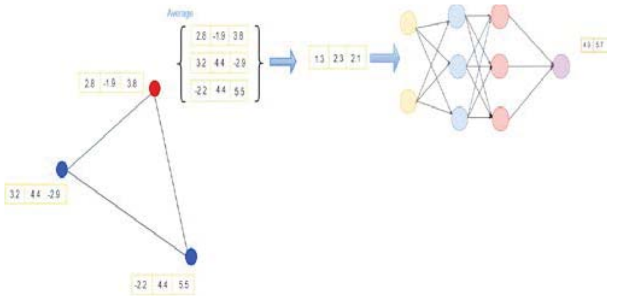


Fig. 1. Message propagation in a GCN layer

III. EXPERIMENTS

A. Implementation

A double-layered GCN "(8)" using *torch_geometric* has been implemented on a feather – Deezer-social dataset. It is a geometric deep learning extension module for PyTorch which offers numerous deep learning iterations on graphs and other irregular structures.

[22] *Torch_geometric.nn.GCNConv* class has been used in this paper which is one of the many classes one can find in PyTorch Geometric.

B. Dataset information

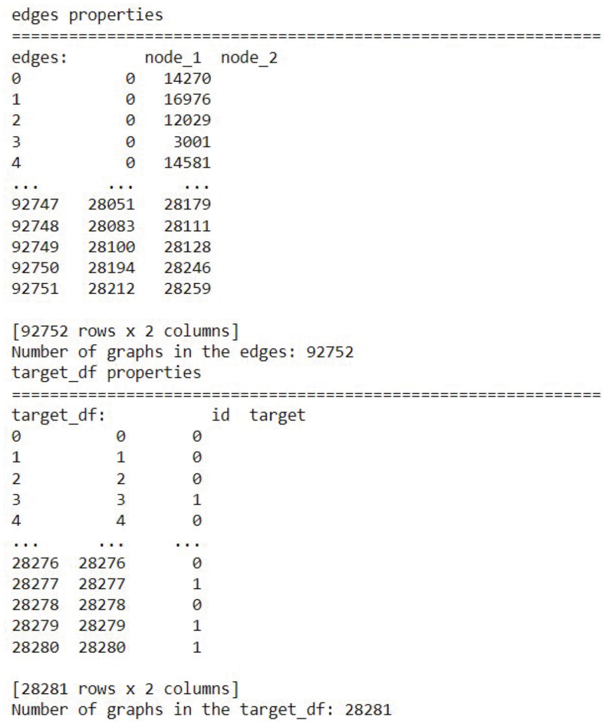
The dataset used in this paper is taken from Stanford Large Network Dataset Collection. The dataset contains information about a music app called Deezer. It provides figures about the number of users in the social network of Deezer that was compiled in March 2020 using the open API. One can consider users of Deezer as the nodes to be used in our GCN and their connections with one another can be used as edges. Vertex characteristics are derived depending on consumers' favourite artists. The name field for each user served as the basis for this target feature.

TABLE I. DATASET INFORMATION

Dataset Statistics	
Directed	No
Temporal	No
Edge features	No
Node labels	Binary classed, Yes
Node features	Yes
Edges	92,752
Nodes	28,281
Transitivity	0.0959
Density	0.0002

C. *Analysis*

In the analysis of the data follow some tasks as Binary node classification, Network Visualization, Community detection, Link prediction. Binary node classification, which requires one to predict the gender of users. Figure 3 shown the binary classification of the tasks.

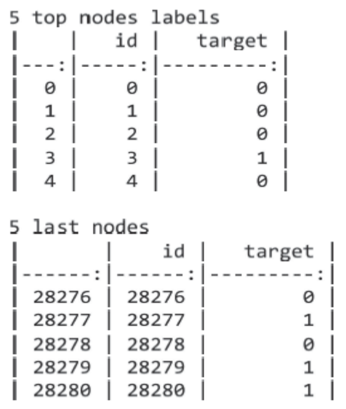


```
edges properties
============================================================
edges:        node_1  node_2
0           0   14270
1           0   16976
2           0   12029
3           0    3001
4           0   14581
...       ...     ...
92747   28051   28179
92748   28083   28111
92749   28100   28128
92750   28194   28246
92751   28212   28259

[92752 rows x 2 columns]
Number of graphs in the edges: 92752
target_df properties
============================================================
target_df:           id  target
0           0       0
1           1       0
2           2       0
3           3       1
4           4       0
...       ...     ...
28276  28276       0
28277  28277       1
28278  28278       0
28279  28279       1
28280  28280       1

[28281 rows x 2 columns]
Number of graphs in the target_df: 28281
```

Fig. 2. Description of the data properties



```
5 top nodes labels
|    |   id |   target |
|---:|-----:|---------:|
|  0 |    0 |        0 |
|  1 |    1 |        0 |
|  2 |    2 |        0 |
|  3 |    3 |        1 |
|  4 |    4 |        0 |

5 last nodes
|       |    id |   target |
|------:|------:|---------:|
| 28276 | 28276 |        0 |
| 28277 | 28277 |        1 |
| 28278 | 28278 |        0 |
| 28279 | 28279 |        1 |
| 28280 | 28280 |        1 |
```

Fig. 3. Binary classification of tasks

The first step is importing of the required packages in the python environment. In the dataset, it can observe how nodes differ according to the number of features. The first histogram shows the class distribution based on features. The second histogram shows the distribution of features among all users of the dataset. The third histogram displays the characteristics that are shared by the majority of users; various peaks can be seen in the distribution. The node features are then one-hot encoded and are used to encode a light subset of the graph (for example, only 60 nodes) for the purpose of visualization. It can be seen how the nodes are connected by edges and labeled by colour, see "Fig 4". A matrix can also be plotted for node features using only the first 250 feature columns of the code for the first 60 nodes, see "Fig. 5".

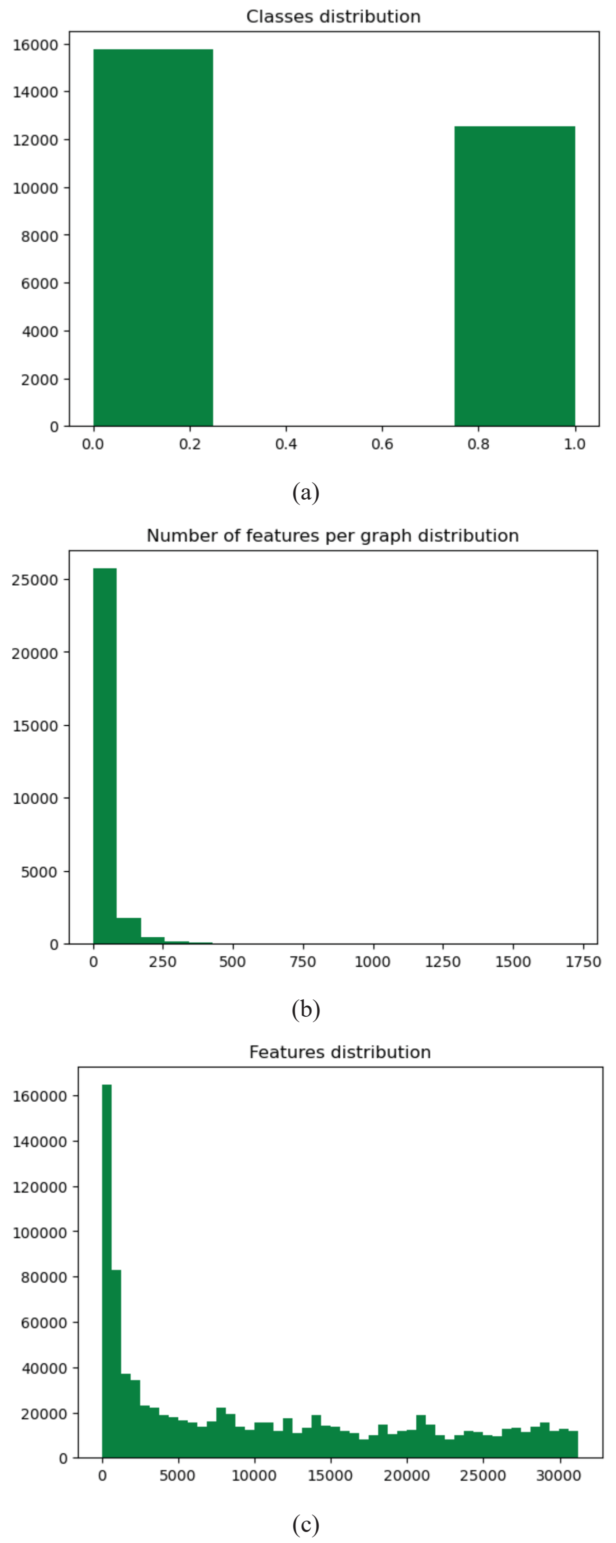


Fig. 4. Representation of various distributions; (a) class distribution, (b) number of features per graph distribution (c) Feature distribution

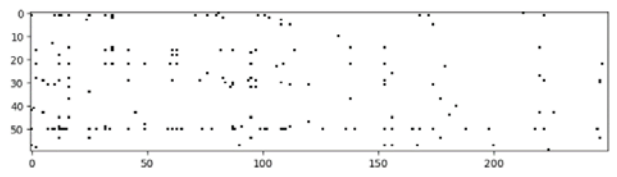


Fig. 5. Matrix representing the first 250 features distribution among the first 60 nodes

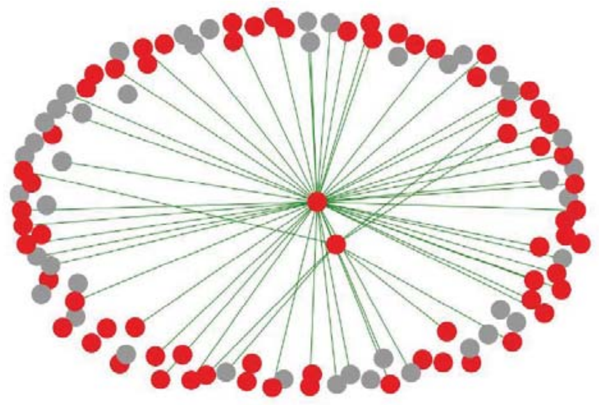


Fig. 6. *A graph displaying the connections made by the edges between the first 60 nodes*

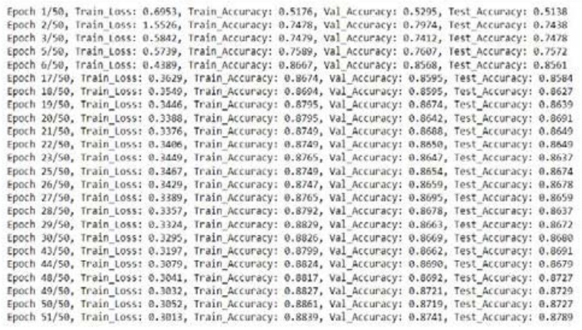
"Fig 6" shown the connection graphs between edges and nodes. After graph representation, the next step is data segregation where a used dataset is divided into 3 subsets for training, validation, and testing at 60%, 30%, and 10%, respectively. This is done to produce a more realistic performance and so that the model doesn't overfit easily. Binary vectors use to give information about nodes to be used in training and testing processes in form of 0 or 1.

The next step is to build the GNN model class. The *torch geometric.nn.GCNConv* class shall be employed as mentioned earlier in the paper. It is required to stack two GCNConv layers. Here used two GCNConv layers, in the graph number of a feature is equal to the input features, and f output features have an arbitrary number. Use of ReLU activation function, the latent characteristics are then delivered to the second layer, which contains output nodes equal to the number of our classes.

The loss and accuracy phase-wise for each specific set will be calculated while the model predicts the class of all nodes in the graph. The goal is to determine the loss and accuracy for each node and multiply those values by the mask to remove any nodes that are unnecessary. *torch.optim.Adam optimizer* is to be used to define the training function. The training will be done over a certain number of epochs, and the best validation accuracy will be monitored.

IV. Result

After building all the necessary functions, the model will be constructed and trained. *nn.CrossEntropyLoss* [25] will be used as the loss criterion and the model will be built with 16 filters. Below, one can see that the model had a very respectable accuracy of almost 88% on the test set.



```
Epoch 1/50, Train_Loss: 0.6953, Train_Accuracy: 0.5176, Val_Accuracy: 0.5295, Test_Accuracy: 0.5138
Epoch 2/50, Train_Loss: 1.5526, Train_Accuracy: 0.7478, Val_Accuracy: 0.7974, Test_Accuracy: 0.7438
Epoch 3/50, Train_Loss: 0.5842, Train_Accuracy: 0.7479, Val_Accuracy: 0.7412, Test_Accuracy: 0.7478
Epoch 5/50, Train_Loss: 0.5739, Train_Accuracy: 0.7589, Val_Accuracy: 0.7607, Test_Accuracy: 0.7572
Epoch 6/50, Train_Loss: 0.4389, Train_Accuracy: 0.8667, Val_Accuracy: 0.8568, Test_Accuracy: 0.8561
Epoch 17/50, Train_Loss: 0.3629, Train_Accuracy: 0.8674, Val_Accuracy: 0.8595, Test_Accuracy: 0.8584
Epoch 18/50, Train_Loss: 0.3549, Train_Accuracy: 0.8694, Val_Accuracy: 0.8595, Test_Accuracy: 0.8627
Epoch 19/50, Train_Loss: 0.3446, Train_Accuracy: 0.8795, Val_Accuracy: 0.8674, Test_Accuracy: 0.8639
Epoch 20/50, Train_Loss: 0.3388, Train_Accuracy: 0.8795, Val_Accuracy: 0.8642, Test_Accuracy: 0.8691
Epoch 21/50, Train_Loss: 0.3376, Train_Accuracy: 0.8749, Val_Accuracy: 0.8688, Test_Accuracy: 0.8649
Epoch 22/50, Train_Loss: 0.3406, Train_Accuracy: 0.8749, Val_Accuracy: 0.8650, Test_Accuracy: 0.8649
Epoch 23/50, Train_Loss: 0.3449, Train_Accuracy: 0.8765, Val_Accuracy: 0.8647, Test_Accuracy: 0.8637
Epoch 25/50, Train_Loss: 0.3467, Train_Accuracy: 0.8749, Val_Accuracy: 0.8654, Test_Accuracy: 0.8674
Epoch 26/50, Train_Loss: 0.3429, Train_Accuracy: 0.8747, Val_Accuracy: 0.8659, Test_Accuracy: 0.8678
Epoch 27/50, Train_Loss: 0.3389, Train_Accuracy: 0.8765, Val_Accuracy: 0.8695, Test_Accuracy: 0.8659
Epoch 28/50, Train_Loss: 0.3357, Train_Accuracy: 0.8792, Val_Accuracy: 0.8678, Test_Accuracy: 0.8637
Epoch 29/50, Train_Loss: 0.3324, Train_Accuracy: 0.8829, Val_Accuracy: 0.8663, Test_Accuracy: 0.8672
Epoch 30/50, Train_Loss: 0.3295, Train_Accuracy: 0.8826, Val_Accuracy: 0.8669, Test_Accuracy: 0.8680
Epoch 43/50, Train_Loss: 0.3197, Train_Accuracy: 0.8799, Val_Accuracy: 0.8662, Test_Accuracy: 0.8691
Epoch 44/50, Train_Loss: 0.3079, Train_Accuracy: 0.8824, Val_Accuracy: 0.8690, Test_Accuracy: 0.8679
Epoch 48/50, Train_Loss: 0.3041, Train_Accuracy: 0.8817, Val_Accuracy: 0.8692, Test_Accuracy: 0.8727
Epoch 49/50, Train_Loss: 0.3032, Train_Accuracy: 0.8827, Val_Accuracy: 0.8721, Test_Accuracy: 0.8729
Epoch 50/50, Train_Loss: 0.3052, Train_Accuracy: 0.8861, Val_Accuracy: 0.8719, Test_Accuracy: 0.8727
Epoch 51/50, Train_Loss: 0.3013, Train_Accuracy: 0.8839, Val_Accuracy: 0.8741, Test_Accuracy: 0.8789
```

Fig. 7. The final output featuring the loss and accuracy at each subset of the dataset at each epoch

V. Conclusion

This paper proposed the role of binary node classification in which a model is prepared that successfully predicts the gender of the users of the Deezer app. The formula involved in convolution at each single layer is being derived using many mathematical concepts. The final model which is being used is of Double-Layered Graph Convolution layer. Using torch geometric, a double-layered GCN is applied to a feather-Deezer-social dataset. The dataset has been taken from Stanford Large Network Dataset Collection. The distribution of the features of the dataset across the first 60 nodes is represented using the histograms. Data segregation is performed, and each subset of a dataset as training, validation, and testing phase respectively to train the model. Binary vectors called masks are used to tell the training phase which nodes should be included during training and the inference phase which nodes are the test data. The GNN model class is then prepared by stacking two GCNConv layers. The loss and accuracy phase-wise for each specific set will be calculated while the model predicts the class of all nodes in the graph. The training function must be defined. The best validation accuracy will be monitored as the training is conducted over a predetermined number of epochs. The model will be created and trained after creating all the required functions. The model will be constructed using 16 filters, and *nn.CrossEntropyLoss* will be used as the loss criterion. The model obtained a very respectable accuracy of almost 88% on the test set.

The model prepared in this paper can be used in many nodes' classification tasks like citation network [26], [27], social networks like GitHub [14], Facebook [28], Twitch [29], road networks [30], user actions [31], Wikipedia networks [32], [33], Amazon product co-purchasing network [34], etc.

References

[1] S. C. Wang, "Artificial neural network," In Interdisciplinary computing in java programming, Springer, Boston, MA, 2003, pp. 81-100.

[2] N. Kriegeskorte, "Deep neural networks: a new framework for modelling biological vision and brain information processing," biorxiv, 2015, 029876.

[3] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," IEEE transactions on neural networks, 20(1), 2008, pp. 61-80.

[4] D. B. West, Introduction to graph theory, Upper Saddle River: Prentice hall., Vol. 2, 2001.

[5] X. M. Zhang, L. Liang, L. Liu, and M. J. Tang, "Graph neural networks and their current applications in bioinformatics," Frontiers in genetics, 12, 2021.

[6] A.K. Awasthi, and A. K. Garov, Agricultural modernization with forecasting stages and machine learning. In Smart Agriculture:

Emerging Pedagogies of Deep Learning, Machine Learning and Internet of Things, 2020, pp. 61-80. CRC Press.

[7] D. H. Lee, "Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks," In Workshop on challenges in representation learning, ICML, vol. 3, no. 2, p. 896, 2013.

[8] S. Bhagat, G. Cormode, and S. Muthukrishnan, "Node classification in social networks," In Social network data analytics, Springer, Boston, MA, 2011, pp. 115-148.

[9] L. Cai, J. Li, J. Wang, and S. Ji, "Line graph neural networks for link prediction," IEEE Transactions on Pattern Analysis and Machine Intelligence, 2021.

[10] A. K. Garov, "Quantity Based weights forecasting for TAIEX," In Journal of Physics: Conference Series, vol. 2267, no. 1, 2022, May p. 012151. IOP Publishing.

[11] M. Jiang, Y. Chen, and L. Chen, "Link prediction in networks with nodes attributes by similarity propagation," arXiv preprint arXiv:1502.04380, 2015.

[12] A. F. Agarap, "Deep learning using rectified linear units (relu)," arXiv preprint arXiv:1803.08375, 2018.

[13] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," Advances in neural information processing systems, vol. 29, 2016.

[14] T. N. Kipf, and M. Welling, "Semi-supervised classification with graph convolutional networks," arXiv preprint arXiv:1609.02907, 2016.

[15] R. Merris, "Laplacian matrices of graphs: a survey," Linear algebra and its applications, vol.197, 1994, pp.143-176.

[16] P. Kabal, and R. P. Ramachandran, "The computation of line spectral frequencies using Chebyshev polynomials," IEEE Transactions on Acoustics, Speech, and Signal Processing, vol. 34, no. 6, 1986, 1419-1426.

[17] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," Advances in neural information processing systems, vol. 29, 2016.

[18] M. D. Porter, and R. Smith, "Network neighborhood analysis," In 2010 IEEE International Conference on Intelligence and Security Informatics, 2010, pp. 31-36, IEEE.

[19] M. Yang, E. Isufi, and G. Leus, "Simplicial convolutional neural networks," In ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2022, pp. 8847-8851, IEEE.

[20] E. Haber, and R. Lars, "Stable architectures for deep neural networks," Inverse problems, vol. 34, no. 1, 2017, pp. 014004.

[21] R. Hu, B. Tian, S. Yin, and S. Wei, "Efficient hardware architecture of softmax layer in deep neural network," In 2018 IEEE 23rd International Conference on Digital Signal Processing (DSP), 2018, November, pp. 1-5, IEEE.

[22] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, and A. Desmaison, "Pytorch: An imperative style, high-performance deep learning library," Advances in neural information processing systems, vol. 32, 2019.

[23] J. Tan, J. Yang, S. Wu, G. Chen, and J. Zhao, "A critical look at the current train/test split in machine learning," arXiv preprint arXiv:2106.04525, 2021.

[24] P. Bharati, and A. Pramanik, "Deep learning techniques—R-CNN to mask R-CNN: a survey," Computational Intelligence in Pattern Recognition, 2020, pp.657-668.

[25] Y. Ho, and S. Wookey, "The real-world-weight cross-entropy loss function: Modeling the costs of mislabeling," IEEE Access, vol. 8, 2019, pp.4806-4813.

[26] J. Gehrke, P. Ginsparg, and J. Kleinberg, "Overview of the 2003 KDD Cup," Acm Sigkdd Explorations Newsletter, vol. 5, no.2, 2003, pp.149-151.

[27] J. Leskovec, J. Kleinberg, and C. Faloutsos, "August. Graphs over time: densification laws, shrinking diameters and possible explanations," In Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining, 2005, pp. 177-187.

[28] J. Leskovec, and J. Mcauley, "Learning to discover social circles in ego networks," Advances in neural information processing systems, vol. 25, 2012.

[29] B. Rozemberczki, C. Allen, and R. Sarkar, "Multi-scale attributed node embedding," Journal of Complex Networks, vol. 9, no.2, 2021 p.cnab014.

[30] J. Leskovec, K. J. Lang, A. Dasgupta, and M. W. Mahoney, "Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. Internet Mathematics, vol. 6, no. 1, 2009, pp.29-123.

[31] S. Kumar, X. Zhang, and J. Leskovec, "Predicting dynamic embedding trajectory in temporal interaction networks," In Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining, 2019, July, pp. 1269-1278.

[32] R. West, J. Pineau, and D. Precup, "Wikispeedia: An online game for inferring semantic distances between concepts," In Twenty-First International Joint Conference on Artificial Intelligence, 2009, June..

[33] J. Leskovec, D. Huttenlocher, and J. Kleinberg, "Predicting positive and negative links in online social networks," In Proceedings of the 19th international conference on World wide web, 2010, April, pp. 641-650.

[34] J. Leskovec, L. A. Adamic, and B. A. Huberman, "The dynamics of viral marketing," ACM Transactions on the Web (TWEB), vol.1, no. 1, 2007 pp.5-es.