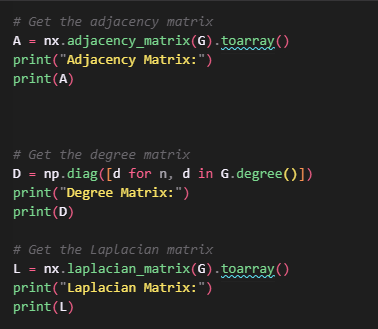
**IT21038914**

DL lab 6 – Graph Neural Networks

1. **NetworkX\_tutorial.ipynb.** 
   * Complete the code sections to get the degree matrix and Laplacian matrix of the created random graph.



* + Calculate the graph density of the random graph in the code. Use the below equation (D = graph density, |V| = number of nodes and |E| = number of edges).

A computer screen shot of text

Description automatically generated

* + Increase the N value from 20 (original value) to 200 with multiple N values in between and observe the change of graph density and degree distribution (i.e., histogram plot). Explain what you observe and write the answer in a word file.

A graph with blue lines and numbers

Description automatically generatedA graph of a degree distribution

Description automatically generated

As the number of nodes (N) increases from 20 to 200, the graph density fluctuates with an initial drop, a peak at N=60, and a general downward trend stabilizing around N=200, while the degree distribution for N=200 approximates a normal distribution centered around degrees 95 to 105

1. **In the KarateClub dataset based GCN code, we use semi-supervised training approach along with the transductive leaning method.**
   * Explain the differences between supervised learning, self-supervised learning and semi-supervised learning methods

 **Supervised Learning**: In supervised learning, the model is trained on labeled data, where each input has a corresponding correct output (label). The goal is to learn a mapping from inputs to outputs that generalizes to unseen data.

 **Self-Supervised Learning**: Self-supervised learning uses unlabeled data but generates its own labels or tasks (like predicting parts of the input) to learn representations. It is often used in domains with abundant unlabeled data, such as computer vision or natural language processing.

 **Semi-Supervised Learning**: Semi-supervised learning combines a small amount of labeled data with a large amount of unlabeled data to improve learning. It aims to leverage the structure in the unlabeled data to enhance the model's performance when labeled data is scarce.

* + Explain the differences between transductive learning and inductive learning.

 **Transductive Learning**: Transductive learning focuses on predicting labels only for the specific set of test data it encounters, rather than learning a general model. It can make better use of the specific data at hand but is limited in its applicability to new, unseen data.

 **Inductive Learning**: Inductive learning seeks to learn a general rule or function from training data that can be applied to new, unseen data. It builds a model that generalizes beyond the examples it was trained on.

1. **KarateClub**

1. Increase the number of epochs from 50 to 500 and observe the change in validation accuracy and write what you observe in the word file.

Increasing the number of epochs from 50 to 500 led to an initial improvement in validation accuracy, reaching 0.7941 by the 50th epoch and eventually stabilizing at 0.8235 by the 250th epoch. The training accuracy remained consistently high at 1.0000 throughout, indicating early convergence and stability in model performance thereafter.

2. Experiment without self-loops added to GCNConv() layers in the GCN() model and detail the model accuracy increase/decrease in the word file.

Without self-loops, the model's validation accuracy peaked at 0.6176 by epoch 350, while with self-loops, it reached 0.8235 by epoch 300 and remained stable.

3. Increase the number of GCNConv() layers in the GCN() model upto 8 layers from original 3 layers. Detail the accuracy increase/decrease in the word file.

Increasing the number of GCNConv layers to 8 showed that with self-loops, validation accuracy improved gradually, reaching 0.6176 by epoch 350 but then slightly dropping to 0.5882 by epoch 450, while without self-loops, validation accuracy quickly stabilized around 0.5294 by epoch 50 and slightly improved to 0.5588 by epoch 350, maintaining stability thereafter.

1. In\_channels and out\_channels in GCNConv() can be considered as hyper-parameters and you can use the best performing values you find.

With in\_channels=8 and out\_channels=8, the validation accuracy peaked at 0.6471 with self-loops and 0.5588 without self-loops. With in\_channels=32 and out\_channels=16, validation accuracy reached 0.6176 with self-loops and 0.6471 without self-loops, indicating better performance without self-loops for this configuration.

1. Add skip connections between some of the GCNConv() layers and try to see if that can improve the model performance.

Adding skip connections with the best hyperparameters (in\_channels=8, out\_channels=8) improved the validation accuracy, achieving 0.7647 with self-loops and 0.6176 without self-loops by epoch 200 and staying stable thereafter. The highest validation accuracy of 0.7941 was observed with hyperparameters (in\_channels=16, out\_channels=32) by epoch 450 when using both self-loops and skip connections.

1. Explain the differences between Message Passing GNN, graph convolution network (GCN), graph attention network (GAT) and GraphSAGE. Write the answers in the word file.

 **Message Passing GNN**:  
Message Passing GNN is a general framework where nodes aggregate information from their neighbors by exchanging messages. This process is repeated over multiple layers, with each node updating its representation based on the messages received from its neighbors.

 **Graph Convolution Network (GCN)**:  
GCN is a type of Message Passing GNN that performs graph convolutions, where each node updates its feature by averaging its neighbors' features. The aggregation process is weighted by the graph's adjacency matrix, normalized to ensure equal contributions.

 **Graph Attention Network (GAT)**:  
GAT uses attention mechanisms to assign different weights to a node's neighbors, determining which neighbors are more important for updating the node's representation. This attention-based aggregation makes GAT more flexible than GCN when capturing node relationships.

 **GraphSAGE**:  
GraphSAGE (Sample and Aggregate) learns aggregation functions that sample a fixed number of neighbors for each node and combine their features. Unlike GCN, which operates over the entire graph, GraphSAGE is designed for inductive learning and can scale better for large graphs by using neighborhood sampling.