CSE 811 Data Mining
Project Proposal

Team Name- Node Navigators
AB Basit Rafi Syed, Roshni Ranjita Bhowmik, Hitesh Malhotra, Hardik Arora

Abstract

This project classifies nodes by leveraging both their individual features and the graph's connectivity. We compare graph-based models like GCN and GAT that incorporate neighboring information. The comparative analysis highlights the benefit of including graph structure in improving classification accuracy.

Introduction

Graphs capture relationships in domains like social networks, biology, and recommendations, making node classification critical for deriving insights. This project classifies nodes in a graph (2480 nodes, 10100 edges, 7 classes, 1390 features) by integrating node features with graph connectivity to enhance predictive performance.

Proposed Methodology and Preliminary Plan

Our approach begins with a simple baseline model and progresses to advanced graph-based models.

- **Baseline Multi-Layer Perceptron (MLP):** Classifies nodes using only their features, ignoring the graph structure. Serves as a baseline to demonstrate the value added by incorporating graph connectivity.
- **Graph Sample and Aggregate (GraphSAGE):** Aggregates features from sampled neighbors, enabling inductive learning and scalability to large graphs, with good generalization to unseen nodes.
- **Graph Convolutional Network (GCN):** Combines node features with local neighborhood information using graph convolutions, effectively capturing relational patterns for improved classification.
- **Graph Attention Network (GAT):** Utilizes attention mechanisms to assign weights to neighbors, allowing the model to focus on the most relevant nodes during feature aggregation.
- **Preprocessing and Comparison:** All models are trained with normalized features and efficient handling of the sparse adjacency matrix. A comparative analysis against the MLP baseline highlights the benefits of graph-aware learning.

1. Dataset Overview

The dataset comprises the following components:

Feature Matrix (features.npy)

This is a NumPy array with 2480 rows and d columns, where each row corresponds to a data point (e.g., a node in a graph) and each column represents a numerical feature. The exact number of features (d) depends on the context (e.g., node embeddings, structural features, etc.).

Label Array (labels.npy)

This is a NumPy array of shape (496,), providing class labels for 496 of the samples. This indicates that only a subset of the data (likely a labeled subset) is used for supervised learning.

Adjacency Matrix (adj.npz)

A sparse matrix representing the connections or edges between nodes in a graph. This structure supports the idea that the dataset is designed for a graph-based machine learning task such as node classification or link prediction.

2. Feature Engineering and Preprocessing

To prepare the data for model training, several preprocessing steps were applied:

2.1 Normalization

Before applying machine learning algorithms, the feature values are standardized using z-score normalization (zero mean and unit variance). This step ensures:

- Features are on a comparable scale.
- Algorithms such as gradient descent converge faster.
- Variance-based methods like PCA are not biased toward higher-magnitude features.

Method Used:

$$X_{norm} = \frac{X - \mu}{\sigma}$$

where μ and σ are the mean and standard deviation of each feature column, respectively.

2.2 Principal Component Analysis (PCA)

To reduce dimensionality and extract meaningful latent features, PCA is applied. PCA projects the data onto a lower-dimensional space while preserving maximum variance. This helps:

- Remove redundant or collinear features.
- Improve model efficiency and generalization.
- Aid visualization (when reduced to 2 or 3 dimensions).

Key Details:

- The number of components is often capped (e.g., to 50 or fewer) or selected based on explained variance.
- PCA is applied *after* normalization.

3. PyTorch Geometric

The preprocessing stage converts all data components into formats compatible with PyTorch and PyTorch Geometric. The feature matrix, originally a NumPy array, is converted to a `FloatTensor` using `torch.from_numpy().float()` to match the input requirements of GCN layers. The adjacency matrix, stored as a SciPy sparse matrix, is transformed into the `edge_index` format using `from_scipy_sparse_matrix`, producing a $2 \times E$ tensor that lists all edges explicitly.

The label array contains known labels for a subset of nodes and is handled in a semi-supervised manner. A full label tensor is initialized with -1 to mark unlabeled nodes, and known labels are inserted at the specified training indices as a `LongTensor`. Additionally, training and test splits from a JSON file are converted to long tensors and moved to the same device as the model. These steps ensure that features, structure, and labels are all ready for training and inference using PyTorch Geometric.

4. Train-Test Split

For model evaluation, the labeled subset of 496 samples is divided into training and validation sets.

- **Split Ratio:** 80% training, 20% validation
- **Randomization:** A fixed random seed is used to ensure reproducibility.

Purpose:

- The *training set* is used to fit the model.
- The *validation set* evaluates model performance on unseen data.

5. Models Implemented

5.1 Baseline: Multi-Layer Perceptron (MLP)

- **Input:** Node features only (no graph structure).
- **Architecture:** Fully connected feedforward network with ReLU activation and dropout.
- **Purpose:** Provides a benchmark for performance without leveraging graph information.

5.2 Graph Convolutional Network (GCN)

- **Input:** Node features + edge index derived from the adjacency matrix.
- **Architecture:**
  - Two-layer GCN using `GCNConv` layers.
  - ReLU activation and dropout between layers.
- **Training:** CrossEntropy loss on labeled training nodes.
- **Optimizer:** Adam with weight decay (5e-4) and learning rate of 0.01.
- **Outcome:** High performance on node classification by incorporating neighborhood information through convolution on graph.

5.3 GraphSAGE

- **Approach:** Sample-based aggregation of neighbor features.
- **Key Advantage:** Scales better for large graphs; allows inductive learning.
- **Architecture:** Two GraphSAGE layers with ReLU and dropout.
- **Used:** For testing how sampling-based GNNs perform vs GCNs.

5.4 Graph Attention Network (GAT)

- **Approach:** Uses self-attention over neighbors instead of uniform or weighted averaging.
- **Key Feature:** Attention coefficients determine the importance of each neighbor.
- **Outcome:** Particularly useful when the importance of neighbors varies significantly.

6. Model Evaluation

We trained each model for 200 epochs and recorded the best-performing version based on validation accuracy. The following metrics represent the best performance achieved by each model during training.

Model	Accuracy (%)	Precision (%)	Recall (%)	F1 Score
Baseline MLP	73	74.36	73.08	0.7319
GraphSAGE	89	89.29	89.01	0.8896
GCN	90	90.13	90.12	0.8994
GAT	84	85.52	84.23	0.8410

Table 1: Best Validation Results

6.1 Baseline MLP:

The Baseline MLP model shows rapid convergence on the training set, but its validation accuracy remains significantly lower and fluctuates across epochs, indicating overfitting. The confusion matrix reveals limited generalization across certain classes, with noticeable misclassifications especially in overlapping categories.

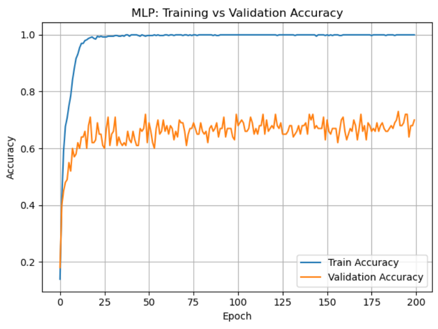


Figure 1: MLP: Training vs Validation Accuracy

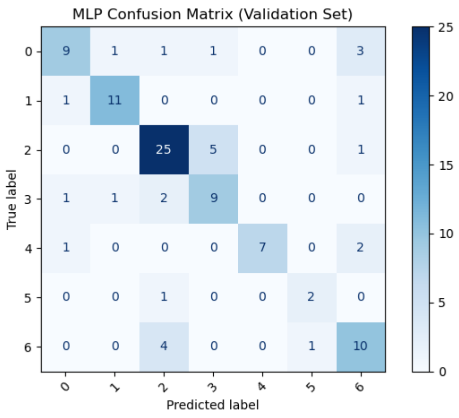


Figure 2: MLP: Confusion Matrix

6.2 GraphSAGE:

The GraphSAGE model demonstrates strong generalization, maintaining consistently high validation accuracy across epochs. The confusion matrix confirms balanced and accurate classification across most classes, with only minor misclassifications, affirming the effectiveness of convolutional message passing in capturing graph topology.

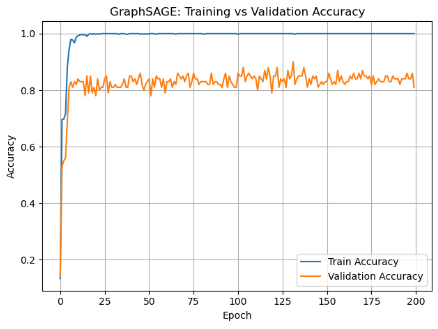


Figure 3: GraphSAGE: Training vs Validation Accuracy

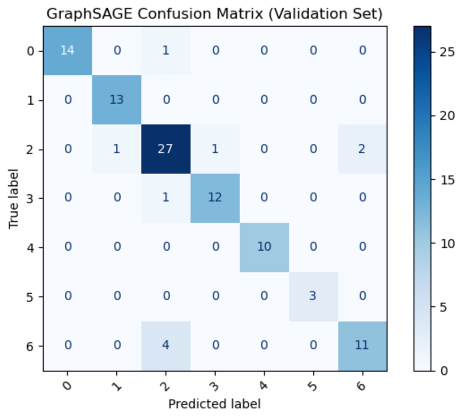


Figure 4: GraphSAGE: Confusion Matrix

6.3 GCN:

GCN achieves consistently high training and validation accuracy, indicating strong learning and generalization capability. The confusion matrix shows accurate predictions across all classes with minimal errors.

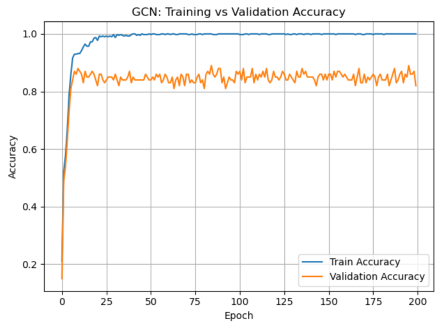


Figure 5: GCN: Training vs Validation Accuracy

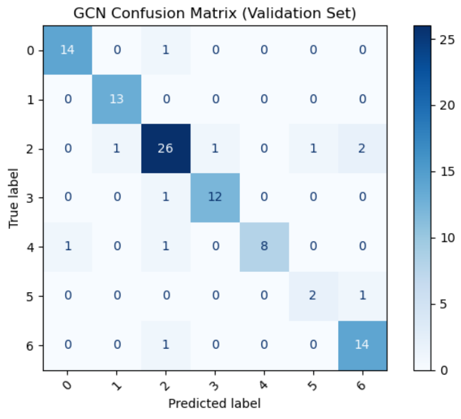


Figure 6: GCN: Confusion Matrix

6.4 GAT:

The GAT model achieves moderate training and validation accuracy with noticeable fluctuations, suggesting sensitivity to attention weight updates. While it performs reasonably well overall, the confusion matrix indicates some inconsistency across classes, pointing to room for improvement in stability and class-wise precision.

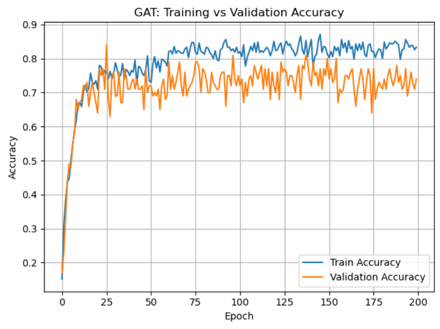
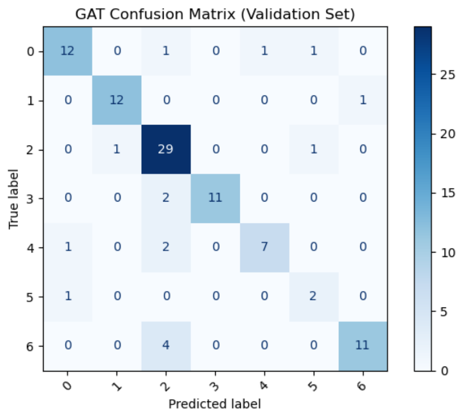


Figure 7: GAT: Training vs Validation Accuracy



True \ Predicted	0	1	2	3	4	5	6
0	12	0	1	0	1	1	0
1	0	12	0	0	0	0	1
2	0	1	29	0	0	1	0
3	0	0	2	11	0	0	0
4	1	0	2	0	7	0	0
5	1	0	0	0	0	2	0
6	0	0	4	0	0	0	11

Figure 8: GAT: Confusion Matrix

7. Conclusion

Among all the models evaluated, the GCN consistently achieved the highest performance across all key metrics, including accuracy, precision, recall, and F1 score. Its stable training behavior and balanced class-wise predictions, as evidenced by the confusion matrix, highlight its strong generalization ability on this dataset. Therefore, we select GCN as the final model for our node classification task.

References

1. **PyTorch Geometric (PyG)**: https://pytorch-geometric.readthedocs.io
A widely used library for implementing graph neural networks using PyTorch.

2. **Deep Graph Library (DGL)**: https://www.dgl.ai
A flexible Python package for deep learning on graphs.

3. **Papers with Code – Graph Neural Networks**: https://paperswithcode.com/task/graph-neural-networks
Tracks state-of-the-art models, code, and benchmarks in the GNN space.

4. **Stanford CS224W – Machine Learning with Graphs**: http://web.stanford.edu/class/cs224w
An excellent academic course covering theoretical and practical aspects of graph-based ML.

5. **ChatGPT (OpenAI)**: https://chat.openai.com
A conversational AI model useful for exploring, debugging, and explaining GNN concepts, code, and research papers.