**CSE 881**

**Final Project Report**

1. **TEAM:** Kanyarasi
2. **TEAM MEMBERS:**
3. Thejesh Mallidi([mallidit@msu.edu](mailto:mallidit@msu.edu))
4. Sai Mohan Gajapaka [(gajapaka@msu.edu)](mailto:gajapaka@msu.edu)
5. Amith Reddy Atla [(atlaamit@msu.edu)](mailto:atlaamit@msu.edu)
6. **ROLE:**

* Thejesh Mallidi:
  + GAT and APPNP Model Development: Develop the Attention-based Graph Neural Network and the APPNP model.
  + Model Stacking and Ensemble Learning: Combine individual model predictions using stacking or other ensemble methods.
  + Result Analysis and Interpretation: Analyze model outcomes, create visual representations, and extract actionable insights.
* Sai Mohan Gajapaka:
  + GCN Model Development: Design, implement, and tune the Graph Convolutional Network model.
  + Model Evaluation and Validation: Set up evaluation metrics and validate model performance to ensure accuracy and robustness.
  + Project Coordination and Review: Oversee project progress and ensure integration of components.
* Amith Reddy Atla:
  + Data Preprocessing and Feature Engineering: Clean, normalize, and prepare data for input into graph-based models.
  + GSage Model Development: Implement and fine-tune the GraphSAGE model.
  + Report Writing and Documentation: Document the processes and compile the comprehensive report.

1. **WORKFLOW**

1. Loading the Data:

* Adjacency matrix (`adj.npz`): Amatrix representing the graph structure, where nodes represent entities and edges represent connections between these entities.
* Features (`features.npy`): A NumPy array containing the attributes or properties of each node.
* Labels (`labels.npy`): A NumPy array containing the class labels for a subset of the nodes.
* Data splits (`splits.json`): A JSON object defining the indices of the nodes to be used for training and testing.

2. Dimensionality Reduction via PCA:

* Principal Component Analysis (PCA) is applied to the node features to reduce the dimensionality from its original size to 128 dimensions. This step is crucial to improve the efficiency and potentially the effectiveness of the learning process by reducing noise and computational cost.

3. Preparation of Graph Data for PyTorch Geometric:

* The adjacency matrix is converted to an edge index format using `from\_scipy\_sparse\_matrix`, which is required by PyTorch Geometric for graph representation.
* Node features are reduced in dimensionality and then converted to a PyTorch tensor.
* Labels are prepared by initially setting all to `-1` (indicating unlabeled), and then setting the labels of training nodes according to the data provided.
* A `Data` object from PyTorch Geometric is created, containing the node features (`x`), graph connectivity (`edge\_index`), and node labels (`y`).

4. Preparation of Training and Testing Masks:

* Masks are boolean tensors indicating whether a node belongs to the training set or the testing set. These masks are used during the training process to ensure that the model learns only from the training data and is evaluated on separate test data.
* Initially, the `train\_mask` is set for nodes in the training index, and the `test\_mask` is set for nodes in the testing index.

5. Adjustment to Training Mask:

* Now training and testing indices are merged, and the training mask is updated to include nodes from both the original training and testing sets. This setup will help us utilize the whole data and help the proposed models to work robustly.

**Overview of Training Process:**

1. Data Preparation:

* For the training part, Indices that are designated for training and testing are combined, this enhances the diversity of data available for model training across different validation folds.

2. Stratified k-Fold Cross-Validation Setup:

* The data is divided into eight folds using the `StratifiedKFold` method to ensure each fold retains a similar distribution of class labels. This method helps manage class imbalance and provides a thorough validation framework.

3. Model Configuration:

* A GCN model is configured with defined parameters for the number of node features, hidden units, and output classes.
* An Adam optimizer and a cross-entropy loss function that ignores nodes labeled `-1` are employed for model training and validation loss calculations.

4. Training and Validation Loop:

* Training and validation masks are established to specify which nodes are to be used for each purpose in every fold.
* The model is trained over 200 epochs for each fold, with parameter updates via backpropagation based on the loss computed from the training nodes.
* Model performance is assessed every 10 epochs on the validation set, focusing solely on nodes with valid labels to ensure accurate evaluations.

5. Performance Monitoring and Model Saving:

* The model state demonstrating the best validation accuracy (above an 84% threshold) is tracked and saved, ensuring that only the most effective configurations are preserved.

6. Output:

* The best model configuration from each fold is saved for potential future analysis or operational deployment.

1. **MODELS**

This section gives a brief description of the models used in our project.

1. **GCN Model**

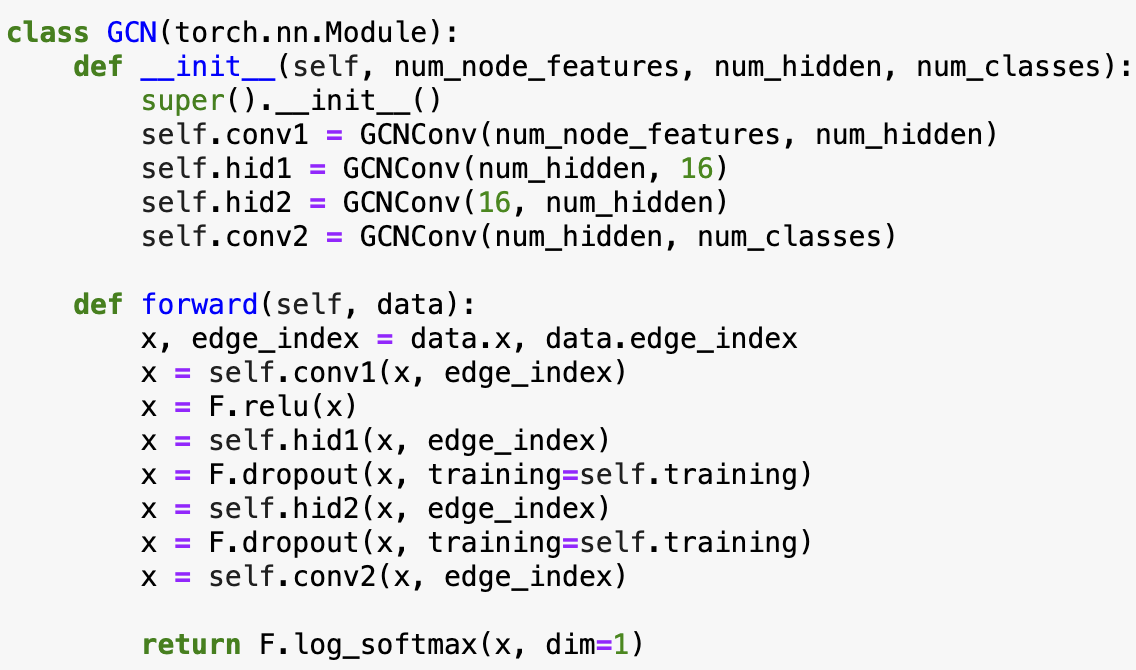


Figure 1: Code snippet showing the implementation of GCN model.

* The Graph Convolutional Networks model consists of several graph convolutional layers, which are specifically designed to work with data that is represented as graphs. A brief description of the architecture is given below:
* Input Layer (conv1):
* This layer takes the initial node features and transforms them to a hidden representation. The transformation involves aggregating features from a node’s neighbors, which helps in capturing local topological features.
* This helps in integrating information from each node's immediate neighborhood, which can reveal patterns relevant to the node's characteristics.
* Intermediate Layers (hid1 and hid2):
* These layers continue the process of feature transformation and aggregation. hid1 reduces the dimension to 16, focusing on extracting more critical features. Then, hid2 expands the features to a higher dimension.
* The transformations are designed to capture more complex patterns in the data, potentially representing higher-order interactions between nodes that are not immediately adjacent.
* Output Layer (conv2):
* The final layer computes the scores for each class for every node. The use of a softmax function ensures that the output can be interpreted as probabilities for each class.
* This determines the final classification of each node based on the learned features, which can be directly used for decision-making or further analysis in the project.

**RESULTS:**

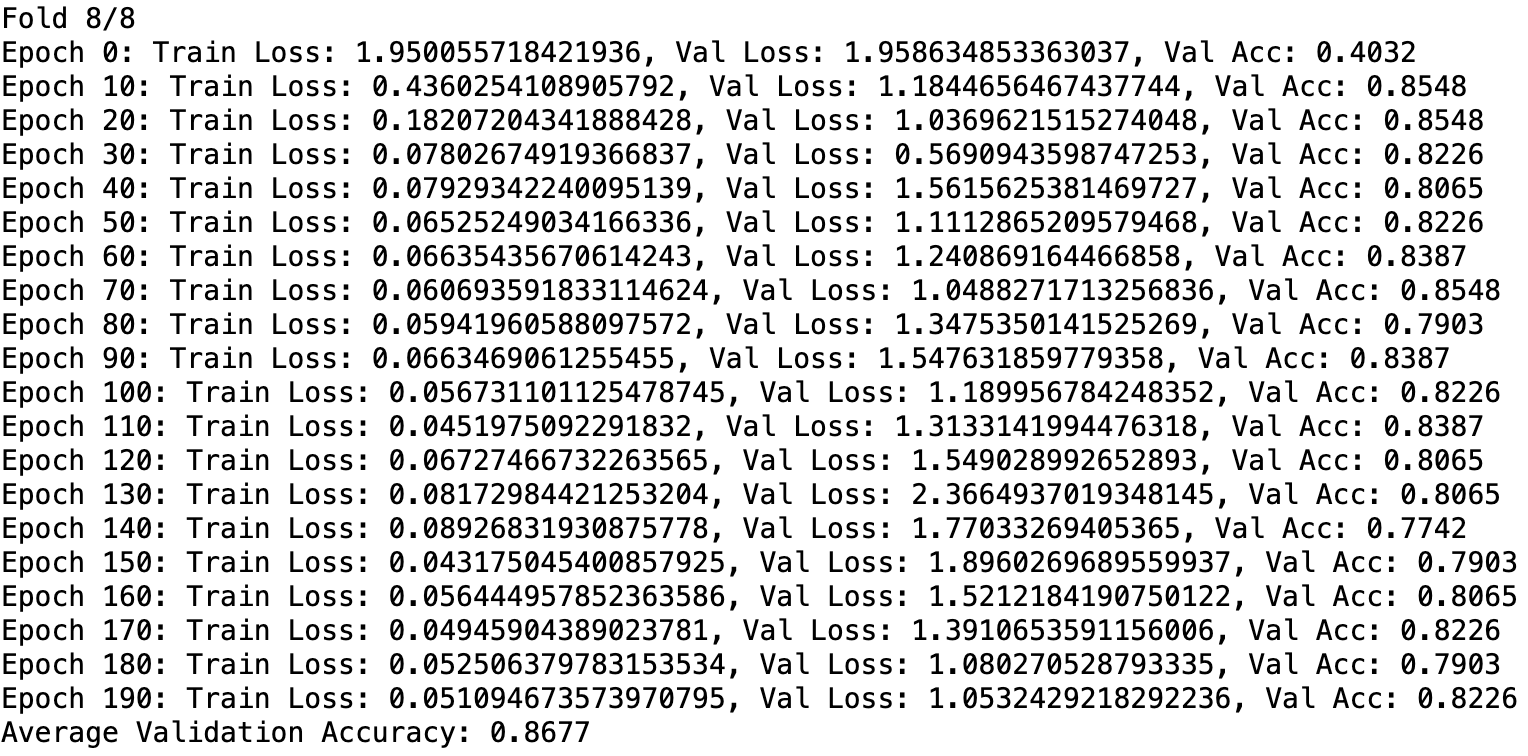
****

Figure 2: Average validation accuracy of the GCN Model.

1. **APPNP Model**

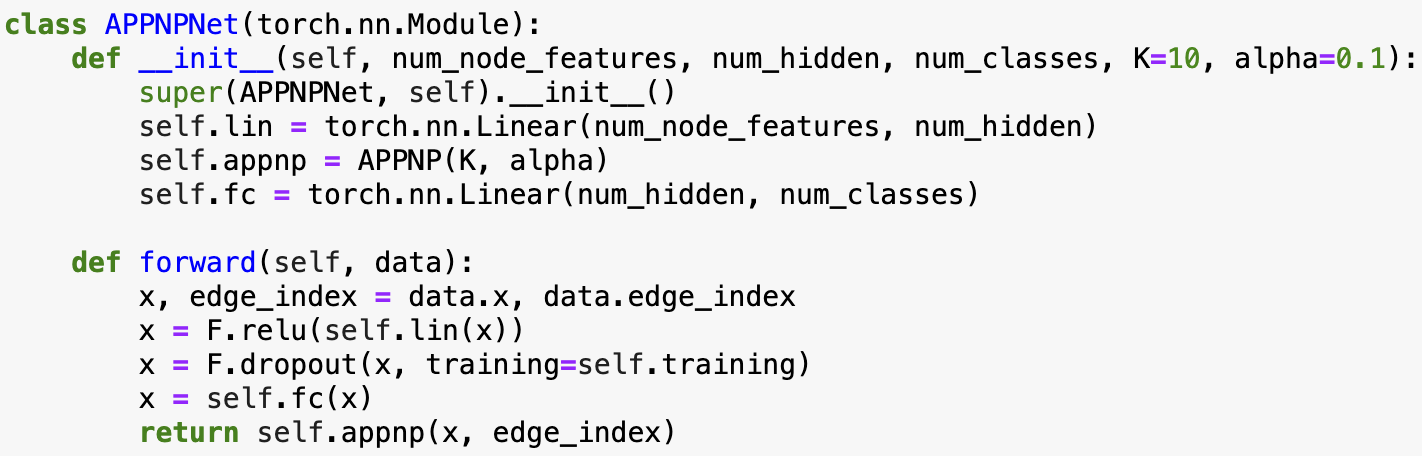


Figure 3: Code snippet showing the implementation of APPNP model.

* The Approximate Personalized Propagation of Neural Predictions (APPNP) model is a type of graph neural network designed to leverage both local node features and the global graph structure through a personalized PageRank scheme. The architecture of the model is discussed below.
* Linear Transformation:
* This layer linearly transforms the input node features into a higher-dimensional hidden space defined by num\_hidden.
* This serves as an initial feature extractor that projects features into a space where the subsequent layers can more effectively learn complex patterns.
* Activation and Dropout
* ReLU activation function is applied to introduce non-linearity, essential for learning complex functions.
* Dropout is used right after the activation during training to prevent overfitting by randomly zeroing some of the elements of the input tensor.
* Second Linear Transformation
* Another linear transformation is applied to the outputs from the dropout layer. It maps the hidden representations to the number of classes, this helps in preparing the feature representation for the final classification.
* APPNP Layer
* This layer uses the methodology of personalized PageRank to propagate node features through the graph for K iterations, blending local and more distant node information based on the teleport probability alpha.
* This allows information to flow over the network, enabling distant nodes to contribute to each node’s feature representation, which is particularly beneficial for capturing global graph properties.

**RESULTS:**

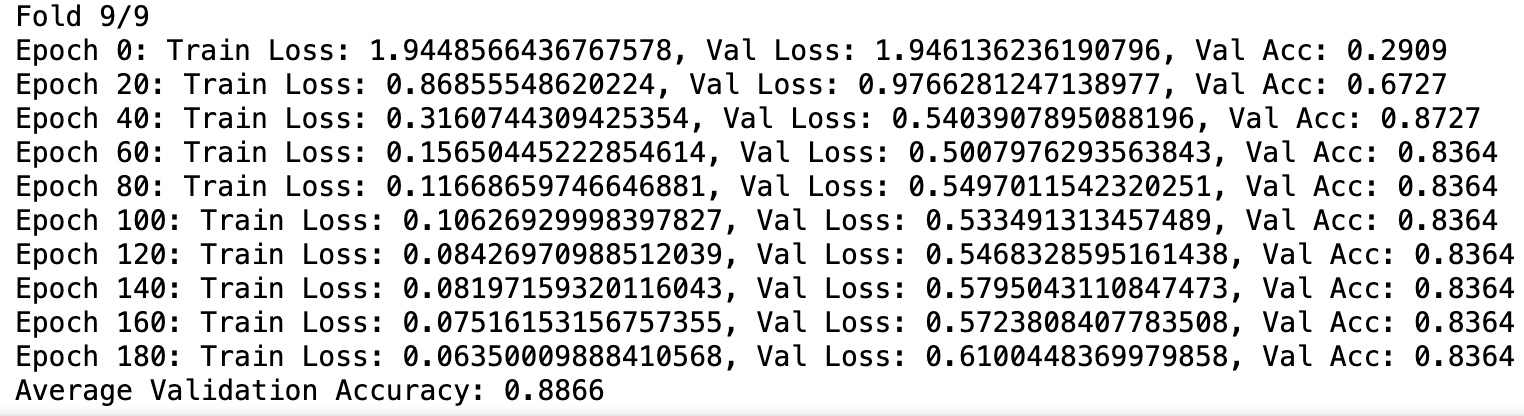
****

Figure 4: Average validation accuracy of the APPNP Model.

1. **GAT Model**

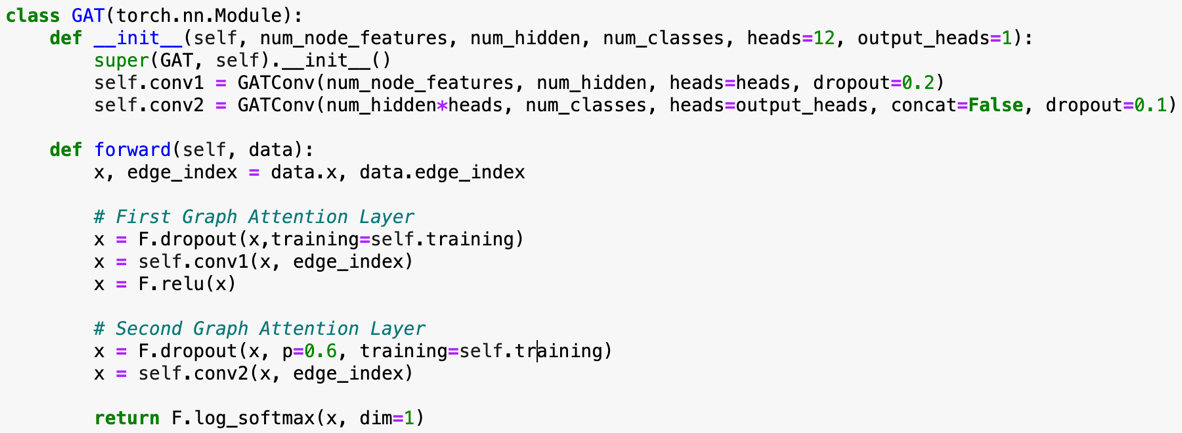


Figure 5:Code snippet showing the implementation of GAT model.

* The Graph Attention Network (GAT) model leverages attention mechanisms within a graph neural network framework. The key idea is to assign different importance to nodes in a neighborhood, allowing the model to focus more on relevant features for each node.
* First Layer(conv1)
* This layer uses a multi-head attention mechanism with heads=12 to transform node features into a higher-dimensional space (num\_hidden). The multi-head setup allows the model to learn different aspects of the feature relationships simultaneously.
* This captures diverse patterns and relationships in the node features by focusing on different parts of the node's neighborhood, which can be crucial for understanding complex interactions in graph data.
* Second Layer(conv2)
* The second layer reduces the dimensionality to the number of classes (num\_classes), essentially preparing for the final output. This layer uses a single output head (output\_heads=1), focusing on integrating the information learned across multiple heads in the first layer.
* The purpose is to aggregate the diverse features learned by multiple attention heads into class scores, critical for classification tasks.

**RESULTS:**

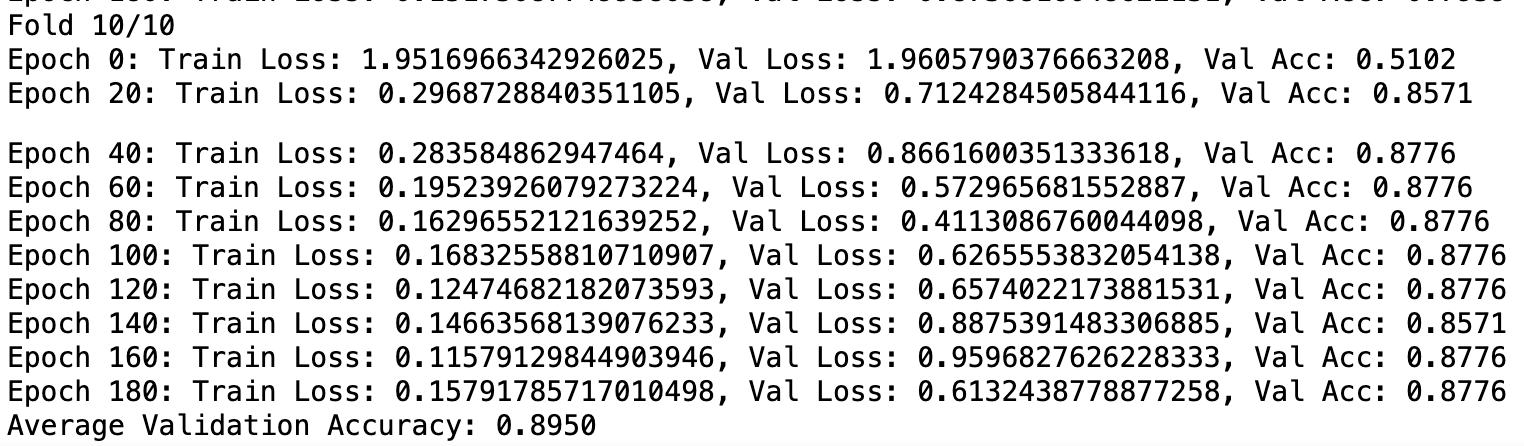
****

Figure 6: Average validation accuracy of the GAT Model.

1. **GSage Model**

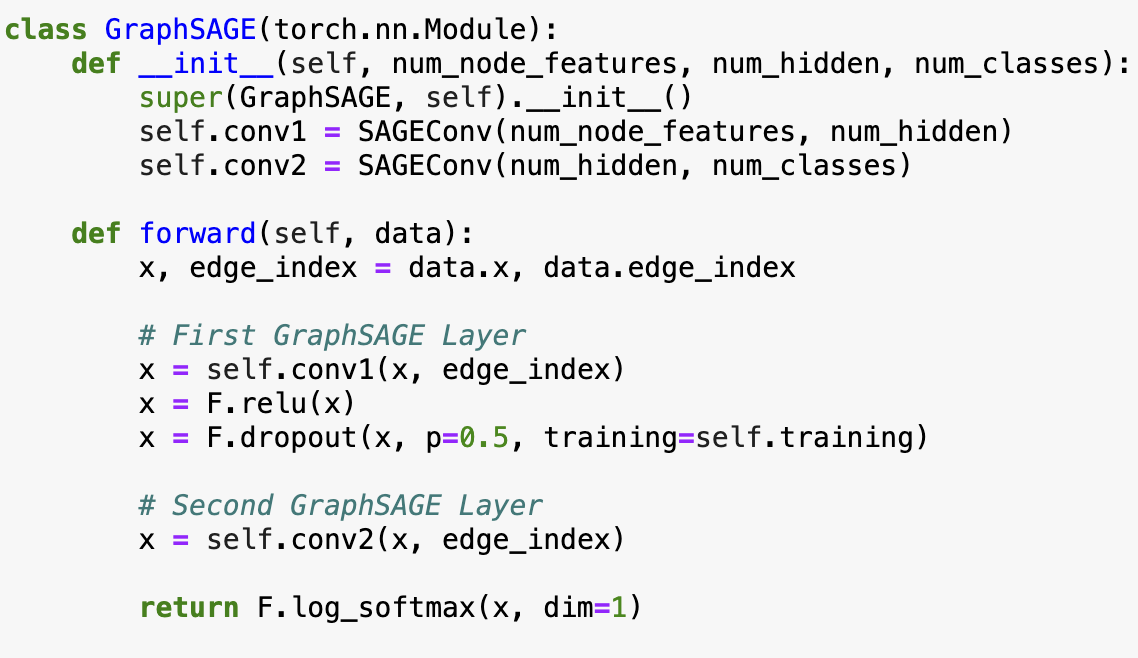


Figure 7: Code snippet showing the implementation of GSage model.

* GraphSAGE (Graph Sample and Aggregate) adapts the neural network approach to graph data by leveraging node feature information with local neighborhood functions. This method enables learning from large graphs effectively.
* First Layer
* This layer uses the SAGEConv module to perform convolution over nodes, aggregating features from the neighbors and combining them with the node's own features.
* This layer is to capture local neighborhood structures efficiently, which is crucial for understanding the immediate context of each node.
* Activation and Dropout
* A ReLU activation function is to introduce non-linearity, allowing the model to learn more complex patterns.
* Dropout is applied after activation with a probability of 0.5, serving as a regularization technique to prevent overfitting during training.
* Second Layer
* The second layer transforms the output from the previous layer directly into the class scores for each node.
* This is to refine and utilize the aggregated features from the first layer for final node classification.

**RESULTS:**

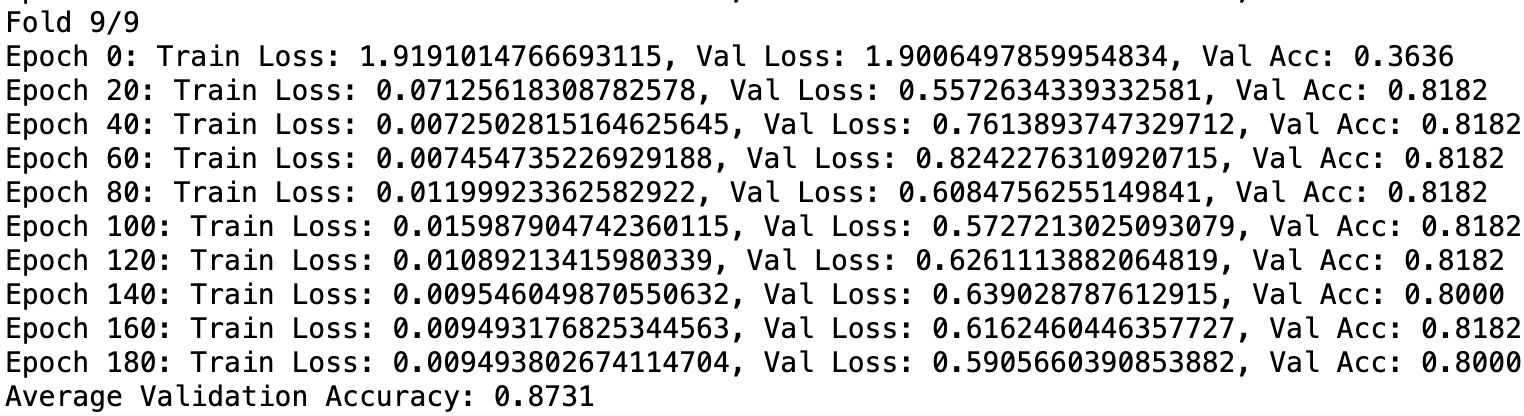
****

Figure 8:Average validation accuracy of the GSage Model

1. **Stacking of GCN, GAT, APPNP, GSage Models**

* In predictive modeling, especially in complex domains like graph data, a single model might not capture all the cases or might be prone to overfitting on particular subsets of the data. Ensemble learning addresses these challenges by combining multiple models to improve the reliability and accuracy of predictions. One effective method within ensemble learning is majority voting.
* The objective of employing a majority voting ensemble method was to harness the collective strengths of various graph neural network architectures to enhance the prediction stability and accuracy across the node classification tasks. This approach aimed to tackle the weaknesses of individual models and reduce the likelihood of wrong predictions influenced by specific model biases.
* Below is a brief description of the majority voting ensemble used in our project.

1. **Model Selection and Training**:

• The graph neural network models including GCN (Graph Convolutional Network), GAT

GraphSAGE, and APPNP was chosen. Each model captures different aspects of graph structure and node features, providing varied perspectives on the data.

• Each model was trained separately on the graph data across 10 different folds, ensuring a robust validation of model performance and reducing the risk of overfitting to specific data partitions.

2. **Prediction Aggregation**:

• Post-training, each model was used to predict the class labels on the same dataset to ensure that the predictions were based on consistent data exposure.

• The predictions from each model were collected and stored for aggregation. This step is crucial as it sets the stage for combining the insights gained from multiple models.

3. **Majority Voting**:

• The final predictions for each node were determined by majority voting among all models. This method calculates the mode of the predicted labels for each node across all models, choosing the most frequently predicted label as the final output.

• Majority voting is particularly effective in reducing noise and errors that might be present in individua model predictions. It leverages the collective intelligence of the ensemble, leading to more reliable and often more accurate predictions.

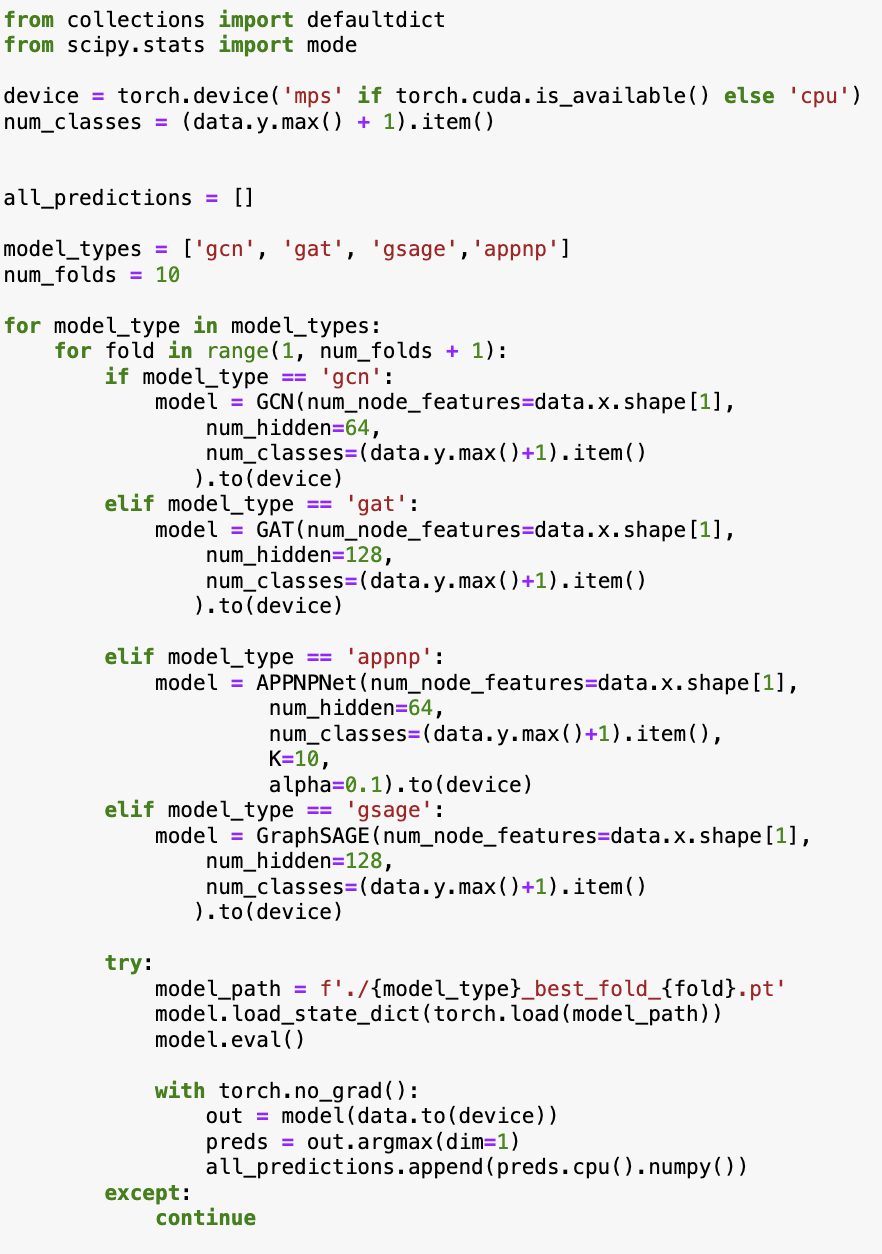


Figure 9: Code snippet of stacking the models.

1. **Key Takeaways**
2. Feature Engineering Impact:

* This step was critical for enhancing model performance by providing context-rich inputs.
* Hence, Demonstrates the importance of understanding data characteristics to optimize model inputs effectively.

1. Diverse Model Architectures:

* Utilizing various graph neural network models (GCN, GAT, GraphSAGE, APPNP) showcase the significance of architectural choices.
* Each architecture offers unique methods for aggregating node information and processing graph structure, highlighting different strategies for data interpretation.

1. Robust Training Procedures:

* Essential for achieving optimal model performance and reliability.
* Incorporating stratified k-fold cross-validation ensured thorough validation and helped mitigate overfitting.

1. Learning Experience:

* The project really highlighted the importance of detailed planning and careful execution throughout the entire process, from how we handle our data to how we evaluate our models.
* It taught us how closely linked the architecture of the models, the engineering of the features, and the procedures for training are when it comes to developing strong machine learning solutions, especially with complex graph data.

1. **Future Directions**

* Looking ahead, here are some exciting advancements we plan to explore to enhance our graph neural network models:

1. Semi-Supervised Learning:

* We aim to implement semi-supervised learning methods to make better use of our limited labeled data alongside the abundant unlabeled data. This approach should help improve our models’ ability to generalize across different datasets.

1. Advanced Networks with Attention Mechanisms:

* By incorporating advanced architectures like attention mechanisms, we hope to deepen our models' understanding of node relationships. This should allow for more dynamic and precise modeling of graph structures.

1. Transformer-Based Models:

* We're also interested in experimenting with transformer-based models, renowned for their success in sequence-based tasks, to capture complex patterns in graph data more effectively.
* These future steps will help us dive deeper into graph data, aiming to push the limits of what our current models can achieve.