CSE 881 Data Mining Project Report

Team Members:

- **Aarthi Padmanabhan:** Data preprocessing, model implementation, hyperparameter optimization
- **Madhurya Shankar**: Model implementation, training process, evaluation metrics
- **Sandhya Kilari:** Model architecture, performance analysis, future directions

Introduction:

Node classification is a fundamental task in the field of graph-based data analysis, offering significant applications in social network analysis, bioinformatics, and recommendation systems, among others. By predicting the labels of nodes based on their features and connections, we can derive meaningful patterns and categorize unseen data effectively. The significance of this task lies in its ability to uncover hidden patterns and categorize nodes effectively, which can be pivotal in understanding and leveraging graph-based data in practical applications.

Dataset Description:

The dataset provided for this project is a graph dataset specifically designed to challenge and test our node classification models. It includes several key components which are crucial for developing and evaluating our predictive algorithms. Below is an overview of these components:

The components of the dataset are:

1. **Adjacency Matrix (adj.npz)**: This matrix represents the graph structure by indicating the connections or edges between nodes. Each entry in this matrix indicates whether a pair of nodes is connected, thereby defining the neighborhood relationships within the graph.
2. **Feature Matrix (features.npy)**: Each node in the graph is associated with a feature vector contained in this matrix. These features provide the attributes or characteristics of the nodes, which are used by the classification models to predict node labels.
3. **Labels (labels.npy)**: This array provides the actual labels for the nodes. Each label corresponds to the category or class that the node belongs to. The dataset includes a total of 7 unique classes into which the nodes can be categorized.
4. **Data Splits (splits.json)**: The dataset is divided into predefined training and testing splits, as detailed in this JSON file. It specifies which nodes should be used for training the models and which should be reserved for testing their performance. The training set comprises approximately 20% of the nodes, while the testing set includes the remaining 80%.

Dataset Statistics:

- Number of Nodes: 2480
- Number of Edges: 10100
- Number of Classes: 7
- Number of Features per Node: 1390
- Training/Test Split: 496 nodes for training and 1984 nodes for testing.

Methodology:

A Graph Convolutional Network (GCN) was implemented using the PyTorch framework, leveraging the specialized graph-based neural network layers provided by PyTorch Geometric. GCNs are particularly effective for node classification tasks because they are designed to work directly with the graph structure, enabling the model to learn node representations by leveraging both node features and the graph topology. This approach facilitates a deep integration of the relational information present in the graph with the node-specific features, optimizing the prediction of node labels based on their contextual and intrinsic properties.

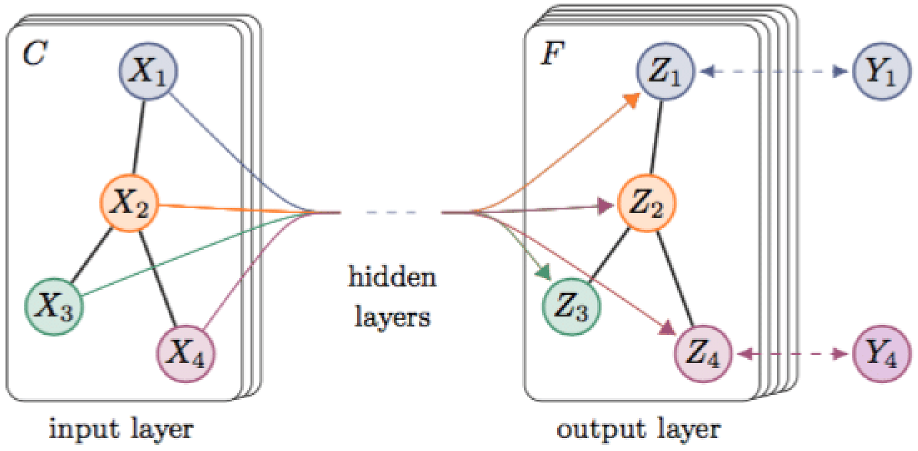


Figure 1: Graph Convolutional Network (GCN)

Model Implementation:

- **Data Preparation**
  The graph dataset provided includes an adjacency matrix, feature matrix, and node labels, all of which were loaded and processed to conform to the structure required for the Graph Convolutional Network (GCN) model. The dataset came with a predefined train-test split; however, the training data was further divided to create a validation set. This additional split aids in model tuning and helps to prevent overfitting by providing a way to validate model performance during the training process. The adjacency matrix of the dataset was converted into an edge index format that is compatible with PyTorch Geometric. This format is essential for the effective handling of graph data within the framework. Furthermore, node features were encapsulated into tensors to facilitate computation, and separate masks were created for the training, validation, and testing phases. These masks are crucial for managing data flow during the model's training and evaluation, ensuring that each phase of the model lifecycle receives the appropriate data.

- **Model Architecture**
  The adjacency matrix of the dataset was converted into an edge index format that is compatible with PyTorch Geometric. This format is essential for the effective handling of graph data within the framework. Furthermore, node features were encapsulated into tensors to facilitate computation, and separate masks were created for the training, validation, and testing phases.

These masks are crucial for managing data flow during the model's training and evaluation, ensuring that each phase of the model lifecycle receives the appropriate data.

- **Training Process**
  Training the model involved the use of the Adam optimizer, which is known for its effectiveness in handling sparse gradients on noisy problems. Hyperparameters were carefully optimized to minimize the cross-entropy loss, which incorporates class weights to address the issue of class imbalance in the training data. The model underwent multiple epochs of training, during which it learned to minimize loss on the training data while its performance was regularly checked against the validation set to monitor for signs of overfitting.

- **Hyperparameter Optimization**
  To achieve the best performance from the Graph Convolutional Network (GCN), Bayesian optimization was employed to systematically search for the optimal set of hyperparameters. This technique uses a Bayesian approach to model the objective function and applies a probabilistic model to predict the performance of the model over the hyperparameter space. By doing so, Bayesian optimization efficiently finds the hyperparameters that maximize the model's accuracy while reducing the number of necessary evaluations when compared to grid or random search methods.

  In this project, Bayesian optimization was particularly useful due to the complex nature of the hyperparameter interactions in deep learning models. It helped in determining the ideal learning rate, number of hidden layers, dropout rate, and other crucial parameters without extensive manual experimentation. This method not only saved valuable computational resources but also significantly improved the model's performance by fine-tuning the parameters more effectively than traditional methods could.

Evaluation Metrics:

- **Performance Metrics**
  To evaluate the performance of the Graph Convolutional Network (GCN) model, several metrics were employed, with a primary focus on accuracy. Accuracy is crucial as it measures the proportion of total correct predictions made by the model across all classes, providing a straightforward assessment of overall effectiveness. In addition to accuracy, other important metrics such as precision, recall, and the F1-score were utilized to gain a more nuanced understanding of the model's performance. Precision highlights the model's ability to identify only relevant instances, while recall focuses on the model's capability to find all relevant instances within the dataset. The F1-score, a harmonic mean of precision and recall, serves as a balance between the two, particularly in situations where the class distribution is uneven. These metrics together offer a comprehensive view of the model's predictive accuracy and its operational efficiency across different class labels.

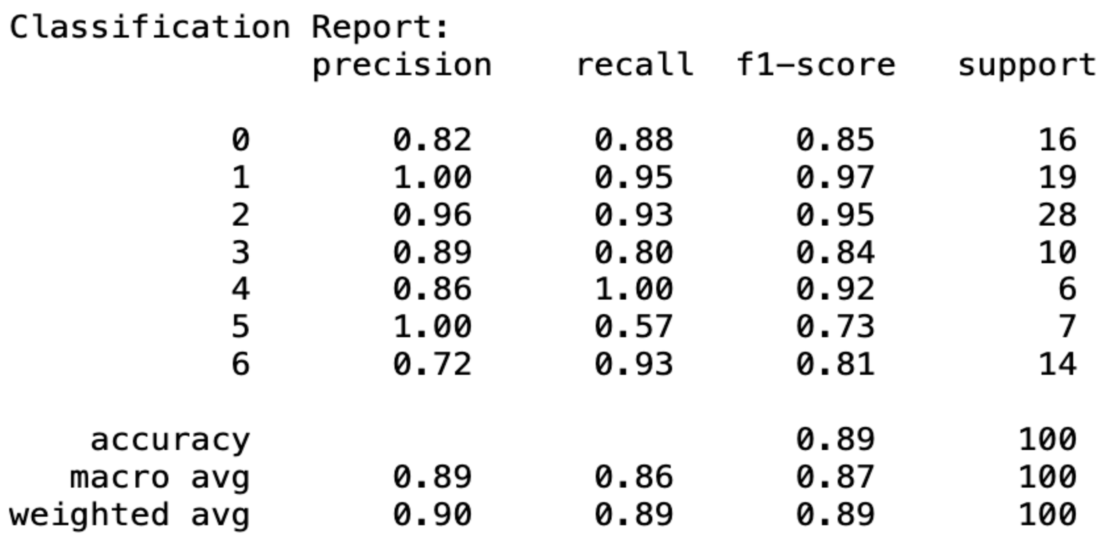
- **Validation Strategy**
  The validation set played a pivotal role in fine-tuning the model and preventing overfitting, which is crucial for maintaining the model's ability to generalize to unseen data. During the training process, the model's performance was continuously monitored on this validation set. This practice allowed for the adjustment of hyperparameters and the early stopping of training when the validation loss ceased to decrease, indicating that the model had begun to overfit to

the training data. By using the validation set in this manner, it was possible to ensure that adjustments to the model's architecture and training parameters were beneficial and that the model remained robust against overfitting. This strategic use of the validation set not only improved the model's performance on the training data but also enhanced its generalization capabilities on new, unseen data, as reflected in the test performance metrics.

Results:

The Graph Convolutional Network (GCN) exhibited strong performance in the node classification task, as detailed in the classification report. The model achieved an overall accuracy of 89%, which indicates a high level of correctness in predicting the labels for the test set. Below is a breakdown of the performance metrics for each class, which provides deeper insights into the model's effectiveness and areas for improvement:

- **Class 0 (Precision: 0.82, Recall: 0.88, F1-score: 0.85)**: The model performed well in identifying this class, showing a good balance between precision and recall. The high recall indicates that the model was able to identify most of the actual instances of this class.
- **Class 1 (Precision: 1.00, Recall: 0.95, F1-score: 0.97)**: This class saw excellent performance with perfect precision, indicating that every prediction made by the model for this class was correct. The slightly lower recall suggests a few instances were missed.
- **Class 2 (Precision: 0.96, Recall: 0.93, F1-score: 0.95)**: Another strong performer, Class 2 had high precision and recall, demonstrating the model's ability to accurately identify and classify nodes in this category.
- **Class 3 (Precision: 0.89, Recall: 0.80, F1-score: 0.84)**: While still good, the performance metrics for Class 3 suggest some room for improvement, especially in increasing the recall.
- **Class 4 (Precision: 0.86, Recall: 1.00, F1-score: 0.92)**: The model showed excellent recall for Class 4, identifying all actual instances correctly, though precision could be slightly improved.
- **Class 5 (Precision: 1.00, Recall: 0.57, F1-score: 0.73)**: The model was highly precise in its predictions for Class 5, but the recall was notably lower, indicating that it missed a significant number of actual instances, which negatively affected the F1-score.
- **Class 6 (Precision: 0.72, Recall: 0.93, F1-score: 0.81)**: Despite lower precision, the model had high recall for Class 6, suggesting it was able to identify most of the actual instances, but at the cost of more false positives.



```
Classification Report:
              precision    recall  f1-score   support

           0       0.82      0.88      0.85        16
           1       1.00      0.95      0.97        19
           2       0.96      0.93      0.95        28
           3       0.89      0.80      0.84        10
           4       0.86      1.00      0.92         6
           5       1.00      0.57      0.73         7
           6       0.72      0.93      0.81        14

    accuracy                           0.89       100
   macro avg       0.89      0.86      0.87       100
weighted avg       0.90      0.89      0.89       100
```

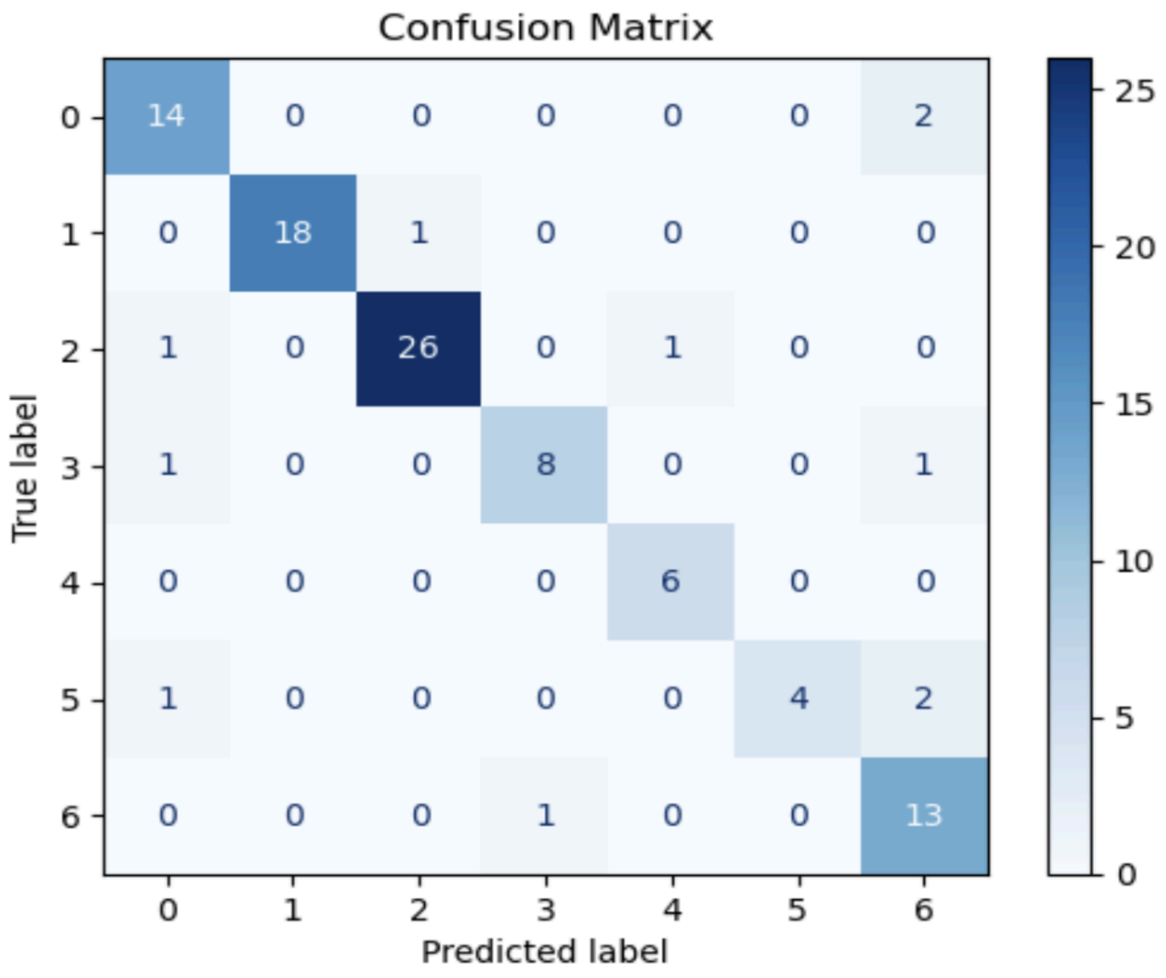


Figure 2: Classification Performance Metrics and Confusion Matrix for Node Label Predictions

The macro and weighted averages for precision, recall, and F1-score (all approximately 0.89) reflect a balanced performance across classes, although the variance among individual classes points to specific areas where model tuning might focus on improving either precision or recall. This detailed evaluation helps in understanding the model's strengths in correctly predicting certain classes while highlighting opportunities to refine its approach to others, particularly those with lower recall or precision scores.

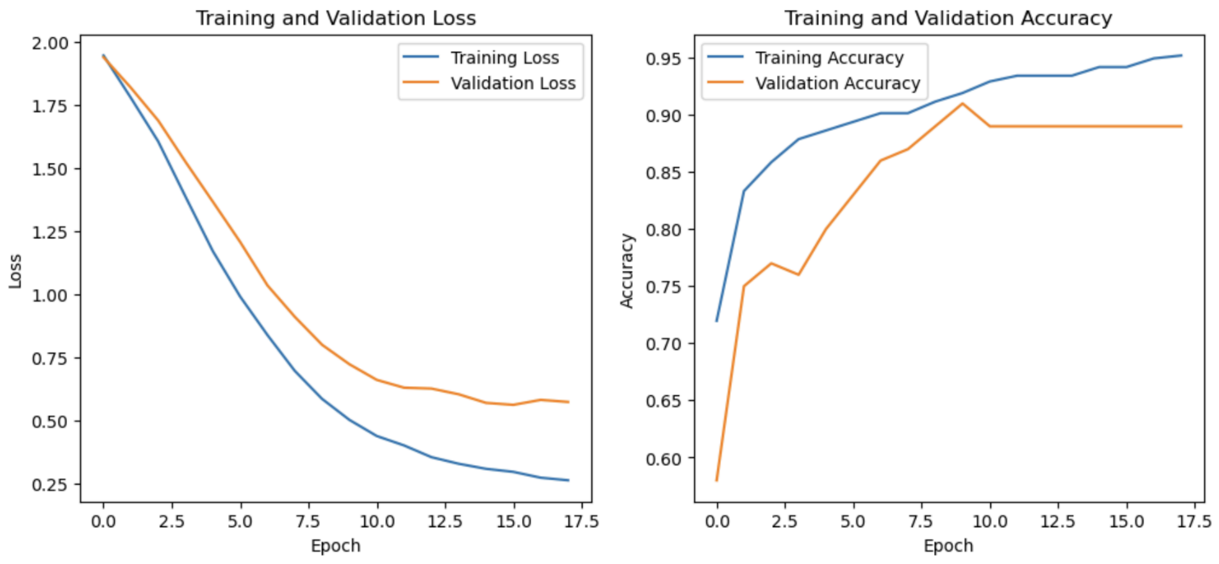


Figure 3: Training and Validation Loss and Accuracy Over Epochs for GCN Model

The first graph, titled "Training and Validation Loss," shows the loss on the training set and the validation set as the number of epochs increases. The loss is an indicator of how well the model is fitting the data: lower values suggest better fit. In this case, both the training and validation loss decrease sharply initially, which indicates rapid learning. As epochs continue, the training loss plateaus, suggesting that the model has started to converge on a solution. The validation loss, while generally following the training loss, begins to level off, which is a sign that the model might be beginning to overfit or has learned as much as it can from the data. The convergence of training and validation loss at a low level is a positive sign, indicating good generalization without significant overfitting.

The second graph, "Training and Validation Accuracy," presents the accuracy of the model on both the training and validation sets across epochs. The accuracy metric represents the proportion of correct predictions made by the model. Initially, the accuracy for both datasets improves rapidly, with the training accuracy typically being higher than the validation accuracy, which is expected as the model is directly learning from the training data. As the number of epochs increases, the training accuracy continues to rise, whereas the validation accuracy experiences some fluctuations. This behavior suggests the model may be beginning to overfit to the training data; it is getting better at predicting the data it has seen but not necessarily improving on unseen data.

Conclusion & Future Work:

The exploration into graph-based machine learning with a Graph Convolutional Network (GCN) has yielded significant insights. The project underscored the effectiveness of GCNs in handling the intricate dependencies within graph data, achieving an impressive accuracy of 89%. Key findings highlight the model's proficiency in precision, though it also revealed challenges with recall for certain classes. The successful application of Bayesian optimization in hyperparameter tuning played a crucial role in enhancing model performance, affirming the potential of probabilistic methods in machine learning pipelines.

Future endeavors should concentrate on extending the model's capabilities and addressing its limitations. Investigations could include experimenting with more advanced graph neural network architectures and employing techniques to better address class imbalance. Further research might also involve the incorporation of unsupervised or semi-supervised learning to enrich the model's understanding of the dataset. Emphasis on real-world applications and scalability will be vital for transitioning from academic insights to practical solutions, driving the next phase of innovation in graph-based machine learning.

References:

- https://towardsdatascience.com/graph-convolutional-networks-introduction-to-gnns-24b3f60d6c95
- https://medium.com/basecs/a-gentle-introduction-to-graph-theory-77969829ead8
- https://medium.com/dvt-engineering/hyper-parameter-tuning-for-scikit-learn-ml-models-860747bc3d72
- https://towardsdatascience.com/a-conceptual-explanation-of-bayesian-model-based-hyperparameter-optimization-for-machine-learning-b8172278050f
- https://towardsdatascience.com/7-hyperparameter-optimization-techniques-every-data-scientist-should-know-12cdebe713da