Graph Classification: An Exploration on the MUTAG Dataset

Stefan Krsteski
Department of Computer Science, EPFL, Switzerland

I. Introduction

Graphs are a way to show how things are connected, like how atoms are linked in molecules. In biomedicine, understanding these connections is crucial for predicting how molecules behave in the body. The MUTAG dataset consists of chemical compounds represented as graphs, where nodes denote atoms and edges represent chemical bonds. It is primarily used to determine whether these compounds have mutagenic effects, meaning they can change the DNA in organisms. Each graph in the dataset is labeled as either mutagenic or non-mutagenic based on this property. In this report, we will look at how well GNNs can classify graphs using the MUTAG dataset.

II. Methods

In the exploration of graph classification two primary approaches were adopted: Nodes only classification; Node and Edge feature classification. The foundational layers of the models explored in the experiments are: Normal Convolution, GraphSAGE and Attention-based Convolution. These layers were used in both approaches, although when incorporating edge features they were slightly altered.

III. Incorporating Edge Features

In this subsection I will thoroughly explain the different methods for incorporating edge features used in my experiments.

1) Frequency-Based Edge Weighting: The straightforward approach computes the frequency of the edge features and assigns constants based on that frequency. This is done by iterating the train set and counting the number of appearances of each of the four vectors. Rare vectors are emphasized by assigning smaller constants to vectors with larger frequencies. Then, the ones in the adjacency matrix (indicating that there is an edge between two nodes) are replaced with these frequency based constants. This method is more of a naive approach since I assume that rare features are more relevant.

2) Node Augmentation via Edge Aggregation: This method enhances the node representations by incorporating aggregated edge attributes for each node in the graph. The process can be broken down into three straightforward steps:

- **Identify Node's Edges**: Firstly, we identify the edges connected to a specific node within the graph.
- **Aggregate Edge Features**: Next, aggregate the edge features associated with a specific node. This aggregation can be carried out using methods like mean, max, sum, or any combination between these three.
- **Concatenate with Node Features**: Finally, we concatenate the result of the aggregation with the node's original feature vector. This concatenation creates an augmented node representation that incorporates both the node attributes and its aggregated edge attributes.

3) Adaptive Embedding and Aggregation: A more dynamic approach utilizes learnable weights in the adjacency matrix. Edge features are embedded to match node feature dimensions and then merged via sum. Subsequently, a neural network, influenced by a Tanh activation function (to introduce non-linearity), processes the concatenated features to derive scalar weights. These weights dictate the significance of each edge feature in relation to its linked node features during neighbor aggregation.

The process can be broken down into the following steps:

- **Embed Edge Features**: First, a neural network embeds the edge features into the node feature dimension.
- **Merge Edge Features with Node Features**: These transformed edge features are then merged with the node features using summation. This results in a merged vector containing both node and edge features.
- **Weight Representation**: The merged vector is fed into a neural network that represents it as a scalar value. The adjacency matrix is then modified to include this value instead of the default ”1”.
- **Update Adjacency Matrix**: The modified adjacency matrix, with scalar weights, is used in the core layers defined above.

IV. Experimental Setup

A. Data Splitting

The MUTAG dataset consists of 188 graphs in total, of which 125 are mutagenic and 63 are non-mutagenic. The dataset was divided into training (70%), validation (15%), and test sets (15%). Due to the data's imbalance, a stratified split was employed to ensure the class distribution remains consistent across these splits.

B. Baseline Model

The baseline model is characterized by a single-layer GraphConv using only node features, with one layer and a hidden feature size of 32. The optimizer employed is Adam with a learning rate of 0.01, and the model is trained for 30 with a batch size of 1. This configuration was selected as the baseline due to its simplicity.

C. Hyperparameter Optimization Setup

A grid search was conducted over a predefined parameter space to optimize hyperparameters. The explored hyperparameters included:

- Number of Layers: [1, 2, 3, 4]
- Learning Rate: [0.1, 0.01, 0.001]
- Layer Type: [GraphConv, AttentionConv, GraphSAGE]
- Hidden Features: [16, 32, 64]
- Pooling: [mean, max]

V. Results

A. Nodes-Only Classification

This approach leveraged only node features while ignoring edge attributes. Table I showcases the performance metrics of the models that were explored using this approach.

Layer Type	Avg. Val. Accuracy	Avg. Val. F1 Score
GraphConv	74.6%	83.93%
GraphSAGEConv	72.81%	83.03%
AttentionConv	73.66%	83.76%

Table I
Performance of models using nodes-only classification.

Figure 1 showcases a boxplot comparison for the average validation F1 scores of the three different layers across all possible network configurations (hyperparameter search space): Attention-based Convolution, Normal Convolution, and GraphSAGE. From the results presented in Table I and Figure 1, we can deduce that the GraphConv layer outperforms the other two layers, both in terms of mean performance and in the upper quartile performance. Also, it achieved the highest validation F1 score of 90.47%. The best model from the nodes-only classification will be with 3 Graph Convolution layers, each with 16 hidden features and with max pooling.

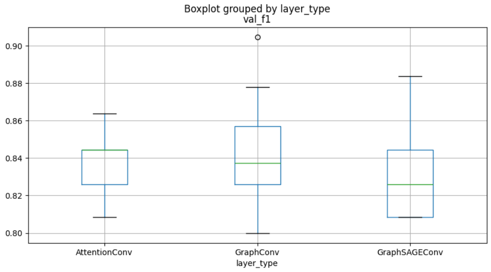

Figure 1. Boxplot comparison of validation F1 scores for the three different layer types.

Table II showcases the performance metrics of the baseline and the optimized models on the test set. As evident, the best model outperforms the baseline, suggesting that the optimization process was fruitful. Specifically, the F1 score for the best model is about 5% higher, and its accuracy is approximately 7% better than the baseline.

Layer Type	Avg. Val. Accuracy	Avg. Val. F1 Score
Baseline	78.05%	68.97%
Best Model	82.93%	75.86%

Table II
PERFORMANCE ON THE TEST SET

B. Node and Edge Feature Classification

In the Methods section, three different edge feature implementations were presented. By observing the boxplot for validation F1 scores across the different edge feature implementations, a few conclusions can be drawn.

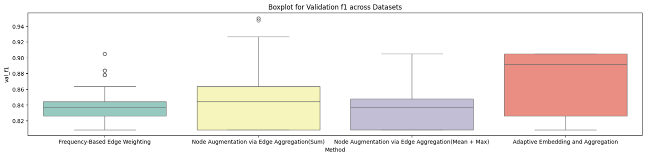

Figure 2. Boxplot comparison of validation F1 scores for the three different methods (Method 2 has two variants, one with sum and the other with mean+max

From Figure 2 we can see that two edge feature implementations particularly stand out: ”Adaptive Embedding and Aggregation”, ”Node Augmentation via Edge Aggregation(Sum)”. Because the peak performance is attained using the ”Node Augmentation via Edge Aggregation(Sum)” method we will use both in the final performance evaluation. The highest F1 score from both methods on the validation set is 90.47% and 95% correspondingly.

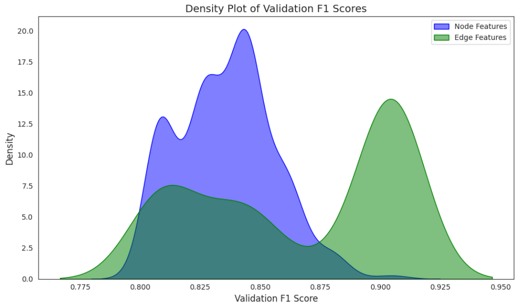

Figure 3. Density plot showing the comparison between the distribution of validation F1 scores for models using node features and those using edge features.

Figure 3 shows how well models using node features (blue) and edge features (green) perform. Models using edge features tend to do a bit better on average. There are areas where both types perform similarly, but edge features often have a slight edge.

C. Final Performance Evaluation on the Test Set

Having observed the performance of different models on the validation set, it is important to evaluate their effectiveness on the test set. This ensures the models are capable of handling unseen data and provides a true reflection of their generalizability.

Method	Test Acc.	Test F1
Baseline	68.96%	78.04%
Best Nodes Model	75.86%	82.92%
Adaptive Embedding and Aggr.	75.86%	82.05%
Node Augmentation via Edge Aggr.(Sum)	82.75%	87.17%

Table III
PERFORMANCE METRICS OF MODELS ON THE TEST SET.

The results from the test set show that the ”Node Augmentation via Edge Aggregation(Sum)” method is leading in terms of both F1 score and accuracy. This method achieved an F1 score of 87.17% and an accuracy of 82.76%, outperforming the best nodes model and the ”Adaptive Embedding and Aggregation” method.

The observation that a simpler method of aggregating edge features to nodes has surpassed a more complicated embedding and transformation process is intriguing. The performance on the test set of the ”Adaptive Embedding and Aggregation” method is the same as the best nodes model, however, on average, the former outperforms the later when comparing on the validation set. Judging from the results, it is safe to conclude that incorporating edge features does indeed enhance model performance.

VI. CONCLUSION

This work shows that using edge features in our model helps improve its performance. Compared to the baseline model, we saw a 9% better F1 score. It’s interesting that a simple method of adding edge features to nodes did better than a more complicated approach. But overall, the complicated method works better (on average).

It is important to note that with only around 28 samples each for validation and testing, the results may carry a degree of uncertainty. Future experiments with larger datasets would provide more robust and generalized insights.