Homework 4: Graph Classification

Name: Anay Abhijit Joshi

Step 1: Data

1) How many data samples are included in the dataset?

The dataset contains **600 graphs**. This corresponds to the number of lines in the "ENZYMES_graph_labels.txt" file, where each line represents the label of a graph.

```
[ ] # Count the number of graphs/data samples in the given dataset
    num_graphs = len(graph_labels_data)
    print(f"Total Number of Data Samples in the dataset: {num_graphs}")
```

```
Total Number of Data Samples in the dataset: 600
```

2) Which problem will this dataset try to address?

This dataset addresses a **graph classification problem**. The goal/task is to assign each graph (enzyme) to one of 6 top-level EC classes.

3) What are nodes, edges and features in your dataset?

In the given dataset, the "nodes" represent the elements of the Enzyme structure, the "edges" represent their connections/relationships, and "features" represent the node attributes providing additional properties.

So, there are 19,580 nodes; 74,564 features; and 18 features per node; as I have depicted below.

```
[ ] # Number of nodes and edges and features in the dataset
    num_nodes = len(graph_indicator_data)
    num_edges = len(edges)
    num_features = node_attributes_data.shape[1]

    print(f"Number of nodes: {num_nodes}")
    print(f"Number of edges: {num_edges}")
    print(f"Number of features per node: {num_features}")
```

```
Number of nodes: 19580
Number of edges: 74564
Number of features per node: 18
```

4) Does the dataset have any missing information? E.g., missing features.

No, the dataset does not have any missing information or missing features.

```
[ ]  # Check for missing node attributes
     if np.isnan(node_attributes_data).any():
         print("Missing node attributes detected.")
     else:
         print("No missing node attributes.")
```

```
No missing node attributes.
```

5) What is the label of this dataset?

The dataset contains **graph-level labels**. Each graph is labeled as one of 6 EC top-level enzyme classes, with labels ranging from **1 to 6**.

In programming terms, I have written them as Class 0-5 in the plot's labels due to zero-based indexing used in Python. To match the dataset's original labels, I can adjust the labels by using **{i+1}**, which will display the labels as 1, 2, 3, 4, 5, 6; where 'i = 0, 1, 2, 3, 4, 5'.

```
[ ]  # Unique class labels in the dataset
     unique_labels = set(graph_labels_data)
     print(f"Unique Labels (classes): {unique_labels}")
```

```
Unique Labels (classes): {1, 2, 3, 4, 5, 6}
```

6) How many percent of data will you use for training, validation and testing?

Typically, I would use **70% for training**, **15% for validation**, and **15% for testing**. This split ensures a balanced approach for model training and evaluation.

7) What kind of data pre-processing will you use for your training dataset?

For my training dataset, I normalize the node features to have zero mean and unit variance to ensure consistent model training. Additionally, I apply Principal Component Analysis (PCA) to reduce the dimensionality of the node features to the top 5 components, improving computational efficiency. I split the dataset into 70% training, 15% validation, and 15% testing to ensure unbiased evaluation. Finally, I use data batching with a batch size of 32 for efficient loading and processing during training and testing.

Step 2: Model

Model	F1	Accuracy
GCN-1 layer	**0.50496**	**0.54056**
GCN-2 layer	**0.55948**	**0.55056**
GCN-3 layer ***(extra)***	**0.55204**	**0.56056**
GCN-4 layer ***(extra)***	**0.51891**	**0.52000**

Step 3: Objective

Cross-Entropy Loss

For my model, I use the cross-entropy loss function to train the graph classification task. This loss function is suitable for multi-class classification, as it calculates the difference between the predicted class probabilities and the true labels. It ensures the model learns to make accurate predictions by penalizing incorrect outputs and encouraging high confidence in the correct class predictions.

Step 4: Optimization

Adam Optimizer

For Homework 4, again, I selected the Adam optimizer to train my GCN models (with layers 1, 2, 3, and 4). The reason for choosing Adam lies in its adaptive learning rate capabilities, which enable efficient training by dynamically adjusting learning rates for each parameter. Although I used a fixed learning rate of 0.01 across all models for consistent testing and evaluation, Adam's momentum and adaptability provided stability during training and ensured robust convergence.

Adam's ability to handle sparse data effectively, such as the graph-structured ENZYMES dataset, was crucial for achieving consistent performance. By maintaining the same learning rate, I ensured a fair comparison of models while leveraging Adam's reliability for training these models. Its efficiency and adaptability made it the ideal choice for this multi-class graph classification task.

Step 5: Model selection

Best Model:

No, I did not use different learning rates for consistent testing and evaluation of these models. The **learning rate** was fixed at **0.01** to ensure a uniform comparison across all GCN models (with Layers as 1, 2, 3, 4).

Based on the table, in Step 2, the **GCN 3-layer model** has the highest **accuracy of 0.56056** and a F1 score of 0.55204, making it the best-performing model overall. However, the **F1 score for the GCN 2-layer model is slightly higher at 0.55948**, with an accuracy of 0.55056, indicating that it handles class balance more effectively. Based on these models, their results suggest that the performance improves from GCN 1-layer to GCN 3-layers but starts to decline significantly with GCN 4-layers, likely due to overfitting or vanishing gradients in deeper models.

Step 6: Model performance

For Homework 4, I evaluated the performance of my GCN models using the AUC-ROC curve. The plot (attached below) illustrates the Receiver Operating Characteristic (ROC) for each of the 6 EC top-level classes (0-5), as Python uses zero-based indexing, along with the micro-average performance. The AUC values for individual classes range from **0.68 to 0.79**, highlighting variations in classification performance across enzyme classes.

- **EC Top-Level Class 0 and 1** achieved an AUC of **0.70**, indicating moderate performance.
- **EC Top-Level Class 3** performed well with an AUC of **0.75**, while **Class 4** achieved the highest AUC of **0.79**, suggesting that the model is particularly effective at distinguishing enzymes in this category. *(Note: Python's Zero-Based Indexing used, so apply, {i+1})*
- **EC Top-Level Class 2** had the lowest AUC of **0.68**, indicating more challenges in classifying this enzyme group.
- The **Micro Avg AUC** was **0.74**, showing overall balanced performance across all classes.

As I stated earlier in the previous Step 5, he **GCN 3-layer model**, identified as the best-performing model in Step 5, achieved the highest accuracy of **0.56056**, which is consistent with the robust overall performance shown by the "Micro Avg AUC". These results demonstrate that deeper GCN models (up to 3 layers), effectively learn and classify enzyme structures, though variations in AUC indicate that some enzyme classes are more challenging to distinguish than others. The AUC values also significantly outperform random guessing (AUC = **0.50**), represented by the diagonal line in the plot (attached below), reinforcing the effectiveness of the GCN models...