# DGL 2025 Coursework 1

**Asia Belfiore**

CID: 02129867 ab6124@ic.ac.uk

Department of Computing
Imperial College London

February 17, 2025

**Abstract**

**Instructions:** This is a structured report template for your DGL 2025 coursework. Please insert your written answers, discussions, and figures in the designated sections. **Do not include any code** in this report. All code should remain in your Jupyter notebooks.

**Note:** We have **kept the structure the same as the Coursework Description PDF** to maintain consistency across your notebooks and this report template. Please keep your headings and subheadings aligned with those in the provided instructions. However, if a section primarily relates to code implementation, you may keep your answers concise (e.g., reference your notebook or provide brief clarifications).

# 1 Graph Classification

## 1.1 Graph-Level Aggregation and Training

### 1.1.a Graph-Level GCN

A function '*global_aggregation()*' was created: based on the '*graph_aggregation_method*' (str) parameter, the function computes the appropriate element-wise aggregation ('**mean**', '**sum**', '**max**', or '**none**') of all the node embeddings (*Tensors*) passed as parameter. The default behaviour (in case no valid aggregation name is passed as parameter) is the '*mean*' aggregation.
In the 'MyGraphNeuralNetwork' class, the '*global_aggregation()*' is called in the forward pass on the output of the last 'MyGCN' layer of the Network, and returns the result of the aggregation of all the learned node embeddings of the graph.
Each aggregation method was implemented using the appropriate built-in **torch** functions. Additionally, self-connections were added to the adjacency matrix of the graph in the forward pass of the Network (in order to comply with the requirements of the forward pass of the GCN layer). Full solutions can be found in the Q1 **.ipynb** file.

**NOTE**: The '*global_aggregation()*' was created outside the 'MyGraphNeuralNetwork' class in order to be available anywhere in the file, and the '*none*' aggregation method was added for testing purposes.

### 1.1.b Graph-Level Training

The given '*train_epoch()*' and '*test()*' functions were merged and adapted into a new function '*run_epoch()*' which, based on a boolean parameter ('*train*'), runs the model's training OR evaluation pipeline for a single epoch. The function returns the average loss, accuracy, true and predicted labels for the current epoch. It optionally returns the learned embeddings (based on a given boolean parameter).
A new function '*train_model()*' as added to perform the entire training, validation and testing pipeline for a given number of epochs. At each epoch, the model is first trained on the given training dataset and then validated on the validation dataset (by sequential calling of the aforementioned '*run_epoch()*' function), storing the loss and accuracy for both steps in four separate lists.
An additional boolean parameter, '*f1*', is added for optional F1 score calculation for both training and validation, based on the true and predicted labels returned by the '*run_epoch()*' function.
Loss, Accuracy and F1 scores for all aggregation methods are shown in Figure 1.
Full solutions can be found in the Q1 **.ipynb** file.

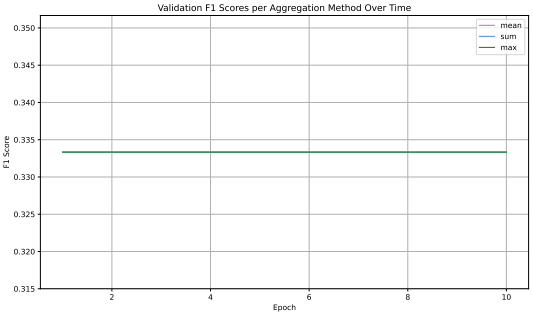
### 1.1.c Training vs. Evaluation F1

The **max** aggregation slightly outperforms the other global aggregations, as it shows slightly higher F1 score for both testing and validation out of all the aggregation methods (Figure 2a). This is probably due to the fact that **max** aggregation retains the most important features of each node, thus allowing the model to learn to distinguish the strongest features of the graph, which characterize it.
Both the **max mean** aggregation methods lead to no increase in the validation F1 and have overall lower F1 scores for both testing and validation (Figures 2b,2c).


(a) Training Loss and Validation Accuracy per epoch with Sum Aggregation.


(b) Training Loss and Validation Accuracy per epoch with Mean Aggregation.


(c) Training Loss and Validation Accuracy per epoch with Max Aggregation.




(d) Validation F1 Score per epoch for every Aggregation method.

Figure 1: Global Sum, Max and Mean Aggregation Loss and Accuracy Comparison

## 1.2 Analyzing the Dataset

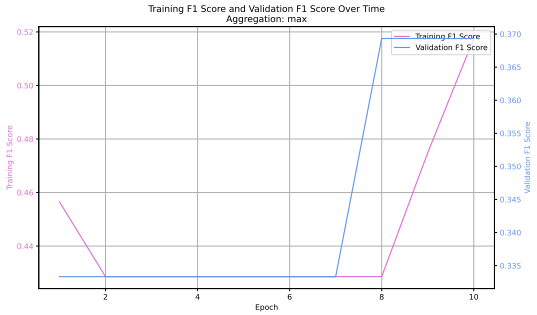
### 1.2.a Plotting

All plots can be found in Figure 3 and in the Q1 **.ipynb** file.

### 1.2.b Discussion

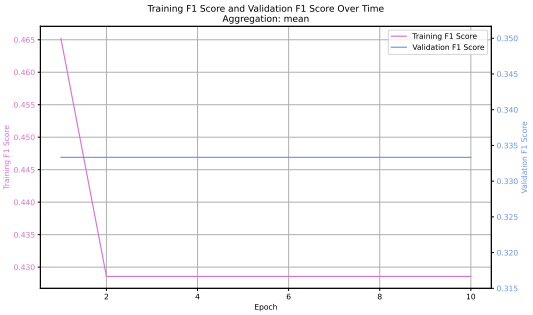
There is an evident class imbalance between the Training and Validation Datasets: while the two node classes (Class 0 and Class 1) are about equally present in the Validation Dataset, there is a clear overpowering of Class 1 in the Training dataset, having about three times the amount of nodes of Class 0 (Figure 3a,3b).
This can help explain the difference in accuracy between training and validation steps.

The distribution of features is about the same between the two datasets (Figure 3c,3d). Both classes have similar feature dimensions, with most nodes having both first and second feature dimensions centred around 0, forming a clear spherical cluster within the [-0.4,+0.4] range in both dimensions.
The Validation dataset is noticeably smaller in size compared to the Training, thus some sparsity is introduced in the feature distribution; however, the overall scattering of the feature dimensions is preserved.
Finally, due to the class imbalance issue between the two datasets, there is a visible difference in density of features between the two datasets, with Class 1 nodes of the Training dataset being much more densely present in the aforementioned central cluster.




(a) Training and Validation F1 Score per epoch with Max Aggregation.


(b) Training and Validation F1 Score per epoch with Sum Aggregation.




(c) Training and Validation F1 Score per epoch with Mean Aggregation.

Figure 2: Comparison of Training and Validation F1 scores for all Aggregation Methods

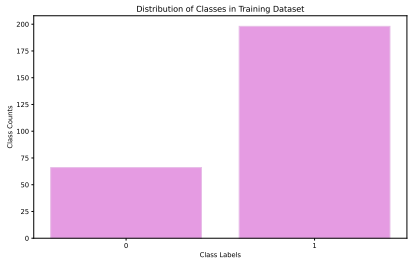
The topologies of both classes are about preserved between the two datasets and so are the topologies of the nodes in the two classes (Figures 3e-3h).
Class 0 nodes resemble a fully connected graph/clique, with most nodes sharing edges.
Class 1 nodes resemble a more modular graph, with both datasets showing three main clusters (say A,B,C), with only one central cluster (B) connected to each of the two other clusters (i.e. A connected to B, C connected to B, A and C not connected). Given the presence of highly important nodes determining the flow of information between the different clusters, these graphs may lead to over-squashing, as the links these nodes create may form a bottleneck and lead to information loss.

## 1.3 Overcoming Dataset Challenges

### 1.3.a Adapting the GCN

A new class '**MyGraphNeuralNetwork2**' and a new class '**MyGCNLayer2**' were created, modelled on the basis of the original 'MyGraphNeuralNetwork' and 'MyGCNLayer' classes. '*MyGraphNeuralNetwork2*' represents the adapted GCN, with two regular GCN layers followed by a **Classification Head**, a Linear Layer that maps the D-dimensional final graph embedding learned by the GCN layers to a 1-dimensional output for classification. The network is defined by 5 parameters:

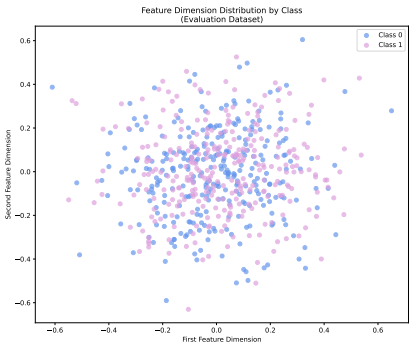
- **Number of Layers** (L) to allow the model to dynamically create L GCN layers using a for loop and the torch *ModuleList()* class.



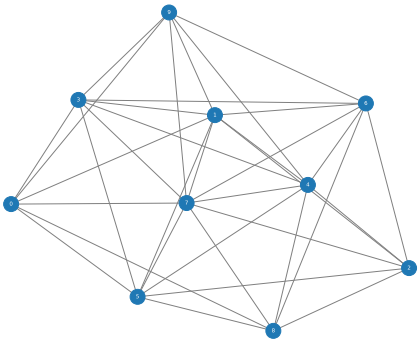

(a) Distribution of classes in the Training Dataset


(b) Distribution of classes in the Validation Dataset

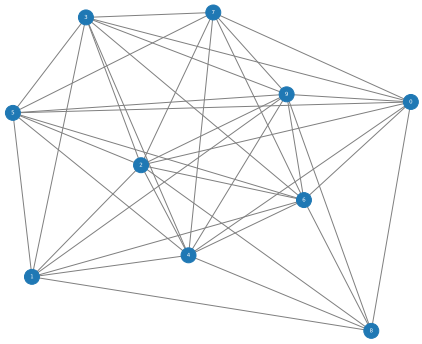

(c) Feature Dimensions distribution per class in the Training Dataset.



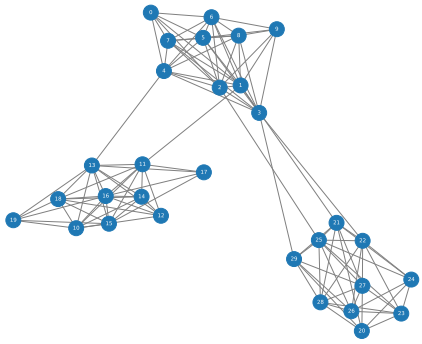

(d) Feature Dimensions distribution per class in the Training Dataset.



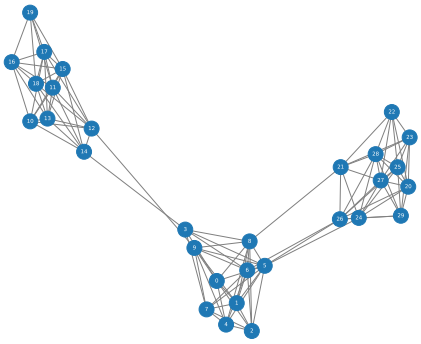
(e) Topology of class 0 nodes of Training Dataset.



(f) Topology of class 0 nodes of Validation Dataset.



(g) Topology of class 1 nodes of Training Dataset.



(h) Topology of class 1 nodes of Validation Dataset.

Figure 3: Topological Analysis of Tranining and Validation Datasets

- **Input dimension** (D=10) of the graph node features. This is used as input dimension for the first GCN layer.
- Desired **hidden dimension** (H) of the GCN layers, used as output dimension of the first layer and as input-output dimension of the following hidden layers.
- Desired **output dimension** (D') of the last GCN layer (i.e. dimension of the graph embeddings), used as output dimension of the last GCN layer and as input dimension for the classification head.
- **Number of Classes** which defines the size of the output of the classification head. By default it is set to 1, given the binary scope of the classification.

For an input graph of size (NxD) (i.e. N nodes with D features each) and model with hidden dimension H and L GCN layers, the model will:

1. (NxD $\rightarrow$ NxH) Pass all N nodes through the first GCN layer to get N nodes with feature size H.
2. (NxH $\rightarrow$ NxH) Pass the embeddings through L-2 GCN layers which will keep the dimensionality unchanged.
3. (NxH $\rightarrow$ NxD') Pass the embeddings through the last GCN layer which will output features of size (NxD').
4. (NxD' $\rightarrow$ 1xD') Perform global aggregation on all the graph nodes.
5. (1xD' $\rightarrow$ 1) Pass the aggregated embedding through the Classification Head to get a 1-D output for binary graph classification.

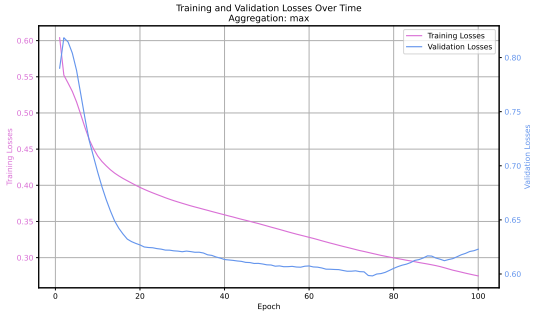
The class performs a chosen **global aggregation** following the methods described in Q1.1.a. during the forward pass, on the embeddings of the last GCN layer and before the classification head is applied. The model optionally returns ALL the N embeddings of every GCN layer as a list.
Full solutions can be found in the Q1 **.ipynb** file.

### 1.3.b Improving the Model

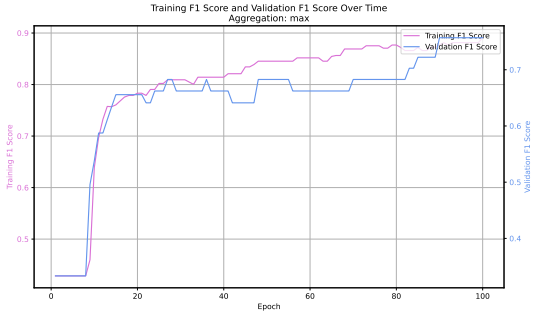
Performance Results of the Improved Model are shown in Figure 4, with the final model reaching **100%** Accuracy on both the Training and Validation Datasets (Figure 4c,4d). Improvement strategies are detailed below in Section **1.3.d 'Final Analysis and Explanation'** and results of experimentations are shown below in Figure 6.
Full implementations and explorations can be found in the Q1 **.ipynb** file.

### 1.3.c Evaluating the Best Model

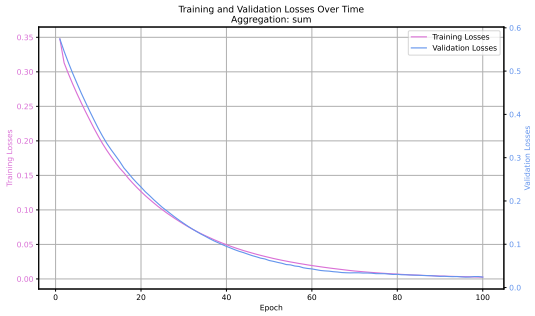
Performance of the best model is shown below in Figure 5, smoothed over 10 different training (and evaluation) runs of 100 epochs each.
The smoothed average was obtained by collecting all the losses and accuracies per epoch for each run in separate lists, and using the *np.mean(..., axis=0)* function on each list. Full implementation can be found in the Q1 **.ipynb** file. Full and detailed implementations can be found in the Q1 **.ipynb** file.



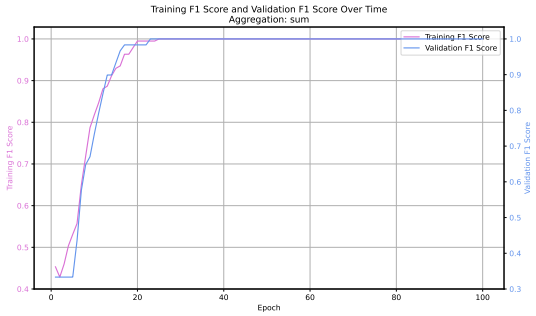

(a) Training and Validation Loss per epoch with Max Aggregation.



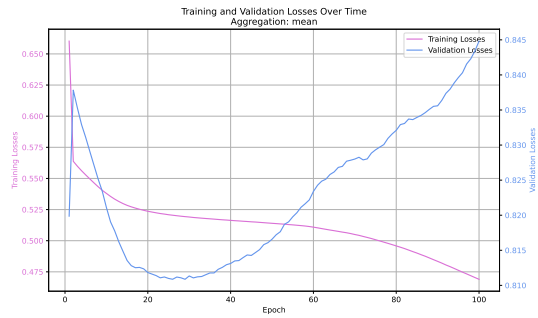

(b) Training and Validation F1 Score per epoch with Max Aggregation.




(c) Training and Validation Loss per epoch with Sum Aggregation.



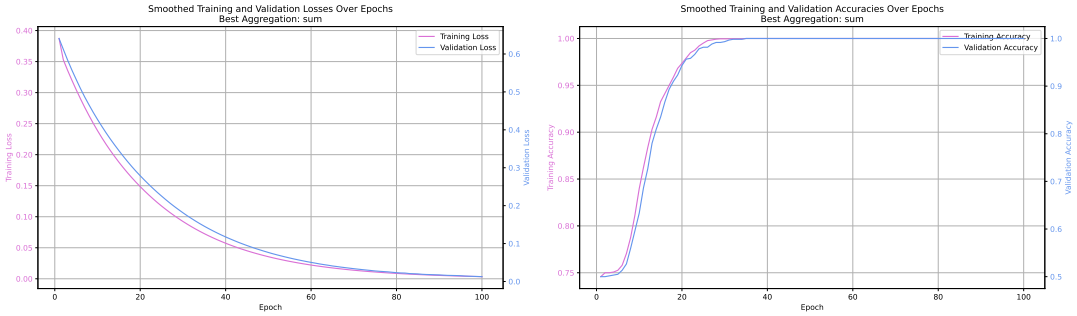

(d) Training and Validation F1 Score per epoch with Sum Aggregation.




(e) Training and Validation Loss per epoch with Mean Aggregation.


(f) Training and Validation F1 Score per epoch with Mean Aggregation.

Figure 4: Comparison of Training and Validation Losses and F1 scores for all Aggregation Methods for the Improved Model




(a) Smoothed Training and Validation Loss per epoch of Best Model.

(b) Smoothed Training and Validation Accuracy per epoch of Best Model.

Figure 5: Smoothed Training and Validation Losses and Accuracy for the Best Model across 10 runs

### 1.3.d Final Analysis and Explanation

The following changes were made to improve model performance:

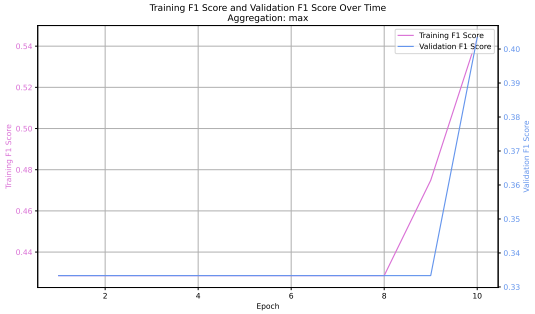
- **Model Architecture**:
  - Reduced dimension of hidden embeddings from 8 to **5**.
  - Increased the number of GCN layers from 2 to **3** to allow for various abstraction levels and to let the network learn more nuanced embeddings.
  - Changed the graph embedding dimension from 8 to **5**.
- **Epoch Number**: Increased the number of epochs from 10 to **100**.
- **Learning Rate**: Increased the learning rate from 0.001 to **0.0015**.
- **Global Aggregation**: Set the global aggregation method to **sum**, after training and evaluating the model on all methods (Figure 4).

I experimented with a series of hyperparameter combinations by changing the size of the output and hidden dimensions, number of hidden layers, learning rate and aggregation methods, and found that:

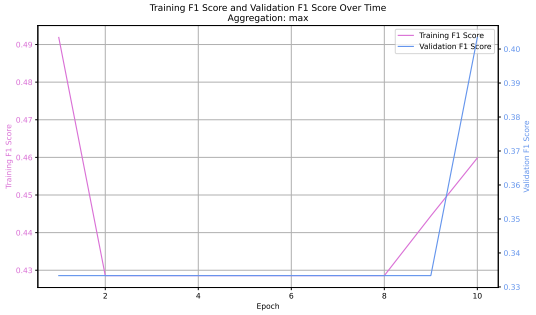
- any learning rate less then *0.005* gave similar optimal performances, and settled on *0.0015* as it gave the smoothest and quickest loss decrease and accuracy peak.
- Mean aggregation gave the worst performer by far on any hyperparameter combination.
- Max and Sum aggregations gave very similar model performances on the same hyperparameters, however Sum aggregation had overall higher accuracies more consistently.
- More layers slowly led to worse performances (probably due to *oversmoothing*), and similarly so did very high hidden embedding sizes (¿15).
- I experimented with different Loss Functions (*Cross Entropy* and *L1 Loss*), but found the best to undoubtedly be the original **Binary Cross Entropy** Loss.

- I explored dataset **normalization**, but that led poorer model performance when compared to the original dataset, and thus discarded this modification.

Results of experimentations are shown in Figure 6.

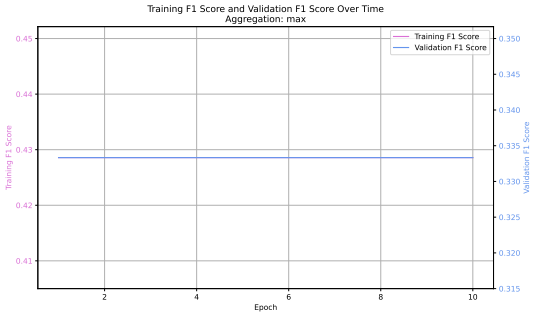



(a) Combo 1: Hidden Dim: 8, Layers: 2, Output Dim: 8, Aggregation: max, LR: 0.001

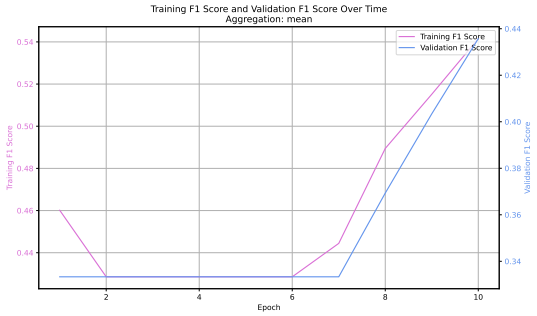



(b) Combo 2: Hidden Dim: 8, Layers: 2, Output Dim: 10, Aggregation: max, LR: 0.001


(c) Combo 3: Hidden Dim: 5, Layers: 3, Output Dim: 5, Aggregation: sum, LR: 0.005




(d) Combo 4: Hidden Dim: 3, Layers: 5, Output Dim: 2, Aggregation: max, LR: 0.0015




(e) Combo 5: Hidden Dim: 8, Layers: 2, Output Dim: 2, Aggregation: mean, LR: 0.01

Figure 6: Comparison of Training and Validation Accuracies for various hyperparameter combinations of the Improved Model

# 2 Node Classification in a Heterogeneous Graph

## 2.1 Dataset

### 2.1.a Problem Challenge

Because the feature dimensions of the nodes differ from each other, it is harder to understand what the determining trait of each node class is: if nodes A and B with different feature dimensions are of the same class, there must be some complex pattern and relationship between features of A and of B that is harder to find compared to nodes of same-sized features.
Standard node classification relies on gathering information on each set of nodes from the same class to understand what traits they share, and this is not easy when comparing nodes of different natures.

### 2.1.b Real-World Analogy

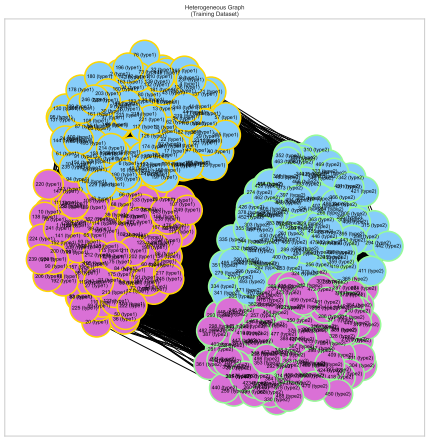
Take as an example a **Social Status Graph**, where each node represents an individual in a Society, and can be of two types based on their Social Status: **R** (Royalty) or **C** (Commoner). The graph shows the relationships between Royal People (Nodes of Class R) and Common People (Nodes of class C) within a (probably ancient) Society.
Now let's assume that each node can also be of different natures, for example based on the individual's biological sex, and have nodes of type **W** (woman), with N features, or **M** (man), with K features.
This scenario creates a dataset similar to the structure of the given one: each node can be of two types ('Type 1' or 'Type 2', like the nodes of nature 'W' and 'M' in the given scenario), and cal be of one of two possible classes, Class 0 and Class 1 (like the two classes 'R' and 'C' in the scenario).
It is important here to understand relationships between nodes of different natures and classes in order to understand the underlying patterns that shape such Society.
For example, it can be important to understand the relationship between Royalty and Sex, investigating if nodes of either nature are less or more likely to be of Royal Status rather than Commoner or vice versa, or to understand how and if each Class relates to the other, or if they tend to isolate from each other. Relationships between such heterogeneous nodes are crucial to uncover complex social dynamics and presence of prejudices (like classism or sexism).

### 2.1.c Interpretation of the Dataset: Plotting the Graph

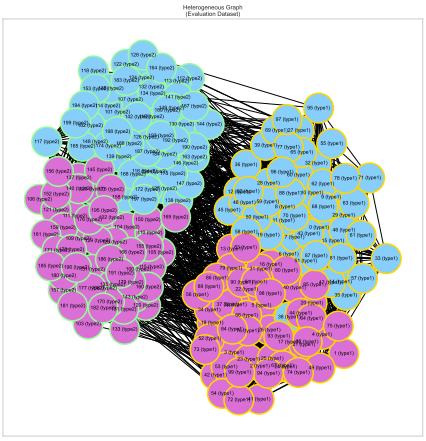
Figure 7 shows the topology of both the Training (Figure 7a,7c) and Validation (Figure 7b,7d) datasets. Graph nodes are identified **class** (0 or 1) and **type** (type 1 or type 2). Node class is indicated by the node colour and node type is indicated by node outline colour:

- pink nodes = class 0
- blue nodes = class 1
- yellow-contoured nodes = type 1
- lime-contoured nodes = type 2

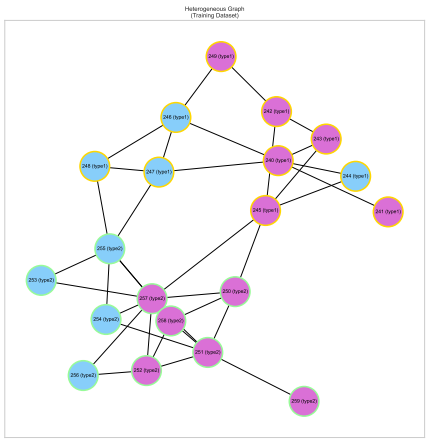
Sub-Graphs (Figure 7c,7d) are shown below in order to visualize edges between heterogeneous nodes.



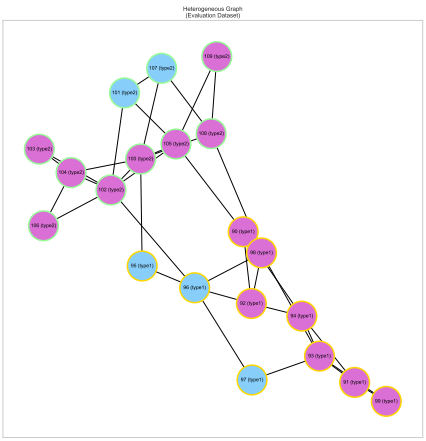
(a) Topology of entire training dataset



(b) Topology of entire Validation dataset



(c) Topology of a subgraph of the training dataset



(d) Topology of subgraph of the Validation dataset

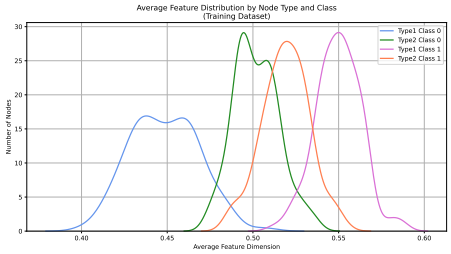
Figure 7: Topology of The Training and Validation Graphs by Node type and class.

Full and detailed implementations can be found in the Q2 **.ipynb** file.

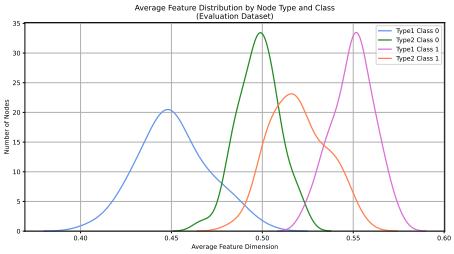
Both datasets retain similar topology, where the nodes are **clustered** by *node type*, with a clear spatial separation between nodes of type 1 (clustered on the top-left) and of type 2 (clustered on the bottom-right). Furthermore, the nodes are also clustered by *node class*, with, again, a clear spatial separation between nodes of class 0 (clustered on the bottom-left) and of class 1 (clustered on the top-right).
Thus there is no mixing of nodes of different type and class, and they tend to stick together to their most similar nodes.
However, there are edges between nodes of different natures (as highlighted by the subgraphs), and thus the nodes are not fully isolated within their own cluster.

### 2.1.d Interpretation of the Dataset: Plotting the Node Feature Distributions

Figure 8 shows the distribution of average values of node features based on the nodes' type and class, for both datasets.




(a) Distribution of node features in Training dataset




(b) Distribution of node features in Validation dataset

Figure 8: Distribution of Mean Node Feature Values by node type and class for both Training and Validation Datasets.

Full and detailed implementations can be found in the Q2 **.ipynb** file.

### 2.1.e Interpretation of the Dataset: Discussion

The feature values are distributed almost identically between the two datasets for the same node types and classes.
However, the values are quite different when comparing nodes of different natures:

- all node classes and types have a clear normal distribution of class values, with different means and (somewhat) similar variances
- type 2 nodes of either class tend to have tighter values in the range [-0.47,+0.57]
- type 1 nodes of either class tend to have more extreme values, respectively in the lower values (below 0.5 for class 0 nodes) and in the higher values (above 0.5 for class 1 nodes)
- nodes of class 0 tend to have feature values below 0.5, while class 1 nodes have feature values above 0.5

## 2.2 Naive Solution: Padding

### 2.2.a Limitations of Naive Solution

The use of padding introduces the following key limitations:

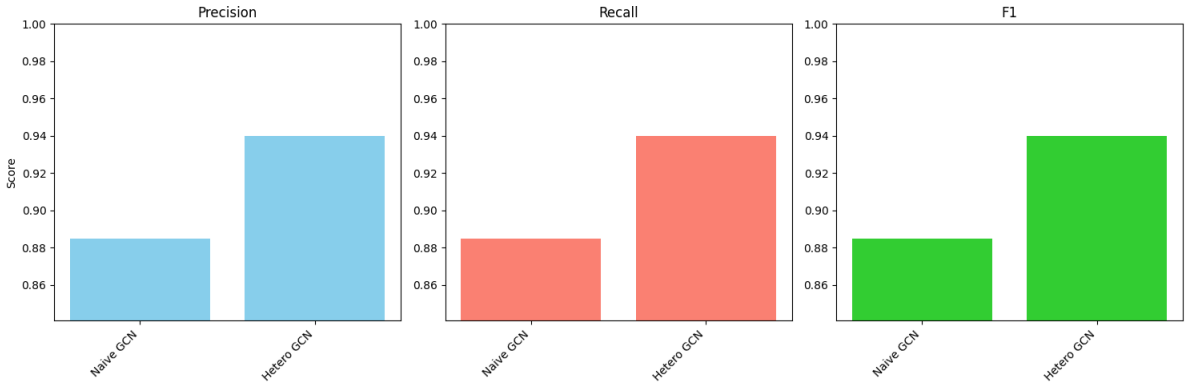
- **Loss of Feature Meaning**: the presence alone of each feature for a given node type may have a specific meaning that crucially characterizes that particular node type, and so is the absence of such feature. Adding the extra features with a value of 0 to the smaller-sized nodes may change and distort the intrinsic nature of the node, leading to confusion (and lower accuracy) in prediction and classification.

- **Computational and Memory Inefficiency**: adding extra values for smaller-sized inputs leads to an increase in computational overhead, especially when the difference between the feature dimensions becomes large, especially due to the numerous expensive matrix multiplication operations that characterize GCNs. Furthermore, this extra computation (and extra memory allocation) is 'useless', as it doesn't carry any information about the node that is being analysed (and may lead to slower loss convergence and training).

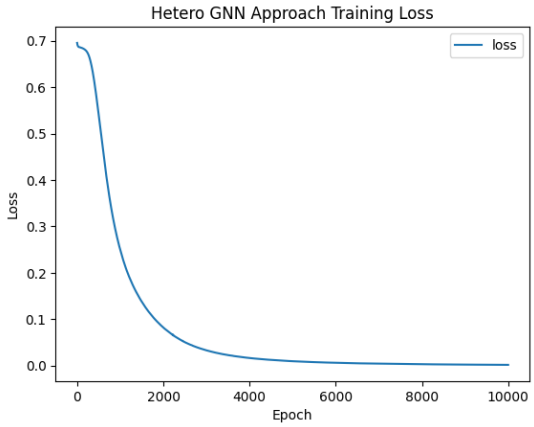
## 2.3 Node-Type Aware GCN

### 2.3.a Implementation

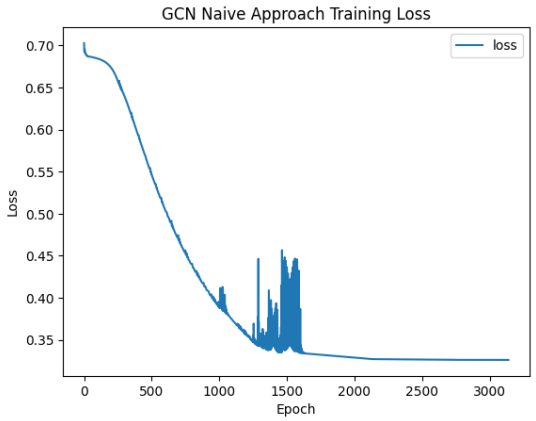
INSERT YOUR ANSWER HERE




(a) Comparison of F1 Scores of Naive GCN and HeteroGCN




(b) Implemented Node-Aware GCN Training Loss




(c) GCN with padding Training Loss

Figure 9: Training Loss and F1 Score comparison of Vanilla GCN with Padding and Implemented Node-Aware Heterogeneous GCN.

### 2.3.b Discussion

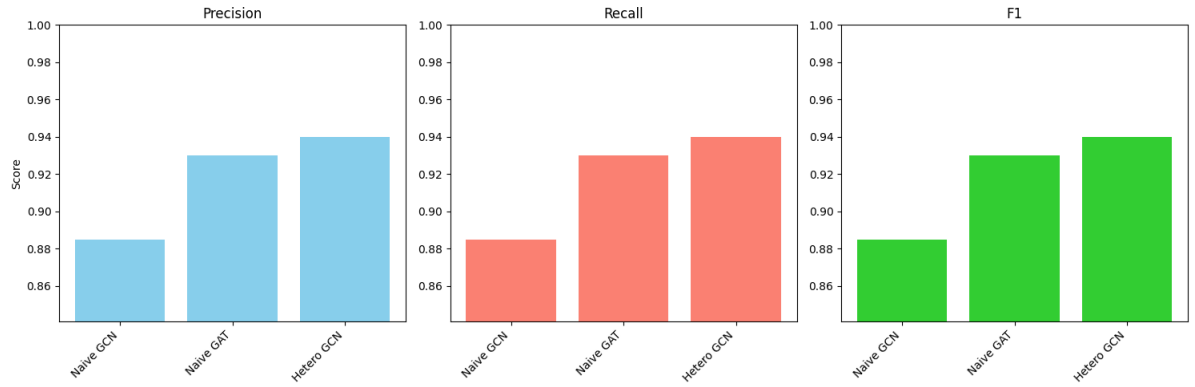
*In your report, you are expected to: • Explain your design choices, including: – The reasoning and logic behind your solution. (1 point) – The limitations of the naive solution that your model addresses. (1 point) – The advantages and potential drawbacks of your*

*approach. (2 points) • Include the loss curves for both the naive approach and your model. These should be automatically generated using the plot loss function provided to you. (2 points) • Describe your hyperparameter tuning process, including the methodology and reasoning behind your choices. (2 points)*

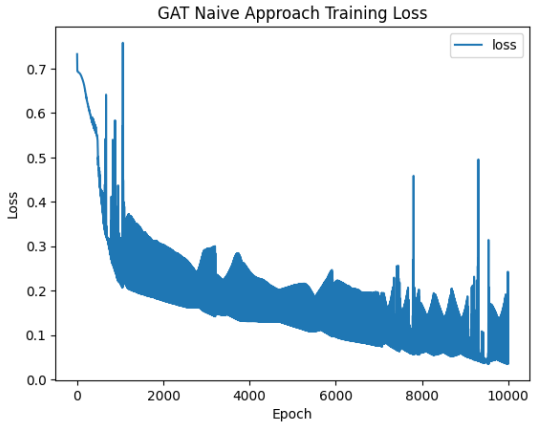
## 2.4 Exploring Attention

### 2.4.a Implementation

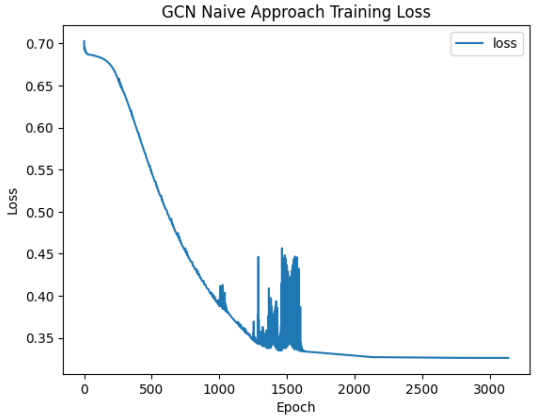
INSERT YOUR ANSWER HERE




(a) Comparison of F1 Scores of Naive GCN, HeteroGCN and Attention-based GCN




(b) Implemented Attention-Based GCN Training Loss




(c) GCN with padding Training Loss

Figure 10: Training Loss and F1 Score comparison of Vanilla GCN with Padding and the two Implemented Node-Aware Heterogeneous GCN and GCN with Attention-based aggregation.

### 2.4.b Discussion

INSERT YOUR ANSWER HERE

## 2.5 Overall Discussion

INSERT YOUR ANSWER HERE

# 3 Investigating Topology in Node-Based Classification Using GNNs

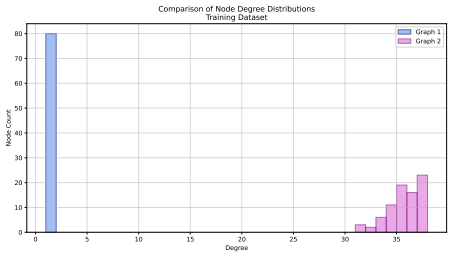
## 3.1 Analyzing the Graphs

### 3.1.a Topological and Geometric Measures

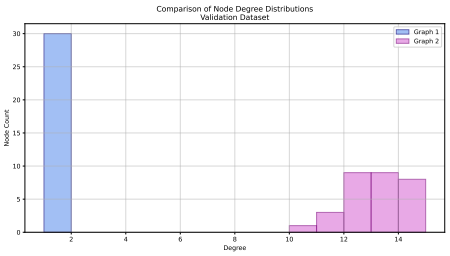
INSERT YOUR ANSWER HERE

### 3.1.b Visualizing and Comparing Topological and Geometric Measures of Two Graphs

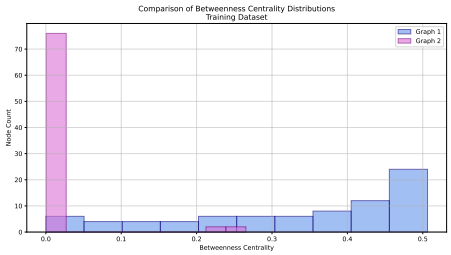
INSERT YOUR ANSWER HERE



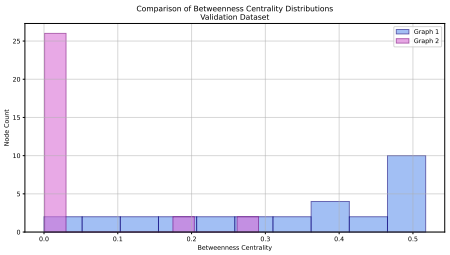

(a) Node Degree Distribution of Training Dataset



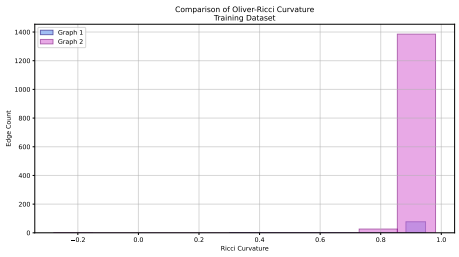

(b) Node Degree Distribution of Validation Dataset



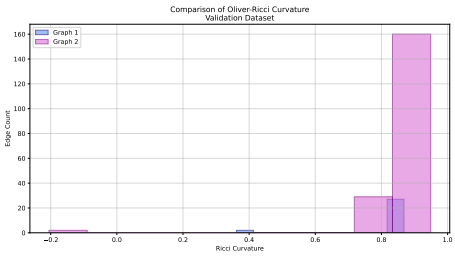

(c) Betweenness Centrality Distribution of Training Dataset




(d) Betweenness Centrality Distribution of Validation Dataset




(e) Ricci Curvature Distribution of Training Dataset

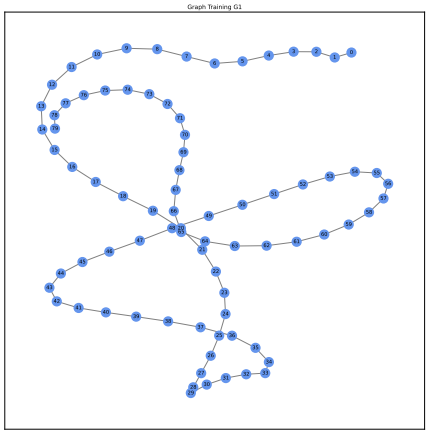



(f) Ricci Curvature Distribution of Validation Dataset

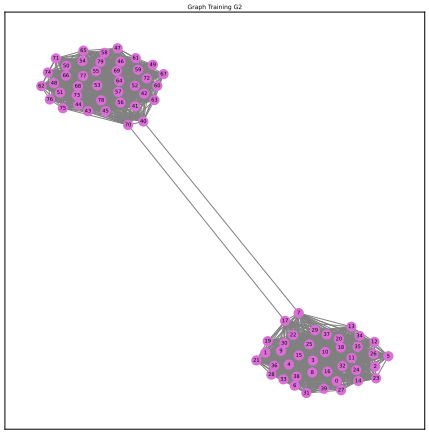
Figure 11: Comparison of Topological Measures of the Training and Validation Graphs.

### 3.1.c Visualizing the Graphs

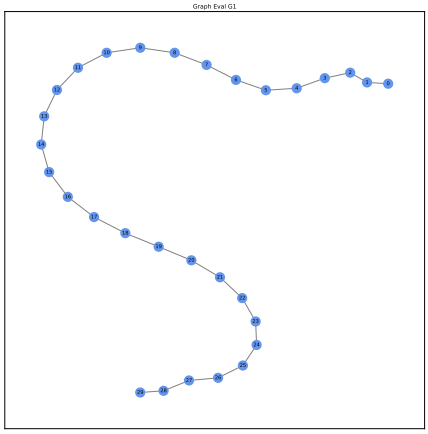
INSERT YOUR ANSWER HERE



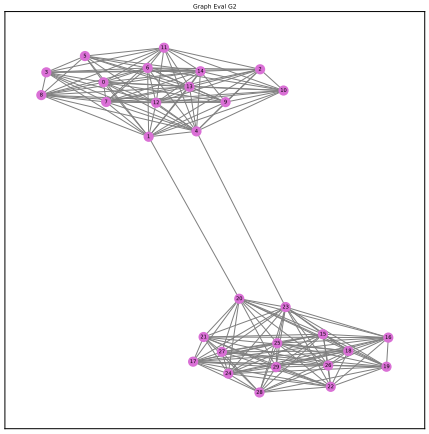

(a) Type 1 Graph of Training Dataset




(b) Type 2 Graph of Training Dataset




(c) Type 1 Graph of Training Dataset




(d) Type 2 Graph of Validation Dataset

#### 3.1.d Visualizing Node Feature Distributions

INSERT YOUR ANSWER HERE

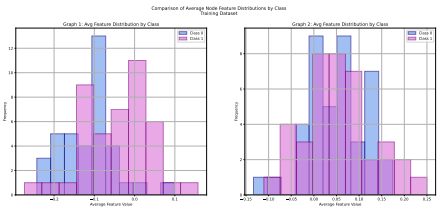
## 3.2 Evaluating GCN Performance on Different Graph Structures

#### 3.2.a Implementation of Layered GCN

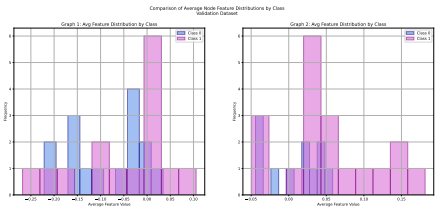
INSERT YOUR ANSWER HERE

#### 3.2.b Plotting of t-SNE Embeddings

INSERT YOUR ANSWER HERE



(a) Feature Distribution of Training Dataset



(b) Feature Distribution of Validation Dataset

#### 3.2.c Training the Model on Merged Graphs $G_1 \cup G_2$

INSERT YOUR ANSWER HERE

#### 3.2.d Joined vs. Independent Training

INSERT YOUR ANSWER HERE

### 3.3 Topological Changes to Improve Training

#### 3.3.a Plot the Ricci Curvature for Each Edge

INSERT YOUR ANSWER HERE

#### 3.3.b Investigate Extreme Case Topologies

INSERT YOUR ANSWER HERE

#### 3.3.c Improving Graph Topology for Better Learning

INSERT YOUR ANSWER HERE

## 4 References