DGL 2025 Coursework 1

Yaqi Zha

CID: 06013006 `yz7724@ic.ac.uk`

Department of Computing
Imperial College London

February 20, 2025

Abstract

This coursework explores the application of Graph Neural Networks (GNNs) to graph-based machine learning tasks, focusing on graph classification, node classification, and the influence of graph topology on model performance.

1 Graph Classification

1.1 Graph-Level Aggregation and Training

1.1.a Graph-Level GCN

The implementation for Q1.1.a is provided in the notebook.
Evaluation Metrics Output (Example Run):

```
# Print out the evaluation metrics
Accuracy: 0.6000
Precision: 0.6190
Recall: 0.6000
F1-score: 0.5833
```

1.1.b Graph-Level Training

The implementation for Q1.1.b is provided in the notebook.The training process records the training loss, training accuracy, and validation accuracy for each epoch. Here is the plot of test F1-scores for all three aggregation methods (Example Run).

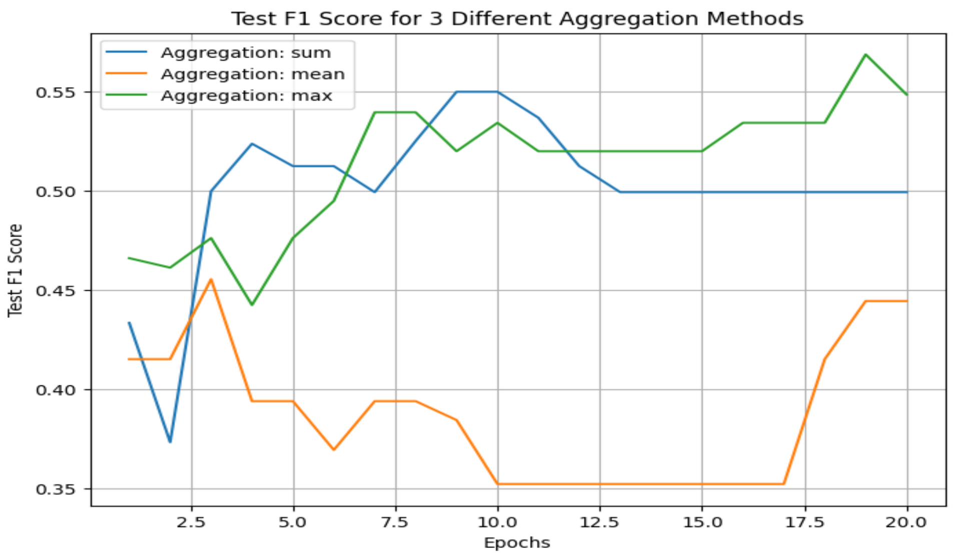


Figure 1: Test F1 Score for 3 Different Aggregation Methods

1.1.c Training vs. Evaluation F1

The implementation for Q1.1.c is provided in the notebook.
Comparison of Training F1 Scores vs. Evaluation F1 Scores:

- **Sum Aggregation:** Training F1-scores improve significantly over epochs, reaching approximately 0.72, but evaluation F1-scores remain around 0.50, showing a large gap. This indicates **overfitting**, where the model performs well on training data but hard to generalize to unseen data.

- **Mean Aggregation:** Training F1-scores improve gradually, peaking at around 0.63, while evaluation F1-scores remain low, fluctuating between 0.35 and 0.45. The consistent gap suggests **underfitting**, as mean aggregation averages features, potentially losing important information.
- **Max Aggregation:** Training F1-scores steadily improve to around 0.53, closely matched by evaluation F1-scores stabilizing between 0.47 and 0.56. The smaller gap shows **better generalization**, as max aggregation captures dominant features, reducing noise.

Which Aggregation Method Performs Best and Why?
The best aggregation method is **max aggregation** because it focuses on the most dominant features from neighboring nodes, effectively prioritizing critical information while reducing noise. This approach results in better generalization, as indicated by its smaller gap between training and evaluation F1-scores compared to sum and mean aggregation.

In comparison, **sum aggregation** amplifies all node features without normalization, leading to overfitting, while **mean aggregation** averages features, which can weaken the impact of key signals and cause underfitting. Max aggregation strikes the right balance, achieving the best evaluation F1-scores.

Here is the plot of Training vs Evaluation F1 Score for all three Aggregation Methods (Example Run).

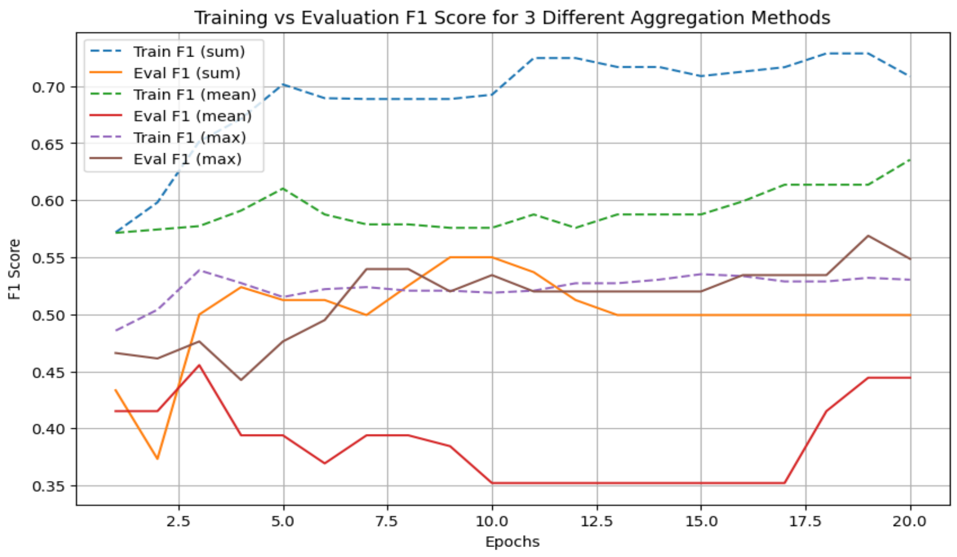


Figure 2: Training vs Evaluation F1 Score for 3 Different Aggregation Methods

1.2 Analyzing the Dataset

1.2.a Plotting

The implementation for Q1.2.a is provided in the notebook. The outputs include the following (Example Run):

Training Data Visualizations

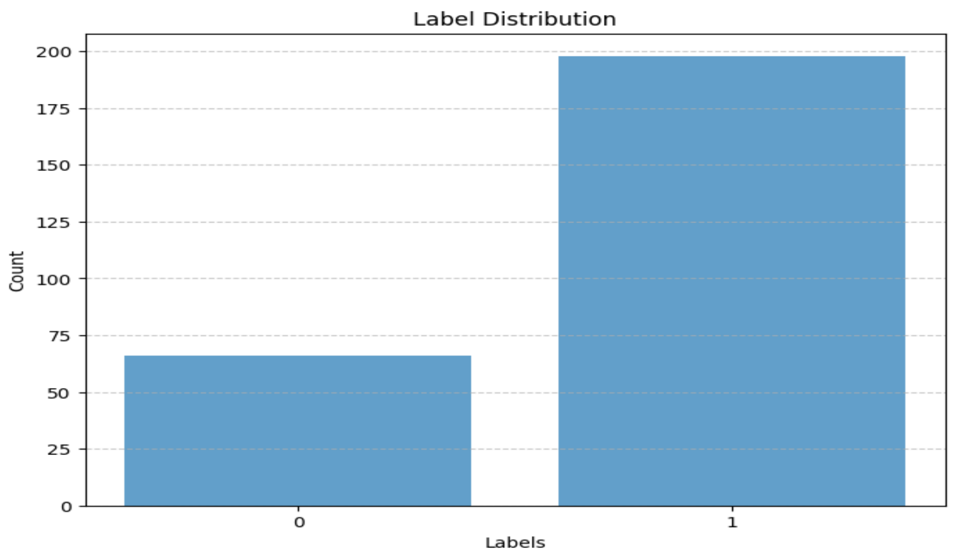


Figure 3: Class distribution of the training dataset

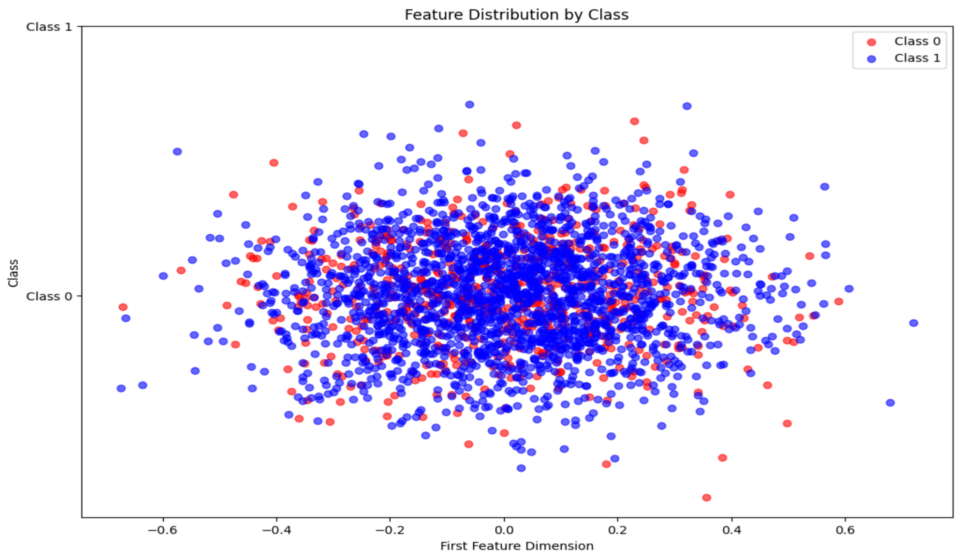


Figure 4: Feature distributions for Class 0 (red) and Class 1 (blue) in the training dataset

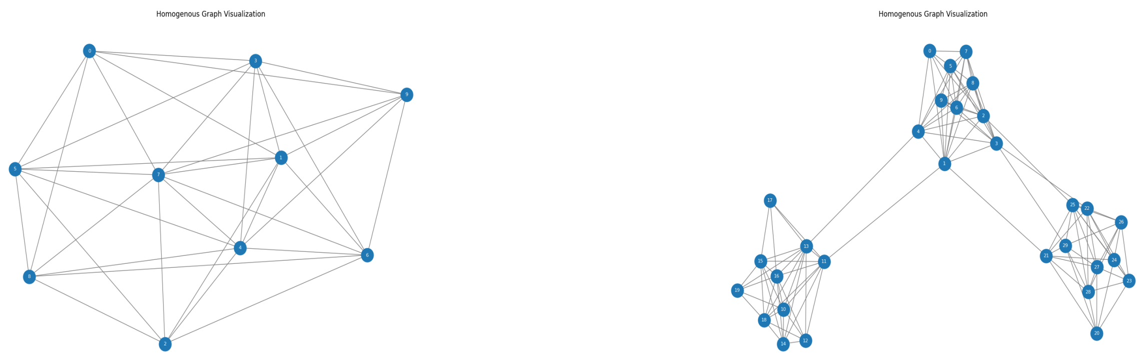

Homogenous Graph Visualization
Homogenous Graph Visualization

Figure 5: Graph Topologies of Class 0 (left) and Class 1 (right) from the Training dataset

Evaluation Data Visualizations

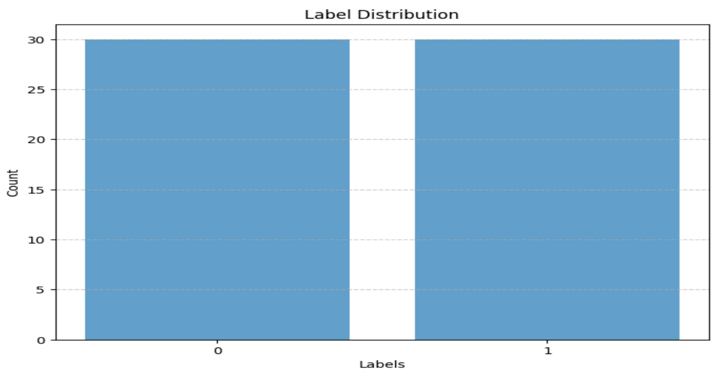


Figure 6: Class distribution of the evaluation dataset.

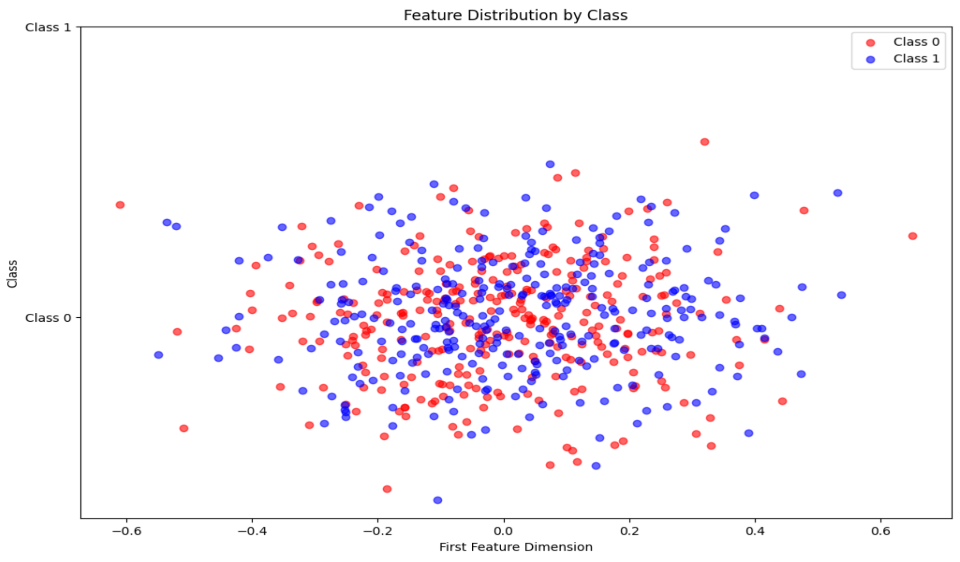


Figure 7: Feature distributions for Class 0 (red) and Class 1 (blue) in the evaluation dataset

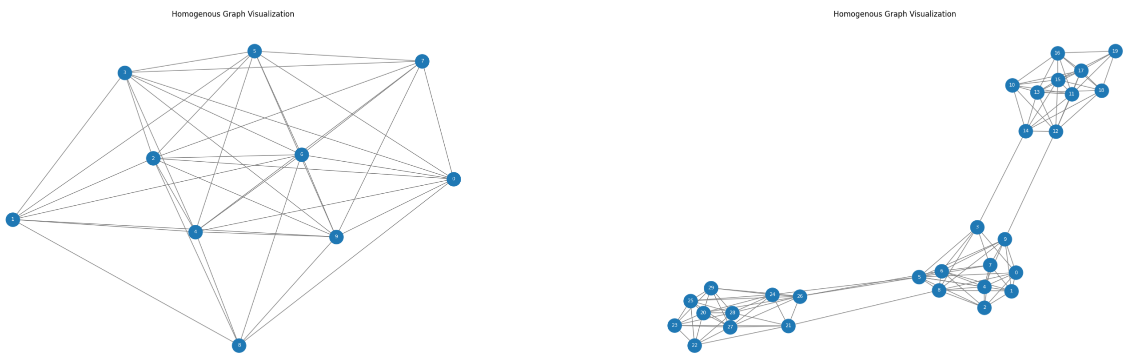

Homogenous Graph Visualization
Homogenous Graph Visualization

Figure 8: Graph Topologies of Class 0 (left) and Class 1 (right) from the Evaluation dataset

1.2.b Discussion

The **graph topologies** reveal noticeable structural differences between the two classes. In the training dataset, the graph for **Class 0** appears relatively **dense**, indicating a highly centralized structure where all nodes are neighbors to each other. In contrast, graphs for **Class 1** demonstrate a **modular structure** with distinct clusters of nodes. These clusters are tightly interconnected internally but only sparsely connected through a few edges. Similar patterns are observed in the evaluation dataset, where Class 0 graphs remain dense, while Class 1 graphs retain their modular structure.

In both dataset, the **feature distributions** for Class 0 (red) and Class 1 (blue) show a significant degree of **overlap** in the scatter plots. This overlap indicates that the two classes are not easily separable in the feature space. Without clear boundaries between the classes, the classification task becomes inherently more challenging.

The training dataset presents an **imbalanced label distribution**, with significantly more samples for Class 1 compared to Class 0. This imbalance could lead to biased predictions during evaluation, where the model may disproportionately favor Class 1 due to its higher representation. However, the evaluation dataset shows a **balanced label distribution**, with an equal number of samples for Class 0 and Class 1. This balance ensures that the model is trained fairly across both classes, without favoring one over the other.

1.3 Overcoming Dataset Challenges

1.3.a Adapting the GCN

The implementation for Q1.3.a is provided in the notebook.

The hyperparameter tuning experiment evaluated the performance of the GCN by varying the hidden dimensions (`hidden_dim`) and the number of layers (`num_layers`). Validation accuracy was tracked over 20 epochs, using the `mean` aggregation method to assess the model's generalization performance.

Increasing hidden dimensions generally improved the model's ability to capture complex patterns. For `HidDim = 8`, validation accuracy stabilized between 0.70–0.75, but this configuration did not achieve the highest scores. In contrast, `HidDim = 16` consistently outperformed smaller dimensions, reaching a peak validation accuracy of 0.775 with 3 layers. While `HidDim = 32` performed similarly to `HidDim = 16`, it exhibited greater variability, particularly with 4 layers, likely due to overfitting caused by increased model complexity.

The number of layers also had a significant impact. Models with `2 layers` performed consistently but tended to underfit the data, as they lacked sufficient capacity to learn complex relationships. Increasing to `3 layers` offered the best balance, with stable and high accuracy across most hidden dimensions. However, models with `4 layers` showed inconsistent trends, especially for larger hidden dimensions, again suggesting overfitting.

Here is the Plot (Example Run):

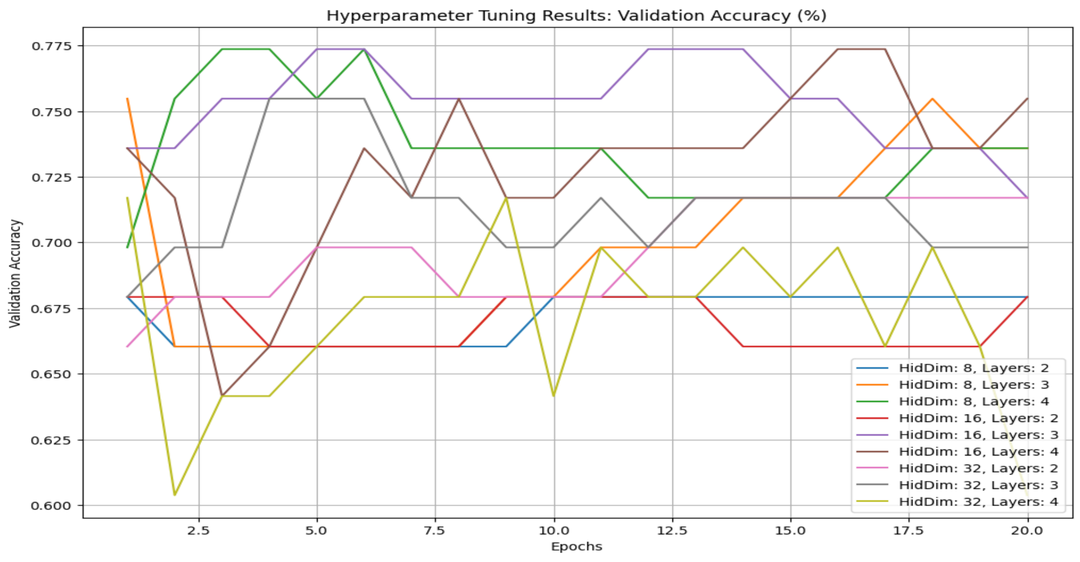


Figure 9: Validation accuracy trends for different combinations of `HidDim` and `Layers`.

1.3.b Improving the Model

The implementation for Q1.3.b is provided in the notebook. The final test accuracy is 0.9333 (Example Run).

1.3.c Evaluating the Best Model

The implementation for Q1.3.c is provided in the notebook. Here is the Plot (Example Run):

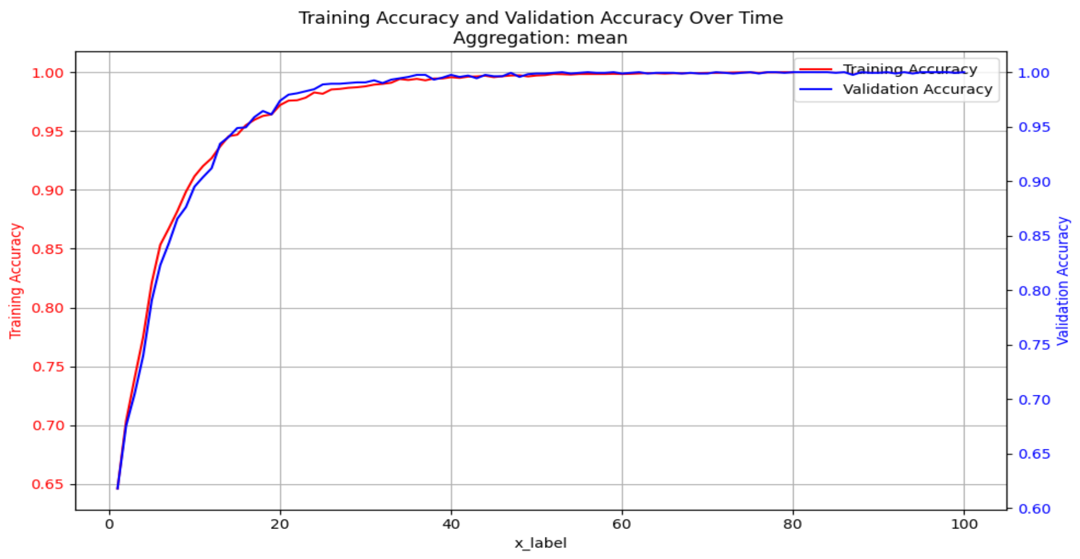


Figure 10: Training and Validation Accuracy on best model

1.3.d Final Analysis and Explanation

I made several modifications to the model, loss function, and dataset to improve graph classification performance. The original model, which consisted of two **Graph Convolutional Network (GCN) layers**, was extended to include a configurable number of layers, with three layers performing best in our experiments. Additionally, **dropout (0.5) regularization** was introduced after each hidden layer to avoid overfitting. The dataset was also refined by **incorporating node degree as an additional feature**, applying **Principal Component Analysis (PCA)** to reduce feature redundancy while retaining 95% variance, and **padding adjacency and feature matrices** to standardize input sizes. Furthermore, to address dataset imbalance, **oversampling of the minority class** was implemented.

Hyperparameter tuning focused on three key aspects: **number of layers, hidden dimension size, and graph aggregation method**. Testing 2, 3, and 4 layers showed that **3 layers provided the best balance** between complexity and generalization. **A hidden dimension of 16 was optimal**, as 8 caused underfitting while 32 increased computational cost without notable performance gains. Among `sum`, `mean`, and `max` aggregation, **`mean` performed best**, likely due to its ability to normalize feature distributions across graphs for more stable learning.

The training and validation accuracy plot, averaged over 20 independent runs, exhibits **a steady learning trajectory with smooth convergence**. Initially, both metrics **rise rapidly**, reflecting effective feature extraction. By epoch 20, accuracy surpasses 95%, and after 40 epochs, it stabilizes near 100%, indicating **the model's high confidence in classification**. A key observation is that there is **a close alignment of training and validation accuracy** throughout suggests minimal overfitting, likely due to dropout regularization (0.5) and dataset balancing through oversampling. Additionally, PCA-driven feature reduction enhanced stability by removing redundancy, improving training efficiency.

2 Node Classification in a Heterogeneous Graph

2.1 Dataset

2.1.a Problem Challenge

This problem is challenging due to the heterogeneous nature of the graph, where **nodes belong to two distinct types, each with different feature dimensions**. Unlike standard node classification that all nodes share a single type with the same feature dimensions, this dataset includes 20 features for t_1 nodes and 30 features for t_2 nodes, requiring the model to handle multiple feature representations efficiently.

Additionally, the graph structure is heterogeneous, meaning **edges connect nodes of different types**. In a standard node classification task, all nodes typically belong to the same type, making message passing uniform. In this task, edges connect different types of nodes, requiring specialized aggregation techniques like Heterogeneous Graph Convolutions.

Another challenge is mapping features correctly from separate matrices. Instead of a single feature matrix, the dataset provides **two distinct feature matrices for t_1 and t_2 nodes**, with a dictionary to track which feature set each node belongs to.

2.1.b Real-World Analogy

A real-world example in a candy shop supply chain.

Node Type t_1: Candy Shops, with 20 features (eg, Location, Monthly Sales Volume, Customer Ratings, Number of Customer Reviews...)

Node Type t_2: Candy Brands, with 30 features (eg, Number of Ingredients, Candy Types, Price, Brand Reputation...)

Edges (Relationships):

- A candy shop sells candies from a candy brand.
- A candy shop is located near another shop.
- A candy brand has partnerships with other brands.

Class c_1: Popular shops or popular brands
Class c_2: Less popular shops or less known brands

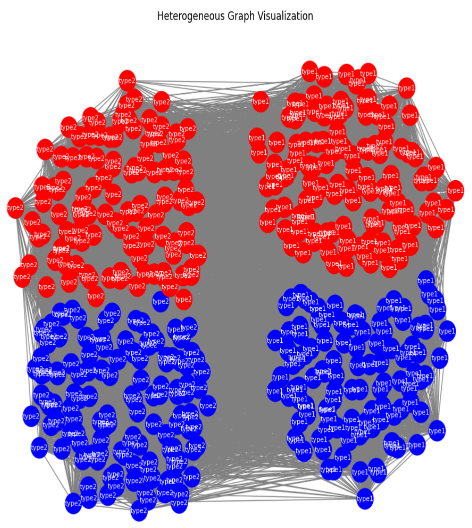
Understanding the relationships between node types and classes is crucial in this task because, unlike standard node classification, nodes belong to different types with distinct feature dimensions. Since nodes of type t_1 and t_2 have different feature spaces, direct feature comparisons are not possible. The model must learn **how to aggregate and transfer information between different node types** while preserving their unique characteristics.

Node classification in a heterogeneous graph relies not only on a node's own features but also on the structure of the graph and its neighbors' attributes. A node's class can be influenced by its connections, especially when interacting with nodes of another type. For example, if a particular t_2 node is heavily connected to t_1 nodes of a specific class, it is likely to share similar properties. **Ignoring these cross-type dependencies can lead to misclassification**.

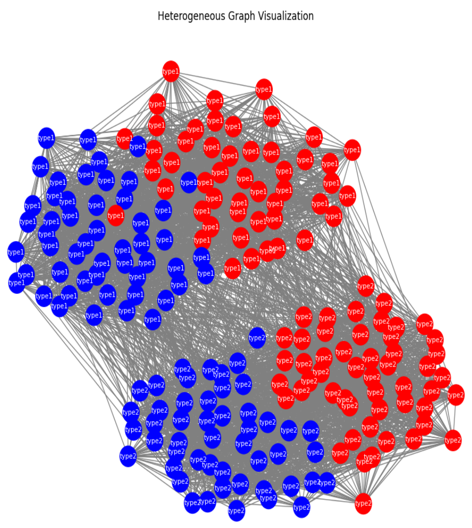
Additionally, **improper message passing can result in information loss**. If the model does not distinguish between different node types and their relationships, it may overlook crucial structural patterns. Differentiating between intra-type and inter-type connections allows the model to make more meaningful predictions.

2.1.c Interpretation of the Dataset: Plotting the Graph

The implementation for Q2.1.c is provided in the notebook.
Here is the plots:



(a) Visualization of the Training Graph



(b) Visualization of the Validation Graph

Figure 11: Visualization of the Training Graph and the Validation Graph

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

The implementation for Q2.1.d is provided in the notebook.
Here is the plot:

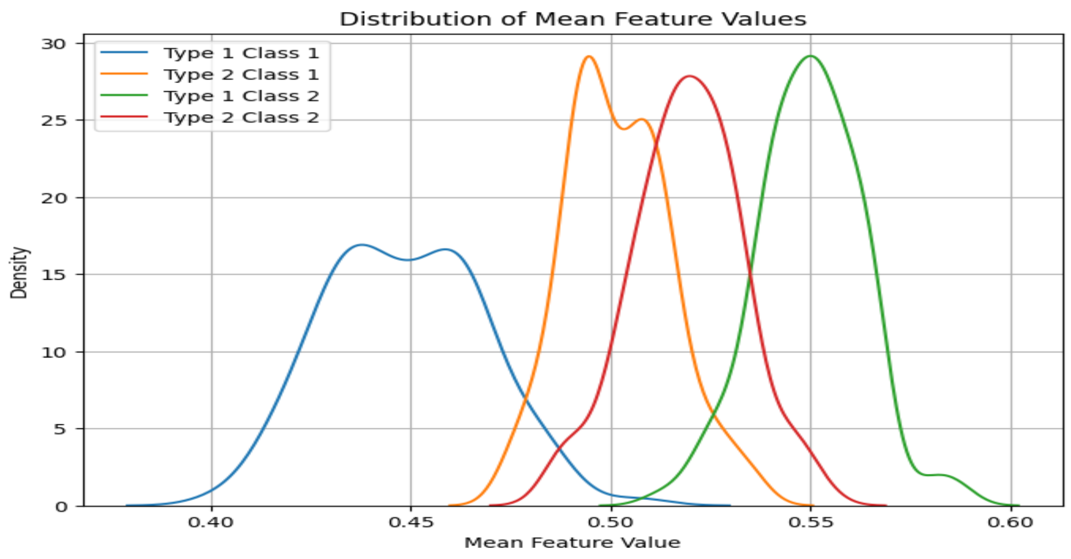


Figure 12: Distribution of Mean Feature Values by Node Type and Class

2.1.e Interpretation of the Dataset: Discussion

The node feature distributions reveal distinct patterns across both node types and classes. **`t1` nodes exhibit a broader spread of feature values**, indicating higher variability, while **`t2` nodes have more tightly clustered distributions**, suggesting greater consistency in their features. This implies that `t1` nodes may represent entities with diverse characteristics, while `t2` nodes are more uniform.

Regarding class differences, **`c1` nodes generally have lower mean feature values**, while **`c2` nodes are shifted toward higher values**. This separation suggests that feature values play a significant role in distinguishing between classes, making them useful for classification. However, overlap between certain groups, particularly within `t2` nodes, may introduce challenges for classification.

2.2 Naive Solution: Padding

2.2.a Limitations of Naive Solution

The naive model fails to differentiate between node types (t_1 and t_2), because of their distinct feature spaces. Treating all nodes as homogeneous through zero-padding ignores structural and semantic differences, making type-specific interactions harder to capture and reducing classification accuracy.

Additionally, **zero-padding increases feature dimensionality inefficiently**, forcing all nodes to adopt a 30-dimensional representation when t_1 nodes only have 20 features. This adds redundant zeros, leading to unnecessary memory usage and higher computational costs, which become significant in large-scale graphs.

Finally, naive approach **treats the adjacency matrix uniformly**, meaning it does not differentiate between intra-type($t_1 \to t_1, t_2 \to t_2$) and inter-type ($t_1 \to t_2, t_2 \to t_1$)

connections. This results in suboptimal message passing, where information is indiscriminately mixed across node types, weakening the model's ability to learn meaningful interactions.

2.3 Node-Type Aware GCN

2.3.a Implementation

The implementation for Q2.3.a is provided in the notebook. The HeteroGCN model achieved an F1-score of 0.9950.

2.3.b Discussion

My main motivation was to explicitly handle the graph's heterogeneity. Since t1 and t2 nodes have different feature dimensions, the model applies distinct transformations to each node type. Unlike a standard GCN, it **separates t1 and t2 nodes**, processing them with **independent linear layers before aggregating information**.

To ensure relevant information flow, **the adjacency matrix A is split into four sub-blocks, allowing type-specific aggregation that prevents feature space mixing**. Additionally, separate normalization stabilizes training by accounting for the broader feature distribution of t1 nodes compared to the more uniform t2 nodes.

Finally, the model uses **type-specific dropout**, applying a higher dropout rate to t1 nodes (more variable) and a lower rate to t2 nodes (more consistent). These adaptations improve robustness by tailoring processing to each node type's unique characteristics.

The limitations of the naive solution that the model addresses:

The HeteroGCN model **explicitly differentiates between node types (t1 and t2) by processing them separately**, **eliminating the need for zero-padding**, which improves feature integrity and memory efficiency. Instead of treating the adjacency matrix as uniform, our model **constructs sub-adjacency matrices** to distinguish between intra-type ($t_1 \to t_1, t_2 \to t_2$) and inter-type ($t_1 \to t_2, t_2 \to t_1$) connections, ensuring type-aware message passing. These enhancements allow the model to better capture node-type-specific interactions, leading to significantly higher classification accuracy and reduced computational overhead compared to the naive approach.

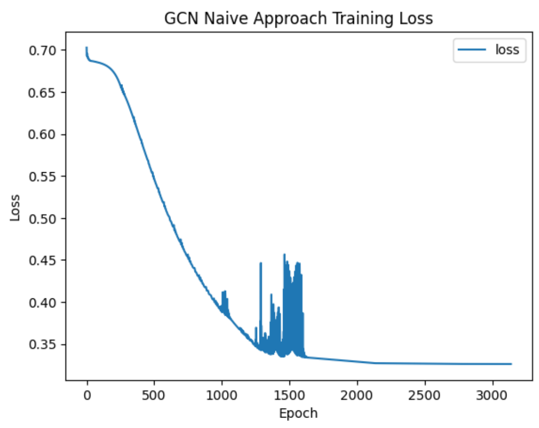
The advantages and potential drawbacks of this approach:

This approach offers several advantages in handling heterogeneous graphs effectively. **Improved feature handling** is a key benefit, as **processing different node types separately** prevents information loss and ensures meaningful feature learning. Additionally, **better message passing** is achieved through **adjacency submatrices**, which respect the heterogeneous graph structure and lead to more accurate node embeddings.

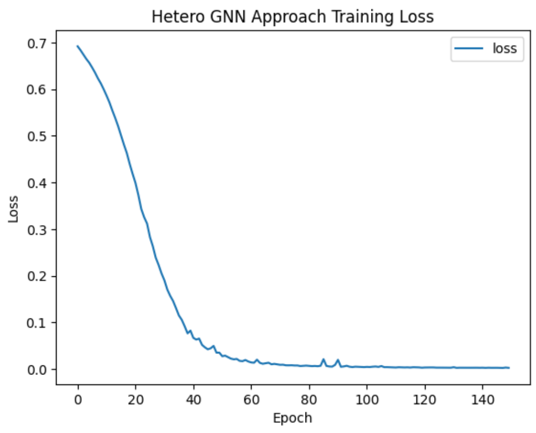
Another major advantage is the **higher generalization ability**. By applying **different dropout rates to different node types**, the model reduces overfitting and improves robustness. This ultimately translates into significantly higher classification performance, with **F1-score of 0.9950**, far outperforming the naive model.

However, there are some potential drawbacks. The approach **introduces increased computational complexity**, as it requires multiple adjacency submatrices and weight matrices, leading to higher memory usage and more matrix operations. Additionally, the **extremely high F1-score suggests the risk of overfitting**, highlighting the need for further validation on unseen data. Lastly, the method demands **additional hyperparameter tuning**, as optimizing different adjacency matrices and dropout rates requires extra effort to fine-tune the model effectively.

Loss Curve Comparison:



(a) Naive GCN



(b) HeteroGCN

Figure 13: Training Loss Curve of the naive GCN and HeteroGCN models

Hyperparameter Tuning Process:
A grid search approach was used to tune the following hyperparameters:

- **Hidden dimensions:** {8, 16, 32, 64}
- **Adjacency matrix versions:** {Unmodified, Self-Looped, Normalized}

Each combination was trained for 150 epochs and evaluated using the **F1-score** on the validation set. The best-performing configuration was found to be **Hidden Dimension = 16 and normalized adjacency**, which achieved the highest F1-score.

We tuned **Hidden Dimension because it controls how well the model captures patterns in node features**. A small dimension may cause underfitting, while a large dimension increases overfitting and computation costs. By testing different values {8, 16, 32, 64}, we found the best balance between performance and efficiency, with 16 providing the highest F1-score.

We tuned **Adjacency Matrix because it affect how node features are aggregated**. Unmodified adjacency only considers direct edges, self-looped adjacency reinforces node self-features, and normalized adjacency smooths feature aggregation. We found that normalized adjacency led to the best feature propagation, providing the highest F1-score.

2.4 Exploring Attention

2.4.a Implementation

The implementation for Q2.4.a is provided in the notebook. The GAN model achieved an F1-score of 0.9850.

2.4.b Discussion

Explain the attention-based aggregation:
In an attention-based GCN, each node's embedding is updated by:

1. Project each node's features (or current embedding) into a new space using a trainable weight matrix Ω.
2. For each pair of nodes (m, n) that connected by an edge, compute a similarity score, which is processing via a learnable vector ϕ applied to the concatenated embeddings of m and n, followed by a non-linearity (e.g., LeakyReLU) to enhance expressiveness.
3. Non-neighbors are masked out (set to $-\infty$), then apply Softmax normalization to ensure that each node's attention weights sum to 1.
4. Each node's new embedding is a weighted sum of its neighbors' transformed embeddings, where the weights are these learned attention coefficients, and then apply a non-linearity(eg. LeakyReLU).

Loss Curve for the Model:

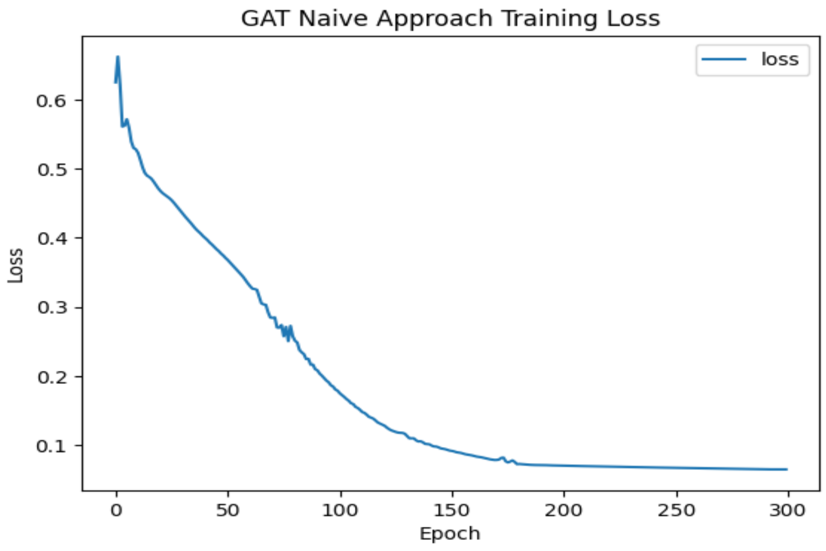


Figure 14: Training Loss Curve of the Attention-Based GCN Model

Hyperparameter Tuning Process:
A grid search approach was used to tune the following hyperparameters:

- **Hidden dimensions:** {8, 16, 32, 64}

- **Learning rates:** 0.001, 0.005, 0.01

Each combination was trained for 300 epochs and evaluated using the **F1-score** on the validation set. The best-performing configuration was found to be **Hidden Dimension = 16 and Learning Rate = 0.001**, which achieved the highest F1-score.

We tuned **Hidden Dimension because it controls how well the model captures patterns in node features**. A small dimension may cause underfitting, while a large dimension increases overfitting and computation costs. By testing different values {8, 16, 32, 64}, we found the best balance between performance and efficiency, with 16 providing the highest F1-score.

We tuned **Learning Rate because it controls the speed and stability of training**. A small learning rate may result in slow convergence, while a large learning rate can cause unstable updates. By testing 0.001, 0.005, 0.01, we found that 0.001 led to the best performance, ensuring fast convergence without instability.

2.5 Overall Discussion

Here is the bar plot:

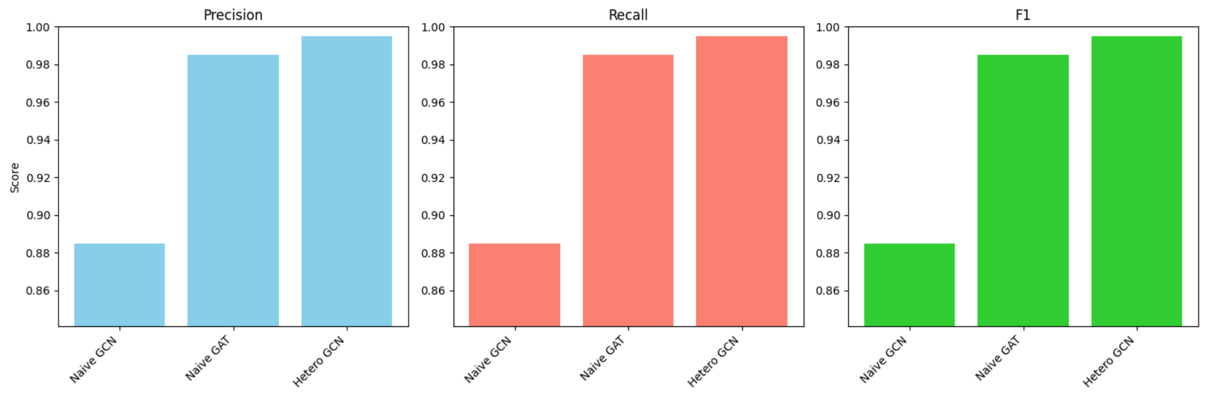


Figure 15: Precision, Recall, and F1-score comparison for GCN models

The bar plot above compares the Precision, Recall, and F1-score of the Naive GCN, Attention-Based GNN (Naive GAT), and NodeType-Aware GCN (Hetero GCN). The results clearly indicate that **Naive GCN performs the worst**, while **Naive GAT improves performance**, and **Hetero GCN achieves the highest scores** across all metrics. The Naive GCN struggles due to **its uniform treatment of all nodes**, ignoring the structural and feature differences between node types. Naive GAT addresses this limitation by **assigning different importance to neighbors through attention mechanisms**, leading to better classification. Finally, Hetero GCN outperforms both models by **explicitly modeling different node types separately**, allowing for more effective feature aggregation and classification.

The **NodeType-Aware GCN outperforms the Naive GCN** by distinguishing between node types during training. The Naive GCN applies **a single transformation**

function to all nodes, which fails to capture variations in feature distributions. In contrast, Hetero GCN **assigns separate transformations to each node type**, allowing for more precise feature learning and improving classification accuracy. This approach leads to better feature propagation and ultimately a higher F1-score.

The **Attention-Based GNN outperforms the Naive GCN** by introducing a learnable attention mechanism that dynamically adjusts the influence of neighboring nodes. Unlike the Naive GCN, which treats **all neighbors equally**, Naive GAT **assigns greater importance to more relevant nodes**, improving information aggregation. This selective weighting prevents less informative neighbors from diluting crucial features, resulting in enhanced classification performance.

3 Investigating Topology in Node-Based Classification Using GNNs

3.1 Analyzing the Graphs

3.1.a Topological and Geometric Measures

Node Degree:
The degree of a node is **the number of edges connected to it**. For a node v, the degree $d(v) = \sum_{u \in N(v)} 1$, $N(v)$ is the set of neighbours of v [3]. Thus, degree **quantifies local connectivity** of a node, which is the number of neighbours a node has. In GNN, **nodes with higher degrees may have more influence**, because they aggregate more information and influences message passing.

Betweenness Centrality:
Betweenness centrality **measures the extent to which a node lies on the shortest paths between other nodes**. For a node v, the betweenness centrality is $BC(v) = \sum_{s \neq v \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}}$, σ_{st} is the total number of shortest paths between s and t and $\sigma_{st}(v)$ is the number passing through v [2]. Thus, it **quantifies the node's role as a bridge within the network**. In GNNs, nodes with high betweenness centrality **control information flow and help identify bottlenecks**. Removing them can fragment the graph and disrupt communication, so preserving their structural integrity is essential for accurate classification [1].

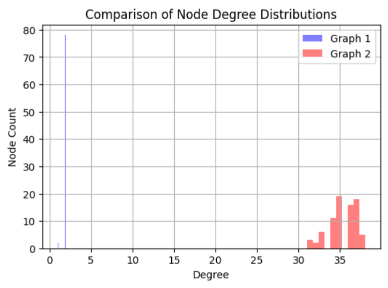
Ollivier-Ricci Curvature:
Ollivier-Ricci Curvature is a geometric measure for edges that **captures the local spread of neighborhoods** using optimal transport. It is defined as, $\kappa(u, v) = 1 - \frac{W_1(m_u, m_v)}{d(u,v)}$, where $W_1(m_u, m_v)$ is the Wasserstein-1 distance between the probability distributions of the neighbors of nodes u and v, and $d(u, v)$ is the shortest path distance between u and v [4].

It **quantifies structural properties**, with positive curvature suggesting clustered regions and negative curvature highlighting divergence or bottlenecks. In GNNs, curvature **helps guide edge weighting to improve message aggregation**, influencing information diffusion and community formation by **distinguishing clusters from key linking**

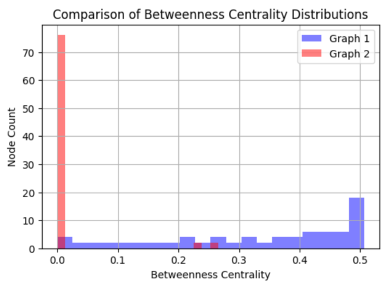
structures [5].

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

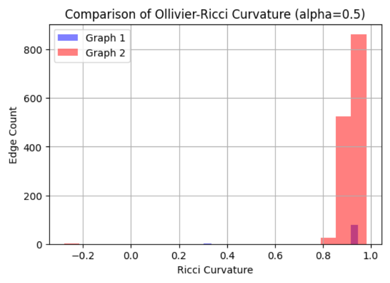
The implementation for Q3.1.b. is provided in the notebook.



(a) Comparison of Node Degree Distributions



(b) Comparison of Betweenness Centrality Distributions



(c) Comparison of Ollivier-Ricci Curvature (alpha=0.5)

Figure 16: Comparison of graph measures for two graphs

In **Node Degree Distribution**, Graph 1 Shows a highly skewed degree distribution with most nodes having very low degrees at degree = 2, while in Graph 2, the degrees are concentrated around a much higher value between 31 and 37. Graph 1 has a more heterogeneous structure, suggesting a highly sparse or even disconnected structure, which may influence information flow. In contrast, Graph 2 appears more homogeneous with a dense network where each node has many connections.

In **Betweenness Centrality Distribution**, Graph 1 shows a broader range of betweenness centrality values, with some nodes having very high centrality, while in Graph 2, Most nodes have extremely low betweenness centrality values near 0.0, with a smaller group having moderate to higher values. Graph 1's high-betweenness centrality values nodes are likely crucial for connecting different components. In contrast, Graph 2 is more evenly connected, reducing reliance on central nodes.

In **Ollivier-Ricci Curvature distribution**, Graph 1 exhibits Ollivier-Ricci Curvature values mainly around 0.9, with a small number of edges having curvature around 0.3. There is no negative curvature in Graph 1, suggesting that Graph 1 has a highly clustered and well-connected structure with minimal bottlenecks. In Graph 2, Ollivier-Ricci Curvature values are mainly between 0.8 and 1.0, with a very small portion in the negative range. Similar to Graph 1, it shows a highly clustered structure but with more curvature variation, indicating a slightly more complex topology. The presence of a few negative curvature edges suggests some bottlenecks or weakly connected regions, though they are rare.

3.1.c Visualizing the Graphs

The implementation for Q3.1.c is provided in the notebook.

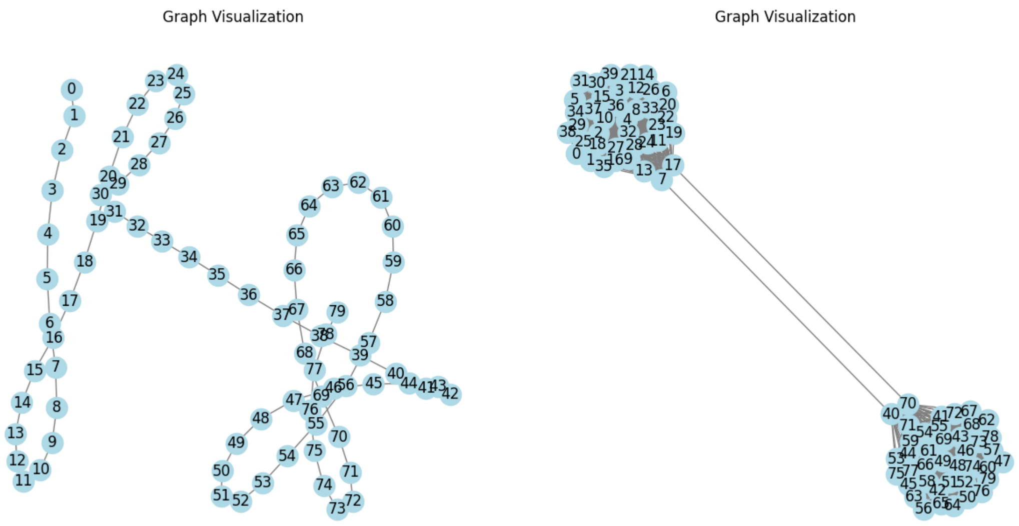


Figure 17: Visual representation of Graph 1 (left) and Graph 2 (right)

3.1.d Visualizing Node Feature Distributions

The implementation for Q3.1.d is provided in the notebook.

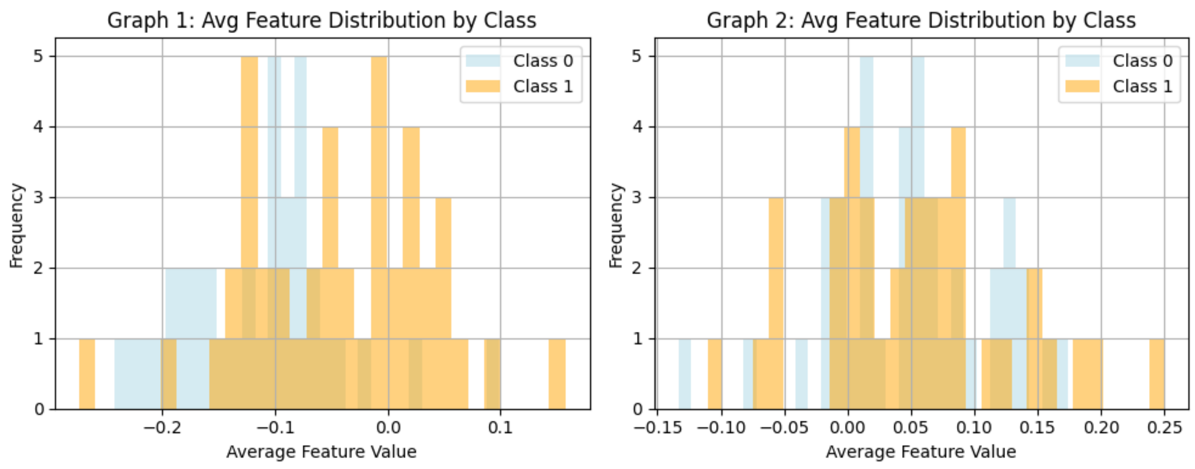


Figure 18: Node feature distribution per class for two graphs

In both Graph 1 and Graph 2, the feature distributions of Class 0 (blue) and Class 1 (orange) exhibit **significant overlap**, indicating poor separation between the classes. This lack of distinction suggests that relying purely on average feature values for classification may be ineffective, making it **difficult for a GNN model to differentiate between the two classes**. Given the overlap, additional information, such as graph structure or more detailed feature-level distinctions, may be necessary to improve class separation.

In Graph 1, feature values are more dispersed, ranging from approximately -0.3 to 0.2, with no distinct feature range dominating either class. In contrast, Graph 2 has a more concentrated distribution near zero, spanning -0.15 to 0.25. Still exhibits significant overlap, but with a more defined shape compared to Graph 1.

3.2 Evaluating GCN Performance on Different Graph Structures

3.2.a Implementation of Layered GCN

The implementation for Q3.2.a is provided in the notebook.

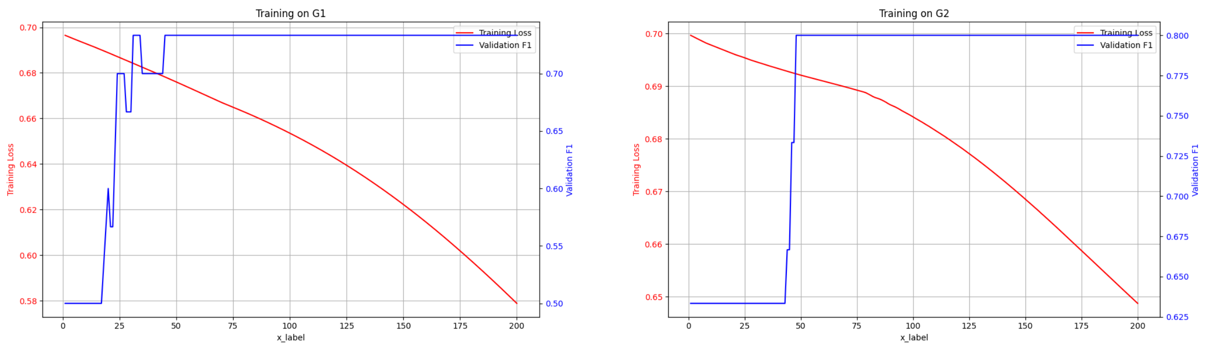
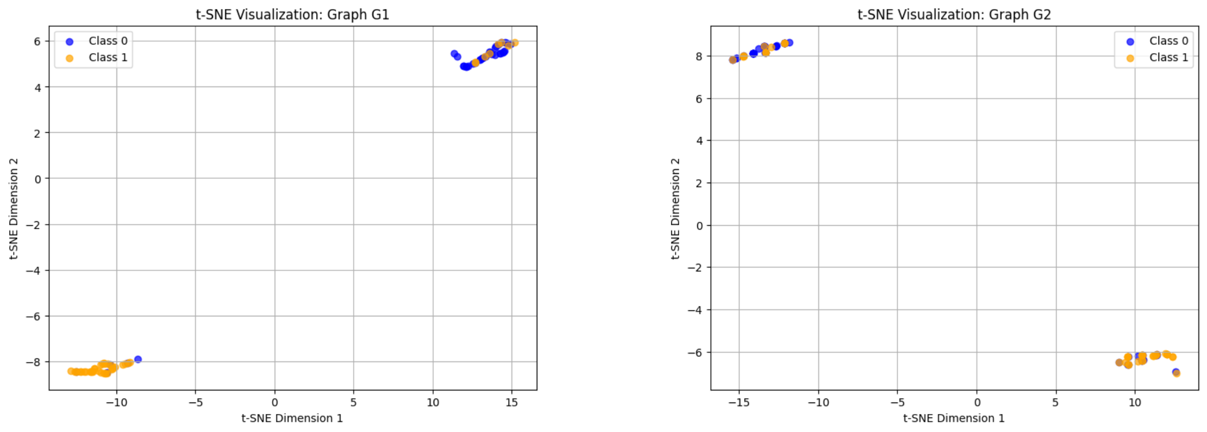


Figure 19: Training the GCN on Graph 1 (left) and Graph 2 (right)

3.2.b Plotting of t-SNE Embeddings

The implementation for Q3.2.b is provided in the notebook.



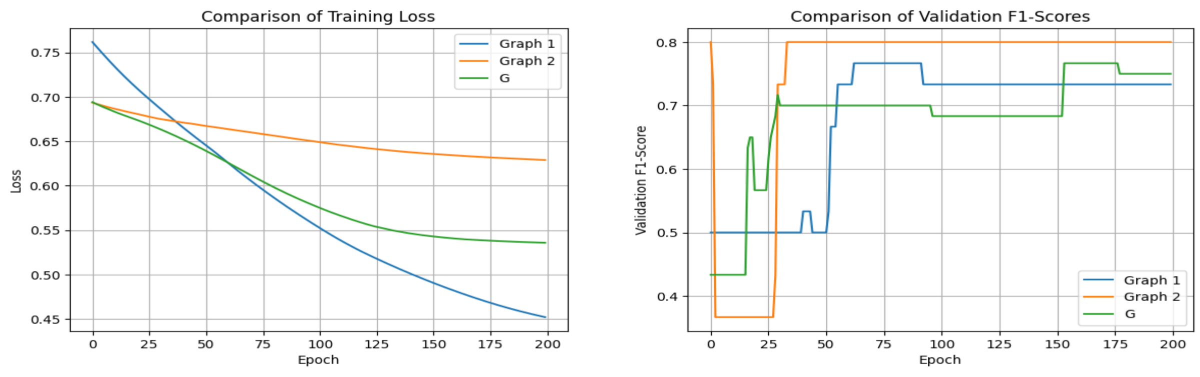
(a) t-SNE for Graph 1

(b) t-SNE for Graph 2

Figure 20: t-SNE visualization of node embeddings for G_1 and G_2

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

The implementation for Q3.2.c is provided in the notebook.

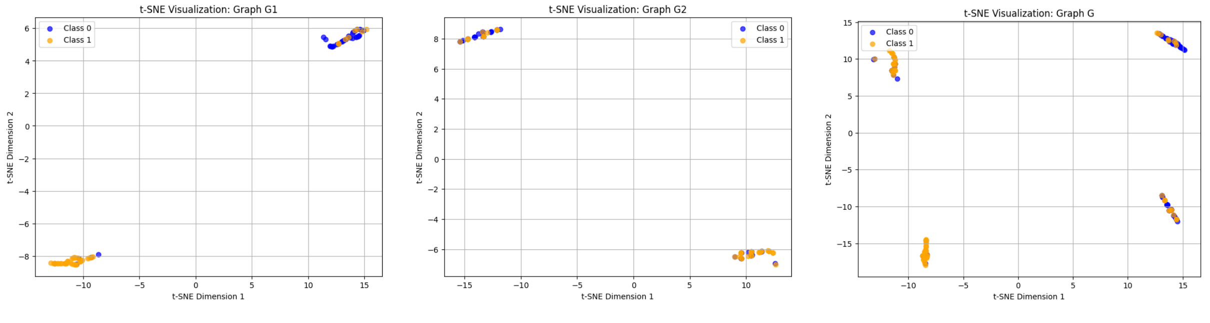


(a) Comparison of Training Loss (b) Comparison of Validation F1-Scores

Figure 21: Training and validation performance of GCN trained on G_1, G_2, and G.

The comparison of training on Graph 1 (G_1), Graph 2 (G_2), and their union ($G = G_1 \cup G_2$) **highlights key differences in learning behavior**. The training loss curves reveal that G_1 **starts with a higher initial loss (0.75) but converges quickly**, suggesting that its structure allows for efficient optimization. G_2 **has a lower initial loss (0.7) but decreases more slowly**, indicating a more complex or less learnable structure. The **graph** G **exhibits a smoother and more balanced decline**, avoiding sharp drops seen in G_1 while learning faster than G_2, suggesting that training on a combined dataset results in more stable convergence.

The validation F1-score plot **reveals differences in how well the models generalize**. G_1 **shows a gradual increase in validation F1-score**, eventually reaching around 0.75, though with minor fluctuations. G_2 **achieves a sharp jump to 0.8** early in training, suggesting that it learns class separability quickly, but remains relatively stable afterward. The merged graph G quickly **reaches an F1-score of 0.7 and maintains a stable performance**, eventually reaching 0.75 in the final epochs. This indicates that while training on G_2 alone achieves a higher early F1-score, it may be more sensitive to training data. In contrast, **merging the graphs results in a more generalized and robust model**, offering greater stability than both individual graphs.



(a) t-SNE for Graph 1 (b) t-SNE for Graph 2 (c) t-SNE for Merged Graph G

Figure 22: t-SNE visualization of node embeddings for G_1, G_2, and the merged graph G.

3.2.d Joined vs. Independent Training

Compare training results and embeddings:
As comparison in 3.2c, for independent training, both G_1 and G_2 show distinct learning characteristics. G_1 shows a gradual increase in validation F1-score, while G_2 achieves a sharp jump in early training. The t-SNE visualizations further support this observation, as **embeddings from independent training show better-defined class separability**, meaning that the model effectively distinguishes between node classes.

In contrast, training on the merged graph G results in more stable loss reduction but does not improve validation performance. The **t-SNE embeddings show increased class overlap**, indicating that merging introduces conflicting structural information, making classification harder. While merging graphs enhances stability, it does not improve class distinction, suggesting the need for processing techniques.

Formulate a hypothesis:
Merging two graphs provides the model with **a larger, more diverse set of training examples**, potentially improving generalization and helping the GCN learn feature-based patterns across graphs. However, **if the graphs have significant structural differences**, such as varying node degree distributions or feature distributions, the model may **struggle to converge** on a unified representation.

That is,

- **If G_1 and G_2 share structural or feature similarities**, merging them yields more robust embeddings and potentially higher validation F1 score.
- **If G_1 and G_2 are quite different**, merging them might mix signals, even leading to worse performance compared to independent training.

Implementation options for training:
Model Modifications

- Using **graph-specific GCN layers** for each graph to extract features before merging representations.
- Using **gattention mechanisms** to weigh more relevant connections within each graph while minimizing interference from merging.
- Training a **gmulti-task model** where individual GCNs process G_1 and G_2 separately before a shared layer integrates their learned representations.

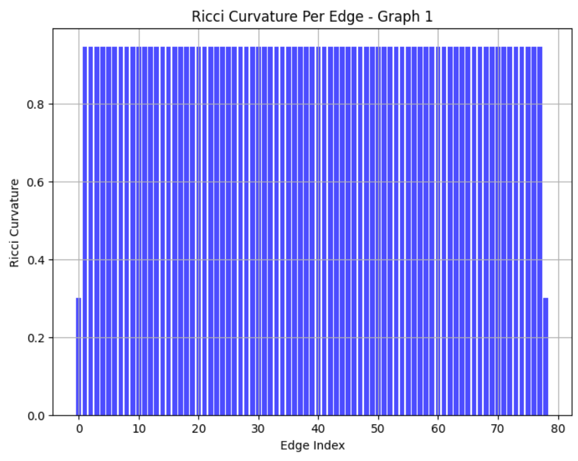
Training Modifications

- Pre-training models on G_1 and G_2 separately, then fine-tuning on G to adapt shared representations.
- Modify the adjacency matrix to preserve critical structural variations while allowing for joint training.
- Apply dropout to prevent overfitting.

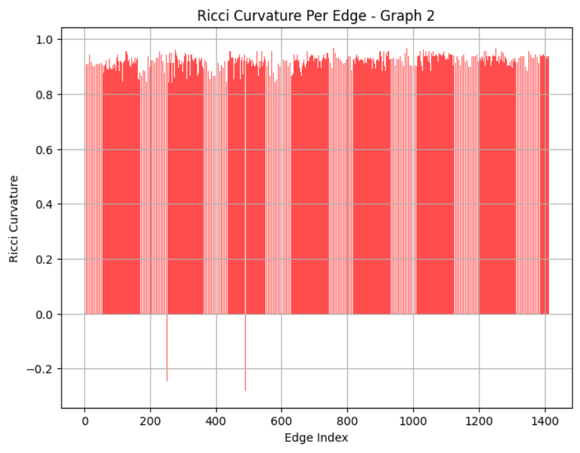
3.3 Topological Changes to Improve Training

3.3.a Plot the Ricci Curvature for Each Edge

The implementation for Q3.3.a is provided in the notebook.



(a) Ricci Curvature Per Edge - Graph 1



(b) Ricci Curvature Per Edge - Graph 2

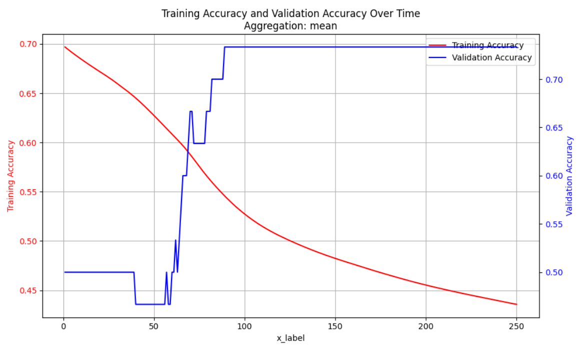
Figure 23: Side-by-side comparison of Ricci curvature per edge for Graph 1 and Graph 2

3.3.b Investigate Extreme Case Topologies

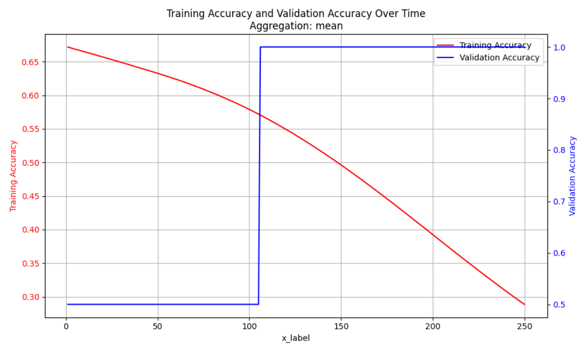
In the **No-Effect Graph**, where the adjacency matrix has been modified to remove meaningful graph structure, the **training accuracy steadily declines, while validation accuracy fluctuates significantly**. Without meaningful graph structure, the GNN behaves like an MLP, failing to capture node dependencies. The **t-SNE embeddings** confirm this, with significant class overlap, making classification difficult.

In contrast, the **Gold-Label Graph** demonstrates a much more **stable and efficient training process**. **The validation accuracy rapidly increases** and remains high, suggesting that a perfect adjacency structure enhances class separation. The **t-SNE plot** reveals well-separated clusters, demonstrating that an optimized adjacency matrix enhances label propagation and improves classification.

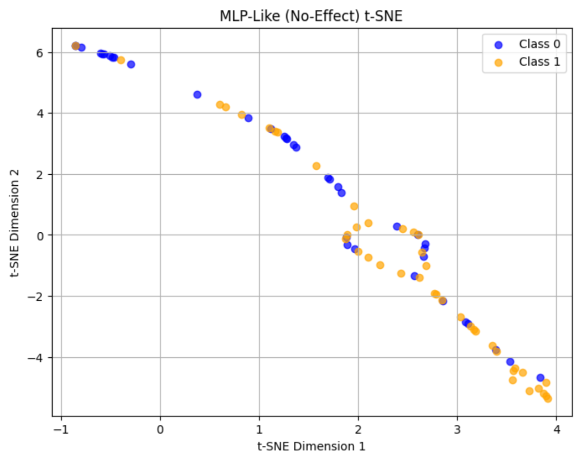
As a result, these results **highlight the importance of graph structure in GNN training**, removing it lead to weak performance, while an ideal topology significantly boosts accuracy and generalization.



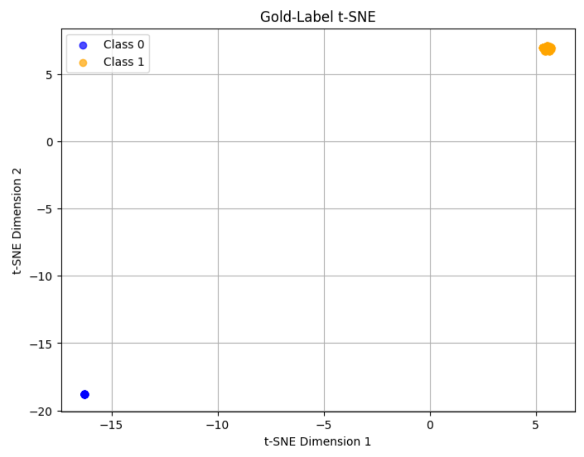
(a) Training and Validation Accuracy (No-Effect Graph)



(b) Training and Validation Accuracy (Gold-Label Graph)



(c) t-SNE Embeddings (No-Effect Graph)



(d) t-SNE Embeddings (Gold-Label Graph)

Figure 24: Training and t-SNE comparison for No-Effect vs. Gold-Label topologies for Graph 1

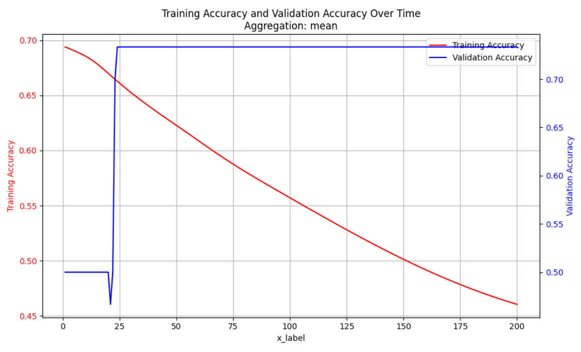
3.3.c Improving Graph Topology for Better Learning

My Modifications:

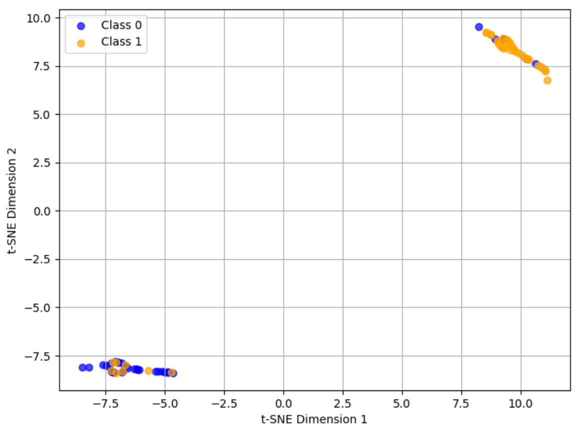
I use two methods to enhance the graph structure:

- **New edges are added between each node and its k nearest neighbors** based on Euclidean distance in feature space. This strengthens connectivity between nodes with similar features, improving message passing in GNNs.
- We computed Ollivier-Ricci curvature for each edge and **removed edges whose curvature was below a certain threshold**. This removes bottleneck connections that could introduce noise in the message-passing process.

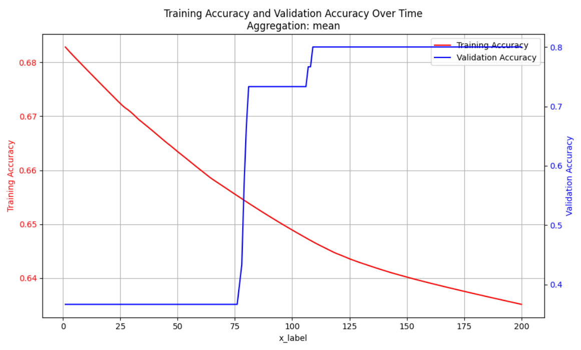
The **modifications** aim to improve intra-class connectivity with KNN edges, ensuring nodes with similar features communicate effectively. This helps the GNN propagate relevant signals. Pruning weak edges removes misleading long-range connections, often indicated by negative curvature, to refine the graph and enhance learning consistency.



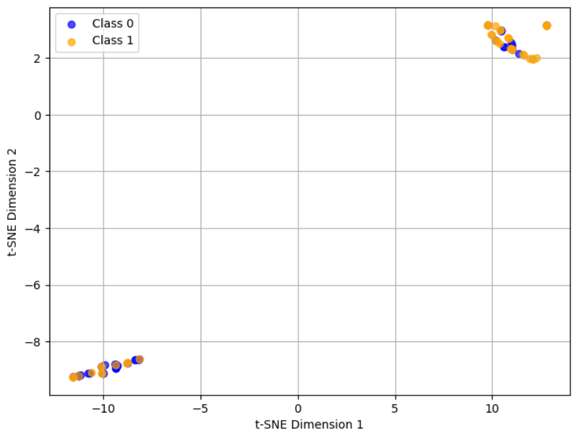
(a) Graph 1: Training and Validation Accuracy



(b) Graph 1: t-SNE of Node Embeddings



(c) Graph 2: Training and Validation Accuracy



(d) Graph 2: t-SNE of Node Embeddings

Figure 25: Comparison of Training Performance and Node Embedding Visualization for Graph 1 and Graph 2

The **modification did not improve performance**, as seen in Figure 25. The validation accuracy plateaued or slightly declined, suggesting that the changes did not enhance generalization. This is likely because the **node features were not strongly correlated with the true labels**, leading KNN edges to arbitrarily connect nodes from different classes.

Additionally, **pruning negative-curvature edges had little effect**, as most edges had high curvature (0.9 from Figure 23), meaning that very few edges were actually removed. Consequently, the overall topology remained largely unchanged, resulting in minimal impact on the learned representations and class separation in the embedding space, as shown in Figure 25b and Figure 25d, where significant overlap between classes persists.

4 References

References

[1] S. P. Borgatti. Centrality and network flow. *Social Networks*, 27(1), 2005.

[2] L. C. Freeman. A set of measures of centrality based on betweenness. *Sociometry*, 1977.

[3] M. E. J. Newman. *Networks: An Introduction.* Oxford University Press, 2018.

[4] Y. Ollivier. Ricci curvature of markov chains on metric spaces. *Journal of Functional Analysis*, 256(3), 2009.

[5] J. Topping, F. Di Giovanni, B. P. Chamberlain, and M. M. Bronstein. Understanding oversquashing in gnns through the lens of curvature. In *Proceedings of the 39th International Conference on Machine Learning*, 2022.