# Data Mining Assignment_3 Report

**Team Name: Knowledge Miners**

**Teammates:**

1. **Abhishek Goyal 2023AIB2073**
2. **M Yagnesh 2023AIB2069**
3. **Potluri Krishna Priyatham 2023AIB2084**

## Q1: Uniformly Distributed Points in High-Dimensional Spaces

**Explanation:** We are first importing required libraries like NumPy, pyplot from matplotlib, time and cdist from SciPy spatial distances. We will use NumPy to generate the data points and for a few other operations like, max, min, copy etc. we will use time to calculate time taken to run our script. We will use pyplot from matplotlib to plot all the necessary graphs. We will use cdist from SciPy spatial distances to calculate the distances L1, L2, L infinity.

After importing the necessary libraries, we will generate 1000000 random points. And then we will choose 100 query points at random from the dataset. And then we will examine the farthest and the nearest data point from each query, compute the distances using L1, L2, and L∞. Finally, we will plot the average ratio of farthest and the nearest distances versus d for the three distance measures. And then we will print all the L1, L2, and L∞ for all the dimensions.

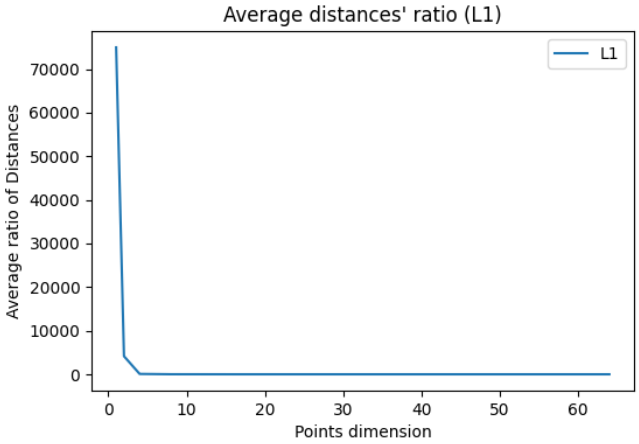
| Dimension | L1 | L2 | L∞ |
|---|---|---|---|
| 1 | 76734.5 | 76734.5 | 76734.5 |
| 2 | 3085.38714 | 3085.38714 | 2751.30658 |
| 4 | 86.10222 | 86.10222 | 70.87354 |
| 8 | 14.24476 | 14.24476 | 11.27181 |
| 16 | 5.15456 | 5.15456 | 4.23091 |
| 32 | 2.8281 | 2.8281 | 2.39541 |
| 64 | 2.02221 | 2.02221 | 1.75728 |

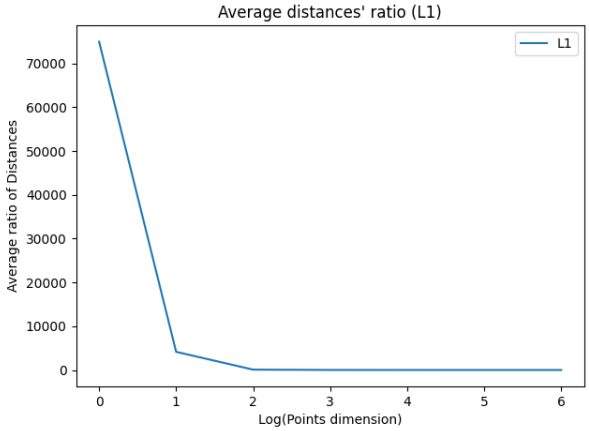
Figure: Table showing L1, L2, and L∞ for various dimensions

As observed from the above data, as the dimension increases points the average ratio of distances decreases exponentially.

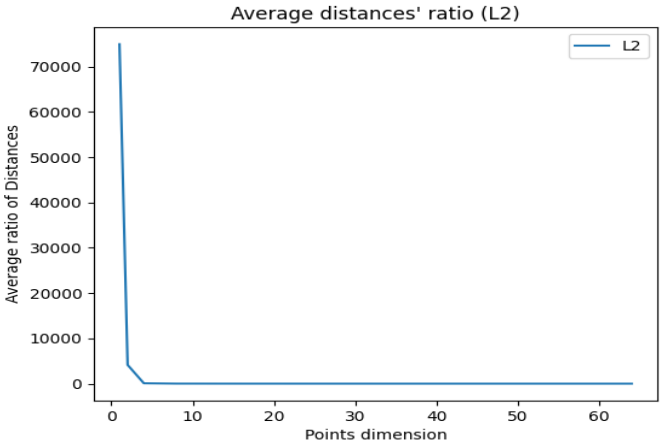
## Visualizations

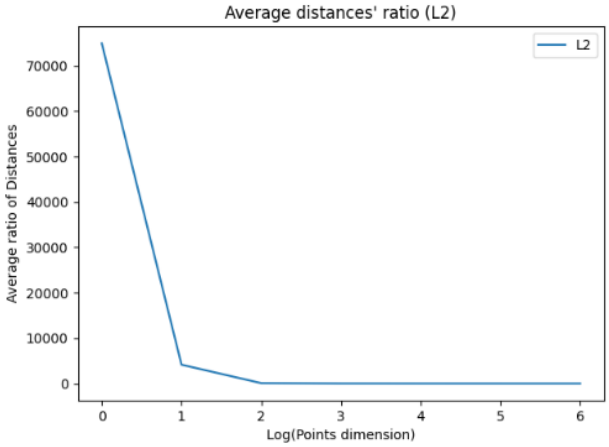
Plots for Average ratio of Distance L1 over points dimension and log of points dimension



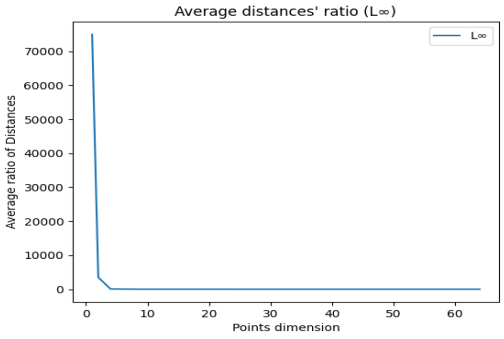



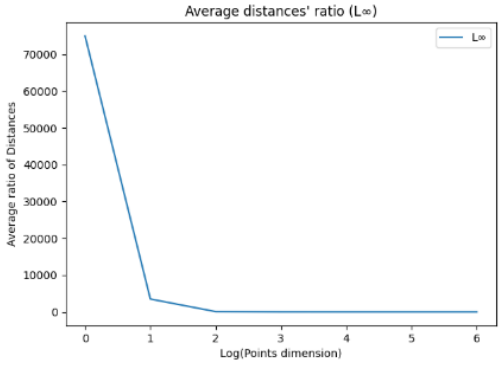

Plots for Average ratio of Distance L2 over points dimension and log of points dimension



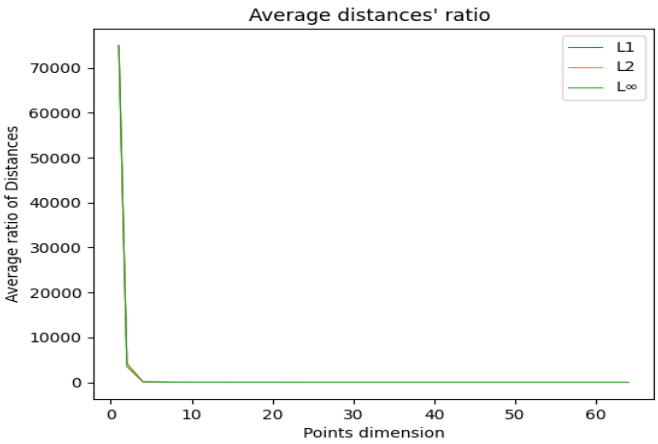



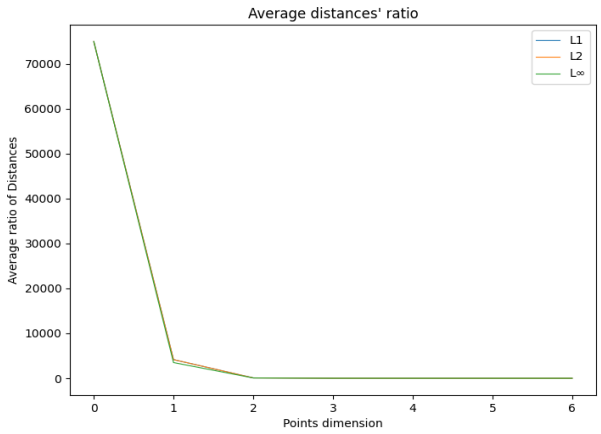

Plots for Average ratio of Distance L infinity over points dimension and log of points dimension



Plots for Average ratio of Distances L1, L2, L infinity over points dimension and log of points dimension




## Q2: Graph Classification and Regression Using Graph Neural Networks

**Approach:** We are using GINE model which is a major improvement over GIN model with the ability to consider edge features during aggregation process. Check GraphClassifier class for complete information of parameters taken.

Power Transformer based normalization is used to exclude outliers. Adam optimizer is used for varying learning rates.

**Reasoning behind model choices:** The dataset has edge features, which is why we will implement the classic GINE model instead of GIN where GIN model is used when the dataset does not have edge features. We have referred the conference paper Strategies for Pre-training Graph Neural Networks presented at ICLR 2020.

Note: We have included a pdf copy of the above conference paper and included a web link to the pdf.

**Approach:** We are using GINE architecture for embeddings. 6 Layer Neural Network is used then for classification and regression.

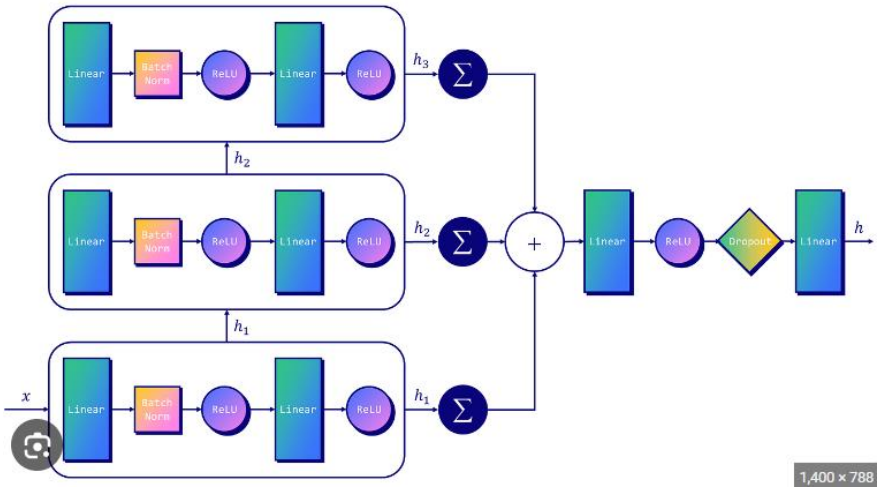



Figure 1: Model Architecture of GINE.

Formula for GINE for edge conditioning

## conv.GINEConv

*class* **GINEConv** ( **nn**: Module, **eps**: float = **0.0**, **train_eps**: bool = **False**, **edge_dim**: Optional[int] = **None**, ****kwargs** ) [source]

Bases: `MessagePassing`

The modified `GINConv` operator from the "Strategies for Pre-training Graph Neural Networks" paper.

$$\mathbf{x}_i' = h_\Theta \left( (1 + \epsilon) \cdot \mathbf{x}_i + \sum_{j \in \mathcal{N}(i)} \mathrm{ReLU}(\mathbf{x}_j + \mathbf{e}_{j,i}) \right)$$

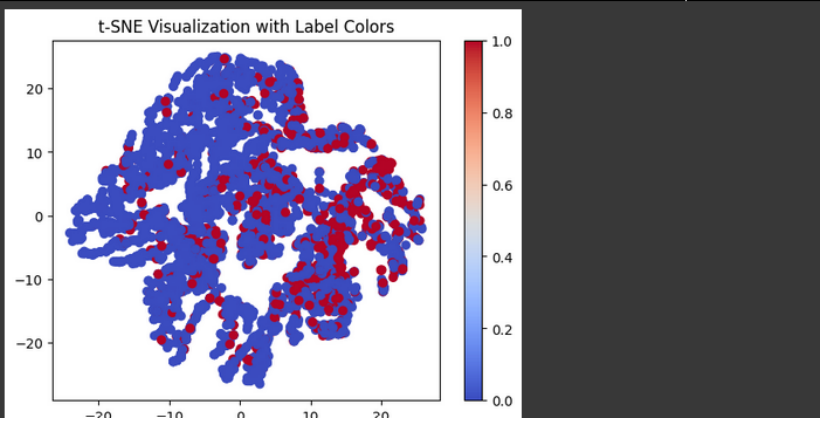
that is able to incorporate edge features $\mathbf{e}_{j,i}$ into the aggregation procedure.

## Analysis:

## Classification:

We first tried to understand the distribution of data, as we know that summing features can act as an injective function. We tried plotting the added sum of node feature of each graph and then dimensionally reducing it using t-sne to visualize the dataset.

```python
import torch_geometric.transforms as T
from torch_geometric.nn import global_add_pool
import torch
from sklearn.manifold import TSNE
import numpy as np
import matplotlib.pyplot as plt

point_list = []
y = []

for i in range(len(data_list)):
    data = data_list[i]
    point_list.append(global_add_pool(data.x, data.batch).detach().numpy())  # Convert to NumPy
    y.append(data.y.item())  # Assuming data.y is a scalar tensor, convert to Python scalar


x_2d = TSNE(n_components=3).fit_transform(np.asarray(point_list).reshape(6352,9))

plt.scatter(x_2d[:, 0], x_2d[:, 1], c=y, cmap='coolwarm')
plt.title('t-SNE Visualization with Label Colors')
plt.colorbar()
plt.savefig("img.png")

```




We found out that summing the node features is not very good input for classification, so we need something like GNN, to include the graph topology. We implemented GINE.

We ran for 200 epoch and BCE loss decreased exponentially in each iteration. The dropout of 0.3 kept regularizing the model. To make the data consistent dummy edges were added of null features [0,0,0].

```
(a3) [aib232073@khas056 ~/DM_761/A3/Knowledge_Miners/A3]
$ ./interface2.sh C train ~/DM_761/A3/Knowledge_miners/A
Training a classification model. Output will be saved at
 dataset will be loaded from.
/home/scai/mtech/aib232073/DM_761/A3/Knowledge_Miners/A3
 slow. Please consider converting the list to a single n
a-bld/pytorch_1666642881969/work/torch/csrc/utils/tensor
  x = torch.tensor(graph_node_features, dtype=torch.floa
Epoch 1/200, Train Loss: 0.8089
Epoch 2/200, Train Loss: 0.5357
Epoch 3/200, Train Loss: 0.4989
Epoch 4/200, Train Loss: 0.4639
Epoch 5/200, Train Loss: 0.4343
Epoch 6/200, Train Loss: 0.4277
Epoch 7/200, Train Loss: 0.4162
Epoch 8/200, Train Loss: 0.4208
Epoch 9/200, Train Loss: 0.4031
Epoch 10/200, Train Loss: 0.3934
Epoch 11/200, Train Loss: 0.3933
Epoch 12/200, Train Loss: 0.3835
Epoch 13/200, Train Loss: 0.3812
Epoch 14/200, Train Loss: 0.3761
Epoch 15/200, Train Loss: 0.3836
Epoch 16/200, Train Loss: 0.3770
Epoch 17/200, Train Loss: 0.3659
Epoch 18/200, Train Loss: 0.3640
Epoch 19/200, Train Loss: 0.3673
Epoch 20/200, Train Loss: 0.3578
Epoch 21/200, Train Loss: 0.3595
Epoch 22/200, Train Loss: 0.3610
Epoch 23/200, Train Loss: 0.3507
Epoch 24/200, Train Loss: 0.3418
Epoch 25/200, Train Loss: 0.3485
```

Figure 2: Training Loss

Below curve is generated on training data in classification folder as Q2_BCELoss.png

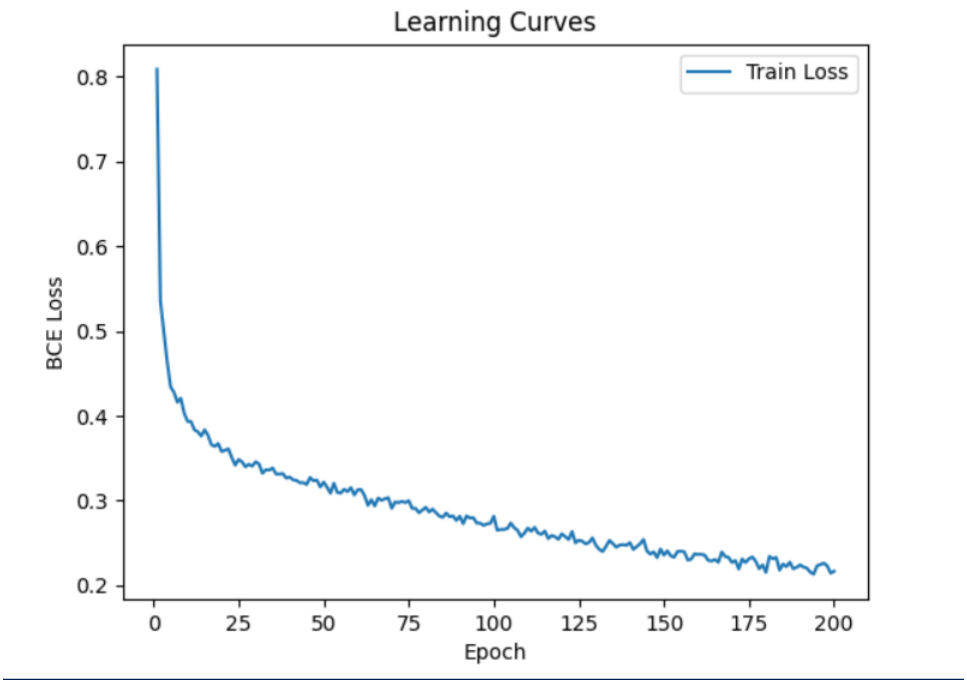



Figure 3: BCE Loss

To analyse the data that was wrongly classified we formulated it into 4 groups as follows and plotted using NetworkX:

1. Desired 1, Predicted 0 (Red)

2. Desired 0, Predicted 1 (Yellow)

3. Desired 1, Predicted 1 (Green)

4. Desired 0, Predicted 0 (Blue)

We did this to get the if there could be any structurally differentiation visible. But no clear evidence was formed (could be due to not plotting node labels). Below picture is taken from a 50 randomly generated graphs from Valid dataset. It is generated in classification folder as Graph_comparison.pdf.

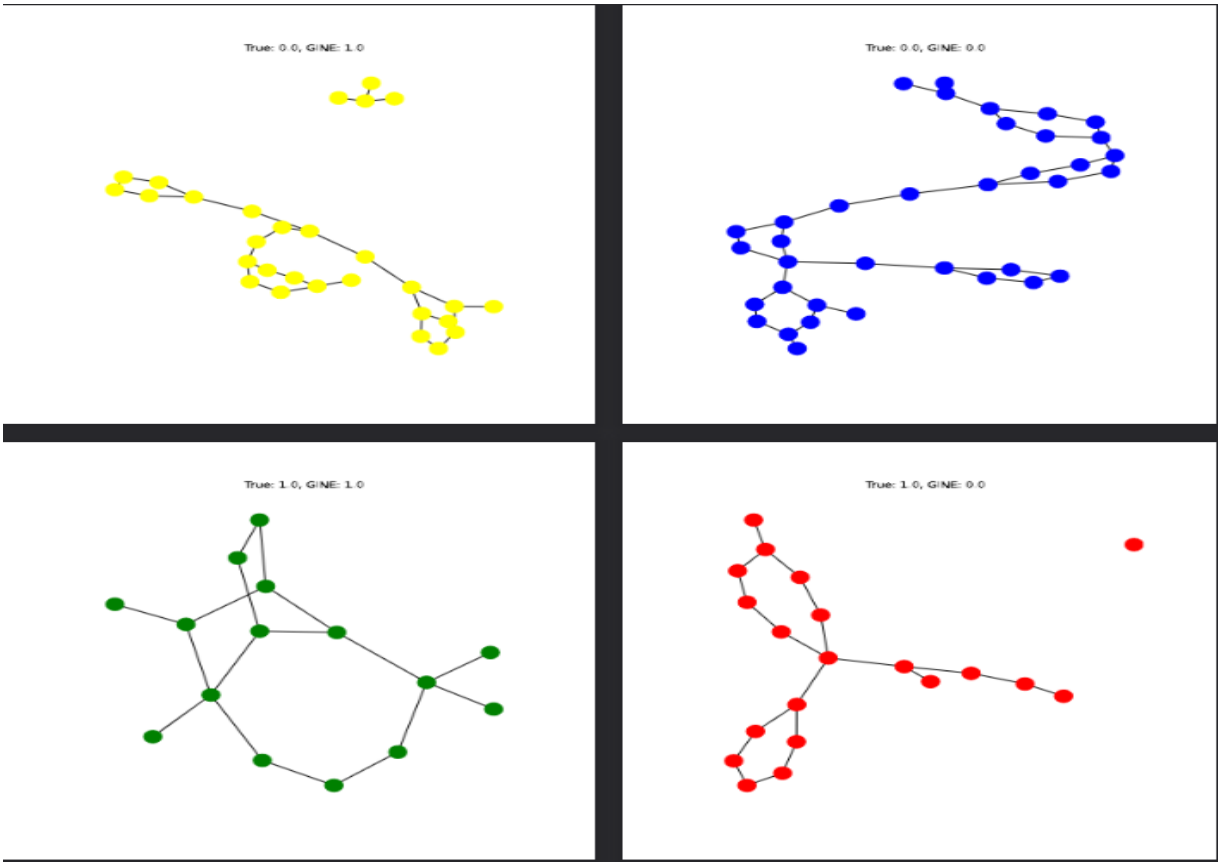



Figure 4: Analysis for precision and recall

## Comparisons with baseline models:

We compared GINE with logistic regression on ROC-AUC on valid dataset .Which shows that GINE can improve further (one way could be ensemble) , but it showing a fine discriminatory power , where as Logistic regression is unable to show any .

```
Epoch 194/200, Train Loss: 0.2132
Epoch 195/200, Train Loss: 0.2230
Epoch 196/200, Train Loss: 0.2245
Epoch 197/200, Train Loss: 0.2263
Epoch 198/200, Train Loss: 0.2230
Epoch 199/200, Train Loss: 0.2144
Epoch 200/200, Train Loss: 0.2166
ROC AUC - GINE: 0.6856, Logistic Regression: 0.5000
```

Figure 5: ROC AUC

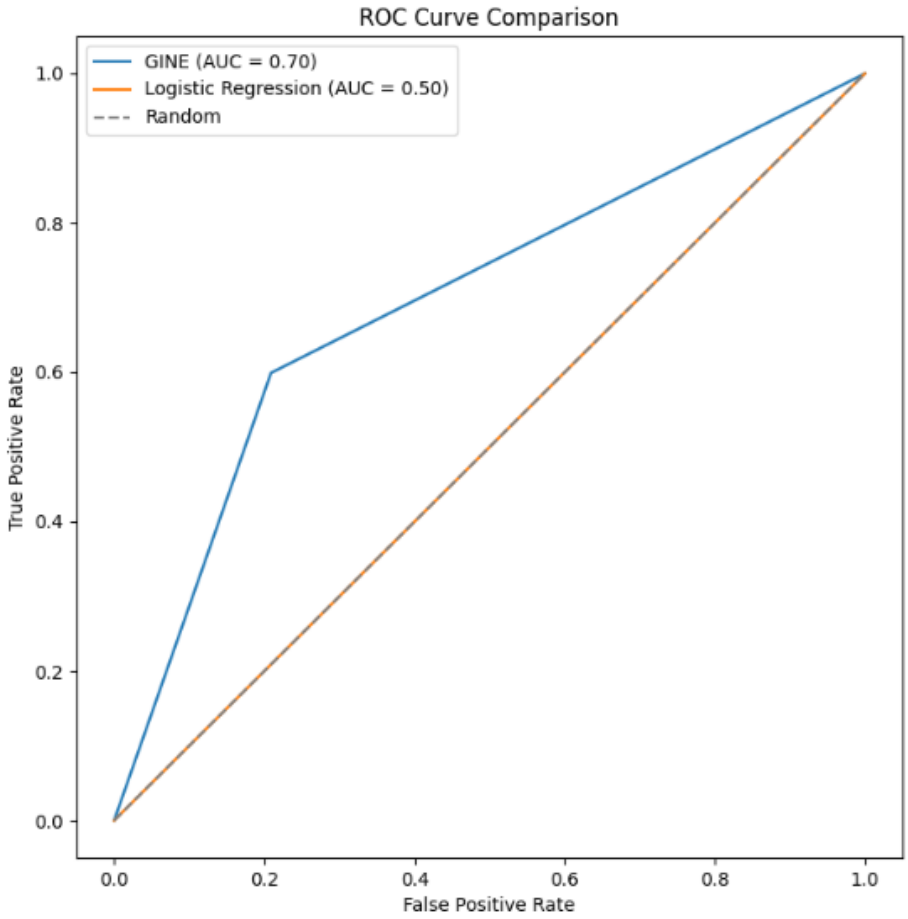



Figure 6: ROC AUC curve

Above figure is also generated as Q2_classification_baseline_comparison.png in classification folder.

## Regression:

MSE Loss is decreasing quite rapidly for regression.

```
(a3) [aib232073@khas056 ~/DM_761/A3/Knowledge_Miners/A3]
$ ./interface2.sh R train ~/DM_761/A3/Knowledge_Miners/A3/regres

Training a regression model. Output will be saved at. Dataset wi
alidation dataset will be loaded from /home/scai/mtech/aib23207
2073/DM_761/A3/Knowledge_Miners/A3/regression/
/home/scai/mtech/aib232073/DM_761/A3/Knowledge_Miners/A3/regress
extremely slow. Please consider converting the list to a single
lly at /opt/conda/conda-bld/pytorch_1666642881969/work/torch/csr
  x = torch.tensor(graph_node_features, dtype=torch.float32)
Epoch 1/100, Train Loss: 24.9395
Epoch 2/100, Train Loss: 3.0400
Epoch 3/100, Train Loss: 2.2591
Epoch 4/100, Train Loss: 1.9045
Epoch 5/100, Train Loss: 1.7516
Epoch 6/100, Train Loss: 1.6129
Epoch 7/100, Train Loss: 1.4888
Epoch 8/100, Train Loss: 1.4923
Epoch 9/100, Train Loss: 1.4132
Epoch 10/100, Train Loss: 1.4397
Epoch 11/100, Train Loss: 1.3710
Epoch 12/100, Train Loss: 1.3644
Epoch 13/100, Train Loss: 1.3354
Epoch 14/100, Train Loss: 1.2815
Epoch 15/100, Train Loss: 1.3314
Epoch 16/100, Train Loss: 1.2691
Epoch 17/100, Train Loss: 1.2430
Epoch 18/100, Train Loss: 1.2788
Epoch 19/100, Train Loss: 1.2428
Epoch 20/100, Train Loss: 1.2363
Epoch 21/100, Train Loss: 1.2265
```

Figure 7: Regression MSE loss

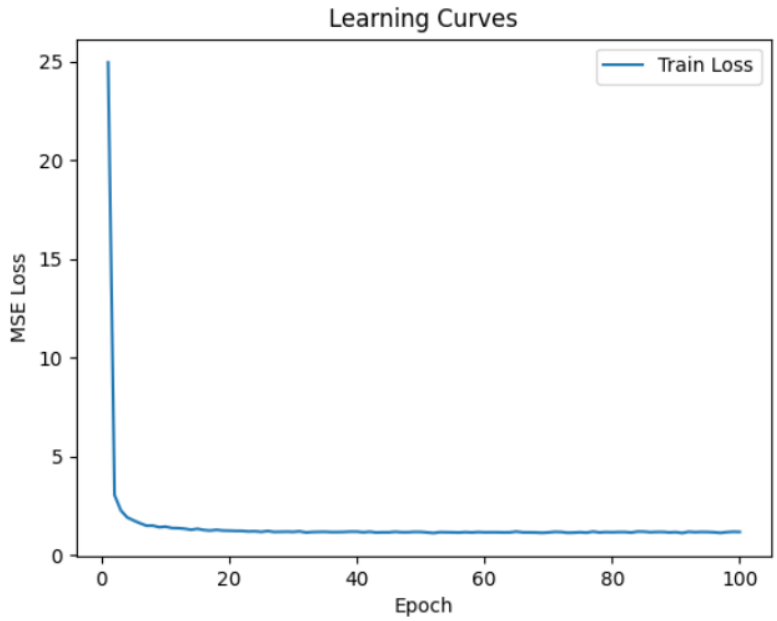



Figure 8: MSE Loss

This could also be because the model has overfit, so to check this we plotted comparison with actual values, along with linear regression. Features for linear regression were encoded by just summing the feature vectors.

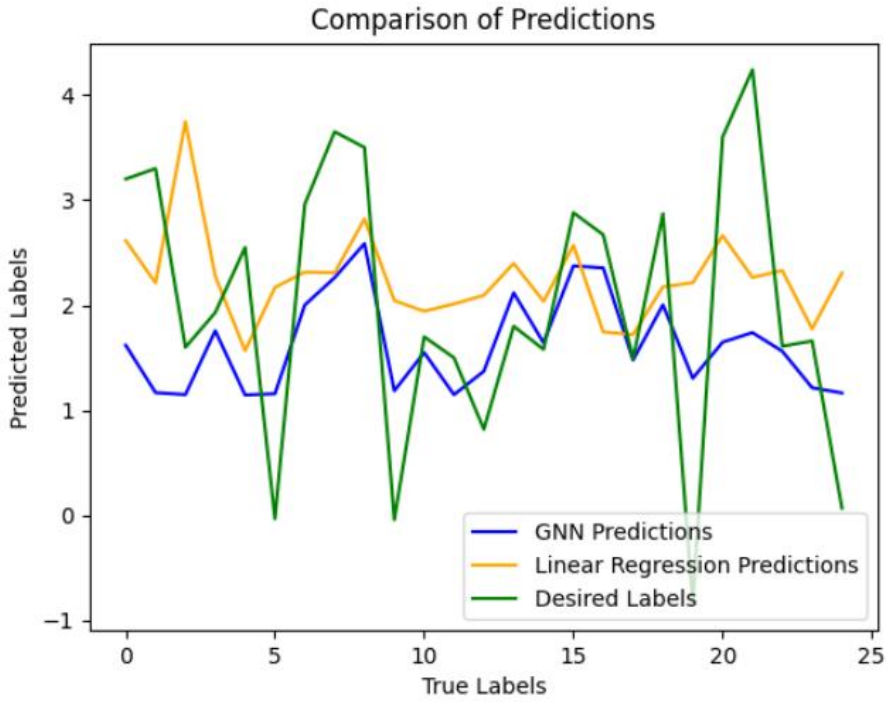



Figure 9: Regression comparison

We did not plot NetworkX graphs for regression because values were continuous, and it won't help in any analysis (For classification it helped). The above graph gives a better idea of how GINE fits the desired curve. The above graph was obtained on valid data and the model was not trained on it.

## References:

1. Strategies for Pre-training Graph Neural Networks

## Contribution :

2023AIB2073 : 34%

2023AIB2069 : 33%

2023AIB2084 : 33%