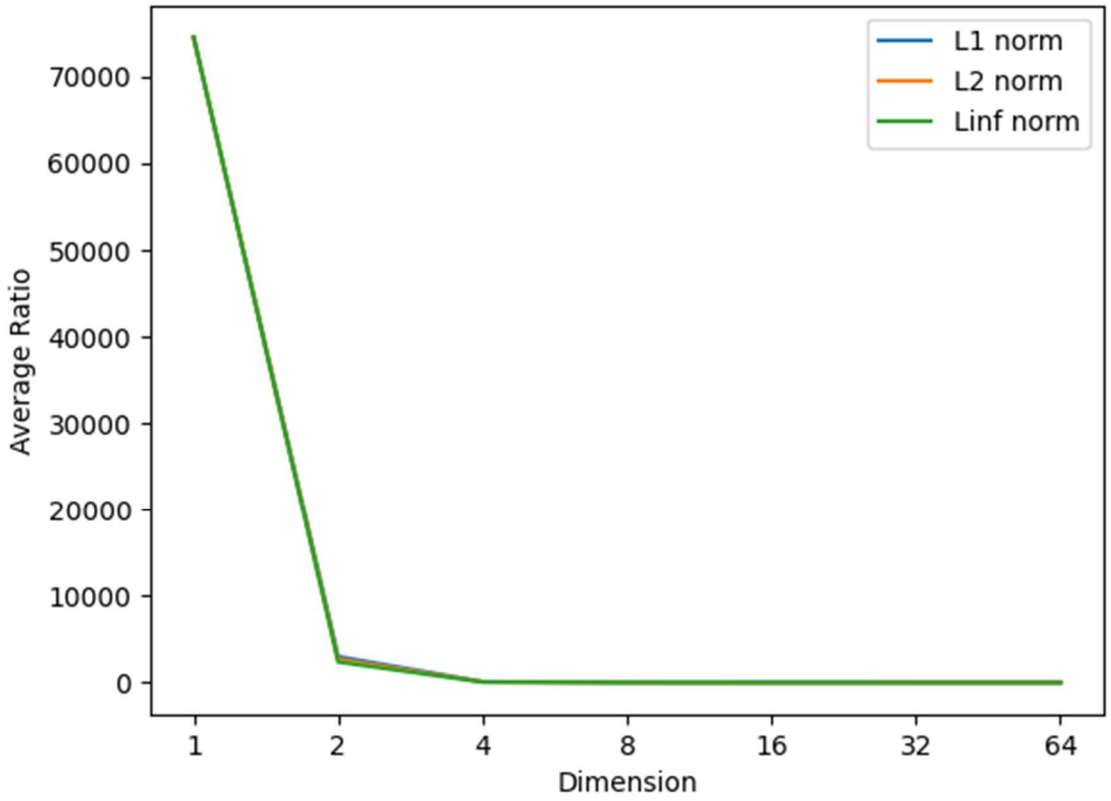
Data Mining (COL 761)
Assignment: 3

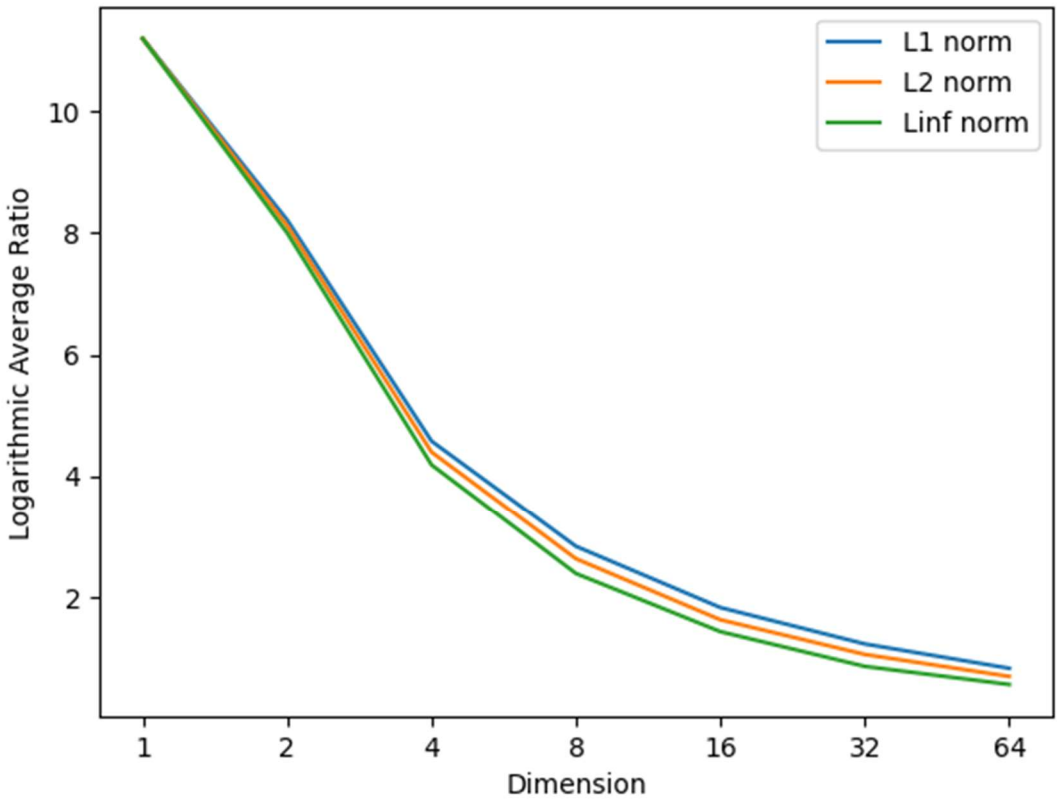
Team: Data Crashers		
Ajay Tomar	Anshul Patel	Pulkit Singal
2023AIB2071	2023AIB2072	2023AIB2064

Question 1:

The plot for the average ratio of farthest and the nearest distances versus d for the three distance measures is shown below:



The logarithmic plot for the average ratio of farthest and the nearest distances versus d for the three distance measures is shown below:



	Average ratio of farthest and the nearest distances						
	Dimension						
	1	2	4	8	16	32	64
L1 norm	7.466e+04	3.562e+03	98.884	16.974	6.073	3.375	2.307
L2 norm	7.466e+04	3.159e+03	81.172	13.906	5.032	2.865	2.015
Linf norm	7.466e+04	2.721e+03	65.263	11.214	4.069	2.418	1.755

It is observed that for d=1, the value of the average ratio of the farthest and the nearest distances is very high for all the L-norms. As the value of d increases, the average ratio decreases and approaches the value of 1 at d=64 for each of the L-norms.

In other words, in high dimensions, the difference between the minimum of the distances between any two points and the maximum of the distances between any two points becomes indiscernible compared to the minimum distance. This is often cited as distance functions losing their usefulness in high dimensions. However, recent research has shown this to only hold in the artificial scenario when the one-dimensional distributions are independent and identically distributed, which is the case as given in the question.

This happens due to the curse of dimensionality, which refers to various challenges and phenomena that arise when working with high-dimensional data. In higher dimensions, the volume of the space increases so fast that the available data becomes sparse. Therefore, the amount of data required to support the same level of statistical significance grows exponentially. This sparsity impacts the effectiveness of distance metrics such as L-norms because they are sensitive to the distances along each dimension. When attributes are correlated, data can become easier and provide higher distance contrast and the signal-to-noise ratio was found to play an important role, thus feature selection should be used.

For example, consider a hypercube in a d-dimensional space. As d increases, a larger proportion of the volume of the hypercube is concentrated in its corners. This can be shown by comparing the proportion of an inscribed hypersphere with radius r and dimension d, to that of a hypercube with edges of length 2r. The volume of such a sphere is:

$$\frac{2r^d \pi^{d/2}}{d\,\Gamma(d/2)}$$

where Γ is the gamma function, while the volume of the cube is $(2r)^d$. As the dimension d of the space increases, the hypersphere becomes an insignificant volume relative to that of the hypercube. This can clearly be seen by comparing the proportions as the dimension d goes to infinity:

$$\frac{V_{\text{hypersphere}}}{V_{\text{hypercube}}} = \frac{\pi^{d/2}}{d2^{d-1}\Gamma(d/2)} \to 0 \text{ as } d \to \infty$$

Furthermore, the distance between the center and the corners is $r*d^{0.5}$, which increases without bound for fixed r.

Question 2:

- Steps (approach, GNN architecture and reasoning behind model choices):

1. First all the data is read and stored in memory.

2. Then a GAT GNN is formed. Below is the architecture of the neural network. The dimensions of weight matrix are set so as to keep the model simple (for quick training and prediction) while maintaining the consistency for matrix multiplication. GAT GNN is used so that edge features can be given as an input, along with the node_features.

   (num_node_features = 9, hidden_chhannels = 16, num_edge_features = 3)

   a) For classification:

      GNN: For converting the node features into embeddings for input to MLP:
      Layer 1 (GAT):
      input vector: num_nodes X num_node_features
      weight matrix: num_node_features X hidden_channels
      output matrix: num_nodes X hidden_channels
      Layer 2 (GAT):
      input vector: num_nodes X hidden_channels
      weight matrix: hidden_channels X hidden_channels
      output matrix: num_nodes X hidden_channels

      Now, a sum pool operation is applied on the output matrix to convert it into a vector of dimensions 1 X hidden_channels for inputting it into the MLP network:

      MLP: For converting the output of GNN into an output class probability value:
      Layer 3 (Linear):
      input vector: 1 X hidden_channels
      weight matrix: hidden_channels X 2
      softmax function

      Loss used: Cross Entropy Loss

      output vector: 1 X 2
      1st element: Probability (class 0)
      2nd element: Probability (class 1)

```
class GNN(torch.nn.Module):
    def __init__(self, num_node_features, hidden_channels, num_edge_features):
        super(GNN, self).__init__()
        torch.manual_seed(12345)
        self.conv1 = GATConv(num_node_features, hidden_channels, edge_dim=num_edge_features)
        self.conv2 = GATConv(hidden_channels, hidden_channels, edge_dim=num_edge_features)
        self.lin = Linear(hidden_channels, 2)

    def forward(self, x, edge_index, edge_attr, batch=None):
        x = self.conv1(x, edge_index, edge_attr=edge_attr)
        x = x.relu()
        x = self.conv2(x, edge_index, edge_attr=edge_attr)
        batch = torch.zeros(x.shape[0], dtype=int) if batch is None else batch
        x = global_add_pool(x, batch)
        x = F.dropout(x, p=0.5, training=self.training)
        x = self.lin(x)
        x = F.softmax(x, dim=1)
        return x
```

b) For regression:

GNN: For converting the node features into embeddings for input to MLP:
Layer 1 (GAT):
input vector: num_nodes X num_node_features
weight matrix: num_node_features X hidden_channels
output matrix: num_nodes X hidden_channels
Layer 2 (GAT):
input vector: num_nodes X hidden_channels
weight matrix: hidden_channels X hidden_channels
output matrix: num_nodes X hidden_channels

Now, a sum pool operation is applied on the output matrix to convert it into a vector of dimensions 1 X hidden_channels for inputting it into the MLP network:

MLP: For converting the output of GNN into an output class probability value:
Layer 3 (Linear):
input vector: 1 X hidden_channels
weight matrix: hidden_channels X 1

Loss used: MSE Loss

output vector: 1 X 1
which is the regression output of the MLP

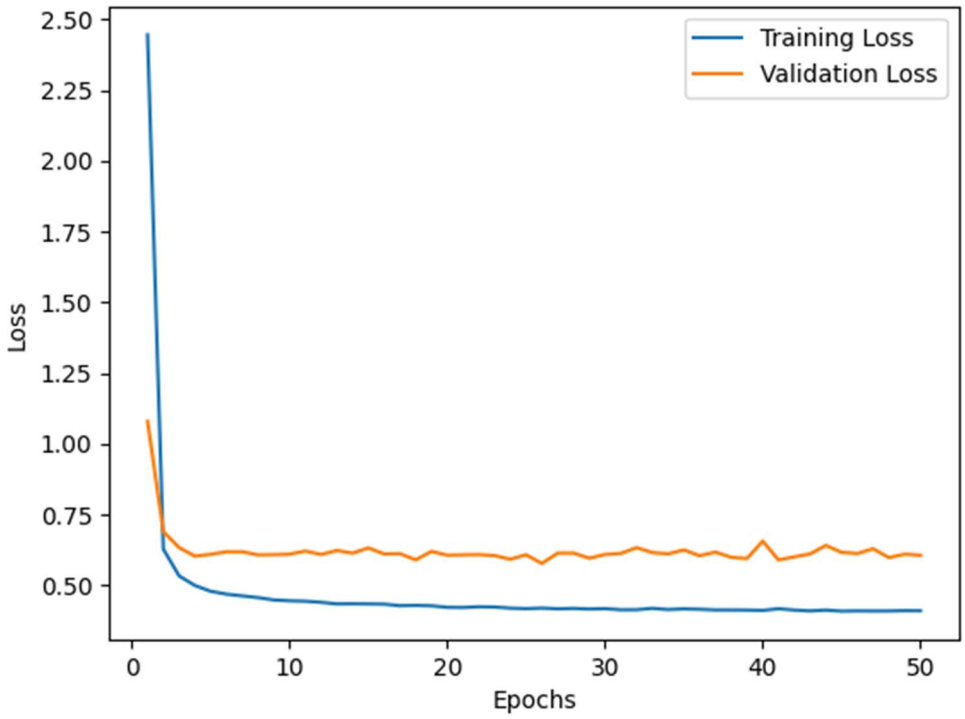
```python
class GNN(torch.nn.Module):
    def __init__(self, num_node_features, hidden_channels, num_edge_features):
        super(GNN, self).__init__()
        torch.manual_seed(12345)
        self.conv1 = GATConv(num_node_features, hidden_channels, edge_dim=num_edge_features)
        self.conv2 = GATConv(hidden_channels, hidden_channels, edge_dim=num_edge_features)
        self.lin = Linear(hidden_channels, 1)

    def forward(self, x, edge_index, edge_attr, batch=None):
        x = self.conv1(x, edge_index, edge_attr=edge_attr)
        x = x.relu()
        x = self.conv2(x, edge_index, edge_attr=edge_attr)
        batch = torch.zeros(x.shape[0], dtype=int) if batch is None else batch
        x = global_add_pool(x, batch)
        x = F.dropout(x, p=0.5, training=self.training)
        x = self.lin(x)
        return x
```

3. Then the model is trained for 50 epochs on the training data set for both the classification and regression tests. The training and validation learning curves are plotted and saved. The trained model is saved to memory.

❖ Analysis:

1. Classification:

   Below figure shows the plot of cross entropy loss for training data and validation data as a function of epochs:



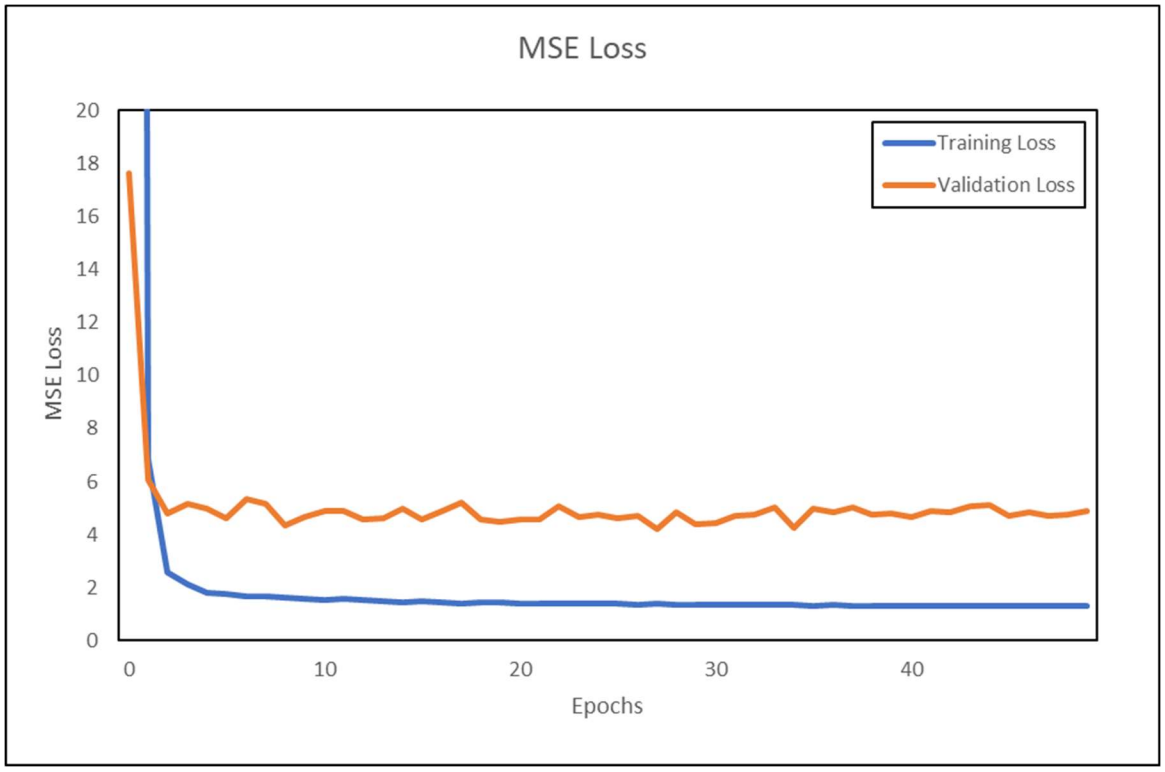
   It is observed that the loss value decreases with increasing number of epochs. At any particular time step, the loss value is lower for training data than for validation data.

   Cross Entropy loss values after 50 epochs:
   Training Data: 0.4104
   Validation Data: 0.6066

   ROC-AUC values after 50 epochs:
   Training Data: 0.621
   Validation Data: 0.581

2. Regression:

Below figure shows the plot of MSE loss for training data and validation data as a function of epochs:



It is observed that the loss value decreases with increasing number of epochs. At any particular time step, the loss value is lower for training data than for validation data.

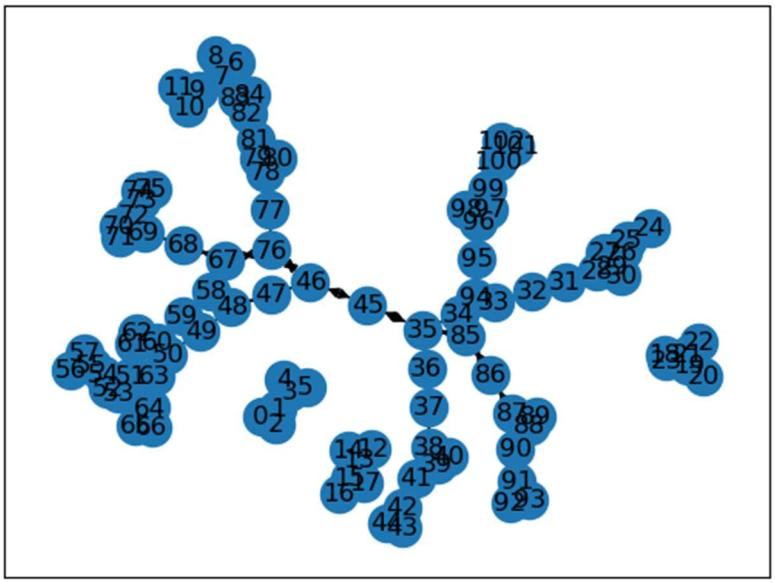
MSE loss values after 50 epochs:
Training Data: 1.2861
Validation Data: 4.9071

RMSE loss values after 50 epochs:
Training Data: 1.1341
Validation Data: 2.2152

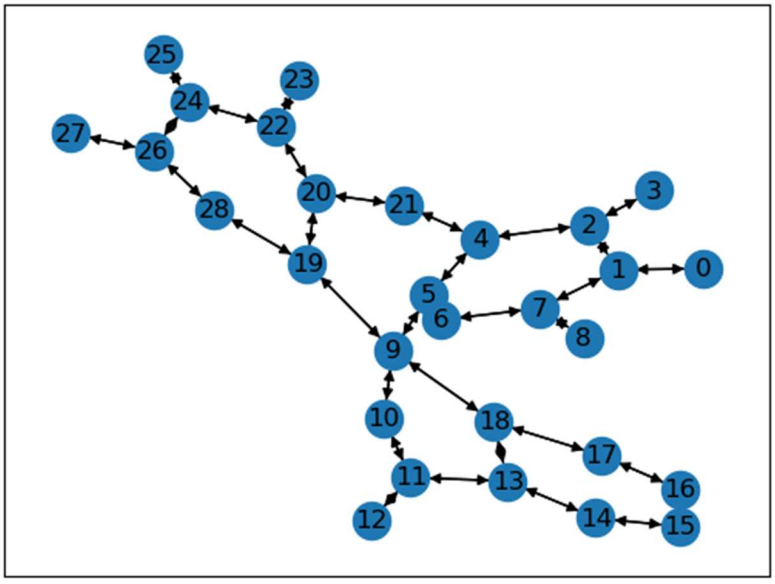
❖ Visualization:

Best Predictions:

Class: 0
Prediction: 9.028546e-08

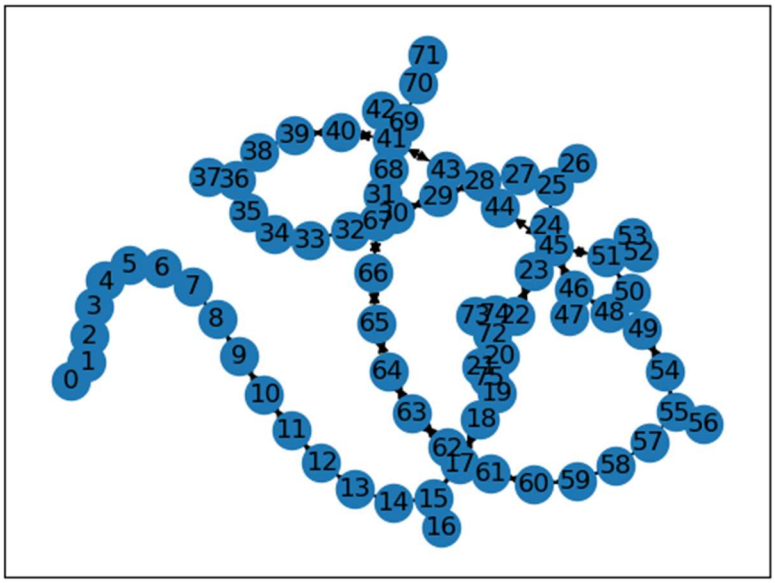


Class: 1
Prediction: 0.988749

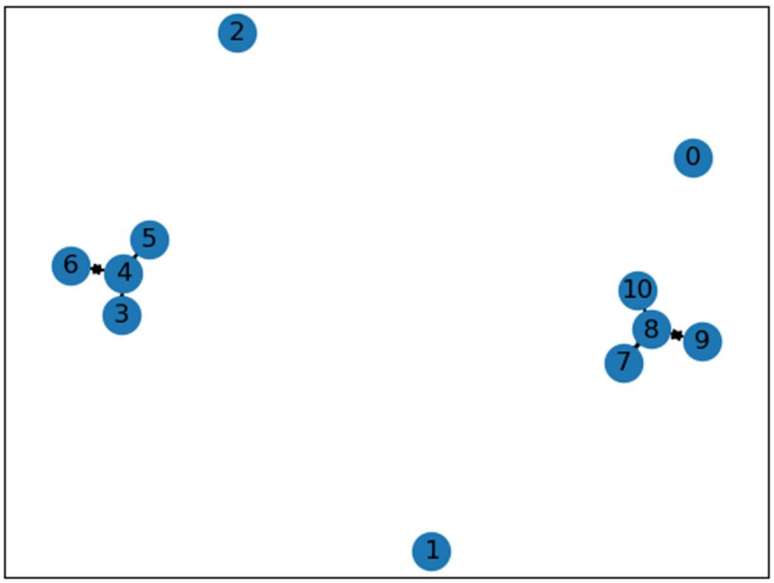


Worst Predictions:

Class: 0
Prediction: 0.9999112



Class: 1
Prediction: 0.016082337



❖ Comparisons with baseline models:

Two baseline models are created for both classification and regression:

a) Random Predictor: It gives a random prediction (0 or 1 for classification and a float value for regression) for a particular graph.

b) MLP Network: It consists only of a deep neural network (without GNN), which is trained using training data and gives a prediction (0 or 1 for classification and a float value for regression) for a particular graph.

For Classification:

Layer 1 (Linear): num_node_features+num_edge_features X hidden_channels
ReLU function
Layer 2 (Linear): hidden_channels X hidden_channels
ReLU function
Layer 3 (Linear): hidden_channels X 2
softmax function
Loss used: Cross Entropy Loss

For Regression:

Layer 1 (Linear): num_node_features+num_edge_features X hidden_channels
ReLU function
Layer 2 (Linear): hidden_channels X hidden_channels
ReLU function
Layer 3 (Linear): hidden_channels X 1
Loss used: MSE Loss

The comparison of the GNN network with the baseline models is as below:

1. Classification

a) Random Predictor:
ROC-AUC value: 0.3245

b) MLP Network:
ROC-AUC value: 0.5434

2. Regression

    a) Random Predictor:
    RMSE value: 1.975

    b) MLP Network:
    RMSE value: 1.616