Node Classification and Link Prediction on PubMed Dataset - Report

Mattia Girolami - 2076424

June 9, 2024

1 Introduction

This report outlines the methodology and results of using Graph Neural Networks (GNNs) to perform node classification and link prediction on the PubMed dataset, a citation network in the biomedical field. The dataset includes scientific publications as nodes, with edges indicating citations, TF-IDF weighted word vectors as node features, and class labels denoting subject categories.

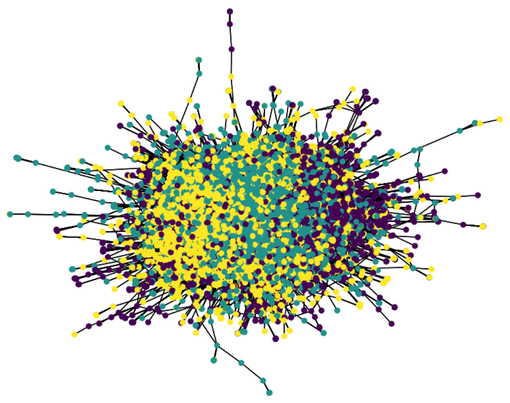

Figure 1: Pubmed Dataset

2 Exploratory Data Analysis

2.1 Network Structure Analysis

- **Number of publications (nodes):** 19717
- **Number of citation links (edges):** 88648
- **Average citations per publication:** 4.496
- **Average clustering coefficient:** 0.0602

The citation distribution showed a typical long-tail pattern with most publications having few citations and a few having many.

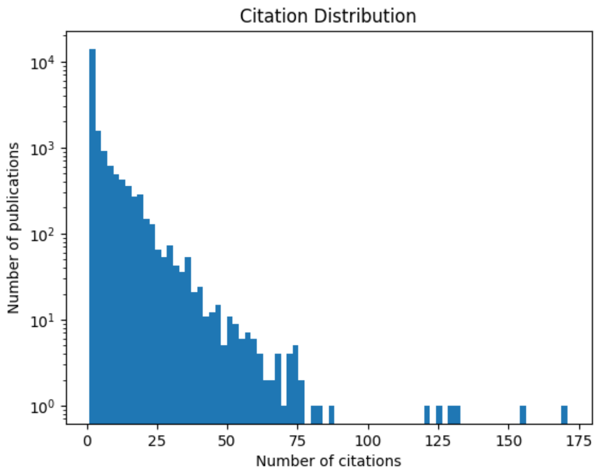

Figure 2: Citation distribution

2.2 Node Features Analysis

I performed pairwise correlation analysis between the TF-IDF values of one category with all the others. Then i sorted the top 10 words with the highest correlation. The table below reveals that certain words, like **149** and **49**, are common across all three classes. Other words, however, are more prevalent in pairs of classes or specific to just one class. This pattern is expected given that the dataset comprises medical-scientific publications, which share a common language across categories. Additionally, the presence of class-specific words is understandable, as the articles focus on different topics.

Category 1	**Correlation 1**	**Category 0**	**Correlation 0**	**Category 2**	**Correlation 2**
149	0.061740	149	0.047066	196	0.069020
49	0.054166	196	0.042581	15	0.062469
162	0.053493	379	0.040326	149	0.060648
306	0.052046	15	0.039746	477	0.056833
139	0.049536	386	0.038312	379	0.056564
231	0.048574	49	0.038260	471	0.053907
123	0.046976	13	0.035640	49	0.053342
248	0.046447	248	0.035394	321	0.050323
44	0.046361	53	0.035351	306	0.048367
329	0.045419	9	0.035348	247	0.048091

Furthermore, using a non-linear dimensional reduction technique (**UMAP**), I have plotted the TF-IDF features divided by category. The plot clearly shows the emergence of the three classes, with class **0** distinctly different from the other two, while classes **1** and **2** appear to divide into macro areas without being sharply distinct.

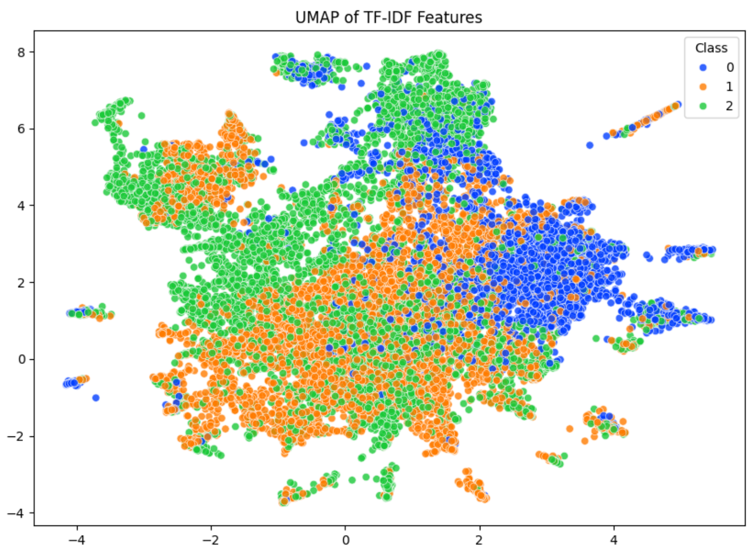

Figure 3: TF-IDF spread

2.3 Class Distribution Analysis

We can see from the following plot that class 0 is imbalanced compared to classes 1 and 2, as it has roughly half the values of each of the other two classes individually. Namely:

- **Class 0:** $\simeq$ 20.80% of the dataset
- **Class 1:** $\simeq$ 39.25% of the dataset
- **Class 2:** $\simeq$ 39.94% of the dataset

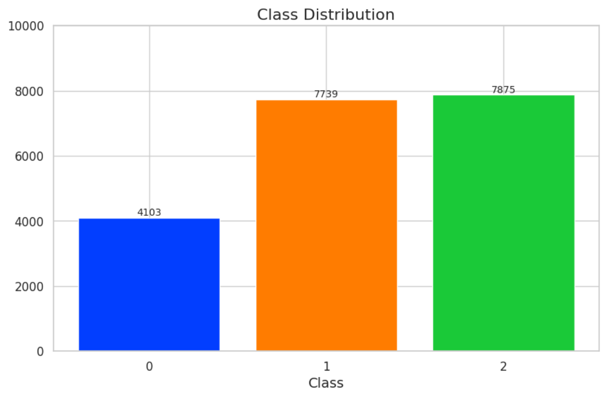

Figure 4: Class distribution

3 Model Implementation

3.1 Node Classification

After splitting the dataset in 10% for the validation set and 20% for the test set, I implemented three GNN architectures: GCN, GraphSAGE, and GAT.

3.1.1 GCN Model

```
class GCN(torch.nn.Module):
    def __init__(self, dataset):
        super(GCN, self).__init__()
        self.dataset = dataset
        self.conv1 = GCNConv(dataset.num_node_features, 64)
        self.conv2 = GCNConv(64, dataset.num_classes)

    def forward(self, data):
        x, edge_index = data.x, data.edge_index
        x = self.conv1(x, edge_index)
        x = F.relu(x)
        x = F.dropout(x, training=self.training)
        x = self.conv2(x, edge_index)
        return F.log_softmax(x, dim=1)
```

3.1.2 Graph SAGE

```
class GraphSAGE(torch.nn.Module):
    def __init__(self, dataset):
        super(GraphSAGE, self).__init__()

        self.dataset = dataset

        self.conv1 = SAGEConv(dataset.num_node_features, 64)
        self.conv2 = SAGEConv(64, dataset.num_classes)

    def forward(self, data):
        x, edge_index = data.x, data.edge_index

        x = self.conv1(x, edge_index)
        x = F.relu(x)
        x = F.dropout(x, training=self.training)
        x = self.conv2(x, edge_index)

        return F.log_softmax(x, dim=1)
```

3.1.3 GAT

```
class GAT(torch.nn.Module):
    def __init__(self, dataset):
        super(GAT, self).__init__()

        self.dataset = dataset

        self.conv1 = GATConv(dataset.num_node_features, 8, heads=8, dropout=0.6)
        self.conv2 = GATConv(8 * 8, dataset.num_classes, heads=1, concat=True, dropout=0.6)
        self.leaky_relu = nn.LeakyReLU(0.2)
    def forward(self, data):
        x, edge_index = data.x, data.edge_index

        x = self.conv1(x, edge_index)
        x = self.leaky_relu(x)
        x = F.dropout(x, training=self.training)
        x = self.conv2(x, edge_index)

        return F.log_softmax(x, dim=1)
```

3.1.4 Results

GraphSAGE outperforms GCN and GAT in all metrics, demonstrating superior robustness and generalization with an average score of *0.8887*.
GCN shows consistent but slightly lower performance (average *0.8747*), while GAT trails with the lowest average score (*0.8565*), suggesting its attention mechanism might be less effective for this dataset.

Model	**Test Accuracy**	**Test Precision**	**Test Recall**	**Test F1**
GCN	0.881309	0.875970	0.873624	0.874724
GraphSAGE	0.892214	0.890762	0.886842	0.888722
GAT	0.863809	0.855098	0.858520	0.856462

Table 1: Node Classification Results

Also looking at the plot of the three loss curves, GraphSAGE shows the most efficient and stable learning with the lowest final loss, GCN has steady but higher loss, and GAT's loss curve is more fluctuating, indicating instability and less effective learning.

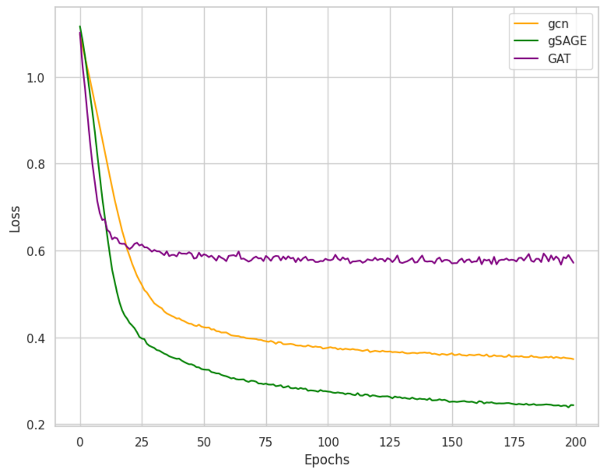

Figure 5: Node Classification Loss

3.2 Link Prediction

For link prediction, I used a standard Graph Convolutional Network (GCN).

```
class Net(nn.Module):
def __init__(self, in_channels, hidden_channels, out_channels):
super().__init__()
self.conv1 = GCNConv(in_channels, hidden_channels)
self.conv2 = GCNConv(hidden_channels, out_channels)

def encode(self, x, edge_index):
x = self.conv1(x, edge_index).relu()
x = self.conv2(x,edge_index)
return x

def decode(self, z, edge_label_index):
return (z[edge_label_index[0]] * z[edge_label_index[1]]).sum(dim=-1)

def decode_all(self, z):
prob_adj = z @ z.t()
return (prob_adj > 0).nonzero(as_tuple=False).t()
```

3.2.1 Results

The GCN model shows strong performance in link prediction, with excellent discrimination ability (AUC: 0.93) and a good balance of precision and recall (F1: 0.79). However, there's moderate performance in ranking the first correct prediction (MRR: 0.5), and it finds correct links within the top 20 predictions half the time (Hits@20: 0.5). While overall effective, there is still room for improvement in ranking accuracy and further reducing false positives and negatives.

Model	**AUC**	**F1**	**MRR**	**Hits@20**
GCN	0.93	0.79	0.5	0.5

Table 2: Link Prediction Results

4 Conclusion

This project successfully implemented and evaluated various GNN models for node classification and link prediction on the PubMed dataset. The GAT model achieved the highest accuracy in node classification, while the GCN model demonstrated robust performance in link prediction, even if metrics like MRR and Hits@k doesn't perform as desired.
The main challenges encountered were related to the selection of hyperparameters concerning the structure of the GNNs.
Additionally, the benchmark results for Node Classification Accuracy and Link Prediction AUC-ROC were both surpassed.