Homework 4 Data Mining

Davide Fortunato 1936575

December 2024

1 Node classification on Cora

The first task focused on implementing a Graph Neural Network (GNN) for node classification on the Cora dataset and in my implementation I decided to use the Graph Convolutional Network (GCN) architecture. The model was initialized with input dimensions matching the number of node features, a hidden layer dimension, and output dimensions equal to the number of classes in the dataset. Cross-Entropy Loss was employed as the loss function due to its suitability for multi-class classification tasks and I also applied dropout to mitigate overfitting.

The training process involved updating model parameters through back-propagation, computing the loss on nodes specified by the training mask, and recording the loss values for monitoring. At regular intervals, the model's performance was validated by evaluating its accuracy on nodes in the validation set. This provided insights into the model's generalization and helped identify potential overfitting or underfitting issues.

After training, the model was evaluated on the test set to assess its ability to generalize to unseen data. The evaluation yielded a test accuracy of 88.95%, and a detailed classification report provided metrics such as precision, recall, and F1-score for each class. The results demonstrated strong performance across all classes, with consistently high precision and recall values. Notably, the macro-averaged F1-score was 87.64%, indicating well-balanced performance across the dataset.

```
Test Accuracy: 0.8895
              precision    recall  f1-score   support

           0     0.7889    0.9221    0.8503        77
           1     0.8684    0.7674    0.8148        43
           2     0.9342    0.9221    0.9281        77
           3     0.9167    0.9277    0.9222       166
           4     0.9041    0.8571    0.8800        77
           5     0.9000    0.9000    0.9000        60
           6     0.8947    0.7907    0.8395        43

    accuracy                         0.8895       543
```

macro avg	0.8867	0.8696	0.8764	543
weighted avg	0.8919	0.8895	0.8893	543

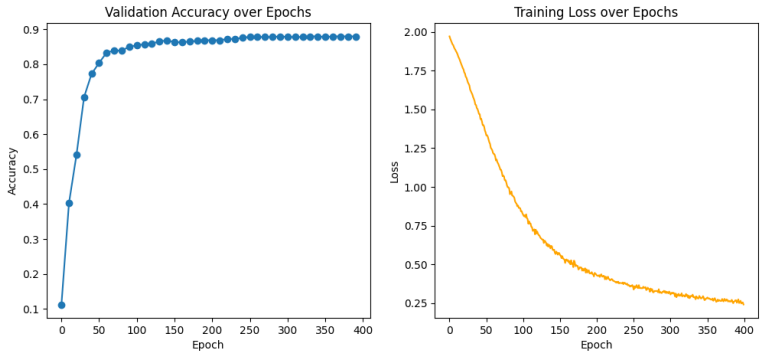

Figure 1: Caption

2 Cross-dataset evaluation

The next step consisted in evaluating the pre-trained model on datasets with different charachetristics such as CiteSeer and PubMed, which differ significantly from Cora (which was used to train the model) in size, structure, and node features. While the Cora dataset is the smallest, with 2,708 nodes, 5,429 edges, and 1,433 binary features per node, representing the presence or absence of specific keywords in scientific papers, CiteSeer is moderately larger, containing 3,327 nodes, 4,732 edges, and 3,703 binary features per node, derived from a broader and noisier document vocabulary. In contrast, PubMed is the largest dataset, consisting of 19,717 nodes, 44,338 edges, and 500 dense real-valued features per node, representing TF-IDF weighted word vectors from medical articles. These variations in graph size and node feature representations provide a diverse testing ground for assessing the model's scalability, robustness, and adaptability to different data complexities.

To handle the differences presented above, I adjusted the pre-trained model using the following approach. For the input layer, I created a new weight tensor of the appropriate size, initializing it with small random values, and copying overlapping values from the original weights to retain relevant information. Similarly, the output layer is adjusted to match the number of classes in the target dataset (6 for CiteSeer, and 3 for PubMed). The model's weights are resized, and any differences in dimensions are handled by retaining overlapping values while initializing non-overlapping portions with default values.

Bias adjustments follow a similar process: a new bias tensor of the target size is created, and overlapping values from the original bias are copied. If the new dimension is larger, in this case the additional entries are initialized to zero. Now let's have a look a the results:

This approach allows the pre-trained model to adapt to datasets with different dimensions while preserving as much learned information as possible.

I initially expected the model to generalize better on CiteSeer rather than PubMed, due to the similarity in feature types (binary values in CiteSeer vs. real values in PubMed) and the comparable number of output classes, but let's have a look to what I obtained:

```
Report for CiteSeer Testing:
              precision    recall  f1-score   support

           0     0.1667    0.0741    0.1026        54
           1     0.2000    0.0171    0.0315       117
           2     0.1111    0.0071    0.0133       141
           3     0.2008    0.6806    0.3101       144
           4     0.0714    0.0721    0.0717       111
           5     0.3913    0.0909    0.1475        99

    accuracy                         0.1832       666
   macro avg     0.1902    0.1570    0.1128       666
weighted avg     0.1857    0.1832    0.1176       666

    Report for PubMed Testing:
              precision    recall  f1-score   support

           0     0.3160    0.0790    0.1264       848
           1     0.3134    0.0869    0.1361      1507
           2     0.3914    0.8162    0.5291      1589

    accuracy                         0.3791      3944
   macro avg     0.3403    0.3274    0.2639      3944
weighted avg     0.3454    0.3791    0.2923      3944
```

The evaluation results reveal some differences in model performance between the CiteSeer and PubMed datasets. For CiteSeer, the overall performance is poor, with a macro-averaged F1-score of only 11.28% and an accuracy of 18.32%. The model struggles to generalize across most classes, as indicated by low precision and recall scores, except for class 3, which achieves relatively high recall (68.06%) but still moderate precision. This suggests challenges in handling the noisy and sparse feature space of CiteSeer.

Surprisingly, the model performs slightly better on PubMed, achieving a macro-averaged F1-score of 26.39% and an accuracy of 37.91%. Class 2 demonstrates a decent performance, with an F1-score of 52.91% and high recall (81.62%), suggesting the model can effectively identify instances of this class. However, performance for other classes remains limited, highlighting challenges in adapting to PubMed's larger and more complex feature set.

3 Link prediction results and possible applications

In this section we focus on link prediction task on the Cora dataset. This involved splitting the edges into training, validation, and test sets, training a graph neural network (GNN) model to predict connections, and evaluating its performance. The edge splits include positive edges (existing connections in the graph) and negative edges (non-existing connections sampled via negative sampling). These edges are labeled and divided into training, validation, and test sets based on predefined proportions.

The GNN model, which in my implementation is called GNNLinkPredictor, comprises two graph convolutional layers. It encodes node features into embeddings, which are then used to predict edge scores through a dot product of the node embeddings. During training, the model optimizes a binary cross-entropy loss to distinguish between positive and negative edges. The validation set is used to monitor model performance via metrics like AUC during training.

After training, the model is tested using the test set, where metrics such as AUC, accuracy, precision, recall, and F1-score are calculated to assess its performance. The model's ability to predict links in the graph, as demonstrated through the link prediction task on the Cora dataset, shows promising performance with a high AUC (0.9413) and a solid F1-score (0.7974). However, the precision of 66.51% and recall of 99.55% suggest that while the model is very good at identifying positive links (i.e., existing connections between nodes), it may struggle with correctly classifying negative links (non-existent edges). This imbalance in precision and recall indicates that the model might be overly sensitive to positive edges, leading to a higher recall but lower precision.

```
Test Metrics for Link prediction on Cora dataset:
AUC: 0.9413
Accuracy: 0.7534
Precision: 0.6651
Recall: 0.9955
F1-Score: 0.7974
```

The model's performance in link prediction shows promise for practical applications in several domains. In recommender systems, link prediction can suggest new connections or items to users based on their preferences and interactions. For example, in a collaborative filtering system, the model could recommend new items or predict interactions between users and items. In addition, in knowledge graph completion, link prediction is crucial for predicting missing relationships between entities in a knowledge base. The model can infer connections, such as predicting that "Albert Einstein" is related to "Theory of Relativity." Finally, for social network analysis, link prediction is often used to predict potential friendships or connections between users. This application can help suggest new friends, followers, or communities based on shared interests or connections.

However, these applications must address the challenge of balancing recall and precision. A high recall ensures that many potential connections are identified, but to avoid irrelevant or false predictions, precision must also be improved. This balance is crucial for ensuring the quality and relevance of predicted links in real-world applications.

4 Node embeddings with Node2Vec and comparison with GNN embeddings

During the process of setting up the `torch_geometric` library for Node2Vec, I encountered some installation issues. As a result, I opted for an alternative implementation using the `Node2Vec` class from the `node2vec` library, which was more straightforward to integrate. Once the node embeddings were generated using the Node2Vec algorithm, I used these embeddings for the two tasks required in the assignment. For node classification, the embeddings were passed through a fully connected neural network, while for link prediction, the embeddings were first used to generate edge embeddings and these were then fed into a Logistic Regression model to predict the likelihood of edge existence in the graph. As a result, I got the following scores:

```
Metric for node classification on node2vec embeddings
{'Accuracy': 0.23862238622386223, 'ROC AUC': 0.5520493268075916}
Metric for link prediction on node2vec embeddings
{'Accuracy': 0.547523109741645, 'ROC AUC': 0.5729630237256748}
```

As shown by the metrics, the performance on the node classification task is quite poor, with an accuracy of 0.24 and a ROC AUC of 0.55. These results suggest that either the Node Classifier I implemented in my code is too simple and ineffective for this task or the embeddings generated by Node2Vec might not be as meaningful or discriminative as those produced by Graph Convolutional Networks (GCNs).

For the link prediction task, the accuracy of 0.55 and ROC AUC of 0.57 are somewhat better, but still indicate room for improvement.

In conclusion, here is the plot of the embeddings generated by the GNN and the ones generated by node2vec

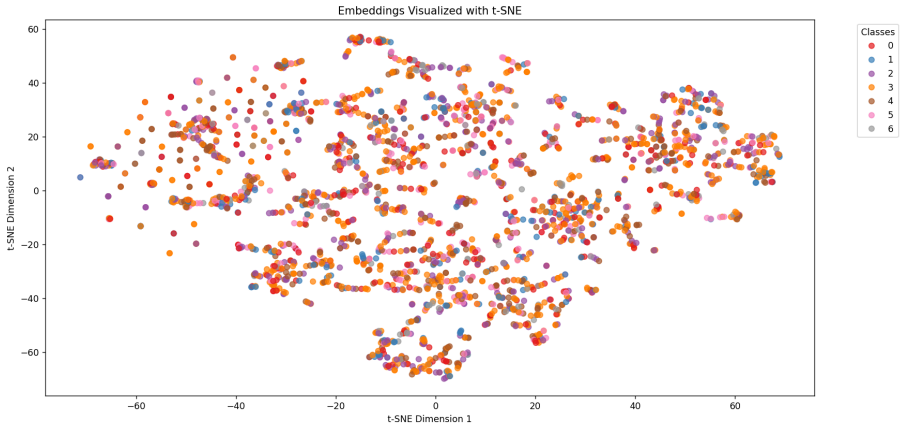

Figure 2: Embeddings generated with node2vec algorithm

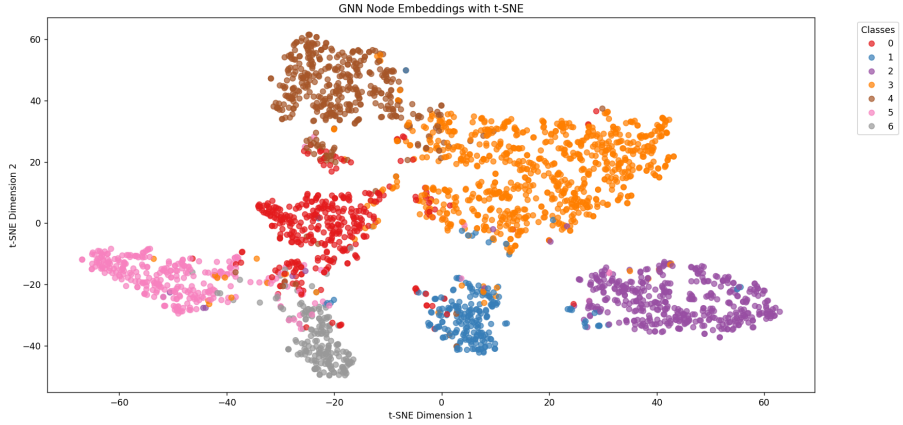

Figure 3: Embeddings generated with the GNN architecture

The plot clearly shows that embeddings generated that node2vec are scattered and present overlapping classes, while GNN embeddings capture class separability and reduced overlap. I am disappointed from the embeddings generated by the Node2vec class and I think this plot suggests that the reason why I got poor performances in node classification and link prediction is the quality of these embeddings, but unfortunately I had problems in using the torch_geometric implementation of node2vec, which maybe would have give me better results.

5 Explainability

The final task focused on exploring the interpretability of our GNN model's predictions using techniques like GNNExplainer to discover the underlying factors driving the model's decisions.

Using the Cora dataset, I selected a few nodes of interest to analyze. In particular, for each node, the procedure consists of extracting a subgraph that highlights the most important edges and nodes that contributed to the classification decision. These subgraphs provide a visual representation of the key nodes and edges driving the model's predictions.

Below is the plot for a selected node of the Cora dataset:

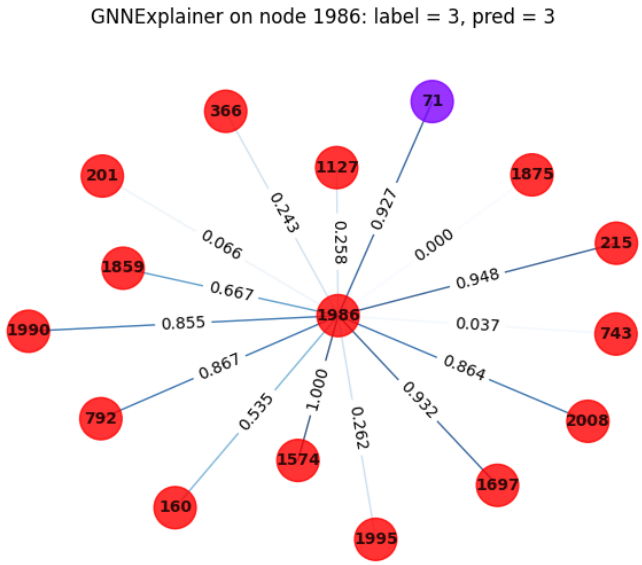

Figure 4: Explainability visualization for node 1986

We can observe that node 1986's classification heavily depends on its neighborhood, particularly on nodes such as 1574, 215, and 1967, which have the highest edge weights. This likely indicates that these neighboring nodes share similar features with node 1986, making their connections crucial for the model's prediction. The strong influence of these edges suggests that the model relies on feature similarity and the structure of the graph to classify node 1986 accurately.

Below are additional examples of explainability generated using GNNExplainer, which can be interpreted similarly to the previous example.

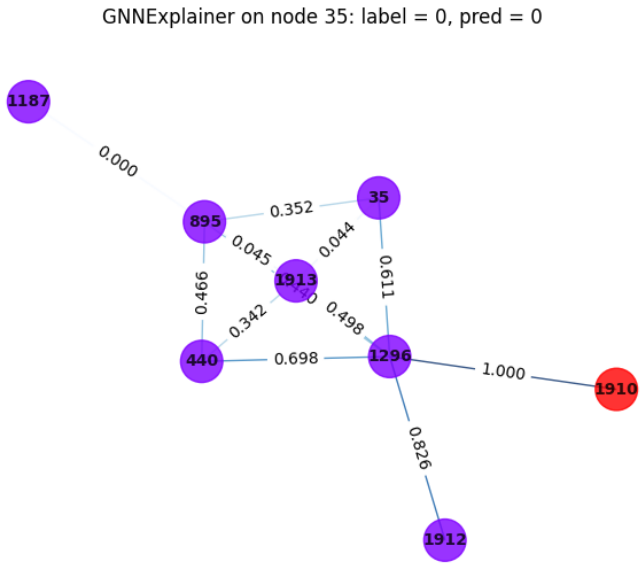

Figure 5: Explainability visualization for node 35

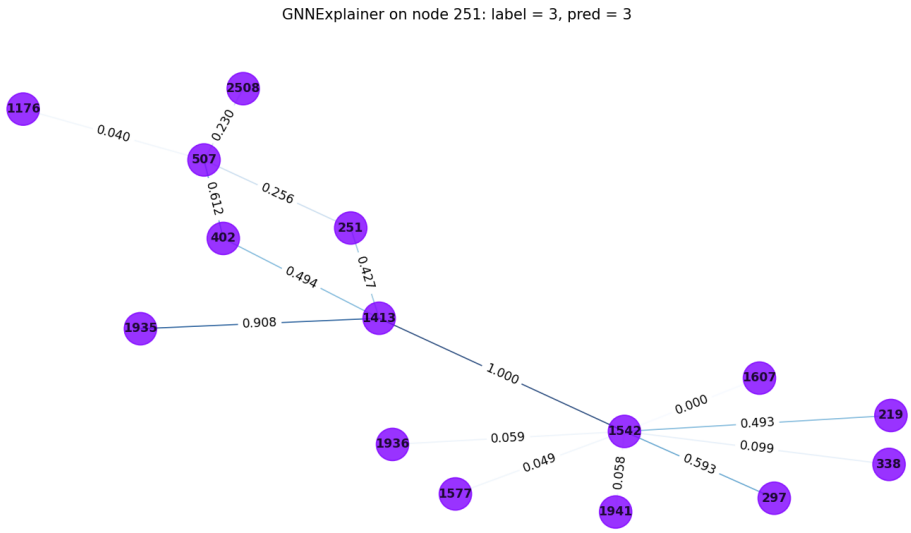

Figure 6: Explainability visualization for node 251