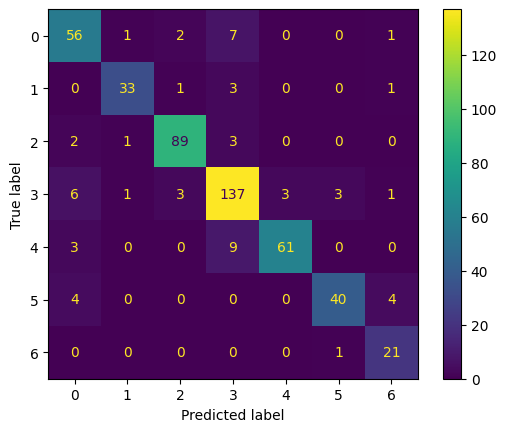
Homework 4 Data Mining

The first thing I did was to find information about the topics of the homework especially GNNs and PyTorch Geometric. I also watched the notebook given in class that helped me a lot in understanding some sections of the exercise.

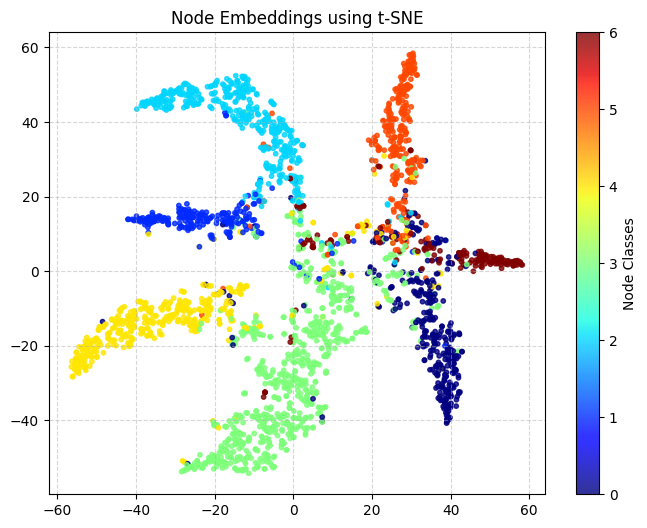
I started by loading the Cora dataset and look at it, both with features like nodes, edges, classes and visually with plots. Thinking about the first task, that is node classification, I looked at the largest connected component of the graph, a hint given by the notebook, in order to train later the neural network with it. Then I defined the GNN for the node classification task and trained and tested the model. I tried different parameters and settings, like learning rate, weight decay, loss function, hidden dimensions and dropout. It was interesting to see that by changing the hidden dimensions the metrics didn’t grow or decrease a lot, especially after 8(I then used 16). Instead, by changing the weight decay, metrics changed much more. Another interesting thing I noticed was when I calculated metrics like precision, accuracy, etc… with sklearn functions and since the classes of the dataset are not balanced I had to use the weighted average to achieve better measures. Plots about metrics of the model are on the notebook. Here I report only the confusion matrix

Here we can see the classes predicted correctly and incorrectly by the trained model. We have 497 nodes (20% of 2485 of the LCC). We can clearly see that the labels are not balanced, for example 3 has much more samples than 1 or 6. In each row there are the true labels of the nodes and in each column the predicted ones from the model. In the biggest diagonal we have the label correctly detected and they are the most.

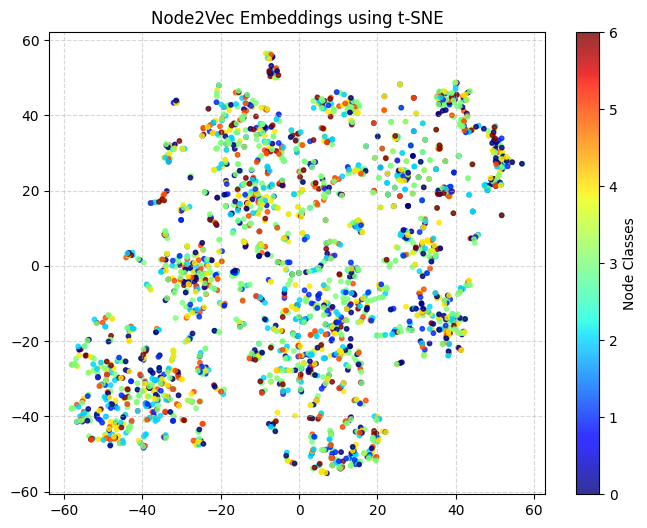
Next step was to test the model previously trained on the Cora dataset, on CiteSeer and PubMed datased. In this section I had some troubles in understanding how I could fit a dataset with different features, like number of classes, into a model that was not fitted for it. I solved the problems by filtering the weights and adapting them to each dataset and at the end I got very bad metrics for both of them, especially CiteSeer with an accuracy of 0.16. I was not able to get a better value and I found out that in case of GNN models don’t work very well for different graphs with different features.

Next task was the one of link prediction. The first problem I faced was how to generate negative edges and what features they had. I came up to a solution in using a function called RandomLinkSplit but I had a problem that I recognized only after I trained and tested the model and it was that the function creates negative samples but only for the training set, so the validation and test have only positive edges. Then I came up to a solution about generating negative edges by taking pairs of nodes and checking if they were connected with an edge and if not add a negative edge. I used a number of negative edges equal to positive so the classes were perfectly balanced. I defined the GNN for link prediction and trained it. I got very surprisingly high metrics when I used the approach of generating negative edges in that way with an accuracy near to 0.96. With the other approach of using the function for generating them I reached at most 0.74. With this model I used standard parameters for hidden dimension and output dimension 128 and 64. The plot of the ROC curve is on the notebook. By the model metrics it seems like it is very good in predicting links between nodes and it could be used in recommender systems like an e-commerce platform where the graph connects users to products they’ve interacted with. Link prediction can recommend new products by predicting potential user-item edges. In social networks, especially on platforms like Instagram, link prediction can suggest friends and people you may know based on shared connections and interests.

Then I plotted the node embeddings from the node classification model both with t-SNE and PCA. Here I report only the embeddings from t-SNE with clear clusters and only some points that don’t have a good embedding.



Second-last task was to create embedding with node2vec and look at the quality and difference of them with the ones generated by the GNNs. In this part I used 2 approaches, the first one, and I think wrong, was to generate embeddings with node2vec and then give those embeddings to the GNNs defined above for the node classification and link prediction tasks. The second approach was to generate the embeddings and use a logistic regression classifier to achieve the tasks. For node classification I had really bad results, also trying with different parameters in the embeddings, with an accuracy of maximum 0.15. I plotted the embedding with t-SNE and infact they were pretty random.



With link prediction things went slightly better by generating negative edges in the same way as before and then creating the embeddings with node2vec. The best accuracy I get is about 0.58. So at the end the embedding generated with node2vec are a lot worst than the ones generated in the convolutional layer of the GNNs and also the time taken by the node2vec approach was a lot higher then GNN(more than 10 minutes instead of some seconds).

The last part of the exercise was about Explanation. I focused on visualizing important edges in a graph based on their contribution to the prediction of specific nodes. I achieved this by visualizing a subgraph containing only the most relevant edges based on their importance scores. The nodes in the subgraph are colored according to their predicted classes, while the edges are colored and annotated based on their normalized importance scores. This representation provides an intuitive way to understand the relationships and dependencies of nodes and edges within the graph. Below there are 2 examples of it.

