SNOM HW1 - Exercise 5

Michele Luca Puzzo 1783133

May 2022

1 Data Preparation

In this exercise I handle a directed graph that represents all incoming and outgoing email between members of an European research institution: there is an edge (u, v) between two people if person u sent person v at least one email. It is noteworthy that if u has written to v not necessarily v has written to u just why is a directed graph! I have a first file.txt in which for each person (total number of people, nodes, is 1005) is associated his department that are 42. The second file.txt is and edgelist from which I have created the graph. Then as we have seen in the lab from the graph I have built the adjacency matrix and the edge_index that is a 2D tensor in which I have all the connections among nodes. As in the lab I have created a PyTorch Geometric dataset, but this time I have split my data a into train (60%), validation (20%) and test sets (20%). Another difference is that I my dataset has to receive in input what feature vector use since I have more than one.

2 Feature vectors

As first feature vector I have built one of length four in which each element is a centrality measure: the degree (sum of out and in degree of a node), the betweenness centrality, the closeness centrality and the eigenvector centrality. So to each node is associated a vector of four elements that contains information about its centrality in the graph. To build the second embedding I have used Node2Vec that is a framework for learning graph embeddings for nodes in graphs. For each node, node2vec simulates biased random walks based on an efficient network-aware search strategy and the nodes appearing in the random walk define neighbourhoods [1]. In the function there are many parameters to choose:

- dimensions: Embedding dimensions, I have chosen 64
- walk_length: Number of nodes in each walk, I have chosen 60
- num_walks: Number of walks per node, I have chosen 20

To practically implement this embedding and have an idea to select the previous parameters I have looked [2].

3 GAT architecture

At the beginning to experiment I have tried a Graph Attention Network (GAT) as neural network architecture. The key difference between GAT and GCN is how the information from the one-hop neighborhood is aggregated. GAT introduces the attention mechanism as a substitute for the statically normalized convolution operation. As opposed to GCNs, the model allows for (implicitly) assigning different importance to nodes of a same neighborhood, enabling a leap in model capacity [4]. To practically implement I have checked out the notebook [3]. I have just used two GATConv layers with eight hidden layer and eight head attention. The forward method is the same of we have written in class except for the activation function (now I have used ELU and not ReLU).

4 Performance Metrics

I have trained my model for 200 epochs, but differently from the lab lecture I have evaluated at each epoch the validation accuracy, keeping in memory which model achieves the maximum one so to use that configuration of the model on the test set. I have noticed that best model is not obtained in the last epoch, but a bit early so I think that 200 epochs was fine.

I have computed all the performance metrics setting the parameter 'weighted' so to calculate the metric for each label, and find their average weighted by support so to account the label imbalance. Looking at the confusion matrix this imbalance will be evident. Using a weighted average to compute these metrics as a consequence that f1-score is not between precision and recall, even if it is defined as the harmonic mean of them.

	Centrality Measures	Node2Vec
Training loss	1.932681	0.645274
Training accuracy	0.548922	0.887231
Validation Accuracy	0.572139	0.731343
Test accuracy	0.457711	0.666667
Best epoch	176.000000	195.000000

	Centrality Measures	Node2Vec
Test accuracy	0.457711	0.666667
precision	0.597562	0.640314
recall	0.381923	0.562438
f1-score	0.295953	0.439348
support	201.000000	201.000000

All the performance metrics achieved through the model which has used the Node2Vec embedding are greater than ones achieved through the model which has used the Centrality Measure embedding. The two embedding that I have used are expression of two different approaches: Centrality Measure embedding is an hand-crafted features based on network properties [1] indeed is based on the knowledge of the graph and measures of centrality. On the other hand Node2Vec learns feature representations by solving an optimization problem and it is constituted by three steps: preprocessing to compute transition probabilities, random walk simulations and optimization using SGD. So since it is a semi-supervised method to learn rich feature representations for nodes in a

network, it was predictable that through this embedding the model performs better than a hand-crafted, shorter feature vector.

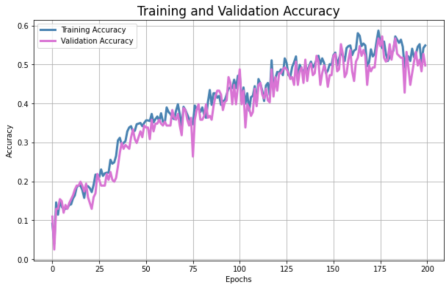


Figure 1: Using Centrality Measure embed.

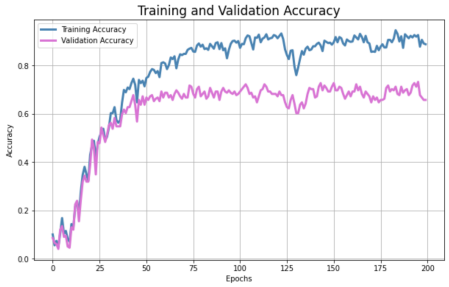


Figure 2: Using Node2Vec embed.

The values obtained as training and validation accuracy using Node2Vec embedding are very satisfying but I think it is important also to see how they are changed during the training. After 75 epochs they more or less remains constant in the second plot, while using the first embedding there is a continuous growing of them. Instead the trend of both loss, except for first epochs is continuously decreasing.

Figure 3: Using Centrality Measure embed.

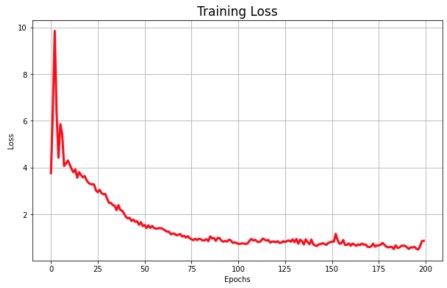


Figure 4: Using Node2Vec embed.

The confusion matrix is useful to understand that there are many people from some departments like, 13, while much less from other like 35. Moreover not all departments are included in test set because in total they are 42. This is due to the fact there is a label imbalance indeed when I have computed the performance metrics I have taken into account this fact.

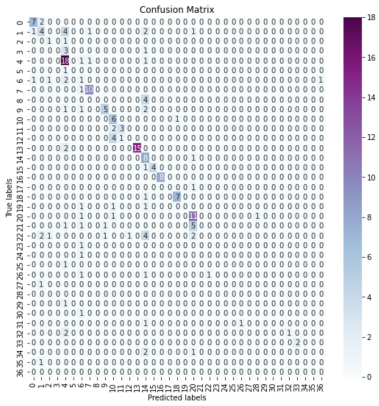


Figure 5: Using Centrality Measure embed.

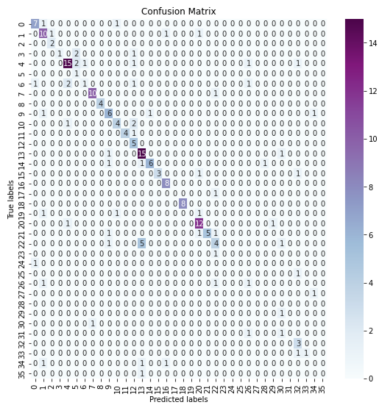


Figure 6: Using Node2Vec embed.

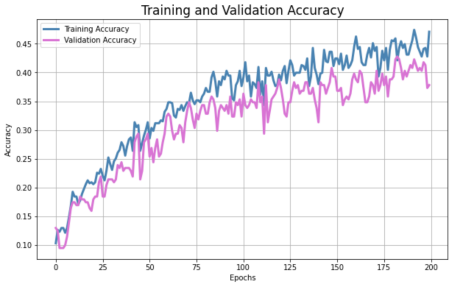
5 Bonus Point: GCN architecture

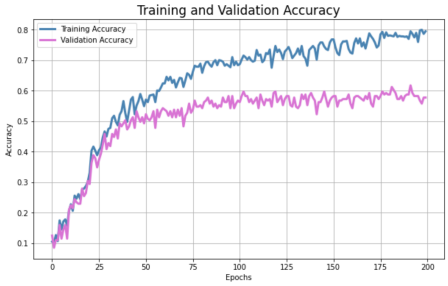
Now I have wanted to test the GCN seen during lab lecture so to compare it with GAT. The results are pretty the same: considering test accuracy GAT seems performing better but just of few decimals, and all the considerations that I have done are still valid because behaviours of embeddings, training losses, and accuracies are pretty much the same.

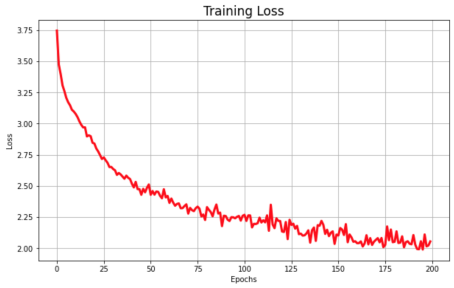
	Centrality Measures	Node2Vec
Training loss	2.053827	1.178211
Training accuracy	0.470978	0.794362
Validation Accuracy	0.427861	0.616915
Test accuracy	0.402985	0.616915
Best epoch	183.000000	192.000000

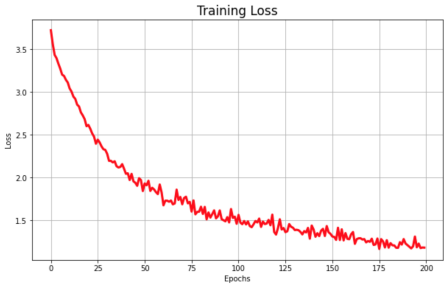
	Centrality Measures	Node2Vec
Test accuracy	0.402985	0.616915
precision	0.568050	0.685040
recall	0.402985	0.616915
f1-score	0.371546	0.572107
support	201.000000	201.000000

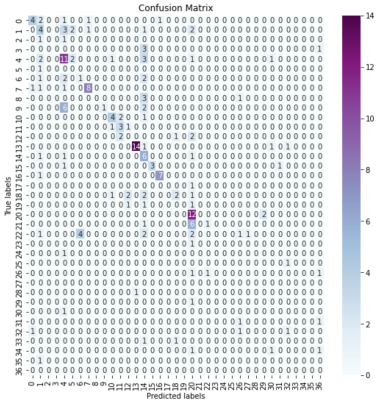
On the other hand here for example f-1 score, recall are slightly greater than before, but on the overall I can conclude that for this case study the two architecture are equivalent. The crucial impact on the performance is given by which embedding is used, and Node2Vec outperform of Centrality Measure embedding.

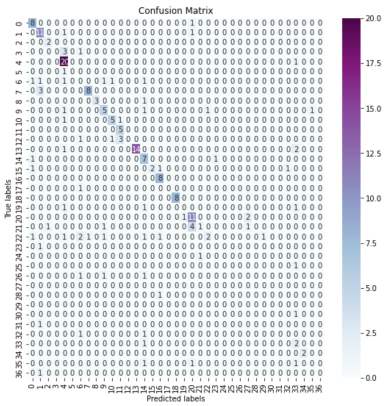












References

[1] Aditya Grover et al., *"node2vec: Scalable Feature Learning for Networks"*, arXiv:1607.00653.

[2] Github, *https://github.com/eliorc/node2vec*.

[3] PyTorch, *https://pytorch-geometric.readthedocs.io/en/latest/notes/colabs.html*

[4] Petar Veličković et al., *Graph Attention Networks*, arXiv:1710.10903