Deep Learning Homework Documentation:

coNNect - A Friend Recommendation GNN

Team “coNNect”:

Horváth Szilárd (MZ7VX5) and Szarvas Dániel (A85UKT)

Introduction and background

We chose the friend recommendation task with graph neural networks as our assignment. In this task data is given in graph form where nodes are people and the edges are friend connections in social media. As our task is to predict friend connections between people, we used graph neural networks for link prediction. We chose this task because we were both interested in a new type of data format to work with and a neural network type we had limited experience with.

Dataset

For our dataset we used the recommended dataset in the task description, we used SNAP from Stanford, this dataset contains anonymized user graphs from three social medias. The nodes have attributes, these attributes were already one hot encoded in the original dataset and it had been anonymized. One related problem to this was that different networks in the dataset from the same social media had different attributes for their nodes, which made it hard to make them the same dimension. This also meant that the node attributes were very sparse.

The dataset could be downloaded both from their website and it could be imported in Python packages as a premade dataset. For Facebook data we used the preprocessed in the imported library, for the Google+ we processed it ourselves. We tried both, because they had different pros and cons. The original dataset from their website was harder to process, however we could choose how we want to handle data.

For the imported Facebook dataset it too had to be processed but much less than the other. It was already preprocessed to a degree, so some decisions were made that we didn’t really agree with during their preprocess. It contained 10 friend networks. We had 4167 nodes and they had 1407 attributes because the attributes were combined between different networks. As data analysis we couldn’t do much because of the anonymization, so we examined the number of neighbours and the number of 1s in each node attribute column. The node attribute data was very sparse, with only a mean of 27 of number of 1s in the columns.

We also processed a google dataset. The google data was much bigger than the Facebook one. We did it in case we need bigger data or if the facebook data’s quality is not that good. For google we downloaded the raw data from their website. And we transformed it into the required data format. For google+ we only used 1 network since its size was already enough.

For GNN the dataformat is required to be a Pytorch geometric dataset. It consists of x, which are the node features and an edge_index, which contains all of the connections between the nodes with node_id pairs. Since the graph is not directional it contains an edge both ways, so we had to take it into consideration when splitting train and test data.

For the Baseline model we chose to only train it based on the node features only, that means tabular data. Since the task was link prediction we had to somehow make the tabular data compatible with that. We had different ideas for it, in the end we chose to combine two nodes into a row and the output value is a binary, based on if the two nodes are connected. We paired all nodes to all nodes, but we only used part of it because of the processing limitations.

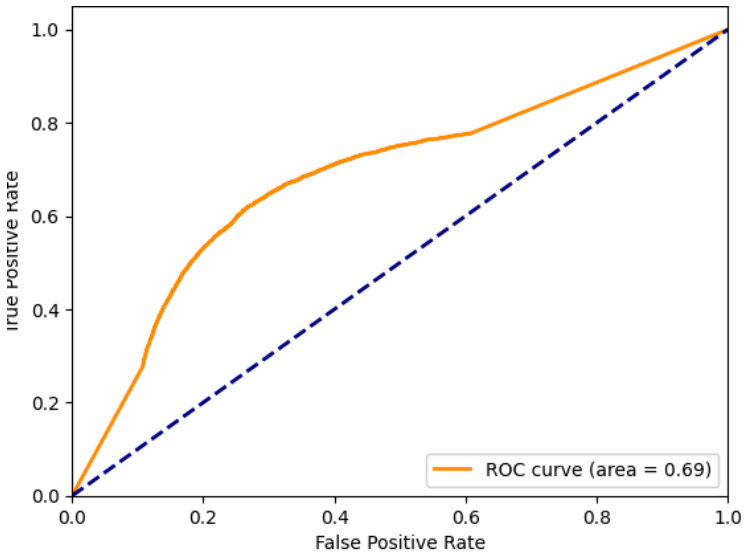
Baseline model

Since the baseline model only uses the features of the nodes to predict connection, and doesn't use the already existing connections in the graph, we need to handle sparse tabular data. We choose a regular fully connected neural network for this purpose. We thought that a neural net can recognize some connection between the features and the connections even if there are only a few present.

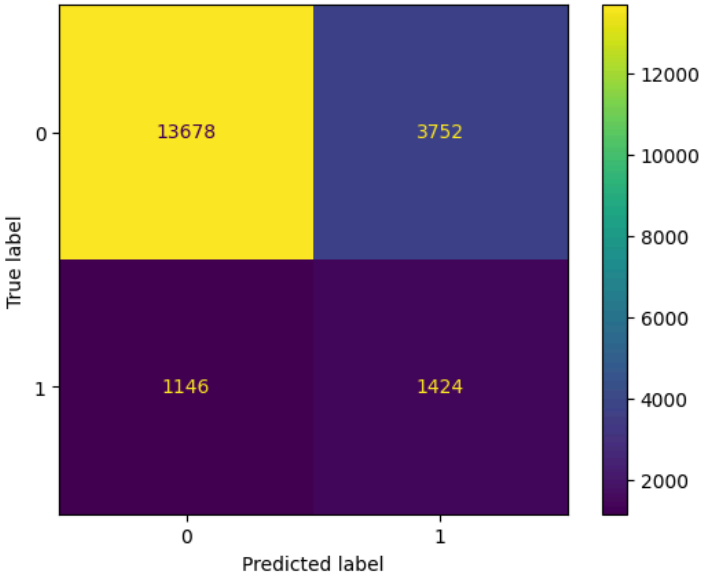
For the NN we created several layers, with adjustable sizes, and to prevent overfitting, since the data has high dimensionality, we used dropout layers with adjustable dropout weight. To find good hyperparameters we set up a sweep in Wandb which tests the net for different layer sizes and dropout rate. We examined accuracy for the models. It is important to note that there weren't huge differences between results, most of the models guessed very close to random guess. Most models overfit from the start and stop at the earliest early stopping.

From the first sweep we concluded that the smaller layer sizes performed better, dropout didn't have much role in the results. So for a second sweep we set up smaller layer sizes to try out. In this case also the results didn't differ that much, most models were still only a little better than random guess

From the sweep we set up a final baseline model. We evaluated it on accuracy and AUC. The dataset was unbalanced so we took that into account when evaluating accuracy. The model performed with 0.69 AUC score. We also examined the confusion matrix of the predictions.



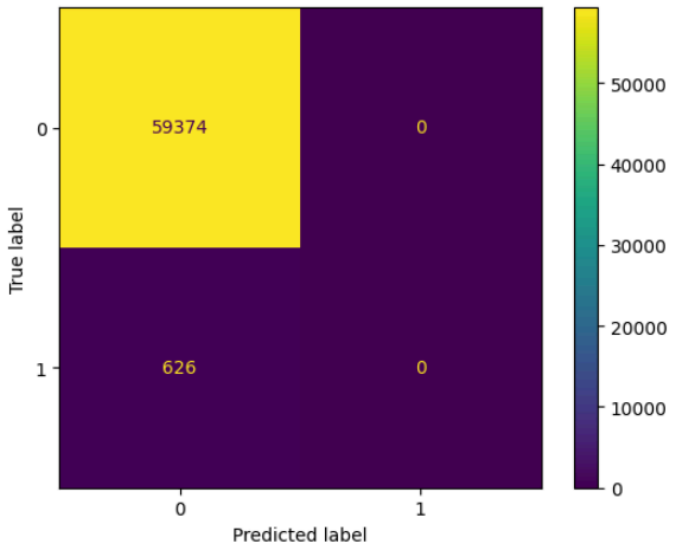
ROC Curve for baseline model Google+



Confusion matrix for baseline model on Google+

Because of the very sparse data we didn't expect much from the baseline model. It could still learn some connections and dependencies. From the confusion matrix we can see that it mostly guessed 0 since it is the dominant output in the data. It guessed nearly randomly for 1 labels.

For facebook Dataset, the model only guessed 0 and nothing else.



Confusion matrix for baseline model on Facebook

Facebook data was heavily dominated by negative labels, and the data was sparse, model model couldn't make any better connection, than to guess only 0s.

To conclude, the baseline model didn't perform that good, but we expected it since it didn't use the majority of the data, which are the graph edges. We were interested in how much will the GNN improve compared to this, with it learning from both node features and graph edges and structure.

GNN model

For our main model we used GNNs. Our choice of GNNs take Pytorch Geometric data as input which contains the nodes and their features in x and the graph edges in edge_index. It could also contain output values in y, but in our case it was not necessary. Our task was link prediction so we split the edges into train val and test datasets. We used RandomLinksplit from Pytorch Geometric for this. We specified that our graph is not directional, so during the split the data doesn't leak because of directionality. For validation and test the function also adds false edges so it can evaluate both positive and negative labels.

When designing the graph neural network, we were faced with a task that we haven't investigated before: predicting links between nodes. After looking for basic architectures that solved the problem in the past, we discovered that graph convolutional layers where we investigated two options to build our GNN: GCNConv (Semi-Supervised Classification with Graph Convolutional Networks, Kipf et al.) and SAGEConv (Inductive Representation Learning on Large Graphs, Hamilton et al.).

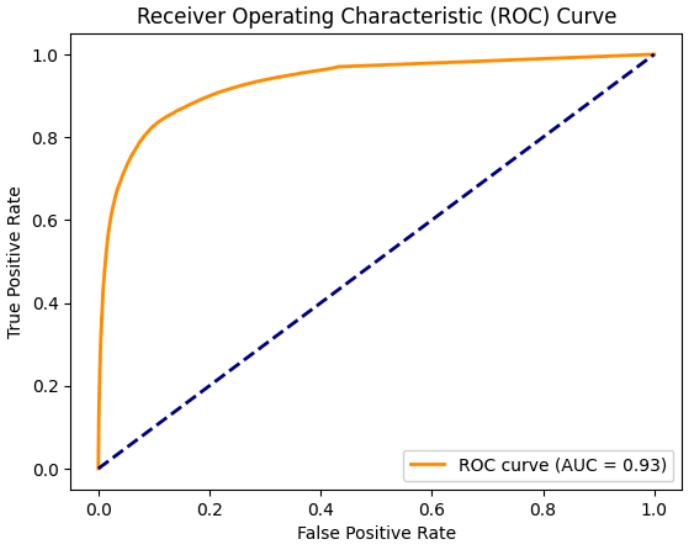
GCNConv layers were created in 2017 to incorporate both local attributes and node features into the graph embeddings. It was first used in knowledge graph and citation network classification tasks, but given its locally convolutional nature it was fit to be used for link prediction in egonets.

A year after the success of GCNConv, the SAGEConv (or GraphSAGE) layers were discovered. These are built upon the idea of incorporating information about the neighboring nodes when creating the node embeddings. It was first developed and tested for classifying nodes, however it can be utilized to predict links between nodes as well.

Since both of these layers are fit for message passing tasks.and we weren't completely sure which convolutional layer to choose, we added them as options for the hyperparameter optimization task which we will present later in the documentation.

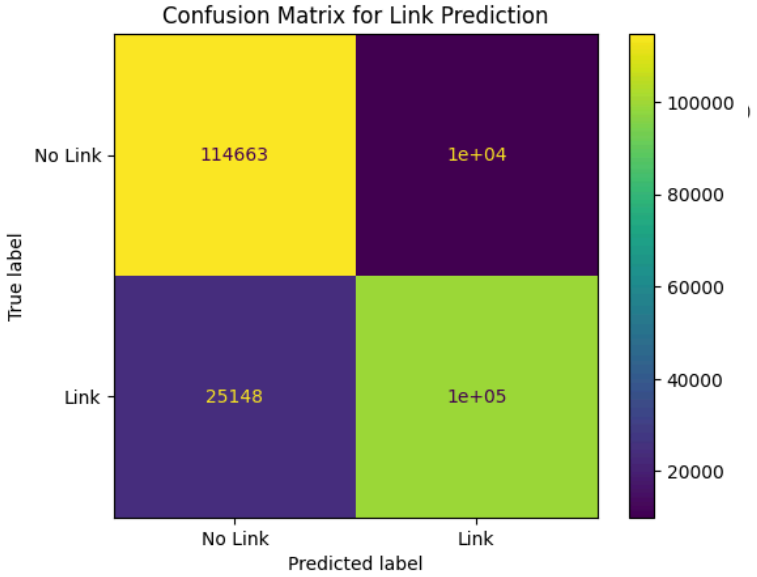
To search for the most optimal model, we employed a hyperparameter optimisation technique. We chose the ranges and possible values of the hidden dimensions, number of graph convolutional layers, type of graph convolution layers, and the dropout rate between the layers to find the best performing model configuration. We utilized random search to create hyperparameter combinations for the model, and we used Wandb's sweeps to manage all the training runs of the different configurations. We trained the model on a Facebook egonet to see and validate the link prediction, with a maximum of 20 epochs, but also applied early stopping for the AUC metric with setting the patience to 3.

We found that the most successful model was one that used 2 layers of GCNConv with a hidden dimensionality of 32 and a dropout rate of 0.3. This model was able to achieve an AUC score of 0.93,significantly higher than that of the baseline model. The performance improvement can be demonstrated using the ROC curve plot, which indicates a much more robust result compared to the baseline model. The GNN is able to gather true positives while eliminating false positives, as seen in the TPR-FPR dimensions of the plot.



ROC curve for the GNN model on Facebook

The confusion matrix is also useful to see that with the help of negative sampling the model can detect both positive and negative edges quite successfully. The model is great at determining negative links, with true negatives having a large amount of entries. The model also predicts positive links with success, only failing a small percentage of times. Nevertheless, our model is able to demonstrate a great ability to predict friend connections in the egonets of Facebook, significantly better than the baseline.



Confusion matrix for the GNN model on Facebook

Evaluation

Link prediction is basically a binary classification task, therefore, we used classification metrics to evaluate our models. For the baseline we chose the simple accuracy metric, while we utilized the average precision for the GNN model. However, for both of the models the main metric was the AUC (Area Under ROC) value.

We decided to use the AUC because it isn't prone to overfitting and can provide a robust way to look at the predictive capabilities of the models. Given that it both takes TPR (True Positive Rate) and FPR (False Positive Rate) into account it can leverage this extra knowledge to avoid favoring overfitting or underfitting solutions.

Also, we used the confusion matrix to get some insight into the nature of the models' predictions. It provides a clearer view into the performance of the binary classification necessary where one can also investigate whether the dataset is balanced or not.

Containerization and UI

As described in our homework task, we implemented further aspects of our solution, the two main being a Docker container for notebook hosting and a Gradio UI for interacting with the trained model.

We created a Docker container to provide a custom and flexible development and running environment for our project. It installs the necessary dependencies (like Pytorch and Pytorch Geometric) and sets up a local Jupyter server to host the necessary notebooks containing the EDA, baseline and GNN pipelines.

We also created a Gradio App inside the Jupyter notebook to enable interacting with the GNN model.

Conclusion

To conclude our project, it can be clearly demonstrated that the GNN performs better on the link prediction task than the baseline deep neural net. It achieved a significantly higher AUC score meaning that it's able to predict links much more efficiently, since it has the extra knowledge of the graph structure.

Our results show that it is indeed beneficial to use GNNs for predicting links in a graph, and it can be clearly seen that there is significant information to be gained from the neighboring nodes when working with deep learning.

LLM usage:

- *Data preparation: Generating basic structure and logic for some helper functions. A lot of correction and change was needed, to make the outputs as desired and correct the bugs*
- *Model definition: Help during debugging and in refining models.*