Deep learning (BMEVITMMA19)

Homework documentation

Team: Solo que

Students: Mihályi Balázs Márk (J8KAR3)

Date: 2023.12.10.

1. Introduction

Our homework project was "Friend recommendation with graph neural networks". The project's goal was to get familiar with the distinct methods used to represent nodes in a graph and use these representations to suggest meaningful connections based on user profiles and interactions. There were three datasets to choose from ([1],[2],[3]), and we ended up going with the Facebook dataset, since it didn't have directed edges in the social networks, making it easier to work with. It was also more manageable in size contrary to the Google+ dataset. After downloading the dataset, we prepared it for modelling. We looked at two baseline methods, node2vec [4], and spectral clustering [5]. We tried to beat these baselines with two other methods, a Graph Auto-Encoder (GAE) and a Variational Graph Auto-Encoder (VGAE) [6].
We trained all our models on the ego network of user 0, with a 70-15-15 train-validation-test split. The graphs contained real and false edges in a 1:1 ratio, the training graph having 3996, the validation graph 854, and the test graph 854 edges in total.
We will discuss the baseline and main methods in detail in Section 2, in Section 3 we will evaluate each method. Finally, in Section 4, we will draw our conclusions.

2. Modeling approaches

Node2Vec

This approach uses random walks to create meaningful node representations for edge prediction. Our training method used 10 random walks and relied heavily on the node2vec implementation in [7]. After the random walks, we used a Logistic Regression model to predict positive and negative edges.

Spectral Clustering

For this method, we used the scikit-learn implementation of spectral embedding, with the embeddings being 16 dimensional. For edge prediction, we used the dot product of two nodes' embedding and a sigmoid activation.

GAE

Our GAE implementation drew inspiration from the PyTorch Geometric tutorials available on YouTube. For encoding, we used two Graph convolutional layers, with 16, and 1 output channels. For the decoding, we simply used a dot product decoding method. We trained the GAE model for 100 epochs. Our optimizer of choice was an Adam optimizer, with a 0.01 learning rate, and for the loss, we used binary cross entropy.

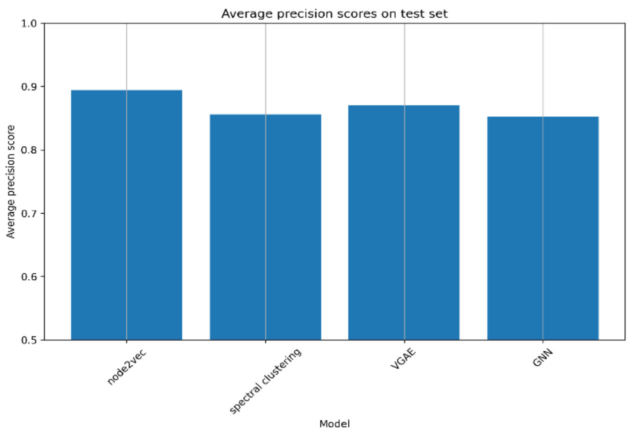
VGAE

Here we used PyTorch Geometric's VGAE implementation with a slight change to the testing function of the model: we needed it to return the confusion matrix as well. Apart from that we needed to define a Graph encoder class "VGraphEncoder", that returned the learnt variance's mean and logarithmic variance for every epoch. It had three convolutional layers apart from the mean and variance layers. During training, we used a VGraphEncoder with 64, and 8 hidden channels and 4 output channels, an AdamW optimizer with 0.01 learning rate and 0.0005 weight decay, and the combination of the reconstruction loss and kl-divergence as usual with standard Variational Auto-Encoders. The above hyperparameters were the result of a parameter grid search where the following values were used for each hyperparameter:

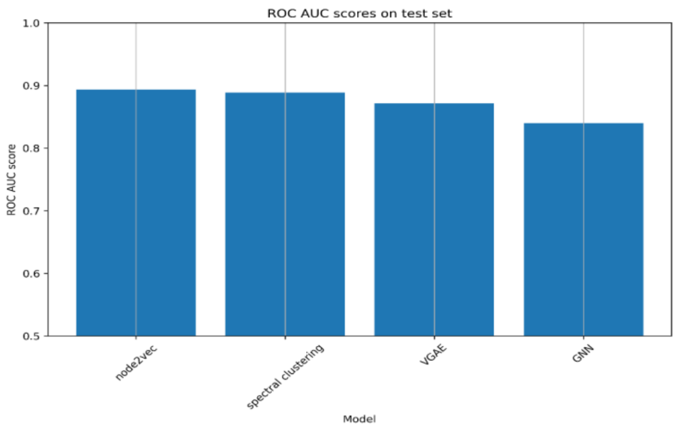
- hidden_channels1 : [8, 16, 32, 64]
- hidden_channels2 : [2, 4, 8, 16]
- learning rate : [0.005, 0.002, 0.001, 0.0008, 0.0005]
- weight decay : [0, 0.0005, 0.001, 0.005, 0.01]

3. Results

For evaluation, we used two metrics: Area Under the ROC Curve (ROC AUC), and Average

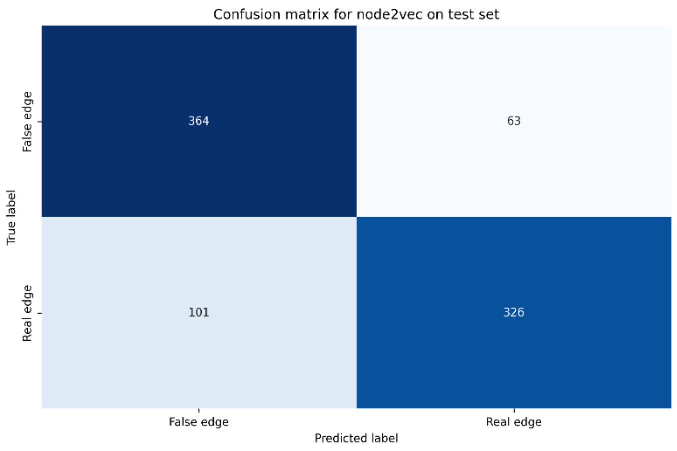


2. Figure - AP scores of the investigated methods

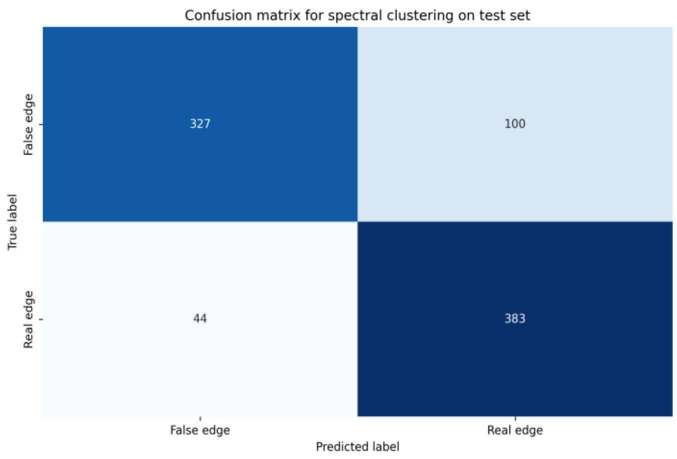


2. Figure - ROC AUC scores of the investigated methods

Precision (AP). Along with these we also looked at the confusion matrices of each method, to gain further insight into their performance. The ROC AUC scores and the AP scores can be seen in Figure 1., and Figure 2. The node2vec approach has the highest scores in both metrics. Our other baseline method, spectral clustering, has the second highest ROC AUC score, beating both of our main approaches. The GAE model (labelled GNN) in fact is the last in both metrics, whereas the VGAE model was able to beat spectral clustering in Average Precision. The bad performance of the GAE and VGAE methods could be explained by examining their confusion matrices. Figures 5. and 6. show these matrices, and the fact that despite being able to detect real edges more accurately than the baseline methods, they were barely better at detecting false edges, than a random guess. The two baselines were much better at this (Figures 3. and 4.), thus beating out the Auto-Encoder-based methods.



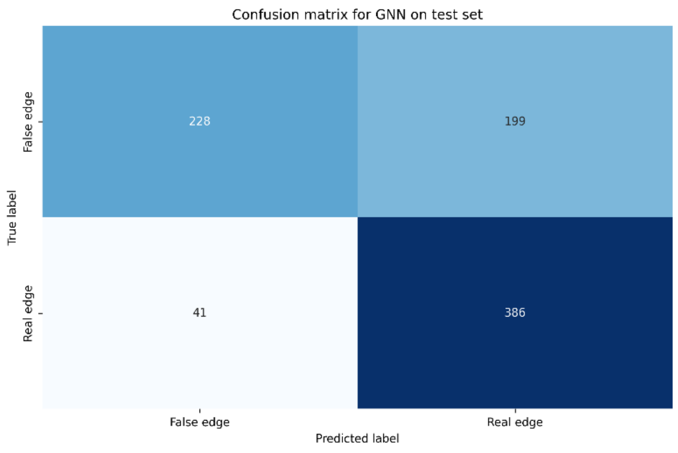
4. Figure - Confusion matrix of the node2vec method



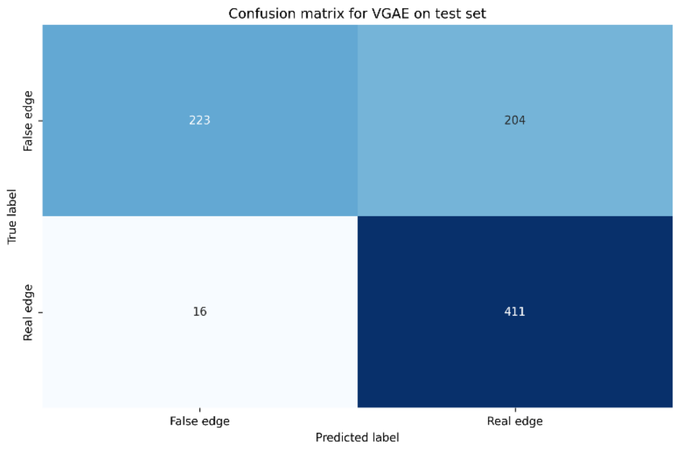
4. Figure - Confusion matrix of the spectral clustering method

4. Conclusions

While conducting our homework project of friend recommendations with Graph Neural Networks, a textbook link prediction task, we tried out four total methods. Our two baselines were node2vec and spectral embeddings, and our two main methods were Graph Auto-encoders and Variational Graph Auto-Encoders. Surprisingly, the Auto-Encoder-based methods could not beat out the baselines due to their inability to accurately detect false edges in the graph. We were unable to confirm whether this was a modelling error on our side (or the developers of PyTorch Geometric). The only thing close to a possible explanation that we could find was a pytorch forum post[1], where it was suggested, that a similar problem using a Graph Auto-Encoder was caused by the structure of node features. If so, it could be, that Auto-Encoder-based methods are not to be used in recommendation systems, as their ability to suggest meaningful new relationships (false edges, that could be real according to the model) is barely better than a random guess.



6. Figure - Confusion matrix of the GAE method



6. Figure - Confusion matrix of the VGAE method

5. Usage of LLM's

During our work, we used GitHub's Copilot, mainly for annotating the source code, and code completion. We also used OpenAI's ChatGPT (3.5). We tried using it for dataset creation and modelling, but it proved nearly useless, especially when asked about the pytorch_geometric library. This was probably because the library is fairly new and had multiple updates since the

[1] https://discuss.pytorch.org/t/imbalanced-positive-negative-edges-graph-link-prediction/84032

last knowledge update of the free version of ChatGPT. It did prove to be useful however when asked to help with visualization functions and sometimes debugging.

6. References

[1] Social circles: Facebook. J. McAuley and J. Leskovec. https://snap.stanford.edu/data/ego-Facebook.html, 2012. (last loaded: 2023.12.10)

[2] Social circles: Google+. J. McAuley and J. Leskovec. https://snap.stanford.edu/data/ego-Gplus.html, 2012. (last loaded: 2023.12.10)

[3] Social circles: Twitter. J. McAuley and J. Leskovec. https://snap.stanford.edu/data/ego-Twitter.html, 2012. (last loaded: 2023.12.10)

[4] Grover, Aditya, and Jure Leskovec. "node2vec: Scalable feature learning for networks." *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. 2016.

[5] Von Luxburg, Ulrike. "A tutorial on spectral clustering." *Statistics and computing* 17 (2007): 395-416.

[6] Kipf, Thomas N., and Max Welling. "Variational graph auto-encoders." *arXiv preprint arXiv:1611.07308* (2016).

[7] Lucas Hu, link-prediction, 2018, https://github.com/lucashu1/link-prediction (last loaded: 2023.12.10)