Introduction

Students were tasked with creating an algorithm for link prediction within an academic co-authorship graph. Given a training network, which is a fragment of the authorship graph, we then had to predict whether or not an edge exists between test node pairs. The undirected training graph G = (V,E) consists of vertices V, representing authors, and edges E, representing authors who have collaborated on a paper together. Neighbours of a node are nodes that are linked by an edge. The ROC-AUC score measures the model's performance where ROC is a probability curve and AUC is the area under this curve.

Approach Taken

I created a feature vector that represented a possible edge between any two given nodes with ten features. The engineered features were in one of two categories: structural features or node features. Structural features are used to gain insight into the structure of the graph, particularly the number of common neighbours they share. Node features were derived from the given nodes.json file to gain insight into characteristics of specific authors.

The structural features I used in my final model were Jaccard Coefficient, Adamic Adar Index, Maximum Triangle and Minimum Clustering Coefficient. The Jaccard Coefficient and Adamic Adar Index are measures of common neighbours for any two nodes. The triangle count was used to see how connected the neighbors of the two nodes are to one another. The clustering coefficient was also used to see how connected the neighbours are, but in relation to the number of all possible edges between neighbours.

The node features I used in my final model were number of shared keywords, number of shared venues, years since first publishing a paper, years since last publishing a paper, number of papers published, and years the author has been active. Shared features give insight into how similar the authors are to each other. The number of papers published gives insight into how prolific the author is. Years since publishing and years active shows the frequency the author works with others.

I then generated positive and negative edges in order to later train a model. I used all of the edges in the training and validation graphs as positive edges. For the negative edges, I generate all of the possible negative edges, choose the number n of desired negative edges, and then randomly choose these n edges from all possible negative edges. After selecting these n edges, I then go through the list of edges and prune off the edges that have a cosine similarity score greater than some value x. I use node2vec edge embeddings to calculate this cosine similarity score. One disadvantage of my model is that this method of negative edge generation is computationally expensive. After hyperparameter tuning, I set n=1.5 and x=.12 in order to achieve the best AUC score. These hyperparameters result in roughly the same amount of positive and negative edges, and negative edges that should not have any link between them.

Finally, I trained a Neural Network on the positive and negative edges. There were two layers, each with 64 neurons, in the neural network. There were 350 iterations performed when training the model, and the activation function used was logistic. Increasing the iterations led to overfitting. These hyperparameters allowed the model to be sufficiently trained without being computationally expensive.

Alternative Approaches

I took two other alternative approaches to my final model. Both of these approaches were structural representations of the network. The first approach was my baseline score of simply comparing the Jaccard Coefficients of the two nodes in a possible edge pair. The second approach was using several different edge embedding techniques to embed the edges in n dimensional space to compare similarity.

When generating a baseline score for my model, I wanted to approach the task in the simplest way. When looking through the networkx documentation, I found a number of built in for link prediction under the Link Prediction tab ("Link Prediction", 2020). Among these built in methods, using the Jaccard Coefficient seemed to be the best metric for a baseline score. This function takes two nodes as input, and then computes the number of nodes in the intersection of the given nodes over the number of nodes in the union of two nodes ("jaccard_coefficient", 2020).

When generating a model based on edge embeddings, I used node2vec to embed the edges. Node2vec uses random walks to see how often you visit the sink node from the source node (Grover and Leskovec, 2017). When initially generating my node2vec model, I used hyperparameters of 16 dimensions, walk lengths of 80, number of walks of 10, and equal p and q values. The p and q values either push the search outwards from the node, or encourage the search to stay local. Increasing the return value, p, means the transitional probability of returning to the node decreases. The opposite is true for increasing the q value. In terms of link prediction, getting a global view of the network is important, so I increased the p value to 5 in order to encourage the search algorithms to go farther away from the node (Goyal and Ferrera, 2018). This led to more accurate values in the edge embeddings, and the highest ROC-AUC score. After I trained the node2vec model, I took the dot Hadamard Product of the two embeddings and then gave these representative edge vectors to my classifier.

Both of these models did not score higher than my final model because there were not any node features included in the final vectors. The node2vec models performed better than the baseline model because the node2vec models had a rich understanding of how the network was globally structured, opposed to only having a local view of the graph. However, node2vec still scored worse than my final model when I added in the node features because it did not take into consideration how well connected the neighbors were. Additionally, embedding the edges in too high of dimensions led to an overly complex model that did not generalize well. The node2vec model was also computationally very expensive.

The baseline score comparing Jaccard Coefficients led to an AUC of 0.91669, the node2vec models led to a maximum AUC of 0.92020, and my final approach combining multiple common link prediction metrics and node features led to a maximum AUC of 0.94407 on the private data. This positive relationship between the scores and model complexity can be seen in Chart 1.

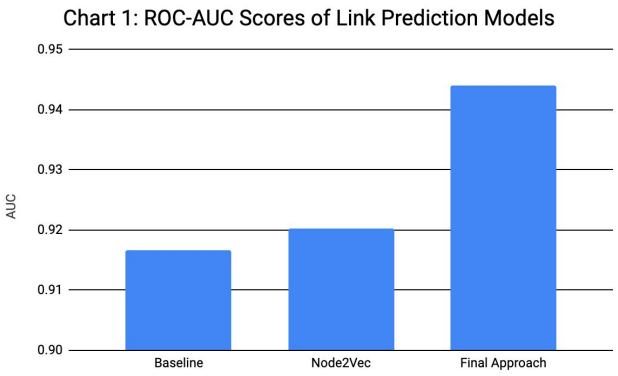

Chart 1.

References

Goyal, P., Ferrara, E.(1 July, 2018). Graph Embedding Techniques, Applications, and Performance: A Survey. *Knowledge Based Systems, 151*, 78-94 doi:10.1016/j.knosys.2018.03.022

Grover, A., Leskovec, J. (August 13, 2017). node2vec: Scalable Feature Learning for Networks. *ACM*. doi: http://dx.doi.org/10.1145/2939672.2939754

Jaccard_coefficient. (2020, April). Retrieved from https://networkx.github.io/documentation/networkx-1.10/reference/generated/networkx.algorithms.link_prediction.jaccard_coefficient.html#networkx.algorithms.link_prediction.jaccard_coefficient

Link Prediction. (2020, April). Retrieved from https://networkx.github.io/documentation/networkx-1.10/reference/algorithms.link_prediction.html