# COMP4332 Project 2: Link Prediction

DO Van Quyet, TRINH Van Hoan, NGUYEN Kha Nhat Long
The Hong Kong University of Science and Technology
{vqdo, vhtrinh, knlnguyen}@connect.ust.hk

## 1. Introduction

In this project, we will conduct Link Prediction, a classic task in Graph mining. The task is to predict whether two nodes (users) are linked (connected) by using existing data. This report will discuss our observations and analyses of the data set, our experiment models along with their details and explanations.

## 2. Graph Analysis

The training graph is directed, has 8343 nodes and 100000 edges, among them, 36087 edges have opposite counterparts and 27826 edges do not. We will perform analyses on the training data.

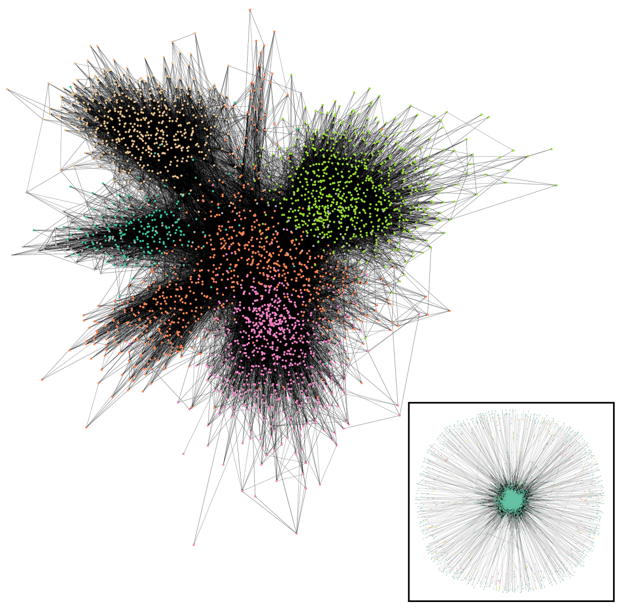

**Figure 1:** Community detection without all nodes with degrees less than 10. Bottom right: w/ full training data.

In Figure 1, we perform community detection on the training data using the Louvain method. It is noticeable that if the whole training data is used, there are outliers that form secluded groups, and nodes inside a community but have only a few edges connected to them. Therefore, when we drop all nodes with degrees less than 10, it generates a much clearer visualization. Note that after filtering, the graph still has 84643 edges (nearly 85% the original amount).

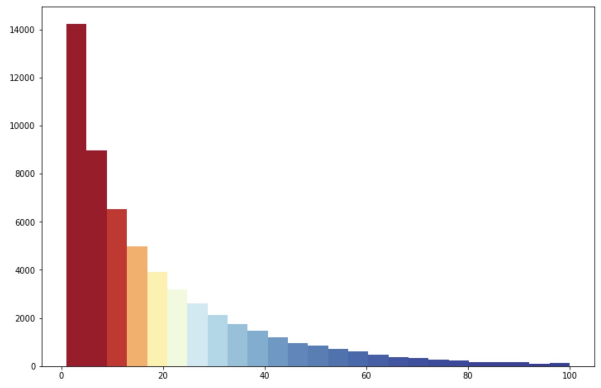

**Figure 2:** Common neighbors for each edge's head-tail pair

In Figure 2, approximately 21% of the edges (21259 out of 100000) connect 2 nodes that share fewer than 5 common neighbors further confirms that the network has many small and local groups.

In Figure 3, nearly 50% of the nodes (4176 out of 8343) has clustering coefficients less than 0.1 indicates that locally, a node's neighborhood seems to be slightly sparse.

Preliminary observations show that the training data is quite evenly distributed, has a significant amount of outliers, is mildly sparse, and is as expected from a social network.

## 3. Experiments

We incrementally built models to perform Link Prediction. The goal is to take two ordered nodes **u** and **v** as the input, and output the (relative [1]) probability that the directed edge (**u**,**v**) exists. We use the ROC-AUC score on the validation set, including provided positive samples and on-the-fly negative samples, to compare models.

[1]ranging from -1 to 1, instead of from 0 to 1

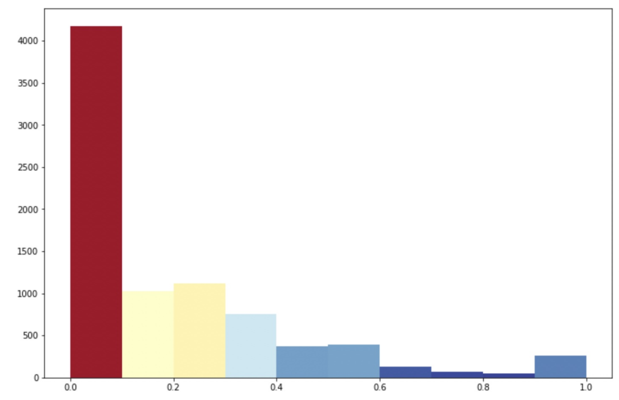

**Figure 3:** Histogram of Clustering coefficients for each node

### 3.1. Deepwalk and Node2Vec

We started with the baseline models: Deepwalk and Node2Vec. Both models are built from Word2Vec, which learns vector representation (i.e. embedding) for words by the Continuous Bag of Words (CBoW) or the Skipgram algorithms. In the context of graphs, to utilize Word2Vec pipeline, Deepwalk and Node2Vec rely on generating random walks over the graph to mimic sentences in Word2Vec. Overall, Node2Vec has a more complex strategy to sample random walks than DeepWalk, but all aim to produce similar representations for nodes that are close to each other, i.e., have a high similarity score.

We used grid search on different sets of hyperparameters of DeepWalk, including parameters of the random walks generator and the core model Word2Vec, to select the best configuration. Likewise, we tuned Node2Vec with different hyperparameters, especially p and q. Since a higher q value would let highly-connected nodes more frequently co-exist in a generated random walk, theoretically their embeddings in such settings are likely similar.

Overall, the obtained results show no improvement by tuning p and q, and the best configuration for DeepWalk is num_walks = 10, walk_length = 20, node_dim = 10, and the number of iterations for Word2Vec iter = 10.

### 3.2. Neural Networks

On top of the embeddings learned by the best DeepWalk, we stack neural network layers to transform these embeddings. In this context, to train neural network models, negative (i.e. non-exist) edges are sampled from the graph and combined with provided training edges to form the training set. As standard regulation of machine learning, we guarantee these edges are different from edges in validation and test set.

Now we describe our models.

**Multi-layer Perceptron (MLP)**: We simply feed the DeepWalk's embeddings through a simple MLP, then use newly obtained embeddings to calculate for each pair of nodes, the relative probability of having a directed edge. The default model has input_dim=10, hidden_dim=32, output_dim=16, and 1 hidden layer. To calculate the relative probabilities, we tried 3 different scorers: cosine similarity, l2-norm, and logit function (Figure 4).

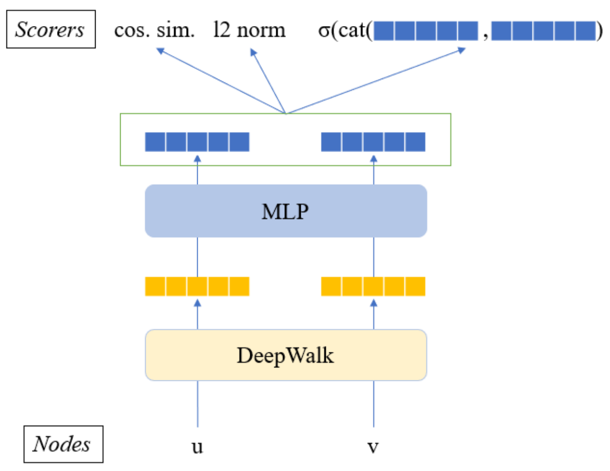

**Figure 4:** MLP Framework

As experiment results suggest, increasing the model size in breath (hidden_dim) or depth (num_hidden_layers), and replacing cosine similarity scorer by l2-norm or logit function would decrease the model performance. Thus, for later experiments, we fixed the scorer as cosine similarity. In addition, the model yields a slightly worse score than Node2Vec. It is expected, because theoretically, a unary function would not enrich the embedding spaces, thus not improving the prediction.

**GraphSAGE**: We apply the idea of GraphSAGE. For each node, we **SA**mple and aggre**G**at**E** messages from the neighbors to strengthen its representation. Theoretically, the algorithm will improve the link prediction, thanks to the fact that two nodes which have a shorter distance are more likely to form a connection. SAGE method would implicitly encode this information to node embeddings, as through SAGE-operations, embeddings of close nodes would become more similar.

We re-implement the method in a naive (and not-yet-efficient) way, by modifying the aforementioned MLP. For neighbor sampling and aggregation, we omit the sampling procedure for simplicity, while we fix the aggregation reducer as sum and try to pass messages in a different way. Since the graph is directed, we consider 5 options for mes-

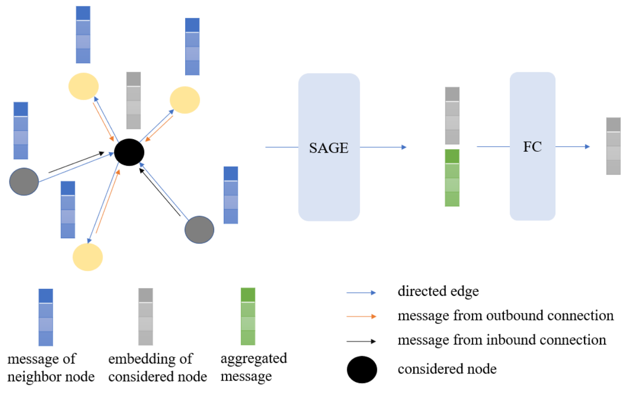
SAGE
FC
message of neighbor node
embedding of considered node
aggregated message
directed edge
message from outbound connection
message from inbound connection
considered node

**Figure 5:** SAGE algorithm

sage passing: for a pair of nodes (**u**,**v**), we aggregate message for source node **u** (target node **v**) from nodes having:
(1) outbound (outbound) connections from **u** (**v**)
(2) inbound (outbound) connections to (from) **u** (**v**)
(3) outbound (inbound) connections from (to) **u** (**v**)
(4) inbound (inbound) connections to **u** (**v**)
(5) in- or out-bound (so called 'all') connections with **u** (**v**)

We also study the effect of model size on the model performance, by changing the hidden_dim to 16 or num_hidden_layers to 1, while their default values are 32 and 0 respectively.

**Graph Attention Network (GAT)**: We further integrate the attention mechanism to GraphSAGE. In GraphSAGE, messages from neighbors are treated equally in the aggregation reducer. Under the attention mechanism, the proportion of these messages in the sum is controlled by the variables which represent to what extent the considered node should attend to (i.e take information from) its neighbors. We experiment with the default configuration as GraphSAGE and message passing of type (2).

All experimental results can be found in Table 1.

### 3.3. Results Summary and Analysis

The results show no improvement from vanilla MLP model, while GraphSAGE did significant improve the prediction as expected. Among the 4 types of single-direction message passing (from (1) to (4)), we observe that type (2) yields the best result. That hints the meaning of edges in the graph as following: for each edge (**u**, **v**), **u** is the followee, and **v** is the follower. The reason is, if the meaning is as described, the act of passing message to source **u** from its inbound connection and to target **v** from outbound connection represents the phenomenon that **v** follows a person **w**, **w** follows **u**, then **v** tends to find information about **u** and later follows **u** (so called follower-of-follower phenomenon).

Besides, by leveraging messages from both connection types, model of type (5) yields the best result while evaluating in the first 5 epochs. We further trained models of type (2) and (5) up to epoch 10, and observed no remarkable difference between the two models' performance, yet no remarkable improvement.

About GAT, despite having more degrees of freedom to fit on the train data, the model did not surpass its predecessor GraphSAGE. A possible reason is that with more parameters, GAT needs more data and epochs to be fully trained. In the future, we will address this problem.

| Models | Score |
|---|---|
| DeepWalk (best) | 0.9331 |
| Node2Vec (p=,q=1) | 0.9321 |
| MLP | 0.9278 |
| GraphSAGE | |
| (1) src ← out, tar ← out | 0.9546 |
| w/ no. hid. layer = 1 | 0.9529 |
| w/ hid. dim = 16 | 0.9535 |
| (2) src ← in , tar ← out | **0.9576** |
| (3) src ← out, tar ← in | 0.9558 |
| (4) src ← in , tar ← in | 0.9551 |
| (5) src ← all, tar ← all | **0.9580** |
| GAT | 0.9372 |

**Table 1:** Experimental results: ROC-AUC scores of all examined models on the validation set. The best scores of GraphSAGE and GAT are among the first 5 epochs, except for the setting type (1)+(w/ no. hid. layer = 1) first 10 epochs.

## 4. Conclusion

In this project, we analyzed the given graph dataset and gained useful insights about its characteristics. Based on that, we applied various models, from simple DeepWalk to more sophisticated GraphSAGE and GAT, to a Link Prediction task for the given graph. The result shows a significant improvement yielded from GraphSAGE. For the sake of interpretability, we selected the GraphSAGE model of type (2) to get predictions for the test data.