COMP4332 Project 2 Report

Group 28
HUI Man Wah, WONG Ho Leong, WONG Yu Ning, YIP Nga Yin

Introduction

This project aims to predict the link existence between users given their user ids. This report details the approaches and techniques, specifically **DeepWalk** and **Node2Vec**, employed to build a model for this link prediction task.

Dataset Description

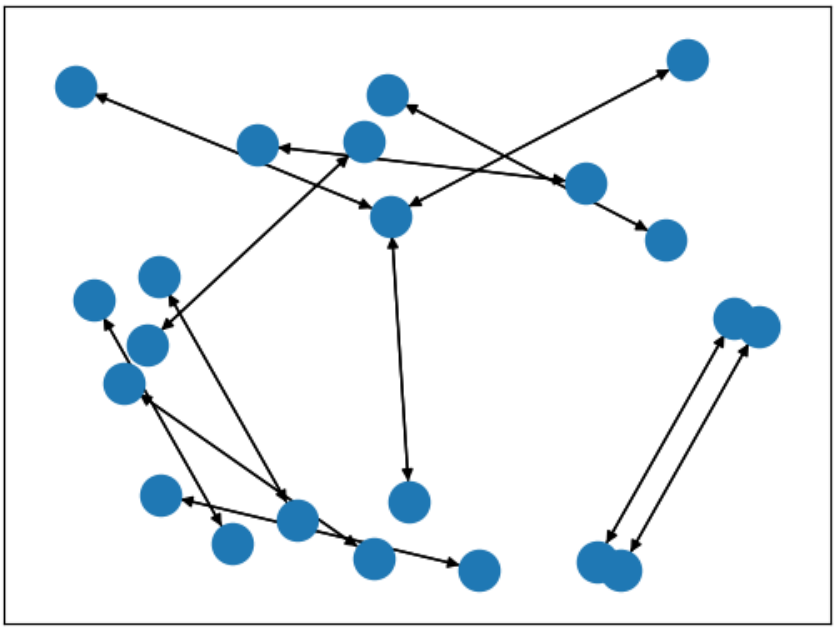
There are 100000, 20000 and 40000 data in the training, validation, and testing dataset respectively. In the training dataset, there are two attributes. The first column is the user id, and the second column is the friend list of the corresponding user. Each user can have more than one friend.

Validation set is also in similar structure. But we artificially added 20000 false edges into the validation set so that the model can be validate with false edges too. In the testing dataset, user ids are appearing in pairs. The pairs that are actually friends are labelled as 1 in the third column and 0 if otherwise. Only 50% of the user pairs in the test set are actually friends.

Exploratory Analysis

Training Dataset Analysis:

In total, there are 100000 edges in the training dataset. The average number of friends per user in training dataset is 1.37. This may indicate that the graph is relatively sparse. To confirm this hypothesis, we randomly sampled 10 nodes from the training graph and plot them and their neighbours out. Below is the plotted graph.



We can see that it is a rather sparse graph and each node has few neighbours. In this case, local connectivity is limited, we may need to focus more on the local information during model training. In other words, shorter walk lengths may result in more accurate predictions.

Model Training:

The project pipelines provided us with 2 different models, Deepwalk and Node2Vec. Node2Vec model often gives better results. To find the optimal parameters in a more efficient manner, we first used the deep walk model to find the optimal value for node dimension, number of walk and walk length respectively.

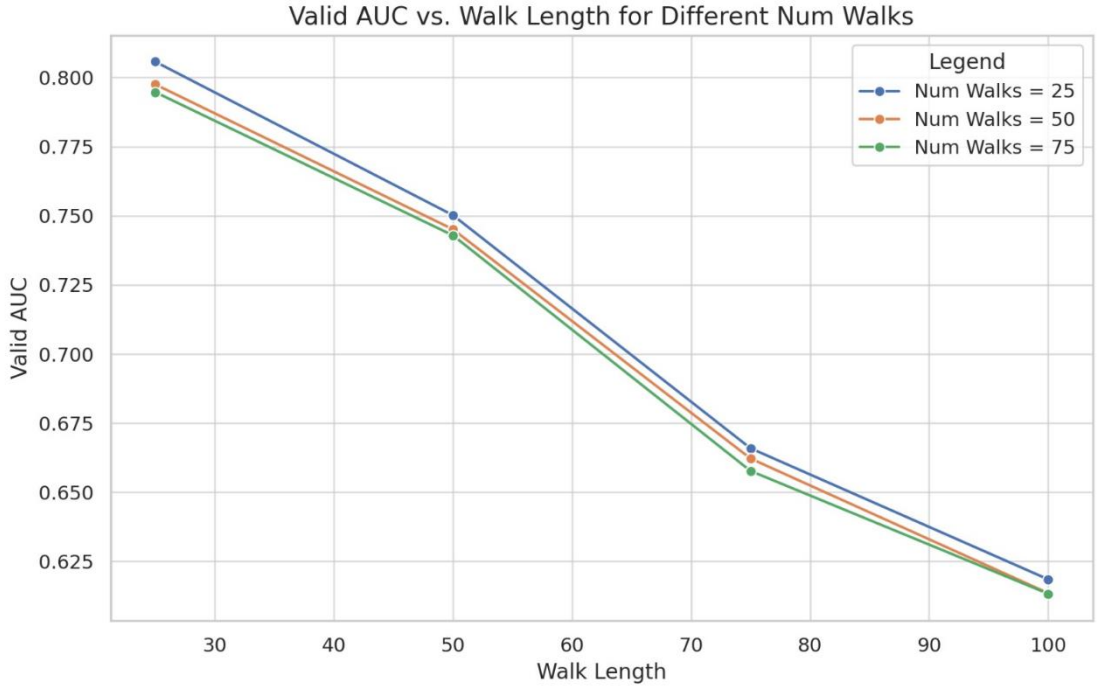
Deepwalk Model:

To find the hyperparameters that can result in the best accuracy within a short time, we try to locate the potential range of the optimal hyperparameters before performing more detailed search.

First, we set **p** and **q** equal to be 1 and optimized the other 3 parameters within the following region:

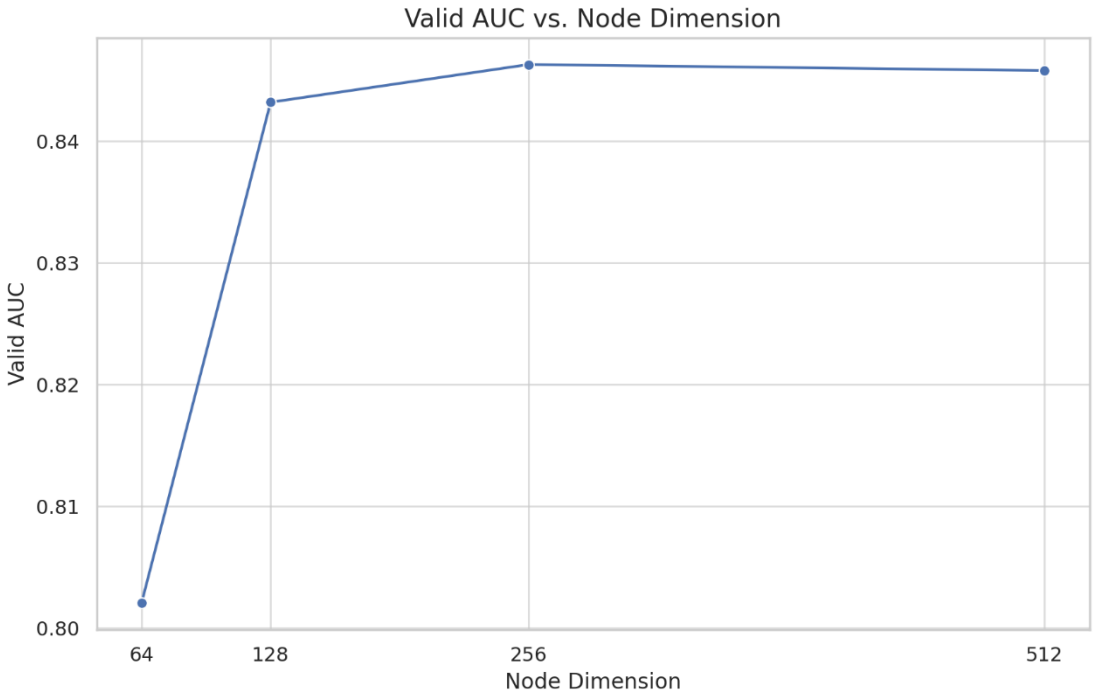
```
node_dim = np.array([64,128,256,512])
num_walks = np.arange(25,76,25)
walk_length = np.arange(25,76 ,25)
```

We found that as walk length increases, validation accuracy reduces. This aligns with our findings in the Training Dataset Analysis section and reflect that shorter walks should be used. We also find that the model achieved over 0.800 accuracy when only 25 walks are used for training. It might because increasing number of walk introduced more noise to the training data, causing the drop in model performance.



Based on our findings, we continued to carry out experiments to look for the optimal node dimension. We fixed both walk length and number of walks to 25 to reduce

model training time. The result is shown below.

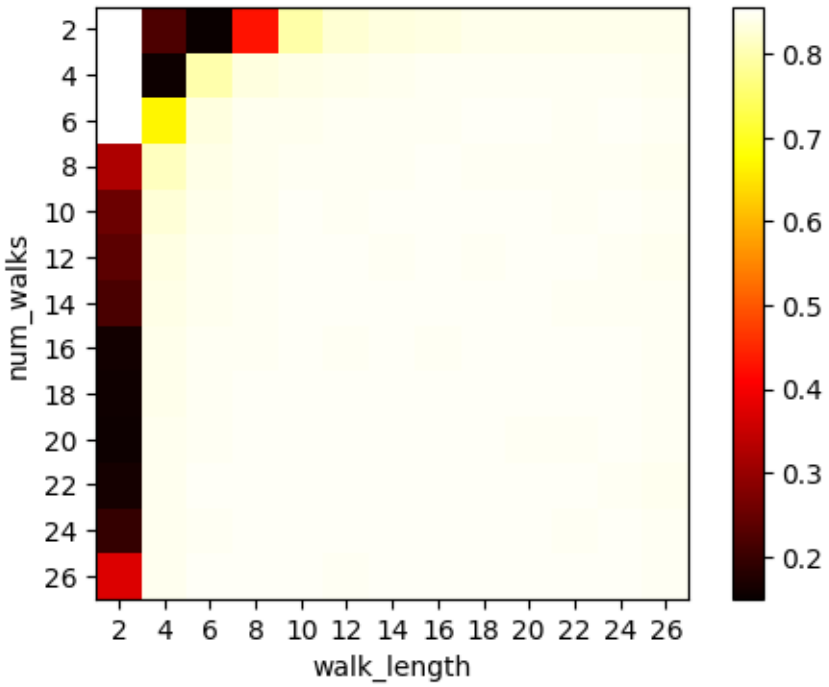


From the chart, we observed a significant jump in valid AUC performance when moving from a node dimension of 64 to 128 and reach highest AUC score when node dimension = 256.

Next, we performed more detailed hyperparameter search on number of walks and walk lengths. Since we observed a decrease in accuracy as number of walk and walk length decreases, we limited the values of these two variables to the following range:

```
num_walks =  np.arange(2,27,2)
walk_length = np.arange(2,27,2)
```

The following heatmap visualises the performance for each model on the validation datasets.



The best validation accuracy and the corresponding model parameters we found are shown below:

```
print("max accuracy = ",max(node2vec_auc_scores.values()))
#print(max(node2vec_auc_scores, key=node2vec_auc_scores.get))
res = [key for key in node2vec_auc_scores if node2vec_auc_scores[key] == max(node2vec_auc_scores.values())]
print(res)
```

✓ 0.0s

```
max accuracy =  0.8561542249999999
[(256, 4, 2, 1, 1, 'valid')]
```

After extensive searches to find the best parameters combination, we discovered that (node_dim=256, num_walks=4 and walk_length=2) gives the best testing accuracy(0.711). Details shown below:

```
#print(res[0])
(node_dim, num_walks, walk_length,p,q, num_window, num_epochs,_) = (256,4,2,1,1,3,10,"v")#res[0]
alias_nodes, alias_edges = preprocess_transition_probs(graph, p, q)
model = build_node2vec(graph, alias_nodes,alias_edges, node_dim, num_walks, walk_length,num_window,num_epochs) # TODO
scores = [get_cosine_sim(model, src, dst) for src, dst in test_edges]
write_pred("data/pred.csv", test_edges, scores)
```

✓ 9.3s

```
building a node2vec model...    number of walks: 291180 average walk length: 2.0000    training time: 7.0274
```

```
get_auc_score_test(model, test_edges, test_scores)
```

✓ 0.1s

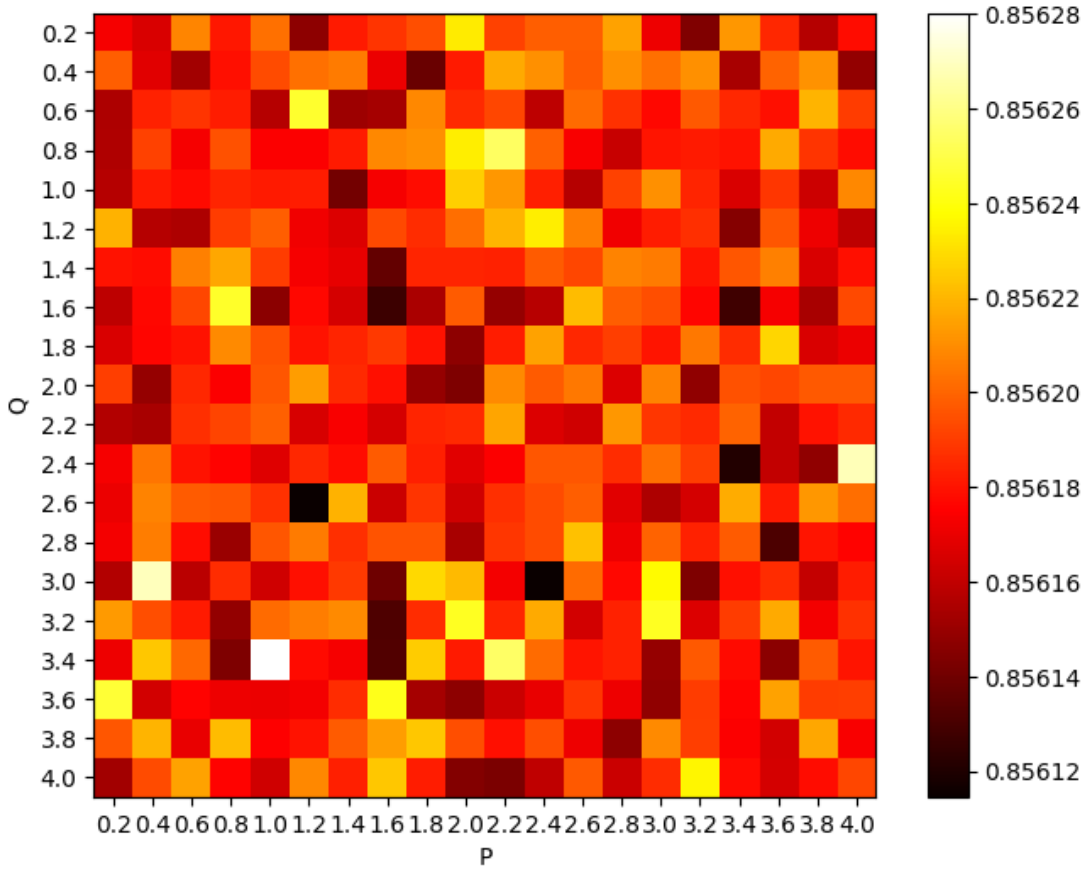
```
0.7113646375
```

Node2Vec Model:

After that, we started to look for the optimal value for p and q. We limit the value of p and q to the following range:

```
P = np.arange(0.2,4.2,0.2)
Q = np.arange(0.2,4.2,0.2)
```

However, no matter how we change the value of P and Q, the model accuracy does not have significant increase. The following heat map visualises the model performances when different p and q are used.



The code snippet below shows best validation accuracy and the corresponding model parameters:

```
print("max accuracy = ",max(node2vec_auc_scores.values()))
#print(max(node2vec_auc_scores, key=node2vec_auc_scores.get))
res = [key for key in node2vec_auc_scores if node2vec_auc_scores[key] == max(node2vec_auc_scores.values())]
print(res)
```

✓ 0.0s

```
max accuracy =  0.8562558899999999
[(256, 4, 2, 2.4000000000000004, 0.2, 'valid')]
```

Our final testing accuracy does not show improvement compared to the Deepwalk model.

```
alias_nodes, alias_edges = preprocess_transition_probs(graph, p=2.4, q=0.2)
model = build_node2vec(graph, alias_nodes,alias_edges, node_dim=256, num_walks=4, walk_length=2)  # TODO
scores = [get_cosine_sim(model, src, dst) for src, dst in test_edges]
write_pred("data/pred.csv", test_edges, scores)
```

✓ 9.6s

```
building a node2vec model...    number of walks: 291180 average walk length: 2.0000    training time: 6.9133
```

```
get_auc_score_test(model, test_edges, test_scores)
```

✓ 0.1s

```
0.71135306125
```

Explore other parameters:

We tried to improve our model by changing the window size and number of epochs. Based on our prior knowledge, we performed grid search using following ranges of values. Since the value of p and q does not affect much on the accuracy result, we use the optimal value obtained in previous experiments.

node_dim = [8,16,32,64,128,256]

num_walks = [1,2,3,4,5]

walk_length = [1,2,3,4,5]

num_window = np.arange(5,21,5)

num_epochs = np.arange(5,21,5)

P = [2.4]

Q = [0.2]

After searching, the optimal model has slightly improved. It obtained 0.858 validation AUC score and 0.712 testing AUC score.

```
#print(res[0])
#(node_dim, num_walks, walk_length,p,q, num_window, num_epochs,_) = res[0]
(node_dim, num_walks, walk_length,p,q, num_window, num_epochs,_) = (256, 5, 3, 2.4, 0.2, 20, 5, 'valid')
alias_nodes, alias_edges = preprocess_transition_probs(graph, p, q)
model = build_node2vec(graph, alias_nodes,alias_edges, node_dim, num_walks, walk_length,num_window,num_epochs) # TODO
scores = [get_cosine_sim(model, src, dst) for src, dst in test_edges]
write_pred("data/pred.csv", test_edges, scores)
```

✓ 10.6s

```
building a node2vec model...    number of walks: 363975 average walk length: 3.0000    training time: 8.1143
```

```
get_auc_score_test(model, test_edges, test_scores)
```

✓ 0.1s

```
0.712061105
```

Conclusion:

In conclusion, our experiments delved into modern graph embedding techniques (DeepWalk and Node2vec). We achieved an AUC score of 0.858 in the validation set and 0.712 in testing dataset.