COMP4332 Project 2 Report - Link Prediction

Group 14

1 Introduction

Social media is inextricable from our daily lives. The ubiquity of social media triggers the study of the interactions and relationships of its entities, which is called social network analysis. From a network, we can observe the properties and behaviour of its structure, roles and positions of the actors involved, and the communities formed. There are many useful applications of social network in machine learning, such as predicting actor's behaviour to deliver the right ads, or monitoring certain communities for security matters.

In this opportunity, we are going to conduct link prediction modeling, a classic machine learning task in graph mining and processing. The task is to predict whether two nodes have a link between them using data from an existing structure. This report will discuss about our experiments, comprising of observation of data set, model of our training environment, the prediction result obtained, and type of metrics used to evaluate it.

2 Data Observation

The data provided for training the model consists of 29,464 graph edges separated by a newline character. Each edge is represented by two integers corresponding to the head node number and tail node number respectively.

The validation data has a similar structure to the training data. The validation data is given in a comma-separated values (CSV) file containing 7,367 lines. The first line of this file represents the type of data contained. The following lines contain four values that store different data. The first value is the index number and can be ignored. The second and forth value are the head node number and tail node number respectively. The third value is a Boolean value which serves as the label of this edge, indicating its existence in the graph.

The last data file is provided for the prediction phase. Similar to the training and validation data, this file contains a list of edges represented by its head node number and tail node number. Note that in the validation and test data file, a line does not necessarily indicate existence of that edge in the graph. Furthermore, it is entirely possible that there are nodes in the test and dev set that were not encountered in the training set. When running the model in the prediction phase, the model has to predict the probability score of each edge given its head node number and tail node number.

3 Training Environment

Our early experiments began with a small number of model parameters that can be executed by most laptops. However, as we started to grid search with different combinations of model parameters, we decided to do our experiments using Intel's DevCloud service. This environment provides free high quality CPUs with up to 48GB of RAM. Since the core python library for this project, NetworkX, does not support GPU acceleration, Intel's DevCloud service provides a better training environment than Google Colab for this project.

4 Experiments and Results

We first tried different **p** and **q** parameters in the Node2Vec algorithm provided in the skeleten code. Bigger **p** will help the model learn the graph in more Depth-First Search (DFS) oriented walks while bigger **q** helps the model learn in more Breadth-First Search (BFS) oriented walks. Intuitively, it may be better to set **q** to be bigger than **p**, making the model BFS oriented, because the window size of each node will be smaller during model training. This helps the model focus on the smaller neighbourhood circle of each node and lead to a decisive model on edge cases.

In addition, after simulating the random walks in the Node2Vec network, we fed these generated paths into the Word2Vec model by treating the paths as sentences and the nodes as words. This model is usually used in natural language processing to find the embedding values of words in sentences. However, we are able to obtain the values for the corresponding nodes, instead of word embeddings, with this approach.

By tuning **p** to be 1 and **q** to be 40, we achieved an Area Under Curve (AUC) score of 0.88. Although this approach seemed promising, we had a much higher target score. We then decided to utilize the free Intel DevCloud service to grid search the hyper parameters.

Intel DevCloud service gives us the convenience to grid search the model parameters by allowing us to submit jobs of Python task. Each Python task utilizes the ParameterGrid library provided by *scikit-learn* to grid search using a subset of possible parameter combinations. Different values for each parameter that were experimented on are shown in Table 1. Based on our experiments, when **p** value is big, increasing both the **walk_length** and **window_size** values tend to also increase the AUC score proportionally. This is correlated with the approach of our model that is more towards the Depth-First Search.

Parameters	Values
dimensions	100, 300, 600
iterations	10, 20, 30
num_walks	15, 35, 45
num_workers	100, 300, 2000
p	1, 10, 20, 40, 50, 60, 80, 100
q	1, 10, 20, 40, 50, 60, 80, 100
walk_length	3, 6, 8, 15, 35, 50, 100, 200
window_size	3, 6, 8, 15, 35, 50, 100, 200

Table 1: Parameter Values in Experiment Phase

4.1 Final Hyperparameters

After further experimentation, the final hyperparameters used are given in Table 2. It is observed that the **DFS** approach yields better result than the **BFS** approach. With large **window_size**, **walk_length**, as well as the **p** parameter, the AUC score of the model with **DFS** approach is better than the **BFS**.

Therefore, the model having **walk_length** and **window_size** of 200, and **p** of 50 is chosen. Meanwhile, **dimension** of size 300 seems to have the best representations on mapping the graph to the specified number of dimensions because increasing the dimension further does not change the AUC score much. In addition, having 30 **iterations** appears to provide the model enough iterations to converge. This is observed through the AUC score, which does not change much when the number of iterations were increased.

As **number of workers** increases, the speed of the training model increases as well when compared to a model with the same parameters. This is due to the number of agent used to perform the random walk when generating the paths into Word2Vec model. The computing power increases as well along with the increase on number of workers.

Parameters	Value
dimensions	300
iterations	30
num_walks	45
num_workers	2000
p	50
q	1
walk_length	200
window_size	200
auc_score	0.958702

Table 2: Final Hyperparameters Used

5 Conclusion

In this project, we have successfully trained a deep learning model which is able to predict existence of edges. Using node embedding and Node2Vec algorithm, we trained the model using **DFS** approach to predict the presence of links or relationships given different edges. As a result, our model with the parameters on Table 2 had been able to achieve 95.87% on AUC score. It is observed from our experiments that **DFS** approach yields better result compared to **BFS** approach on our dataset. In addition, having large window size, number of workers, and walk length requires expensive computational power as the current libraries do not support any hardware acceleration yet (i.e. GPU or TPU).