COMP4332 / RMBI 4310 Project 2 Report (Social Network Mining)

Team Members: SETHI Aryan (20634962), CHOW Hau Cheung Jasper (20589533), KAUSHAL Kaustubh (20634039), PAREKH Yashasvi Gopalbahi (20634089)

Team Name: AKJY **Group ID:** 05

Introduction

With social media becoming more and more prominent in everyone's daily lives, understanding the interactions and relationships between users has been vital for the profitability of large tech companies. The study of these interactions between users on social media platforms is often termed as 'social network analysis,' whereby the traits and behavioural patterns of its structure and users involved can be observed. Social network mining techniques can be used to determine what ads the platform should target to a certain user, or predict the chance of two users knowing each other based on certain features (e.g. number of mutual friends).

For this project, link prediction modelling will be conducted, which is a classic learning task in machine learning for graph mining and processing. Specifically, link prediction modelling will be used to predict whether a link between two nodes exists using existing data.

Preprocessing and Exploratory Data Analysis

The provided training dataset contains 8343 users (nodes), with 100000 **directed** edges in the training set. The dataset consists of two columns "user_id" and "friends". The first column in each row refers to a particular user/node and the second column refers to all the nodes where there is an edge going from that user to them. The validation set is similarly structured, containing 5457 nodes and 19267 edges. The test set is a list of 40000 node pairs (src, dst), which involve a total of 52425 nodes. Out of the 40000 edges, 19268 are positive edges, while the rest are negative. A positive edge means that the node pair (src, dst) implies there exists an edge from src to dst, while a negative edge implies the reverse. The goal of link prediction is: given a source user **src** and a destination user **dst,** compute the probability $P_{(src,dst)} \in [0, 1]$ there is a link from **src** to **dst.** It should be noted that this relation is one-way, so if src = user1 and dst = user2, the score between these two users may not necessarily be the same as src = user2 and dst = user1.

Due to the structure of the data provided in the training and validation set, the only data preprocessing required is extracting the edges of the graph as a list. An edge is denoted by a node pair (u, v). However, as the loaded graph from the data only consists of positive edges (i.e. node pairs where there is a link), we will need to randomly create some 'false/negative' edges whereby two nodes do not have a link between them. We choose to generate roughly ~20000 negative edges so that the positive and negative edges are class-balanced. To facilitate our analysis, various modelling techniques such as DeepWalk, Node2Vec, and ensemble methods will be used. During data modelling, we will conduct hyperparameter tuning to further finetune the model's performance. For each model, we choose the best hyperparameters based on the AUC score of the model on the validation set, and we illustrate the difference in performance between different hyperparameter selections via heatmap analysis.

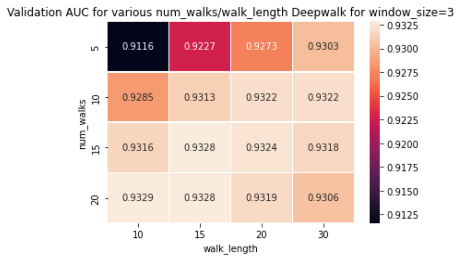
Data Modelling

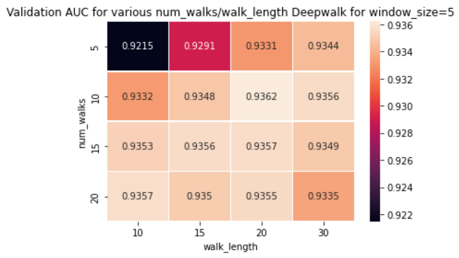
In order to predict whether a link exists between two nodes or before, we first need to define how we quantify the information in a node, and how we measure similarity between two nodes. Node information is stored in an *embedding*, and nodes should be considered similar if they are connected and have similar neighbourhood structures. Therefore, similar nodes must have node embeddings whose inner product most closely resembles the weighted adjacency matrix A. However, the weighted adjacency matrix of a graph $G = (V, E)$ is of size $O(|V^2|)$ and many graphs (especially those used in social network mining) are sparse, meaning the number of edges $|E| << |V|^2$.As computing the weighted adjacency matrix is not very scalable, we use the mathematical concept of random walk, which serves as the foundation for DeepWalk and Node2Vec. Random walk is a process which involves repeatedly sampling chains (or paths) which each consist of a random sequence of nodes. Utilising ideas from language modelling, we can compute the similarity of two nodes **u** and **v** based on how often **u** appears in the k-hop neighbourhood of **v** (and vice versa). We can then optimise the embeddings such that $sim(u, v) \approx z_u^T z_v$ where z_u, z_v are the embeddings of u and v respectively.

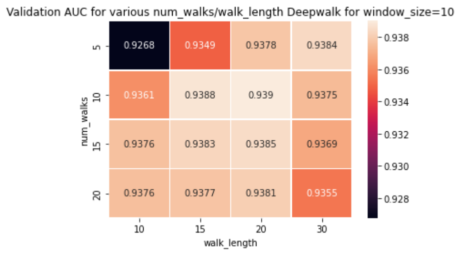
DeepWalk

The purpose of DeepWalk is to estimate the probability of visiting node v on a random walk from node u on a first-order random walk (i.e. the transition probability depends only on our current position). Based on the co-occurrences of the node pairs that appeared over all the random walks, a prediction about the link classification can be made. The hyperparameters to consider for DeepWalk are the window size **k** (which dictates the size of the neighbourhood to consider when determining co-occurrence at each step of the random walk), the number of walks and the walk length. The node embedding dimension was also a hyperparameter, but we chose to fix it at 10. To tune the hyperparameters, we looked at the AUC score on the validation set based on the following hyperparameters:

Parameters	Values
Node dimensions	10
Number of walks	5, 10, 15, 20
Walk length	10, 15, 20, 30
Window size	3, 5, 10







COMP4332 / RMBI 4310 Project 2 Report (Social Network Mining)

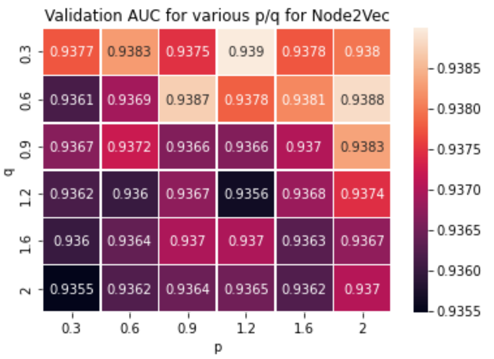
Team Members: SETHI Aryan (20634962), CHOW Hau Cheung Jasper (20589533), KAUSHAL Kaustubh (20634039), PAREKH Yashasvi Gopalbahi (20634089)

Team Name: AKJY **Group ID:** 05

As seen in the above three heatmaps, the validation AUC tended to increase when the window size increased, suggesting that it is beneficial to consider a larger ho p neighbourhood structure when constructing node embeddings. Across all three window size values, it can be shown that performance decreases significantly when walk length and number of random walks is low, and the sweet spot seems to be with num_walks = 10 or 15 and walk_length = 15 or 20. As walk_length increases to 30 and num_walks increases to 20, the validation AUC suffers slightly. We found that the highest AUC score of 0.939 was achieved when the window size was the largest value (10), the number of walks was 10, and the walk length was 20. Since p and q are not used in 1st-order random walks, we just set them to the defaults of p = 1.0 and q = 1.0.

Node2Vec

The difference between DeepWalk and Node2Vec is that Node2Vec utilises biassed second-order random walks (where the transition probability to the next node depends on both the previous node and the current node). In Node2Vec, the additional hyperparameters p and q are used to control the random walk's strategy. Lower values of q (q<1) makes the random walk similar to DFS, while lower values of p more approximate BFS since the random walk is likely to backtrack to already visited nodes. When testing the best p and q values, use the same hyperparameters as the DeepWalk for the node dimension, number of walks, walk length, and window size. These are the different p and q values we tested:



Parameter	Value
p	0.3, 0.6, 0.9, 1.2, 1.6, 2
q	0.3, 0.6, 0.9, 1.2, 1.6, 2

As seen from the heatmap, the highest AUC score of 0.939 is achieved when p = 1.2 and q = 0.3. Since p is high and q is low, this suggests that our random walk strategy is more similar to DFS than BFS. In general, we notice that large values of q seem to yield worse performance than smaller values of q, likely because it means the 2nd order random walk is less DFS and more BFS-like (i.e. the random walk is less likely to visit nodes further away from the current node).

Ensembling

We use an ensemble of first-order (DeepWalk) and 2nd-order (Node2Vec) random walk approaches and use two copies of each. Then we take the average of the four models to get our final prediction, which has reduced variance.

Parameter	Deepwalk Value	Node2Vec Value
final_node_dim	10	10
final_num_walks	10	10
final_walk_length	20	20
final_p	1.0	1.2
final_q	1.0	0.3
finalw_size	10	10

The table above also shows our final parameters we used in our DeepWalk and Node2Vec models respectively. Ensembling by averaging the results yielded an AUC score of 0.939 on the validation set.

Conclusion

In conclusion, we were able to successfully train a deep-learning model which accurately predicts the existence of edges. Through ensembling 2 copies of both Deepwalk and Node2vec, we trained the random walk to adopt a DFS-like approach and as a result, our parameters as displayed in the table above has the accuracy score of 93.9% AUC, a significant improvement over the baseline of 92.9%. It was also observed that the DFS approach gives better results than the BFS approach, given that: the random walks are sufficiently long; enough random walks are conducted; and the window size for computing co-occurrence of nodes is large. Further work could be done to improve performance, such as by implementing graph convolutional networks to aggregate information from their k-hop neighbourhood.