# *Report (Project Two)*

Student Name: Yupeng Wu
Student ID: 45960600
Student Email: yupeng.wu@uqconnect.edu.au

## *Task*

**Katz method:**

This method ranks the prediction according to the Katz score. The math formulation is as followed:

$$score(x, y) = \sum_{l=1} \beta^l (A^l)_{xy}, A \text{ is the adjacency matrix and } \beta \in (0,1)$$

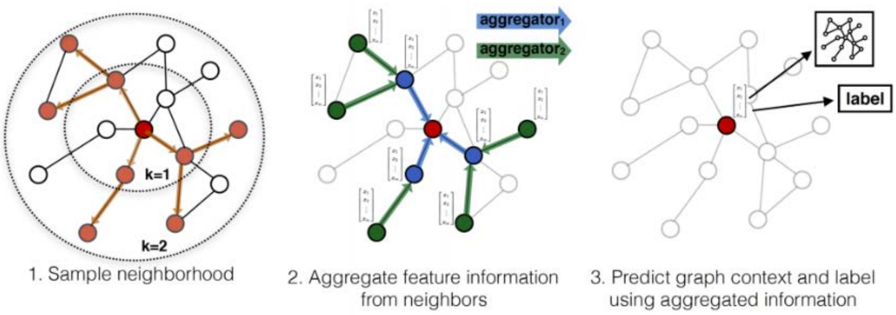
First we generate a graph by the training data. Then we can get the Katz score with a given source and target in the validation set. Finally, compute the Katz score with the test data and output the top 100 pairs as a prediction.

Here I got 85% accuracy by computing the cosine similarity with $\beta = 0.5$ and $l = 1$.

**GraphSAGE method:**

The graph sample and aggregate model generates node embeddings on the fly. It trains an aggregator instead of training specific node embeddings in order to handle unseen nodes.

1. Pick one node. Pick some eighbor nodes in each layer.
2. Aggregate nodes in each layer.
3. Feed embeddings to the neural network and predict it.




GraphSAGE sample and aggregate approach (Hamilton et al., 2018)

**Deep walk method:**

In this method I tried to generate two different random walks: biased random walk and BFS&DFS mixed walk.

Generate different walks by different random walk methods in order to train a skip-gram model. We can use the word2vec function in the model and get a vector which represents a node in the network.

**Biased random walk (biased RW):**

Unlike randomly choose a neighbor of a node, the probabilities of the potential new nodes are unequal. Here I took the betweenness centrality of node $i$ as the probability of jumping to node $i$. In other words it will be more likely to jump to a neighbor with higher betweenness centrality, which is defined as below:

$$B(i) = \frac{Total\ number\ of\ shortest\ paths\ through\ node\ i}{Total\ number\ if\ shortest\ paths}$$

**BFS&DFS mixed walk:**

Instead of walking randomly, here we use Breadth-Fast-Sampling (BFS) to reach immediate neighbors while Depth-First-Sampling (DFS) prefers node away from the source.

## ***Summary***

| Method | Katz | Biased RW | BFS&DFS | GraphSAGE |
|---|---|---|---|---|
| Predicted Acc | 85% | ≈ 45% | ≈ 60% | 50.25% |
| Real Acc | 1% | 19% | 28% | 36.10% |

(The simplest random walk method achieves 30% of ground truth (real) accuracy)

For the four methods I've introduced above, the Katz method has the best performance on the validation data, which have 85% of accuracy. This is a very different finding because I thought GraphSAGE and deep walk will be a better method since they are more complicated. This reminds us that the advanced method is not always a good method to a project and menmorized method can also do a good job on small size data.

For the biased random walk method, I choose the 'network.betweenness_centrality' function to compute every nodes' centrality in the graph, which causes the code to be tremendous slow. The output of this method is also not satisfied so this might be a very bad method to this task.

However, the accuracy of the GraphSAGE is very low so that I do not use it for prediction. The lack of the features in the data might be one of the reasons and I just simply assumed that all the nodes have a same feature. The batch size, network depth and other parameters have been changed for a higher accuracy, but the result is still not good. I believe that this model has a great potential improvement space but the time is up. So, this task remains to me in the future to concatenate the embedding of each layer and feed them to fully-connected MLP to do the prediction.

Finally it is a pity to me that I failed in using GCN to do the link prediction. I challenged myself and tried my best to build a GCN for training and testing. But I keep getting an accuracy of zero as a result. This will also be my goal to complete this task by GCN. Currently my GCN always give me a meaningless output and I believe I can finish it in the future.

Edit: (2020.7.7)

After having the ground truth:

The table above shows the ground truth accuracy. And obviously the GraphSAGE method has a great advantage. The reason why I did not handle the result from the GraphSAGE is because I do not know how to extract the predicted link from the stellargraph.LinkSequence ☹. Also I doubt that the way I use the GraphSAGE model, the loss and acc do not changed after training model, which is strange. Maybe it's because of the lack of feature vector in the links.

```
Train Set Metrics of the initial (untrained) model:
	loss: 4.7745
	acc: 0.6869

Test Set Metrics of the initial (untrained) model:
	loss: 9.7441
	acc: 0.3610
```

```
Train Set Metrics of the trained model:
	loss: 4.7745
	acc: 0.6869

Test Set Metrics of the trained model:
	loss: 9.7441
	acc: 0.3610
```

# *Reference*

[1]. Random Walk in Node Embeddings (DeepWalk, node2vec, LINE, and GraphSAGE), Edward Ma, https://medium.com/towards-artificial-intelligence/random-walk-in-node-embeddings-deepwalk-node2vec-line-and-graphsage-ca23df60e493
[2]. https://github.com/pranavkulkarni/Link_prediction_social_network/blob/a1928c162450c93ada0d2dec80aa4b5bfb341e2e/link_prediction.py#L64
[3]. https://stellargraph.readthedocs.io/en/stable/demos/link-prediction/index.html#find-algorithms-and-demos-for-a-graph
[4]. Tutorials in INFS7450 by Junliang Yu.