Comp 4332 Project 2

Wu Chi hsuan (20657213), Wu Tai Ling (20561658), Lee Hao Yu (20656013)

Introduction

In this project, we tried two approaches: tuning the hyper-parameters for the first/second random walk model and building our own prediction model using GraphSage.

First/Second Order Random Walk

For both 1st and 2nd order random walk, after tuning hyper-parameters and Word2Vec window size like grid-search method, we found that *node_dim* = 15, *num_walks* = 12 gave us the best result with validation AUC over 94% and almost 95% for 1st and 2nd order random walk respectively. The longer *walk_length* provided better performance in general but the training time would also be longer (see Appendix).

For 2nd random walk, larger *p* and smaller *q* gave us better results with its essentially "outward" progression trend. Finally, the validation performance was in general increasing with respect to the *window* size, but it varied between first and second order random walks. We found out that the improvement of second order random walk performance due to larger window size is more significant and sensitive to the first order one. The performance of different *p*, *q* is compared through Heatmap is in the Appendix.

GraphSage

From our observation on NodetoVec, the more unique paths that random walks can cover, the better performance the model can make. This insight triggers us to use the GNN based method, which generates embeddings by sampling and aggregating features from a node's local neighborhood.

We implemented 3 layers of GraphSage layers to generate node embeddings of length 10, and conduct link prediction by feeding the element-wise multiplication of the two embeddings to MLP. To train the model, 100000 false edges are generated to pair with the training edges, and the model is trained to predict whether a link is true or false.

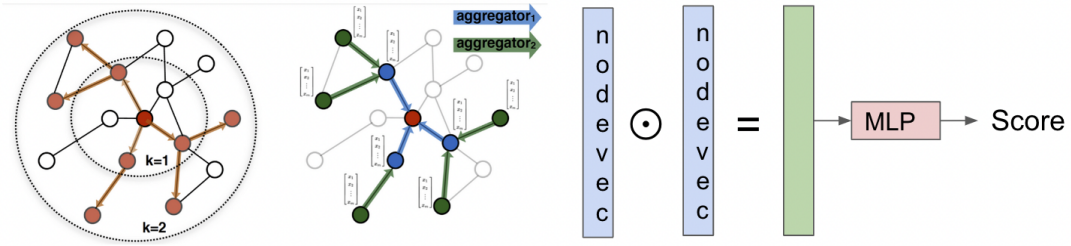

Figure 1. Structure of using GraphSage

Experiment Result

Below table shows the best results on validation set for each model:

	DeepWalk (1st order)	NodetoVec (2nd order)	GraphSage
ROC_AUC_Score	0.9481	0.9496	0.9803

DeepWalk: *node_dim* = 15, *num_walks* = 12, *walk_length* = 50, *window* = 25
NodetoVec: *node_dim* = 15, *num_walks* = 12, *walk_length* = 50, *p* = 100, *q* = 0.01, *window* = 25

From the result we can speculate that NodetoVec, which favors DFS with small q and large p, can outperform DeepWalk. On the other hand, GraphSage performs better than Node2vec. The mechanism of GraphSage mimics selecting all possible paths from a starting node, which solves the problem of neglecting some important paths through random walks.

Analysis

For random walk models, we tried to interpret the optimal hyper-parameter values using some graph theory knowledge. In particular, we conjecture that *num_walks* has some relationship with the degree of nodes. Since all nodes are passed through random walk altogether for *num_walks* times, mathematically, we use *num_walks* as "the mean degree of nodes." The rationale is that by running the random walks for that time, in the best scenario, most nodes could initiate their random-walks for all their corresponding neighboring nodes. The exact number approximately equals 12, which is aligned to our experiments. In general, those hyperparameters should be related to the essence of the graph such as number of nodes, edges and node degrees.

As for GraphSage model, we use kmeans clustering on the node vectors provided by Graphsage. By analyzing the increase of "cross cluster" edges (provided in appendix, we decided to set the number of clusters to 7. We provide LDA plot and color based on the clustering labels.

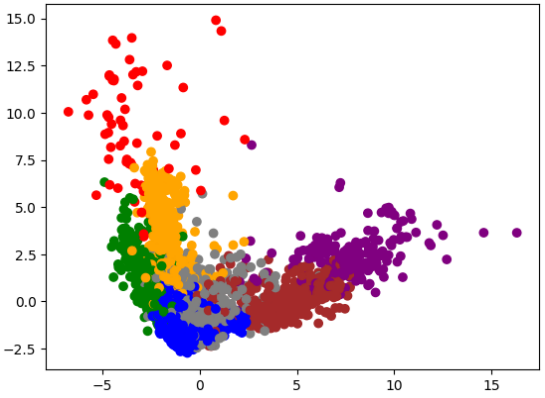

Figure 2. LDA Scatter Plot on node vectors with kmeans clustering generated labels

Conclusion

We explored that all three models can achieve strong baseline with proper hyper-parameter tuning along with the following points:

- Larger *walk_length* i.e., deeper walk yields better result by extracting more information
- An appropriate *num_walks* value can be selected based on average degree of nodes to expectedly extract the neighboring information as much as possible
- NodetoVec (especially larger q) outperforms DeepWalk, which indicates second-order "outward" information is informative to the task
- In NodetoVec, it is favored to apply DFS rather than staying locally in this network
- GNN based model such as GraphSage can generate better node embeddings than random walk methods as it essentially extract all information from neighboring node

Appendix

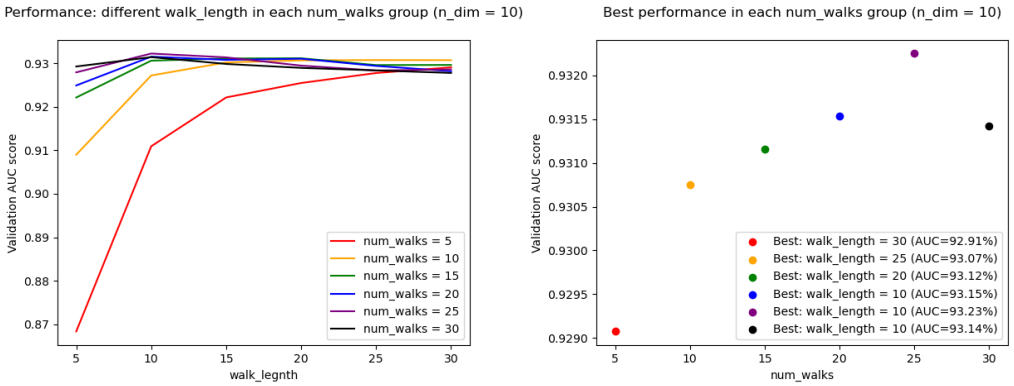

Figure 3. Grid search for DeepWalk model under different *num_walks* and *walk_length* (5,10,15, … ,30) under *node_dim* = 10

```
[28] np.random.seed(0)

     node_dim = 15 # TODO
     num_walks = 12 # TODO
     walk_length = 50 # TODO

     deepwalk_auc_scores = dict()


     print("node dim: %d,\tnum_walks: %d,\twalk_length: %d" % (node_dim, num_walks, walk_length), end="\t")
     model = build_deepwalk(graph, alias_nodes,
                                                    node_dim=node_dim, num_walks=num_walks, walk_length=walk_length)
     deepwalk_auc_scores[(node_dim, num_walks, walk_length,'valid')] = get_auc_score(model, valid_edges, false_edges)
     print("valid auc: %.4f" % (deepwalk_auc_scores[(node_dim, num_walks, walk_length,'valid')]))

     node dim: 15,    num_walks: 12,   walk_length: 50 building a DeepWalk model...     number of walks: 100008 average walk length: 49.2404
     valid auc: 0.9481
```

Figure 4. Validation statistics for 1st order random walk

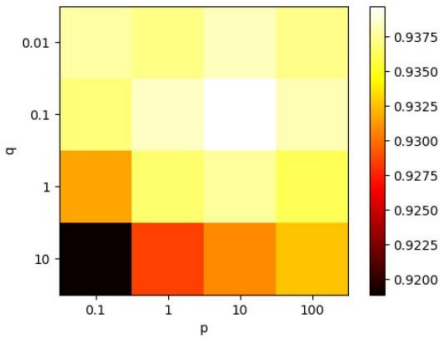

Figure 5. Heatmap of validation AUC score w.r.t. different p = [0.1, 1, 10, 100] and q = [0.01, 0.1, 1,10] (*num_walks* = 13, *walk_length* = 13 under *node_dim* = 10)

```
np.random.seed(0)

node_dim = 15 # TODO
num_walks = mean_degree # TODO
walk_length = 50 # TODO
p = 100 # TODO
q = 0.01 # TODO

node2vec_auc_scores = dict()

print("node dim: %d,\tnum_walks: %d,\twalk_length: %d,\tp: %.2f,\tq: %.2f" % (
    node_dim, num_walks, walk_length, p, q), end="\t")
alias_nodes, alias_edges = preprocess_transition_probs(graph, p=p, q=q)
model = build_node2vec(graph, alias_nodes, alias_edges,
                       node_dim=node_dim, num_walks=num_walks, walk_length=walk_length)
node2vec_auc_scores[(node_dim, num_walks, walk_length, p, q,'valid')] = get_auc_score(model, valid_edges, false_edge
print("valid auc: %.4f" % (node2vec_auc_scores[(node_dim, num_walks, walk_length, p, q,'valid')]))
```

```
node dim: 15,   num_walks: 12,  walk_length: 50,        p: 100.00,      q: 0.01
```

```
/var/folders/pc/yr8w3tcn6r94t0jztwlht69h0000gn/T/ipykernel_23946/129575388.py:8: DeprecationWarning: `np.int` is a
deprecated alias for the builtin `int`. To silence this warning, use `int` by itself. Doing this will not modify an
y behavior and is safe. When replacing `np.int`, you may wish to use e.g. `np.int64` or `np.int32` to specify the p
recision. If you wish to review your current use, check the release note link for additional information.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#deprec
ations
  J = np.zeros(K, dtype=np.int)
```

```
building a node2vec model...    number of walks: 100008 average walk length: 49.2755    training time: 310.2377
valid auc: 0.9496
```

Figure 6. Validation statistics for 2nd order random walk

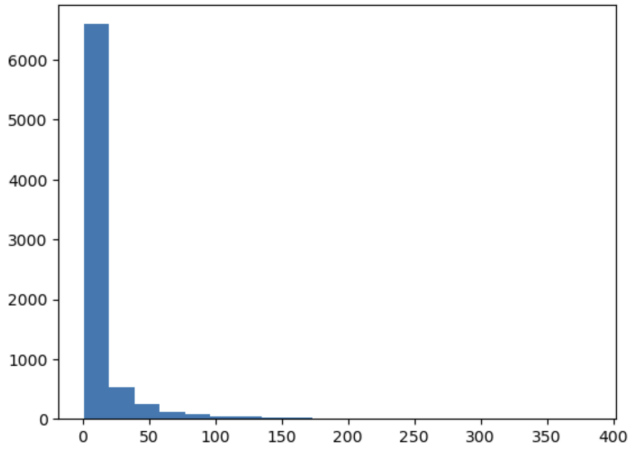

Figure 7. Node degree histogram for conjecturing optimal *num_walk*

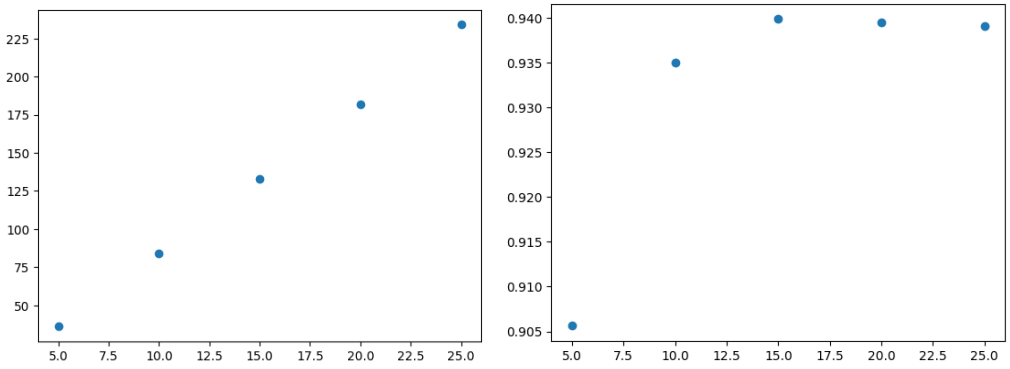

Figure 8. Training time (left) and validation AUC score (right) w.r.t. walk length when using Deep Walk

Among the edges in the validation set, we measured the number of “cross cluster” edges when we split the nodes into different numbers of clusters. We found that there is a significant increase in “cross cluster” edges when k is increased from 7 to 8.

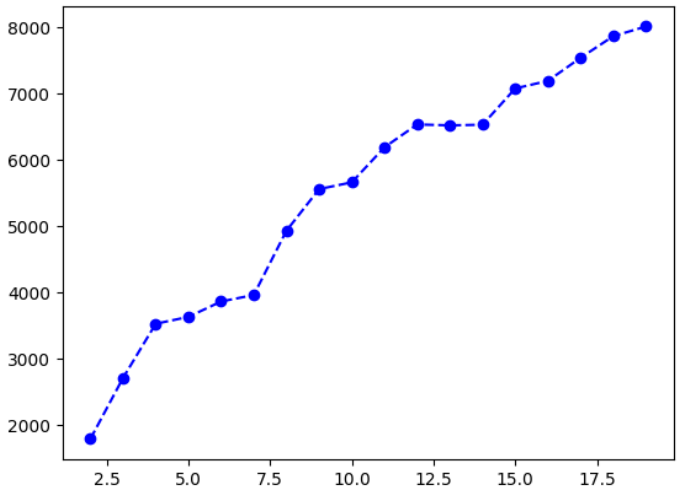

Figure 9. Number of “Cross cluster” edges w.r.t. number of clusters k