# SML Report

Jiacheng Ye, Zixin Ye, Linan Jia

## Introduction

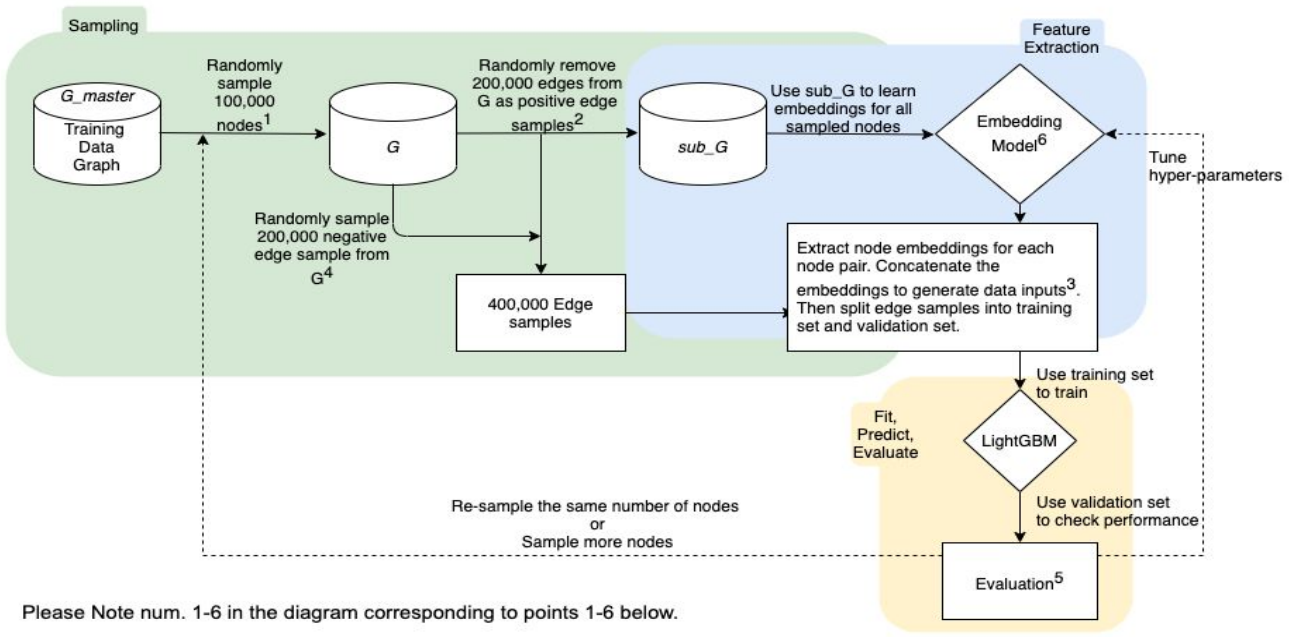
Link prediction is to predict whether there exists connections between two nodes in a graph. It is a popular research field in machine learning recently. Link prediction has enormous applications in real life such as the friend recommendation system in social networks and even in some biological researches (Dong et al., 2013).

This report will first illustrate the final approach we applied. This approach only uses 100,000 sampled nodes to build the graph with around 400,000 edges for feature generation (half actual edges treated as positive samples while second half unlinked edges as negative samples). *Node2Vec* (Grover et al., 2020) is applied for feature generation and *lightGBM* ("Parameter tuning") is adopted for model building. Other sampling and feature embedding approaches will also be discussed in the later section of this report to compare and analyze the performance.

The provided training graph data is in edge list format, where each row represents a node and its out neighbours. Each node is represented by an integer value as user id. The given dataset contains 20,000 source nodes with the maximum user id being 4,867,136. Public test-set contains 2,000 node pairs between source and sink.

## Final Approach

Notations: *G_master*: the given training data graph; *G*, the graph generated by randomly sampling a subset of the nodes from *G_master*; *sub_G*, subgraph of *G*, but with randomly selected positive edges removed.

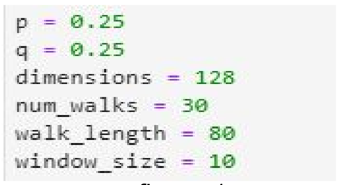

Sampling
G_master
Training Data Graph
Randomly sample 100,000 nodes1
G
Randomly remove 200,000 edges from G as positive edge samples2
sub_G
Use sub_G to learn embeddings for all sampled nodes
Feature Extraction
Embedding Model6
Tune hyper-parameters
Randomly sample 200,000 negative edge sample from G4
400,000 Edge samples
Extract node embeddings for each node pair. Concatenate the embeddings to generate data inputs3. Then split edge samples into training set and validation set.
Use training set to train
Fit, Predict, Evaluate
LightGBM
Use validation set to check performance
Re-sample the same number of nodes or Sample more nodes
Evaluation5

Please Note num. 1-6 in the diagram corresponding to points 1-6 below.

1. In order to ensure the embedding model can learn the embeddings of node pairs in a public-test set, we have made all test nodes to be included in our sampled graph *G*.
2. The total number of edges in sampled graph *G* is about 2000,000, within which we randomly selected 10% edges as our positive samples (labeled 1);
3. We use vector production to concatenate 2 vector embeddings. This binary operation surprisingly out-performs a lot compared to all the other methods we have tried. E.g, simple list concatenation, L2 and L1.

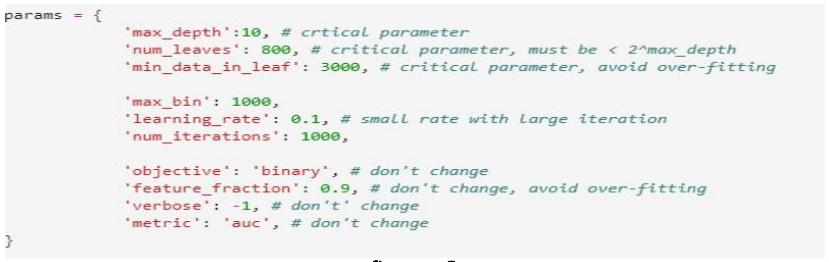
4. While sampling the negative edges, we also need to confirm that true positive edges in *G_master* are not mistakenly selected as negative edges.
5. After evaluation, we can either 1) use the same graph, but tune the hyper-parameters of *Node2Vec* to improve performance; 2) re-sample the graph *G*, and repeat the whole process to improve performance.
6. We use *Node2Vec* (Grover et al., 2020) algorithm to learn a node embedding model. (more detailed explanation is described in section below)

We generate the node embeddings by using the *Node2Vec* algorithm with StellarGraph Library ("Link prediction with Node2Vec") to perform biased random walk("Link prediction with Node2Vec"), and then pass the walk in to *Word2Vec* in Gensim Library to build an embedding model ("Link prediction with Node2Vec"). Based on the *Node2Vec* paper (Grover et al., 2020), we define our parameters as listed in figure 1. After we set those hyper-parameters, we use the *BiasedRandomWalk* function from StellarGarph to perform biased walks [3]. Then we pass the walks to *Word2Vec* and obtain a feature model. After our embeddings are computed for each node in our data set, we concatenate them via a binary operator called *Hadamard*, which is simply a vector product of 2 vectors. For example, for each node pair in the training edge set, we first extract both nodes' embeddings, and perform a vector product on them, and then use the result as training input data.



```
p = 0.25
q = 0.25
dimensions = 128
num_walks = 30
walk_length = 80
window_size = 10
```

figure 1



```
params = {
            'max_depth':10, # crtical parameter
            'num_leaves': 800, # critical parameter, must be < 2^max_depth
            'min_data_in_leaf': 3000, # critical parameter, avoid over-fitting

            'max_bin': 1000,
            'learning_rate': 0.1, # small rate with large iteration
            'num_iterations': 1000,

            'objective': 'binary', # don't change
            'feature_fraction': 0.9, # don't change, avoid over-fitting
            'verbose': -1, # don't' change
            'metric': 'auc', # don't change
}
```

figure 2

After we extract all the embeddings and replace our training data from node pair to concatenated node embeddings, we shuffle them and randomly split them into training and validation data sets. We then use *LightGBM* ("Parameter tuning") from Microsoft to fit the training data, and use the validation data set to examine its performance. To train a *LightGBM* model, we define the hyper-parameters as listed in *figure 2* ("Parameter tuning"). At the end, we obtained 0.93 for validation score, and 0.89 for public test score in Kaggle.

## Result and Discussion

Link prediction can be similar to predicting the next word in NLP. We are maximizing the likelihood of if there is an edge existing between node 1 and node 2, given all the neighbours of the two nodes. Each walk generated is a sentence of the corpus (graph). In NLP, there are 2 really good algorithms, skip-gram and CBOW, which can be used to classify if a given word is the next word of an unfinished sentence. Moreover, there is a really well-known pre-trained model based on those 2 algorithms in NLP, which is the Word2Vec. Ideally, we can pass the walks (sentences) into Word2Vec to re-learn the node embeddings. This workaround can work well because, in NLP Word2Vec learns word embeddings by the local information, i.e, the neighbour words; and this is exactly what we need in our task! Furthermore, we choose *Node2Vec* over *DeepWalk* because *Node2Vec* uses biased random walk, which saves both space complexity and time complexity (Grover et al., 2020).

However, our result is not great in the Kaggle leaderboard. Based on the performances over different hyper-parameter values, and the results from the original work of Node2Vec(Grover et al., 2020), we believe that the hyper-parameters are at their best values. We believe the critical thing that affects the performance the most now is the size of G. This makes sense because no matter how well we generate walks, they are the walks of subG, which means they are estimations of true walks, i.e, estimations of true embeddings. If we tune the parameters too well, it will start to overfit. On the other hand, increasing the sample size will simply increase the likelihood of the estimated walks being close to the true walks. Thus, increasing sample size will theoretically increase the performance. However, due to Node2Vec not being efficient on large graphs, we unfortunately don't have time to experiment on this.

## Alternative sampling method

We also introduced a different sampling approach based on all nodes from the training data set. We randomly selected 200,000 edges from *G_master* as positive samples as well as 200,000 node pairs which are not linked in *G_master* to be negative ones. After selecting both positive and negative samples, we deleted the positive samples from the sampled graph to avoid the overfitting problem.

## Alternative feature embedding  - Local similarities

After sampling, we generate some local similarities: Adar Index, number of followers, followees for both source and sink and inter followers and followees between them. Adar Index measures the closeness of two nodes based on their shared neighbors. A value 0 will be obtained if two nodes are not close, while higher values indicate nodes are closer ("Link prediction with Node2Vec"). After feature generation, we use LightGBM with the same parameters as discussed above to train our models. However, we encounter an extremely serious overfitting problem because we obtain one hundred per-cent accuracy in our validation data no matter in lightGBM model or logistic regression model.

We also tried Networkx to build undirected graphs on top of our sampled graph, and 4 local similarity indices (RA,JC,AA,PA) ("Link prediction with Node2Vec") are tested to learn the features of positive and negative sampled node pairs. The intuition behind these measures is to learn the topological structure/ relationships between the nodes and neighbors, and a score is calculated by these unsupervised methods indicating the likelihood of new node pairs to be linked by time. The Accuracy of this method is 83 on validation set and 75 on test set, which performs lower than the LightGBM method we discussed in this report. We suspect the difference of performance might be caused by lack of features and sampling method. Further investigation is required to discover the causation of this relatively low performance.

## Conclusion

In this report, we proposed an approach with Node2Vec feature generation algorithm and LightGBM model for link prediction. We obtained the samples from a graph with only partial nodes from the whole training dataset. We also examined the results through using different approaches with different sampling methods, graph types, binary operators and features. From comparisons, we identified that sampling, feature extraction methods and a correct binary operator for combining embeddings are critical for link prediction tasks. Further experiments on how sample sizes might affect prediction accuracy can also be conducted. Besides, a more efficient feature learning algorithm for large graph dataset should be considered as well.

### References

Dong, L., Li, Y., Yin, H., Le, H., & Rui, M. (2013). The algorithm of link prediction on social network. *Mathematical Problems in Engineering*, *2013*.

Grover, A., & Leskovec, J. (2016, August). node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 855-864).

Parameters Tuning. Retrieved September 17, 2020, from https://lightgbm.readthedocs.io/en/latest/Parameters-Tuning.html

Link prediction with Node2Vec. Retrieved September 17, 2020, from https://stellargraph.readthedocs.io/en/stable/demos/link-prediction/node2vec-link-prediction.html#refs