A Comparative Study of Network Representation Learning for Link Prediction

Aanshi Patwari [1227546595] **Himali Gajare [1228110730]**

Shrey Malvi [1225539809] **Kavil Parikh [1227933111]**

Arizona State University
Group 10

Abstract

Link prediction is a critical issue in graph mining, and generating an effective network representation is integral to this task. In this study, we provide an overview of various network representation learning (NRL), also known as network embedding. We discuss the basic concepts and models of network embedding and predict the link prediction. We review major research works in each category and describe datasets and applications of network embedding, including its use in network mining tasks such as link prediction. Finally, we provide directions for future research to enhance further development.

1 Introduction

Networks are found in various domains, and link prediction is a crucial task in network analysis. This task involves predicting missing or future links within a network, and accurate link prediction can provide insights into the underlying structure and dynamics of a system. Network Representation Learning (NRL) has emerged as a powerful technique for link prediction by learning low-dimensional representations of nodes and edges in a network that capture their structural and semantic properties (5). However, there is still a lack of consensus on the most effective NRL methods for link prediction, and the performance of existing approaches varies significantly across different networks and tasks (14). A comparative study of NRL methods can help identify the most appropriate technique for a specific application and improve link prediction accuracy and efficiency.

1.1 Motivation

Link prediction has important implications in social network analysis, recommendation systems, and bioinformatics. For instance, social network analysis can use link prediction to identify potential friends or collaborators of a user based on their existing connections (16). In recommendation systems, link prediction can suggest new products or services to users based on their past behavior. In bioinformatics, link prediction can predict protein-protein interactions or gene-disease associations which can be valuable for drug discovery and understanding the mechanisms of various diseases (7). As the amount of data generated in these fields continues to grow, link prediction will become increasingly important in uncovering hidden relationships and patterns.

1.2 Why is it Important?

Network Representation Learning (NRL) has shown promising results in various applications, and its comparative study is essential to evaluate its strengths and weaknesses (2). By conducting a

comparative study of NRL methods for link prediction, we can identify their capabilities, limitations, and factors that affect their performance. This can help researchers and practitioners develop more effective NRL methods and improve the accuracy and efficiency of link prediction in various domains.

2 Project Description

Our study aim is to analyze, assess, and compare various network representation learning algorithms on the CORA and SNAP-Facebook datasets. We are examining various algorithms, including Random Walk-based techniques like DeepWalk, Node2vec, and Attri2vec, and Neural Network-based methods such as Graph Convolution Network (GCN), Graph Attention Network (GAT) and, GraphSAGE.

3 Contributions

Aanshi Patwari [1227546595] focused on implementing and evaluating Node2vec model. She analyzed how the hyperparameters of Node2vec like P,Q and dimensions affects the performance. She also evaluated the model on CORA and SNAP-Facebook datasets and analyzed the results.

Himali Gajare [1228110730] focused on implementing and evaluating Graph Convolution Network (GCN) and Graph Attention Network (GAT) models. She analyzed how GCN uses spectral graph convolutions to generate node embeddings and GAT uses attention mechanism to aggregate neighborhood information. She evaluated the performance of GCN and GAT models on CORA and SNAP-Facebook datasets with different hyperparameters.

Shrey Malvi [1225539809] focused on implementing and evaluating Random Walk-based approaches, such as DeepWalk and Attri2vec. He analyzed how these models generate node embeddings by simulating random walks on the graph. He evaluated the performance of DeepWalk and Attri2vec on CORA and SNAP-Facebook datasets with different hyperparameters like walk length, number of walks, and dimensions.

Kavil Parikh [1227933111] focused on implementing and evaluating Neural Network-based approaches, such as GraphSAGE. He analyzed how GraphSAGE generates node embeddings by aggregating neighborhood information using a hierarchical sampling strategy. He evaluated the performance of GraphSAGE on CORA and SNAP-Facebook datasets with different hyperparameters.

4 Literature Review

In recent years, NRL has gained popularity due to its ability to learn high-quality node embeddings in a network that captures the structural and semantic properties of the network. In the existing literature, a variety of techniques have been proposed for link prediction in network theory, with many emerging in recent years. We focus on network representation learning techniques for link prediction, specifically comparing Random walk-based techniques and Neural network-based techniques.
Random walk-based techniques such as DeepWalk, node2vec, and attri2vec have been proposed to learn node embeddings. DeepWalk(11) proposed by Perozzi et al. (2014) learns node representations by performing random walks on the graph and then using a skip-gram model to learn a low-dimensional representation of each node. node2vec(3) proposed by Grover and Leskovec (2016) is an extension of DeepWalk that allows for more control over the properties of the random walks. Attributed Network Embedding via Subspace Discovery(15), or attri2vec proposed by Daokun Zhang, Jie Yin, Xingquan Zhu, and Chengqi Zhang(2019) is an extension of node2vec that takes into account the node attributes when learning node representations. However, these techniques have limitations in handling large-scale graphs, incorporating rich node attributes, and capturing long-range dependencies in the network beyond local neighborhoods.
Neural network-based techniques, such as Graph Convolutional Networks (GCNs), SDNE (Structural Deep Network Embedding), and GraphSAGE, have been shown to be advantageous for network representation learning. These techniques can handle large graphs with rich node attributes and can capture structural patterns beyond local neighborhoods. SDNE(13) proposed by Wang et al. (2016) is a deep autoencoder-based approach that learns embeddings by preserving both first-order and second-order proximity in the network , which has been shown to achieve state-of-the-art performance on various graph-based tasks such as node clustering, link prediction, and visualization. GCNs(8), proposed by Kipf and Welling (2016), apply a convolution operation to node features to compute a

dot product of the kernel and the signal at each point in the signal which enables GCNs to capture rich information and dependencies present in the input graph while still being computationally efficient. GraphSAGE(4), proposed by Hamilton et al. (2017), aggregates node features of a node's neighbors, allowing it to learn and generalize well to new nodes and graphs. This technique is particularly useful when working with graphs that have varying structures. Graph Attention Networks (GATs)(12), proposed by Veličković et al. (2018), utilizes an anisotropic operation during recursive neighborhood diffusion to improve the learning capacity of the model. This is achieved through an attention mechanism which assigns varying degrees of importance to the contributions of each neighbor, thereby exploiting the anisotropy paradigm.

5 Dataset

5.1 CORA

The CORA dataset is a well-known benchmark dataset for citation network analysis or citation recommendation. It has 5,429 citation links between 2,708 papers that are divided into seven sections based on the research area. A directed acyclic network is formed by the connections between papers, and each article is represented as a bag-of-words feature vector. The feature vector length is 1433. Citation recommendation seeks to foretell which papers, depending on the content and citation network of a particular paper, should be mentioned. The CORA dataset possesses a total of 2485 distinct papers as nodes, and 5209 citation links connecting them. We divided the dataset into three sets for evaluation: a training set comprising 702 examples, a validation set containing 234 examples, and a test set consisting of 1040 examples.

5.2 SNAP ego-Facebook

The SNAP ego-Facebook dataset is a subset of the larger SNAP Facebook dataset, which contains a collection of anonymized Facebook social network data of 10 ego-networks. An ego-network is a social network consisting of an individual (referred to as an "ego") and their immediate social contacts (referred to as "alters"). Each ego-network(10) in the dataset contains the profiles of the ego and their alters, along with information on their friendship relationships and other attributes such as gender and age. Here, we have considered 1 ego-network namely ego 0. The feature vector length is 224. The SNAP ego-network of ego 0 dataset includes 347 nodes and 5038 links. We divided the dataset into three sets for evaluation: a training set with 1209 examples, a validation set with 403 examples, and a test set with 2014 examples.

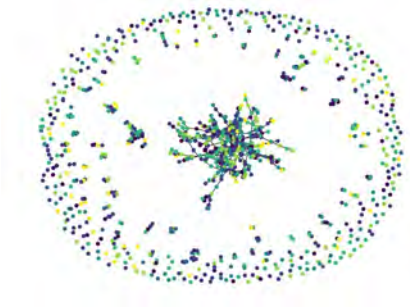

Figure 1: Visualization of CORA graph. Each different node color represents the subject of the paper

Figure 2: Visualization of 0th circle of ego-facebook undirected network. Diameter of nodes is based on the degree of it

6 Approach

6.1 Algorithm Study

"Embedding" is the process of mapping network nodes to a low-dimensional space to reveal information about their similarity and network structure. In embedding-based network representation, there

are two main types of embeddings: node-based and edge-based. In node-based embedding, we use vectors to represent the nodes of a network. Random walk defines node similarity in a flexible and stochastic way, incorporating local and higher-order neighborhood information, and doesn't require considering all node pairs during training (1). We will be mapping node u and v to low-dimensional vectors z_u and z_v. Specifically, we can define nearby nodes $N_R(u)$ as the neighborhood of node u obtained by some strategy R. We could run short fixed length random walks starting from each node on the graph using some strategy R to collect $N_R(u)$, which is the multiset of nodes visited on random walks starting from u (1). To optimize embeddings to maximize the likelihood of random walk co-occurrences, we compute loss function as:

$$\mathcal{L} = \sum_{u \in V} \sum_{v \in N_R(u)} -log\left(\frac{exp(z_u^T z_v)}{exp(\sum_{n \in V} z_u^T z_n)}\right) \tag{1}$$

Our implementation of node-based embedding includes Deepwalk, Node2vec, and Attri2vec.

6.1.1 DeepWalk

DeepWalk uses random paths in graphs to reveal network patterns, which are then learned and encoded by neural networks (Word2Vec model) to generate embeddings. Paths are generated by randomly selecting neighbors of the current node and continuing through the walk until the desired number of steps is reached (11). The idea behind DeepWalk is to treat each random walk as a sentence and each node in the walk as a word, and then apply a skip-gram model from natural language processing to learn node embeddings. The skip-gram model predicts the probability of observing a context node given a target node, and uses stochastic gradient descent to minimize the negative log-likelihood of the observed context nodes.

The objective function for the skip-gram model used in DeepWalk is given by:

$$J(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{c \in \mathcal{C}_t} \log p(c|v_t, \theta) \tag{2}$$

For our evaluation, we have implemented the sampling strategy in DeepWalk algorithm, where we took the node2vec algorithm as a baseline with the tuning parameter P (return parameter) and Q (in-out parameter) equal to 1 (3).

6.1.2 Node2vec

Node2vec algorithm generates node embeddings using second-order biased random walks, where transition probabilities depend on the present and previous nodes. It is based on the idea of generating random walks on the graph and using these walks to learn node embeddings. The first step of the algorithm involves generating a large number of random walks on the graph. For each random walk, the algorithm assigns a probability distribution over the next node to visit based on the current node and the previous node. This probability distribution is determined by tuning parameters called the "return parameter" and the "in-out parameter", which control how likely the random walk is to revisit nodes already visited and how likely it is to explore new nodes, respectively. Once the random walks are generated, the context pairs are obtained and used to train a word2vec model to generate node embeddings. The objective function for the skip-gram model used in Node2vec is identical to the one used in DeepWalk, but with a different set of context nodes. (3).

6.1.3 Attri2vec

Attri2vec takes into account the node attributes when learning node embeddings, while DeepWalk does not. It does this by first generating a set of random walks on the graph. These random walks are then used to train a continuous bag-of-words model, which learns a representation of each node as a vector of word counts. The word counts in this vector represent the frequency with which the node appears in the context of other nodes. The final node embeddings are then obtained by dimensionality reduction on the learned word vectors. The joint objective function used in Attri2vec is given by:

$$J(\theta) = -\sum_{i=1}^{N} \log p(v_i, a_i | \mathcal{N}(v_i), \theta) \tag{3}$$

where θ represents the parameters of the model, v_i represents the i-th node in the network, a_i represents the attributes of node v_i, $\mathcal{N}(v_i)$ represents the neighbors of node v_i, and $p(v_i, a_i|\mathcal{N}(v_i), \theta)$ represents the joint probability of observing node v_i and its attributes a_i given its neighbors and model parameters θ.

6.1.4 Graph Convolutional Networks(GCN)

The key idea behind GCNs is to learn representations of nodes that capture both local and global information. In GCN, first, the node features are initialized for each node in the graph. Then it performs a convolution operation on the node features, where each node aggregates information from its neighbors. This operation is similar to a traditional convolutional neural network (CNN), but instead of a fixed filter, the filter is learned based on the graph structure and lastly, ReLU activation and pooling layers are applied. The above steps are repeated for multiple layers to learn higher-level features. Finally, the output of the last layer is used to perform the link prediction. The following is the mathematical equation for a graph convolutional layer:

$$\mathbf{H}^{(l+1)} = \sigma(\mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}\mathbf{W}^{(l)}\mathbf{H}^{(l)}) \tag{4}$$

In this equation, $\mathbf{H}^{(l)}$ is the hidden representation of the nodes in the lth layer, $\mathbf{W}^{(l)}$ is the weight matrix of the lth layer, $\mathbf{A}$ is the adjacency matrix of the graph, and σ is a non-linear activation function.

6.1.5 GraphSAGE

While GCN aggregates information by computing a weighted average of features from neighboring nodes, GraphSage uses a more general framework of sampling and aggregation. Specifically, GraphSage samples a fixed-size set of neighbors for each node and then aggregates their features using a neural network that is learned during training. The GraphSAGE algorithm can be defined as follows:

$$h_{\mathcal{N}(v)}^{(l)} = \text{AGGREGATE}^{(l)}\left(h_u^{(l-1)} : u \in \mathcal{N}(v)\right) \tag{5}$$

$$h_v^{(l)} = \sigma\left(W^{(l)} \cdot \text{CONCAT}\left(h_v^{(l-1)}, h_{\mathcal{N}(v)}^{(l)}\right)\right) \tag{6}$$

where $h_v^{(l)}$ is the embedding of node v at layer l, $\mathcal{N}(v)$ is the set of neighbors of node v, $\text{AGGREGATE}^{(l)}$ is the aggregation function used at layer l, CONCAT is the concatenation operation, $W^{(l)}$ is the weight matrix of layer l, $h_u^{(l-1)}$ is the embedding of node u at layer $l-1$, and σ is the activation function.

6.1.6 Graph Attention Networks (GAT)

GAT uses attention mechanisms to weight the contribution of neighboring nodes during the convolution operation unlike GCN which uses a fixed-weight parameter matrix. This allows GAT to selectively focus on the most relevant nodes during the aggregation step, which can improve performance in tasks where some nodes are more important than others. The following equations describe the GAT architecture:

$$\mathbf{h}_i^{(l+1)} = \sigma\left(\sum_{j \in \mathcal{N}(i)} \alpha_{ij}^{(l)} \mathbf{W}^{(l)} \mathbf{h}_j^{(l)}\right) \tag{7}$$

where $\mathbf{h}_i^{(l)}$ is the representation of node i at layer l, $\mathcal{N}(i)$ is the set of neighbors of node i, $\alpha_{ij}^{(l)}$ is the attention weight between nodes i and j at layer l, $\mathbf{W}^{(l)}$ is the weight matrix at layer l, and σ is a non-linear activation function.

The attention weights are computed using the following equation:

$$\alpha_{ij}^{(l)} = \frac{\exp\left(\mathbf{a}_i^{(l)} \cdot \mathbf{a}_j^{(l)}\right)}{\sum_{k \in \mathcal{N}(i)} \exp\left(\mathbf{a}_i^{(l)} \cdot \mathbf{a}_k^{(l)}\right)} \tag{8}$$

where $\mathbf{a}_i^{(l)}$ is the attention vector for node i at layer l. The attention vector is computed using the following equation:

$$\mathbf{a}_i^{(l)} = \mathbf{W}^{(a)} \mathbf{h}_i^{(l)} \tag{9}$$

6.2 Parameter Settings

Deepwalk, Node2vec and Attri2vec employ a strategy of capturing the similarity of context nodes in random walks. On the other hand, GAT, GraphSAGE and GCN employ unsupervised learning to make nodes that co-occur in short random walks represented closely in the embedding space. For all six algorithms, node embeddings were learned via an unsupervised learning procedure that generated short random walks from the graph and trained node embeddings on batches of target and context pairs. The dimensionality of the embeddings was set to 128 for all algorithms, while the other parameters varied. Specifically, Node2vec was trained with 20 walks from each node, while Attri2vec, GraphSAGE, GAT and GCN were trained with 4, 1, 1, and 1 walks from each node, respectively. The batch size was fixed at 50 for all algorithms, while the number of epochs was set to 6.

6.3 Implementation Details

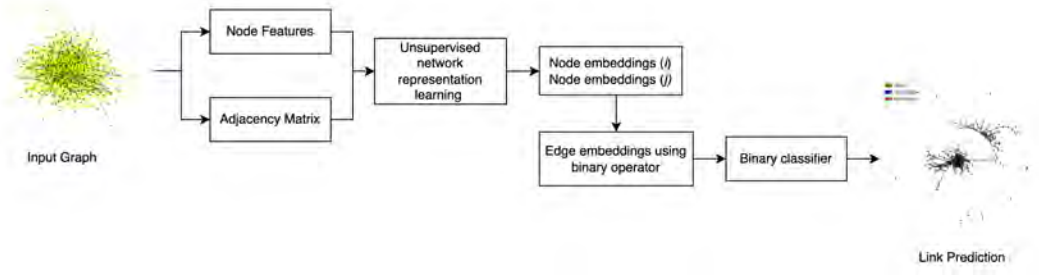

Figure 3: End-to-end block diagram of our methodology for link prediction

Link prediction task involves prediction of missing edges in the graph. Our Input data comprises of Node features and Adjacency matrix. First, we use representation learning algorithms to compute the node embeddings using unsupervised setting. Then, we apply binary operator to generate edge embedding based on two node embeddings. Lastly, these edge representations are used as an input and output with 0/1 label (0 for negative link, 1 for positive link) in logistic regression for the link prediction.

One important detail in link prediction task is the splitting of graph into train/val/test set to avoid data leakage while evaluating the model. The Train Graph was used to compute node embeddings, while the train set was composed of positive and negative edges that were not used in computing node embeddings. The validation set included edges that were not used in computing node embeddings or training the classifier and was used to determine the best classifier. Finally, the Test Graph and test set were used to evaluate the performance of the final model.

7 Results

We have evaluated the performance of the link prediction using the AUC metric for all the algorithms on the validation set. Table 1 shows the quantitative analysis of embedding algorithms for CORA and ego-Facebook dataset classified based on operators. In this evaluation, we used four binary embeddings namely Hadamard, Weighted L1, Weighted L2, and Average to compute edge embeddings from the combination of node embeddings of dimension 128. Hadamard combines node embeddings using dot product, whereas Weighted L1 and L2 compute absolute and euclidean distance and Average

simply takes the average of both. We chose the best-performing binary operator (highlighted in the table) from this evaluation for each of the algorithm and dataset combinations and then finally evaluated our model on the test set. We used the same classification model (logistic regression) with the default setting for the link prediction task to conduct a fair evaluation of embedding algorithms. A comparison of these algorithms on the test set has been shown in Table 2.

Operators	Datasets	DeepWalk	Node2vec	Attri2vec	GCN	GAT	GraphSAGE
Hadamard	Cora	0.759	0.779	0.897	**0.872**	**0.901**	0.922
	Facebook	0.678	0.731	0.743	**0.945**	0.907	**0.904**
Weighted-L1	Cora	0.806	0.850	0.942	0.822	0.851	0.924
	Facebook	0.779	0.832	**0.760**	0.920	0.942	0.899
Weighted-L2	Cora	**0.823**	**0.872**	**0.950**	0.771	0.848	**0.926**
	Facebook	**0.810**	**0.855**	0.734	0.906	**0.946**	0.901
Average	Cora	0.612	0.545	0.516	0.559	0.511	0.563
	Facebook	0.723	0.706	0.684	0.852	0.820	0.778

Table 1: Comparison of representation learning algorithms classified based on binary operators using Area Under the Curve (AUC) metric

As shown in Table 2, Attri2vec generated the best result for CORA dataset and GAT outperformed all other algorithms for ego-Facebook dataset. From this quantitative analysis, we can imply that neural network-based embedding algorithms are performing better than random walk-based methods for the ego-Facebook dataset (typically a larger dataset) because aggregation parameters in graph-based methods are learned to have similar representations for pairs of nodes that occur in short random walks. As smaller datasets generally have more homogeneous structures i.e. nodes have similar attributes and behave in a similar manner, Attri2vec performed better for CORA because it is relatively easier for Attri2vec to learn meaningful representations that capture the similarities and differences between the nodes.

Algorithms	Cora	Facebook
Deepwalk	0.8493010355	0.8505506144
Node2vec	0.8317159763	0.8495920809
Attri2vec	**0.9434948225**	0.759442098
GCN	0.8989534024	0.9340411558
GAT	0.8776941568	**0.93667663**
GraphSAGE	0.9241383136	0.8923986908

Table 2: Comparision of embedding algorithms on test set for CORA and ego-Facebook dataset using the best performing binary operators

Figure 4: Predicted test edges for CORA dataset using Attri2vec algorithm

Figure 5: Predicted test edges for ego-Facebook dataset using GAT algorithm

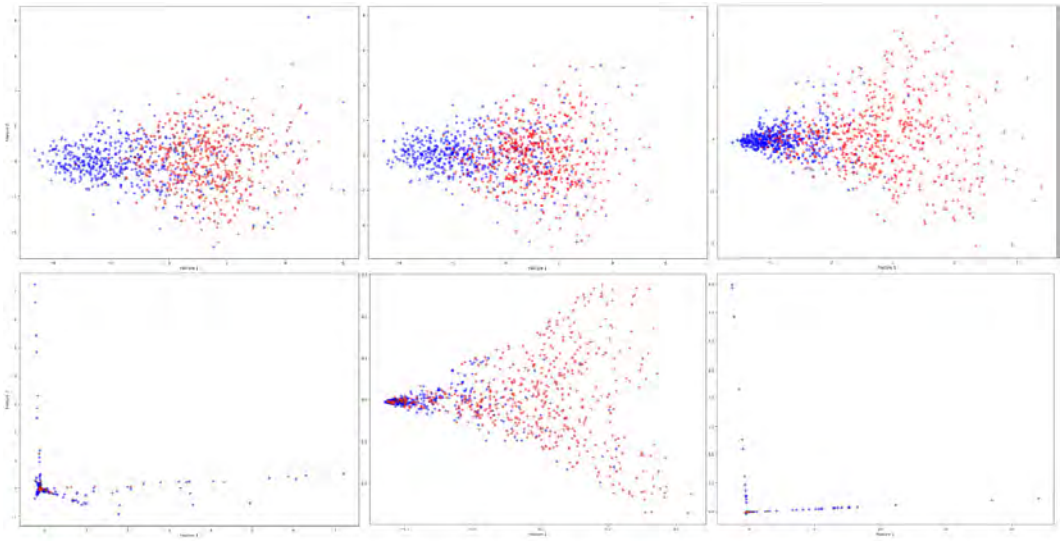

Figure 6: Edge embeddings representation in 2D space of Deepwalk, Node2vec, Attri2vec, GCN, GraphSAGE and GAT (left to right, top to bottom) for CORA dataset. Red and blue represent predicted positive and negative links respectively.

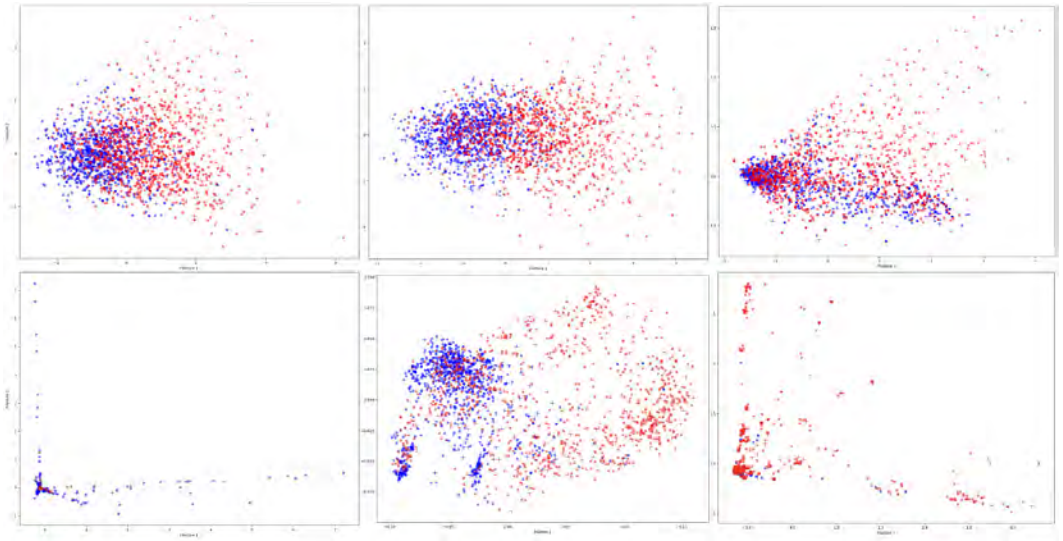

Figure 7: Edge embeddings representation in 2D space of Deepwalk, Node2vec, Attri2vec, GCN, GraphSAGE and GAT (left to right, top to bottom) for ego-Facebook dataset. Red and blue represent predicted positive and negative links respectively.

The edge embeddings were generated with 128 dimensions, which were then projected to 2 dimensions using the Principal Components Analysis (PCA) algorithm for visualization purposes which are shown in the figures 6 and 7. Figures 4 and 5 represents the qualitative analysis of the best-performing embedding algorithms on the CORA and ego-facebook dataset. The test examples consist of all nodes, with green edges representing correct predictions made by our model. Meanwhile, false negatives and false positives are depicted by the blue and red edges, respectively.

8 Conclusion

Within this research, we provided a comparative analysis of diverse network representation learning algorithms for the purpose of link prediction. We conducted experiments on two esteemed datasets, namely CORA and SNAP-Facebook, and we evaluated a plethora of Random Walk-based and

Neural Network-based approaches. Our findings indicate that Random Walk-based methods, for instance, DeepWalk and Node2vec, exhibit exceptional performance on networks that are sparse and homogeneous. Whereas, Neural Network-based approaches, such as GCN, GAT, and GraphSAGE, present superior outcomes on networks that are dense and heterogeneous. In light of our analysis, we drew the conclusion that each algorithm possesses its distinct advantages and limitations, and the selection of an algorithm is dependent on the characteristics of the dataset and the anticipated performance metrics.

9 Future Work

Our research provides several suggestions for future investigations in network representation learning (6). Firstly, we propose a deeper exploration of the interpretability of the learned representations and their utility in revealing the network's underlying structure (5). Secondly, we suggest considering the adoption of alternative graph embedding methods to enhance the accuracy of link prediction such as Variational Graph Autoencoder (9) which has demonstrated state-of-the-art results. Lastly, we recommend exploring the application of transfer learning approaches to leverage pre-trained models on comparable datasets for improved link prediction performance on new datasets.

A Appendix

Random Walk Model

In the Random Walk model, we generate a set of random walks on the graph and use these walks to learn the node embeddings. Let $G = (V, E)$ be a graph with $|V| = n$ nodes and $|E| = m$ edges. We define $P_{u,v}$ as the probability of moving from node u to node v in one step, where $u, v \in V$. The probability distribution P is defined as follows:

$$P_{u,v} = \begin{cases} \frac{1}{deg(u)} & \text{if } (u,v) \in E \ 0 \\ \text{otherwise} \end{cases} \tag{10}$$

where $deg(u)$ represents the degree of node u i.e. the number of edges incident on node u.

We generate k random walks of length l starting from each node in the graph. Let $w_{i,j}$ denote the j^{th} node in the i^{th} random walk. The probability of generating a random walk $w = (w_1, w_2, ..., w_l)$ starting from node u is given by:

$$Pr(w|u) = \prod_{i=1}^{l-1} P_{w_i, w_{i+1}} \tag{11}$$

The objective of the Random Walk model is to maximize the log-likelihood of observing the set of generated random walks:

$$\max_{\Theta} \sum_{u \in V} \sum_{w \in W_u} \log Pr(w|u, \Theta) \tag{12}$$

where Θ represents the model parameters and W_u represents the set of all random walks starting from node u.

Skip Gram Model

In the Skip Gram model, we aim to learn the embeddings of a node by predicting its context nodes in a fixed size window. Let $G = (V, E)$ be a graph with $|V| = n$ nodes and $|E| = m$ edges. We define d_u as the embedding of node u and C_u as the set of context nodes of node u.

Given a node u, we define the probability of observing a context node v given u as follows:

$$P(v|u) = \frac{\exp(d_u \cdot d_v)}{\sum_{w \in V} \exp(d_u \cdot d_w)} \tag{13}$$

where $\cdot$ represents the dot product of two embeddings.

We use the Skip Gram model to maximize the log-likelihood of observing the set of context nodes for each node in the graph:

$$\max_{\Theta} \sum_{u \in V} \sum_{v \in C_u} \log P(v|u, \Theta) \tag{14}$$

where Θ represents the model parameters.

References

[1] Goldberg, Y., Levy, O.: word2vec explained: deriving mikolov et al.'s negative-sampling word-embedding method. arXiv preprint arXiv:1402.3722 (2014)

[2] Goyal, P., Ferrara, E.: Graph embedding techniques, applications, and performance: A survey. Knowledge-Based Systems **151**, 78–94 (2018)

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

[4] Hamilton, W., Ying, Z., Leskovec, J.: Inductive representation learning on large graphs. Advances in neural information processing systems **30** (2017)

[5] Hamilton, W.L., Ying, R., Leskovec, J.: Representation learning on graphs: Methods and applications. arXiv preprint arXiv:1709.05584 (2017)

[6] Hinkel, C., Kühne, T., Kramer, S., Schüller, T., Schuhmann, F.: A comparative study of network representation learning for link prediction. arXiv preprint arXiv:2006.02525 (2020)

[7] Hopkins, A.L.: Network pharmacology: the next paradigm in drug discovery. Nature chemical biology **4**(11), 682–690 (2008)

[8] Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907 (2016)

[9] Kipf, T.N., Welling, M.: Variational graph auto-encoders. arXiv preprint arXiv:1611.07308 (2016)

[10] Leskovec, J., Mcauley, J.: Learning to discover social circles in ego networks. Advances in neural information processing systems **25** (2012)

[11] Perozzi, B., Al-Rfou, R., Skiena, S.: Deepwalk: Online learning of social representations. In: Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 701–710 (2014)

[12] Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., Bengio, Y.: Graph attention networks. arXiv preprint arXiv:1710.10903 (2017)

[13] Wang, D., Cui, P., Zhu, W.: Structural deep network embedding. In: Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 1225–1234 (2016)

[14] Zhang, D., Yin, J., Zhu, X., Zhang, C.: Network representation learning: A survey. IEEE transactions on Big Data **6**(1), 3–28 (2018)

[15] Zhang, D., Yin, J., Zhu, X., Zhang, C.: Attributed network embedding via subspace discovery. Data Mining and Knowledge Discovery **33**, 1953–1980 (2019)

[16] Zhang, M., Chen, Y.: Link prediction based on graph neural networks. Advances in neural information processing systems **31** (2018)