Feature Engineering on Link Prediction

Astrit Tola
UTD NetID:axt200006
Astrit.Tola@UTDallas.edu
Department of Mathematical Sciences
University of Texas at Dallas

Dona Hasini Gammune
UTD NetID:dvg190000
DonaHasini.Gammune@UTDallas.edu
Department of Mathematical Sciences
University of Texas at Dallas

Brighton Nuwagira
UTD NetID:bxb210001
Brighton.Nuwagira@UTDallas.edu
Department of Mathematical Sciences
University of Texas at Dallas

Mahesh Ranpati Dewage
UTD NetID:mkr200000
ManjulaMahesh.RanpatiDewage@UTDallas.edu
Department of Mathematical Sciences
University of Texas at Dallas

***Abstract*—Link prediction plays a crucial role in the field of network analysis as it aims to estimate the likelihood of future connections between nodes in a graph based on available data. In this project, we make use of the publicly accessible CiteSeer Benchmark dataset and leverage the functionality provided by the networkx package for performing various operations on graphs. To construct our Citeseer graph, we introduce hypothetical negative edges along with existing positive edges. From this modified graph, we extract feature sets using five different methods for feature extraction including Shortest Path, Jaccard Index, Salton Index, Sorensen Index, and Common Field. For predicting link presence within our test set, we utilize both a deep neural network model and an XGBoost algorithm. The findings highlight the efficacy of this method in forecasting forthcoming citation links among scholarly articles. This contributes significantly to gaining insights into scholarly communication patterns and identifying trends within academic research.**

***Index Terms*—Link Prediction, Graph Analysis, Neural Network, XGBoost, Feature Engineering.**

I. Introduction

Link prediction is a widely used technique in various domains. Its purpose is to forecast the likelihood of a link between two nodes within a graph, using information derived from observed graph data. By closely examining patterns among existing connections within networks, link prediction algorithms offer valuable insights into prospective links between nodes. Understanding complex systems and gaining practical applications across multiple fields are notable benefits achieved through this approach. Graph machine learning methods are frequently applied to tackle this task effectively [14].

In the field of social network analysis, methods for link prediction are utilized to identify potential connections between individuals in networks [15], such as friendships and collaborations. Anticipating future associations can offer several benefits across various domains. For instance, in online social platforms, accurate predictions enable enhanced personalized recommendations and more effective targeted marketing strategies.

Link prediction plays a crucial role in the field of security [16] as well. Its significance lies in its ability to uncover concealed patterns and connections that are indicative of malicious activities or individuals involved in criminal behavior. By identifying potential links between individuals within security networks, link prediction aids in dismantling criminal networks and safeguarding communities against various threats.

We can better comprehend scholarly communication, research trends, and the advancement of scientific knowledge by making predictions about future citation relationships between academic works. Identifying which academic articles are most likely to cite one another includes utilizing computer tools to forecast future citation linkages. Academic publications frequently quote or reference other sources to give background information, bolster arguments, or recognize earlier studies. As a result, the citation network these references create is a useful source of data regarding the connections and influence between various works. Predicting connections that haven't yet happened but are expected to emerge in the future is the subject of the link prediction issue. The objective is to use the current citation network and historical citation trends to predict probable links in the future.

In academia and research, it is important to use methods for forecasting future citation relationships. Researchers and academics might use it to find potentially important publications or new trends. Additionally, it can increase the effectiveness of information retrieval by suggesting relevant works while doing literature searches. Understanding citation trends can also assist academic publishers and journals refine their publication plans and estimating their impact factors. Numerous techniques from network analysis and machine learning, including graph-based similarity, approaches based on random walks, latent space models, deep learning models, etc., can be utilized to achieve this goal.

In this project, we aim to develop a robust model, incorporating a neural network component, that can accurately predict future citation links between academic papers. We employed the publicly available CiteSeer Benchmark data set, comprising 3312 scientific publications categorized into six distinct fields. The data set includes a citation network with 4732 links representing the citations between publications. We employ the Citesser benchmark dataset and the 'networkx' package for graph operations. The process involves introducing equivalent negative edges alongside existing positive edges in the Citeseer graph. We extract a feature set from the original graph using five feature extraction functions, namely Shortest path, Jaccard Index, Salton Index, Sorensen Index, and Common Field, applied to a selected training set. We employ two methods to predict link existence within the test set: our custom implementation of a Neural Network (NN) and the XGboost algorithm. By applying these steps, we aim to gain valuable insights into citation network dynamics and predict future citation links between academic papers.

II. Background Work

"Semi-Supervised Classification with Graph Convolutional Networks" by Thomas Kipf and Max Welling (2016) may focus on semi-supervised classification. However, it has laid the foundation for graph convolutional networks in link prediction tasks [10].

"Representation Learning on Graphs: Methods and Applications" by William L. Hamilton, Rex Ying, and Jure Leskovec (2017) provides a comprehensive survey of representation learning methods on graphs, including those applicable to link prediction using neural networks [12].

"Heterogeneous Graph Attention Network" by Chuxu Zhang et al. (2019) introduces the concept of using attention mechanisms for link prediction in heterogeneous information networks through the HAN model [13].

"Representation Learning for Attributed Multiplex Heterogeneous Network" by Hongchang Gao et al. (2018) presents a method for representation learning in attributed multiplex heterogeneous networks, proving useful for link prediction tasks within such networks [11].

Collectively, these papers demonstrate how neural networks and relevant mechanisms can effectively learn graph representations and patterns, thus significantly advancing link prediction capabilities across various network types.

III. Problem Definition

Let $G = (V, E)$ be an undirected graph where $V = \{v_1, v_2, ..., v_N\}$ is the set of nodes and E the node pairs within the graph. The adjacency matrix of G is $A \in \{0,1\}^{N \times N}$, where $A_{i,j} = 1$ if there is an edge between nodes v_i and v_j, and 0 otherwise. We denote the feature matrix of edges as $X \in \mathbb{R}^{M \times F}$, where F indicates the number of edge features, M number of edges and x_m represents the feature vector of edge e_m, $1 \leq m \leq M$.

IV. Theory And Conceptual Study

For our experiments we use a neural network model with two hidden layers. We used for output layer the sigmoid activation function and for hidden layers ReLU activation function. We also add the $l2$ penalty term.

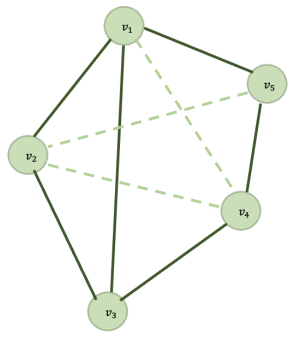


Fig. 1. The Link Prediction Graph (v_1v_4, v_2v_4, and v_2v_5 are missing links)

A. Deep neural network

A deep neural network is a collection of different layers. The input layer captures and passes the input signals to the next layer. The hidden layers perform nonlinear transformations of the inputs and pass the result to the final layer and the output layer, which delivers the final results.

A model with an input layer, k number of hidden layers, and an output layer can be written as

$$X \to \boldsymbol{h}^{(1)}(X) \to \cdots \to \boldsymbol{h}^{(k)}\{\boldsymbol{h}^{(k-1)}\} \to a\{\boldsymbol{h}^{(k)}\} \to \hat{\boldsymbol{Y}}$$

implying the target f is assumed have the form

$$f(X;\theta) = a\{\boldsymbol{h}^k\{\boldsymbol{h}^{(k-1)}\{\ldots\{\boldsymbol{h}^{(1)}(X)\}\ldots\}\}\}$$

with specified activation function a. Parameter θ consists of biases and weights. Figure 2 represents a simple neural network with one hidden layer.

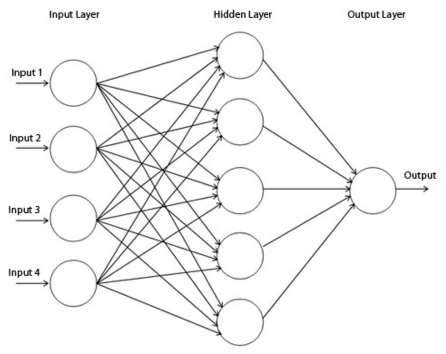


Fig. 2. A simple neural network with one hidden layer

We built a deep neural network using the following steps:

- Choosing activation function
- Forward propagation
- Choosing loss (cost) function
- Backward propagation
- Parameter tuning

1) Activation function: The activation function takes the sum of the weighted inputs as input and computes the output of a neuron. The sigmoid function is one of the popular activation functions used in neural networks. It can transform any real number to a value between 0 and 1. This property is beneficial as it enables the neural network to learn and capture non-linear relationships in the data. The sigmoid function is mathematically represented as:

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

derivative of sigmoid would be

$$\frac{\partial \sigma}{\partial x} = \sigma(x)(1-\sigma(x))$$

Also, the $ReLU$ activation function can be represented as:

$$ReLU(x) = \max(0, x)$$

derivative of $ReLU$ would be

$$\frac{\partial(ReLU(x))}{\partial x} = \mathbf{I}_{x>0} = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$

2) Forward Propagation: During forward propagation, information flows in a forward direction through the network. The input information is first propagated through the layers, reaching the hidden units at each layer, where computations and transformations occur. Finally, the process culminates in producing the predicted output $\hat{\boldsymbol{y}}$, representing the network's prediction for a given input.

The forward propagation equations for a neural network with one hidden layer can be formulated as:

$$\begin{aligned} \boldsymbol{Z}_1 &= \boldsymbol{W}_1^T \boldsymbol{X} + \boldsymbol{b}_1 \\ \boldsymbol{A}_1 &= \sigma(\boldsymbol{Z}_1) \\ \boldsymbol{Z}_{out} &= \boldsymbol{W}_{out}^T \boldsymbol{A}_1 + \boldsymbol{b}_{out} \\ \boldsymbol{O} &= \sigma(\boldsymbol{Z}_{out}) \end{aligned}$$

where, $\boldsymbol{X}, \boldsymbol{W}, \boldsymbol{b}, a, \boldsymbol{O}$ represents corresponding input, weights, biases, activation function, and output.

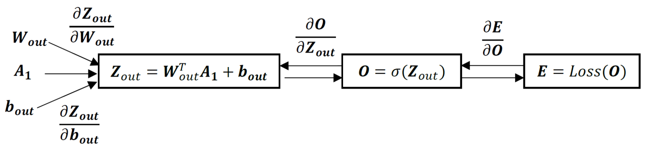
3) Calculating loss: Next, we compare the result from the forward propagation with the actual output. The goal is to minimize the difference between predicted and actual output. This difference is quantified using a loss function (cost function) that measures the dissimilarity between predicted and actual output. Commonly used loss functions in different problems include mean squared error (MSE) for regression tasks and cross-entropy loss for classification tasks.

4) Backward propagation: Now, we minimize the loss by updating weights and biases while traveling back. This process is known as "Backward Propagation". We employ a standard algorithm called "Gradient Descent", which iteratively adjusts the weights and biases in the opposite direction of the gradients, gradually reducing the loss with an added penalty term. We update weights using the rule

$$\boldsymbol{W} \rightarrow \boldsymbol{W} - \eta(\frac{\partial \boldsymbol{E}}{\partial \boldsymbol{W}} + \lambda \|\boldsymbol{W}\|_2)$$

Here we consider two cases,

(a) *Updating weights and bias at output layer*: Our goal is to update the weights ($\boldsymbol{W}_{out}$) and bias ($\boldsymbol{b}_{out}$) of the output layer. Let's take sigmoid as our activation and MSE as our loss function (figure 3). The values of $\frac{\partial \boldsymbol{E}}{\partial \boldsymbol{W}_{out}}$ and $\frac{\partial \boldsymbol{E}}{\partial \boldsymbol{b}_{out}}$ can be calculated as



$\frac{\partial Z_{out}}{\partial W_{out}}$
W_{out}
A_1
b_{out}
$\frac{\partial Z_{out}}{\partial b_{out}}$
$Z_{out} = W_{out}^T A_1 + b_{out}$
$\frac{\partial O}{\partial Z_{out}}$
$O = \sigma(Z_{out})$
$\frac{\partial E}{\partial O}$
$E = Loss(O)$

Fig. 3. updating weights and bias at output layer

$$\frac{\partial \boldsymbol{E}}{\partial \boldsymbol{W}_{out}} = \frac{\partial \boldsymbol{Z}_{out}}{\partial \boldsymbol{W}_{out}} \times \left[\frac{\partial \boldsymbol{O}}{\partial \boldsymbol{Z}_{out}} \cdot \frac{\partial \boldsymbol{E}}{\partial \boldsymbol{O}}\right]^T$$

and

$$\frac{\partial \boldsymbol{E}}{\partial \boldsymbol{b}_{out}} = \frac{\partial \boldsymbol{b}_{out}}{\partial \boldsymbol{W}_{out}} \times \left[\frac{\partial \boldsymbol{O}}{\partial \boldsymbol{Z}_{out}} \cdot \frac{\partial \boldsymbol{E}}{\partial \boldsymbol{O}}\right]^T$$

where,

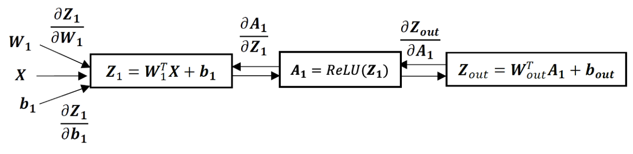
$$\frac{\partial \boldsymbol{E}}{\partial \boldsymbol{O}} = -(\boldsymbol{Y} - \boldsymbol{O}), \quad \frac{\partial \boldsymbol{O}}{\partial \boldsymbol{Z}_{out}} = \boldsymbol{O}(1-\boldsymbol{O})$$
$$\frac{\partial \boldsymbol{Z}_{out}}{\partial \boldsymbol{W}_{out}} = \boldsymbol{A}_1, \quad \frac{\partial \boldsymbol{Z}_{out}}{\partial \boldsymbol{b}_{out}} = \mathbf{1}$$

$\times$ represent dot product and $\cdot$ represent element wise multiplication.
Then we can update weights using

$$\boldsymbol{W}_{out} \rightarrow \boldsymbol{W}_{out} - \eta(\frac{\partial \boldsymbol{E}}{\partial \boldsymbol{W}_{out}} + \lambda \|\boldsymbol{W}_{out}\|_2)$$

$$\boldsymbol{b}_{out} \rightarrow \boldsymbol{b}_{out} - \eta(\frac{\partial \boldsymbol{E}}{\partial \boldsymbol{b}_{out}} + \lambda \|\boldsymbol{b}_{out}\|_2)$$

(b) *Updating weights and bias at hidden layers*: Our goal is to update the weight($\boldsymbol{W}_{out}$) and bias($\boldsymbol{b}_{out}$) of the hidden layer (figure 4). As the activation function is $ReLU$, the values of $\frac{\partial \boldsymbol{E}}{\partial \boldsymbol{W}_1}$ and $\frac{\partial \boldsymbol{E}}{\partial \boldsymbol{b}_1}$ can be calculated as



$\frac{\partial Z_1}{\partial W_1}$
W_1
X
b_1
$\frac{\partial Z_1}{\partial b_1}$
$Z_1 = W_1^T X + b_1$
$\frac{\partial A_1}{\partial Z_1}$
$A_1 = ReLU(Z_1)$
$\frac{\partial Z_{out}}{\partial A_1}$
$Z_{out} = W_{out}^T A_1 + b_{out}$

Fig. 4. updating weights and bias at output layer

$$\frac{\partial \boldsymbol{E}}{\partial \boldsymbol{W}_1} = \frac{\partial \boldsymbol{Z}_1}{\partial \boldsymbol{W}_1} \times \left[\frac{\partial \boldsymbol{A}_1}{\partial \boldsymbol{Z}_1} \cdot \frac{\partial \boldsymbol{Z}_{out}}{\partial \boldsymbol{A}_1} \times \left[\frac{\partial \boldsymbol{O}}{\partial \boldsymbol{Z}_{out}} \cdot \frac{\partial \boldsymbol{E}}{\partial \boldsymbol{O}}\right]\right]^T$$

and

$$\frac{\partial \boldsymbol{E}}{\partial \boldsymbol{b}_1} = \frac{\partial \boldsymbol{Z}_1}{\partial \boldsymbol{b}_1} \times \left[\frac{\partial \boldsymbol{A}_1}{\partial \boldsymbol{Z}_1} \cdot \frac{\partial \boldsymbol{Z}_{out}}{\partial \boldsymbol{A}_1} \times \left[\frac{\partial \boldsymbol{O}}{\partial \boldsymbol{Z}_{out}} \cdot \frac{\partial \boldsymbol{E}}{\partial \boldsymbol{O}}\right]\right]^T$$

where,

$$\frac{\partial \boldsymbol{E}}{\partial \boldsymbol{O}} = -(\boldsymbol{Y} - \boldsymbol{O}), \quad \frac{\partial \boldsymbol{O}}{\partial \boldsymbol{Z}_{out}} = \boldsymbol{O}(1 - \boldsymbol{O}),$$
$$\frac{\partial \boldsymbol{Z}_{out}}{\partial \boldsymbol{A}_1} = \boldsymbol{W}_{out}, \quad \frac{\partial \boldsymbol{A}_1}{\partial \boldsymbol{Z}_1} = \mathbf{I}_{x>0},$$
$$\frac{\partial \boldsymbol{Z}_1}{\partial \boldsymbol{W}_1} = \boldsymbol{X}, \quad \frac{\partial \boldsymbol{Z}_1}{\partial \boldsymbol{b}_1} = \mathbf{1}.$$

we can update weights using

$$\boldsymbol{W}_1 \to \boldsymbol{W}_1 - \eta(\frac{\partial \boldsymbol{E}}{\partial \boldsymbol{W}_1} + \lambda \|\boldsymbol{W}_1\|_2)$$

$$\boldsymbol{b}_1 \to \boldsymbol{b}_1 - \eta(\frac{\partial \boldsymbol{E}}{\partial \boldsymbol{b}_1} + \lambda \|\boldsymbol{b}_1\|_2)$$

B. XGBoost

Extreme Gradient Boosting (XGBoost) is a widely acclaimed and potent machine learning technique renowned for its efficiency, speed, and remarkable predictive capabilities. Falling into the ensemble learning category, XGBoost harnesses the collective wisdom of multiple base models, usually decision trees, to generate a final accurate and robust prediction. However, owing to its potency, optimizing XGBoost through hyperparameter tuning can be intricate and computationally demanding, particularly for deep and intricate models. As with any machine learning algorithm, the efficacy of XGBoost heavily relies on the quality and relevance of the features utilized and the careful calibration of hyperparameters tailored to the specific problem at hand.

C. Feature Extraction Functions

(a) Shortest path distance.
The possibility of friends of a friend becoming friends suggests that the path distance between nodes in a social network can influence link formation. Smaller path distances imply a higher likelihood of link creation. However, it is essential to consider the small-world phenomenon [1], as only a few vertices separate most node pairs in social networks. Consequently, this feature may only sometimes perform well in link prediction tasks. In a study by Hasan et al. [2] focusing on link prediction in a biological co-authorship network, this feature ranked, on average, at 4 out of 9 features used. A similar finding of limited effectiveness for this feature was also noted in another study. [3].

(b) Jaccard Coefficient
An alternative to the non-normalized common neighbors metric is the Jaccard Coefficient, which offers normalization by considering the ratio of the size of common neighbors to the total number of unique neighbors between two nodes. This normalization enhances the comparability of similarity scores across different node pairs. Note that:

$$\text{Jaccard-coefficient}(x, y) = \frac{|\Gamma(x) \cap \Gamma(y)|}{|\Gamma(x) \cup \Gamma(y)|} \tag{1}$$

The Jaccard Coefficient quantifies the likelihood of selecting a common neighbor between two vertices, x and y, if neighbors are randomly chosen from the combined neighbor sets of x and y. Thus, a higher score is obtained when there are more common neighbors.

(c) Sorensen Index
The Sorensen Index (in equation 2), proposed by Thorvald Sorensen in 1948, is a similarity index primarily used for analyzing ecological data samples [5]. It bears a close resemblance to the Jaccard.

$$\text{Sorensen Index } (x, y) = \frac{2|\Gamma(x) \cap \Gamma(y)|}{k_x + k_y} \tag{2}$$

McCune et al. [4] demonstrated that the Sorensen Index exhibits greater robustness against outliers compared to the Jaccard index.

(d) Salton Index
To compute document similarities, the Salton index, also known as Cosine similarity, is employed [6]. This similarity index measures the similarity between two records by calculating their Cosine similarity. The Cosine similarity metric focuses solely on the orientation of the vectors and disregards their magnitudes.

$$\text{Salton Index } (x, y) = \frac{|\Gamma(x) \cap \Gamma(y)|}{\sqrt{k_x . k_y}} \tag{3}$$

(e) Common Field (CF)
We deduced this function by considering the node features of the dataset. For each node it's field or class were provided, and using this information we construct the function

$$\text{CF}(x, y) = \begin{cases} 1 & \text{if } field(x) = field(y) \\ 0 & otherwise \end{cases} \tag{4}$$

V. Experimental Setup

We conduct experiments on the CiteSeer benchmark dataset, which consist of three essential steps and ten iterations.

Step 1. Data preprocessing: In the first step, we preprocess the CiteSeer benchmark dataset to ensure its suitability for conducting experiments. Let us denote by $G = (V, E)$ the graph of the citation network graph of CiteSeer, where V is the vertex set deduced from G and E is the edge set. We name E the positive edges list. Then we generate the same number of non-existing node pairs as the rank of E, denoted E' and called negative edges list, generated from the nodes in V. Afterwards, we randomly separate each E and E' into training, validation and test set.

Step 2. Feature Extraction: In this step, we will extract the feature vectors employing the feature extraction functions: shortest path distance, Jaccard coefficient, Sorensen index, Salton index and Common Field, over the training graph G_{train}, created from positive training edges list for each pair of nodes in E and E'. The outputs of each function for each pair are put together in a vector, producing in this way $5-$dimensional feature vectors.

Step 3. AUC results: In this step, we use our self-constructed Neural Network code and xgboost (xgb) to get the AUC results, by finding the best hyper-parameters over the validation set and then utilizing them to find the AUC over the test set.

VI. Results

Table I and II shows the results we had from our experiments. We represented in each table the hyper-parameters for which the AUC over the validation is the highest, which is the Valid. AUC value. The Test AUC shows the AUC values we got by fitting the concatenation of validation and train set into our model with the hyper-parameters we got from validation set and predicting over the test set. These values are given for each iteration.

TABLE I
Link Prediction Performance with Neural Network

Itr	n_1	n_2	Epochs	λ	η	Valid. AUC	Test AUC
0	2	2	50	1e-6	0.001	86.87	86.80
1	8	4	100	1e-5	0.001	88.02	70.63
2	2	8	100	1e-5	0.001	87.71	87.94
3	8	4	100	1e-5	0.001	86.68	85.05
4	8	8	50	1e-5	0.001	85.01	70.66
5	8	8	100	0	0.001	89.20	85.68
6	8	4	50	1e-5	0.001	88.93	70.63
7	4	4	100	0	0.001	89.31	87.82
8	2	4	50	1e-6	0.01	87.16	69.64
9	2	2	100	1e-6	0.001	87.30	85.75

Itr: Number of iterations performed, $\mathbf{n_1}$: Number of neurons in the first hidden layer, $\mathbf{n_2}$: Number of neurons in the second hidden layer, **Epochs:** Number of training epochs, λ : Regularization parameter, η : Learning rate, **AUC:** Area Under the Curve (performance metric)

For neural networks we tune the hyper-parameters on the following values: $\lambda = [0, 1e{-}5, 1e{-}6]$, $\eta = [0.1, 0.01, 0.001]$, epochs$= [10, 50, 100]$, $n_1 = [2, 4, 8]$, $n_2 = [2, 4, 8]$.

TABLE II
Link Prediction Performance with XGboost

Itr	md	η	subs	cb		Valid. AUC	Test AUC
0	5	0.1	0.1	0.5	300	87.71	87.90
1	5	0.2	0.1	0.5	100	88.36	88.33
2	3	0.5	0.1	0.1	200	88.95	88.22
3	3	0.1	0.1	0.5	100	87.79	86.32
4	3	0.1	0.5	0.1	100	85.64	87.35
5	3	0.1	0.5	0.8	100	89.40	87.07
6	3	0.1	0.5	0.1	100	90.16	87.98
7	3	0.5	0.1	0.8	200	89.80	88.37
8	3	0.1	0.1	0.8	100	87.96	88.26
9	5	0.2	0.1	0.5	100	88.43	86.14

Itr: Number of iterations performed, **md:** max depth, η : Learning rate, **subs:** subsample, **cb:** colsample bytree, **nest:** n estimators, **AUC:** Area Under the Curve (performance metric)

For xgboost we tune the hyper-parameters on the following values: max depth$= [3, 5, 7]$, $\eta = [0.1, 0.2, 0.5]$, subsample$=$ $[0.1, 0.5, 0.8]$, colsample bytree$= [0.1, 0.5, 0.8]$, n estimators$=$ $[100, 200, 300]$.

In this study, we have conducted an extensive comparative analysis to evaluate the performance of our proposed method in comparison to a range of advanced graph neural network models. Specifically, we assess the effectiveness of our approach by comparing its results on the test dataset with those achieved by Node2Vec [17], MVGRL (Multi-View Graph Representation Learning) [18], VGAE (Variational Graph Autoencoder) [19], SEAL (Semi-Supervised Graph Classification with Graph Neural Networks) [20], LGLP (Label-Graph Propagation) [21], GCN (Graph Convolutional Network) [10], GSAGE (GraphSAGE) [22], JKNET (Jumping Knowledge Network) [23], and three hybrid variants, namely CFLP w/ GCN, CFLP w/ GSAGE, and CFLP w/ JKNET [7].

The AUC (Area Under the Receiver Operating Characteristic Curve) metric was used as an evaluation measure for comparison. In our experiments, we utilized the xgboost algorithm and achieved a commendable sixth ranking in terms of AUC performance, as shown in Table III. It is worth mentioning that our approach employed compact five-dimensional features, highlighting the effectiveness of our method despite its lower dimensionality compared to more complex GNN models. Our research outcomes make a valuable contribution to the wider field of link prediction studies and offer promising prospects for the advancement of high-performing and efficient methods in graph learning.

TABLE III
Link prediction performances measured AUC.

By AUC	CITESEER
Node2Vec	80.00 ± 0.68
MVGRL	61.20 ± 0.55
VGAE	85.35 ± 0.60
SEAL	85.82 ± 0.44
LGLP	89.41 ± 0.13
GCN	71.47 ± 1.40
GSAGE	87.38 ± 1.39
JKNet	88.58 ± 1.78
CFLP w/ GCN	89.65 ± 0.20
CFLP w/ GSAGE	91.84 ± 0.20
CFLP w/ JKNET	92.12 ± 0.47
Our model (NN)	80.06 ± 7.94
Our model (xgb)	87.59 ± 0.79

VII. Conclusion

In this research, we extracted the feature vectors employing the feature extraction functions: shortest path distance, Jaccard coefficient, Sorensen index, Salton index and Common Field over the graph deduced by CiteSeer benchmark dataset. Our investigation into link prediction has yielded encouraging outcomes, surpassing the performance of many other GNN methods. Remarkably, we achieved this using only a modest set of five meticulously chosen features. Our forthcoming research seeks to elevate link prediction efficacy through the inclusion of more significant attributes, permitting deeper comprehension of network dynamics and enhancing predictive precision.

References

[1] Al Hasan, M., Chaoji, V., Salem, S., Zaki, M. Link prediction using supervised learning, Applied Mechanics and Materials, vol.30, 2015

[2] Li, M., Ou, R. Q., Song, Y. L. Principal-Agent Activities on Small World Network, Applied Mechanics and Materials, vol.727, pp.798–805, 2006.

[3] Liben-Nowell, D., Kleinberg, J. The link prediction problem for social networks, Applied Mechanics and Materials, vol.727, pp.556–559, 2003.
[4] McCune, B., Grace, J., Analysis of ecological communities, 2002
[5] Sørensen, Th. J. A method of establishing groups of equal amplitude in plant sociology based on similarity of species content and its application to analyses of the vegetation on Danish commons, I kommission hos E. Munksgaard, 1948.
[6] Salton, M., Introduction to Modern Information Retrieval, McGraw-Hill, Inc., New York, NY, USA, 1986.
[7] Zhao, T., Liu, G., Wang, D., Yu, W., Jiang, M. Learning from counterfactual links for link prediction, In International Conference on Machine Learning, pp. 26911-26926. PMLR, 2022.
[8] Zhao, T. Learning to Augment Data in Graphs. University of Notre Dame, 2022.
[9] Choi, J., Ko, T., Choi, Y., Byun, H., Chong-kwon, K. Dynamic Graph Convolutional Networks with Attention Mechanism for Rumor Detection on Social Media. PLoS One 16 (8): e0256039, 2021.
[10] Kipf, T. N., Welling, M. Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907, 2016.
[11] Cen, Y., Zou, X., Zhang, J., Yang, H., Zhou, J., Tang, J. Representation learning for attributed multiplex heterogeneous network. In Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining, pp. 1358-1368, 2019.
[12] Hamilton, W. L., Ying, R., Leskovec, J. Representation learning on graphs: Methods and applications. arXiv preprint arXiv:1709.05584 ,2017.
[13] Zhang, Ch., Song, D., Huang, Ch., Swami, A., Chawla, N. V. Heterogeneous graph neural network. In Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining, pp. 793-803. 2019.
[14] Dou, W., Zhang, W., Weng, Z., Xia, Z. Graph Embedding Framework Based on Adversarial and Random Walk Regularization. Ieee Access, 9, 1454-1464, 2021. https://doi.org/10.1109/access.2020.3047116
[15] Wang, L., Chen, C., Li, H. Link Prediction of Complex Network Based on Eigenvector Centrality. Journal of Physics Conference Series, 2337(1), 2022. https://doi.org/10.1088/1742-6596/2337/1/012018
[16] Wang, T., Wang, H., Wang, X. CD-Based Indices for Link Prediction in Complex Network, 2016. https://scite.ai/reports/10.1371/journal.pone.0146727
[17] 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.
[18] Hassani, K., Khasahmadi, A. H. Contrastive multi-view representation learning on graphs. In International Conference on Machine Learning, pp. 4116–4126. PMLR, 2020.
[19] Kipf, T. N., Welling, M. Variational graph auto-encoders. arXiv preprint arXiv:1611.07308, 2016.
[20] Zhang, M., Chen, Y. Link prediction based on graph neural networks. In Advances in Neural Information Processing Systems, 2018.
[21] Cai, L., Li, J., Wang, J., Ji, S. Line graph neural networks for link prediction. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2021.
[22] Hamilton, W. L., Ying, R., Leskovec, J. Inductive representation learning on large graphs. arXiv preprint arXiv:1706.02216, 2017
[23] Xu, K., Li, C., Tian, Y., Sonobe, T., Kawarabayashi, K.-i., Jegelka, S. Representation learning on graphs with jumping knowledge networks. In International Conference on Machine Learning, pp. 5453–5462, 2018.