Fea2Fea: Exploring Graph Feature Correlations via Graph Neural Networks

Anonymous

No Institute Given

Abstract. Node features are of high dimension. There is no relevant research on exploring node feature correlation on graphs to reduce dimensionality before applying graph neural network models. In this paper, we propose graph feature to feature (**Fea2Fea**) prediction pipeline models in a low dimensional space to explore some preliminary results. The results show that there exists high correlation between certain features where we need to remove redundancy features before training. We also compare the difference between concatenation methods on connecting graph embeddings between features. We tuned the hyper-parameters to find some trends in common features in node classification tasks. We generalize on the synthetic geometric graphs and certify parts of our assumptions. Transfering to normal node or graph classfication with input feature filtering by the pipeline models is in progress.

Keywords: Graph neural networks · feature engineering

1 Introduction

Node and graph datasets are commonly high dimensional. Implementing a model based on graph neural networks to solve node or graph classification problems is highly related with model parameters [9], which include input feature dimensions. Studies have that optimal time complexity of a graph neural network model is related to node hidden features [9], but no relevant research has explained how input node features are correlated with each other and whether a pre-selection of input node features via feature correlation will improve the classification accuracy. In this case, we define a new *node classification* task called **graph feature to feature prediction**, where graph features refer to structural features of any geometric graph, such as degree, clustering coefficient etc. The aim of the task is to:

- illustrate on input feature correlations with low dimensionality
- filter redundant input features via graph neural network based models
- view the performance of graph neural network on feature prediction
* transfer the proposed methods to traditional node or graph classification problems.

In this paper, we propose a framework for processing graph *feature* to *feature* prediction, called **Fea2Fea**. The framework includes two parts, which are single

feature to single prediction and multiple features to single feature prediction. When predicting a single feature from another single feature, we build a GNN-based model plus a post-multilayer perceptron to explore the feature's mutual correlation and save them in a matrix, where pipeline 1 in part 3.2 gives detailed instructions on it. When predicting a single feature via multiple features, we use a simple threshold mechanism to filter features that are highly related to other input features (search the saved matrix), which also generates all possible feature combinations. Three concatenation methods are mentioned and described in part 3.3. We will achieve the average accuracy for each number of graph input feature dimensions between 2 to 4 via graph neural networks. The second part is mainly described in pipeline 2 in part 3.3. We conducted experiments on benchmark node datasets *Planetoid* [11] and graph datasets *TUDataset* [6] to compare the similarities and differences in predicting feature. We tested different hyper-parameters, such as model depth, number of bins and different threhold values. We compared the results of three concatenation methods. Finally we generated synthetic geometric graphs with different node sizes to generalize on our preliminary conclusions.

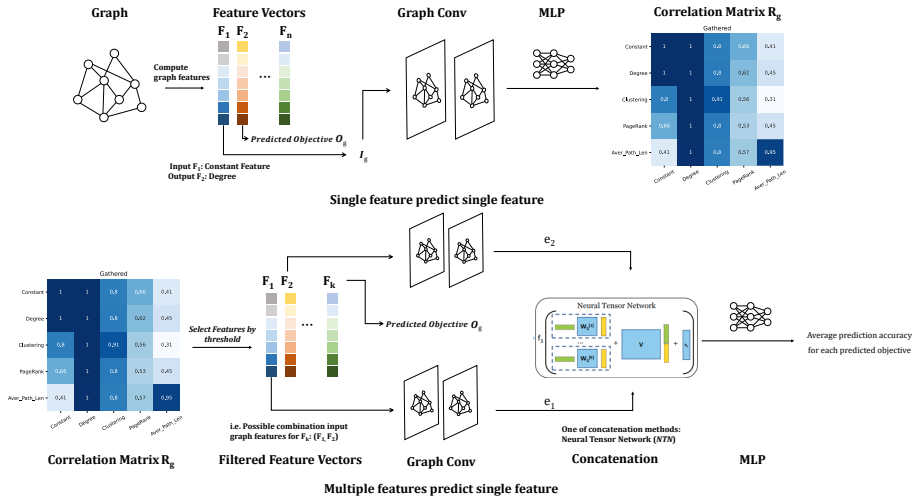

Fig. 1. Pipeline model for feature mutual prediction task, where the neural tensor network is originated in NLP domain [7] and sketch of the NTN block comes from *SimGNN* [1].

2 Related Works

Recent years, many researches have been conducted on learning graph embeddings. Many efficient and powerful graph embedding methods have been

proposed, including graph convolution network(GCN)[4], graph attention network(GAT)[8], graph sampling and aggregating network(graphSAGE)[3] and graph isomorphism network(GIN)[10]. They are regarded as the four main graph convolution methods in the graph neural network block in our model.

No relevant research paper has collected potential graph features and discussed the correlation between each graph feature either by giving closed-form solutions or solutions based on deep neural networks . However some python packages such as **networkx** [2] have provided us with some useful interfaces to easily generate graph features. In addition, **networkx** can help us construct some randomly generated synthetic geometric graphs, which can be encapsulated into torch_geometric dataset. Speaking of feature concatenation methods, we mainly use neural tensor networks (**NTN**)[7] which is originally used in natural language processing models. It includes the idea of simple concatenation and bilinear concatenation,also known as inner product. These two methods are also implemented in this paper to see if they have an advantage or disadvantage compared to **NTN** method.

3 Methods

3.1 preliminary graph feature extraction

Given a graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$. We want to implement a full-scale feature matrix $\mathbf{x}_{\mathcal{G}} \in \mathbb{R}^{|\mathcal{V}| \times \mathcal{D}}$ from $\mathcal{G}$ which requires an adjacency matrix $\mathcal{A}_g \in \mathbb{R}^{|\mathcal{V}|*|\mathcal{V}|}$ of $\mathcal{G}$, where $\mathcal{V}$ is the vertex set, $\mathcal{E}$ is the edge set of the graph and $\mathcal{D}$ is the total number(input dimension) of graph features. In this paper we mainly choose five of graph features to show exploratory results, which are constant feature ($Cons$), degree (Deg), clustering coefficient (Clu), average path length ($AvgLen$) and PageRank (PR) . Therefore $\mathcal{D}$ is equal to 5.

Constant feature of one node u $\underset{u\in\mathcal{V}}{Cons}(u)$ is given by c, where c $\in \mathbb{R}^+$. Each node's constant feature is set to 1 in this paper for standardization. Degree of node u $\underset{u\in\mathcal{V}}{Deg}(u)$ is equal to the number of node u's neighbours. Clustering coefficient of node u $\underset{u\in\mathcal{V}}{Clu}(u)$ is given by $\frac{2e_{jk}}{k_i*(k_i-1)}$, where $j, k \in \mathcal{V}$ and e_{jk} represents the total possible edges between node u's neighbours and k_i is the number of node u's neighbours. Pagerank of a node u $\underset{u\in\mathcal{V}}{PR}(u)$ is given by: $\underset{u\in\mathcal{V}}{PR}(u) = \frac{1-q}{|\mathcal{V}|} + q \times \sum_{v\in\mathcal{N}(u)} \frac{PR(v)}{\mathcal{L}(v)}$, where $\mathcal{N}(\cdot)$ means the node u's neighbours and $\mathcal{L}(\cdot)$ means numbers of outbound link from node u to its neighbours. In the traditional analysis of pagerank algorithm, the object is a directed graph. However we can treat undirected graph as bidirectional graph, so outbound link is equal to the number of edges from node u to its neigbours. q is the residual probability and equal to 0.85. Average path length of a node $\underset{u\in\mathcal{V}}{Avglen}(u)$ is given by: $\underset{u\in\mathcal{V}}{Avglen}(u) = \frac{1}{\mathcal{V}'} \sum_{v\in\mathcal{V}\neq u} I(u,v) * d_{min}(u,v)$, where $I(u, v)$ indicates whether there's a path from node u to node v. If node v is reachable for node u, then $I(u, v)$ is equal to 1 otherwise it is 0. $\mathcal{V}'$ is equal to the total number of reachable nodes for node u, more specifically, $\mathcal{V}'$

$= \sum_{v \in \mathcal{V} \neq u} I(u, v)$. $d_{min}(u, v)$ determines the shortest path length from node u to node v. Therefore, a complete entry $\mathbf{I}(v), v \in \mathcal{V}$ can be written as: $I(v) = Cons(v) \oplus Deg(v) \oplus Clu(v) \oplus PR(v) \oplus Avglen(v)$ where $\oplus$ is the feature symbolic link function(concatenation). If we have $\mathcal{V}$ vertices of a graph, then the full-scale feature matrix $\mathbf{x}_{\mathcal{G}} \in \mathcal{R}^{|\mathcal{V}|*5}$.

3.2 pipeline 1: Single feature predict single feature

Algorithm 1(refer to appendix) shows the complete process of how to construct a correlation matrix for feature mutual prediction. Suppose we have successfully computed a feature matrix $\mathbf{x}_{\mathcal{G}} \in \mathbb{R}^{|\mathcal{V}| \times \mathcal{K}}$. In particular, $\mathcal{K}$ is equal to 5 here since five of graph features are researched. Importantly graph features have been indexed to simplify analysis. The feature order is given by constant feature, degree, clustering coefficient, pagerank and average path length, which are indexed from 1 to 5 respectively. Suppose we have built a model $\mathcal{M}$ which includes one of the four selected graph embedding methods together with multi-layer perceptrons(part 2). Given a feature, we can find its column index from $\mathbf{x}_{\mathcal{G}}$ and extract them by taking that column, such as $\mathbf{x}_{\mathcal{G}}(:, 1)$ for constant feature and $\mathbf{x}_{\mathcal{G}}(:, 4)$ for average path length. We extract two columns specifically, one for input $\mathbf{I}_{\mathcal{G}}$ and the other for output $\mathbf{O}_{\mathcal{G}}$. We set bins for the output since we focus more on classification task rather than regression task(part 1). After outputs are classified into each of its own classes, we use model $\mathcal{M}$ to predict $\mathbf{O}_{\mathcal{G}}$. The trained model will generate predictive output $\mathbf{O}_{\mathcal{G}}^{'}$. We use classification accuracy metrics as the main evaluator in this paper. Micro-$\mathcal{F}_1$ and macro-$\mathcal{F}_1$ can also be metrics, but we only write $\mathcal{P}$ to indicate there are many interesting metrics to evaluate the model performance. Better model performance indicates that it is easier to predict from input feature to output feature. Finally we will implement a feature correlation matrix $\mathcal{R} \in \mathbb{R}^{5*5}$. We should note that constant feature can be only taken as input but not as output. It is the most basic feature for all graphs and will not lead to a classification problem.

3.3 pipeline 2: Mutiple features predict single feature

We obtain feature correlation matrix from pipeline 1. The value $\mathcal{R}(i, j)$ in the matrix indicates whether feature f_i and feature f_j can be easy or difficult to predict each other. Moreover, we want to add additional features to f_i to see if it will predict f_j more accurately. However we cannot add features arbitrarily since adding similar features may cause data redundancy. Therefore we need to collect all possible feature combinations without redundant information by applying a threshold mechanism. Given a threhold t, we filter out and discard such feature combination : (f_i, f_j) where $\mathcal{R}(i, j)$ is greater than t. Consider two extreme cases. All feature combinations are considered for training when t is equal to 1 while no feature combinations are considered when t is equal to 0. When t $\in$(0,1), the number of possible concatenations $\mathcal{N} \in$[1,$2^d - 1$).

We initialize an array called *Comb* which includes all possible concatenation between features. The time complexity of this generation is $\Omega(2^d)$, where d

is equal to the input feature dimension. It works efficiently when input feature dimension is small. When it comes to a practical high dimensional dataset,espeically when d is greater than 1000, we can apply *reservoir sampling* to reduce time complexity. The filtering work is applied after generating the array. For each f_i and f_j in each element of $Comb$, if both $\mathcal{R}(i,j)$ and $\mathcal{R}(j,i)$ are less than threshold t, then the combination of f_i and f_j is valid, otherwise this combination is moved from $Comb$. Finally we achieve a filtered $Comb$. The choice of t is very important. If we set a low threshold such as 0.1, unfortunately, almost no feature correlation meets this threshold, resulting in no feature combination. In addition, the threshold should not be too low, because a high threshold will consider all the combinations.

Given a valid combination [f_1, f_2,...,f_k], where k $\in$\{2,3,4\} and the combination order has nothing to do with the initial feature order. We implement graph convolution layers for each feature to map them into graph embeddings: [e_1, e_2,...,e_k]. We provide three methods to concatenate those features after graph embeddings, which are simple concatenation, biliear concatenation and neural tensor network(NTN) concatenation. Assume that the graph embedding space of each feature is d. A simple concatenation is like this:

$$g_{(t)} = g_{(t-1)} \oplus e_t, g_{(0)} = e_1 \in \mathbb{R}^d \tag{1}$$

where $g_{(t)} \in \mathbb{R}^{td}$ represents the temporal embedding after t-th concatenation, $\oplus$ means the function that connects two features together with the summation of their dimensions. $g_{(0)}$ is initialized by e_1. The advantage is that the time complexity and computational cost are low, but important interactive information between graph features may be lost. Consider bilinear concatenation which is given by:

$$g_{(t)} = g_{(t-1)}^T \mathbf{W}_{(t)} e_t, g_{(0)} = e_1 \in \mathbb{R}^d \tag{2}$$

where the weight matrix $\mathbf{W}_{(t)} \in \mathbb{R}^{(t-1)d*td*d}$ with t$\geq$2 and obtained graph embedding $g_{(t)} \in \mathbb{R}^{td}$. Another concatenation method neural tensor network(NTN) [7] is based on both simple concatenation and bilinear product, which is given by:

$$g_{(t)} = \mathbf{u}^T \mathbf{h}(g_{(t-1)}^T \mathbf{W}_{(t)} e_t + g_{(t-1)} \oplus e_t + \mathbf{b}), g_{(0)} = e_1 \in \mathbb{R}^d \tag{3}$$

where weight vector $\mathbf{u}^T \in \mathbb{R}^{td}$ and $\mathbf{h}$ represents the activation *tanh* function as the same as the activation function in original paper. However, we cannot conclude that the neural tensor network method is superior to the other two connection methods. We compare the training and test performance of three methods. After the cascade is a multilayer perceptron model with a log softmax output layer. The pipeline will achieve the average accuracy of each combination, as well as the average accuracy of different numbers of input dimensions. The details of experiments are described in part 4.

4 Experiments

4.1 Datasets

In our preliminary experiments, we selected six benchmark datasets from graph database. Generation of supervised geometric graphs is also included. For node datasets, we choose *Planetoid*[3], which are citation datasets including CORA, CITESEER and PUBMED. The feature matrix of each dataset is given by sparse bags-of-words vectors. For graph datasets, we choose from *TUDataset*[6]. It is a collection of benchmark datasets including data from small molecules, bioinformatics, social networks, computer vision and some synthetic datasets. In this paper, we choose two datasets PROTEINS and ENZYMES from bioinformatics domain and NCI1 from molecule domain. After generating graph feature matrix for each dataset, we plot the distribution for each graph feature to search for imbalance (shown in the appendix). According to the distribution for *Planetoid* dataset, *degree* and *clustering coefficient* features will cause class inbalance. Therefore, we divide the unbalanced data into one bin, and the other data into remaining bins, especially more than half of the nodes' *clustering coefficient* are zero. *NCI1* dataset in *TUDataset* has a worse situation where nearly all of the nodes in the graph have a zero *clustering coefficient*. One solution is to remove this feature from the feature set. There might be other solutions but the main idea is that we should pay attention to the distribution of features before implementing the graph neural network model.

4.2 Experiment set-up

For pipeline 1, we used four graph convolution methods: **GCN**, **GraphSAGE**, **GAT** and **GIN** included in GNN block, followed by multilayer-perceptrons. We set the defualt numbers of graph model depth $d = 2$ and hidden embedding size $h = 64$.We record the feature correlation matrix $\mathcal{R}$. The experiments were carried out by 10 times and we took mean accuracy as the metrics for how features are correlated with each other. Loss function is based on crossentropy loss. We perform hyperparameter tests on binning method and model depth to record optimal number of bins and depth, where the number of bins $\mathcal{B} \in \{2, 3, 4, 5, 6, 7, 8, 9, 10\}$ and model depth $\mathcal{D} \in \{2, 4, 6, 8, 10\}$. We consider adding batch-norm layers when peforming model depth tests. We added a pure multi-layer perceptron baseline model without graph convolution(**MLP**) for comparison.

For pipeline 2, we used the optimal graph convolution method from pipeline 1 with the same graph model depth and hidden embedding size. Default threshold $\mathcal{T}$ is set to 0.85. We can regard it as a hyper-parameter, for example $\mathcal{T} \in \{0.6, 0.8\}$ to make comparison. For neural tensor network, the hidden dimension and number of neurons(output dimsension) are equal to 64. We compare three concatenation methods with showing their corresponding test and training accuracy on predicting certain graph feature. We achieve each combination results for each predicted objective feature under a threhold and average accuracy for

a certain number of input graph features. Additionally, we add some synthetic graph datasets to generalize on our initial conclusion. We split node datasets into training, validation and test dataset with a fixed ratio in original setting[3]. We split graph dataset into traing, validation and test dataset with a ratio of 8:1:1.

In short, the purposes of the experiments are to: 1) compare the performance between baseline model and gnn based models on node and graph datasets 2) the difficulty of predicting features, i.e. which features are easy/hard to predict with the gnn model 3) achieve numbers of input feature set (dimension) which leads to the best feature prediction results 4) compare concatenation methods with training and test accuracy. 5)adjust hyper-parameters for the model 6) generate geometric graphs for generalization. The generalization and hyper-parameter tunning tests are shown in the appendix.

4.3 Results

Table 1. Feature to Feature Prediction on Cora and Proteins Datasets (bins = 6)

Aim	Cora					Proteins				
	GCN	GIN	SAGE	GAT	MLP	GCN	GIN	SAGE	GAT	MLP
Cons → *Deg*	0.509	**1.000**	0.213	0.202	0.206	0.560	**0.662**	0.469	0.469	0.469
Deg → *Deg*	0.741	1.000	0.967	0.514	**1.000**	0.633	**0.662**	0.688	0.470	0.640
Clu → *Deg*	0.423	**1.000**	0.504	0.474	0.285	0.537	**0.652**	0.482	0.469	0.467
PR →*Deg*	0.308	**1.000**	0.311	0.223	0.197	0.430	**0.662**	0.446	0.469	0.469
Avglen → *Deg*	0.409	**1.000**	0.499	0.228	0.274	0.522	**0.657**	0.482	0.469	0.469
Cons → *Clu*	0.523	**0.533**	0.461	0.461	0.461	0.299	**0.436**	0.202	0.202	0.236
Deg → *Clu*	**0.550**	0.542	0.548	0.506	0.531	0.435	0.387	**0.454**	0.312	0.346
Clu → *Clu*	0.724	0.765	**0.968**	0.668	0.968	0.570	0.610	**0.707**	0.549	0.723
PR → *Clu*	0.490	**0.538**	0.486	0.461	0.461	0.247	**0.330**	0.278	0.219	0.236
Avglen → *Clu*	0.508	**0.538**	0.498	0.460	0.465	**0.276**	0.258	0.328	0.202	0.273
Cons → *PR*	0.639	**0.756**	0.160	0.160	0.160	0.645	**0.648**	0.170	0.170	0.169
Deg → *PR*	0.573	**0.792**	0.750	0.392	0.603	0.622	0.699	**0.722**	0.239	0.461
Clu → *PR*	0.427	**0.695**	0.403	0.199	0.395	**0.575**	0.465	0.467	0.224	0.349
PR → *PR*	0.345	**0.714**	0.323	0.185	0.160	0.170	**0.403**	0.251	0.170	0.176
Avglen → *PR*	0.450	**0.741**	0.490	0.202	0.247	**0.579**	0.453	0.395	0.170	0.175
Cons → *Avglen*	0.357	**0.384**	0.169	0.169	0.169	**0.182**	0.171	0.171	0.171	0.171
Deg → *Avglen*	0.420	0.435	**0.440**	0.340	0.199	**0.212**	0.184	0.200	0.171	0.175
Clu → *Avglen*	0.286	**0.310**	0.263	0.219	0.202	0.227	0.196	**0.254**	0.228	0.216
PR → *Avglen*	0.215	**0.421**	0.266	0.185	0.169	0.171	**0.172**	0.171	0.171	0.171
Avglen → *Avglen*	0.503	0.445	0.774	0.490	**0.958**	0.549	0.483	**0.612**	0.545	0.513

We use the default training, validation and test masks in the *Planetoid*[3]. We use the training, validattion and test ratio of 8:1:1 in *TUdataset*.

model performance In terms of node datasets, *GIN* overall performs the best among all graph embedding method based models and baseline MLP model. The results are presented in table 1 and table 4(appendix). Especially when predicting feature *degree*, *GIN* can reach 100% mean test accuracy for 10 time experiments.*GraphSAGE* and *GCN* can predict features at an acceptable level but they are far from the performance of *GIN*. *GAT* does not perform well in predicting any graph feature. *MLP* can achieve self-prediction very well, but it is very poor without self-prediction. On the graph dataset, *GIN* performs the best while performance of *GraphSAGE* and *GCN* is much better than on node datasets, which may be related to our training-test ratio setting. Results of *GAT* and *MLP* are close to the optimal value at some tasks but generally can not reach the performance provided by *GIN*.

difficulty of prediction From the table 1 or table 4, we can find that feature *degree* is easy to predict whether it's in a node or a graph dataset. Predicting *clustering coefficient* and *average path length* is hard generally for both kinds of datasets. From the difficulty of prediction, we can also explain the relationship between each graph feature. For example, *degree* feature is a basic feature other than *constant feature* which is easier to predict from others. However, *clustering coefficient* is sparse (part 4.1). *average path length* and *pagerank* require the entire information from the whole graph which may bring redundant information.

average accuracy of multi-to-one prediction Threshold value is set to 0.85. We take the simple concatenation to illustrate an example. When predicting *clustering coefficient* from other features in Planetoid dataset, possible combination number of features is only equal to 2 or 3. For Citeseer dataset, a 3-set input is better than a 2-set input. A possible 3-set concatenation for Cora dataset is {*Cons*, *PR*, *Avglen*}, which is much better than a 1-set prediction for each of the feature in this 3-set. When predicting *Pagerank*, the prediction accuracy is not stable, which is shown in the figure 2. A 3-set input features is worse than a 2-set input features for PubMed and Citeseer dataset and even worse than 1-set input feature prediction (pipeline 1). In *TUdataset*, predicting *pagerank* and *average path length* might exist the situation of 4-set features, but this 4-set feature set does not perform well and the average accuracy is descending.

concatenation comparison *NTN* method is worse than simple concatenation in the process of training the predicted *pagerank* of Cora dataset, and it generalizes worse on test datasets. The *bilinear* method faces the problem of over-fitting . In general, the *simple concatenation* has the best performance on test datasets. When predicting *average path length* on *Cora* dataset, *NTN* is far better than *bilinear* and *simple concatenation* but they show similar results on test datasets. More specifically, *Bilinear* and *NTN* have a serious fluctuation problem on accuracy, but overall *NTN* is the optimal method. These experiments indicate that complex concatenation method such as *NTN* does not always have the best performance. Sometimes the simplest way is the best way.

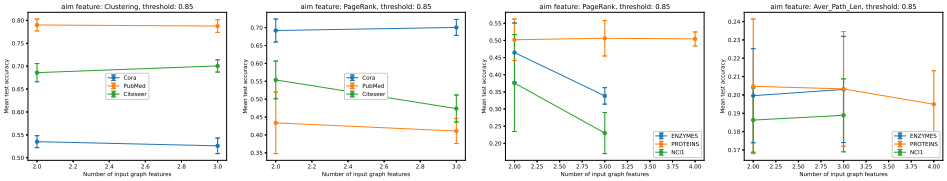

Fig. 2. Different numbers of features contribute to different prediction results for node and graph datasets. We perform 10 experiments and take the average accuracy with standard deviation shown in the figures.

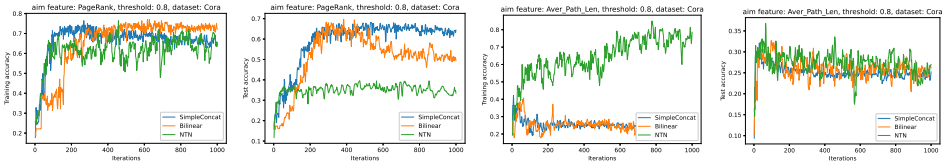

Fig. 3. Training and test accuracy for different concatenation results when predicting clustering coefficient and pagerank in pipeline 2

5 Conclusions and future works

In this paper, we perform graph feature to feature prediction to analyze the correlation between features. We implement multiple features to single feature prediction to judge if redundant features have been removed or it impacts the prediction accuracy. During the prediction we take the advantage of graph embedding methods and feature concatenation method. Experiments show that 1) pre-selection of features is necessary before training 2) *GIN* is overall the best graph embedding method when predicting graph features 3) hyper-parameter adjustment, such as tunning numebrs of bins, model depth and threshold value is important 4)NTN concatenation on graph embeddings is not always better than a simple concatenation 5) the results from the benchmark datasets is also suitable to self-generated synthetic datasets. In our future works, we plan to add more graph features to enrich our feature set since we must find enough graph features to illustrate the trend on mutli-to-one prediction. Also the graph embedding methods should be sufficient to ensure that *GIN* is the best. Besides, we hope to transfer the pipeline model in the future to ordinary node classification task which is located in a higher dimensional space.

References

1. Bai, Y., Ding, H., Bian, S., Chen, T., Sun, Y., Wang, W.: Simgnn: A neural network approach to fast graph similarity computation (2020)

2. Hagberg, A., Swart, P., S Chult, D.: Exploring network structure, dynamics, and function using networkx (1 2008), https://www.osti.gov/biblio/960616
3. Hamilton, W., Ying, Z., Leskovec, J.: Inductive representation learning on large graphs. In: Advances in neural information processing systems. pp. 1024–1034 (2017)
4. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907 (2016)
5. Li, G., Müller, M., Thabet, A., Ghanem, B.: Deepgcns: Can gcns go as deep as cnns? (2019)
6. Morris, C., Kriege, N.M., Bause, F., Kersting, K., Mutzel, P., Neumann, M.: Tu-dataset: A collection of benchmark datasets for learning with graphs (2020)
7. Socher, R., Chen, D., Manning, C.D., Ng, A.: Reasoning with neural tensor networks for knowledge base completion. In: Burges, C.J.C., Bottou, L., Welling, M., Ghahramani, Z., Weinberger, K.Q. (eds.) Advances in Neural Information Processing Systems. vol. 26. Curran Associates, Inc. (2013), https://proceedings.neurips.cc/paper/2013/file/b337e84de8752b27eda3a12363109e80-Paper.pdf
8. Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., Bengio, Y.: Graph attention networks. arXiv preprint arXiv:1710.10903 (2017)
9. Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., Yu, P.S.: A comprehensive survey on graph neural networks. IEEE Transactions on Neural Networks and Learning Systems **32**(1), 4–24 (Jan 2021). https://doi.org/10.1109/tnnls.2020.2978386, http://dx.doi.org/10.1109/TNNLS.2020.2978386
10. Xu, K., Hu, W., Leskovec, J., Jegelka, S.: How powerful are graph neural networks? arXiv preprint arXiv:1810.00826 (2018)
11. Yang, Z., Cohen, W.W., Salakhutdinov, R.: Revisiting semi-supervised learning with graph embeddings (2016)

A Algorithm Pseudocode

Algorithm 1: Get Feature Correlation Matrix

Input: full-scale feature matrix $\mathbf{x}_{\mathcal{G}}$; model architecture $\mathcal{M}$; metrics $\mathcal{P}$
Output: Feature correlation matrix $\mathcal{R}$

$\mathcal{R} \leftarrow \mathbf{0}, \mathcal{K} \leftarrow 5$
for $i \leftarrow 1$ ***to*** $\mathcal{K}$ **do**
  $\mathbf{I}_{\mathcal{G}} \leftarrow \mathbf{x}_{\mathcal{G}}(:, i)$
  for $j \leftarrow 2$ ***to*** $\mathcal{K}$ **do**
    $\mathbf{O}_{\mathcal{G}} \leftarrow \text{Binning}\left(\mathbf{x}_{\mathcal{G}}(:, j)\right)$
    $\mathbf{O}_{\mathcal{G}}^{'} \leftarrow \mathcal{M}\left(\mathbf{I}_{\mathcal{G}}, \mathbf{O}_{\mathcal{G}}\right)$
    $\mathcal{R}(i, j) \leftarrow \mathcal{P}(\mathbf{O}_{\mathcal{G}}^{'}, \mathbf{O}_{\mathcal{G}})$
  end
  if $i == j == \mathcal{K}$ **then**
    return $\mathcal{R}$;
  end
end

Algorithm 2: Get $\mathcal{A}_{\mathcal{G}}$ which records multiple feature to single feature prediction results

Input: Feature Correlation Matrix $\mathbf{x}_{\mathcal{G}}$; model architecture $\mathcal{M}'$; metrics $\mathcal{P}$; output feature index **idx**; threshold $\mathcal{T} = 0.85$
Output: Array $\mathcal{A}_{\mathcal{G}}$

Initialize $Comb \leftarrow \{\{0, 1\}, ..., \{0, 1, 2, 3, \mathcal{K}\}\}$
for $comb \in Comb$ **do**
  for $\forall\ \mathcal{F}_i, F_j \in comb$ **do**
    if $\mathcal{R}(F_i, F_j) \geq \mathcal{T}$ *or* $\mathcal{R}(F_j, F_i) \geq \mathcal{T}$ *or* ***idx*** *in* $comb$ **then**
      Remove $comb$ from $Comb$;
    end
  end
end
for $comb \in Comb$ **do**
  $\mathbf{I}_{\mathcal{G}} \leftarrow \mathbf{x}_{\mathcal{G}}(:, comb)$
  $\mathbf{O}_{\mathcal{G}} \leftarrow \mathbf{x}_{\mathcal{G}}(:, \mathbf{idx})$
  $\mathbf{O}_{\mathcal{G}}^{'} \leftarrow \mathcal{M}^{'}\left(\mathbf{I}_{\mathcal{G}}, \mathbf{O}_{\mathcal{G}}\right)$
  $\mathcal{A}_{\mathcal{G}}(comb|\mathbf{idx}) \leftarrow \mathcal{P}(\mathbf{O}_{\mathcal{G}}^{'}, \mathbf{O}_{\mathcal{G}})$
end

B Hyper-parameter tunning

Three hyper-parameters are nthe number of bins, depth of graph convolution layers and threshold. We evaluated the tasks on *Citeseer* and *ENZYMES* dataset.

Tasks are: from *pagrank* predict *average path length* and from *average path length* predict *clustering coefficient*. The accuracy is high when there're only two bins since the partition of classes is not strict. When the number of bins increases, the average prediction accuracy is descending. When model depth is shallow, the prediction effect is not good. Increasing the number of graph embedding layers with batch normalization will result in a better test generalization. We can also implement *SkipLayerGNN*[5] when layers are deep.

Table 2. Hyper-parameter tests on node and graph datasets

Tasks	Param	Citeseer		Enzymes	
	order	PR→AvgLen	AvgLen→Clu	PR→AvgLen	AvgLen→Clu
Bins	2	**0.774 ± 0.008**	**0.836 ± 0.003**	**0.856 ± 0.028**	**0.550 ± 0.001**
	3	0.727 ± 0.006	0.781 ± 0.003	0.751 ± 0.007	0.433 ± 0.058
	4	0.649 ± 0.010	0.754 ± 0.008	0.642 ± 0.016	0.327 ± 0.053
	5	0.616 ± 0.009	0.722 ± 0.004	0.551 ± 0.004	0.284 ± 0.023
	6	0.584 ± 0.006	0.719 ± 0.002	0.400 ± 0.157	0.294 ± 0.044
	7	0.558 ± 0.008	0.711 ± 0.003	0.390 ± 0.130	0.246 ± 0.002
	8	0.505 ± 0.006	0.703 ± 0.003	0.306 ± 0.158	0.251 ± 0.021
	9	0.491 ± 0.010	0.696 ± 0.002	0.293 ± 0.122	0.262 ± 0.026
	10	0.478 ± 0.008	0.694 ± 0.003	0.180 ± 0.119	0.249 ± 0.024
Depth	2	0.579 ± 0.005	0.720 ± 0.003	**0.505 ± 0.007**	0.281 ± 0.038
	4	0.387 ± 0.057	0.713 ± 0.004	0.237 ± 0.021	0.330 ± 0.050
	6	0.512 ± 0.055	0.707 ± 0.007	0.275 ± 0.032	0.330 ± 0.038
	8	0.638 ± 0.059	0.709 ± 0.006	0.348 ± 0.041	0.336 ± 0.030
	10	**0.651 ± 0.042**	**0.722 ± 0.007**	0.370 ± 0.059	**0.347 ± 0.013**

We set two thresholds: 0.6 and 0.8, which are based on pipeline 2 and experiments are executed 10 times and we obtain average accuracy from the experiments. When threshold value is equal to 0.6, valid combined input feature sets are: (*Cons*,*AvgLen*). Mean accuracy is higher than the four combinations when threshold is equal to 0.8. For *ENZYMES* dataset, a higher threshold helps to obtain better results. Therefore, it infers that more feature combinations are not always the best which shows that we must try all combinations.

Table 3. Threshold tests on node and graph datasets on predicting pagerank

Tasks	Param	Citeseer		Enzymes	
	order	Valid Combination	Accuracy	Valid Combination	Accuracy
Threshold	0.6	(0,4)	**0.611 ± 0.039**	(0,2),(0,4),(2,4),(0,2,4)	0.405 ± 0.081
	0.8	(0,2),(0,4),(2,4),(0,2,4)	0.534 ± 0.067	(0,2),(0,4),(1,2),(2,4),(0,2,4))	**0.440 ± 0.079**

C Supplementary Results on Feature Prediction

Table 4. Feature to Feature Prediction on Planetoid Datasets (bins = 6)

Aim	Citeseer					PubMed				
	GCN	GIN	SAGE	GAT	MLP	GCN	GIN	SAGE	GAT	MLP
Cons → *Deg*	0.587	**1.000**	0.379	0.379	0.379	0.678	**0.996**	0.478	0.478	0.478
Deg → *Deg*	0.899	**1.000**	0.994	0.395	1.000	0.725	**1.000**	0.958	0.477	1.000
Clu → *Deg*	0.628	**1.000**	0.631	0.492	0.670	0.643	**1.000**	0.603	0.471	0.574
PR →*Deg*	0.466	**1.000**	0.409	0.376	0.379	0.539	**1.000**	0.600	0.478	0.478
Avglen → *Deg*	0.536	**0.997**	0.686	0.476	0.473	0.696	**1.000**	0.785	0.478	0.524
Cons → *Clu*	0.671	**0.693**	0.658	0.658	0.658	0.799	**0.805**	0.780	0.780	0.780
Deg → *Clu*	0.688	0.714	**0.736**	0.658	0.726	0.794	**0.804**	**0.804**	0.780	0.805
Clu → *Clu*	0.857	0.882	0.980	0.846	**0.992**	0.831	0.839	**0.939**	0.762	0.932
PR → *Clu*	0.672	**0.687**	0.670	0.658	0.658	0.792	**0.805**	0.780	0.780	0.780
Avglen → *Clu*	0.681	**0.696**	0.684	0.658	0.676	**0.794**	0.785	0.790	0.780	0.780
Cons → *PR*	**0.702**	0.671	0.202	0.190	0.190	**0.669**	0.529	0.161	0.141	0.161
Deg → *PR*	0.637	0.750	**0.752**	0.266	0.443	0.564	**0.629**	0.617	0.175	0.565
Clu → *PR*	0.549	**0.575**	0.435	0.279	0.315	0.437	**0.559**	0.409	0.196	0.326
PR → *PR*	0.192	**0.635**	0.415	0.263	0.190	0.478	**0.554**	0.336	0.161	0.161
Avglen → *PR*	0.529	**0.691**	0.602	0.312	0.333	0.541	**0.591**	0.537	0.274	0.263
Cons → *Avglen*	0.442	**0.503**	0.178	0.166	0.173	0.294	**0.394**	0.168	0.168	0.168
Deg → *Avglen*	0.528	0.542	**0.553**	0.285	0.330	0.415	**0.443**	0.437	0.153	0.313
Clu → *Avglen*	**0.420**	0.377	0.387	0.248	0.246	0.296	**0.330**	0.300	0.171	0.184
PR → *Avglen*	0.268	**0.466**	0.207	0.310	0.173	0.316	**0.459**	0.198	0.197	0.168
Avglen → *Avglen*	0.734	0.597	0.937	0.596	**0.979**	0.450	0.378	0.860	0.270	**0.984**

D Graph Embedding Analysis

We extract graph embedding vectors and linear layer embedding vectors to perform tsne visualization and comparsion. When predicting *pagerank* from node *degree* on *Citeseer* dataset, the graph embedding is similiar to the mlp embedding. On *Proteins* dataset, graph embedding cannot seperate classes well compared with mlp embedding. When performing multiple features to single featuere prediction, input embedding indicates that input features are mixed which cannot classify the class well but mlp embedding can seperate the class well. It emphasizes the importance of a MLP after the GNN block.

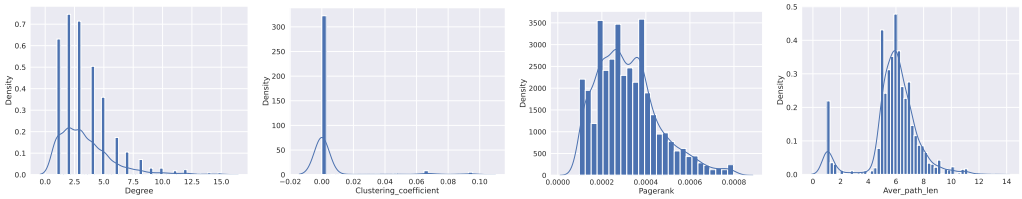

Fig. 4. Distribution of graph feature of Cora Dataset

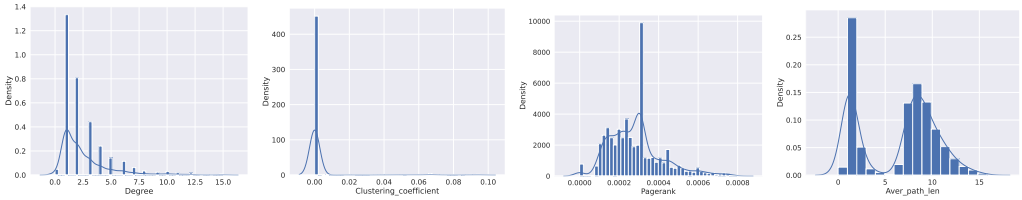

Fig. 5. Distribution of graph feature of Citeseer Dataset

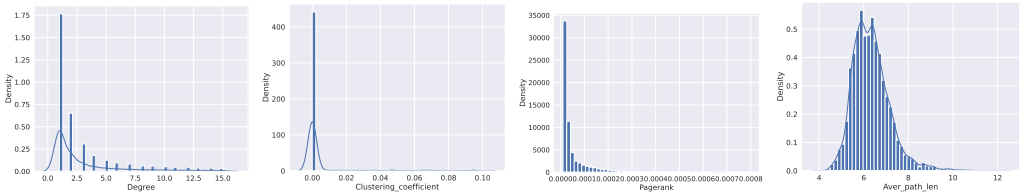

Fig. 6. Distribution of graph feature of Citeseer Dataset

B Hyper-parameter tunning

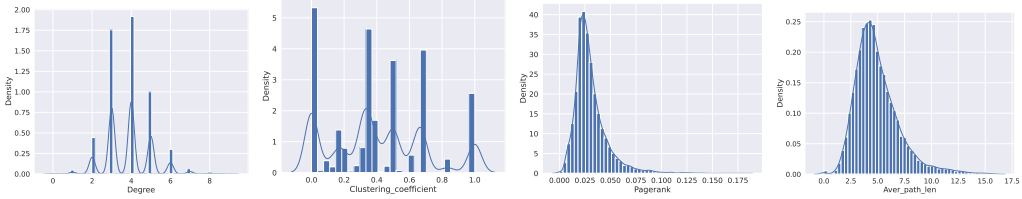

Fig. 7. Distribution of graph feature of ENZYMES Dataset

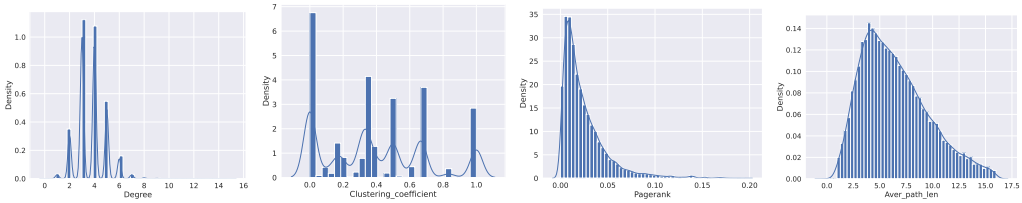

Fig. 8. Distribution of graph feature of PROTEINS Dataset

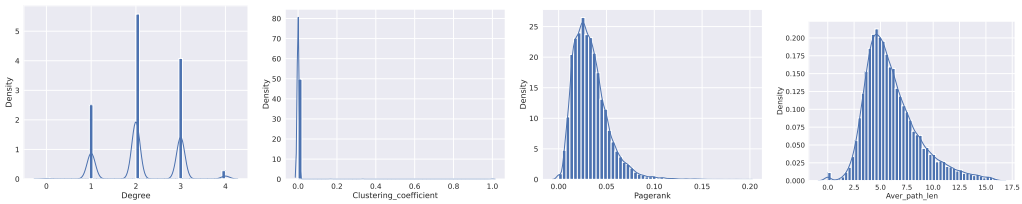

Fig. 9. Distribution of graph feature of NCI1 Dataset

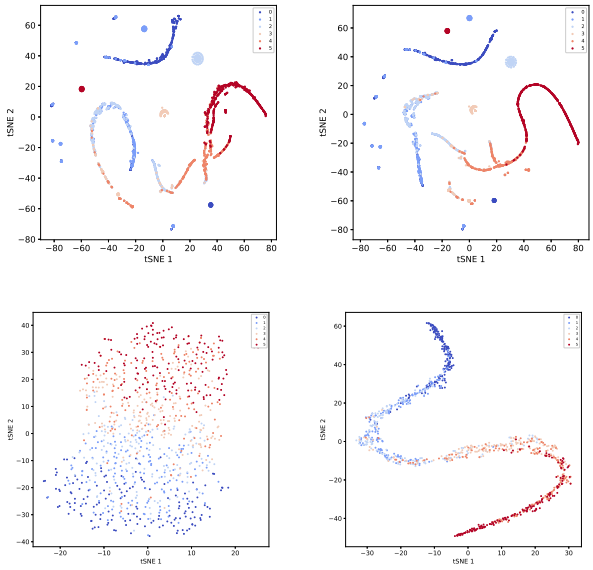

Fig. 10. tSNE on graph and MLP embeddings with Degree predicting PageRank, test on CiteSeer and Proteins datasets(from left to right: graph embedding, MLP embedding, from top to bottom: Citeseer dataset, Proteins dataset)

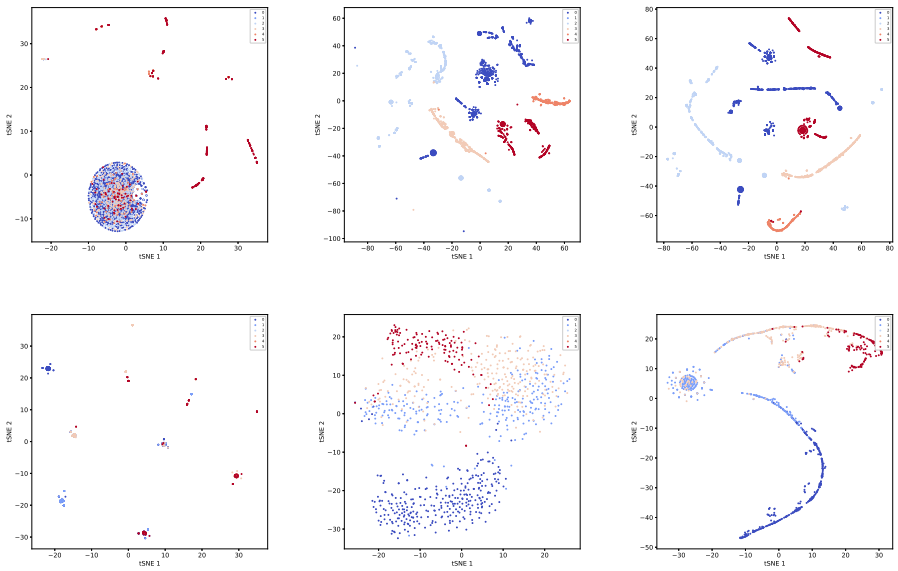

Fig. 11. tSNE on initial, graph and MLP embeddings with Constant feature together with Clustering Coefficient predicting Degree, test on CiteSeer and Proteins datasets(from left to right: initial embedding, graph embedding, MLP embedding, from top to bottom: Citeseer dataset, Proteins dataset)

E generalization

We extended the results that we reached from benchmark datasets on synthetic dataset. We set up five new graph with number of nodes n ∈ {50, 200, 400, 800, 1000}. We found that the feature clustering coefficient is still the most difficult to predict and degree is the easiest to predict, which is the same as what we've previously found in Planetoid and TUdataset. As the number of nodes in the network increases, feature mutual prediction becomes more accurate.

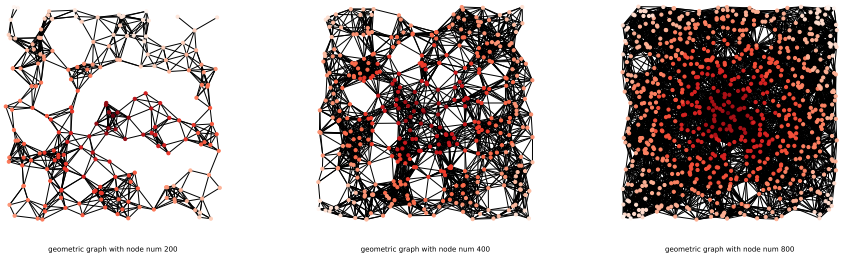

Fig. 12. examples of generated geometric dataset from networkx with nodes 200, 400 and 800

Table 5. Feature to Feature Prediction on Planetoid Datasets (bins = 6)

Aim	Number of nodes				
	50	200	400	800	1000
Cons → *Deg*	0.800	0.950	**1.000**	0.975	**1.000**
Deg → *Deg*	0.800	0.950	**1.000**	0.975	**1.000**
Clu → *Deg*	**1.000**	**1.000**	**1.000**	0.975	**1.000**
PR →*Deg*	0.800	0.950	**1.000**	**1.000**	0.990
Avglen → *Deg*	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**
Cons → *Clu*	**0.800**	0.750	0.550	0.650	**0.800**
Deg → *Clu*	0.800	**0.900**	0.525	0.775	0.780
Clu → *Clu*	**1.000**	0.850	0.675	0.700	0.850
PR → *Clu*	0.800	0.800	0.500	0.762	**0.810**
Avglen → *Clu*	**0.800**	0.700	0.625	0.688	0.720
Cons → *PR*	**1.000**	0.850	0.850	0.912	0.910
Deg → *PR*	0.800	0.900	0.850	0.912	**0.950**
Clu → *PR*	0.600	0.850	0.825	0.825	**0.930**
PR → *PR*	**1.000**	0.850	0.875	0.850	0.930
Avglen → *PR*	0.600	0.850	0.825	0.863	**0.890**