Feature Augmentation on Small Graphs

Jiaqing Xie
University of Edinburgh
Edinburgh, United Kingdom

Rex Ying
Stanford University
USA

ABSTRACT

KEYWORDS

datasets, neural networks, gaze detection, text tagging

ACM Reference Format:
Jiaqing Xie and Rex Ying. 2021. Feature Augmentation on Small Graphs. In *Proceedings of (KDD'21).* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

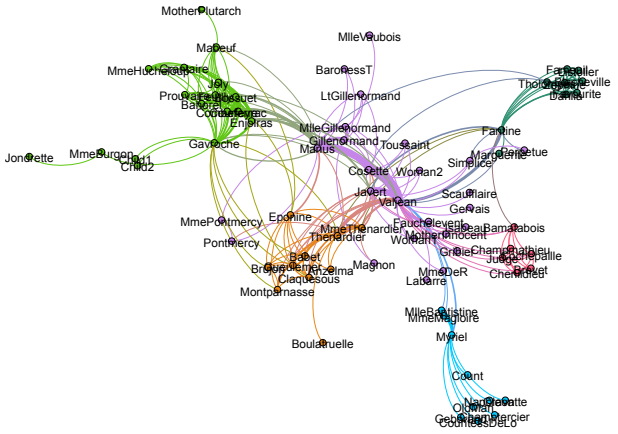

Figure 1: Question: In LesMiserables dataset, if already know the character Valjean has 35 neighbours(degrees), can we predict the feature PageRank of Valjean through graph neural network?)

Graph neural networks(GNN) are widely used in node classification, graph classification, link prediction problems and graph embedding extractions. Graph neural networks have broad application scenarios including knowledge graph, social network, computer network and also recommender systems. Better learning of graph structures and comprehensive learning of features' relationship on those tasks and application scenarios based on GNN might enable researchers to make predictions precisely. In a regular graph classification task, adjacency matrix and feature matrix are considered as the default inputs of a graph neural network. For example, benchmark Cora dataset has an input feature matrix $F \in \mathbb{R}^{2708\times1433}$ where it represents 2708 papers with 1433 filtered words for each paper entry, and it also has an adjacency matrix $A \in \mathbb{R}^{2708\times2708}$, where each element represents the link information between each two papers (both 0-1 matrix). Graph properties are not added to our input but instead sometimes we analyze them alone by taking each property into consideration. By giving an directed or undirected graph, we can easily get the property of degree of each node or the nodes' clustering coefficients by counting edges but we do not predict them at all, which means the prediction objects are always the nodes' or graph's classes. We do not know if adding those properties can help improve node or graph classification accuracy or not, or if there exists a strong relationship between features or not.

In this paper, we build models that are based on various kinds of graph convolutional layer types, including Graph Attention Netowrk(GAT), GraphSAGE, Graph Isomorphic Network(GIN) and Graph Convolutional Network(GCN) with multi-layer perceptron to perform mutual feature predictions instead of node classifications on the benchmark dataset to see the relationship between features. The features that selected for our research are constant feature, node degrees, clustering coefficient, average path length and pagerank. We choose Planetoid, TUDataset, Reddit and PPI dataset for the research and also include the random generated graph for further validation. If one feature predict the other feature precisely, they are considered to be redundant to concatenated features' representation. Finally we insert our artificial generated features to see which combination is the best, which is the feature augmentation period. Different datasets may have different feature combinations to augment their feature representations.

We design totally different experiments instead of traditional graph classification or link prediction problems in order to know features' relationship: (i) feature to feature prediction (ii) concatenated features to feature prediction (iii) original features with concatenated features to feature prediction(augmentation). Our task is to know (i) if GNN can perform these tasks well, including our own models, to test if GNN is powerful at predicting everything related (ii) of which features are closely related according our predictions, high value means redundant (iii) do augmentation really works on mutual feature prediction (iv) to see if node tasks and graph tasks are both effective to augmentation.

2 RELATED WORKS

2.1 Graph Representation Learning

2.2 Feature Importance

3 METHODS

3.1 Baseline Graph embedding methods

GCN. GCN is the abbreviation of **Graph Convolutional Network**[2]. It derives from signal processing domain which takes the

KDD'21, 14-18 August, Singapore

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

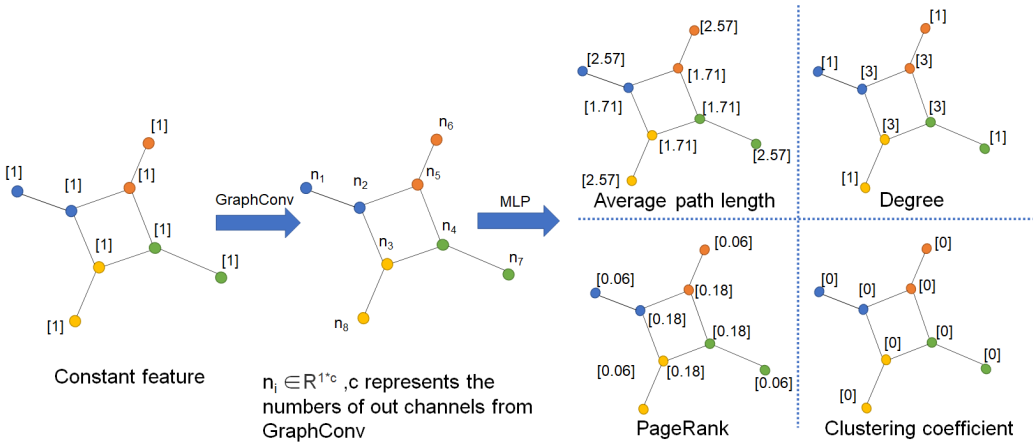

Figure 2: Baeline model for feature mutual prediction task

advantage of Laplacian matrix representation and filters by applying fourier transforms to graph signals. Here's the propagation rule of a multi-layer GCN:

$$X^{(l+1)} = \sigma\left(\widetilde{D}^{-\frac{1}{2}}\widetilde{A}\widetilde{D}^{-\frac{1}{2}}X^{(l)}W^{(l)}\right) \tag{1}$$

$X^{(l)} \in \mathbb{R}^{N_l * D_l}$ is the input of layer $(l+1)$, where $X^{(0)}$ is the initial feature matrix of the graph. $\widetilde{D}_{ii} = \sum_j \widetilde{A}_{ij}$ is a learnable degree matrix and $W^{(l)}$ is a learnable weight matrix. $\widetilde{A} = I_n + A$ is an adjacency matrix with self-connection and I_n is the identity matrix. It performs normalization to alleviate gradient vanishing problems. GCN has the time complexity of $O(V)$ where V is equal to number of nodes if adjacency matrix $\widetilde{A}$ is sparse. GCN embedding is more efficient in small graphs than large graphs. It's a transductive method which takes the whole graph into account. However, it's not influential when encoding small graphs that we want to explore.

GraphSAGE. GraphSAGE is an inductive graph representation learning method[1], which performs better than transductive methods such as GCN on large graphs. According to small graphs, they might show no difference on running time. The iteration of graph representation is computed by the aggregation of neighbourhood message of a given node, not considering all the edges in the graph. It concatenates itself to the message passing from its neighbours at k-th iteration with an MLP:

$$h_v^{(k)} = \sigma\left(\mathbf{W}^{(k)} \cdot \text{CONCAT}\left(h_v^{(k-1)}, h_{\mathcal{N}(v)}^{(k)}\right)\right), v \in \mathcal{V} \tag{2}$$

Here $h_v^{(k)}$ is the representation of node v at k-th iteration. $h_{\mathcal{N}(v)}^{(k)}$ is the aggregation of node v's neighbours at k-th iterarion. $\mathbf{W}^{(k)}$ is the trainbale weight matrix at k-th iteration. $\sigma(\cdot)$ is the activation function. The nearby nodes show similar output structures while disparate nodes have totally different representations[1]. Aggregation methods include mean, maxpooling, sum and LSTM aggregator[1].

GIN. GIN is the abbreviation of **Graph Isomorphism Network**[4]. Experiments show that GIN is as powerful as WL test, which can differentiate two different graph structures and identify two isomorphic graphs. Consider the propagation rule of GIN:

$$h_v^{(k)} = \text{MLP}^{(k)}\left(\left(1+\varepsilon^{(k)}\right)\cdot h_v^{(k-1)} + \sum_{u\in\mathcal{N}(v)} h_u^{(k-1)}\right) \tag{3}$$

Here h_v^k is the representation of node v at k-th iteration. It can be also written by $f \circ \varphi$. GIN uses MLP to learn f and φ. Injective function avoids two multisets generate the same embeddings from GNN. Meanwhile, similar graph embeddings have similar embeddings in GIN. Output module is essential in GIN, which includes a concatenation process and a readout module. Readout module is sum in GIN, while max and mean dissatisfy injective property. GraphSAGE embedding with mean aggregator performs as well as GIN. They have great performance on graph but not generally on node classification.

GAT. GAT is the abbreviation of **Graph Attention Network**[3]. It takes the advantage of attention mechanism, which is first proposed in a transformer architecture in NLP domain. The attention weight between node i and one of its neigbours node j can be expressed as α_{ij}:

$$\alpha_{ij} = \frac{\exp\left(\text{LeakyReLU}\left(\vec{\mathbf{a}}^T[\mathbf{W}\vec{h_i}||\mathbf{W}\vec{h_j}]\right)\right)}{\sum_{k\in\mathcal{N}(i)}\exp\left(\text{LeakyReLU}\left(\vec{\mathbf{a}}^T[\mathbf{W}\vec{h_i}||\mathbf{W}\vec{h_k}]\right)\right)} \tag{4}$$

Here it performs softmax operation. $\vec{h_i'}$ is the node embedding of node i.$\vec{h_j'} \in \mathbb{R}^{F'}$ is representation of one of node i's neighbours. $\vec{\mathbf{a}}^T \in \mathbb{R}^{2F'}$ is parameterized weight vector. LeakyReLU is a nonlinear activation function. Output feature is $\vec{h_i'}$:

$$\vec{h_i'} = \sigma\left(\sum_{j\in\mathcal{N}_i}\alpha_{ij}\mathbf{W}\vec{h_j}\right) \tag{5}$$

It'a a linear combination with non-linearity $\sigma(\cdot)$. The combination method of multi-head attention in GAT is take averaging result. The advantage of GAT is that we can perform inductive learning without knowing everything about the whole graph.

3.2 Feature to Feature Prediction

Algorithm 1: Get Matrix of Feature Mutual Relationship

Input: Graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$; node property length $\mathcal{K}$; Input feature matrix $\mathbf{x}_{\mathcal{G}} \in \mathbb{R}^{|\mathcal{V}|\times\mathcal{K}}$, second dimension order is constant feature, degree, clustering coefficient, pagerank and average path length or more features; model architecture $\mathcal{M}$, including GNN embedding(GCN, GraphSAGE, GIN, GAT) with MLP model; metrics $\mathcal{P}$

Output: Feature mutual relationship matrix $\mathcal{R} \in \mathbb{R}^{\mathcal{K}\times\mathcal{K}}$

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

Given a graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$. We want to achieve the desired feature matrix $\mathbf{x}_{\mathcal{G}} \in \mathbb{R}^{|\mathcal{V}|\times\mathcal{K}}$ from $\mathcal{G}$ which requires an adjacency matrix $\mathcal{A}_g \in \mathbb{R}^{|\mathcal{V}|*|\mathcal{V}|}$ of $\mathcal{G}$. Suppose that we choose five of all graph features to show exploratory results, which are constant feature (Cons), degree (Deg), clustering coefficient (Clu), average path length (AvgLen) and PageRank (PR).

Constant feature of one node u $\underset{u\in\mathcal{V}}{Cons}(u)$ is given by c, where c is equal to a constant value. We set to 1 in this paper for normalization. Degree of node u is equal to $\underset{u\in\mathcal{V}}{Deg}(u)$ is equal to the number of node u's neighbours. Clustering coefficient of node u $\underset{u\in\mathcal{V}}{Clu}(u) = \frac{2e_{jk}}{k_i*(k_i-1)}$, where $j, k \in \mathcal{V}$,e_{jk} represents the total possible edges between node u's neighbours and k_i is the number of node u's neighbours. $\underset{u\in\mathcal{V}}{PR}(u)$ is given by:

$$\underset{u\in\mathcal{V}}{PR}(u) = \frac{1-q}{|\mathcal{V}|} + q \times \sum_{v\in\mathcal{N}(u)} \frac{PR(v)}{\mathcal{L}(v)} \tag{6}$$

where $\mathcal{N}(\cdot)$ means the node u's neighbours and $\mathcal{L}(\cdot)$ means the outbound link numbers of the node u to its neighbours. In the traditional analysis of pagerank algorithm, the object is directed graph. However we can treat undirected graph as bidirectional graph, so outbound links is equal to the edge numbers from node u to its neigbours. q is the residual probability, which is defualt as 0.85. We treat it as a hyper-parameter. $\underset{u\in\mathcal{V}}{Avglen}(u)$ is calculated by:

$$\underset{u\in\mathcal{V}}{Avglen}(u) = \frac{1}{\mathcal{V}'} \sum_{v\in\mathcal{V}\neq u} I(u, v) * d_{min}(u, v) \tag{7}$$

where $I(u, v)$ indicates whether there's a path from node u to node v. If node v is reachable according to node u, then $I(u, v)$ is equal to 1 else 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.

The algorithm 1 shows a complete process of how to construct a matrix for mutual feature prediction. Suppose that we have successfully compute a feature matrix $\mathbf{x}_{\mathcal{G}} \in \mathbb{R}^{|\mathcal{V}|\times\mathcal{K}}$. Specifically $\mathcal{K}$ is equal to 5 here since five of graph features are being researched. The features have been indexed. Suppose we have constructed a model $\mathcal{M}$ which uses the baseline graph embedding methods together with multi-layer perceptrons. Given a feature, we can find its column index from $\mathbf{x}_{\mathcal{G}}$ and extract them by only taking that column, such as constant feature: $\mathbf{x}_{\mathcal{G}}(:, 1)$ and average path length: $\mathbf{x}_{\mathcal{G}}(:, 4)$. We need to extract two columns, one for input $\mathbf{I}_{\mathcal{G}}$ and the other for output $\mathbf{O}_{\mathcal{G}}$. We need to set bins for the output since the main task is classification not regression task. After outputs are classified into each own class, we use that constructed model $\mathcal{M}$ to predict output $\mathbf{O}'_{\mathcal{G}}$. We use a metrics to evaluate the result between $\mathbf{O}'_{\mathcal{G}}$ and $\mathbf{O}_{\mathcal{G}}$, which is the classification accuracy here. Finally we will achieve a feature mutual relationshiop matrix $\mathcal{R}$. The binning results on the datasets that we have chosen are dicussed in part 4.3. We should be careful that constant feature can be only taken as input but not as output. It is the most basic feature for all graphs which could not lead to a self classification problem.

3.3 Feature Concatenation

Algorithm 2: Get Matrix of Feature Mutual Relationship

Input: Graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$; node property length $\mathcal{K}$; Input property matrix $\mathbf{x}_{\mathcal{G}} \in \mathbb{R}^{|\mathcal{V}|\times\mathcal{K}}$, second dimension order is constant feature, degree, clustering coefficient, pagerank and average path length; model architecture $\mathcal{M}$, including GNN embedding(GCN, GraphSAGE, GIN, GAT) with MLP model; metrics $\mathcal{P}$

Output: Feature mutual relationship matrix $\mathcal{R} \in \mathbb{R}^{\mathcal{K}\times\mathcal{K}}$

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

We have a feature list $\mathcal{F} = \{\mathcal{F}_1, \mathcal{F}_2, ..., \mathcal{F}_n\}$, which contains n features of the graph. The value $\mathcal{R}(i, j)$ in relationship matrix indicates if $\mathcal{R}_i$ and $\mathcal{R}_j$ are predictable to each other mutually. Moreover, we want to add additional features to $\mathcal{R}_i$ to see if this will predict $\mathcal{R}_j$ more accurately. However we cannot add features arbitrarily since

adding similar features might bring about redundancy in data. In this paper we provide three main ideas for concatenating features.

simple concatenation. If we

Neural Tensor Network. Apart from concatenating two or more features directly, we use try to use neural tensor network, which is presented by:

$$g_{new} = u^T a(f_1^T W_R f_2 + V[f_1, f_2] + b) \tag{8}$$

SkipLast. We
We now go through the algorithm 2.

3.4 model architectures

4 EXPERIMEMTS

We compute the feature matrices for each graph. We implement four graph embedding methods: GIN, GCN, GraphSAGE and GAT on graph feature inputs to predict features' relationship matrices, which is regarded as the traditional GNN result on new tasks. We also compare them with the added GNN blocks that we have designed, as the comparison of GNN models in order to interpret Graph Neural Network's robustness. We've also handled with some important issues in the following parts. Typical datasets that meet the requirement of small graphs are listed in 4.1. Coding environment settings and parameter settings are detailed in 4.2.

4.1 Datasets

In our preliminary experiments, we choose ten small graph-based datasets. A generation of supervised graph is also included. However, according to the data that mentioned in original GraphSage paper, which are the aimed large graph, we do not care about them at this stage.

Planetoid. Planetoid datasets are citation datasets including Cora, CiteSeer and PubMed. One node in the graph represents each paper. Edge index means there is a citation link between two papaers and nodes in the graph are undirectly linked. The feature matrix of each dataset is given by sparse bags-of-words vectors. It's one-hot encoding which is at a lower level embedding space. In Cora dataset, there are 2708 nodes with 1433 features and 5429 edges. In CiteSeer dataset, there are 3327 nodes with 3703 features and 4732 edges. In PubMed dataset, there are 19717 nodes with 500 features and 44338 edges. In this work, we choose all of three datasets to show the generalization effects, which is that firstly we test on Cora dataset, then we test on CiteSeer and PubMed.

TUDataset. TUDataset is a collection of benchmark datasets for learning graph representations, which gathers numerous domains and is authored by different experts. More specifically, it includes data from small molecules, bioinformatics, social networks, computer vision and other synthetic kernels. In this work, we choose two datasets Proteins and Enzymes from Bioinformatics domain. In Enzymes dataset, there are 600 graphs with 6 classes. Each graph has an average edges of 62.1 and an average edges of 32.6. In Proteins dataset, there are 1113 graphs with 2 classes. Each graph has an average egdes of 39.1 and an average edges of 72.8. We will make more experiments on other datasets, such as Zinc and Qm9 from molecular datasets or Reddit-Binary and Reddit_Threads from social networks.

PPI. Protein to Protein Interaction(PPI) network is generally used in biochemistry, containing positional gene sets, motif gene sets and immunological signatures as features (50 in total) and gene ontology sets as labels (121 in total). It is mentioned in this paper.

4.2 Experiment Set-up

We've mentioned four graph embedding methods, which are GCN, GraphSAGE, GIN and GAT. Our baseline model is constructed by using graph embeddings and followed by MLPs. We take

In single feature to feature prediction tasks, we set up a 2-layer GNN with each layer followed by *batch norm* layer, activated by*relu function* and using *dropout* method. Especially

GPU settings: GeForce RTX 2060 Super

Add parameter setting and coding environment here:

We mainly use Pytorch framework together with torch_geometric API for building our model architecture. In the single feature to single feature task, our default model consists of two GNN blocks and a MLP block with two linear layers. We both have classification tasks for feature prediction and also regression tasks. The metrics for classification task is the accuracy score and macro-F1 score. Input dimension is set to 1 which is input_channel parameter.

4.3 Feature properties of graphs

We choose five of all graph features to show exploratory results, which are constant feature(Cons), degree(Deg), clustering coefficient(Clu), average path length(AvgLen) and PageRank(PR).

Given a graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, constant feature of one node u $\underset{u\in\mathcal{V}}{Cons}(u)$ is given by c, where c is equal to a constant value. We set to 1 in this project for normalization. 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 $\underset{u\in\mathcal{V}}{Clu}(u) = \frac{2e_{jk}}{k_i*(k_i-1)}$, where $j, k \in \mathcal{V}$,e_{jk} represents the total possible edges between node u's neighbours and k_i is the number of node u's neighbours.

Having the prerequisite of these basic knowledge, we compute the feature matrix for each dataset. The feature matrix serves to $\mathcal{R}^{|V|*5}$. We should bear in mind that this feature matrix is both used for input and output, as described in Algorithm 1. One important step before fed into training session is to measure the distribution of each feature. Just as imbalanced data is not favoured in the graph or node classification tasks, we observe all the feature distributions and illustrate potential issues among all datasets.

binning methods. Suppose the clustering coefficient is the feature that we want to predict. Clustering coefficient is a non-integer value as we've discussed before. We firstly take it as the classification problem. Binning methods is then used to identify the Therefore we set bins in order to change the outputs to integers. Figure 2 shows the graph properties on planetoid datasets with violin plots. The property order is: degree, clustering coefficient, pagerank and average path length. Density distribution is very different between each dataset and each property as well. Generally, we set 4-8 bins. Too large bins may lead to a more sparse confusion matrix while

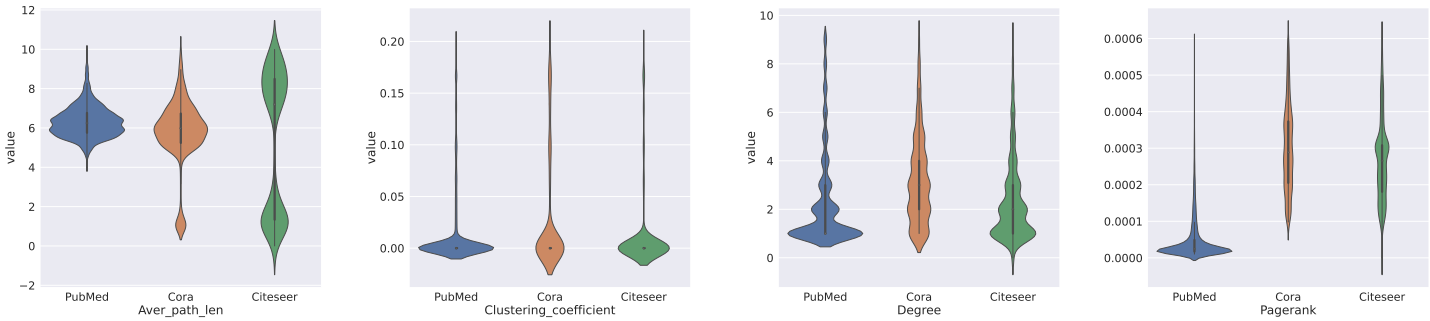

Figure 3: Violin plots of graph properties on planetoid datasets

too small bins may lead to over concentration on one class. So the number of bins should be not too big nor too small and the number of 4 to 8 satisfies this condition. Specifically, we take the Cora dataset as an example to illustrate the point. Figure 3 shows the example of the property's density distribution on Cora dataset with distplots. We can set 4 bins for Degree, Clustering_coefficient, Pagerank and Aver_path_len according to the values that shown on the x axis of the distplots.

Through the violin plot we find that most of the clustering coefficients are 0 in planetoid datasets. Therefore we treat 0 as a single class. When setting bins for all the data, we remove all the zeros since it's not reasonable to set bins where many of the bin values are 0. Specifically in figure 3, we see that in Cora dataset, the density function concentrates on the 0.00 since most of nodes(1126 out of 2708) in Cora dataset has a zero clustering coefficient. Similarly 2104 out of 3327 nodes has a zero clustering coefficient in PubMed dataset while 14899 out of 19717 nodes have a zero clustering coefficient in Citeseer dataset.

4.4 Feature to feature prediction results

4.4.1 GNN performance. We test the baseline GNN models to conclude their performance on *Planetoid* datasets and *TUDatasets*. We need to ensure that model parameters are the same when testing each baseline model. The depth of the graph embedding method is set to 2 and the bin of number of classes is set to 6. For the first graph embedding layer, input dimension is 1 and output dimension is 256, while for the second graph embedding layer input dimension is 256 and output dimension is 64. A relu unit and dropout layer is followed by each graph embedding layer. After the graph embedding block, multi-layer perceptrons are connected. The input dimension is 64 for the MLP block while the output dimension is equal to value of bins, which is 6.

According to the results on node datasets, **GIN** performs the best among all baseline models. Of all 48 test tasks on three different dataset, it takes up the 41 best test results. The results is based on the average of 10 times of testing. We figure out that the feature *degree* can be predicted precisely on *Planetoid* dataset. Predicting accuracy can reach 1.000 when testing Cora dataset. For Citeseer and PubMed dataset, accuracy on predicting *degree* can reach 0.999 and 0.998. Feature *Pagerank* can be predicted to a large extent for Cora but more difficult for Citeseer and PubMed. Feature *Clustering coefficient* can be predicted easily on Citeseer and PubMed dataset while it's more difficult for **GIN** to predict *Clustering coefficient*. Overall it's not easy for all baseline models to predict average path length when model depth is shallow.

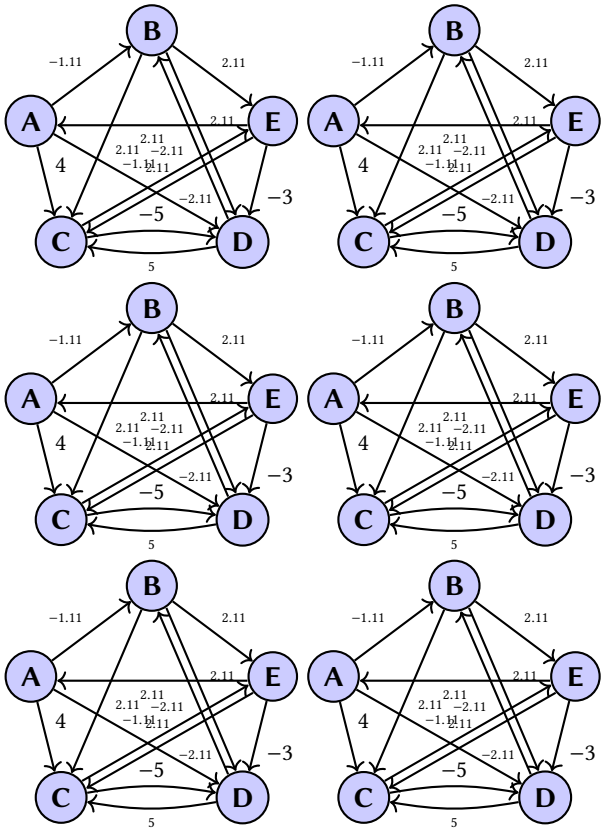

Figure 4: jiaqing

Table 1: Feature to Feature Prediction Results on Citation Datasets (bins = 6)

Aim	Cora				CiteSeer				PubMed			
	GCN	GIN	SAGE	GAT	GCN	GIN	SAGE	GAT	GCN	GIN	SAGE	GAT
1 -> 2	0.509	**1.000**	0.213	0.202	0.578	**0.999**	0.379	0.379	0.653	**0.998**	0.478	0.478
1 -> 3	0.523	**0.533**	0.461	0.461	0.673	**0.707**	0.658	0.658	**0.789**	0.797	0.780	0.780
1 -> 4	0.639	**0.756**	0.160	0.160	0.706	**0.662**	0.201	0.185	**0.664**	0.526	0.161	0.143
1 -> 5	0.357	**0.384**	0.169	0.169	0.406	**0.495**	0.182	0.170	0.300	**0.410**	0.166	0.171
2 -> 3	0.550	0.542	**0.548**	0.506	0.704	0.706	**0.718**	0.674	0.800	**0.802**	0.799	0.781
2 -> 4	0.573	**0.792**	0.750	0.392	0.619	**0.754**	0.756	0.350	0.532	**0.617**	0.609	0.344
2 -> 5	0.420	0.435	**0.440**	0.340	0.532	**0.546**	0.544	0.497	0.422	**0.450**	0.426	0.338
3 -> 2	0.423	**1.000**	0.504	0.285	0.582	**0.999**	0.609	0.489	0.638	**0.998**	0.568	0.475
3 -> 4	0.427	**0.695**	0.403	0.199	0.530	**0.554**	0.427	0.290	0.441	**0.565**	0.394	0.254
3 -> 5	0.286	**0.310**	0.263	0.219	**0.409**	0.383	0.387	0.356	0.255	**0.314**	0.285	0.187
4 -> 2	0.308	**1.000**	0.311	0.223	0.297	**1.000**	0.414	0.383	0.585	**1.000**	0.482	0.478
4 -> 3	0.490	**0.538**	0.486	0.461	0.672	**0.698**	0.665	0.658	0.796	**0.805**	0.780	0.780
4 -> 5	0.215	**0.421**	0.266	0.185	0.249	**0.401**	0.238	0.282	0.313	**0.449**	0.192	0.174
5 -> 2	0.409	**1.000**	0.499	0.228	0.542	**0.999**	0.672	0.419	0.648	**0.996**	0.704	0.478
5 -> 3	0.508	**0.538**	0.498	0.460	0.683	**0.701**	0.679	0.658	**0.798**	0.797	0.788	0.780
5 -> 4	0.450	**0.741**	0.490	0.202	0.530	**0.645**	0.603	0.300	0.551	**0.542**	0.573	0.152

Table 2: Feature to Feature Prediction Results on TUDatasets (bins = 6)

Aim	Proteins				Enzymes				NCI1			
	GCN	GIN	SAGE	GAT	GCN	GIN	SAGE	GAT	GCN	GIN	SAGE	GAT
1 -> 2	0.560	**0.662**	0.469	0.469	0.630	**0.881**	0.418	0.418	0.833	**0.916**	0.406	0.401
1 -> 3	0.299	**0.436**	0.202	0.202	0.315	**0.344**	0.262	0.261	/	/	/	/
1 -> 4	0.645	**0.648**	0.170	0.170	0.597	**0.639**	0.181	0.169	0.630	**0.636**	0.172	0.169
1 -> 5	**0.182**	0.171	0.171	0.171	0.169	**0.200**	0.173	0.169	0.217	**0.223**	0.172	0.170
2 -> 3	0.435	0.387	**0.454**	0.312	0.397	0.359	**0.407**	0.314	/	/	/	/
2 -> 4	0.622	0.699	**0.722**	0.239	0.623	0.679	**0.702**	0.249	0.689	0.705	**0.740**	0.304
2 -> 5	**0.212**	0.184	0.200	0.171	0.233	0.216	**0.245**	0.173	**0.237**	0.222	0.235	0.177
3 -> 2	0.537	**0.652**	0.482	0.469	0.569	**0.788**	0.544	0.418	/	/	/	/
3 -> 4	**0.575**	0.465	0.467	0.549	**0.539**	0.395	0.369	0.198	/	/	/	/
3 -> 5	0.227	0.196	**0.254**	0.228	0.197	0.169	**0.203**	0.169	/	/	/	/
4 -> 2	0.430	**0.662**	0.446	0.469	0.240	0.336	**0.458**	0.418	0.100	**0.827**	0.417	0.341
4 -> 3	0.247	**0.330**	0.278	0.219	0.269	**0.305**	0.270	0.262	/	/	/	/
4 -> 5	0.171	**0.172**	0.171	0.171	0.169	**0.180**	0.171	0.168	0.169	**0.339**	0.173	0.170
5 -> 2	0.522	**0.657**	0.482	0.469	0.572	**0.881**	0.484	0.418	0.784	**0.919**	0.836	0.401
5 -> 3	0.276	0.258	**0.328**	0.202	0.283	0.308	**0.354**	0.262	/	/	/	/
5 -> 4	**0.579**	0.453	0.395	0.170	**0.542**	0.430	0.360	0.170	**0.629**	0.534	0.588	0.180

4.4.2 hyper-parameter tunning. We also compare the performance between node datasets (for node classifications) and graph datasets (for graph classifications).

We also compare the performance between node datasets (for node classifications) and graph datasets (for graph classifications).

We also compare the performance between node datasets (for node classifications) and graph datasets (for graph classifications).

We also compare the performance between node datasets (for node classifications) and graph datasets (for graph classifications).

We also compare the performance between node datasets (for node classifications) and graph datasets (for graph classifications).

We also compare the performance between node datasets (for node classifications) and graph datasets (for graph classifications).

4.4.3 graph embedding vs linear embedding.

4.5 Feature Augmentation Results

4.6 Additional Features

4.7 self-generated graphs

.

Table 3: Hyper-parameter : number of bins on node datasets

Bins	Citeseer			PubMed		
	3->2	4->5	5->3	3->2	4->5	5->3
2	1.000	0.752	0.838	1.000	0.868	0.894
3	0.999	0.684	0.781	1.000	0.695	0.853
4	1.000	0.570	0.752	0.999	0.618	0.833
5	0.999	0.491	0.723	1.000	0.488	0.817
6	1.000	0.491	0.719	1.000	0.423	0.805
7	1.000	0.360	0.712	1.000	0.345	0.799
8	1.000	0.386	0.703	0.999	0.335	0.794
9	1.000	0.373	0.696	0.998	0.272	0.794
10	1.000	0.369	0.656	0.988	0.262	0.792

Table 4: Hyper-parameter : number of bins on graph datasets

Bins	Enzymes			Proteins		
	3->2	4->5	5->3	3->2	4->5	5->3
2	0.881	0.540	0.550	0.891	0.515	0.525
3	0.869	0.356	0.377	1.000	0.340	0.418
4	0.861	0.260	0.307	0.792	0.253	0.340
5	0.590	0.202	0.285	0.775	0.205	0.284
6	0.696	0.174	0.274	0.662	0.171	0.232
7	0.505	0.144	0.251	0.601	0.148	0.289
8	0.525	0.127	0.241	0.389	0.128	0.266
9	0.408	0.115	0.238	0.437	0.121	0.269
10	0.510	0.107	0.238	0.335	0.108	0.257

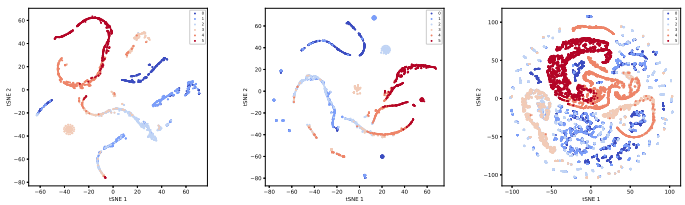

Figure 5: tSNE on graph embeddings with Degree predicting PageRank

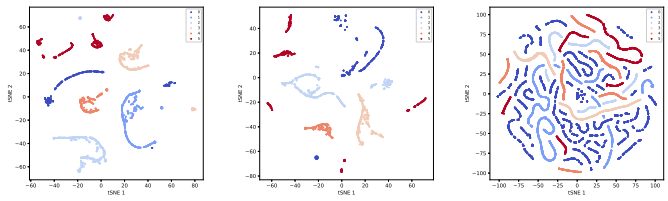

Figure 6: tSNE on graph embeddings with Clustering predicting Degree

4.8 Regression tasks

4.9 Comparison between datasets

5 CONCLUSIONS AND DISCUSSIONS

Future works.

REFERENCES

[1] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *Advances in neural information processing systems*. 1024–1034.

[2] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).

[3] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903* (2017).

[4] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2018. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826* (2018).