Fea2Fea : Feature to Feature Prediction and 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. Fea2Fea : Feature to Feature Prediction and Augmentation on Small Graphs. In *Proceedings of (KDD'21).* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

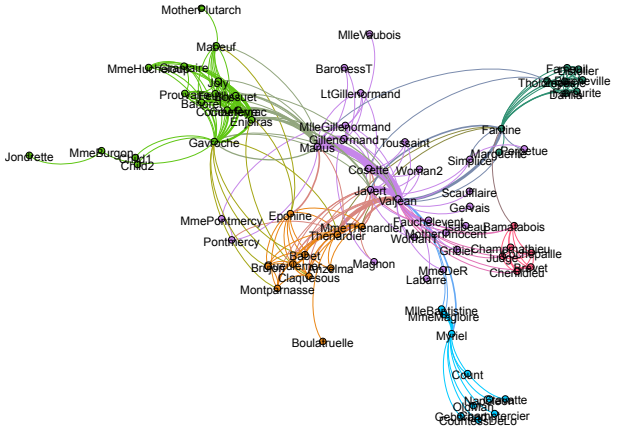

Figure 1: Question: In LesMiserables dataset, if already know the character Valjean has 35 neighbours(degrees), can we predict the feature PageRank of Valjean through graph neural network?)

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.

KDD'21, 14-18 August, Singapore

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

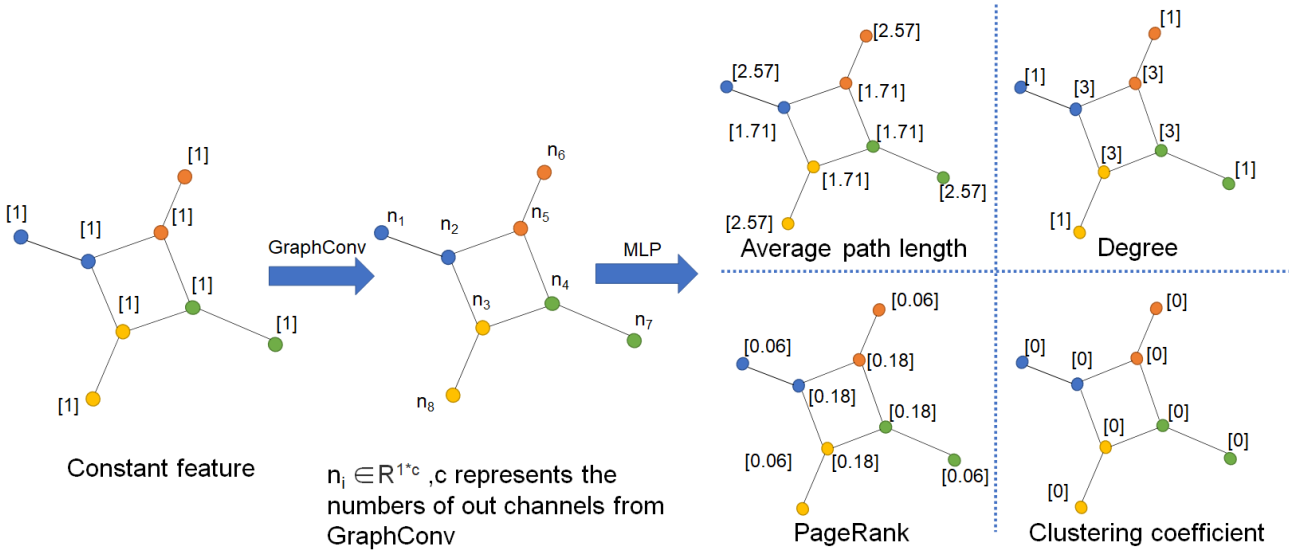

Figure 2: Baeline model for feature mutual prediction task

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 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 non-linear 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 Approach

Algorithm 1: 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 $\mathcal{R} \in \mathbb{R}^{\mathcal{K}\times\mathcal{K}}$

```
R ← 0
for i ← 1 to K do
    R(i, i) ← 1
    I_G ← x_G(:, i)
    if i == K then
        return R;
    end
    for j ← i + 1 to K do
        O_G ← Bin(x_G(:, j))
        O'_G ← M(I_G, O_G)
        R(i, j), R(j, i) ← P(O'_G, O_G)
    end
end
```

The task for the graph feature to feature prediction is implemented by baseline models, aiming to test a GNN model's robustness on small graphs. Here the input is one signal feature $\mathbf{x}_\mathcal{G}$

3.3 Feature Concatenation Methods

SkipLast.

Concatenation by importance.

Inner Product Concatenation.

3.4

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

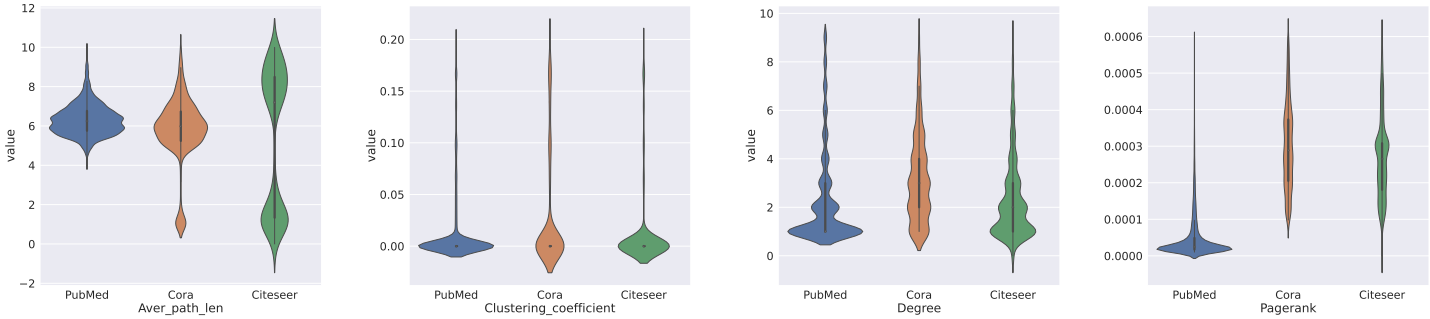

Figure 3: Violin plots of graph properties on planetoid datasets

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

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 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. The experiments for testing the GNN robustness are on single feature to single feature predictions. Datasets are Planetoid and TUDataset.

The results of the experiments for planetoid datasets show that *GIN* has the best performance on single feature to feature prediction. Especially when predicting *Degree* feature, average accuracy for *GIN* is equal to 1.000. However, it's difficult for a *GNN* to predict average path length.

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

Binning results. As we mentioned before, the number of bins depends on how we set the number of classes for certain feature. The performance test of graph neural network has selected the best GNN model for binning test, which is **GIN**. Here, we perform two experiments. First, we change the GNN stack to find the best parameters of the GIN model. Secondly, we Draw a violin chart

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	0.548	**1.000**	0.379	0.379	0.637	**0.997**	0.478	0.478
1 -> 3	0.523	**0.533**	0.461	0.461	0.675	**0.698**	0.658	0.658	**0.796**	0.790	0.780	0.780
1 -> 4	0.639	**0.756**	0.160	0.160	0.574	**0.671**	0.190	0.190	**0.648**	0.513	0.161	0.141
1 -> 5	0.357	**0.384**	0.169	0.169	0.250	**0.514**	0.178	0.166	0.305	**0.423**	0.166	0.175
2 -> 3	0.550	0.542	**0.548**	0.506	0.706	0.700	**0.725**	0.692	0.795	**0.799**	0.799	0.780
2 -> 4	0.573	**0.792**	0.750	0.392	0.555	**0.780**	0.748	0.307	0.459	**0.608**	0.609	0.242
2 -> 5	0.420	0.435	**0.440**	0.340	0.542	**0.543**	0.539	0.391	0.414	**0.457**	0.426	0.204
3 -> 2	0.423	**1.000**	0.504	0.285	0.617	**1.000**	0.632	0.497	0.612	**0.995**	0.568	0.449
3 -> 4	0.427	**0.695**	0.403	0.199	0.536	**0.561**	0.437	0.299	0.408	**0.579**	0.394	0.206
3 -> 5	0.286	**0.310**	0.263	0.219	**0.433**	0.398	0.383	0.355	0.254	**0.287**	0.285	0.209
4 -> 2	0.308	**1.000**	0.311	0.223	0.500	**1.000**	0.379	0.379	0.593	**1.000**	0.482	0.478
4 -> 3	0.490	**0.538**	0.486	0.461	0.672	**0.689**	0.658	0.658	0.794	**0.800**	0.780	0.780
4 -> 5	0.215	**0.421**	0.266	0.185	0.182	**0.489**	0.178	0.166	0.273	**0.455**	0.192	0.175
5 -> 2	0.409	**1.000**	0.499	0.228	0.505	**0.998**	0.644	0.403	0.591	**0.995**	0.704	0.478
5 -> 3	0.508	**0.538**	0.498	0.460	0.676	**0.709**	0.658	0.657	**0.796**	0.793	0.788	0.780
5 -> 4	0.450	**0.741**	0.490	0.202	0.424	**0.692**	0.596	0.316	0.430	**0.586**	0.573	0.161

or distribution map to find out the feature distribution of each data set. Figure 3 shows the feature distribution of the Planetoid dataset. However, each distribution is not suitable for Gaussian distribution, especially in the case of clustering coefficients. More than half of the features are 0, resulting in sparse vectors. But we want to accurately predict the clustering coefficient, so we set bins containing all zeros as partitions, and set the retention rates of other bins as a percentage division rule based on non-zero values.

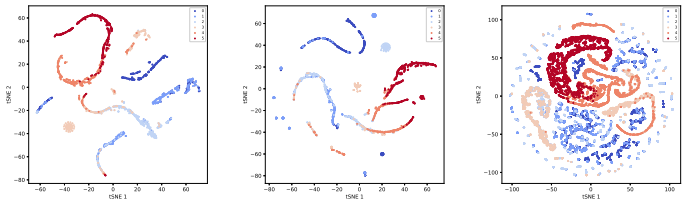

Figure 4: tSNE on graph embeddings with Degree predicting PageRank

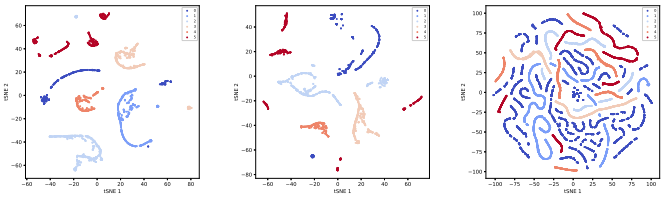

Figure 5: tSNE on graph embeddings with Clustering predicting Degree

4.5 Feature Augmentation Results

4.6 Additional Features

4.7 self-generated graphs

.

4.8 Regression tasks

single
(finish) concatenated
(on-going) original with concatenated
(on-going) GNN and our own model
(on-going)

Multi-graph input(Graph Classification), we need to compute each graph's property. Batch size is 1 in this case. (Graph classification problem instead of node classification problem)

We have the

violin plot here and also the rectified-bin algo.

intro + selected property

feature distribution pics
comparison by graph
set by binning method -> 1)classification or regression 2) how many classes by binning 3) using normalization or not

innovation1: different from

4.9 Comparison between datasets

visualization, graph and node, and also which is easy to interpret through feature and which not

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.*

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

Aim	NC1				Proteins				Enzymes			
	GCN	GIN	SAGE	GAT	GCN	GIN	SAGE	GAT	GCN	GIN	SAGE	GAT
1 -> 2												
1 -> 3												
1 -> 4												
1 -> 5												
2 -> 3												
2 -> 4												
2 -> 5												
3 -> 2												
4 -> 2												
4 -> 3												
4 -> 5												
5 -> 2												
5 -> 3												
5 -> 4												

Table 3: Hyper-parameter 1 : number of bins

Bins	Citeseer			PubMed		
	3->2	4->5	5->3	3->2	4->5	5->3
2	1.0000	0.7620	0.8350			
3	1.0000	0.6800	0.7820			
4	1.0000	0.5640	0.7440			
5	0.9960	0.5310	0.7200			
6	1.0000	0.4680	0.7250			
7	1.0000	0.3870	0.7110			
8	1.0000	0.3620	0.7020			
9	1.0000	0.3800	0.6910			
10	1.0000	0.3910	0.6900			

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).