ORIGINAL RESEARCH

Feature selection: Key to enhance node classification with graph neural networks

Sunil Kumar Maurya[1,2] 🔘 | Xin Liu[2] | Tsuyoshi Murata[1,2]

[1]Tokyo Institute of Technology, Tokyo, Japan

[2]AIRC, AIST, Tokyo, Japan

Correspondence

Sunil Kumar Maurya, Tokyo Institute of Technology, Tokyo, Japan.
Email: skmaurya@net.c.titech.ac.jp

Abstract

Graphs help to define the relationships between entities in the data. These relationships, represented by edges, often provide additional context information which can be utilised to discover patterns in the data. Graph Neural Networks (GNNs) employ the inductive bias of the graph structure to learn and predict on various tasks. The primary operation of graph neural networks is the feature aggregation step performed over neighbours of the node based on the structure of the graph. In addition to its own features, for each hop, the node gets additional combined features from its neighbours. These aggregated features help define the similarity or dissimilarity of the nodes with respect to the labels and are useful for tasks like node classification. However, in real-world data, features of neighbours at different hops may not correlate with the node's features. Thus, any indiscriminate feature aggregation by GNN might cause the addition of noisy features leading to degradation in model's performance. In this work, we show that selective aggregation of node features from various hops leads to better performance than default aggregation on the node classification task. Furthermore, we propose a Dual-Net GNN architecture with a classifier model and a selector model. The classifier model trains over a subset of input node features to predict node labels while the selector model learns to provide optimal input subset to the classifier for the best performance. These two models are trained jointly to learn the best subset of features that give higher accuracy in node label predictions. With extensive experiments, we show that our proposed model outperforms both feature selection methods and state-of-the-art GNN models with remarkable improvements up to 27.8%.

KEYWORDS

classification, feature selection, neural network

1 | INTRODUCTION

Recent years have seen rapid growth of Graph Neural Networks (GNNs) being utilised on real-world applications in multitude of domains. These applications range in wide variety of tasks like recommendation systems [1–3], graph data mining [4, 5], molecular properties predictions [6–8], natural language processing [9, 10], vision related tasks [11, 12], physics simulations [13], node ranking [14, 15], etc.

GNNs are designed to operate on graph structure derived from the dataset and learn from node/edge features. The primary component of GNNs which separates them from other neural network architectures is the feature aggregation operation. The neighbours of the nodes are explored via the graph structure, and their features are combined with the node's features. The underlying assumption is that nodes are connected to other nodes with similar properties in the graph (homophily). Hence combining features from the neighbours helps to improve the signal, which helps in the learning process. Recently, some works [16–18] have shown that in some datasets, dissimilar nodes are connected in the graph (heterophily) and they may require different aggregation schemes for the design of the GNN.

The aggregation is performed in K steps, where K is the number of hops from the source node to the neighbours in the graph. Thus, the extent of neighbourhood feature aggregation

expands with K. If the features aggregated from neighbours at different hops are considered separately, the node gets a set of features based on its neighbourhood. Comparing these features can help to study homophily/heterophily characteristics of the neighbours from the node's perspective. There are many different ways of aggregation schemes proposed in current GNN literature. Many GNN models use simple indiscriminate aggregation [19–21], some use random sampling of neighbours [22, 23], attention-based aggregation [24, 25], filter coefficient based aggregation [26–28], etc.

Real-world data often have noisy components and do not follow the assumptions of GNN models strictly. For example, all neighbours of the node may not be similar to it in feature information especially considering nodes lying at various hop-distance away from the source node. Thus, any feature aggregation by GNN with strict homophily assumption may often cause the addition of noisy features and may reduce the signal present in node features to learn effectively. Researchers have explored many ways to tackle this challenge by proposing many improvements in model design, for example, making models deeper [27, 29, 30], using attention layers [24], residual layers [27, 31, 32], etc. However, most of these solutions attempt to fit the model to the noisy data in an effective way [33, 34].

Instead of designing a GNN model to fit the noisy data, we propose to treat the problem with a feature selection approach. Feature selection approaches aim to choose small subset of the relevant features from all available input features by removing irrelevant and noisy components [35]. The presence of noisy features can lead to model fitting on noisy components which can lead to sub-optimal learning and reduced generalisation. By selecting informative features relevant to the given task, the model can learn more effectively leading to a better performance on the prediction task. In our approach, we pre-compute a set of node features by feature aggregation over the graph. Node features aggregated from various hops are stored as feature matrices. With extensive experiments, we show that using all node features for learning on node classification task leads to sub-optimal performance. The reason can be attributed to presence of noisy/non-informative features present in the set. We demonstrate that learning on certain subset of these features can significantly improve the performance on a wide variety of datasets. Based on these observations, we propose a Dual-Net GNN model, which learns to select the subset of node features that are optimal for learning. The model consists of two neural networks: *classifier* and *selector*. The role of the classifier is to predict the labels of the nodes based on a given input. The role of the selector is to predict the subset of input features on which the classifier will have the best performance. These two networks are trained jointly, providing feedback to each other. The following are the main contributions of our work:

- We are the first to run extensive experiments on standard node classification datasets to show that feature selection can improve the GNN model's performance.
- We propose a novel Dual-net GNN architecture to find the optimal subset of features which leads to the better prediction accuracy of the model.

- In experimental results, our proposed model outperforms the state-of-the-art (SOTA) GNN models such as GPRGNN & GloGNN on the node classification task and achieves up to 27.8% higher accuracy.

Source code for main experiments are available at https://github.com/sunilkmaurya/DualNetGNN.

Note *This paper is an extension to our previous short paper accepted at CIKM' 2022 [36]. In this paper, we provide more comprehensive details on the model architecture and training methodology (include training algorithm), and extend our result section by including experiments on feature selection methods. We demonstrate that our model (and graph neural networks in general) perform better than conventional feature selection methods with significantly less computational time. In addition, we include detailed analysis of learnt hops selected by the model, validation loss characteristics during training, training time comparison and discuss model complexity. Finally, to demonstrate the scalability of our model, we include additional results on recently proposed large benchmark datasets [37] for node classification.*

2 | PRELIMINARIES

2.1 | Notations

Let $G = (V, E)$ be an undirected graph with n nodes and m edges. The graph is represented as adjacency matrix denoted by $A \in \{0, 1\}^{n \times n}$ with each element $A_{ij} = 1$ if there exists an edge between node v_i and v_j, otherwise $A_{ij} = 0$. When self-loops are added to the graph, then the resultant adjacency matrix is denoted as $\tilde{A} = A + I$. Most of the GNNs employ self-looped adjacency matrix, however recent publications have shown that a simple/no-loop adjacency matrix is useful for learning heterophily properties in graph [17, 38]. Each node is associated with a d-dimensional feature vector, and the feature matrix for all nodes is represented as $X \in \mathbb{R}^{n \times d}$. The aggregation step is performed with matrix multiplication of adjacency matrix and feature matrix (Figure 1). For K-step aggregation, aggregated features are calculated as $A^k X$, $1 \leq k \leq K$. We denote $\mathcal{X} = \{X, X_1, X_2, X_3, …, X_K\}$ as a set of node feature matrices generated after aggregation step including initial node feature
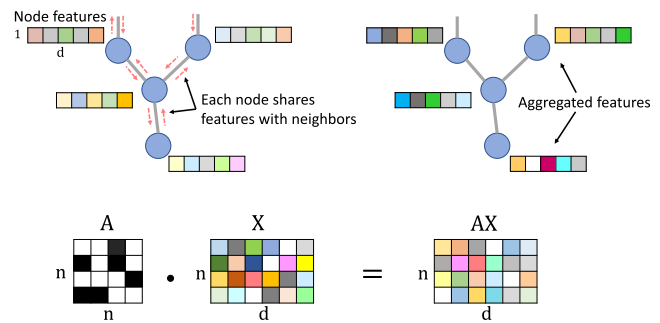

FIGURE 1 Every aggregation step generates additional feature vector for each node.

matrix. Since each of these matrices has a feature vector corresponding to every node, in our discussion, we will interchangeably refer to feature matrices as features of the nodes.

2.2 | Graph neural networks

Graph Neural Networks (GNNs) leverage feature propagation mechanism [6] to aggregate neighbourhood information of a node and use non-linear transformation with trainable weight matrix to get the final embeddings for the nodes. Conventionally, a simple GNN layer is defined as:

$$H^{(i+1)} = \sigma(\tilde{A}_{sym}H^{(i)}W^{(i)}) \qquad (1)$$

where $\tilde{A}_{sym} = \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}$ is a symmetric normalised adjacency matrix with added self-loops. H^i represents features from the previous layer, W^i denotes the learnable weight matrix, and σ is a non-linear activation function, which is usually ReLU in most implementations of GNNs. However, this formulation is suitable for homophily datasets as features are cumulatively aggregated, that is, node's own features are added together with neighbour's features. The cumulative aggregation of node's self-features with that of neighbours reinforces the signal corresponding to the label and helps to improve accuracy of the predictions. While in the case of heterophily, nodes are assumed to have dissimilar features and labels to their neighbours. For heterophily datasets, H2GCN [17] proposes a propagation scheme to separate features of neighbours from node's own features. So we use the following formulation for the GNN layer:

$$H^{(i+1)} = \sigma\left(A_{sym}H^{(i)}W^{(i)}\right) \qquad (2)$$

where $A_{sym} = D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$ is symmetric normalised adjacency matrix without added self-loops. To combine features from multiple hops, concatenation operator can be used before the final layer.

Following the conventional GNN formulation [19] using $\tilde{A}$, a simple 2-layered GNN can be represented as:

$$Z = \tilde{A}_{sym}\sigma(\tilde{A}_{sym}XW^{(0)})W^{(1)} \qquad (3)$$

2.3 | Node classification

Node classification is an extensively studied graph based semi-supervised learning problem. It encompasses training the GNN to predict labels of nodes based on the features and neighbourhood structure of the nodes. GNN model is considered as a function $f(X, A)$ conditioned on node features X and adjacency matrix A. Taking the example of Equation (3), GNN aggregates the features of two hops of neighbours and outputs Z. Softmax function is applied row-wise, and cross-entropy error is calculated over all labelled training examples.

The gradients of loss are back-propagated through the GNN layers. Once trained, the model can be used to predict labels of nodes in the test set.

3 | IMPORTANCE OF HOP FEATURES

In this section, we aim to experimentally study the effect of feature aggregation over multiple hops on the task of node classification. We use nine benchmark node classification datasets (Table 2) with a wide range of structural and homophily properties.

3.1 | Experiment design

In most GNN models, feature aggregation operation is embedded within the GNN layer. It presents a challenge to discern the importance of aggregated node features at various hops. To overcome this difficulty, we separate the feature aggregation step from the neural network model. This approach is similar to models like SGC [20], SIGN [39], etc.

3.1.1 | Node feature subsets

Aggregated node features are calculated by multiplying the adjacency matrix with the node feature matrix. As we analyse both heterophily and homophily properties in a graph, we calculate both self-looped and no-loop features of the nodes. Hence for $K = 3$, the feature set has total of 7 $(1 + 2 \times 3)$ different features for the node: $\mathcal{X} = \{X, AX, (A+I)X, A^2X, (A+I)^2X, A^3X, (A+I)^3X\}$.

3.1.2 | Neural network model

We use a simple two-layered MLP model. Node feature matrices from $\mathcal{X}$ are input in parallel to the first layer. Each feature matrix is linearly transformed and normalised with L2-normalisation. Then, the output is averaged, passed through non-linearity (ReLU), and finally mapped to the second layer. Using this simple and shallow model enables us to study hop features more effectively by avoiding architectural biases of more complex models with deeper layers, skip connections, etc.

3.1.3 | Input settings

For our experiments, the model is trained under the following three settings (Figure 2):

(a) Single setting: Input to the model is a single feature matrix out of 7 from $\mathcal{X}$. This setting helps to identify how informative are individual hop features for the node classification task.

(b) All setting: All feature matrices are used as input to the model. Under this setting, we aim to find how using all features affect the model's learning.

(c) Subset setting: Model is trained on all combinations of features (PowerSet($\mathcal{X}$)\∅) and the best result is reported. One example of such a subset is $\{X, (A + I)X, A^3X\}$ with the total of 119 subsets. This setting shows the effect on model training with a subset of node features while excluding possible noisy features.

3.2 | Experimental observations

Table 1 shows the accuracy values for models trained on different input settings. Following are a few key findings from these results:

- Following results in Single setting, we observe that features at various hops contribute differently. For homophily datasets like Cora, Citeseer and Pubmed, self-looped features have higher classification accuracy. On the other hand, no-looped features are performing better for Chameleon and Squirrel datasets. For other datasets, the node's own features are the best for the model's performance, and graph structure does not play a significant role.
- In the case of All setting, homophily datasets benefit and show improvements in accuracy. However, other datasets
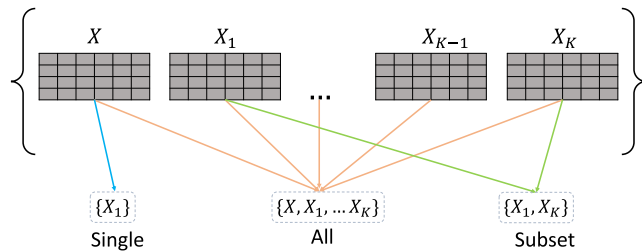
show a significant reduction in accuracy. It implies that for these datasets, using all node features resulted in the addition of noisy and non-informative features causing the model to learn poorly. These results also highlight that a neural network model may not always filter out noisy components in the input, and an explicit feature selection scheme might be helpful.

- In the case of the Subset setting, we observe improvement in the performance of the model on the node classification task for most datasets. This result implies that among all features in $\mathcal{X}$, there are certain features when combined together, are more informative. Please note that in an optimal subset, all features do not contribute equally. However, their combination provides enough information for the model to have the best performance.

Our experimental results show that with a certain optimal subset of node feature matrices, GNNs can achieve higher performance on the node classification task. However, calculating the model's performance for all possible feature subsets is computationally expensive, especially with higher hops due to combinatorial explosion, and practically not feasible.

3.3 | Feature selection in GNN

Problems to select optimal features has been explored in machine learning literature using various feature selection strategies [40, 41]. However, with GNNs, it is more challenging in two ways. First, node features in many datasets are sparse. For example, most datasets in Table 2 has more than 95% sparsity for node feature matrix. Second, the total dimensionality of node features increases rapidly with each aggregation step due to multiple feature matrices after aggregation. Due to these challenges, many feature selection algorithms are not optimal for the task. Furthermore, feature selection algorithms are used for selecting individual indices of input feature vectors. However, in the case of the node classification task, selecting hop-features is more useful as it helps

FIGURE 2 Generation of input feature set in different settings.

TABLE 1 Mean Classification Accuracy on node classification task on 2-layered MLP with hidden dimension as 64

Dataset	Single feature							All	Subset
	X	AX	(A + I)X	A^2X	(A + I)2X	A3X	$(A + I)^3X$		
Cora	73.40	79.55	84.28	83.86	85.47	83.58	85.41	<u>87.50</u>	**88.04**
Citeseer	71.66	69.10	73.53	72.38	74.07	70.55	73.92	<u>77.09</u>	**77.43**
Pubmed	87.79	81.77	88.27	84.70	88.06	83.06	86.63	<u>89.55</u>	**89.83**
Chameleon	46.05	<u>77.74</u>	71.22	76.07	71.77	75.26	71.62	72.25	**78.55**
Wisconsin	<u>87.45</u>	63.13	58.03	62.54	52.94	60.00	51.76	79.8	**88.62**
Texas	<u>85.40</u>	66.21	61.35	67.29	58.64	62.43	58.10	78.91	**88.91**
Cornell	<u>85.94</u>	58.64	63.51	58.64	61.62	58.91	60.27	72.25	**86.75**
Squirrel	30.24	<u>73.18</u>	63.79	71.28	63.37	64.42	62.82	64.68	**73.12**
Actor	35.32	25.47	29.22	25.38	27.95	25.27	26.43	<u>35.39</u>	**35.67**

Note: Results with Subset setting (bold) are compared with best results (underlined) in other input feature settings.

to establish the similarity/dissimilarity of nodes from their neighbours. Hence it becomes challenging to use feature selection algorithms, and there is a need for new approaches to tackle this problem.

3.4 | Problem formulation

Given the input $\mathcal{X}$ as a set of $2K + 1$ node feature matrices and $\mathcal{Y}$ as labels of the nodes. We are interested in identifying a subset of $\mathcal{X}$ of length up to $p < 2K + 1$, for which the GNN model gives the best performance. All such possible subsets are defined as $\mathcal{M}$ and $|\mathcal{M}| = \sum_{j=1}^{p} \binom{2K+1}{j}$. Let $m \in \mathcal{M}$ denote any one of such possible subsets. Our objective is to identify the optimal subset m^* for which GNN model performs best.

4 | DUAL-NET GNN ARCHITECTURE

As discussed in Section 3, the GNN model needs to have the capability to select useful node features while discarding noisy/uninformative features. In this section, we propose a new GNN model that takes all node feature matrices as input and during training learns to select optimal node feature set for the task and discards the remaining ones. Details of the model architecture and training methodology are as follows:

4.1 | Model description

The model consists of two separate neural networks: *classifier* network and *selector* network.

4.1.1 | Classifier Network

It is a two-layered MLP neural network, similar to the one used in Section 3.1 for predicting labels of the nodes. The classifier

network is parameterised with θ, represented as $f_c(\theta; \mathcal{X}, m)$. The input to the network is a subset of node feature matrices, $m \in \mathcal{M}$. In the first layer, each input matrix is linearly transformed and the output is summed. The non-linearity (ReLU) is applied and then mapped to the second layer. After training of the joint model, $f_c(\theta^*; \mathcal{X}, m^*)$ is applied on the test split of the dataset, where θ^* denotes learned parameters of the classifier and m^* is the optimal input subset of node feature matrices.

4.1.2 | Selector Network

The selector network is also a two-layered MLP, yet with a different input/output configuration. The role of the selector is to find out an optimal feature subset of $\mathcal{X}$ based on the performance feedback of the classifier. The selector model is parameterised with ϕ, represented as $f_s(\phi, m)$. The input to the selector network is a one-hot encoding vector $\overrightarrow{m}$ of dimension $2K + 1$ representing the subset of feature matrices to be selected as per m. For example, for $K = 3$, the subset $\{X, (A + I)X, A^3X\}$ has $\overrightarrow{m} = (1, 0, 1, 0, 0, 1, 0)$. The output is a single scalar value. The purpose of the selector net is to learn to predict the average performance of the classifier to the mask corresponding to the input subset.

Both networks are trained jointly in an alternate manner. However, the training objective of both models are different.

4.2 | Loss functions

4.2.1 | Classifier loss

The classifier is trained with simple Cross-Entropy loss commonly used in the node classification task. For each training step, the input is a subset of node features, $m \in \mathcal{M}$. In implementation, it represents the masking operation of input features to the model. Loss is calculated between output logits of the model and true labels of the nodes,

$$\mathcal{L}_c(\mathcal{X}, m; \theta) = \frac{1}{|V|} l(\mathcal{X}, m, y; \theta) \quad (4)$$

4.2.2 | Selector loss

The loss of selector is MSE (mean squared error) loss between the output of selector, $\mathcal{O}_s = f_s(\phi; \overrightarrow{m})$ and the classifier loss value $\mathcal{L}_c$,

$$\mathcal{L}_s(\overrightarrow{m}; \phi) = \left(f_s(\phi; \overrightarrow{m}) - \mathcal{L}_c(\mathcal{X}, m; \theta)\right)^2 \quad (5)$$

The selector learns to map the input mask vector to the classifier's loss. Thus the output of the selector varies with different input subset masks in congruence with the classifier's performance. With a good input subset, the classifier will have a lower loss, and correspondingly selector will have a lower output value and vice versa.

TABLE 2 Statistics of the node classification datasets

Datasets	Homophily Ratio	Nodes	Edges	Features	Feature Sparsity (%)	Classes
Cora	0.81	2708	5429	1433	98.7	7
Citeseer	0.74	3327	4732	3703	99.1	6
Pubmed	0.80	19,717	44,338	500	90.0	3
Chameleon	0.23	2,277	36,101	2325	99.4	4
Wisconsin	0.21	251	499	1703	94.4	5
Texas	0.11	183	309	1703	95.1	5
Cornell	0.30	183	295	1703	95.1	5
Squirrel	0.22	5,201	198,353	2089	99.1	5
Actor	0.22	7,600	26,659	932	99.4	5

4.3 │ Training methodology

Our objective in training is to start with all input feature matrices in $\mathcal{X}$, and during the process, learn the importance of each feature matrix. Based on the feature ranking, we then aim to select the optimal subset m^* to get the maximum prediction accuracy. We train our model Dual-Net GNN in three stages, with each stage having a different objective Figure 3 (Algorithm 1).

Stage 1. First stage is the exploration stage, where the model is trained on possible combinations of node feature matrices. The objective is to explore the performance characteristics of subsets of $\mathcal{M}$ in terms of the loss value of the classifier. For each forward pass, a random subset m sampled from $\mathcal{M}$, and corresponding node feature matrices are used as input to the classifier. Similarly, one-hot encoded vector representation $\overrightarrow{m}$ is set as input to the selector. Cross-entropy loss for the classifier is calculated and backpropagated. For the selector, MSE loss is calculated between the output of the selector $\mathcal{O}_s$ and loss of the classifier $\mathcal{L}_c$ and backpropagated to update selector network weights. The training is continued in this manner till the classifier yields different losses for different mask combinations stably, and the selector learns to map these masks to the classifier's performance. The number of training epochs in this stage is set as a hyperparameter, $E1$.
Stage 2. In this stage, our objective is to exploit the learned selector to identify indices of useful feature matrices and generate possible optimal masks for the classifier. In machine learning literature, given a trained model, gradients with respect to the input are often used to identify important input components [42–44]. The input components with larger gradients contribute more towards the model's output. Similarly, we aim to identify important indices of the input mask vector to the trained selector model. The node feature matrices corresponding to these indices would lead to better performance of the classifier model.

We use a mask vector with all indices initialised with equal initial weights $\left(\frac{1}{2} \cdots \frac{1}{2}\right)$ as input to the selector meaning all indices have an equal chance of being selected. We then calculate the gradients of the input vector with respect to the model. Indices with top-p gradients (i_{top-p}) are identified. It reduces the number of indices to consider from $2K + 1 \rightarrow p$. However, the optimal subset m^* might be a smaller subset of these indices, that is, $len(m^*) \leq p$ and the number of possible candidates for optimal subset is $\sum_{j=1}^{p} \binom{p}{j}$. A fixed number of combinations of these indices are sampled, and validation loss on the classifier is calculated. Mask with the lowest loss value is selected, and both classifier and selector are trained on it for one epoch. This step is repeated each epoch, and the model is trained for a fixed number of times, $E2$.
Stage 3. After Stage 2, the input mask with the lowest validation loss is identified as m^*. The training is continued with only the classifier with node feature matrices corresponding to m^* as input. The selector model is not utilized in this stage. The training of the classifier is continued till the stopping criterion is satisfied.

Algorithm 1 Pseudo Code Dual-Net GNN

Input : Classifier $f_c(\theta; \mathcal{X}, m)$, Selector $f_s(\phi; \overline{m})$

Input : Node feature set $\mathcal{X}$, Max. subset size p,
Stage 1 epochs E_1, Stage 2 epochs E_2,
hidden dimensions h

Output : Logits (Classifier)

/* Stage-1 */

1 **for** $ii \leftarrow 0$ **to** $E1$ **do**
2   $\quad m = Random(\mathcal{M}, i_{all}, p)$
3   $\quad \mathcal{L}_c(\mathcal{X}, m; \theta) = \frac{1}{|V|} l(\mathcal{X}, m, y; \theta)$
4   $\quad \theta'' \triangleq \theta' - \eta \nabla_\theta \mathcal{L}_c(\mathcal{X}, m; \theta)|_{\theta=\theta'}$
5   $\quad \mathcal{L}_s(\overline{m}; \phi) = \left(f_s(\phi; \overline{m}) - \mathcal{L}_c(\mathcal{X}, m; \theta) \right)^2$
6   $\quad \phi'' \triangleq \phi' - \eta \nabla_\phi \mathcal{L}_s(\overline{m}; \phi)|_{\phi=\phi'}$
7 **end**

/* Stage-2 */

8 $m_{eq} = \left(\frac{1}{2}, \frac{1}{2}, ..., \frac{1}{2} \right)$
9 $\delta_{m_{eq}} = \frac{\partial f_s(\phi, m)}{\partial m}|_{m=m_{eq}}$
10 $i_{top-p} = \text{argsort}(\delta_{m_{eq}})[-p :]$
11 **for** $jj \leftarrow 0$ **to** $E2$ **do**
12 $\quad m_p = Random(\mathcal{M}, i_{top-p}, p)$
13 $\quad \mathcal{L}_c(\mathcal{X}, m_p; \theta) = \frac{1}{|V|} l(\mathcal{X}, m_p, y; \theta)$
14 $\quad \theta'' \triangleq \theta' - \eta \nabla_\theta \mathcal{L}_c(\mathcal{X}, m_p; \theta)|_{\theta=\theta'}$
15 $\quad \mathcal{L}_s(\overrightarrow{m_p}; \phi) = \left(f_s(\phi; \overrightarrow{m_p}) - \mathcal{L}_c(\mathcal{X}, m_p; \theta) \right)^2$
16 $\quad \phi'' \triangleq \phi' - \eta \nabla_\phi \mathcal{L}_s(\overrightarrow{m_p}; \phi)|_{\phi=\phi'}$
17 **end**

/* Stage-3 */

18 $m^* = \text{argmin}(\mathcal{L}_{c(val)})$
19 **for** $kk \leftarrow 0$ **to** $E3$ **do**
20 $\quad \mathcal{L}_c(\mathcal{X}, m^*; \theta) = \frac{1}{|V|} l(\mathcal{X}, m^*, y; \theta)$
21 $\quad \theta'' \triangleq \theta' - \eta \nabla_\theta \mathcal{L}_c(\mathcal{X}, m^*; \theta)|_{\theta=\theta'}$
22 $\quad$ **if** $checkEarlyStopping(\mathcal{L}_c(\mathcal{X}, m^*; \theta))$ **then**
23   $\quad\quad \theta_t = restoreBestweights()$
24   $\quad\quad$ break
25 $\quad$ **end**
26 **end**

---

5 │ RELATED WORK

In recent years, GNNs have become one of the most important tools to solve the node classification problem. GNNs often provide high performance with end-to-end training with
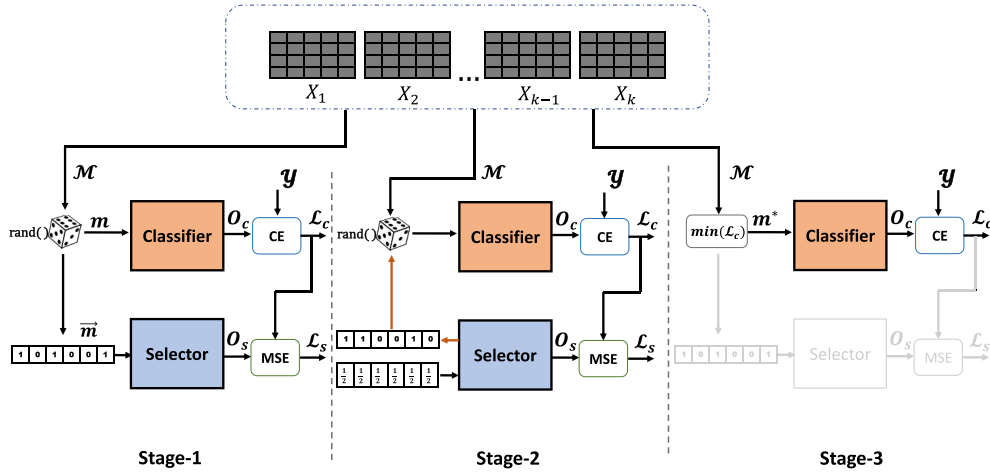
FIGURE 3 The model operation in 3-stages. Classifier and selector are trained jointly in Stage 1 & 2, while only classifier is trained in Stage 3.

little manual intervention. In addition, they can take advantage of recent hardware improvements, especially GPUs, to reduce the training time and yield faster predictions. [19, 26] proposed one of the earliest GNN based end-to-end training framework using approximations of spectral graph convolutions. Since then, there has been tremendous progress in the field with a wide range of GNN models aiming to tackle different problems. GCN [19] uses a fixed propagation scheme to aggregate features from all neighbours. However, it makes it challenging to scale the model for large graphs. To solve this problem, GraphSAGE [22] and FastGCN [23] introduces neighbour sampling approach. GAT [24] improves GNN architecture by using attention mechanism. APPNP [31], JK [32], Geom-GCN [16], SimP-GCN [45], and CPGNN [18] aim to improve the feature propagation scheme within layers of the model. More recently, researchers are proposing to make GNN models deeper [27, 29, 30]. However, deeper models suffer from over-smoothing, where after stacking many GNN layers, features of the node become indistinguishable from each other, and there is a drop in the performance of the model. DropEdge [46] proposes to drop a certain number of edges to reduce the speed of convergence of over-smoothing and relieves the information loss. GCNII [27] use residual connections and identity mapping in GNN layers to enable deeper networks. RevGNN [29] uses deep reversible architectures and [30] uses noise regularisation to train deep GNN models. Recently researchers have proposed new datasets that do not follow homophily assumption in traditional GNN models. Several models propose to handle heterophily properties in graph data: H2GCN [17], CPGNN [18], TDGNN [47], Geom-GCN [16], FSGNN [48],GPRGNN [28], and GloGNN [49]. However, a recent work [38] reveals that GCNs can achieve strong performance on heterophily graphs under certain conditions.

In machine learning field, feature selection has been used a way to discard noisy features to improve performance [35, 41, 50–52]. It provides computational benefits (lower storage and faster computation) and statistical benefits including increased model interpretability [53]. There are many supervised feature selection algorithms in use. LASSO [54], random forest (RF)

[55], RFE (Recursive Feature Elimination) [56] is one of the off-the-shelf method. HSIC LASSO (Hilbert Schmidt Independence Criterion Lasso) [57] and CCM (Conditional Covariance Minimisation) [53] are two kernel based methods for feature selection. Recently neural network based models [41, 58] for feature selection have been proposed.

6 | EXPERIMENTS

In this section, we evaluate the results of experiments with our proposed model on real-world node classification datasets. We compare the performance of our model with other GNN models.

6.1 | Datasets

We perform experiments on fully-supervised node classification task with nine benchmark datasets commonly used in GNN literature. Details of these datasets are provided in Table 2. Homophily ratio [17] denotes the fraction of edges that connects two nodes of the same label. A higher value (closer to 1) indicates strong homophily, while a lower value (closer to 0) indicates strong heterophily in the dataset. Cora, Citeseer, and Pubmed [59] are citation networks based datasets and, in general, are considered as homophily datasets. Graphs in Wisconsin, Cornell, Texas [16] represent links between web-pages, Actor [60] represent actor co-occurrence in Wikipedia pages, Chameleon and Squirrel [61] represent the web pages in Wikipedia discussing corresponding topics. These datasets are considered as heterophily datasets. To provide a fair comparison, we use publicly available data splits taken from [16].[1] We compare the experiment results of other GNN models with our proposed model on this same split.

---

[1]https://github.com/graphdml-uiuc-jlu/geom-gcn.

6.2 | Preprocessing

We follow the same preprocessing steps used by [16, 27]. Other models to which we compare our results also follow the same set of procedures. Initial node features are row-normalised. To account for both homophily and heterophily, we use the adjacency matrix and adjacency matrix with added self-loops for feature transformation. Both matrices are symmetrically normalised. For efficient computation, adjacency matrices are stored and used as sparse matrices.

6.3 | Settings and baselines

For a fully-supervised node classification task, each dataset is split evenly for each class into 48%, 32%, and 20% for training, validation, and testing [16, 17]. We report the performance as mean classification accuracy over 10 random splits. To provide a comprehensive evaluation of our model's performance, we compare the accuracy results with feature selection methods as well as graph neural network models.

6.3.1 | Feature Selection Baselines

We ran experiments on 4 feature selections methods: RFE [56], HSIC LASSO [57], CCM [53], and DFS (Deep Feature Selection) [58]. RFE, HSIC LASSO, and CCM algorithms work directly on feature matrices. Hence, we concatenated all pre-computed seven node feature matrices to get a single feature matrix, $\mathcal{X}_{cat} \in \mathbb{R}^{n \times 7d}$ and provide it as input along with labels of the nodes in the training split. For parity with our model, we set the number of dimensions that can be selected by the algorithms up to pd. Due to intensive memory requirements with high dimensional data, feature selection algorithms were run on a machine with 800 GB memory. Once the feature indices are selected by the algorithm, we trained a two-layered MLP (similar to the classifier in our model) on selected features and report the prediction accuracy. DFS is a neural network based feature selection model. We modified the DFS model to select hop features similar to our model keeping other architectural details and loss functions same.

6.3.2 | GNN Baselines

We compare our model to 13 different GNN baselines and use the published results as the best performance of these models. GCNII [27], TDGNN [47], H2GCN [17] and GloGNN [49] have proposed multiple variants of their model. We have chosen the variant with the best performance on most datasets. GPRGNN uses random splits in their published results. For a fair comparison, we ran their publicly available code on our standard splits while keeping other settings the same. Hyperparameters of the model training are provided in Appendix A.2.

6.4 | Results and discussion

6.4.1 | Accuracy results

Table 3 shows the comparison of the mean classification accuracy of our model with other feature selection methods and GNN methods. These feature selection methods overall do not perform better despite high execution time and intensive memory requirements. CCM method runs out of memory on many datasets despite using a high memory machine. The performance of DFS is comparable to a few GNN models, however, it still is significantly lower than our model. Comparing the results with other GNNs, we observe that our model overcomes the effect of noisy features and significantly improves the performance on Chameleon (+10.2%), Wisconsin (+1.6%), Texas (+4.19%), Cornell (+2.51%) and Squirrel (+27.8%) datasets. For Cora (−0.8%), Citeseer (−1.1%), Pubmed (−0.7%), and Actor (−1.1%), the performance of our model is on par with other GNN models. Figure 4 shows the plot of node embeddings learned by the model on Squirrel and Chameleon datasets.

Comparing the results of All Setting with Subset setting, for homophily datasets, the difference for Cora (0.54%), Citeseer (0.34%), and Pubmed (0.28%) is significantly smaller compared to heterophily datasets: Chameleon (6.30%), Wisconsin (8.82%), Texas (10.00%), Cornell (14.5%), Squirrel (8.44%), and Actor (0.28%). From these results, we infer that the effect of noisy features is less in homophily datasets while in most heterophily datasets the noisy features significantly affect the model's performance. As our proposed model is designed to identify possible noisy features and learn on relevant informative features, the improvements are more apparent with the heterophily datasets with Chameleon (+6.21%), Wisconsin (+9.61%), Texas (+8.66%), Cornell (+15.86%), Squirrel (+9.29%) and Actor (+1.9%). Comparing with results in All Settings, our model improves performance in all homophily datasets, Cora (+0.27%), Citeseer (+0.06%), and Pubmed (+0.09%) although these improvements are smaller when compared with heterophily datasets. These observations indicate that the model performs well if the noisy features present in the input cause a noticeable difference in the performance. On the other hand, it can be challenging to detect noisy features that have a relatively smaller effect on the performance of the model.

6.4.2 | Training time

Due to its simple architecture, the training time of our model is relatively low. Figure 5 shows a comparison of the training time of our model with a few SOTA GNN models on commonly used benchmark datasets Cora, Citeseer, and Pubmed. Training time of our model is similar to GCN and GPRGNN. Due to its deep structure with up to 64 layers, the training time of GCNII is one order magnitude higher than our model. For feature selection methods, most of the execution time is spent on running feature selection algorithm. In our experiments, feature selection algorithms took

TABLE 3 Mean classification accuracy on fully-supervised node classification task

	Cora	Citeseer	Pubmed	Chameleon	Wisconsin	Texas	Cornell	Squirrel	Actor	Mean acc.
RFE	85.31 ± 1.47	75.24 ± 1.51	88.71 ± 0.38	67.19 ± 1.35	71.18 ± 2.91	69.46 ± 6.29	64.86 ± 5.80	51.93 ± 2.03	34.17 ± 1.19	67.56
HSIC LASSO	84.55 ± 2.00	73.48 ± 1.75	89.19 ± 0.51	63.22 ± 2.63	79.02 ± 3.83	79.73 ± 8.31	77.03 ± 7.07	44.41 ± 1.69	33.96 ± 1.60	69.40
CCM	85.47 ± 1.12	OOM	OOM	67.48 ± 1.54	72.35 ± 3.66	70.81 ± 4.80	64.59 ± 5.60	OOM	OOM	-
DFS	85.31 ± 1.47	75.24 ± 1.51	88.71 ± 0.38	67.19 ± 1.35	71.18 ± 2.91	72.16 ± 10.18	65.95 ± 11.85	60.11 ± 8.05	33.79 ± 1.94	69.93
MLP	75.69 ± 2.00	74.02 ± 1.90	87.16 ± 0.37	46.21 ± 2.99	85.29 ± 3.31	80.81 ± 4.75	81.89 ± 6.40	28.77 ± 1.56	36.53 ± 0.70	67.56
GCN	87.28 ± 1.26	76.68 ± 1.64	87.38 ± 0.66	59.82 ± 2.58	59.80 ± 6.99	59.46 ± 5.25	57.03 ± 4.67	36.89 ± 1.34	30.26 ± 0.79	61.62
GAT	82.68 ± 1.80	75.46 ± 1.72	84.68 ± 0.44	54.69 ± 1.95	55.29 ± 8.71	58.38 ± 4.45	58.92 ± 3.32	30.62 ± 2.11	26.28 ± 1.73	58.55
GraphSAGE	86.90 ± 1.04	76.04 ± 1.30	88.45 ± 0.50	58.73 ± 1.68	81.18 ± 5.56	82.43 ± 6.14	75.95 ± 5.01	41.61 ± 0.74	34.23 ± 0.99	69.50
Cheby + JK	85.49 ± 1.27	74.98 ± 1.18	89.07 ± 0.30	63.79 ± 2.27	82.55 ± 4.57	78.38 ± 6.37	74.59 ± 7.87	45.03 ± 1.73	35.14 ± 1.37	69.89
MixHop	87.61 ± 0.85	76.26 ± 1.33	85.31 ± 0.61	60.50 ± 2.53	75.88 ± 4.90	77.84 ± 7.73	73.51 ± 6.34	43.80 ± 1.48	32.22 ± 2.34	68.10
GEOM-GCN	85.27	**77.99**	90.05	60.90	64.12	67.57	60.81	38.14	31.63	64.05
GCNII	88.01 ± 1.33	77.13 ± 1.38	**90.30 ± 0.37**	62.48 ± 2.74	81.57 ± 4.98	77.84 ± 5.64	76.49 ± 4.37	N/A	N/A	-
H2GCN–1	86.92 ± 1.37	77.07 ± 1.64	89.40 ± 0.34	57.11 ± 1.58	86.67 ± 4.69	84.86 ± 6.77	82.16 ± 4.80	36.42 ± 1.89	35.86 ± 1.03	70.71
TDGNN-w	88.01 ± 1.32	76.58 ± 1.40	89.22 ± 0.41	46.31 ± 2.42	85.57 ± 3.78	83.00 ± 4.50	82.92 ± 6.61	26.73 ± 1.47	37.11 ± 0.96	68.38
WRGAT	88.20 ± 2.26	76.81 ± 1.89	88.52 ± 0.92	65.24 ± 0.87	86.98 ± 3.78	83.62 ± 5.50	81.62 ± 3.90	48.85 ± 0.78	36.53 ± 0.77	72.93
GGCN	87.95 ± 1.05	77.14 ± 1.45	89.15 ± 0.37	71.14 ± 1.84	86.86 ± 3.29	84.86 ± 4.55	85.68 ± 6.63	55.17 ± 1.58	37.54 ± 1.56	75.05
GPRGNN	**88.49 ± 0.95**	77.08 ± 1.63	88.99 ± 0.40	66.47 ± 2.47	85.88 ± 3.70	86.49 ± 4.83	81.89 ± 6.17	49.03 ± 1.28	36.04 ± 0.96	73.37
GloGNN++	88.33 ± 1.09	77.22 ± 1.78	89.24 ± 0.39	71.21 ± 1.84	88.04 ± 3.22	84.05 ± 4.90	85.95 ± 5.81	57.88 ± 1.76	**37.70 ± 1.40**	75.51
Dual-Net GNN	87.77 ± 1.16	77.15 ± 1.58	89.64 ± 0.43	**78.46 ± 0.66**	**89.41 ± 3.64**	**87.57 ± 3.24**	**88.11 ± 5.69**	**73.97 ± 1.89**	37.29 ± 0.80	**78.81**

Note: Results for MLP, GCN, GAT, GraphSAGE, Cheby + JK, MixHop and H2GCN–1 are taken from [17, 49]. For GEOM-GCN, GCNII, TDGNN, WRGAT, and GloGNN++, results are taken from the respective article. Best performance for each dataset is marked as bold. OOM means algorithm run out of memory.

FIGURE 4 t-SNE plots of node embeddings by GPRGNN (left) and Dual-Net Graph Neural Network (GNN) (right) models on Squirrel and Chameleon datasets.
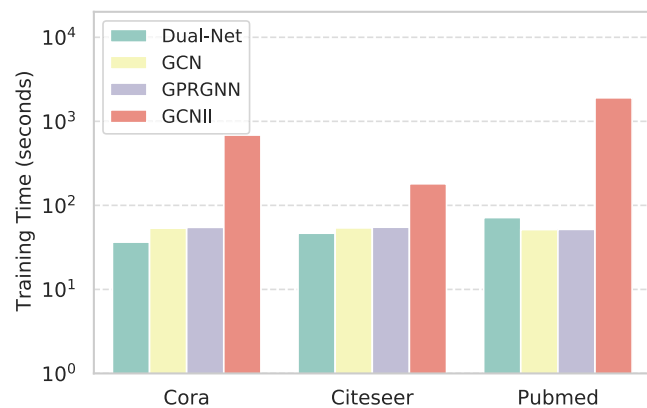

FIGURE 5 Comparison of training time of Graph Neural Network (GNN) models on Cora, Citeseer and Pubmed datasets. Time (seconds) is plotted in log scale due to the training time of GCNII being significantly higher than other models.

FIGURE 6 Heatmap of indices of hop features selected for all 10 splits for each dataset.

significantly higher time (>2+ hours) than graph neural network models.

6.4.3 | Analysis of selected feature subsets

In this section, we provide an analysis of the feature subset selected by our model during training. For each dataset, we train the model on 10 random splits. As the training/validation/test nodes are different for each split, the model learns the optimal feature subset separately for each split. Figure 6 shows the heatmap of hop feature matrices selected for all 10
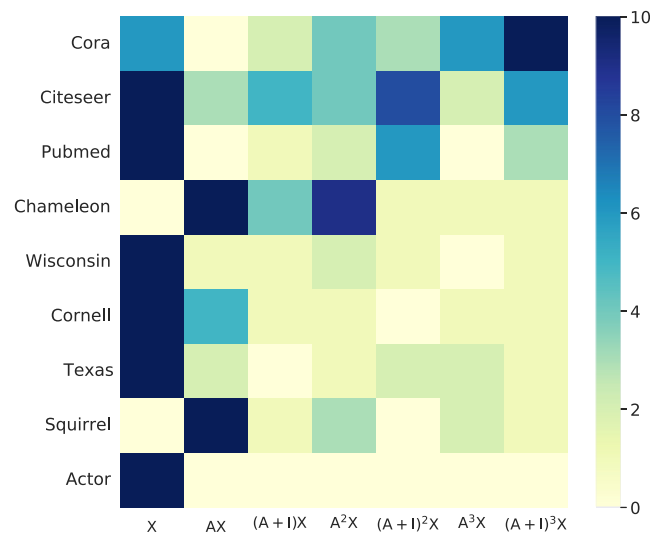
splits. For homophily datasets Cora, Citeseer, and Pubmed, we observe that the model learnt to select node's own features and self-looped feature matrices as optimal subsets. For Chameleon and Squirrel datasets, the model learnt to select no-looped feature matrices. For Wisconsin, Cornell, Texas, and Actor datasets, the model learnt to predominantly select node's own features as optimal features. These observations also match our experiment results in Table 1. It shows that our proposed model reliably learns to select appropriate hop features of the nodes on a wide variety of benchmark datasets.
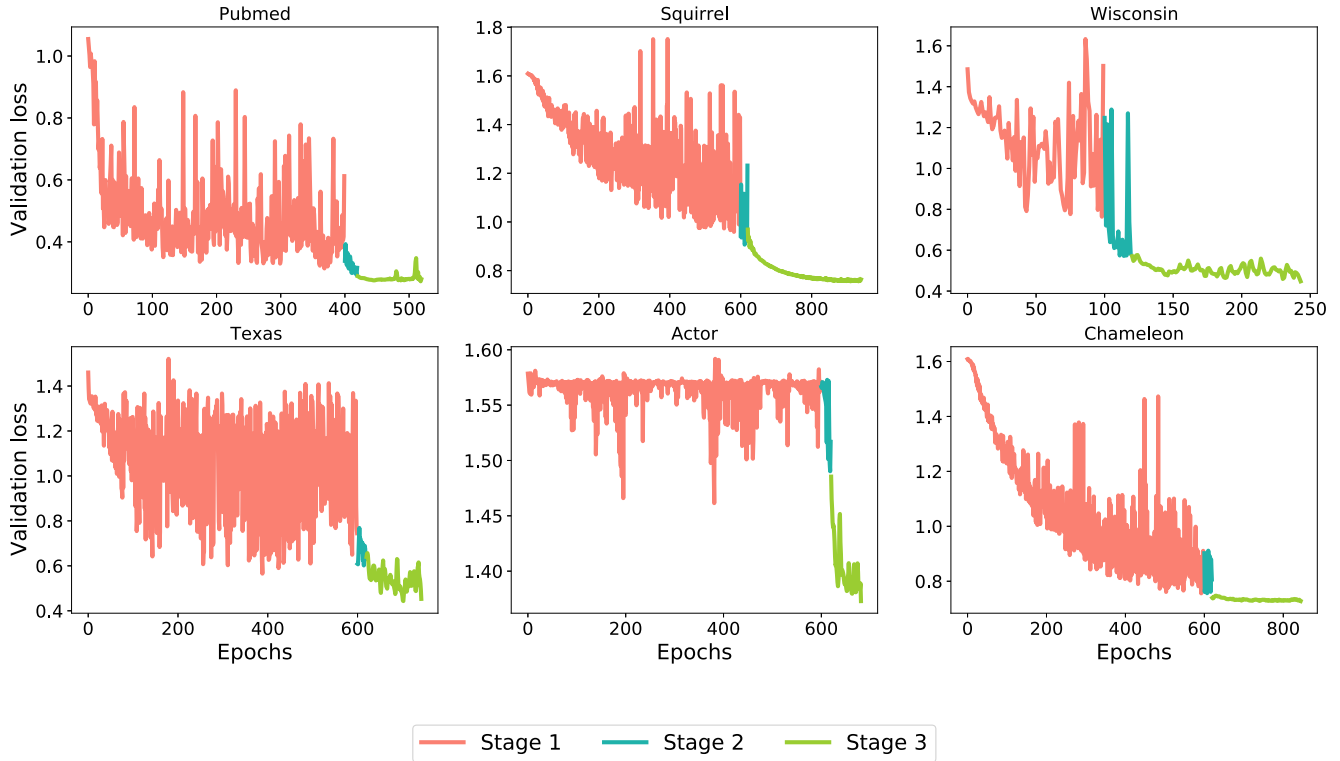
FIGURE 7 Validation loss of the model under different stages of the training.

6.4.4 | Analysis of validation loss during training

In this section, we discuss training dynamics of the classifier in terms of the validation loss. As discussed in Section 4.3, training of the model is divided into three stages.

Stage 1 During this stage, for each epoch a random subset of node feature matrices is selected from $\mathcal{X}$. The quality of the subset with respect to the label can vary among all subsets. Hence, in Figure 7, we observe wide fluctuations in the validation loss of the classifier during Stage 1 training. However, since the classifier is training against the labels, overall the average loss is decreasing albeit, a bit slowly in some datasets.

Stage 2 In Stage 2, top-p best indices and corresponding node feature matrices are identified. Subset of feature matrices based on these top-p features are randomly sampled to train the model. During Stage 2, we observe significantly less fluctuations in the validation loss as well as rapid drop in loss value.

Stage 3 In this stage, we identify the optimal input feature subset (m^*) to train the classifier. The validation loss rapidly drops and significantly lower than Stage 1.

6.4.5 | Model complexity and scalability

Both networks in our model are simple two-layered MLPs. However, with more complex input features, the model can be generalised to have more layers. Hence, the model complexity

is similar to a MLP network. Assuming the model with L_{ff} number of feedforward layers and r input feature matrices, the computational complexity of the classifier would be $\mathcal{O}\left(rL_{ff}Nd^2\right)$ [33, 39, 62]. N is the number of the nodes, and d is feature dimensionality. For the selector network with row vector as input, the complexity is $\mathcal{O}\left(L_{ff}d^2\right)$.

The datasets used in our main experiments consist of graphs with a few thousand of nodes. In real-world applications, the graph size can be a few hundred of thousands of nodes to millions of nodes. Hence, the benchmarking datasets for GNN models need to be much larger than commonly used datasets. Recently [37] has proposed new benchmark datasets with large graph sizes with fixed data splits and also provided benchmark results on various GNN models. Statistics of the datasets are provided in Table A4. To demonstrate the scalability of our model, we run experiments with Dual-Net GNN model on these datasets. Table 4 shows the performance comparison of our model with other SOTA models. Our model, in general, outperforms other GNN models on most datasets. For more details please refer to Appendix A.3.

7 | CONCLUSION

In this work, we demonstrate that feature aggregation steps in GNN can generate noisy features for nodes, which in turn can reduce the performance of the GNN model. We tackle this challenge by proposing a Dual-Net GNN architecture to learn

T A B L E 4 Experimental results on large node classification datasets

	Penn94	pokec	arXiv-year	snap-patents	genius	twitch-gamers	Mean val.
MLP	73.61 ± 0.40	62.37 ± 0.02	36.70 ± 0.21	31.34 ± 0.05	86.68 ± 0.09	60.92 ± 0.07	58.60
GCN	82.47 ± 0.27	75.45 ± 0.17	46.02 ± 0.26	45.65 ± 0.04	87.42 ± 0.37	62.18 ± 0.26	66.53
GAT	81.53 ± 0.55	71.77 ± 6.18	46.05 ± 0.51	45.37 ± 0.44	55.80 ± 0.87	59.89 ± 4.12	60.07
GCNJK	81.63 ± 0.54	77.00 ± 0.14	46.28 ± 0.29	46.88 ± 0.13	89.30 ± 0.19	63.45 ± 0.22	67.42
GATJK	80.69 ± 0.36	71.19 ± 6.96	45.80 ± 0.72	44.78 ± 0.50	56.70 ± 2.07	59.98 ± 2.87	59.86
APPNP	74.33 ± 0.38	62.58 ± 0.08	38.15 ± 0.26	32.19 ± 0.07	85.36 ± 0.62	60.97 ± 0.10	58.93
H2GCN	OOM	OOM	49.09 ± 0.10	OOM	OOM	OOM	-
MixHop	83.47 ± 0.71	81.07 ± 0.16	51.81 ± 0.17	52.16 ± 0.09	90.58 ± 0.16	65.64 ± 0.27	70.79
GPR-GNN	81.38 ± 0.16	78.83 ± 0.05	45.07 ± 0.21	40.19 ± 0.03	90.05 ± 0.31	61.89 ± 0.29	66.24
GCNII	82.92 ± 0.59	78.94 ± 0.11	47.21 ± 0.28	37.88 ± 0.69	90.24 ± 0.09	63.39 ± 0.61	66.76
WRGAT	74.32 ± 0.53	OOM	OOM	OOM	OOM	OOM	-
LINKX	84.71 ± 0.52	82.04 ± 0.07	56.00 ± 1.34	61.95 ± 0.12	90.77 ± 0.27	66.06 ± 0.19	73.59
GloGNN++	85.74 ± 0.42	**83.05 ± 0.07**	54.79 ± 0.25	62.03 ± 0.21	90.91 ± 0.13	66.34 ± 0.29	73.81
Dual-Net GNN	**86.09 ± 0.56**	81.55 ± 0.09	**62.65 ± 0.39**	**70.22 ± 0.44**	**91.45 ± 0.12**	**66.36 ± 0.11**	**76.39**

Note: Mean Test accuracy is shown for most datasets, while genius shows test ROC AUC. GloGNN++ & WRGAT results are taken from [49] and results for other models are taken from [37]. Average is taken over default five train/val/test splits. OOM denotes experiment ran out of memory. Best result for a dataset is shown in bold.

the optimal subset of hop features for better performance of the classifier on the node classification task. With extensive experiments, we show that our proposed model can reduce the effect of noisy features by selecting optimal feature subset and outperforms feature selection methods and SOTA GNN models. In the current approach, we use MLP models for classifier and selector. However, more sophisticated architectures with the use of attention layers, residual layers, deeper layers, etc., can be used to further improve the performance of the model. In this paper, we have primarily focussed on homogenous graphs datasets with particular focus on homophily and heterophily properties. Real-world data can come from multitude of domains like e-commerce systems, financial systems, biological networks, contact networks, etc., and can be represented as different types of graphs like multi-layered graphs, heterogenous graphs, dynamic graphs, etc. In these graphs, there can different types of nodes and edges (e.g. heterogenous graphs) that have varying attributes that may or may not be relevant to the task at hand. Our current framework can be extended to these datasets by modifying the preprocessing and feature aggregation step relevant to the dataset and using the propose model to learn from informative features and avoid the noisy features. We consider exploring sophisticated model architecture and extending the model to different new tasks as the direction for our future work.

ACKNOWLEDGEMENTS

CONFLICT OF INTEREST
The authors declare that there is no conflict of interest.

DATA AVAILABILITY STATEMENT
We provide source code and associated data for our experiments via GitHub link.

ORCID
Sunil Kumar Maurya https://orcid.org/0000-0001-8839-8559

REFERENCES
1. Fan, W., et al.: Graph neural networks for social recommendation. In: The World Wide Web Conference WWW '19, pp. 417–426. Association for Computing Machinery (2019). https://doi.org/10.1145/3308558.3313488
2. Ying, R., et al.: Graph convolutional neural networks for web-scale recommender systems. In: KDD '18 (2018)
3. Wu, S., et al.: Graph Neural Networks in Recommender Systems: A Survey (2022). http://arxiv.org/abs/2011.02260
4. Li, Y., et al.: Graph matching networks for learning the similarity of graph structured objects. In: ICML (2019)
5. Ying, R., et al.: Hierarchical graph representation learning with differentiable pooling. In: NIPS'18, pp. 4805–4815 (2018). http://dl.acm.org/citation.cfm?id=3327345.3327389
6. Gilmer, J., et al.: Neural message passing for quantum chemistry. In: Proceedings of the 34th International Conference on Machine Learning - Volume 70 ICML'17 (2017). http://dl.acm.org/citation.cfm?id=3305381.3305512
7. Madhawa, K., et al.: GraphNVP: An Invertible Flow Model for Generating Molecular Graphs (2019). http://arxiv.org/abs/1905.11600
8. Jin, W., et al.: Learning Multimodal Graph-To-Graph Translation for Molecular Optimization (2019). http://arxiv.org/abs/1812.01070
9. Yin, Y., et al.: A novel graph-based multi-modal fusion encoder for neural machine translation. In: ACL (2020)

10. Guo, Z., Zhang, Y., Lu, W.: Attention guided graph convolutional networks for relation extraction. In: Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, pp. 241–251. Association for Computational Linguistics (2019). https://aclanthology.org/P19-1024

11. Yang, J., et al.: Graph R-CNN for scene graph generation. In: Ferrari, V., et al. (eds.) Computer Vision – ECCV 2018 Lecture Notes in Computer Science, pp. 690–706. Springer International Publishing (2018)

12. Woo, S., et al.: LinkNet: relational embedding for scene graph. In: Proceedings of the 32nd International Conference on Neural Information Processing Systems NIPS'18, pp. 558–568. Curran Associates Inc. (2018)

13. Pfaff, T., et al.: Learning Mesh-Based Simulation with Graph Networks (2021). http://arxiv.org/abs/2010.03409

14. Maurya, S.K., Liu, X., Murata, T.: Fast approximations of betweenness centrality using graph neural networks. In: International Conference on Information and Knowledge Management (CIKM) (2019)

15. Maurya, S.K., Liu, X., Murata, T.: Graph neural networks for fast node ranking approximation. ACM Trans. Knowl. Discov. Data 15(5), 78: 1–78:32 (2021). https://doi.org/10.1145/3446217

16. Pei, H., et al.: Geom-GCN: geometric graph convolutional networks. In: ICLR (2020)

17. Zhu, J., et al.: Beyond homophily in graph neural networks: current limitations and effective designs. NeurIPS 33 (2020)

18. Zhu, J., et al.: Graph neural networks with heterophily. In: AAAI (2021)

19. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. In: ICLR (2017). http://arxiv.org/abs/1609.02907

20. Wu, F., et al.: Simplifying graph convolutional networks. In: ICML (2019)

21. Xu, K., et al.: How powerful are graph neural networks? In: International Conference on Learning Representation (ICLR) (2019). http://arxiv.org/abs/1810.00826

22. Hamilton, W., Ying, Z., Leskovec, J.: Inductive representation learning on large graphs. In: NIPS, pp. 1024–1034 (2017). http://papers.nips.cc/paper/6703-inductive-representation-learning-on-large-graphs.pdf

23. Chen, J., Ma, T., FastGCN, X.C.: Fast learning with graph convolutional networks via importance sampling. In: ICLR (2018)

24. Velickovic, P., et al.: Graph Attention Networks (2018)

25. Zhang, J., et al.: Gated attention networks for learning on large and spatiotemporal graphs. In: UAI (2018)

26. Defferrard, M., Bresson, X., Vandergheynst, P.: Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering (2016). http://arxiv.org/abs/1606.09375

27. Chen, M., et al.: Simple and Deep Graph Convolutional Networks (2020)

28. Chien, E., et al.: Adaptive universal generalized PageRank graph neural network. In: ICLR (2021)

29. Li, G., et al.: Training graph neural networks with 1000 layers. In: ICML (2021)

30. Godwin, J., et al.: Very Deep Graph Neural Networks via Noise Regularisation (2021). http://arxiv.org/abs/2106.07971

31. Klicpera, J., Bojchevski, A., Günnemann, S.: Predict Then Propagate: Combining Neural Networks with Personalized Pagerank for Classification on Graphs (2019)

32. Xu, K., et al.: Representation learning on graphs with jumping knowledge networks. In: PMLR (2018)

33. Wu, Z., et al.: A Comprehensive Survey on Graph Neural Networks (2019). http://arxiv.org/abs/1901.00596

34. Zhou, J., et al.: Graph Neural Networks: A Review of Methods and Applications (2019). http://arxiv.org/abs/1812.08434

35. Tang, J., Alelyani, S., Liu, H.: Feature Selection for Classification: A Review. Data Classification: Algorithms and Applications, pp. 37–64. CRC Press (2014). https://asu.pure.elsevier.com/en/publications/feature-selection-for-classification-a-review,%20publisher

36. Maurya, S.K., Liu, X., Murata, T.: Not all neighbors are friendly: learning to choose hop features to improve node classification. In: International Conference on Information and Knowledgement (CIKM) (2022)

37. Lim, D., et al.: Large scale learning on non-homophilous graphs: new benchmarks and strong simple methods. arXiv (2021). http://arxiv.org/abs/2110.14446

38. Ma, Y., et al.: Is Homophily a Necessity for Graph Neural Networks? (2021). http://arxiv.org/abs/2106.06134

39. Frasca, F., et al.: SIGN: Scalable Inception Graph Neural Networks (2020). http://arxiv.org/abs/2004.11198

40. Li, Y., Chen, C.Y., Wasserman, W.W.: Deep feature selection: theory and application to identify enhancers and promoters. In: Research in Computational Molecular Biology Lecture Notes in Computer Science, pp. 205–217. Springer International Publishing (2015)

41. Wojtas, M., Chen, K.: Feature Importance Ranking for Deep Learning (2020). http://arxiv.org/abs/2010.08973

42. Ross, A., Hughes, M.C., Doshi-Velez, F.: Right for the right reasons: training differentiable models by constraining their explanations. In: IJCAI (2017)

43. Tsang, M., Cheng, D., Liu, Y.: Detecting Statistical Interactions from Neural Network Weights (2018). http://arxiv.org/abs/1705.04977

44. Hechtlinger, Y.: Interpretation of Prediction Models Using the Input Gradient (2016). http://arxiv.org/abs/1611.07634

45. Jin, W., et al.: Node similarity preserving graph convolutional networks. In: WSDM '21, pp. 148–156. Association for Computing Machinery (2021). https://doi.org/10.1145/3437963.3441735

46. Rong, Y., et al.: DropEdge: towards deep graph convolutional networks on node classification. In: ICLR (2020)

47. Wang, Y., Derr, T.: Tree Decomposed Graph Neural Network (2021). http://arxiv.org/abs/2108.11022

48. Maurya, S.K., Liu, X., Murata, T.: Simplifying approach to node classification in graph neural networks. J. Comput. Sci. 62, 101695 (2022). https://www.sciencedirect.com/science/article/pii/S1877750322000990

49. Li, X., et al.: Finding Global Homophily in Graph Neural Networks when Meeting Heterophily (2022). type: article http://arxiv.org/abs/2205.07308

50. Chandrashekar, G., Sahin, F.: A survey on feature selection methods. Comput. Electr. Eng. 40(1), 16–28 (2014). https://www.sciencedirect.com/science/article/pii/S0045790613003066

51. Li, J., et al.: Feature selection: a data perspective. ACM Comput. Surv. 50(6), 94:1–94:45 (2017). https://doi.org/10.1145/3136625

52. Fukumizu, K., Leng, C.: Gradient-based kernel method for feature extraction and variable selection. In: NIPS (2012)

53. Chen, J., et al.: Kernel feature selection via conditional covariance minimization. In: Proceedings of the 31st International Conference on Neural Information Processing Systems NIPS'17, pp. 6949–6958. Curran Associates Inc. (2017)

54. Tibshirani, R.: Regression shrinkage and selection via the Lasso. J. Roy. Stat. Soc. B 58(1), 267–288 (1996). https://onlinelibrary.wiley.com/doi/abs/10.1111/j.2517-6161.1996.tb02080.x

55. Breiman, L.: Random forests. Mach. Learn. 45(1), 5–32 (2001). https://doi.org/10.1023/A:1010933404324

56. Guyon, I., et al.: Gene selection for cancer classification using support vector machines. Mach. Learn. 46(1), 389–422 (2022). https://doi.org/10.1023/A:1012487302797

57. Yamada, M., et al.: High-dimensional feature selection by feature-wise kernelized Lasso. Neural Comput. 26(1), 185–207 (2014). https://doi.org/10.1162/NECO_a_00537

58. Li, Y., Chen, C.Y., Wasserman, W.W.: Deep feature selection: theory and application to identify enhancers and promoters. J. Comput. Biol. 23(5), 322–336 (2016)

59. Sen, P., et al.: Collective Classification in Network Data (2008)

60. Tang, J., et al.: Social influence analysis in large-scale networks. In: KDD '09, pp. 807–816. Association for Computing Machinery (2009). https://doi.org/10.1145/1557019.1557108

61. Rozemberczki, B., Allen, C., Sarkar, R.: Multi-scale Attributed Node Embedding (2020). http://arxiv.org/abs/1909.13021

62. Zhang, Z., Cui, P., Zhu, W.: Deep learning on graphs: a survey. In: Conference Name: IEEE Transactions on Knowledge and Data Engineering, p. 1 (2022)

63. Akiba, T., et al.: Optuna: a next-generation hyperparameter optimization framework. In: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining KDD '19,

pp. 2623–2631. Association for Computing Machinery (2019). https://doi.org/10.1145/3292500.3330701

64. Bergstra, J., et al.: Algorithms for hyper-parameter optimization. In: Advances in Neural Information Processing Systems, Vol. 24. Curran Associates, Inc. (2011). https://papers.nips.cc/paper/2011/hash/86e8f7ab32cfd12577bc2619bc635690-Abstract.html

APPENDIX A

Experiment setup

A.1 │ Hardware (main experiments)
Experiments related feature selection algorithms (RFE, HSIC LASSO and CCM) were conducted on machine with Intel Xeon Gold 6148 processor (limited to 8 cores) and 800 GB memory. Training on feature selection algorithm, DFS and other GNN related experiments were conducted on machine with Intel Xeon Gold 6148 processor (limited to 5 cores), 60 GB memory and V100 GPU.

A.2 │ Model hyperparameters
For Dual-net model, the number of hops for feature aggregation, K is set to 3 and p is set as 4. Hidden dimension of both selector and classifier are set to 64. Tables A1 and A2 lists hyperparameters related to classifier and selector model respectively. WD represents weight decay, LR represents learning rate, $fc1$ & $fc2$ denote layers and $E1$ & $E2$ are the number of epochs of training in Stage 1 and Stage 2 respectively. Patience for Stage 3 training is set to 100. For the classifier network, learning rate was selected from {0.001, 0.01, 0.05}, weight decay was selected from {0.0, 1e–05, 1e–04, 1e–03} and dropout from {0.5, 0.6, 0.7}. For the selector network, learning rate was selected from {0.01, 0.005}, weight decay was

selected from {1e–06, 1e–05, 1e–04} and dropout is kept at constant value of 0.5. Training epochs $E1$ for Stage 1, were selected from 100, 200, 300, 400, 500. Search was done with mix of parameter sweep with manual intervention to avoid unfruitful hyperparameter combinations. Table A3 shows indices of feature matrices learnt for all 10 splits.

A.3 │ Details of experiments on large datasets
Code for experiments on large datasets is available at https://github.com/sunilkmaurya/DualNetGNN_large.

T A B L E A2 Hyperparameters of the selector

Datasets	WD_{both}	LR_{both}	Dropout
Cora	1e–05	0.005	0.5
Citeseer	1e–05	0.005	0.5
Pubmed	1e–05	0.01	0.5
Chameleon	1e–06	0.01	0.5
Wisconsin	1e–04	0.01	0.5
Texas	1e–04	0.005	0.5
Cornell	1e–04	0.005	0.5
Squirrel	1e–06	0.01	0.5
Actor	1e–04	0.01	0.5

T A B L E A3 Statistics of feature matrices selected in 10 random splits for each dataset

Datasets	X	AX	$(A+I)X$	A^2X	$(A+I)^2X$	A^3X	$(A+I)^3X$
Cora	6	0	2	4	3	6	10
Citeseer	10	3	5	4	8	2	6
Pubmed	10	0	1	2	6	0	3
Chameleon	0	10	4	9	1	1	1
Wisconsin	10	1	1	2	1	0	1
Texas	10	5	1	1	0	1	1
Cornell	10	2	0	1	2	2	1
Squirrel	0	10	1	3	0	2	1
Actor	10	0	0	0	0	0	0

T A B L E A4 Statistics for large graph datasets

Dataset	# Nodes	# Edges	# Feat.	# Classes
Penn94	41,554	1,362,229	5	2
pokec	1,632,803	30,622,564	65	2
arXiv-year	169,343	1,166,243	128	5
snap-patents	2,923,922	13,975,788	269	5
genius	421,961	984,979	12	2
twitch-gamers	168,114	6,797,557	7	2

T A B L E A1 Hyperparameters of the classifier

Datasets	WD_{fc1}	WD_{fc2}	LR_{fc}	Dropout	$E1$	$E2$
Cora	1e–04	1e–04	0.01	0.7	400	20
Citeseer	1e–05	1e–03	0.05	0.6	500	20
Pubmed	1e–05	1e–03	0.05	0.6	400	20
Chameleon	0.0	0.0	0.001	0.6	600	20
Wisconsin	1e–04	1e–04	0.05	0.5	100	20
Texas	1e–04	1e–04	0.05	0.7	600	20
Cornell	1e–04	0.0	0.05	0.5	600	20
Squirrel	1e–04	0.0	0.001	0.5	600	20
Actor	1e–03	1e–05	0.05	0.7	600	20

A.3.1 | Data-Splits

For a fair comparison, we use the same data splits as provided by [37] at their code repository (https://github.com/CUAI/Non-Homophily-Large-Scale). Each dataset has five training/validation/test splits and we report mean accuracy/ROC AUC scores in Table 4.

A.3.2 | Feature Calculation

We pre-calculate feature matrices with aggregation of up to 4-hops. In addition, we also include adjacency matrices as feature inputs. Due to the higher complexity of datasets, we use a 3-layered MLP model for the *Classifier*. Results for Penn94, pokec, genius, and twitch-gamers datasets are for aggregation up to 4 hops, while the results for arXiv-year and snap-patents are for up to 3 hops. In the LINKX paper [37] that proposed these benchmark datasets, both arXiv-year and snap-patents are described as directed graph datasets. Hence,

we have used the asymmetric adjacency matrix and its transpose to calculate aggregated features for the nodes. Please note that due to the large size of these datasets and longer training time, analysis similar to Table 1 is computationally prohibitive.

A.3.3 | Hardware

Experiments were conducted on machine with Intel Xeon Platinum 8360Y processor, and A100 GPU.

A.3.4 | Parameter Optimisation

To find the best hyperparameters for the model, we use Optuna [63], a hyperparameter optimisation framework. For sampling hyperparameters during the training, we used multivariate TPESampler (tree-structured Parzen Estimator) algorithm [64] available in Optuna.