Graph Neural Networks for Homogeneous and Heterogeneous Graphs: Algorithms and Applications

A thesis submitted to Indian Statistical Institute

in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Computer Science

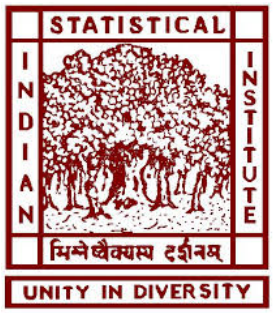
by

Sucheta Dawn

Senior Research Fellow

Under the supervision of

Prof. Sanghamitra Bandyopadhyay



Machine Intelligence Unit

Indian Statistical Institute, Kolkata

July, 2024

To my family,
who means the world to me

ACKNOWLEDGEMENTS

PhD is a journey of having Patience, doing Hard work, and being Determined. In this long journey, there were several ups and downs. It would not be possible to reach the completion of this thesis without those people around me who have always motivated me in every way possible. When I look back on this journey, I realize how fortunate I am. Now, the time has arrived when I should make little effort to express gratitude towards them.

Before all else, I am grateful to the universe for always providing me with the right amount of positivity, strength, and determination to get to where I am today.

It is a pleasure for me to express my heartfelt gratitude to my thesis supervisor, Prof. Sanghamitra Bandyopadhyay, for her endless guidance, encouragement, and support. She will always remain an inspiration for me. I still remember the day when I was completely clueless about my research topic, and she introduced me to my research passion: Graph Neural Networks. Thanks a lot, Madam, for being so supportive and inspiring.

I am also grateful to Dr. Monidipa Das for providing valuable suggestions, which formed the stepping stones for my work in my early PhD days.

I am extremely thankful to the Indian Statistical Institute(ISI) for providing me with the fellowship, which helped me to conduct my research without any financial constraints. I am also grateful to Prof. Sanghamitra Bandyopadhyay, the Director of ISI, for providing an excellent research facility and environment in the institute. I am grateful to the faculties of ISI, especially of the Machine Intelligence Unit, for their valuable suggestions for the betterment of my work.

I sincerely thank my seniors, Dr. Monalisa Pal, Dr. Snehalika Lall, and Suparna Saha, who always guided me like big sisters. I thank all of my labmates, Sayan Saha, Mrittika Chakraborty, and Aishik Chanda, for having fruitful discussions, which have enriched

my knowledge.

I don't have enough words to express gratitude to my friends and colleagues for always being there for me. I extend my earnest gratitude to Sukumar Mandal and Nivedita Chatterjee, who have never stopped believing in me since my college days, even though I doubted my ability. Special thanks to Sukriti Roy, Sankar Mondal, Abhinav Chakraborty, Bibhuti Das, and Rathindra Nath Dutta for making this journey smooth and cherishable.

Finally, words are inadequate to express my gratitude to my family. I am thankful to my mother, Mrs. Sukriti Daw, for her constant support and encouragement during my PhD career and throughout my life. I thank my father, Mr. Deb Ranjan Daw, for teaching me to work with love and passion. Thanks to my sister Soumita Dawn for her unconditional love and support. My heartfelt gratitude to my husband, Dr. Subhadip Pramanik, for his assistance and presence throughout this journey. I would not have enrolled for a PhD if he was not there. Last but not least, my son, Siddhartha, is a gift of my life. His smile brightens my day every morning and encourages me to work harder.

Sucheta Dawn

Abstract

A graph is used to represent complex systems where both entities and their interconnections are equally important. Real-life situations, e.g., social networks, biological networks, recommender systems, etc., are better modeled in terms of graphical structures, as the information about individual entities is not enough to understand the whole system. Due to the existence of non-uniformity in graphical data, traditional machine learning algorithms that perform tasks like prediction, classification, etc., can not be applied directly to such data. Graph Neural Networks (GNNs) are robust variants of deep neural network models that are typically designed to learn from such graphical data. GNN involves transforming graph data into Euclidean representations that various machine-learning algorithms can utilize.

In this thesis, two types of graphs have been studied. In the first two contributory chapters, the graphs considered are homogeneous, where all nodes are of the same type. Chapter 2 describes a model called Interval-Valued Graph Neural Network (IV-GNN), which has been developed to handle homogeneous graphs with interval-valued node features. This model relaxes the restriction that the node features should be single-valued. Here, interval-valued features are allowed, and the corresponding GNN model, along with its mathematical analysis, is presented.

Chapter 3 discusses the importance of hierarchical structure learning within a graph. It describes a model called GraMMy, which is designed for hierarchical semantics-driven graph representation learning based on Micro-Macro analysis. It focuses on the graph at different levels of abstraction to allow the flexible flow of information between the higher-order neighborhoods. The task that we aim to perform on the homogeneous graphs in Chapter 2 and 3 is graph classification.

The second part of the thesis deals with heterogeneous graphs. We consider the social recommender system as an area of application. We have modeled the problem of predicting missing rating value for a user to an item as a link prediction task in a heterogeneous graph setting where multiple types of nodes are present in the data. In our third contribution (Chapter 4), the aim is to quantify the usefulness of the ratings given by the user to an item. For this purpose, a metric called Influence Score of a user has been defined and incorporated into a GNN-based recommender system to develop a Social Influence-aware recommendation system, SInGER.

Although SInGER improves the prediction quality, a limitation of the approach is the uniform definition of the Influence Score, irrespective of the data set considered. To overcome this, in the fourth work (Chapter 5), we develop a neural architecture to cap-

ture user trust without explicitly defining it. It provides an effective means of implicitly accounting for trust propagation and composability while performing GNN-based analyses to accomplish the overall task of item rating prediction.

Related Publications by the Author

Papers in Peer-reviewed Journals

- S. Dawn and S. Bandyopadhyay. IV-GNN: interval valued data handling using graph neural network. Applied Intelligence, $53(5) : 5697–5713, 2023$.
  DOI- https://doi.org/10.1007/s10489-022-03780-1.

- S. Dawn, M. Das, and S. Bandyopadhyay. GraMMy: Graph representation learning based on micro–macro analysis. Neurocomputing, $506 : 84–95, 2022$.
  DOI- https://doi.org/10.1016/j.neucom.2022.07.013.

- S. Dawn, M. Das, and S. Bandyopadhyay. SoURA: a user-reliability-aware social recommendation system based on graph neural network. Neural Computing and Applications, pages $1–19, 2023$.
  DOI- https://doi.org/10.1007/s00521-023-08679-7.

Papers in Peer-reviewed Book Chapters

- S. Dawn, M. Das, and S. Bandyopadhyay. Graph representation learning for protein classification. In Artificial Intelligence Technologies for Computational Biology, pages $1–28$. CRC Press.
  DOI- https://doi.org/10.1201/9781003246688.

Papers in Peer-reviewed Proceedings of International Conferences

- S. Dawn, M. Das, and S. Bandyopadhyay. Singer: A recommendation system based on social-influence-aware graph embedding approach. In 2021 IEEE 18th India Council International Conference (INDICON), pages $1–6$. IEEE, 2021.
  DOI- https://doi.org/10.1109/indicon52576.2021.9691733.

- S. Dawn, M. Das, and S. Bandyopadhyay. Caterer: A graph neural network-based model for category-wise reliability-aware recommendation. In 9th International Conference on Pattern Recognition and Machine Intelligence (PReMI'21), 2021. (In Print).

Contents

List of Figures

List of Tables

List of Algorithms

List of Abbreviations

Abbreviations:

GNN	Graph Neural Network
CNN	Convolutional Neural Network
RNN	Recurrent Neural Networks
DNN	Deep Neural Network
DFS	Depth-First Search
BFS	Breadth-First Search
SCC	Strongly Connected Components
WL	Weisfeiler-Lehman
GI	Graph Isomorphism
MST	Minimum-Spanning Tree
GRL	Graph Representation Learning
LSTM	Long Short Term Memory
MLP	Multi Layer Perceptron
LSH	Locality Sensitive Hashing
MAE	Mean Absolute Error
RMSE	Root Mean Square Error

Chapter 1

Introduction and Scope of the Thesis

1.1 Introduction

Much of the real-life data in this data-driven era are high dimensional and complex, and do not necessarily follow the Euclidean structure. Often, they are better represented as non-Euclidean data, such as graphs and manifolds. For instance, in the e-commerce system [7], the interactions between users and products can be represented as graphs. Additionally, bio-active molecules and their bio-activity [8] can be modeled as graph. In citation networks [9], papers can be viewed as nodes of a graph, and the link between different papers via citation can be modeled as edges of that graph. In computer graphics and vision applications, each shape is modeled as a manifold to find similarity and correspondence between shapes [1].

One of the basic differences of this kind of data and Euclidean data is that the direct connection between any two points in a Non-Euclidean space is not necessarily the shortest path between them: on the contrary a straight line in a Euclidean space is actually the shortest distance between the two. Therefore, applying the Artificial Neural Network technique to solve different tasks on the Non-Euclidean domain is not straightforward and has faced several challenges [10] as follows:

- Graph data is non-uniform. Each graph consists of different number of nodes, and each node in a graph has variable number of neighbors. Hence, convolution

type operations are not directly applicable here.

- Application of the existing machine learning algorithms is not always possible because instances in non-Euclidean space have interdependence. For example, in graph, the nodes share connections with other nodes via edges. So, to extract maximum information about the data while performing any specified task, it is necessary to capture this inter dependency among the nodes.

Graph Neural Networks (GNNs) play a crucial role in filling this gap between Graph data and existing Artificial Neural Network techniques, overcoming the aforementioned challenges. It finds a low-dimensional representation for nodes in a large graph, which captures the structural information as well as feature-related information of the nodes. Therefore, the whole graph data will be transferred to a set of low dimensional vectors, which makes several machine learning tasks such as Recognition, Classification, and Prediction easy to perform.

1.2 Graph - The Power of Connectivity

A graph is a data structure representing a network of entities and their relationships. In this section, some formal definitions pertaining to graphs are provided.

1.2.1 Definitions

Definition 1. *A graph $G = (V, E)$ consists of a set of vertices V and the set of edges $E \subseteq V \times V$. A node $v_i \in V$ is related to another node $v_j \in V$ if $e_{ij} = (v_i, v_j) \in E$. A graph can be directed or undirected based on the relationship defined among the nodes of the edges. The directed edges are usually denoted by an arrow showing the direction.*

Definition 2. ***Neighborhood of a node*** *v_i in a graph is defined as the set of nodes which are connected by edges to the node v_i, i.e.,*

$$\mathcal{N}(v_i) = \{v_j \in V \mid e_{ij} = (v_i, v_j) \in E\}. \tag{1.2.1}$$

Definition 3. ***Degree of a node*** $d(v_i)$ *is the number of nodes in the neighborhood of the node* v_i*, i.e.,*

$$d(v_i) =\mid \mathcal{N}(v_i) \mid, \tag{1.2.2}$$

where $|.|$ *denotes the cardinality of a set.*

Definition 4. ***Adjacency Matrix*** A *for a graph* $G = (V, E)$ *with* N *nodes, i.e.,* $\mid V \mid= N$ *is a* $N \times N$ *matrix, where*

$$\begin{aligned} A_{ij} &= w_{ij}, \textit{ if } e_{ij} \in E, \\ &= 0, \textit{Otherwise,} \end{aligned} \tag{1.2.3}$$

where w_{ij} *is the weightage of the edge* e_{ij}*. For unweighted graphs, all edges are given the same weightage, i.e.,* $w_{ij} = 1, \forall e_{ij} \in E$*.*

1.2.2 Types of Graphs

Graphs are useful for understanding and modeling various real-world systems and relationships. Graphs can be usually classified into two main types based on the nature of their vertices: homogeneous graphs and heterogeneous graphs.

Definition 5. ***Homogeneous Graph.*** *Formally, a graph* $G = (V, E)$ *is considered homogeneous if all vertices in the vertex set* V *are of the same type.*

Definition 6. ***Heterogeneous Graph.*** *Formally, a graph* $G = (V, E)$ *is considered heterogeneous if vertices in the vertex set* V *are of different types.*
In a heterogeneous graph, the neighborhood of a node is the set of nodes that are directly connected to it, taking into account the different types of nodes. Similarly, a node have different degrees based on the different nodes types, which is known as Type-specific degree of a node in literature. Also, a heterogeneous graph has a Multi-dimensional adjacency matrix where each entry is a vector or a tensor that represents the connections between nodes of different types.

1.2.3 Common Graph Theoretic Problems and their Solutions

Various tasks can be performed on graphs, each with its own algorithms and applications. We divide these into two groups: (i) algorithms that use traditional graph theoretic approaches, and (ii) those that use machine learning approaches. A detailed overview of some common graph-related tasks is presented in the following subsections.

1.2.3.1 Graph Problems and Traditional Algorithms

Some of the common tasks in graphs and graph-theoretic approaches for solving them are mentioned below.

- Graph Search and Traversal:

    a. Depth-First Search (DFS) and Breadth-First Search (BFS): These classical algorithms are used to explore or traverse graphs, find paths, cycles, or connected components.

    b. Dijkstra's Algorithm [11]: This algorithm is used to find the shortest paths between nodes in a weighted graph.

    c. Bellman-Ford Algorithm [12]: This algorithm overcomes the limitation of Dijkstra's algorithm by finding the shortest paths in a graph with negative edge weights.

- Graph Connectivity and Components:

    a. Connected Components: A connected component of an undirected graph is a subgraph in which each pair of nodes is connected with each other via a path. Algorithms such as Depth-First search find connected components in undirected graphs.

    b. Strongly Connected Components (SCC): A strongly connected component is a subgraph of a directed graph where each pair of nodes is connected with each other via a route which is a path with directed edges. Tarjan's

algorithm and Kosaraju's algorithm are used to find SCC in directed graphs [13].

- Graph Matching and Isomorphism:

  a. Graph Isomorphism Testing: The graph isomorphism problem, the problem of determining whether two graphs are isomorphic or not, is not only a fundamental problem of graph theory but also a very practical problem in studying and enumerating combinatorial structures in real-life scenarios. Graph Isomorphism (GI) became significant in the theory community during the 1970s. It stood out as one of the rare natural problems in the complexity class NP that could neither be categorized as NP-complete nor solved by a polynomial time algorithm [14]. Traditionally algorithms such as the Weisfeiler-Lehman (WL) algorithm [15] or the Nauty algorithm [16] are used to determine whether two graphs are isomorphic or not. In this thesis, our works are inspired by the WL algorithm to solve the GI problem, and hence, this algorithm has been discussed in detail in a later section.

  b. Subgraph Matching: Here the job is to find occurrences of a pattern graph within a larger graph. Techniques like subgraph isomorphism or graph matching are used to solve this problem.

- Network Flow and Optimization:

  a. Max-Flow Min-Cut Theorem: The max-flow min-cut theorem is a fundamental result in network flow theory, which connects two key concepts: the maximum flow through a network and the minimum cut of the network. Algorithms such as Ford-Fulkerson [17] or Edmonds-Karp [18] are used to find the maximum flow in a network.

  b. Minimum-Spanning Tree (MST): A spanning tree is a tree-like subgraph of a connected, undirected graph that includes all the graph's vertices. The spanning tree that has the minimum possible weight among all possible

spanning trees is called MST. Finding an MST is an important task in network design, social network analysis, etc., to identify important connections and relationships among the entities and reduce the connection cost. Algorithms like Kruskal's [19] and Prim's [20] are well known for finding the MSTs of a graph.

c. Graph Colouring: Graph coloring is the task of coloring vertices of a graph with the fewest possible colors such that no two adjacent vertices have the same color. Techniques such as greedy coloring algorithms or backtracking are used to solve this task.

- Network Analysis:

  a. Centrality Measures: Algorithms like degree centrality, betweenness centrality, and closeness centrality are used to identify important nodes in social networks.

  b. Community Detection: This is an important problem in graph theory, where the objective is to find groups or clusters of nodes that are more connected and/or similar to each other than nodes from different groups or clusters. Techniques such as modularity optimization or hierarchical clustering are used to find communities or clusters within social networks.

1.2.3.2 Graph Algorithms using Machine Learning

Graph-related tasks have expanded significantly in the machine learning era, leveraging advancements in graph theory, machine learning, and deep learning. Following are some key graph-related tasks and applications:

- Graph Representation Learnings (GRLs): Graph Representation Learning is a field where machine learning and graphs come together. It focuses on how to learn and encode graphs into low-dimensional vector spaces. These vectors capture the structural and semantic properties of the graph, making it easier to apply

machine learning algorithms to graph data. The different levels of representation learning on graphs are as follows,

a. Node Embedding: Learning low-dimensional vector representations of nodes that preserve graph topology and node attributes.

b. Graph Embedding: Learning representations for entire graphs to capture structural information that can be useful for graph classification tasks.

- Graph Neural Networks (GNNs): GNNs are a subset of GRL techniques that specifically use neural network-based methods to learn node, edge, or graph embeddings. Here are different tasks that can be performed on graphs using GNNs.

  a. Node Classification: It is the task of predicting the labels of nodes based on graph structure and node features[21].

  b. Link Prediction: Link prediction is a useful task in recommendation systems and social network analysis. Here, the job is to predict the existence of edges between pairs of nodes[22].

  c. Graph Classification: In this problem, one has to classify entire graphs into categories. This has important applications in bioinformatics (e.g., protein function prediction) and chemistry (e.g., molecular property prediction)[23].

  As this thesis mainly focuses on GNNs, and their use for graph classification and link prediction, these will be discussed formally in a later part of this chapter.

- Graph-based Semi-Supervised Learning: Graph-based Semi-Supervised Learning focuses on using graph structure to propagate labels from a small set of labeled nodes to a larger set of unlabeled nodes. The goal here is to improve the classification performance with limited labeled data [24].

- Graph Generative Models:

a. Graph Generation: Graph generative models such as GraphRNN [25], GraphVAE [26] creates new graphs with similar properties to a given set of graphs and these have been found useful in drug discovery and material science.

b. Graph Completion: Graph Completion is done to fill in the missing parts of a graph, such as predicting missing nodes or edges.

- Graph-based Clustering for Community Detection: Identifying clusters or communities within a graph has been useful in several domains, especially for social network analysis and for detecting functional modules in biological networks.

- Explainability in Graph Models: The job here is to improve trust and transparency in AI applications by developing methods to interpret the decisions made by GNNs.

This discussion above shows that the integration of graph theory with machine learning has opened up a multitude of tasks and applications that leverage the rich structural information contained in graphs. As the field evolves, further advancements are expected in graph algorithms, models, and real-world applications driven by continuous innovations in machine learning and deep learning.

1.3 Graph Neural Networks: From Nodes to Knowledge

GNNs are robust variants of deep network models, typically designed to learn from graph. It is an emerging branch of machine learning in non-Euclidean space, especially performing well in different tasks where graph-structured data is involved [27]. GNN approaches can extract structural and feature-related information from a graph and find Euclidean embeddings for non-Euclidean data, which help to perform several tasks like classification, regression, link prediction, etc., on graphs [28]. In this thesis, we focus on graph classification and link prediction tasks on graphs using GNN-based approaches.

Let $G = (V, E)$ be a graph, and x_v be the node feature vector associated with a node $v \in V$.

- **Graph Classification** Let $\{G_1, ..., G_N\}$ be a collection of graphs, where $\{y_1, ..., y_N\}$ is the set of associated labels of the graphs. The aim of Graph Classification task is to learn an association between the graphs and the labels. Towards this, the objective of a GNN-based approach is to learn a Euclidean representation z_G for a particular graph G such that y_G is a function of z_G, i.e., $y_G = h(z_G)$.

- **Link Prediction** Given a pair of nodes $(u, v) \in V \times V$, the goal is to predict the likelihood $p(u, v)$ of the existence of an edge between them. The objective of a GNN-based approach is to get vector representations z_u and z_v for u and v, respectively, such that $p(u, v)$ is a function of z_u and z_v, i.e., $p(u, v) = g(z_u, z_v)$.

GNN model approaches can be classified into two categories, namely spectral and spatial methods [10]. Specifically, early GNN models were implemented using spectral theory. In this section, first the basic steps of spectral GNNs, some examples, and their drawbacks are discussed. This is followed by a detailed discussion on spatial GNNs, which is the main topic of this thesis.

1.3.1 Spectral Graph Neural Networks [1]

Spectral GNNs are a type of neural networks designed to operate on graph-structured data that rely on spectral decomposition and convolutional filters to capture graph structure and node relationships.

Spectral GNNs use the following steps:

a. Spectral decomposition: Decompose the graph's Laplacian or adjacency matrix into eigenvalues and eigenvectors.

b. Filtering: Apply filters to the eigenvalues and eigenvectors to extract relevant information.

c. Node embedding: Compute node representations by combining the filtered eigenvalues and eigenvectors.

d. Learning based on neural networks: Feed the node embeddings into a neural network to predict graph properties or node labels.

Some spectral convolution approaches are SpectralCNN [29], SyncSpecCNN [30], SSF-CNN [31], ChebNet [8] and Motifnet [32].

1.3.1.1 Drawbacks of Spectral Approaches

Some of the drawbacks of spectral approaches are listed below.

a. Computational cost: Spectral decomposition can be computationally expensive for large graphs.

b. Eigenvalue sensitivity: Small changes in eigenvalues can significantly affect node embeddings.

c. Over-smoothing: Spectral GNNs suffer from over-smoothing, losing node-specific information.

d. Lack of spatial awareness: Spectral GNNs may not capture local spatial information, relying solely on spectral features.

e. Limited interpretability: While the spectral domain provides some insights, node embeddings can be difficult to interpret.

In order to address the aforementioned limitations, subsequent developments have taken place is, Spatial GNNs. Before introducing the notion of a spatial GNNs, the intuition behind its development is first elaborated. For this purpose, it is necessary to first introduce the Weisfeiler-Lehman (WL) test.

1.3.1.2 The Weisfeiler-Lehman Isomorphism Test: An Isomorphism Check for Graphs

The Weisfeiler-Lehman(WL) test is a procedure introduced in 1968 by Albert Weisfeiler and Roman L. Lehman to determine whether two graphs are isomorphic (i.e., structurally identical) or not [15].

The main idea of the WL algorithm is to decompose the graph into substructures. It extracts node-level features using the iterative neighborhood aggregation approach, which enables the model to gather neighborhood information and accumulate this rich information into a graph-level representation. To test whether two given graphs are isomorphic or not, the key idea of the algorithm is to iteratively assign a new color to a node from the sorted alphabet of colors, based on the node's current color and the multi-set of colors of neighboring nodes. Refer Fig. 1.1 for an illustration. Here the task is to check whether the two graphs, G_1 and G_2 are isomorphic or not. As shown in the figure, initially (in 0-th iteration) both the graphs G_1 and G_2 have the same frequency distribution of node labels, which is nothing but the degrees of the nodes. But in the next iteration ($i = 1$), the labels of the neighbors of a node get concatenated with its own; the two graphs end up having different representations. Hence it can be concluded that G_1 and G_2 are non-isomorphic, and the algorithm terminates. The steps of the i-th iteration of the WL test are summarized in the following algorithm.

Algorithm:

i. The nodes are initiated with the node label. If node label is not given, node labeling function $f^{(0)}$ is used to assign label to each node. Possible node labeling functions are assigning degree, centralities or clustering coefficient of the node. In Fig. 1.1, Step-1, the node degree is used as the initial label of the node.

ii. The multiset of the node label is sorted in a defined order. The feature vector representation for the graph is the frequency distribution of the node label.

iii. For iteration $i > 0$, the node labelling function $f^{(i)}$ will take the target node v's

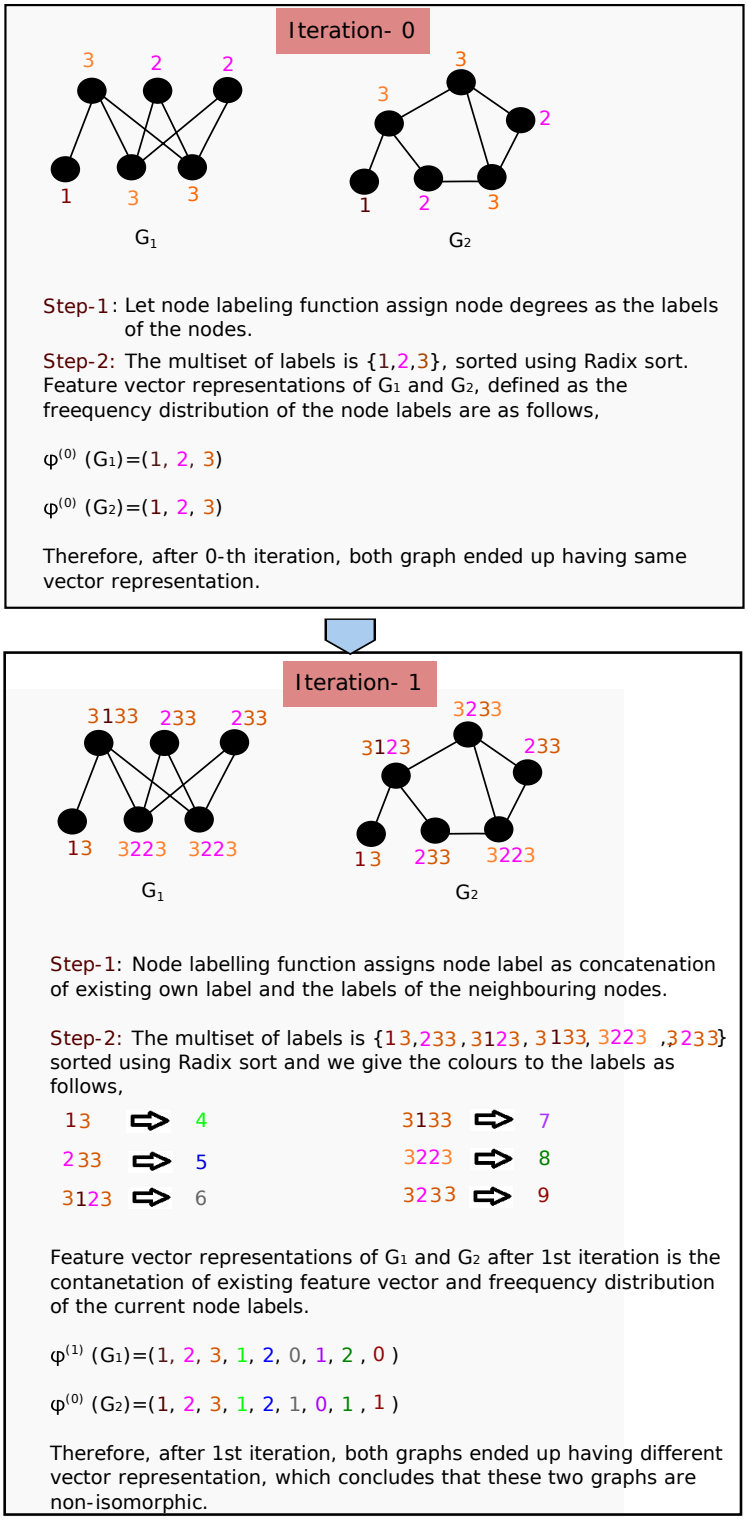


Figure 1.1: Weisfeiler-Lehman test to check whether G_1 and G_2 are isomorphic or not [6].

node label $l^{(i-1)}(v)$ and the multiset$\{l^{(i-1)}(u) : u \in \mathcal{N}(v)\}$ and will output a unique label $l^{(i)}(v)$ for the node v, i.e.,

$$l^{(i)}(v) = f^{(i)}(l^{(i-1)}(v), \{l^{(i-1)}(u) : u \in \mathcal{N}(v)\}). \tag{1.3.1}$$

Here, $\mathcal{N}(v)$ is the set of nodes directly connected to v. Step-2 in Fig. 1.1 describes this step.

iv. Repeat Steps $(ii) - (iii)$ until i reaches the pre-defined iteration bound or the multisets for the graphs end up being different.

After the execution of the WL algorithm, if two multisets of node labels for two graphs are different, it can be concluded that the two graphs are non-isomorphic. However, if two multisets are the same, nothing can be concluded about the two graphs being isomorphic. The WL test of graph isomorphism works effectively on certain classes of graphs but has limitations on others (See Fig. 1.2 for an example where WL test fails). A detailed discussion of where the WL test works well and where it does not is provided below.

Graph Classes where WL Test Succeeds

- Trees: The WL test is very effective on trees and can distinguish non-isomorphic trees because the tree structure allows the test to propagate and refine node colors effectively.
- Planar Graphs: The WL test distinguishes planar graphs because the planar structure often provides sufficient variation in local neighborhoods to uniquely refine colors.
- Graphs with Unique Node Degrees: If all nodes in a graph have unique degrees, the WL test can easily distinguish between non-isomorphic graphs, as the initial coloring (based on degree) will be unique.

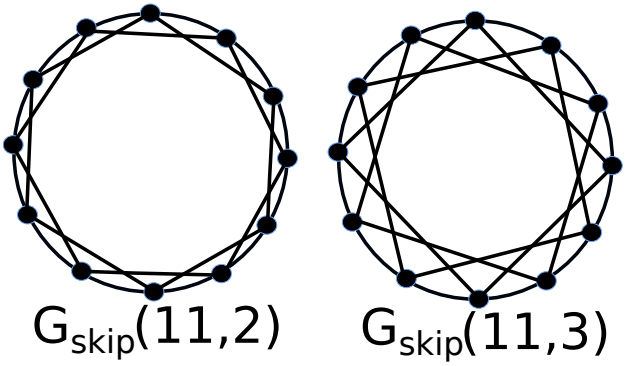


Figure 1.2: Examples of two non-isomorphic graphs that WL test fails to distinguish

- Graphs with a High Degree of Symmetry Breaking: If the graph has a structure that allows the color refinement process to quickly break symmetries, the WL test can often distinguish between non-isomorphic graphs.

Graph Classes where WL Test Fails

- Regular Graphs: Graphs where all nodes have the same degree (k-regular graphs) often cause the WL test to fail because the initial coloring based on degree does not provide any distinction, and further refinement may not break the symmetry.
- Cographs: Cographs (complement-reducible graphs) are another class where the WL test fails because they have highly symmetric structures that the WL test cannot distinguish.
- Strongly Regular Graphs: Strongly regular graphs have very symmetric structures and specific properties that make them indistinguishable by the WL test. These graphs have the same number of nodes, the same degree for each node, and the same number of common neighbors between pairs of adjacent and non-adjacent nodes.
- Certain Highly Symmetric Graphs: Any graph with a high degree of symmetry,

where nodes have similar local structures, can defeat the WL test. Examples include the Petersen graph [33] and other symmetric graphs where local neighborhoods look identical under the refinement process.

Fig. 1.2 gives an example of two non-isomorphic graphs where the WL test can not conclude that they are non-isomorphic. Note that, in both graphs, each node is connected to four nodes of degree 4. Wl test could not distinguish between them as the degrees of each node in both the graphs are the same.

Spatial GNNs follow a similar technique when finding the representation of a node where, at each iteration, a node's representation is modified based on its current representation and its surroundings' representation. The next subsection details the Spatial GNN and various developments in GNN literature.

1.3.2 Spatial Graph Neural Networks

In spatial GNNs, information is passed to each node as a message from all of its neighbors in an iterative manner. This approach is usually referred to as the neural message-passing technique, which allows nodes to exchange information with their neighbors, thus enabling the GNN to capture node relationships and graph structure. The messages received at a node from its neighbors are aggregated and further combined with the information at that node. In this way, the information at every node is updated, after which the updated information is again propagated to the neighbors following the same neural message-passing process. Both the aggregate and the combine operators involve parameters that are learned through various approaches.

1.3.2.1 Embedding Generation Approach of a Spatial GNN

Let us consider an example of a graph as shown in Fig. 1.3, a typical input to a GNN for embedding generation. The graph contains 9 nodes of two types, which are denoted by Green and Yellow colors. Fig. 1.4 depicts the representation learning for node "A" in Fig. 1.3 following the Neural Message Passing Technique as mentioned earlier. We consider the 2- hop neighborhood of "A" and represent it as the subtree structure rooted

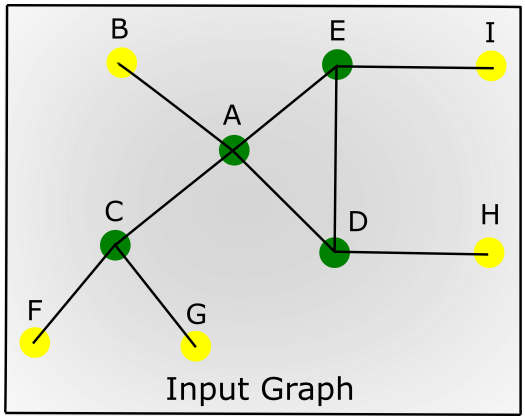


Figure 1.3: Example of a graph that is to be used as input graph [6].

at "A" (Fig. 1.3). At each layer, as it can be seen that the "Blue" rectangle denotes the AGGREGATE function, which collects information from the immediate neighbors of a node and generates a message. The "Purple" triangle denotes the COMBINE function, which accepts the message from the neighborhood and the node itself to generate a new representation for the node. The right-hand side of Fig. 1.4 depicts how embeddings are updated (denoted by new colors) of the nodes.

More formally, the k-th iteration of the neural message passing technique can be expressed in two steps,

$$\mathcal{M}^k_{\mathcal{N}(v)} = AGGREGATE^k(\{h^{k-1}_u : u \in \mathcal{N}(v)\}), \tag{1.3.2}$$

and

$$h^k_v = COMBINE^k(h^{k-1}_v, \mathcal{M}^{(i)}_{\mathcal{N}(v)}). \tag{1.3.3}$$

$AGGREGATE^k$ and $COMBINE^k$ are two trainable neural networks. We summarize the forward propagation algorithm of a GNN model, which is used to generate the embeddings of nodes using the graph structure and node features in Algorithm 1. The node embeddings are initialized with node features in step 1. Then for each node v, $AGGREGATE^k$ accumulates the feature information about the neighboring nodes

($\mathcal{N}(v)$) that has been gathered so far (i.e. upto $k-1$-th iteration) and generates the message $\mathcal{M}^k_{\mathcal{N}(v)}$ for v in k-th iteration (as shown in Step 4). In Step 5, the job of $COMBINE^k$ function is to accept the message $\mathcal{M}^k_{\mathcal{N}(v)}$ and, based on the received message, update the existing feature vector of v (See Fig. 1.4). Therefore, building a powerful GNN requires finding powerful $AGGREGATE$ and $COMBINE$ functions for the model.

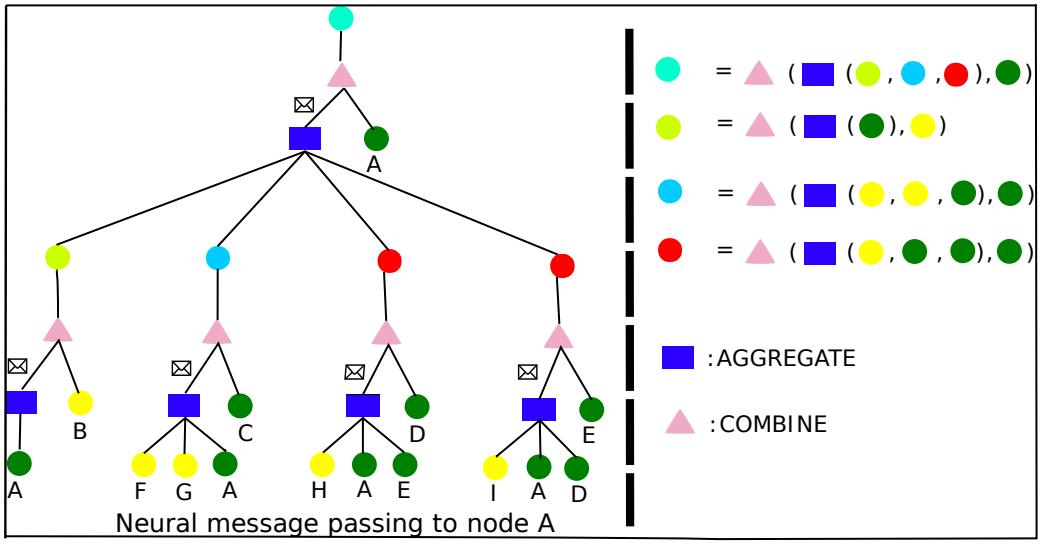


Figure 1.4: Representation learning using Neural Message Passing Technique [6].

As this thesis primarily aims to perform graph classification, there is a need of another function named READOUT, which will accept the representation of every node after the final iteration K, and will predict the graph label as follows:

$$z_G = READOUT(\{z_v | v \in V\}).$$

1.3.2.2 Some Widely Used GNN Models

Some widely used GNN models are discussed in this section.

Algorithm 1 GNN framework for embedding generation

Input : Graph $G(V, E)$; input features $\{x_v, \forall v \in V\}$; depth K;
aggregator functions $AGGREGATE^k, \forall k \in \{1, \cdots, K\}$;
combine functions $COMBINE^k, \forall k \in \{1, \cdots, K\}$
Output : Vector representations $z_v, \forall v \in V$

$h_v^0 \leftarrow x_v, \forall v \in V$
for $k = 1, \cdots, K$ **do**
  for $v \in V$ **do**
    $\mathcal{M}_{\mathcal{N}(v)}^k = AGGREGATE^k(\{h_u^{k-1} : u \in \mathcal{N}(v)\})$
    $h_v^k = COMBINE^k(h_v^{k-1}, \mathcal{M}_{\mathcal{N}(v)}^k)$
  end for
end for
$z_v \leftarrow h_v^K; \forall v \in V$

1. **Graph Convolutional Network (GCN)** [24]: GCN is a type of neural network architecture designed for processing and analyzing graph-structured data. GCNs leverage convolutional operations to capture and aggregate information from neighboring nodes in a graph, enabling them to learn meaningful representations of nodes that account for the relational structure within the graph. The following are the key mathematical equations for GCN.

   (a) Graph Convolutional Layer:

   Let $G = (V, E)$ be a graph with nodes V and edges E. Let $X \in \mathbb{R}^{N \times F}$ be the node feature matrix, where $N = |V|$ and F is the number of features. Let $A \in \mathbb{R}^{N \times N}$ be the adjacency matrix of the graph.

   The graph convolutional layer computes the following:

$$H = \sigma(\hat{A}XW). \tag{1.3.4}$$

   where:

   - $\hat{A} = D^{-1/2}AD^{-1/2}$ is the normalized adjacency matrix.

   - D is the degree matrix (diagonal matrix with node degrees).

   - $W \in \mathbb{R}^{F \times F'}$ is the weight matrix and F' is the dimension of the embedding space.

- σ is an activation function (e.g., ReLU [34]).

- $H \in \mathbb{R}^{N \times F'}$ is the output feature matrix.

(b) Node Representation Update:

The node representation is updated as follows,

$$H_i = \sigma\left(\sum_{j \in \mathcal{N}(i)} \frac{1}{\sqrt{D_{ii} D_{jj}}} A_{ij} X_j W\right), \tag{1.3.5}$$

where:

- H_i is the updated representation of node i.

- $\mathcal{N}(i)$ is the neighborhood of node i.

- D_{ii} and D_{jj} are the degrees of nodes i and j, respectively.

- σ is an activation function (e.g., ReLU [34]).

(c) Multi-Layer GCN:

A multi-layer GCN stacks multiple graph convolutional layers as follows,

$$H^{k+1} = \sigma\left(\hat{A} H^k W^k\right). \tag{1.3.6}$$

where,

- H^k is the output of the previous layer,

- W^k is the weight matrix of the current layer,

- H^K is the final output of the GCN (after K layers),

- σ is an activation function (e.g., ReLU [34]).

Despite their success in multiple applications, GCNs face challenges, such as scalability issues with large graphs and sensitivity to the chosen graph structure. These prompt ongoing research to enhance their efficiency and robustness in diverse applications. The next model GraphSAGE was developed to solve the scalability issue found in the GCN model.

2. **GraphSAGE** [35]: GraphSAGE, or Graph Sample and Aggregated Embed-

dings, is a powerful GNN model designed for learning node embeddings in large-scale graphs. One of the key innovations of GraphSAGE is its ability to generate embeddings for nodes by sampling and aggregating information from their local neighborhoods, allowing it to scale efficiently to massive graphs. GraphSAGE employs a sampling strategy to select a fixed-size neighborhood around each node, capturing both the local and global structural information of the graph. The model employs an aggregation function, such as mean or Long Short Term Memory (LSTM), to combine the embeddings of sampled neighbors, enabling it to generate expressive and context-aware node representations.

(a) Node Representation Update:

The node representation is updated as follows:

$$h_v^{k+1} = k\sigma\left(W \cdot \mathrm{CONCAT}\left(h_v^k, \mathcal{M}_{\mathcal{N}(v)}^k\right)\right), \tag{1.3.7}$$

where,

- h_v^k is the representation of node v at layer k,
- $\mathcal{M}_{\mathcal{N}(v)}^k$ is the representation of the neighborhood of node v at layer k,
- W is the weight matrix,
- σ is an activation function,
- CONCAT concatenates the node representation with the neighborhood representation.

(b) Neighborhood Representation:

The aggregator function computes the neighborhood representation as follows,

$$\mathcal{M}_{\mathcal{N}(v)}^k = AGGREGATE\left(\left\{h_u^k \mid u \in \mathcal{N}(v)\right\}\right), \tag{1.3.8}$$

where,

- $AGGREGATE$ is a pooling function (e.g. mean, sum, max).

(c) Final Representation:

The final representation of a node is computed after K layers as follows,

$$h_v = h_v^K. \tag{1.3.9}$$

(d) Readout Layer:

The readout layer computes the final output as

$$y_G = \sigma\left(W_y \cdot h_v\right), \tag{1.3.10}$$

where,

- W_y is the weight matrix of the readout layer,
- y_G is the final output,
- σ is an activation function (e.g., ReLU [34]).

While GraphSAGE scales well to large graphs, neither GCN nor GraphSAGE can achieve the same expressive power as that of the WL test. This paves the way for more powerful models as discussed below.

3. **Graph Isomorphism Network (GIN)** [36]: The GIN model is a deep learning architecture specifically designed for solving graph isomorphism problems, a fundamental challenge in graph theory and computer science. GIN employs a message-passing neural network framework, enabling it to capture and learn the structural features of graphs by iteratively updating node representations based on their neighborhood information. Unlike traditional graph isomorphism algorithms, GIN utilizes learnable parameters to adaptively process and distinguish different graph structures, making it more versatile and capable of handling a wide range of graph types. The model's success lies in its ability to efficiently determine whether two given graphs are isomorphic, meaning they have the same underlying structure despite potentially different node and edge labels. This model uses one of the powerful pooling techniques SUM for aggregation and a

non-linear function for combination, same as in GCN.

(a) Graph Isomorphism Network (GIN) Layer:

The GIN layer computes the following,

$$h_v^{k+1} = \mathrm{MLP}\left((1+\epsilon)\cdot h_v^k + \sum_{u\in\mathcal{N}(v)} h_u^k\right), \tag{1.3.11}$$

where,

- h_v^k is the representation of node v at layer k,

- $\mathcal{N}(v)$ is the neighborhood of node v,

- ϵ is a learnable parameter,

- MLP is a multi-layer perceptron (neural network).

(b) Node Representation Update:

The node representation is updated as,

$$h_v^{k+1} = \mathrm{UPDATE}\left(h_v^k, \sum_{u\in\mathcal{N}(v)} h_u^k\right), \tag{1.3.12}$$

where,

- UPDATE is a learnable update function.

(c) Final Representation:

The final representation of a node is computed after K layers as

$$h_v = h_v^K. \tag{1.3.13}$$

(d) Readout Layer:

The readout layer computes the final output as

$$y_G = \mathrm{READOUT}\left(\{h_v \mid v \in G\}\right), \tag{1.3.14}$$

where,

- READOUT is a readout function (e.g. sum, mean).

- G is the input graph.

It is to be noted that the fundamental basis for the models we have seen so far is the WL test. GIN achieves equal discriminative power as that of WL test. However, a more powerful test than the WL test is the higher-order WL test. The higher-order WL test is better than the traditional WL test because it can find more complex structural patterns and might be better at telling the difference between graphs that are not isomorphic. The next model, k-GNN, based on the higher-order WL test is discussed below.

4. **k-GNN** [37]: k-GNN explores the integration of higher-order GNNs into the WL graph isomorphism test. It leverages the expressive power of higher-order neural networks to capture more complex graph structures. By extending the WL test to incorporate neural components, the model gains the ability to discern subtle structural differences between graphs, leading to improved performance on various graph-related tasks.

   (a) k-GNN Layer:

   The k-GNN layer computes the following:

$$h_v^{k+1} = \sigma\left(W \cdot \text{AGGREGATE}_{(k)}\left(\left\{h_u^k \mid u \in \mathcal{N}_k(v)\right\}\right)\right), \tag{1.3.15}$$

   where,

   - h_v^k is the representation of node v at layer k,

   - $\mathcal{N}_k(v)$ is the k-hop neighborhood of node v,

   - W is the weight matrix,

   - σ is an activation function (e.g., ReLU [34]),

   - AGGREGATE^k is a k-hop aggregation function (e.g. mean, sum).

   (b) k-Hop Aggregation:

The k-hop aggregation is computed as follows:

$$\text{AGGREGATE}^k\left(\left\{h_u^k \mid u \in \mathcal{N}_k(v)\right\}\right) = \frac{1}{|\mathcal{N}_k(v)|}\sum_{u\in\mathcal{N}_k(v)} h_u^k. \quad (1.3.16)$$

This is a mean aggregation over the representations of the nodes in the k-hop neighborhood.

(c) Node Representation Update:

The node representation is updated as,

$$h_v^{k+1} = \text{UPDATE}\left(h_v^k, \text{AGGREGATE}^k\left(\left\{h_u^k \mid u \in \mathcal{N}_k(v)\right\}\right)\right), \quad (1.3.17)$$

where,

- UPDATE is a learnable update function.

(d) Final Representation:

The final representation of a node is computed after K layers as

$$h_v = h_v^K. \quad (1.3.18)$$

(e) Readout Layer:

The readout layer computes the final output as

$$y_G = \text{READOUT}\left(\{h_v \mid v \in G\}\right), \quad (1.3.19)$$

where,

- READOUT is a readout function (e.g., sum, mean).
- G is the input graph.

k-GNN is capable of handling higher-order graph patterns, allowing the model to capture more intricate relationships and dependencies within graph-structured data. This enhancement proves crucial in scenarios where traditional GNNs may

fall short in capturing nuanced structural information.

5. **Graph U-Net (g-U-Net)** [38]: g-U-Net extends the traditional U-Net architecture to handle irregularly shaped data by incorporating graph convolutional layers, enabling the model to capture dependencies and relationships in graph-structured inputs. The architecture consists of an encoder-decoder structure, with graph convolutional layers used to process information at different hierarchical levels. It leverages the concept of message passing to aggregate information from neighboring nodes, enabling the model to learn complex patterns and representations in graph data. The skip connections in Graph U-Net facilitate the flow of information between corresponding layers of the encoder and decoder, aiding in the preservation of important features. This model adatively selects a small set of nodes to form a smaller subgraph using a pooling layer (such as mean, max) and its inverse operation by another unpooling layer (such as interpolation) to restore the original graph structure.

   (a) Graph Convolutional Layer:

   The graph convolutional layer computes the following,

$$H^{k+1} = \sigma\left(\hat{A}H^{k}W^{k}\right), \tag{1.3.20}$$

   where,

   - H^k is the node representation at layer k,
   - $\hat{A}$ is the normalized adjacency matrix
   - W^k is the weight matrix,
   - σ is an activation function (e.g., ReLU).

   (b) Graph Pooling Layer:

   The graph pooling layer computes the following,

$$H^{k+1} = \mathrm{POOL}\left(H^{k}\right), \tag{1.3.21}$$

where,

- POOL is a pooling function (e.g., mean, max).

(c) Graph Unpooling Layer:

The graph unpooling layer computes the following,

$$H^{k+1} = \text{UNPOOL}\left(H^{k}\right), \tag{1.3.22}$$

where,

- UNPOOL is an unpooling function (e.g., interpolation).

(d) Graph Convolutional Block:

The graph convolutional block computes the following,

$$H^{k+1} = \sigma\left(\hat{A} \cdot \text{ReLU}\left(\hat{A}H^{k}W_{1}^{k}\right)W_{2}^{k}\right), \tag{1.3.23}$$

where,

- W_1^k and W_2^k are weight matrices.

(e) g-U-Net Architecture:

The g-U-Net architecture consists of multiple graph convolutional blocks, followed by graph pooling layers, and then graph unpooling layers:

$$H^{(K)} = \text{UNPOOL}\left(\text{POOL}\left(H^{(K-1)}\right)\right), \tag{1.3.24}$$

where,

- K is the number of layers.

6. **Principal neighborhood Aggregation (PNA)** [39]: PNA is designed to capture higher-order relationships by incorporating the structural information from neighboring nodes in a principled manner, allowing it to model complex graph structures better. One key feature of PNA is its ability to learn adaptive weights for different neighborhood nodes, enabling it to assign varying importance to dif-

ferent parts of the graph during the aggregation process. This adaptive weighting mechanism in PNA makes it particularly effective in handling graphs with non-uniform structures and varying degrees of connectivity. PNA introduces a set of shared parameters across different aggregation layers, promoting parameter efficiency and facilitating the transfer of information across multiple layers of the neural network. The PNA model leverages learnable aggregation functions, enabling it to adapt to diverse graph datasets and capture intricate patterns in the underlying graph topology. By incorporating both global and local information, PNA strikes a balance between capturing overall graph-level features and preserving detailed local node characteristics.

(a) Node Representation:

The node representation is computed as,

$$h_v = \sigma\left(W \cdot x_v\right), \tag{1.3.25}$$

where,

- h_v is the representation of node v,
- x_v is the input feature of node v,
- W is the weight matrix,
- σ is an activation function (e.g., ReLU).

(b) Neighborhood Aggregation:

The neighborhood aggregation is computed as,

$$\mathcal{M}^k_{\mathcal{N}(v)} = \frac{1}{|\mathcal{N}(v)|} \sum_{u \in \mathcal{N}(v)} h_u, \tag{1.3.26}$$

where,

- $\mathcal{M}^k_{\mathcal{N}(v)}$ is the aggregated representation of the neighborhood of node v,
- $\mathcal{N}(v)$ is the neighborhood of node v.

(c) Principal Neighborhood Aggregation:

The principal neighborhood aggregation is computed as:

$$h_v^{pna} = \sigma\left(W_p \cdot \mathrm{CONCAT}\left(h_v, \mathcal{M}^k_{\mathcal{N}(v)}\right)\right), \tag{1.3.27}$$

where,

- h_v^{pna} is the output of the PNA layer,

- W_p is the weight matrix,

- CONCAT concatenates the node representation and the neighborhood aggregation.

(d) Node Representation Update:

The node representation is updated as,

$$h_v^{k+1} = h_v^{pna} + h_v^k, \tag{1.3.28}$$

where,

- h_v^k is the representation of node v at layer k.

(e) Final Representation:

The final representation of a node is computed after K layers,

$$h_v = h_v^K. \tag{1.3.29}$$

1.4 Scope of the Thesis

This thesis deals with different techniques of generating embedding for graphs. These can be broadly categorized into two parts, namely, methods for embedding Homogeneous graphs and for Heterogeneous graphs. In homogeneous graph setting, the task addressed here is graph classification. Hence the embeddings are generated keeping this task in mind. This thesis aims to deal with two drawbacks of existing approaches. First one is to design a GNN-based architecture that can handle those graphs where the node features are intervals. And, secondly, to encode hierarchical structure of a

graph while generating graph embedding.On the other hand, for heterogeneous graph, the focus is on performing link prediction and the embeddings are generated accordingly. An application in the area of social recommender system is demonstrated. In this heterogeneous graph setting, this thesis aims to capture Influence of an user in social recommender system in both unsupervised and supervised ways. The contents of the different chapters are mentioned briefly below.

1.4.1 Graph Classification on Homogeneous Graphs

In this subsection, we introduce the chapters where methods have been developed for graph classification on homogeneous graphs and discuss the research gaps that they address.

1.4.1.1 Handling Interval Valued Data in Graph Neural Network: IV-GNN [2]

Interval-valued data is an effective way to represent complex information where uncertainty, inaccuracy etc. are involved in the data space. Interval analysis together with neural network has proven to work well on Euclidean data. However, in many real-life scenario, data follows a complex structure and is been represented as graphs, which is non-Euclidean in nature. Although GNN is a powerful tool to handle graph data, studies on GNNs with interval-valued data is limited. Thus, there is a research gap between the interval-valued data handling approaches and the existing GNN model.Chapter 2 proposes an Interval-Valued Graph Neural Network, a novel GNN model, where we relax the restriction of the feature space taking single-value, without compromising the time complexity of the best performing GNN model of the literature. This model is more general than existing models as any finite set is always a subset of the universal set $\mathbb{R}^n$, which is uncountable. In this chapter, to deal with interval-valued feature vectors, a new aggregation scheme of intervals has been proposed and its expressive power to capture different interval structures is demonstrated. The theoretical findings are experimentally substantiated for graph classification task by comparing its performance with those of the state-of-the-art models on several benchmark and synthetic network datasets.

1.4.1.2 Graph Representation Learning based on Micro-Macro Analysis: GraMMy [3]

Despite the recent advancement of GNNs, the basic message passing scheme of learning often holds back these models in effectively capturing the influence of the nodes from higher order neighborhood. Further, the state-of-the-art approaches mostly ignore the contextual significance of the paths through which the message/information propagates to a node. In order to deal with these two issues, Chapter 3 deals with a novel framework for hierarchical semantics-driven graph representation learning based on Micro-Macro analysis, GraMMy, proposed in [3]. The key idea is to study the graph structure from different levels of abstraction, which not only provides an opportunity for flexible flow of information from both local and higher-order neighbors but also helps in more concretely capturing how information travels within various hierarchical structures of the graph. We incorporate the knowledge gained from micro and macro level semantics into the embedding of a node and use this to perform graph classification. Experimentations on four bio-informatics and two social datasets exhibit the superiority of GraMMy over state-of-the-art GNN-based graph classifiers.

1.4.2 Link Prediction on Heterogeneous Graphs

1.4.2.1 A Recommendation System Based On Social-Influence-aware Graph Embedding Approach: SInGER [4]

Due to the massive explosion of data in social networks in the recent past, exploring user influence has gained popularity in recommender systems. The role of the users and their collaborative activities towards the others are important to model the overall influential behaviours of the users in social networks. In Chapter 4, an unsupervised measure of Influence Score has been proposed to estimate how influential a user is to the other users so that the rating given to an item by that person can affect the other users' perception of the item. Typically, the influence score is estimated considering the trustworthiness/reliability of the user in reviewing the items under each specific category. Later, the users' Influence score has been utilized to predict the missing rat-

ings by employing a social recommender system based on GNN.

1.4.3 User-Reliability-Aware Social Recommendation Framework based on Graph Neural Network [5]

Exploiting user trust information for developing a recommendation system has gained increasing research interest in recent years. Due to the exchange of opinions about items over the social network, trust plays a crucial role for a user to like or dislike an item. GNNs, which have the intrinsic power of integrating node information and topological structure, have a high potential to advance the field of trust-aware social recommendation. However, as of now, this area is little explored, with most of the existing GNN-based models ignoring the trust propagation and trust composition properties. To address this issue, in Chapter 5, a novel GNN-based framework has been proposed that can capture such trust propagation and trust composition aspects by incorporating the concept of 'user-reliability'. The proposed user-reliability-aware social recommendation framework, termed as SoURA, generates the user-embedding and item-embedding considering the user-reliability values, which, in turn, helps in better evaluation of the user trust. Experimental evaluations on the benchmark Ciao and Epinion datasets demonstrate the effectiveness of incorporating user-reliability for finding user-embedding and item embedding in a social recommendation system.

Although the GNNs, with their natural ability to integrate node information and topological structure, have shown enormous potential in the trust-aware social recommendation, these do not implicitly deal with external factors, such as 'item category', that may have a significant impact on user-trust. In Chapter 5, we present a novel approach that projects trust as dependent on the product category. Subsequently, GNN-based social recommendation has been augmented by defining a concept of category-based user-reliability value. The proposed GNN-based model in Chapter 5 for category-wise reliability-aware recommendation (CateReR) finds user-embedding and item-embedding considering the variation of user's reliability over different product cate-

gories. CateReR is also capable of dealing with trust propagation and trust composition, which are often ignored by existing GNN-based models.

Finally, Chapter 6 concludes the thesis with a mention of some directions of future research.

Chapter 2

Handling Interval Valued Data in Graph Neural Network: IV-GNN

2.1 Introduction

Graph Neural Network (GNN) has emerged as a powerful tool in the last few years to find low-dimensional vector embedding of nodes in large graphs. These embeddings aim to capture the structural information of the graph as well as the feature information of the nodes. However, in general, GNNs are focused on graphs where the input feature space is single-valued. However, it is not uncommon for data to be recorded as intervals instead of precise point values in statistics. In general, interval-valued data arises due to two types of situations. Firstly, when there is an uncertainty involved in the feature space, which needs to be captured, and secondly, when there is a class, collection, or group involved instead of individuals. The common example of interval-valued data includes recording systolic and diastolic pressure, an individual's weekly expense range, daily temperature, stock price, etc. Also, time series data can sometimes be treated as interval-valued data. A potential approach to perform this transformation is to use visibility algorithm [40], i.e., assigning visibility range interval on both sides (left and right) of a data point. In this chapter, we describe in detail a GNN model called IV-GNN [2] that is able to handle interval-valued node features. Before going

into the details of IV-GNN, we provide a brief literature survey of handling interval-valued data.

2.1.1 Related Works and their Limitations

The traditional approach to analyze interval-valued data was the Center Method (CM), which uses the midpoint of the interval as an input to a regression model [41]. However, the Center Method (CM) does not consider the variations of the intervals. To overcome this drawback, the Center-Range Method (CRM) uses two interval-valued regression models for mid-points and the ranges of the interval values [42]. However, in general, the mid-points and half ranges are related, which has not been taken into account by CRM [43]. The Bivariate Center and Range Method (BCRM) uses two regression models using mid-points and half-ranges of the intervals, which take care of the effects of interval widths [44].

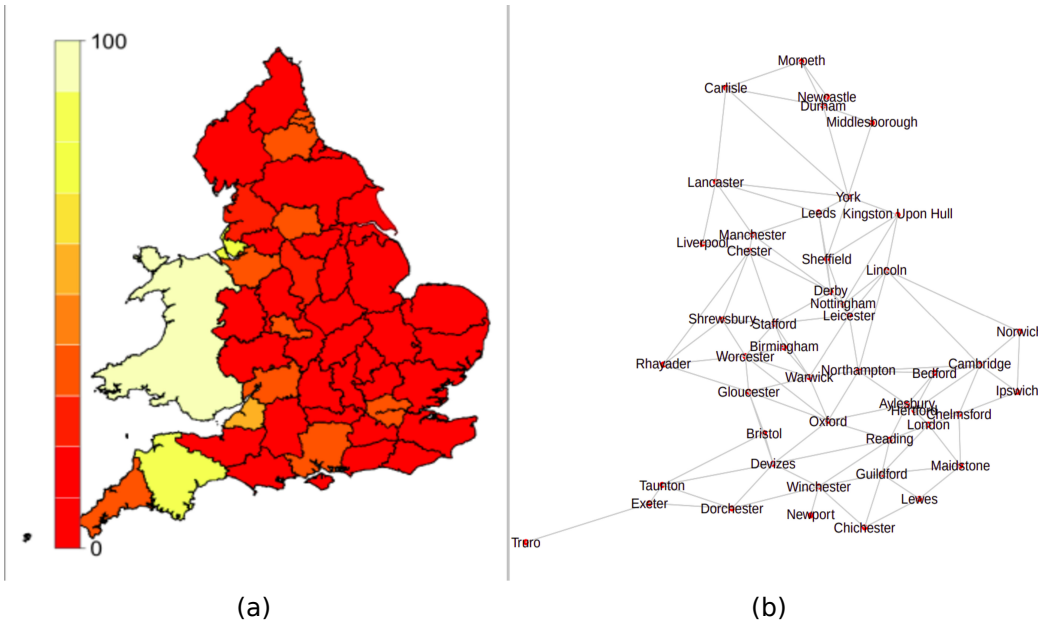


Figure 2.1: a: Weekly cases of Mumps at county level for 2005 in counties of England and Wales. b: Network structure using distances between county towns [2].

All these linear regression models can be used to analyze data that follows a linear pattern. In addition, there are several instances where the interconnectivity between the entities is also important. For example, in cases of infectious diseases such as the mumps in UK at country level in 2005 [See Fig. 2.1[1]] or the global pandemic COVID-19, the intensity of the disease in an area is highly dependent on the other areas it is linked with. The link might be in the form of people's movement, weather patterns such as wind, geographical connection, etc. Also, the daily new cases over a period of time can be modeled as an interval with minimum and maximum new cases. Therefore, problems like modeling the number of infected people in the next few days or deciding whether the transportation between two places should be stopped or not can be modeled as prediction tasks on graphs with interval-valued features. The aim of this chapter is to develop an appropriate GNN architecture where the model accepts the interval-valued feature, performs the embedding generation efficiently, and finally, executes the specific task using those embeddings.

An interesting question to be asked here would be, why not use two end points of an interval as two separate features of a node and apply an existing GNN architecture accepting single-valued node features of a graph. The answer to this question is that although the neural network has the ability to capture relationships among different node features, the interval is a quantitative measurement where there is an order, and the difference between two endpoints is meaningful. Our aim is to exploit this property of interval and develop an appropriate GNN architecture where the model is capable enough to accept multiple intervals and output a single representation on its own. This will allow us to consider an interval as a unit throughout the progress of the algorithm and perform the classification task as a function of the interval-valued feature as well as the structure of the graph.

[1]Figure source: https://www.newton.ac.uk/files/seminar/20140115093010001-153908.pdf

2.1.2 Research Contribution

Motivated by the above discussion, in this chapter, we describe a new aggregation function of intervals, called agr_{new}, and develop a neural architecture called IV-GNN (Interval-valued Graph Neural Network) [2] to apply on interval-valued data using MLPs and the newly developed agr_{new} as its basic building blocks.
Below, we give a summary of the main contributions:

1. A new interval aggregation function agr_{new} is introduced and its representational power is elaborated.

2. Interval structures that previously available interval aggregation functions can not distinguish, are identified.

3. We develop a neural-based architecture Interval-Valued Graph Neural Network (IV-GNN), that can deal with interval-valued features.

4. We discuss the space and time complexities of our proposed algorithm and validate our theoretical findings by performing graph classification tasks on several datasets.

2.2 Background And Definition

As we have already discussed the notion of GNNs in Chapter 1 (see Sec. 1.3.2), here we overview the basics Mathematics of Intervals and Aggregation Functions.

2.2.1 Basic Mathematics of Intervals

The existing works discussed so far in Chapter 1 (see Sec. 1.3.2.2) consider situations when the feature space is single-valued. Before generalizing this idea to interval-valued feature space, in this section we first introduce the mathematical concepts regarding intervals. Note that if the node feature is single-valued, it can be considered a degenerate interval, i.e., the start and the end points of the interval are the same. Here,

we discuss two existing order relations defined on the set of intervals.
Let us consider $\mathcal{U} = \{[a, b] | 0 \leq a \leq b \leq 1\}$ along with $\cap_0$ and $\cup_0$ defined by

$$[x_1, x_2] \cap_0 [y_1, y_2] = [m, m'], \text{ and,}$$

$$[x_1, x_2] \cup_0 [y_1, y_2] = [M, M'],$$

where, $min(x_1, y_1) = m$, $max(x_1, y_1) = M$, $min(x_2, y_2) = m'$ and, $max(x_2, y_2) = M'$

Note that, $(\mathcal{U}, \cap_0, \cup_0)$ forms a complete lattice [45]. A partially ordered set is said to be a complete lattice if all subsets have both a supremum (join) and an infimum (meet). It may be noted that the $\cap_0$ as defined above is biased towards the interval with a lower value. To overcome this drawback, in [46], $(\mathcal{U}, \cap_e, \cup_e)$ is defined as a lattice where the greatest lower bound, $\cap_e$ and the least upper bound, $\cup_e$ are defined as follows,

$$\begin{aligned} [x_1, x_2] \cap_e [y_1, y_2] &= [M, m'], \text{ if } M \leq m', \\ &= [m', m'] \text{ , otherwise.} \end{aligned}$$

$$\begin{aligned} [x_1, x_2] \cup_e [y_1, y_2] &= [M', M'], \text{ if } x_1 = x_2, y_1 = y_2, \\ &= [M, M'], \text{ if } x_1 = x_2 < y_1 < y_2, \\ &= [m, M'] \text{ , otherwise,} \end{aligned}$$

where, $min(x_1, y_1) = m$, $max(x_1, y_1) = M$, $min(x_2, y_2) = m'$ and, $max(x_2, y_2) = M'$.
Note that, $(\mathcal{U}, \cap_e, \cup_e)$ is also a complete lattice. However, $\cup_e$ also has a drawback. One can easily note that whenever two intervals are not intersecting, $\cup_e$ generates a degenerate interval. The new order relation defined in IV-GNN [2] overcomes this drawback.

2.2.2 Aggregation Functions

As mentioned in Chapter 1, a function is required to aggregate the intervals to make them suitable for processing. We denote the aggregation functions as agr_0 and agr_e based on $\cap_0$ and $\cap_e$, respectively. In order to define a more powerful aggregation function, we discuss the properties of the aggregation function that need to be satisfied. Let us denote the minimal and maximal possible input intervals as I_{min} and I_{max}, respectively. Then, any aggregation function [47] for $n \geq 2$, where n is the number of intervals, is defined by $agr : [[0,1] \times [0,1]]^n \rightarrow [0,1] \times [0,1]$ fulfilling at least the two following conditions.

- **Aggregation of n number of I_{min} is I_{min} and Aggregation of n number of I_{max} is I_{max}.**
  More formally,

$$agr(I_{min}, I_{min}, ..., I_{min}) = I_{min},$$
$$agr(I_{max}, I_{max}, ..., I_{max}) = I_{max}.$$

  This condition is denoted as the Boundary condition.

- **Aggregation function must be Monotonically increasing.**
  More formally, for two sets, each containing n number of intervals $(I_1, I_2, ..., I_n) \in [[0,1] \times [0,1]]^n$ and $(J_1, J_2, ..., J_n) \in [[0,1] \times [0,1]]^n$,

$$\text{if } I_i \leq J_i, \forall i \in \{1, 2, ..., n\} \text{ then } agr(I_1, I_2, ..., I_n) \leq agr(J_1, J_2, ..., J_n).$$

  Here, $\leq$ is the order relation defined on the lattice $(\mathcal{U}, \cap, \cup)$ such that $I \leq J \iff I \cap J = I$.

Besides these properties, we want our aggregation function to satisfy two additional properties.

- **Aggregation function should be symmetric.**

  More formally, $agr(I_1, I_2, ..., I_n) = agr(I_{p(1)}, I_{p(2)}, ..., I_{p(n)})$ for any permutation p on $\{1, 2, ..., n\}$.

- **Aggregation function should be idempotent.**

  More formally, $agr(I, I, ..., I) = I, \forall I \in [0, 1] \times [0, 1]$.

2.3 Theoretical Framework

We introduce a new order relation $\subseteq_{new}$ on $\mathcal{U}$ such that $(\mathcal{U}, \cap_{new}, \cup_{new})$ forms a lattice.

Definition 7. *$\subseteq_{new}$ is a binary relation on $\mathcal{U}$ defined as below*

$$[x_1, x_2] \subseteq_{new} [y_1, y_2] \iff (y_1 < x_1) \textit{ or } (x_1 = y_1 \textit{ and } x_2 \leq y_2).$$

Proposition 1. *$(\mathcal{U}, \subseteq_{new})$ forms a poset.*

To show $(\mathcal{U}, \subseteq_{new})$ forms a poset, we have to show the relation $\subseteq_{new}$ is reflexive, antisymmetric, and transitive.

- **Reflexivity**

  $$[x_1, x_2] \subseteq_{new} [x_1, x_2] \text{ as } x_1 = x_1 \text{ and } x_2 \leq x_2.$$

  Hence, $\subseteq_{new}$ is reflexive.

- **Antisymmetricity**

$$[x_1, x_2] \subseteq_{new} [y_1, y_2] \implies y_1 < x_1 \text{ or}$$
$$x_1 = y_1 \text{ and } x_2 \leq y_2 ....(i)$$

$$[y_1, y_2] \subseteq_{new} [x_1, x_2] \implies x_1 < y_1 \text{ or}$$
$$y_1 = x_1 \text{ and } y_2 \leq x_2 ....(ii)$$

(i) and (ii) imply

$$x_1 = y_1 \text{ and } x_2 = y_2.$$

Hence,

$$[x_1, x_2] = [y_1, y_2].$$

Thus, $\subseteq_{new}$ is antisymmetric.

- **Transitivity** can be shown similarly.

Proposition 2. *($\mathcal{U}, \subseteq_{new}$) forms a lattice.*

($\mathcal{U}, \subseteq_{new}$) is a lattice where the greatest lower bound, say $\cap_{new}$ and the least upper bound, say $\cup_{new}$ are defined as follows,

$$\begin{aligned}
[x_1, x_2] \cap_{new} [y_1, y_2] &= [M, m'], \text{ if } M \leq m' \leq M' \neq 1, \\
&= [M, 1] \text{ , otherwise.} \\
[x_1, x_2] \cup_{new} [y_1, y_2] &= [m, M'], \text{ if } M' \neq 1, \\
&= [m, m'] \text{ , otherwise.}
\end{aligned}$$

where, $min(x_1, y_1) = m$, $max(x_1, y_1) = M$, $min(x_2, y_2) = m'$ and, $max(x_2, y_2) = M'$

Proposition 3. *($\mathcal{U}, \subseteq_{new}$) forms a bounded lattice.*

For an arbitrary interval $[x, y] \in \mathcal{U}$, we have $0 \leq x \leq y \leq 1$. Hence $[1, 1] \subseteq_{new} [x, y] \subseteq_{new} [0, 1]$.
We denote the aggregation function as agr_{new}, which is based on $\cap_{new}$.

Proposition 4. *agr_{new} satisfies all four conditions of aggregation function.*

agr_{new} satisfies boundary conditions where $[1, 1]$ and $[0, 1]$ are the minimal and maximal possible inputs respectively.

2.4 Proposed Framework: Interval-Valued Graph Neural Network (IV-GNN)

We develop a general GNN model where the feature need not be single-valued. Releasing this constraint on the feature space, our model can capture the graph's structural properties and extract useful information from interval-valued features of the nodes. As a result, our proposed architecture is much more general in nature, and to the best of our knowledge, no existing models of GNN in literature can accept nodes' features, which are intervals.

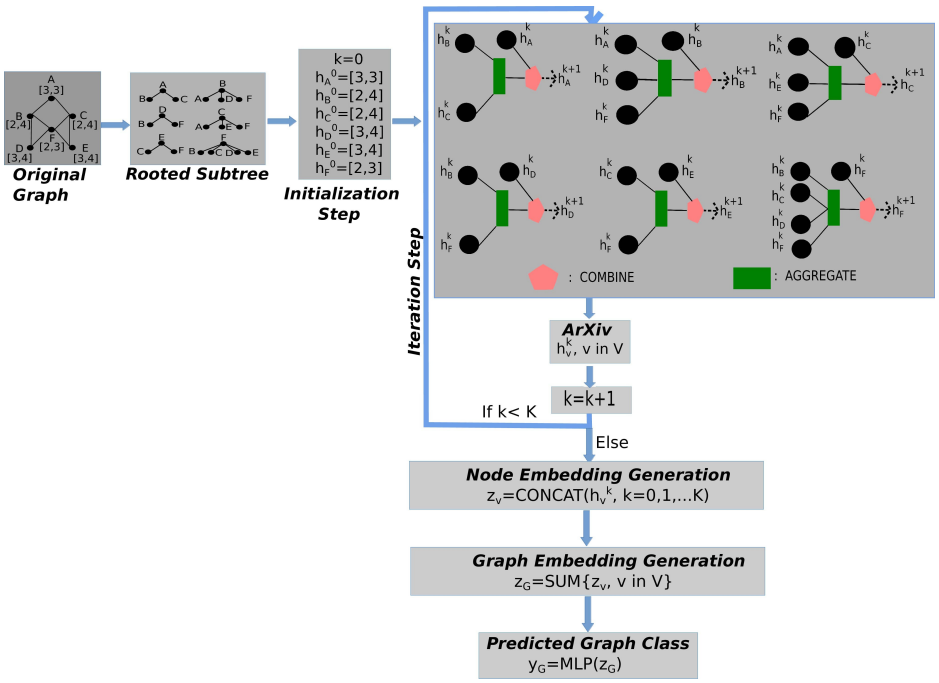


Figure 2.2: An overview of the proposed framework: IV-GNN [2].

2.4.1 AGGREGATE and UPDATE functions of IV-GNN

As we have already discussed in Chapter 1 (see Sec. 1.3.2), GNN architecture has two primary functions, namely $AGGREGATE$ and $COMBINE$, we use our newly developed interval aggregation function agr_{new} function to aggregate the neighboring

nodes' embedding to extract maximum information out of it. As shown in Fig. 2.2, we develop our model with aggregation and update function on k-th iteration defined as

$$h_v^k = \Phi(h_v^{k-1}, agr_{new}(\{h_u^{k-1} : u \in N(v)\})). \tag{2.4.1}$$

, where Φ is the update function. To choose Φ, we take the help of the Universal Approximation Theorem [48], which states that using the multi-layer feed-forward architecture in a neural network framework makes it a universal approximator of any continuous functions under mild assumptions on the activation function [49]. However, the limitation of continuity of the function was released much later. It has been shown that a single hidden layer feed-forward neural network (SLFNN) can approximate any real, piece-wise continuous function almost uniformly [50].
Therefore, IV-GNN has the updating step as,

$$h_v^k = MLP^k(agr_{new}((1+\epsilon^k)h_v^{k-1}, agr_{new}\{h_u^{k-1} : u \in N(v)\})). \tag{2.4.2}$$

Here, ϵ is a parameter that decides how much weight to give to the node's existing embedding. It can be learned or assigned as a fixed scalar.

2.4.2 Details of Updation Step

The proposed Interval-Valued Graph Neural Network (IV-GNN) operates on interval-valued input and output in order to generate embeddings. As discussed earlier, for any given node, the newly developed aggregation function agr_{new} has been used twice in every update step. Firstly, it will accumulate the neighbors' intervals and express them as a single interval. And secondly, it will combine neighborhood information with the node itself. We have discussed the agr_{new} in detail in the previous section. Now, we give the details of the neural architecture of our model.

2.4.2.1 Neural Architecture of IV-GNN

One of the basic building blocks of our proposed model IV-GNN is Multi-Layer Perceptron (MLP). However, unlike the commonly used architecture, the MLP used in the proposed model deals with interval-valued inputs and outputs, but the weights and biases are single-valued [51]. In order to understand, we assume that, the MLP has only one hidden layer with h units.

Let us consider the interval-valued inputs as $X_i = [X_i^{lower}, X_i^{upper}]$, with $i = 1, 2, ..., n$. The output of the j-th hidden unit is the single-value weighted linear combination of inputs and the bias as follows,

$$S_j =< S_j^1, S_j^2 >=< w_{j0} + \sum_{i=1}^{n} w_{ji} \frac{X_i^{lower} + X_i^{upper}}{2}, \sum_{i=1}^{n} |w_{ji}| \frac{X_i^{upper} - X_i^{lower}}{2} > .$$

After that, the $tanh$ function was used as an activation function,

$$A_j = tanh(S_j) = [tanh(S_j^1 - S_j^2), tanh(S_j^1 + S_j^2)].$$

Finally, the output is the linear combination of the hidden layer output and bias, which is the output of the whole updation step.

2.4.3 Graph-Level READOUT function of IV-GNN

The purpose of this graph-level read-out function is to get the embedding of the graph using the embedding of the nodes. If we want to perform jobs like node classification [52] or link prediction [53] within a graph, then the node embedding using aggregation and update function at the node level is sufficient.

While selecting the Graph-level READOUT function, we want to focus on the following aspects:

- We want to use the structural information and node embedding that we have gotten after every iteration. It may happen that the node embedding from an earlier iteration captures more information about the graph than the final iteration.

- The graph-level function should be injective to have our GNN variant as powerful as the WL-test of isomorphism by distinguishing between different structures/node features.

Hence, to achieve a skip connection-like architecture similar to Jumping Knowledge [54] and maximal distinguishing power, we concatenate the SUM of the node embeddings after every iteration.

$$z_G = CONCAT(SUM(\{h_v^k | v \in G\}) | k = 0, 1, ..., K). \tag{2.4.3}$$

2.4.4 Challenging Structures for agr_0 and agr_e .

The main idea of choosing a powerful aggregation function is to capture and compress the amount of structural and feature information from the nodes in its aggregated output value. Also, the aggregation function should be permutation invariant. That is, the order of the interval during aggregation should be immaterial. In this context, agr_0 , agr_e and agr_{new}, all are satisfying this condition. In Fig. 2.3, we have shown the ranking of three aggregation functions pictorially with respect to their representational power. We have denoted the root node as the red node and the adjacent nodes of the root node as the black node whose features need to be aggregated and combined with the root node. In Table. 2.1, we have illustrated these facts with the help of examples. In the Fig. 2.4a, we have three intervals, $I_1 = [0.1, 0.2]$, $I_2 = [0.1, 0.3]$ and $I_3 = [0.15, 0.3]$. We construct two sets of intervals S_1 and S_2, where $S_1 = \{I_1, I_2\}$ and $S_2 = \{I_1, I_3\}$, which need to be aggregated. Now,

$$agr_0(S_1) = agr_0(S_2) = I_1.$$

Hence, in this case, agr_0 fails to capture the desired information about the intervals. However, agr_e and agr_{new} will give the aggregated intervals as the intersecting subintervals.

$$agr_e(S_1) = agr_{new}(S_1) = [0.1, 0.2].$$

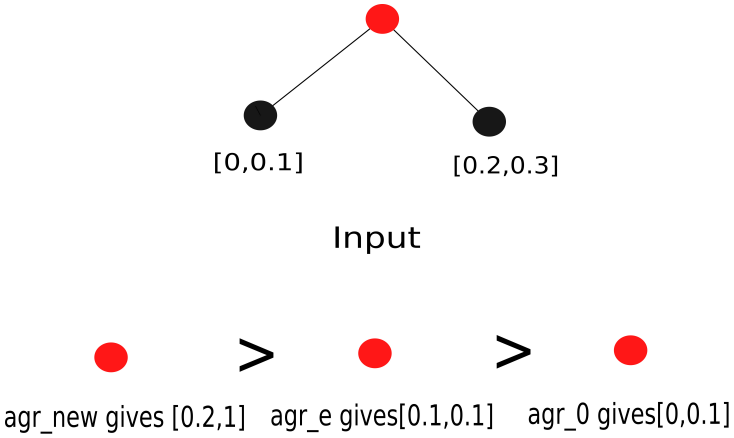


Figure 2.3: Ranking by expressive power for agr_0, agr_e, agr_{new}. Among these three aggregation functions, agr_{new} has the maximum ability to capture the structural and feature related information. One thing to notice that, the right end point of an resulting interval equals to 1 expresses the fact that two aggregating intervals are non overlapping. agr_e is equally powerful as agr_{new} when two intervals have non-null intersection. agr_0 captures the smaller interval (according to the order relation) and ignores the other interval [2].

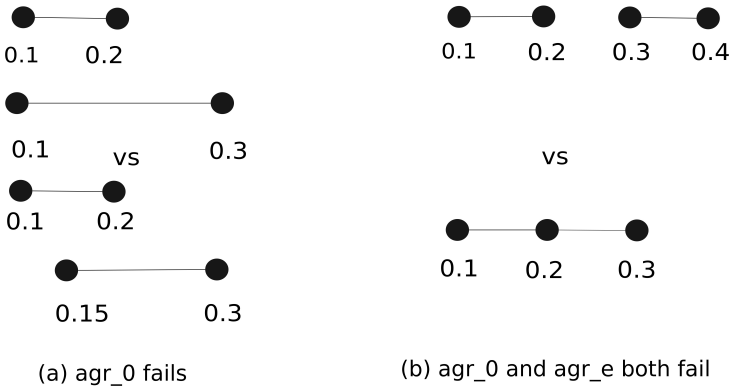


Figure 2.4: Examples of interval structures that agr_0 and agr_e fail to distinguish: In the left picture, agr_0 is giving the same aggregated interval even though two sets of intervals are different. In the right picture, agr_0 and agr_e both are unable to recognize the differences between the two sets of intervals [2].

$$agr_e(S_2) = agr_{new}(S_2) = [0.15, 0.2].$$

In Fig. 2.4b, two sets of intervals need to be aggregated: $S_3 = \{I_1, I_4\}$ and $S_4 = \{I_1, I_5\}$, where $I_4 = [0.3, 0.4]\}$ and $I_5 = [0.2, 0.3]$. Here also, agr_0 will fail to distinguish between them.

$$agr_0(S_1) = agr_0(S_2) = I_1.$$

But according to definition, agr_e will give same degenerate interval $[0.2, 0.2]$ for both S_3 and S_4. However, agr_{new} can differentiate between them as it will aggregate and give the resultant intervals as $[0.3, 1]$ and $[0.2, 0.2]$ for S_3 and S_4 respectively.

$$agr_e(S_3) = agr_e(S_4) = [0.2, 0.2].$$

$$agr_{new}(S_3) = [0.3, 1],$$
$$agr_{new}(S_4) = [0.2, 0.2].$$

As I_1 and I_4 are non-intersecting, agr_{new} will be able to capture the uncertainty and express it by assigning a broader interval.

Table 2.1: Comparison between three interval aggregators.

Interval 1	Interval 2	agr_0	agr_e	agr_{new}
[0.1,0.2]	[0.1,0.3]	[0.1,0.2]	[0.1,0.2]	[0.1,0.2]
[0.1,0.2]	[0.15,0.3]	[0.1,0.2]	[0.15,0.2]	[0.15,0.2]
[0.1,0.2]	[0.3,0.4]	[0.1,0.2]	[0.2,0.2]	[0.3,1]
[0.1,0.2]	[0.2,0.3]	[0.1,0.2]	[0.2,0.2]	[0.2,0.2]

The agr_e will perform well if the two intervals have a non-null intersection. So, whenever the node features are not very diverse, agr_e will be as powerful as the agr_{new} aggregator.

2.4.5 Model Training

To estimate the model parameters of IV-GNN, we need to specify an objective function to optimize. Since the task we focus on in this work is Graph Classification task, loss is computed as the sum of the difference between the actual graph class and the predicted graph class for the graphs in the dataset. The objective function L can be mathematically formulated as follows,

$$L = \sum_{G \in \mathcal{G}} |y_G - y'_G|, \tag{2.4.4}$$

where y_G be the actual class label of graph G, and y'_G be the predicted class label of graph G.

2.4.6 Time and Space Complexity of Training the Embedding Generation Process of IV-GNN

In order to find the worst-case scenario, we assume that all $||E||$ edges are connected to all $||V||$ nodes of the graph G. As IV-GNN is a full-batch gradient descent process, it requires storing all the embeddings found from the intermediate layers, which requires $O(Kn)$ storage for one node. Here, K denotes the number of layers, and n denotes the dimension of the embedding space. For the sake of simplicity, we keep the embedding space dimension the same for every layer. Furthermore, at every layer, a weight matrix of size $n \times n$ is involved, which includes $O(Kn^2)$ storage in total. Therefore, overall, IV-GNN has a space complexity of $O(K||V||n + Kn^2)$.

Now, we illustrate the time complexity of our proposed model. As discussed previously, IV-GNN stores intermediate embeddings of every node generated from each lower layer. In contrast to mini-batch algorithms like GraphSAGE [35], IV-GNN utilizes those saved embeddings and reuses those in the upper layer. Therefore, at every layer, previous layers' embeddings are multiplied with the weight matrix of size $n \times n$, which includes n^2-many multiplications, followed by some element-wise operations. Therefore, as a whole, for K many layers and $||V||$ many nodes, IV-GNN has time

complexity $O(K||V||n^2 + K||E||n)$.

2.5 Experimental Results

This section discusses the dataset used for experiments, and we evaluate our theoretical findings by comparing the training and test set performances of IV-GNN on the synthetic and real-life datasets.

2.5.1 Datasets

We have used six synthetic datasets and six real-life datasets (four bioinformatics datasets and two social network datasets) to demonstrate the efficiency of our model.

1. Synthetic Datasets: Our basic idea is to generate random graphs with various numbers of nodes. Then based on two topological properties, we give tag and feature interval to the nodes and classify every graph in the datasets. The summary statistics of these synthetic datasets are provided in Table-2.2. The topological properties are listed below,

   - **Density** The density of a graph is defined as the ratio of the number of edges and the number of nodes [55], i.e.,

   $$\text{density}(G) = \frac{|E|}{|V|},$$

   where $G = (V, E)$ denotes a graph, V is the set of nodes and E is the edge set of the graph G. Average density of the dataset $D = \{G_i, i = 1, ..., n\}$ can be calculated as,

   $$avg\_density = \frac{\sum_{i=1}^{n} density(G_i)}{n}.$$

   We assign,

   $$\text{Graph class (G)} = \begin{cases} 1, & \text{if density } (G) < \text{avg\_density}, \\ 0, & \text{otherwise.} \end{cases}$$

We assign,

$$\text{tag}(v) = \begin{cases} 1, & \text{if degree } (v) < \text{average degree}, \\ 0, & \text{otherwise}, \end{cases}$$

where the average degree is calculated over all nodes in the dataset.

To assign the interval-valued feature to a node v, we follow the following rule,

$$\text{feature interval}(v) = \begin{cases} [d_{min}, d_{max}], & \text{if} |N(v)| > 1, \\ [-1, d_{max}], & \text{if} |N(v)| = 1, \\ [-1, 0], & \text{otherwise}, \end{cases}$$

where

$$d_{min} = \min_{u \in N(v)} degree(u),$$

$$d_{max} = \max_{u \in N(v)} degree(u).$$

We have created three datasets, SYNTHETIC_1_200, SYNTHETIC_1_1000, SYNTHETIC_1_2000, with 200, 1000, 2000 graphs, respectively, according to this construction.

- **Average clustering coefficient** The clustering coefficient $c(u)$ of a node u,is a measure of the likelihood that any two neighbors of u are connected [56]. Mathematically, the clustering coefficient of a node u can be formulated as:

$$c(u) = \frac{\lambda(u)}{\tau(u)}.$$

where $\lambda(u)$ is the number of triangles engaging the node u. By triangle, we mean a complete graph with three nodes and

$$\tau(u) = \frac{degree(u)(degree(u) - 1)}{2},$$

i.e., the number of triples a node u has.

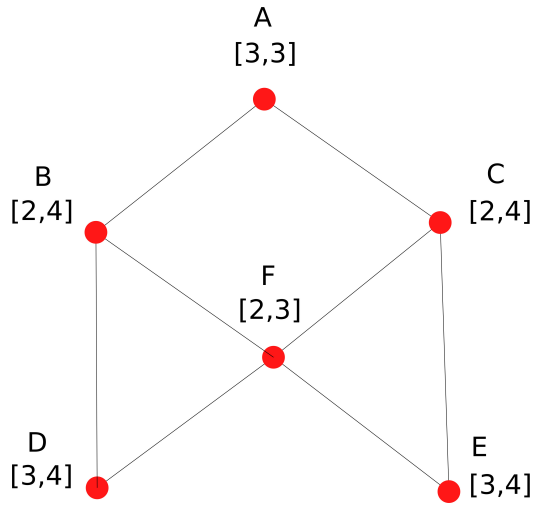


Figure 2.5: Explanation of assigning feature interval to a node. For node A, it has two neighbors of degree 3, resulting in $d_{min} = d_{max} = 3$. Hence the feature interval$(A) = [d_{min}, d_{max}] = [3, 3]$. Similarly, node F has 4 neighbors of degree $2, 2, 3, 3$. Hence d_{min} and d_{max} are 2 and 3 respectively. Therefore, feature interval$(F) = [2, 3]$ [2].

In other words, the clustering coefficient for node u is the ratio of the actual number of edges between two nodes from the neighbors of u and the maximally possible numbers of edges between them. The clustering coefficient $C(G)$ of a graph G is the average of $c(u)$ taken over all the nodes in the graph, i.e.,

$$C(G) = \frac{1}{n}\sum_{i=1}^{n} c(u_i)$$

. Average clustering coefficient of the dataset $D = \{G_i, i = 1, ..., n\}$ can be calculated as,

$$avg\_cluster = \frac{\sum_{i=1}^{n} C(G_i)}{n}.$$

We assign,

$$\text{Graph class}(G) = \begin{cases} 1, & \text{if } C(G) < avg\_cluster, \\ 0, & \text{otherwise.} \end{cases}$$

We use the node's degree as its tag.

$$\text{tag}(v) = degree(v).$$

To assign the interval-valued feature to a node v, we follow the following rule,

$$\text{feature interval}(v) = [c_{min}, c_{max}],$$

where

$$c_{min} = \min_{u \in N(v)} c(u),$$

$$c_{max} = \max_{u \in N(v)} c(u).$$

We have created three datasets SYNTHETIC_2_200, SYNTHETIC_2_1000, SYNTHETIC_2_2000 with 200, 1000, 2000 graphs respectively according to this construction.

Table 2.2: Statistics of the synthetic datasets

Dataset	***Size***	***Classes***	***Avg. nodes***	***labels***
SYNTHETIC_1_200	200	2	19.94	2
SYNTHETIC_1_1000	1000	2	19.83	2
SYNTHETIC_1_2000	2000	2	19.92	2
SYNTHETIC_2_200	200	2	19.94	23
SYNTHETIC_2_1000	1000	2	19.83	25
SYNTHETIC_2_2000	2000	2	19.92	25

2. Real-life datasets: We have used 4 bioinformatic datasets namely MUTAG, PROTEINS, PTC, NCI1 and 2 datasets namely IMDB-BINARY and COLLAB [57] for our experiment.

   (a) Bio-informatics datasets: 4 datasets MUTAG, PROTEINS, PTC and NCI1 [57] have been used for our experiment. The summary statistics of these bioinformatic datasets are provided in Table-2.3.

- MUTAG consists of 188 graphs which have 7 discrete node labels. Each graph in the dataset represents a chemical compound [58].
- In the dataset PROTEINS, nodes represent secondary structure elements (SSEs) and two nodes share an edge if they appear as adjacent in the amino-acid sequence. Graph nodes have 3 different labels such as helix, sheet or turn [59].
- PTC includes 344 chemical compounds that describes the carciogenicity for male and female rats having 19 discrete node labels [60].
- NCI1 is a balanced dataset with 4100 nodes with 37 discrete labels, published by the National Cancer Institute (NCI). It contains chemical compounds, that are found to have the ability to suppress or inhibit the growth of a panel of human tumor cell lines [61].

Table 2.3: Statistics of the Bioinformatics datasets used

Dataset	***Size***	***Classes***	***Avg. nodes***	***labels***
MUTAG	188	2	17.9	7
PTC	344	2	25.5	19
PROTEINS	1113	2	39.1	3
NCI1	4110	2	29.8	37

(b) Social network datasets: 2 datasets, IMDB-BINARY and COLLAB [57] have been used for our experiment. The summary statistics of these datasets are provided in Table-2.4.

- Movie Collaboration Dataset: IMDB-BINARY is a dataset of movie collaboration, wherein in each graph, nodes represent actors/actresses and two nodes share an edge if two actors/actresses act in the same movie. There are two graph classes Action and Romance genres.
- Scientific collaboration dataset: COLLAB is a dataset of scientific collaboration, acquired from 3 public collaboration datasets, namely High Energy Physics, Condensed Matter Physics and Astro physics [62]. In [63], ego-networks of various researchers from each field has been

generated, and each graph has a label according to the field of the researcher. Now the task will be to determine an ego-collaboration graph's label of a researcher.

Table 2.4: Statistics of the Social network datasets used

Dataset	***Size***	***Classes***	***Avg. nodes***	***labels***
IMDBBINARY	1000	2	19.8	-
COLLAB	5000	3	74.5	-

2.5.2 Results of Comparative Study on Model Performance

Our goal is to allow the model to capture structural information and feature information from the graph. Therefore, we like to evaluate our model IV-GNN on Graph Classification task with interval-valued features of the nodes.

2.5.2.1 Data Preparation

To convert the feature values into intervals, we prepare the data as follows: rather than using the tag of a node as a node feature, we give a bias of k_1 and k_2, where k_1 and k_2 are selected from the normal distribution $N(0, 1)$ to generate an interval-valued feature for that particular node. For example, if in a graph, a node has a tag as c, then we assign $[c - k_1, c + k_2]$ as its feature interval.

2.5.2.2 Baselines

We evaluate our IV-GNN by comparing it with other frameworks with different interval aggregation functions. As there is no existing model that can handle interval-valued features of the nodes, we use existing interval aggregation functions with the same neural architecture as of IV-GNN, which will show expressive power of agr_{new} against that of others experimentally. The details of the baselines are discussed below.

- agr_0**-based GNN:** In this model, we choose agr_0 for interval-aggregation and SUM as Graph-Level $READOUT$ function.

- agr_e**-based GNN:** In this model, interval-aggregation function agr_e has been used as $AGGREGATE$ function. Like before, we use SUM as the Graph-Level $READOUT$ function.

We report the training set performance and test set performance in Figure 2.6 and Table 2.5, respectively.

2.5.3 Performance with Degenerate Interval: A Special Case of IV-GNN

In order to examine the performance of IV-GNN for single-valued features, we compare IV-GNN with respect to the state-of-the-art approaches that accept single-valued features. For this experiment, we treat the exact value of the feature as a degenerate interval, i.e., we use the exact feature value as the same starting and end points of the interval. We report the training set performance and test set performance in Figure 2.7 and Table 2.6, respectively.

2.5.3.1 Baselines

We compare IV-GNN with *three* state-of-the-art GNN models GraphSage [35], GCN [9], and GIN [36] as briefly discussed in the introduction section(See-1.3.2.2).

2.5.4 Parameter Settings

For IV-GNN, we adopt the parameter settings based on our hyperparameter study [see 2.5.7], and details are as follows

- We have used 5-layers of each GNN block where every MLP will have 2 layers excluding the input layer.
- Each hidden layer has $\{32, 128\}$ hidden units. We have used Batch Normalization in each hidden layer.
- We have used Adam optimizer [64] with the initial learning rate of 0.01 and decay the learning rate by 0.5 after every 50 epochs.

- Input batch size of training is $\{16, 64\}$.
- The final layer dropout is 0.5 [65].

As recommended for GIN, we have performed a 10-fold cross-validation with LIBSVM [57, 66] for 350 epochs.

2.5.5 Results and Discussion

2.5.5.1 Training Set Performance

We have already theoretically analyzed the representational power of our proposed IV-GNN. Now, we validate our theoretical findings by comparing training accuracies on various datasets. Ideally, the architecture, which has stronger representational power, should fit the data more accurately, resulting in better training performance. Training set performance gives an idea about how well the model learned from training data.

Fig. 2.6 shows training curves of IV-GNN and interval-valued GNN with alternative aggregation functions with the same hyperparameter settings. In comparison, the GNN variants, which use a less powerful interval aggregator as $AGGREGATE$ function, cannot learn from many datasets. The reason behind this observation is that agr_{new} has more distinguishing power than agr_0 and agr_e. Between agr_0 and agr_e, agr_e performs better because, as we have seen theoretically, with nonrepetitive feature value, agr_e is equally powerful as agr_{new}.

Fig. 2.7 shows training curves of IV-GNN accepting degenerate interval-valued features and other state-of-the-art GNN models that accept single-valued features. As we can see, the curve of IV-GNN outgrows that of others, which proves that IV-GNN learns from the data much better than other models in most of the cases. In cases of dataset IMDB-BINARY, GIN captures the dataset slightly better than IV-GNN. However, our proposed model IV-GNN is able to beat the other two models quite efficiently.

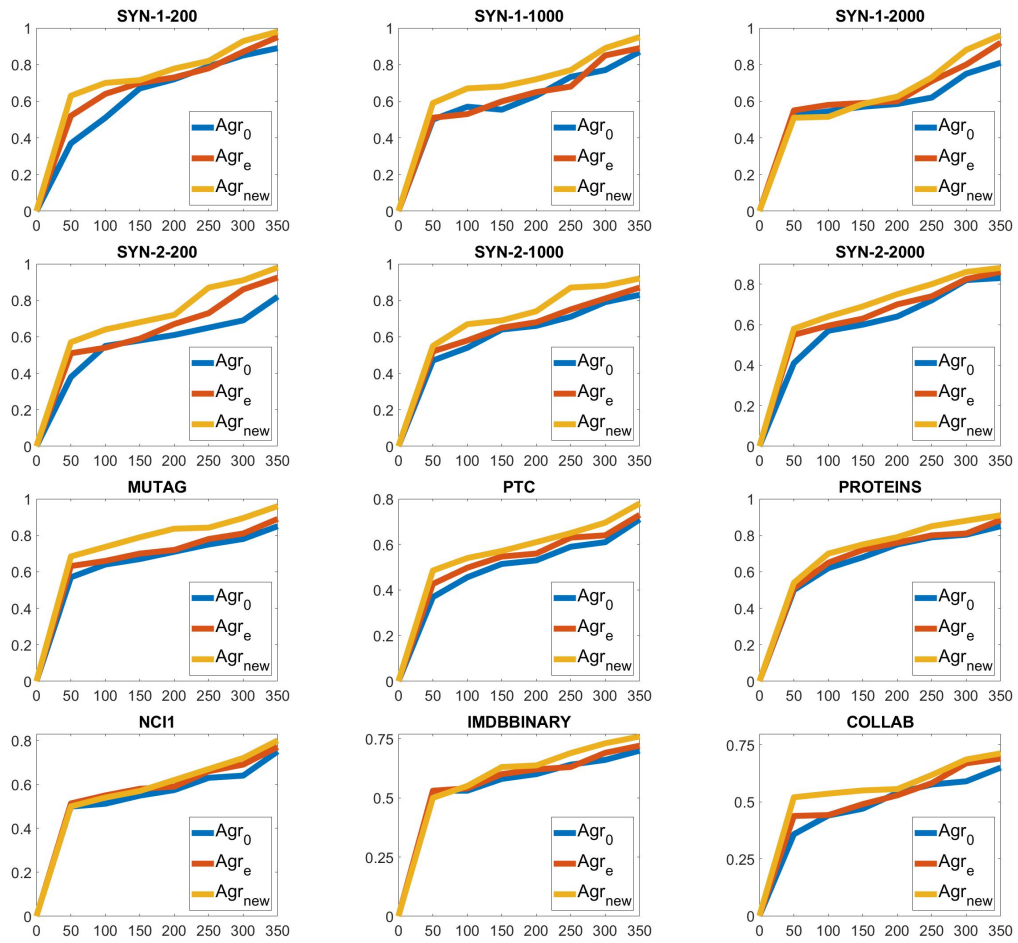


Figure 2.6: Training set performance of IV-GNN and less powerful interval-valued feature accepting GNNs. The X-axis and Y-axis show the epoch number and the accuracy of training respectively [2].

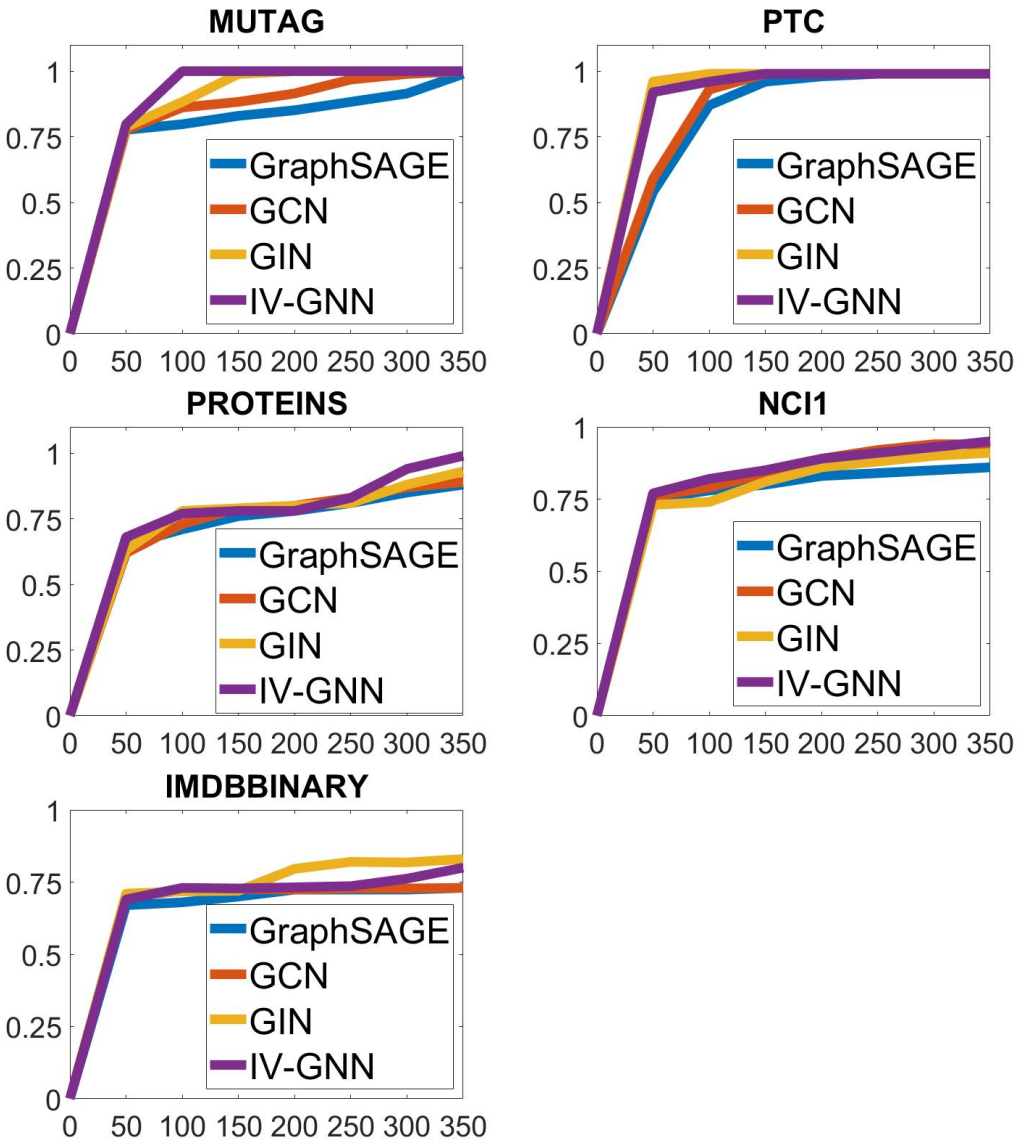


Figure 2.7: Training set performance of IV-GNN accepting degenerate interval value and other single value feature accepting GNN. The X-axis and Y-axis show the epoch number and the accuracy of training, respectively [2].

2.5.5.2 Test Set Performance

Now, we compare test set accuracies with respect to different interval aggregation-based GNN models and state-of-the-art GNN models.

We have performed this experiment for the synthetic as well as real-life datasets for 350 epochs and report the best cross-validation accuracy mean and standard deviation averaged over the 10 folds after performing each experiment *ten* times. In Table 2.5, we have seen that IV-GNN can capture graph structures and generalize well in most of the cases among other variants, showing 4% better accuracy on average.

Table 2.5: Test set classification accuracies in percentage

Dataset	agr_0***-based model***	agr_e***-based model***	agr_{new}***-based model IV-GNN***
SYNTHETIC_1_200	89.33 $\pm$ 0.13	95.00 $\pm$ 0.19	98.39 $\pm$ 0.24
SYNTHETIC_1_1000	87.82 $\pm$ 0.66	89.82 $\pm$ 0.42	95.01 $\pm$ 0.24
SYNTHETIC_1_2000	81.80 $\pm$ 0.40	92.88 $\pm$ 0.36	96.05 $\pm$ 0.31
SYNTHETIC_2_200	82.98 $\pm$ 0.10	92.48 $\pm$ 0.89	98.92 $\pm$ 0.39
SYNTHETIC_2_1000	83.92 $\pm$ 0.55	87.75 $\pm$ 0.26	92.10 $\pm$ 0.79
SYNTHETIC_2_2000	83.31$\pm$ 0.19	86.75 $\pm$ 0.27	88.54 $\pm$ 0.67
MUTAG	85.79 $\pm$ 0.04	89.85 $\pm$ 0.22	92.37 $\pm$ 0.26
PTC	61.10 $\pm$ 0.19	63.6 $\pm$ 0.23	67.6 $\pm$ 0.25
PROTEINS	78.09 $\pm$ 0.04	80.7 $\pm$ 0.54	83.3 $\pm$ 0.06
NCI1	75.6 $\pm$ 0.05	77.7 $\pm$ 0.03	80.3 $\pm$ 0.10
IMDB-B	69.24 $\pm$ 0.03	72.41 $\pm$ 0.12	72.5 $\pm$ 0.16
COLLAB	65.6 $\pm$ 0.02	69.4 $\pm$ 0.02	71.1 $\pm$ 0.16

In the special case for degenerate intervals, we perform experiments *ten* times on real-life datasets only and report the mean and standard deviation of the accuracies. We could not perform experiments of GraphSAGE and GCN on the dataset COLLAB due to memory bound. As we can see in the table 2.6, IV-GNN performs much better than the other state-of-the-art approaches on four out of six datasets and achieves a performance gain of 7% on average. On the other hand, for datasets like IMDB-BINARY and COLLAB, IV-GNN's accuracy is not strictly the highest among GNN variants. However, IV-GNN is still comparable to the best-performing GNN because a paired t-test at significance level 10% does not distinguish IV-GNN from the best.

Table 2.6: Test set classification accuracies in percentage

Dataset	*GraphSAGE*	*GCN*	*GIN*	*IV-GNN*
MUTAG	85.4 ± 0.77	82.9 ± 0.66	89.4 ± 0.84	94.7 ± 0.24
PTC	63.3 ± 0.94	66.9 ± 0.19	64.6 ± 0.74	68.5 ± 0.34
PROTEINS	75.8 ± 0.34	76.23 ± 0.14	76.75 ± 0.98	88.1 ± 0.63
NCI1	78.1 ± 0.34	79.2 ± 0.75	82.5 ± 0.11	83.92 ± 0.96
IMDB-B	71.38 ± 0.97	72.41 ± 0.9	74.1 ± 0.2	73.35 ± 0.68
COLLAB	-	-	80.2 ± 0.53	79.85 ± 0.3

2.5.6 Runtime Comparison

We have conducted an extensive experimental evaluation of IV-GNN, comparing its performance with three other state-of-the-art approaches, namely GraphSAGE, GCN, and GIN, on five diverse datasets. Unfortunately, we were unable to include the COLLAB dataset in our experiment due to memory constraints, as both GraphSAGE and GCN exceeded the available memory. Our results, illustrated in Fig. 2.8, reveal that IV-GNN and GIN exhibit comparable computational efficiency, with similar runtime performances. This is consistent with our theoretical expectations, as both IV-GNN and GIN leverage the previously computed nodes' embeddings to reduce computational overhead. In contrast, GraphSAGE and GCN rely on more traditional methods, which result in higher computational costs. Notably, GraphSAGE failed to outperform GCN on these datasets, which can be attributed to the fact that mini-batch-based algorithms, such as GraphSAGE, are more effective when dealing with extremely large graphs, where the benefits of batching can be fully realized. Overall, our experimental results demonstrate the efficacy of IV-GNN and GIN in achieving a balance between accuracy and computational efficiency, making them suitable choices for large-scale graph learning tasks. Additionally, the results highlight the importance of considering the trade-offs between different algorithmic approaches and the characteristics of the datasets being used.

2.5.7 Empirical Study on Hyperparameter Setting

We have performed empirical study on various hyperparameters involved in the model and our experimental finding can be depicted in Fig. 2.9.

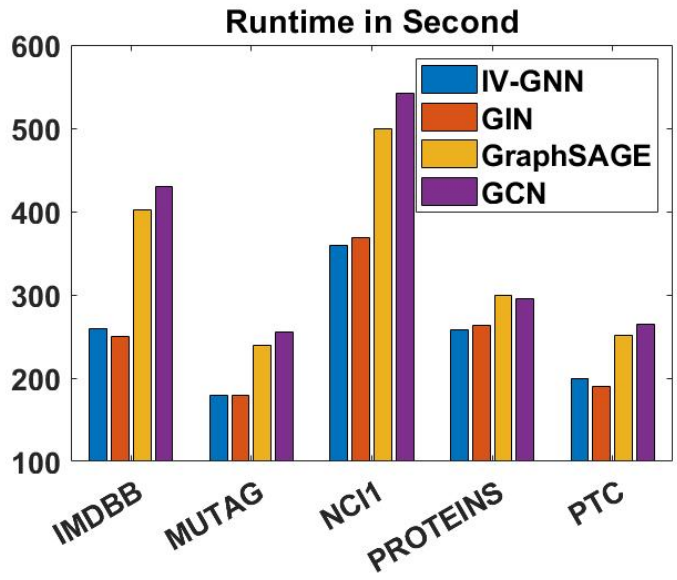


Figure 2.8: Comparison in runtime of different GNN models [2].

Effect of Batch Size Batch size means the number of training examples to work through before updating the internal model parameters. We have experimented with the batch size $\{16, 32, 64, 128\}$ and as we can see for the relatively smaller datasets such as MUTAG, PTC, and PROTEINS, smaller batch size works better and learn from the more enormous datasets such as COLLAB, NCI1 and IMDB-BINARY, larger batch size performs better. So we take batch size as $\{16, 64\}$ depending on the size of the datasets.

Effect of Different Learning Rate The learning rate is a measure of the step size towards moving to the minimum of the loss function [67]. A lower learning rate can slow down the convergence process, while a too-high learning rate may jump over the minima and oscillate. Based on the experimental result, we find that between $\{0.1, 0.01, 0.001\}$, 0.01 gives maximum accuracy for IV-GNN.

Effect of Hidden Dimension We have experimented with the different number of units in the hidden layer of the MLP, such as $\{16, 32, 64, 128\}$. Here also, smaller hidden dimension works well for smaller datasets as larger hidden dimension may cause underfitting. So, we take 32 as a hidden dimension for the bioinformatics dataset. On the

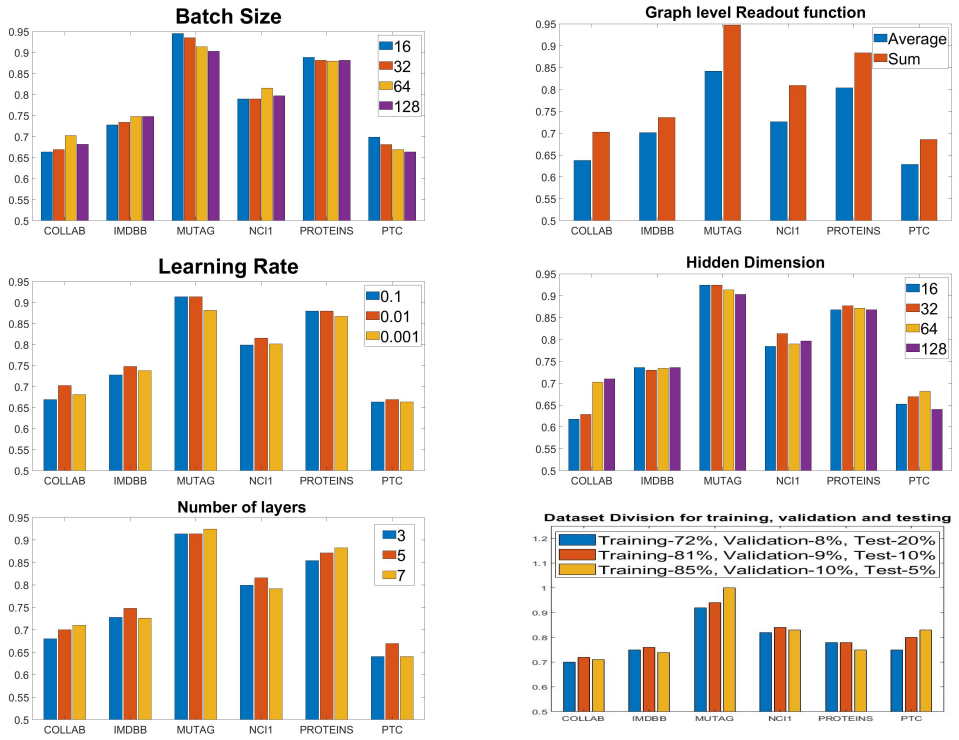


Figure 2.9: Empirical study on hyper-parameter setting. X-axis and Y-axis show the name of the dataset and the accuracy of training respectively [2].

other hand, for the social network datasets, hidden dimension 128 captures the structures efficiently.

Effect of Graph-level Readout Function We have experimented with two different Graph-level Readout functions, Sum and $Average$. As shown in the figure, Sum performs much better than the function $Average$, which establishes the fact that Sum has better representational capacity than $Average$.

Effect of Number of GNN layers We have tried our model with three different numbers of GNN layers (including the input layer), such as $\{3, 5, 7\}$. As we can see from the output, using 5 GNN layers in our model gives a better performance than others.

Effect of Division of Datasets We have divided the datasets in three ways, as shown in the figure, and found out that using $81\%, 9\%$ and 10% of the datasets for training, validation, and testing respectively fits best for most of the datasets.

2.6 Conclusion

This chapter describes a new interval aggregation function and compares its discriminative power between different intervals with that of the existing aggregation function. Keeping this newly developed function as an AGGREGATE function, a GNN-based architecture IV-GNN has been developed to deal with those graphs having node features as intervals.

However, in the embedding generation process, IV-GNN fails to capture the flow of information from the higher-order neighborhood of a node without stacking a large number of GNN layers. To deal with this situation, in the next chapter, we aim to develop a more powerful model that will capture the hierarchical structure hidden behind the graph, which will make the flow of information from a distant node easier. We propose GraMMY as a novel framework for hierarchical semantics-driven graph representation learning based on Micro-Macro analysis.

Chapter 3

Graph Representation Learning Based on Micro-Macro Analysis: GraMMy

3.1 Introduction

The aggregation and combination strategies used in the GNN layer by the existing models primarily accommodate only local information from the surroundings of each node. To encode features of the higher-order neighborhood of a node in its node embedding, two strategies can be applied [68]. The first one is to increase the iterations so that the GNN learning process spreads all nodes' information over the entire graph. The second strategy can be stacking more GNN layers. However, both strategies have some practical drawbacks. Increasing iterations will need a large number of training examples to train the model [69], and increasing the GNN layer will lead to a vanishing gradient problem during training [70].

3.1.1 Related Works and their limitations

Though the generalized k-dimensional GNNs (k-GNNs) [37] take into account the higher-order structures by employing k-dimensional WL algorithm (k-WL), these only look into the overall feature information received from the neighbors. Consequently,

similar to the other GNN models, the k-GNNs also ignore several interesting facts, such as through which path the information flows from a specific neighbor. Such information may be crucial for generating better embedding of the target node [71].

In order to encode the information flow within the graph, the existing Random-Walk based methods, such as Node2vec [72] and Struc2vec [73] have their own limitations. Node2vec performs graph classification tasks but fails to maintain the structural equivalence within the graph. Two nodes with neighborhoods that are structurally similar but that are far apart will not have similar latent representations in Node2vec. Capturing the hierarchy within the graph may be one way to address this issue. On the other hand, Struc2vec only considers structural similarity but does not consider any node feature-related information. To explain this with an example, see Figure-3.1. Let there be two triangles, one with three nodes of color yellow and the one with a mix of yellow and green. As struc2vec only considers the structural similarity, the model will ignore the difference in feature of the nodes, i.e., the node color, and as a result, these two triangles will be treated as equal. In a nutshell, we can say that neither node2vec nor struc2vec can consider the feature as well as structural information of the graph simultaneously.

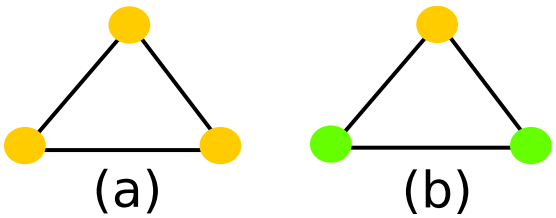


Figure 3.1: Two triangles with similar structures but differently colored nodes [3].

In this chapter, we demonstrate that hierarchically analyzing the semantics behind graph structure can help GNN in better capturing the information flow from both local and higher-order neighbors while eliminating the need for increasing training samples and/or increasing layers in the GNN model.

3.1.2 Research Contribution

Intending to address the above-discussed limitations of the existing models in learning global and local information together from a graph, in this chapter, we describe GraMMy, a novel framework for hierarchical semantics-driven graph representation learning based on Micro-Macro analysis [3]. GraMMy allows a flexible flow of information from higher-order neighborhood and better captures the neighborhood information. The hierarchical study is conducted using Locality Sensitive Hashing (LSH) as a micro-macro scalar, while the semantics-driven analyses are accomplished by employing a recurrent autoencoder-based context modeling scheme. We also theoretically explain how the proposed micro-macro analysis approach maintains a trade-off between information loss and flexibility of information flow while dealing with macro and micro views of the graph structure. Thus, our major contributions are as follows.

- A GNN-based framework, GraMMy, is proposed, which learns from graphs by focusing on its different hierarchical levels through micro-macro analysis of the structure.
- We theoretically investigate the variations of several statistical and network properties with the change in abstraction levels of the graph structure.
- We capture semantics through context generation, which magnifies the flow of the information passing through various nodes of the graph in different abstraction levels.
- We empirically evaluate our model concerning the graph classification task on six benchmark datasets.

3.2 Proposed Framework: GraMMy

An overview of the proposed framework GraMMy is shown in Fig. 3.2. As depicted in the figure, the framework is comprised of three key modules engaged in *micro-macro analyses* of the graph structure, *semantic modeling* of the graph nodes, and

node embedding through flat message passing, respectively. Each module, along with the relevant theoretical background, is discussed in subsequent subsections.

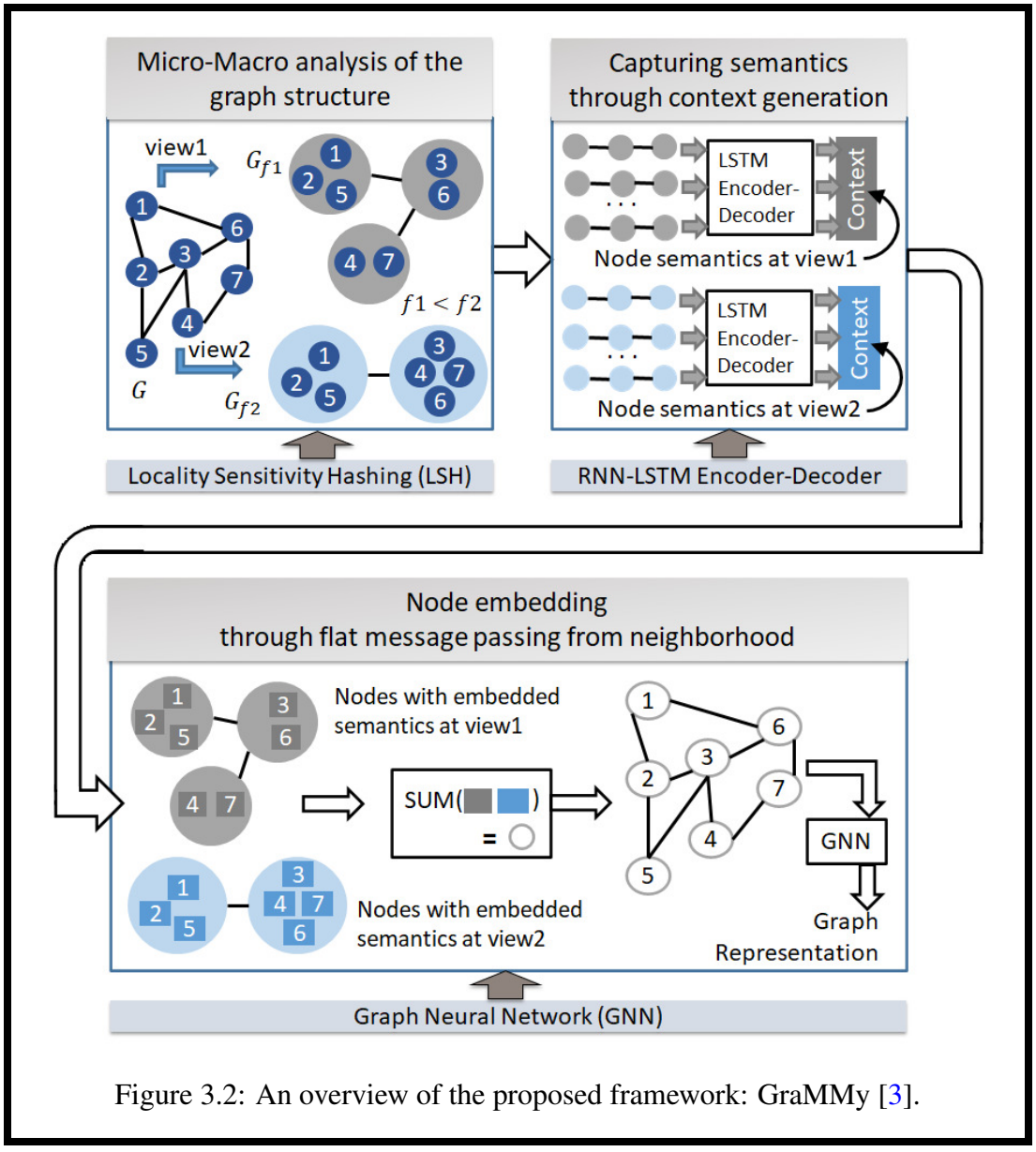


Figure 3.2: An overview of the proposed framework: GraMMy [3].

3.2.1 Micro-Macro Analysis of the Graph Structure

The Micro-Macro analysis follows a similar notion of human vision. For example, to understand the complete structure of a house, it is necessary to look into the bricks as well as the formation/organization of the bricks to construct the whole building. Sim-

ilarly, to view a graph from various levels of observability, we consider a framework based on Micro-Macro analysis [74]. It sheds light on the graph with respect to the level of abstraction, i.e., with a certain level of detailing. The closer a graph is to the "micro" level, the more precisely it will capture the connectivity of each node. Conversely, the more we go towards the "macro" level of the graph, the more cumulative behaviors of the nodes it will capture.

Now, we formally define the notion of Micro-Macro analysis of a graph. μ is said to be a Micro-Macro scalar that takes a graph G and an amount of detail f as inputs and outputs another graph G'_f and an epimorphism Φ_f such that,

- if $f = 1$ then $\mu(G, f) = (G'_1, \Phi_1)$ with $|G'_1|_V = 1$.
- if $f < f'$ then $|G'_f|_V \geq |G'_{f'}|_V$.
- if $f < f'$ then $|G'_f|_E \geq |G'_{f'}|_E$.

Where, $|G|_V$ = The number of nodes in the graph G, $|G|_E$ = The number of edges in the graph G and f is a real number in the range $(0, 1]$. That is, for any graph G and level of abstraction f, we receive a modified version of G, G'_f and an epimorphism Φ_f such that Φ_f will decide correspondence between G and the new graph G'_f. A smaller value of f corresponds to the micro view of the graph, and with $f = 1$, the whole graph will be seen as a single node graph with the maximum macro view possible.

To be noted, as per [74], f is allowed to take any value from the interval $[0, 1]$. However, we alter this definition a little since there may be several cases where two or more nodes in a graph have exactly the same set of neighbors. For example, in the Fig. 3.3(a), the node 1 and 4 have neighbor set as $\{2, 3\}$. Then with $f = 0$, the nodes $\{1, 4\}$ and $\{2, 3\}$ will be clubbed together and will end up looking like Fig. 3.3(b), which needs to be exempted.

Given a micro-macro scalar and a set of *abstraction levels* or *detailing levels* $\{f_1, f_2, ..., f_k\}$, the first module of our framework generates a spectrum of different abstract views of the graph G, $\{G'_{f_1}, G'_{f_2}, ..., G'_{f_k}\}$. Each of these is exploited in subsequent

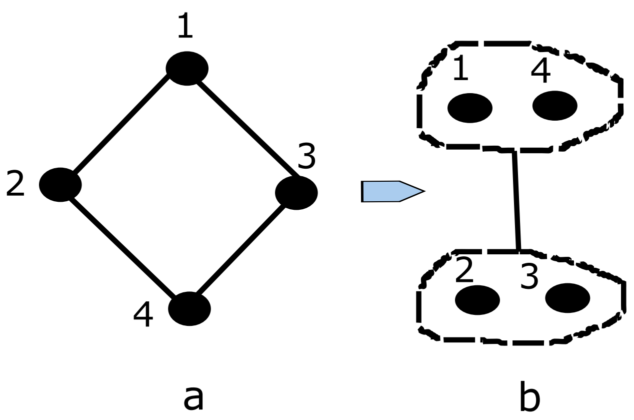


Figure 3.3: Micro-Macro analysis of a graph [3].

modules of GraMMy to describe the behaviour of the network from different levels of detail.

In the form of a Micro-Macro scalar, we have chosen Locality Sensitive Hashing (LSH), denoted by μ^L henceforth. In LSH, similar items are hashed to the same bucket with high probability. In order to use LSH as a micro-macro scalar, we define the following rule: With a given graph G and a level $f \in (0,1]$, whether two nodes should be grouped or not is decided based on the Jaccard Similarity coefficient, which is a commonly used indicator of the similarity between two sets[75]. Now, we explain how we have used Jaccard Similarity in order to perform this task. In case the node labels n are known, for any two nodes u and v, we consider the sets as follows,

$$\begin{aligned} \mathcal{L}_u &= \{n_w \mid w \in \mathcal{N}(u)\}, \text{ and} \\ \mathcal{L}_v &= \{n_w \mid w \in \mathcal{N}(v)\}, \text{ where } \mathcal{N}(u) \text{ and } \mathcal{N}(v) \text{ are neighboring nodes of } u \text{ and } v \\ &\text{, respectively.} \end{aligned} \tag{3.2.1}$$

Let $J(A, B)$ be the Jaccard Similarity between two sets A and B. For a given $f \in (0,1]$, if $J(\mathcal{L}_u, \mathcal{L}_v) \geq (1-f)$, then u and v will be clubbed together in G'_f, otherwise not.

On the other hand, when no node labels are present, we construct the sets $\mathcal{L}_u$ and $\mathcal{L}_v$ using the degree d as follows,

$$\begin{aligned} \mathcal{L}_u &= \{d_w \mid w \in \mathcal{N}(u)\}, \text{ and} \\ \mathcal{L}_v &= \{d_w \mid w \in \mathcal{N}(v)\} \end{aligned}, \tag{3.2.2}$$

and the remaining procedure is the same.

When there are additional feature-related information available other than node labels, we calculate the Jaccard Similarity between the sets as follows,

$$\begin{aligned} \mathcal{L}_u^{f_i} &= \{f_i(w) \mid w \in \mathcal{N}(u)\}, \\ \mathcal{L}_v^{f_i} &= \{f_i(w) \mid w \in \mathcal{N}(v)\}, \end{aligned} \tag{3.2.3}$$

where $f_i(w)$ is the i-th feature value of the node w. Then, $J(\mathcal{L}_u, \mathcal{L}_v) = \min_i\{J(\mathcal{L}_u^{f_i}, \mathcal{L}_u^{f_i})\}$.

LSH based on the Jaccard Similarity measure in the above-mentioned way generates a modified graph structure, considering each cluster to be a node at a higher level of abstraction. We also discard the original edges, which completely lie within a cluster. We denote the correspondence between two graphs G and G'_f as Φ_f^L. Below, we show that Φ_f^L is an epimorphism and μ^L satisfies three properties as mentioned in the micro-macro scalar definition.

Proposition 5. *Φ_f^L is an epimorphism.*

To show, Φ_f^L is an epimorphism, we have to show that, $\forall v \in G, \Phi_f^L(v) \in G'_f$. According to the definition of μ^L, for a detailing level $f \in (0, 1]$, a node v in G will be mapped to the cluster of nodes having atleast $(1 - f)$ Jaccard Similarity, as defined previously. Hence, every node in G has an image in G'_f, proves the fact that, Φ_f^L is an epimorphism.

Proposition 6. *If $f = 1$ then $\mu^L(G, f) = (G'_1, \Phi_1)$ with $|G'_1|_V = 1$.*

When $f = 1$, two nodes will be clustered together if they have at least $(1-1)\times 100\% = 0\%$ similarity in the sets of neighbors. Essentially, the whole graph will be clustered to

one node with no edge.

Proposition 7. *If* $f < f'$ *then* $|G'_f|_V \geq |G'_{f'}|_V$.

For a given f, nodes having at least $(1 - f)$ Jaccard Similarity, as defined earlier, will be clustered and let c_f many clusters are generated, i.e., $|G'_f|_V = c_f$.
Similarly, for f, $c_{f'}$ many clusters are generated, i.e., $|G'_{f'}|_V = c_{f'}$.
Now, we have the following relations, $f \sim \frac{1}{c_f}$, as the increase in the level of detail will decrease the number of clusters. Therefore, $f \leq f' \implies c_f \geq c_{f'}$, equality occurs as the nodes having atleast $(1 - f')$ Jaccard Similarity may have $(1 - f)$ Similarity also. Hence, $|G'_f|_V \geq |G'_{f'}|_V$.

Proposition 8. *If* $f < f'$ *then* $|G'_f|_E \geq |G'_{f'}|_E$.

From the Proposition 3, we have $|G'_f|_V \geq |G'_{f'}|_V$. As we are left with only inter-cluster edges and discarding the intra-cluster edges, the greater number of clusters will lead to a greater number of edges in the resulting graph. Hence, $|G'_f|_E \geq |G'_{f'}|_E$.

3.2.1.1 Significance of Micro-Macro Analysis

Our proposed Micro-Macro analysis is primarily driven by the need to maintain a trade-off between *information propagation* and *information loss*. The higher the detailing value (f), the larger the information loss. On the other hand, a higher degree of abstraction (Macro view) ensures more flexible propagation of information from distant nodes in a graph, compared to its lower level of abstraction (Micro view).

Information propagation flexibility vs. Micro-Macro analysis. Typically, the flexibility of *information propagation* can be attributed by *influence score* [54] and *characteristic path length* [76]. The influence score reflects the effect of a node v on another node u by calculating the amount of change in the representation of u with respect to the changes in the input feature of v. Formally, the influence score is defined as follows [54].

Definition 8. *Let $G = (V, E)$ be a simple graph. h_v be the input feature of node v and h'_u be the modified feature of node u. Then, the influence score $I(u, v)$ of u by v is defined by the sum of the entries in the Jacobian matrix* $\left[\frac{\delta h'_u}{\delta h_v}\right]$.

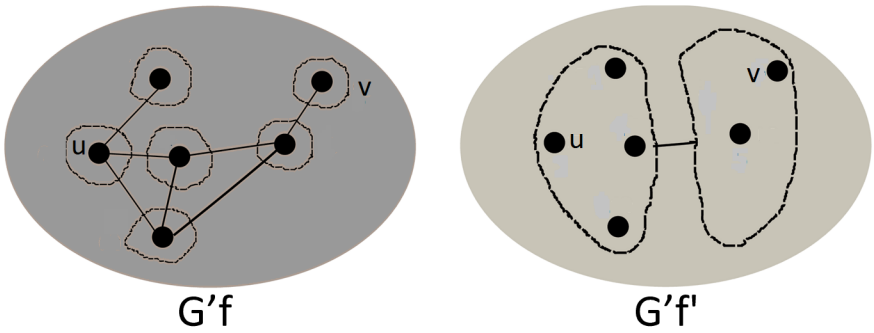


Figure 3.4: Impact of the amount of detail in Influence Score [3].

As we have already mentioned, for the newly constructed graph G' from the original graph G at a particular micro-macro level, we treat a cluster of nodes as a node. It is to be noted that, in order to generate feature vector h'_u of a node u in a different level of abstraction, we select a node with maximal degree from the cluster to treat as a representative of the cluster. We give preference to the node with a higher degree because we believe that a node with greater connectivity will be more informative about the cluster of the nodes (influenced by the concept of degree centrality in Social Network [77]). After that, we generate the context for the representative node using Algorithm 2 and share this generated context among the other nodes in the cluster.
In order to find the influence score $I(u, v)$ of u by v, applying chain rule, we get:

$$\left[\frac{\delta h'_u}{\delta h_v}\right] = \sum_{p=1}^{\mathcal{P}} \left[\frac{\delta h'_u}{\delta h_v}\right]_p . \tag{3.2.4}$$

where, $\mathcal{P}$ is the number of distinct paths from v to u through which changes in feature vector of v will affect u. It is assumed that the information that reaches a particular node will be equally distributed among its neighbors, and hence, it can be normalized by multiplying with the inverse of the degree (deg) of that node. Hence, we can express

the influence score as follows.

$$\left[\frac{\delta h'_u}{\delta h_v}\right] = \sum_{p=1}^{\mathcal{P}} \prod_{l=k_p}^{1} \frac{1}{deg(v_{l-1}^p)} \left[\frac{\delta h_{v_{l-1}^p}}{\delta h_{v_l^p}}\right]. \tag{3.2.5}$$

where k_p is the length of the p-th path and v_l^p is the l-th node in the p-th path starting from v and ending at u. It is evident here that the influence score value decreases exponentially with the increasing value of k_p. Since, for any u and v, the k_p at the macro-level view would be lesser than that in the micro-level view of a graph (for example, refer Fig. 3.4), we can arrive at the following proposition.

Proposition 9. *If $f < f'$, the influence score of any node u by v, $I(u, v)$ in G'_f is greater than or equal to $I(u, v)$ in $G'_{f'}$.*

For example, consider the two abstract views G'_f and $G'_{f'}$ of a graph G, where $f < f'$, as depicted in Figure 3.4. Accordingly, the calculation of the influence score of node 6 on node 1 in G'_f and $G'_{f'}$ can be done as follows: As per the figure, we have 4 paths from node 6 to node 1 in G'_f. Let the change in the feature vector of node 6 be δh_6. Then the flow of change from 6 to node 1 is as shown below,

- $1 \xleftarrow{\left[\frac{1}{81}\delta h_6\right]} 2 \xleftarrow{\left[\frac{1}{27}\delta h_6\right]} 3 \xleftarrow{\left[\frac{1}{9}\delta h_6\right]} 4 \xleftarrow{\left[\frac{1}{3}\delta h_6\right]} 5 \xleftarrow{[\delta h_6]} 6.$
- $1 \xleftarrow{\left[\frac{1}{27}\delta h_6\right]} 2 \xleftarrow{\left[\frac{1}{9}\delta h_6\right]} 3 \xleftarrow{\left[\frac{1}{3}\delta h_6\right]} 5 \xleftarrow{[\delta h_6]} 6.$
- $1 \xleftarrow{\left[\frac{1}{81}\delta h_6\right]} 2 \xleftarrow{\left[\frac{1}{27}\delta h_6\right]} 4 \xleftarrow{\left[\frac{1}{9}\delta h_6\right]} 3 \xleftarrow{\left[\frac{1}{3}\delta h_6\right]} 5 \xleftarrow{[\delta h_6]} 6.$
- $1 \xleftarrow{\left[\frac{1}{27}\delta h_6\right]} 2 \xleftarrow{\left[\frac{1}{9}\delta h_6\right]} 4 \xleftarrow{\left[\frac{1}{3}\delta h_6\right]} 5 \xleftarrow{[\delta h_6]} 6.$

The amount written over the arrows is the fraction of the information that reaches the next node in the path. Hence, for G'_f, we have

$$\left[\frac{\delta h'_u}{\delta h_v}\right] = \sum_{p=1}^{\mathcal{P}} \left[\frac{\delta h'_u}{\delta h_v}\right]_p = (\frac{1}{81} + \frac{1}{27} + \frac{1}{81} + \frac{1}{27})\delta h_6 = \frac{8}{81}\delta h_6,$$

whereas, for $G'_{f'}$, we achieve the following influence score.

$$\left[\frac{\delta h'_u}{\delta h_v}\right] = \delta h_6.$$

This supports our Proposition 9.

As mentioned earlier, the flexibility of influence propagation is also attributed to the characteristic path length, which measures how two vertices in a graph are separated. Formally, the characteristic path length can be defined as follows [76].

Definition 9. *Let $G = (V, E)$ be a simple graph, where $n = |V|$ is the number of nodes in the graph and $dist_{ij}$ is the smallest path length between any pair of nodes $v_i \in V$ and $v_j \in V$. Then, the characteristic path length of the graph becomes $L(G) = \frac{1}{n(n-1)} \sum_{v_i \neq v_j \in V} dist_{ij}$.*

Since, in the clustered graph two distinct nodes can be thought of as a single node based on their similarity and the level of abstraction provided, we define $dist_{ij} = 0$ in G'_f if $v_i \neq v_j \in G$ but $v_i = v_j \in G'_f$. Intuitively, we can say that a smaller characteristic path length assures a better flow of information within the graph. Our claim is also supported by the definition of influence score in Eq. 3.2.5. Eventually, we came up with the following proposition that shows how the characteristic path length is related to the amount of detail provided for the micro-macro scalar.

Proposition 10. *If $f < f'$ then $L(G'_f) \geq L(G'_{f'})$.*

As the whole set of nodes of the graph G is present in both G'_f and $G'_{f'}$ in different clustered form, it is enough to show that

$$\sum_{v_i \neq v_j \in G'_f} dist_{ij} \geq \sum_{v_i \neq v_j \in G'_{f'}} dist_{ij}.$$

From Proposition 3 and Proposition 4, we have $|G'_f|_V \geq |G'_{f'}|_V$ and $|G'_f|_E \geq |G'_{f'}|_E$. As shown in Fig. 3.5, smaller f' will result in more nodes collapsing into a single cluster. Therefore, reaching from one node to another via the shortest path will take

less number of hops through different clusters. Hence proved.

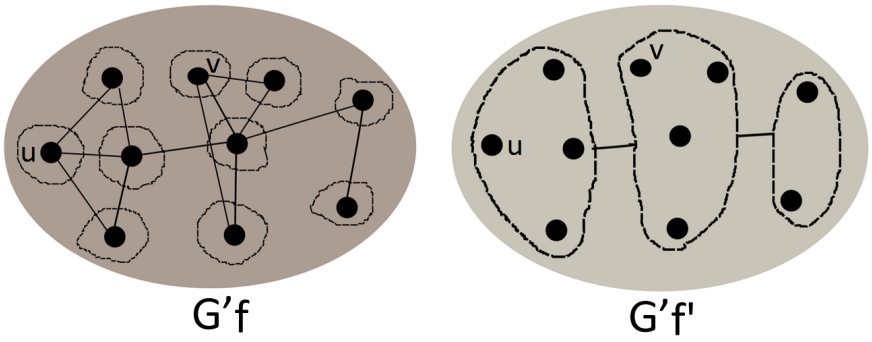


Figure 3.5: Impact of the amount of detailing in the Characteristic path length of a graph. Two graphs shown in the picture are G'_f (left) and $G'_{f'}$ (right), where $f < f'$. In G'_f, let us assume that each node cluster is a singleton node. Here, to reach v from u, the shortest path is of length 3. Whereas in $G'_{f'}$, an increase in abstraction level will allow more nodes to collide and make only 3 distinct clusters. Hence, the node v will be just 1 hop away from u [3].

Information Loss vs. Micro-Macro analysis. The basic idea behind the micro-macro level analysis is to place nodes with certain levels of similarities into the same buckets or clusters. As mentioned earlier, by similar nodes, we mean that the distance between the feature vectors of two nodes, as calculated based on Jaccard Similarity, is less than a specific threshold value. For a particular LSH function μ^L, where $\mu^L(G, f) = G'_f$, a collision occurs when two different nodes are mapped to the same cluster in G'_f, which, eventually, leads to loss of information. Accordingly, we can define the information loss for a micro-macro scalar μ^L_f as the sum of the divergence or distance between the feature vectors of the nodes in the original graph (G) and that in the resultant graph (G'_f). That is,

$$IL(G, G'_f) = \sum_{v \in G} Distance(h_v, h'_v) \tag{3.2.6}$$

where, h_v is the feature vector of v in G and h'_v is the feature vector of v in G'_f. Hence, the larger the divergence between the original and modified feature vectors of the nodes in the two graphs, the more the amount of information loss.

As a consequence, we arrive at the following proposition that indicates that the higher the level of abstraction (f), the larger the information loss (IL).

Proposition 11. *If $f < f'$, then $IL(G, G'_f) \leq IL(G, G'_{f'})$.*

$$
\begin{aligned}
IL(G, G'_f) &= \sum_{v \in G} Distance(h_v, h'_v) \\
&\propto -|G'_f|_V, \text{ as more number of nodes in } G'_f \text{ implies} \\
&\text{more nodes' feature remains unchanged after hashing.} \\
&\propto -(1-f), \text{ as threshold of similarity, } (1-f), \\
&\text{is proportional to } |G'_f|_V. \\
&\propto f
\end{aligned}
$$

Therefore, if $f < f'$, then $IL(G, G'_f) \leq IL(G, G'_{f'})$

Trade off between Information propagation flexibility and Information Loss.

From the above-discussed Propositions 10 and 11, we have seen that, with an increasing value of f, the influence score of a node on another node increases. On the other hand, with a larger value of f, two nodes with Jaccard Similarity more than the threshold value ($(1-f)$) are treated as the same node, although they are not exactly equal. Therefore, as a result of oversimplification of the graph, the information loss increases. The motivation behind doing micro-macro analysis on a graph is to capture every possible information at different levels of abstraction. For example, as shown in Fig. 3.6, node 1 is at a 5 hop distance from node 8. To convey the information about node 1 to node 8 using the original graph will need the GNN algorithm to run for a relatively large number of epochs (in this case, the algorithm needs to run for 5 epochs), which is not computationally feasible and also, it might cause vanishing gradient problem in such a small graph. Therefore, with a larger value of f, when the graph appears as a graph of its most prominent clusters, the influence propagation within the graph is

much easier. In this example, one can reach from node 1 to node 8 using only 1 hop in the macro-view of the graph (refer Fig. 3.6(b)). On the other hand, the reduction of a graph with 8 nodes to a graph with 4 nodes leads to information loss as finer details of the individual nodes are being neglected at the macro level.

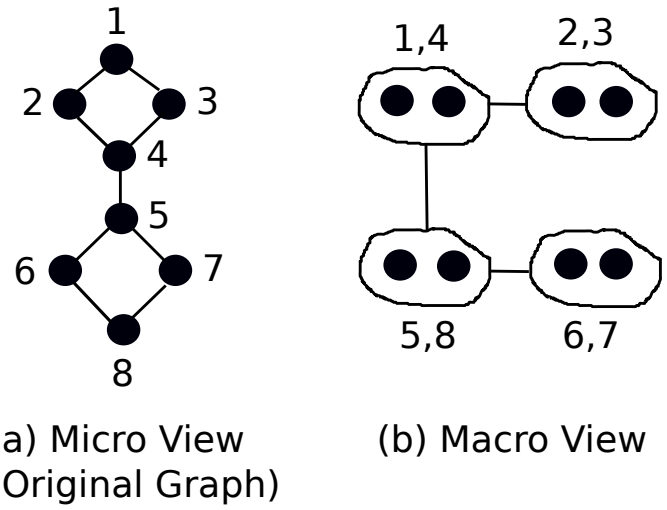


Figure 3.6: Information propagation flexibility vs Information Loss. (a) The graph on the left side is the original graph, a micro view of that graph, which focuses on the local information of every node. (b) The graph on the right-hand side is a more macro view of the graph, where the graph is seen in a more clustered manner based on the similarity of the node's neighborhood set [3].

3.2.2 Capturing Semantics through node Context Generation from Different Levels of Abstraction

This module of GraMMy aims to capture the flow of information from various local or higher-order neighbors to a target node, considering the different abstract views of the graph as obtained through micro-macro analyses in the previous module. In order to account for the semantics of information propagation through various nodes within each information flow path, we propose a novel approach of *node context generation* by employing an Encoder-Decoder model based on a recurrent neural network with long short-term memory (LSTM). To the best of our knowledge, this is the first work

in GNN literature that attempts to explicitly capture the impact of the information flow path in the graph representation learning process.

Typically, we generate the context of a node at a particular view G_f of the graph G, using all k-length node sequences ending at that node. We use a pre-trained LSTM Sequence-to-Sequence Encoder-Decoder model (primarily an LSTM auto-encoder model), which takes every k-length sequence as an input to the encoder and generates a fixed-length context vector. We have trained the LSTM auto-encoder with $10,000$ sequences with sequence lengths ranging from 2 to 100. This context vector is treated as the semantic representation of the information flow path, consisting of the considered node sequence. To ensure that the node will encode the details from every precision level, we repeat this process for every level of abstraction. Finally, the sum of all these has been taken to obtain the overall context of the node, which is utilized in the subsequent module of GraMMy to generate the graph representation in low dimensional space. Note that our objective is not to find out the optimum f, which will capture the maximum information about the network, but to study the network from different micro/macro views. Hence, by using sum, we primarily ensure that the context from every view gets equal importance. Further, as per the existing research findings, the other functions, like $mean$, max, or min, have less discriminative power than sum [36]. The max and the min may be useful for identifying representative elements. The $mean$ function may capture the distributional information. However, to accumulate information from different detailing views with equal importance, we find sum to be the most appropriate function. Algorithm 2 formally presents the proposed method of node context generation. The final context for a node u is generated using the following equation and treated as the feature value x_u for the node u for the rest of the model

$$x_u = \sum_{f \in \{f_1, f_2, \ldots, f_k\}} C_{u_f}. \tag{3.2.7}$$

Algorithm 2 Context generation for a node u through sequence learning in the graph G'_f from a particular level of abstraction f

Input : Node u; neighborhood $\mathcal{N}(u)$ in G'_f; depth k;
a pre-trained Encoder-Decoder Model $\mathcal{M} = [Enc, Dec]$
Output : Context C_{u_f}

Function: SequenceGeneration(v,l)

if $l > 1$ **then**
  for $w \in \mathcal{N}(v)$ **do**
    $seq = SequenceGeneration(w, l-1)$
  end for
  for $s \in seq$ **do**
    $s = CONCAT(s, w)$
    $Finalsequence = Finalsequence \cup \{s\}$
  end for
else
  $Finalsequence = \{v\}$
end if
return $Finalsequence$

for $v \in \mathcal{N}(u)$ **do**
  $sequenceset = SequenceGeneration(v, k-1)$
end for
for $sequence \in sequenceset$ **do**
  $sequence = sequence \cup \{u\}$; $C_{u_f} = C_{u_f} + Enc(sequence)$
end for

3.2.2.1 Significance of Context Generation

It may be noted here that, in existing GNN models, the feature vector of any node initially contains information about merely the node itself. Contrarily, in our approach, the feature vector of a node not only embeds information about $O(n^k)$ nodes, that are present in any k-length path coming towards this node, but also the sequential behaviour within the path. Since the context generation is performed using a pre-trained LSTM encoder, this ultimately becomes a constant time operation. However, the pre-training of the LSTM encoder has a computational complexity of $O(W)$, where W is the number of parameters involved in the LSTM model.

3.2.3 Information Capturing of neighborhood using Flat Message Passing

This module of GraMMy finally generates the node embedding in lower dimensional space by employing GNN message passing scheme. As mentioned earlier, rather than using the original feature value of the nodes, we use the context-aware feature vector of the nodes as it is already rich with knowledge about local as well as higher order neighborhood, gained from both micro and macro views of the original graph. The overall process is formally presented as Algorithm 1 in the Introduction chapter. As our aim is to perform a graph classification task, we generate embedding h_G for the graph G from the node embeddings h_u^k for all $u \in G$ from depth $k = 0, 1, ..., K$. Finally using a MLP, the class of the graph is predicted from the embedding of the graph.

$$h_G = CONCAT(\sum_{u \in G} h_u^k \mid k = 0, 1, ..., K) \tag{3.2.8}$$

Here, $h_u^K = z_u$, $h_u^0 = x_u$ and $x_u = \sum_{f \in \{f_1, f_2, ..., f_k\}} C_{u_f}$.

3.3 Experimental Results

We empirically validate our theoretical findings and evaluate GraMMy in comparison with state-of-the-art GNN models.

3.3.1 Datasets

Experimentation is carried out using *four* bio-informatic datasets, namely MUTAG, PROTEINS, PTC and NCI1, and *two* social datasets, namely IMDB-BINARY and COLLAB [57]. The details of the datasets are provided in the Sec 2.5.1.

3.3.2 Experimental Settings

Since the increase in path length for context generation can be computationally expensive, we consider a 3-length sequence for context generation. We choose $f \in \{0.1, 0.2, 0.3,$

$0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1\}$ to generate the spectrum of graphs with different levels of abstraction. We sum all the context-aware features of a node extracted from different micro and macro views of the graph. This gives a semantically enriched feature representation of the node, which is subsequently utilized by the core GNN model to accomplish the graph classification task. For the GNN model, as used in GraMMy to perform graph representation learning, we adopt the parameter settings recommended for GIN [36] that achieved the best performance reported in the literature. Specifically, we use 5-layers of each GNN block where every MLP has 2-layers excluding the input layer. Each hidden layer contains 64 hidden units and uses batch normalization. We employ 2-layer MLP [48, 78] as the UPDATE function and Sum as the READOUT function. The number of epochs is set as 350 and the input batch size of training is considered as 32. We use Adam optimizer [64] with the initial learning rate 0.01. The learning rate is decayed by 0.5 after every 50 epochs. The final layer dropout is set as 0.5 [79]. We perform 10-fold cross-validation with LIB-SVM [57, 80].

3.3.3 Baselines Models

The performance of GraMMy is compared with those of *six* state-of-the-art GNN models, namely GraphSage [35], GCN [9], GIN [36], k-GNN [37], g-U-Nets [38] and PNA [39] as briefly discussed in the Sec 1.3.2.2 of Chapter 1. As a representative of k-GNN, we use the hierarchical variant 1-2-3-GNN. The hyper-parameter settings for all these baselines are kept the same as that of our proposed framework.

Table 3.1: Test set classification accuracies

Models	*MUTAG*	*PTC*	*PROTEINS*	*NCI1*	*IMDB-B*	*COLLAB*
GraMMy	0.91862± 0.055	0.63889 ± 0.091	0.78571± 0.059	0.82500 ± 0.022	0.75520 ± 0.054	0.72459 ± 0.061
k-GNN	0.89528 ± 0.046	0.62812 ± 0.054	0.76438 ± 0.040	0.81738 ± 0.023	0.74755 ± 0.106	0.72131 ± 0.092
g-U-Nets	0.84211 ± 0.113	0.6484 ± 0.132	0.76786 ± 0.200	0.71776 ± 0.191	0.746 ± 0.148	0.718 ± 0.108
PNA	0.84043 ± 0.197	0.60465 ± 0.162	0.70916 ± 0.114	0.62336 ± 0.120	0.730 ± 0.158	0.715 ± 0.115
GIN	0.88421± 0.073	0.62209 ± 0.039	0.76840 ± 0.021	0.79913± 0.029	0.74482 ± 0.155	0.71851 ± 0.081
GCN	0.85645± 0.186	0.62391 ± 0.086	0.76062 ± 0.075	0.78245 ± 0.043	0.73653 ± 0.018	0.70402± 0.155
GraphSAGE	0.82314± 0.018	0.59737± 0.024	0.72968 ± 0.108	0.77512 ± 0.092	0.72177 ± 0.047	-

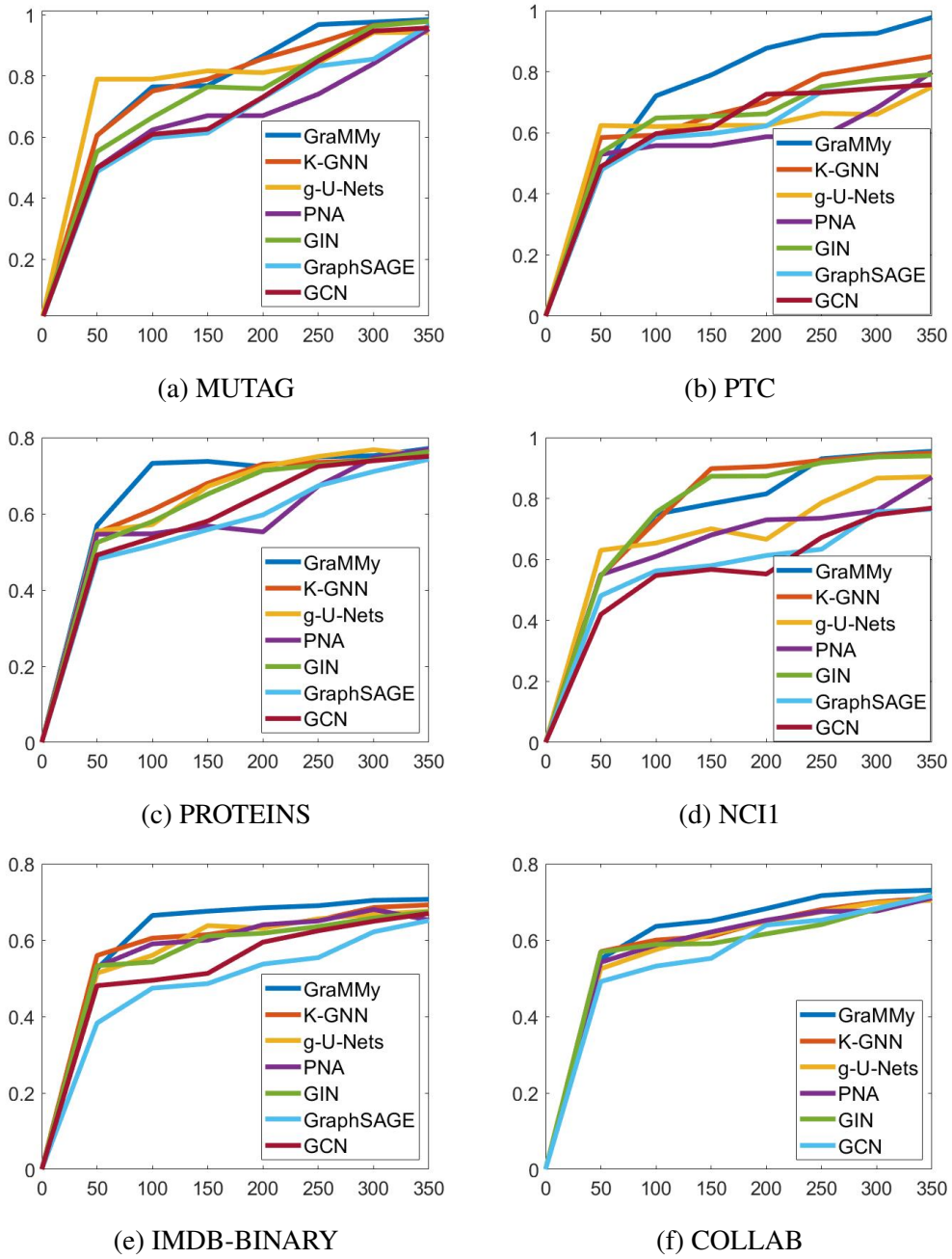
3.3.4 Results and Discussions

The results of the experimental study are summarized in Table 3.1 and also depicted in Figs. 3.7-3.10.

3.3.4.1 Graph Classification Performance

In order to evaluate the performance of our model, we compare both training and test accuracies of GraMMy with that of the state-of-the-art algorithms. It is evident from Fig. 3.7 that, for all the datasets, GraMMy achieves better training performance compared to the considered baselines. Further, note that, although we have run our experiment for 350 epochs for each model, to give chance to converge, GraMMy convergences to its optimal training accuracy within 250 epochs in most of the datasets. As in general, GNN algorithms are generally computationally expensive, faster convergence is always desired. Our proposed framework has better knowledge gaining capability with faster convergence while being trained under the same experimental set-up.

Similar competitive performance of GraMMy can be noticed when compared with respect to test accuracies also (refer Table 3.1). We have performed the experiments 10 times and reported the mean and standard deviation of the accuracy for each model. It may be noticed that our proposed scheme of viewing the graph from different levels of abstraction and utilization of contextual information about the nodes helps GraMMy to always attain the best performance in graph classification. (We could not execute GraphSAGE on COLLAB due to memory constraints). Although PNA uses four aggregation functions in order to increase the expressiveness of the model, it can not outperform our model GraMMy because it is designed in such a way that it relies on feature-related information of the nodes of the graph. The datasets that we have used for validation of our model are rich in structural information rather than feature-related information. Therefore, the hierarchical semantics-driven approach used in our model turns out to be more useful in this case.



(a) MUTAG (b) PTC

(c) PROTEINS (d) NCI1

(e) IMDB-BINARY (f) COLLAB

Figure 3.7: Comparison of Training set performance of GraMMy with the state-of-the-art models on several benchmark datasets. The X-axis and Y-axis denote the number of iterations and training accuracy, respectively [3].

3.3.4.2 Validation of Theoretical Findings

In order to empirically validate our theoretical findings, we have applied LSH-based micro-macro scalar Φ_f^L on the aforesaid datasets and visualized the effect of the amount

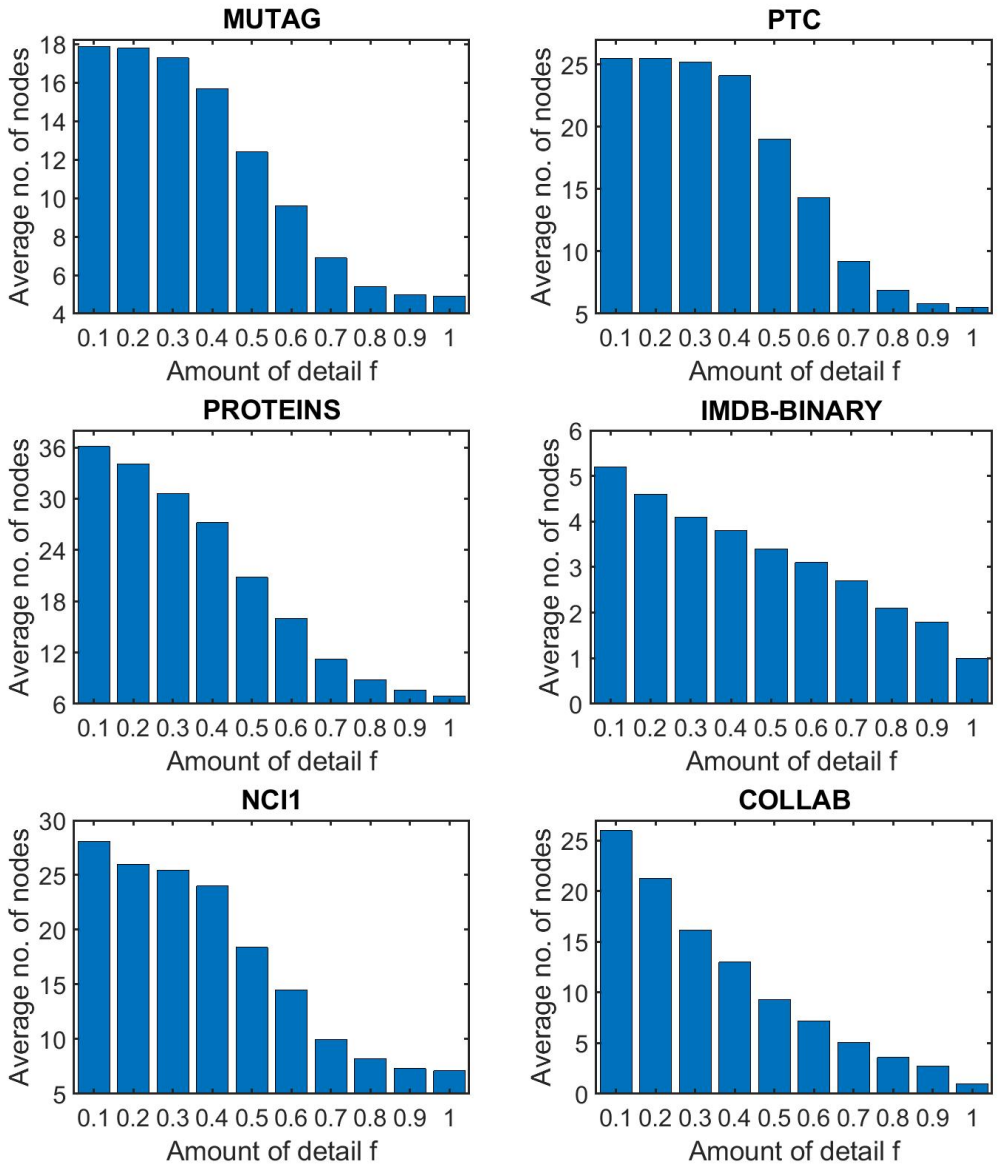


Figure 3.8: f vs. Average no. of nodes: The effect of the amount of detail f on an Average number of nodes [3].

of detailing (f) on the graph properties. Fig. 3.8 shows the changes in *average number of nodes* and Fig. 3.9 shows the changes in *average characteristic path length* with respect to the change in f. Further, we have depicted the change in *information loss* with respect to the change in f in Fig. 3.10. As expected, both the average number of nodes and average characteristic path length decrease with the increasing value of

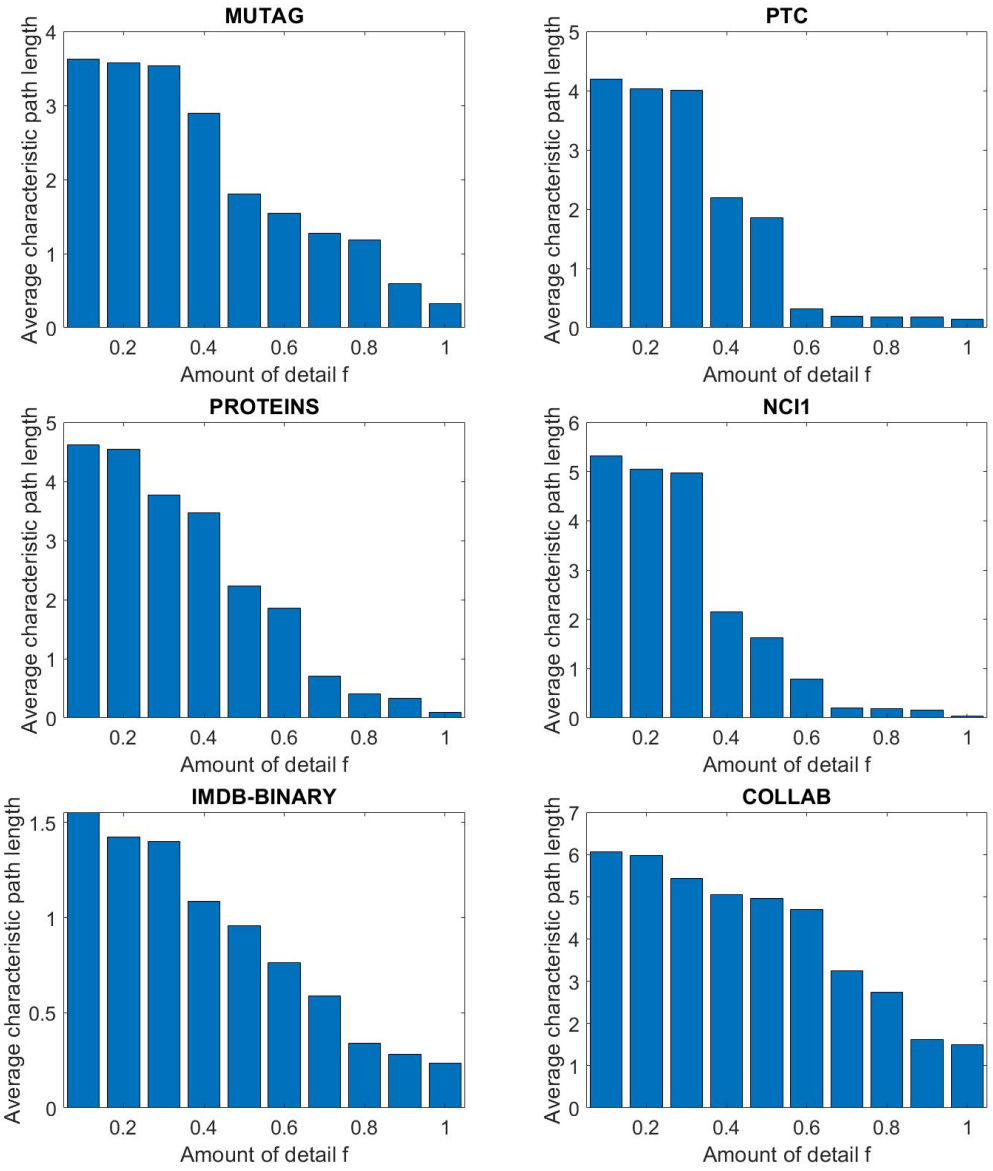


Figure 3.9: Effect of the amount of detail (f) on Average characteristic path length [3].

f. Moreover, Fig. 3.10 shows that the information loss increases as the value of f increases, and this is consistent with our Proposition 11. Further, we see that, for IMDB-BINARY, the average count of nodes with $f = 0.1$ is 5.2, that is much less than the average node count 19.8 in the original dataset. This means here, the neighbor sets of various nodes in the graphs are not very different. So, even if the threshold of

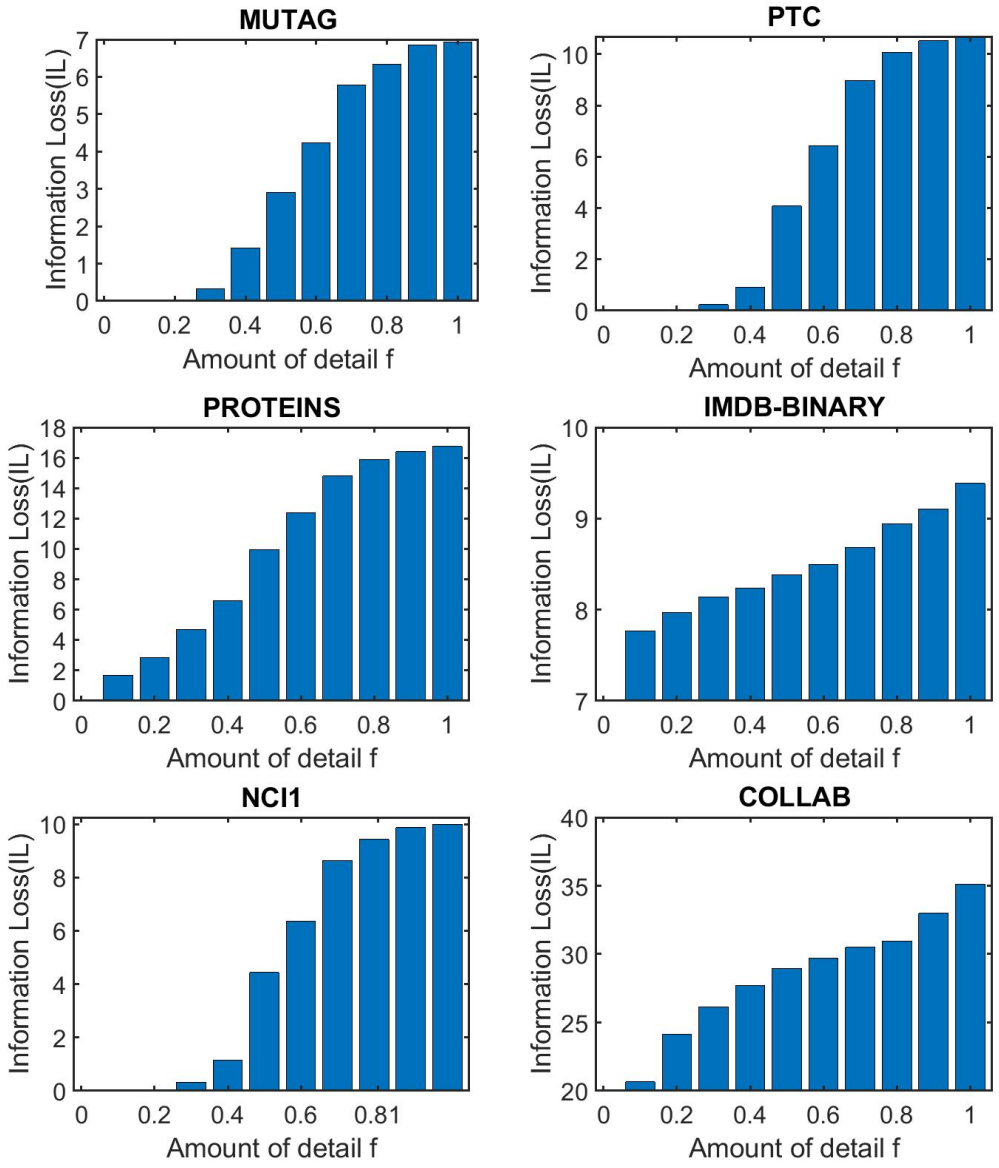


Figure 3.10: f vs. IL :The effect of amount of detail f on Information Loss (IL) [3].

similarity to be hashed to the same cluster is very high, these can easily be clustered together. As a result, the information loss for this dataset is much higher, even for low values of f. In contrast, the average node count in MUTAG for $f = 0.1$ and $f = 0.2$ are 17.9 and 17.8, respectively, which are almost equal to that in the original dataset (17.9). Hence, the nodes in this dataset are less likely to be clustered together at a

particular graph view.

3.4 Conclusion

This chapter has introduced a novel approach to graph representation learning based on the notion of hierarchical information extraction from higher-order neighborhoods. The idea is inspired by the human vision mechanism, which studies an object from different levels of abstraction. GraMMy also offers a unique way of aggregating neighboring node information in a context-aware fashion. Apart from providing the theoretical foundation, we have demonstrated that our approach outperforms state-of-the-art GNNs on benchmark graph classification tasks.

While the strategies developed across chapters 2 and 3 target to solve the Graph Classification Task on the homogeneous graph, next in chapters 4 and 5, we develop two models for social recommender system where the graph has a heterogeneous structure.

Chapter 4

SInGER: A Recommendation System Based on Social-Influence-aware Graph Embedding Approach

4.1 Introduction

With the growth of the e-commerce system, people can browse, purchase, review, or rate different products online. A customer can choose from a wide range of products, compare the prices on various websites, and read the reviews of the users in the e-commerce system who have already bought those products. From the seller's point of view, launching a product to the e-commerce system essentially opens up the market to a large scale in front of him. However, the explosive growth of products can be an overburden to a person [81]. Moreover, knowing the set of potential customers beforehand will leverage the seller to adopt a proper marketing strategy. Thus, highlighting relevant items to a user has become the most challenging task for a recommender system.

The job of a recommender system is to provide a better experience to the customers by suggesting products relevant to their lives. For example, from the previously bought

products of a customer, an e-commerce system can understand that the customer is interested in buying electronic items. Then recommending electronic items may increase the business of the system. Now, there are numerous electronic items present in the e-commerce system. Here, the rating given by customers who have already bought these and the relationship between the customers are essential.

Recommending relevant items to the customer is about modeling the user's previously rated items, opinions of other customers about that item, the user's connection in social networks, and, lastly, the choices of products of the user's friends. Information propagation in large networks using the User-User Interaction and User-Item Interaction has become an interesting area of research, and many extensive studies have been proposed so far [82, 83]. However, in most cases, while capturing the flow of information, existing models overlook the influence of a user that spreads within a community and has an impact on other users in the community as well.

To tackle this issue, this chapter aims to discover the type of interaction, termed as the influence of a user on another based on the type/category of items. The structural information expressed by explicit friendship in the social interaction graph is not capable enough to describe the strength of the communication and does not explain one important emotional factor, namely the influence of a user within the community [84]. Moreover, people tend to form a group among themselves based on the choice of items and tend to ask opinions about an item before buying from other users within the group only. For example, a person who loves to travel can easily be imagined as a part of a community where other people also travel and share their views on several products required for trekking or traveling. Therefore, it is evident that the information will spread much more rapidly than when shared by any person from another domain, say a musician or a sportsman. The intuition behind this would be the frequent traveler is much more influential on other users regarding travel.

Several studies [85, 86] have been carried out to establish the fact that the most influential person in social media need not be the person with the maximum number of friends. Following this idea, instead of considering the structural connections within

the users, this chapter aims to define the Influence Score of a user based on the category of items. Finally, this category-based influence of each user has been used to develop a recommendation system, SInGER [4], that is based on a Social-Influence-aware Graph Neural Network to predict missing ratings.
The major contributions of our work [4] can be summarized as follows:

- We propose a novel technique to quantify the Influence Score of a user in the social recommendation system by constructing the Category-wise information exchange graph, which approximates the usefulness of the ratings given by the user for the Category of the items rated.
- We incorporate the Category-wise User-Influence Score in a Graph Neural Network model to develop a Social Influence-aware recommendation system, SInGER [4], to perform the missing rating prediction task.
- We examine the usefulness of the Category-wise User-Influence Score in rating prediction task, comparing the performance with the state-of-the-art models on two real-life benchmark datasets, namely Ciao and Epinion.

4.2 Related Works and their limitations

There is a rich literature on the missing rating prediction problem for social recommender systems. We can categorize the various approaches for solving this problem into two principal groups, namely *Matrix Factorization*-Based Recommender System and *Neural Network*-Based Recommender System. (see Fig. 4.1).

4.2.1 Matrix Factorization (MF)-Based Recommender System:

This kind of system solves the rating prediction problem by factorizing the information matrix into smaller matrices since, in general, the data in recommender systems are very sparse. This can incorporate implicit information that is not directly given in the data but can be derived by analyzing the data.

Among the various popular MF-based social recommendation systems, the works of Mnih and Salakhutdinov [87], Ma et al. [88],Jamali and Ester [89], Zhao et al. [90], and Yang et al. [91] are worth mentioning. The social recommendation system proposed by Mnih and Salakhutdinov [87] is based on a probabilistic matrix factorization method (PMF) that models the user preference matrix as a product of two lower-rank matrices, namely user matrix and product matrix, using Gaussian distribution. However, it uses the User-Item graph only. The probability matrix factorization approach has also been exploited in SoRec, a social recommendation system proposed by Ma et al. [88]. SoRec captures the fact that the users on the Web are not independent. A person's behavior also affects his/her friends' behaviors on the Web. To implement this, the probabilistic matrix factorization in SoRec has been performed over both the Social Network Matrix and User-Item Matrix. Ma et al. have proposed two other variants of MF-based social recommendation, namely RSTE [92] and SoReg [93], to learn the implicit/explicit social relations and to capture the taste diversity of users' friends, respectively. RSTE fuses the user's choices and the choices of his/her trusted friends simultaneously, using an ensemble parameter, whereas SoReg employs a regularization-based factorization technique to serve the purpose. The MF-based social recommendation approach (SocialMF), as proposed in [89], is one of the pioneering works that captures the trust propagation to improve the prediction quality for new users and has given very few ratings. In the work of Zhou et al. [90], the MF technique has been utilized to achieve tag-based social recommendations. Their model, termed TTMF, captures topic-specific trust relations, where the term "topic" actually refers to a latent factor, and it is extracted from the tag information. The TrustMF, another recent variant of the social recommendation system proposed in [91], has utilized the MF technique to generate two distinct latent feature vectors corresponding to truster and trustee, respectively. These help in characterizing the behaviors "to trust others" and "to be trusted by others" for each user.

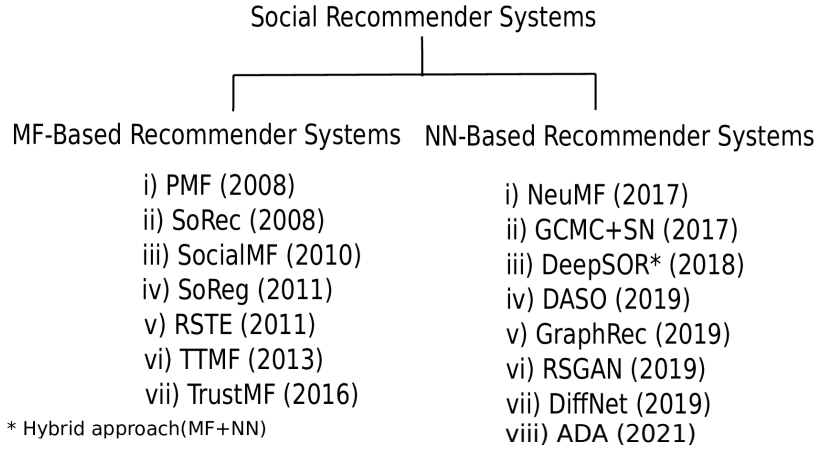


Figure 4.1: Summary of works on Social Recommender Systems.

4.2.2 Neural Network (NN)-Based Recommender Systems:

One of the major limitations of matrix factorization-based recommendation systems is that they are not always capable of estimating low-dimensional vector representations for user-item interaction and social interaction. Increasing dimensions may also lead to the overfitting of these models. To overcome these shortcomings, NN-based approaches have been introduced, which have more generalization ability than MF-based recommender systems. Typically, the NN-based approaches solve the rating prediction problem by employing multi-layer perceptrons (MLPs). Since MLP, followed by a non-linear activation, can approximate any continuous function, it becomes suitable enough for modeling the features of users and items in a recommendation system.

Among the various NN-based approaches for recommendation, NeuMF [94], DeepSoR [95], GCMC+SN [96], DASO [97], GraphRec [98], RSGAN [99], DiffNet [100] and ADA [101] are recent and well-known. Though NeuMF [94] uses a neural network model to learn the latent features of the users and the items, it does not include the social network information of the user to predict the rating. In the DeepSoR method [102], the intrinsic non-linear feature of the user is extracted from its social relation by using a deep neural network, and the final rating prediction is performed based

on the probabilistic matrix factorization technique. DNN-MF[103] method infuses neural networks (DNN) and matrix factorization (MF) with social spider optimization technique to develop a multi-criteria recommender system. Therefore, both DeepSoR and DNN-MF turn out to be hybrid approaches of MF and NN. GCMC+SN [96] uses the Graph-Auto encoder framework, which generates a representation of users and items through a message-passing technique. DASO [97] uses a bidirectional mapping method to transfer users' information between the social domain and item domain using adversarial learning. GraphRec [98] is a state-of-the-art approach to capturing information about the item and user using the GNN framework. This model can capture information about both interaction and rating for a user-item connection. DiffNet [100] uses a layer-wise diffusion method to capture the change of user embedding with respect to the change of its surroundings. RSGAN [99] generates reliable friends who can perfectly predict the current user's preference using GAN.

4.3 The Proposed Framework

This section introduces basic definitions and notations required to develop our model [4]. Subsequently, we present the model architecture and methodological details.

4.3.1 Problem Scenario

Let $G = \{N, E\}$ be a graph, where N is the set of nodes, and E is the set of edges. $N = (U \cup I)$ consists of two types of nodes, namely User and Item, which makes Social Recommender system a heterogeneous graph. $U = \{u_1, u_2, ..., u_n\}$ represents the set of users and $I = \{i_1, i_2, ..., i_m\}$ represents the set of items, where n denotes the number of users and m denotes the number of items. Edges are of two types: Social Interaction between users and User-Item Interaction between users and items (refer Fig. 4.2).

Two users are connected through an edge in the Social Interaction graph if they are socially connected. In the User-Item Interaction graph, an edge between a user and an

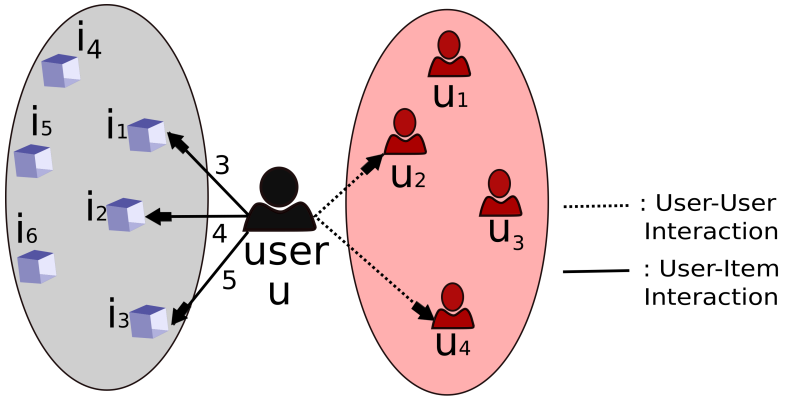


Figure 4.2: In the Social Recommender system, the graph contains two sub-graphs, user-item interaction [left sub-graph] and Social interaction [right sub-graph]. As per the figure, the user u trusts users u_2 and u_4 and rated i_1, i_2 and i_3. The numbers mentioned on the user-item connection denote the ratings given by the user u to the respective items [4].

item indicates the user has an opinion about the item. For the Social Interaction, we denote $\mathcal{F} = \{f_{kj}\} \in \mathbb{R}^{n \times n}$ as the Friendship matrix, where

$$\begin{aligned} f_{kj} &= 1, \text{ if user } u_k \text{ trusts } u_j, \\ &= 0, \text{ Otherwise.} \end{aligned} \tag{4.3.1}$$

For User-Item Interaction, we denote $\mathcal{R} = \{r_{kj}\} \in \mathbb{R}^{n \times m}$ as the Rating matrix, where

$$\begin{aligned} r_{kj} &\neq 0, \text{if user } u_k \text{ has rated the item } i_j. \\ &= 0, \text{if user } u_k \text{ has not given his/her opinion on } i_j. \end{aligned} \tag{4.3.2}$$

Let $\mathcal{O} = \{< u_k, i_j > | \ r_{kj} \neq 0\}$ be the set of observed ratings and $\mathcal{T} = \{< u_k, i_j > | \ r_{kj} = 0\}$ be the set of unknown ratings. In the data, three types of "neighboring" concepts are present.

- **neighbor of user in U**: $\mathcal{N}(k)$ is the set of users who share direct connections with u_k, i.e.,

$$\mathcal{N}(k) = \{u_j \mid f_{kj} \neq 0\}. \tag{4.3.3}$$

- **neighbor of a user in I**: $\mathcal{D}(k)$ is the set of items, which are directly connected with u_k.

$$\mathcal{D}(k) = \{i_j \mid r_{kj} \neq 0\}. \tag{4.3.4}$$

- **neighbor of item in U**: $\mathcal{B}(j)$ is the set of users, who have rated item i_j, i.e.,

$$\mathcal{B}(j) = \{u_k \mid r_{kj} \neq 0\}. \tag{4.3.5}$$

Our aim is to predict the unknown ratings, i.e., we have to predict $r_{kj} \in \mathcal{R}$ such that $< u_k, i_j > \in \mathcal{T}$, using Graph Neural Network.

4.3.2 An Overview of the Proposed Model: SInGER

As depicted in Fig. 4.3, the proposed model SInGER in [4] comprises three major modules, namely Item-category-based User Influence Estimation, Social-Influence-Aware Graph Embedding Generation, and Rating Prediction.

4.3.3 Item-category-Based Influence Estimation of a User in Social Networks

This section briefly discusses the method of estimating the influence of a user among others in social networks. By this Category-based Influence Score, we aim to estimate the authoritativeness of each user in the considered category of items. This section can be further divided into two modules, namely Category-based Influence Propagation Graph generation, and Category-based Influence Score generation, a real number between $[0, 1]$, which will be treated as a feature of a user that expresses the influence of each user on other users based on the considered category of item. Therefore, each user will acquire different influence scores based on the categories of items. We discuss the details of the modules in the following subsections.

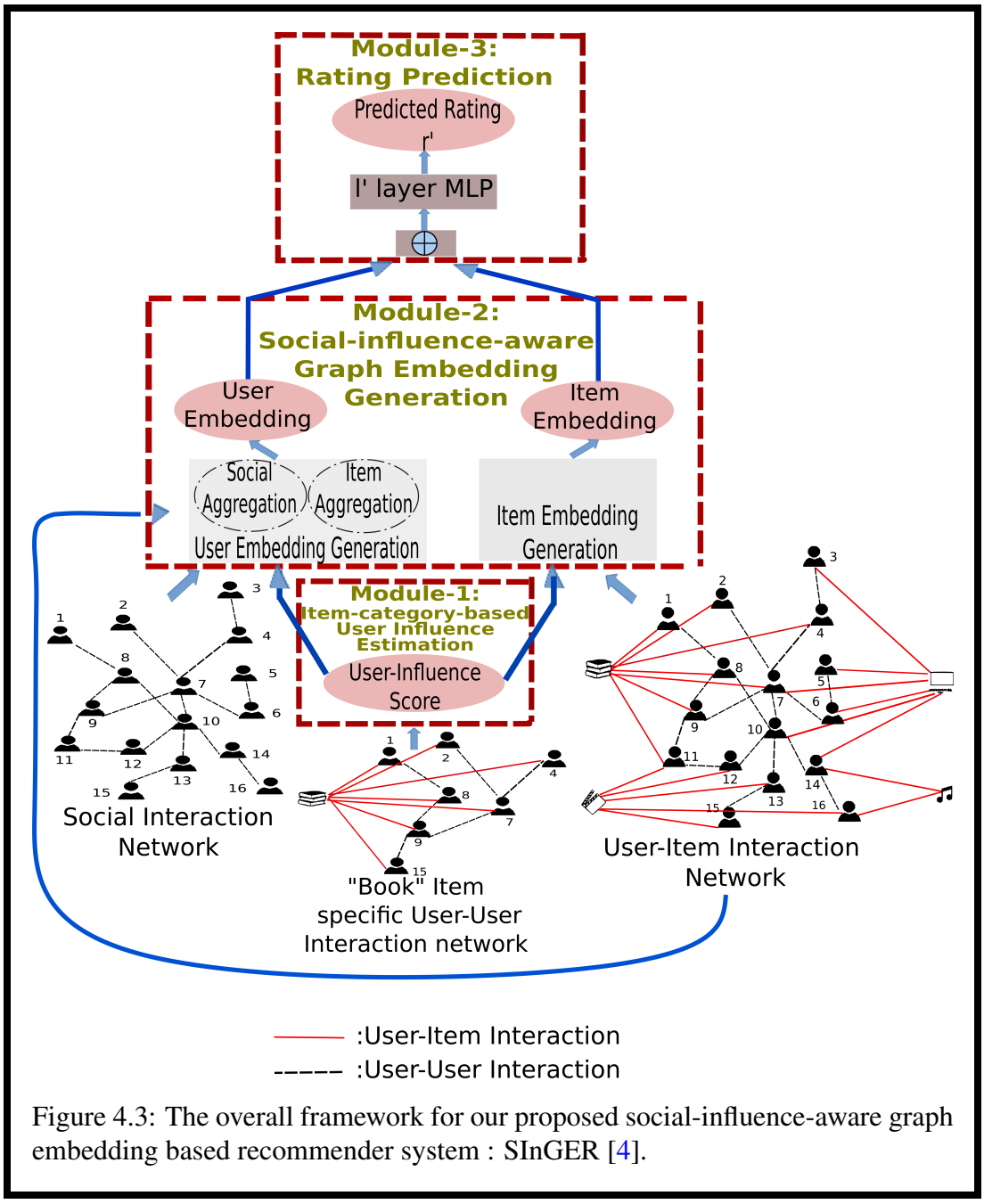


Figure 4.3: The overall framework for our proposed social-influence-aware graph embedding based recommender system : SInGER [4].

Table 4.1: Table of Notations

Notations	Descriptions
$\oplus$	The concatenation operator of two vectors
$\mathcal{B}$(j)	The set of neighbors of the item i_j in the User-Item Interaction Graph
c_j	The context, generated by Encoder LSTM for the user u_j
d_j	The degree of the user u_j in Social Interaction Graph
$\mathcal{D}$(k)	The set of neighbors of the user u_k in the User-Item Interaction Graph
e_r	The embedding of the rating $r \in \{1, 2, 3, 4, 5\}$
f_{kj}	The kj-th entry of the adjacency matrix of Social Interaction graph
$\mathcal{F}$	The Friendship matrix from the Social Interaction Graph
h_j^I	The item-space embedding for the user u_j
h_j^S	The Social Interaction embedding for the user u_j
h_j	The user embedding combining item-space embedding and Social Interaction embedding of the user u_j
I	The set of all items
$\mathcal{N}$(k)	The set of neighbors of the user u_k in the Social Interaction graph
$\mathcal{O}$	The set of observed ratings
p_k	The embedding of the user u_k
q_j	The embedding of the item i_j
r_{kj}	The actual rating that user u_k has given to the item i_j
r'_{kj}	The predicted rating that user u_k may give to the item i_j
$\mathcal{R}$	The Rating matrix from the User-Item Interaction Graph
t_l	The reliability of the user u_l
$\mathcal{T}$	The set of unknown ratings
U	The set of all users
x_{ja}	The opinion and User-Reliability-aware representation of the interaction of the user u_j with the item i_a
y_{jl}	The opinion and User-Reliability-aware representation of the interaction of the item i_j with the user u_l
z_j	The item embedding of the item i_j

4.3.3.1 Item-Category Specific Influence Propagation Graph Generation

An elementary goal of our approach is to capture information propagation among the users in the social network based on the category of products. To consider the Propa-

gating and Composing nature of the influence of users [86], we define Item-category specific Influence propagation graph, $G_c(U_c, E_c)$, where c is a given category of item, U_c consists of those users who have rated at least one item of the category c, and E_c is the set of undirected edges, and two users share an edge if they are friends on social network.

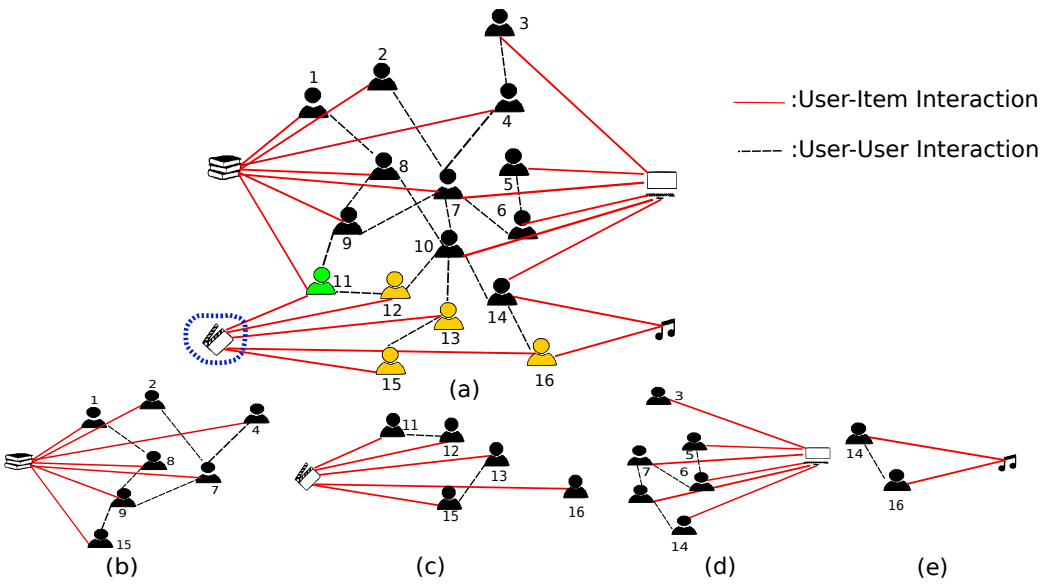


Figure 4.4: Overview of our proposed Item-Category specific Influence propagation graph generation process:**(a)**-Social Recommender System framework, where the items can be categorized into 4 types, namely Books, Movies, Electronics and Music. **(b)**- influence propagation graph for item category books. **(c)**- influence propagation graph for item category movies. **(d)**- influence propagation graph for item category electronics. **(e)**- influence propagation graph for item category music. For example, the Influence Score of user no. 11 (marked in green) regarding the "Movie" category (marked in blue circle), the set of users, who have rated the movie that u_{11} has also rated, will be considered, i.e.$\{u_{12}, u_{13}, u_{15}, u_{16}\}$, all are marked in yellow [4].

4.3.3.2 Influence Score of a User based on Item-Category

In this module, we define a measure, Influence Score, denoted by $inf(u_{i,c})$, which represents the influence of the i^{th} user for rating item belonging to category c. More formally, the Influence Score of a user u_i with respect to items of a category c is defined

as

$$inf(u_{i,c}) = \sum_{item \in I_{i,c}} \left(\frac{\sum_{u_j \in U_{item}} \frac{|I_{j,c}|}{dist(u_i, u_j)}}{|U_{item}|}\right), \tag{4.3.6}$$

where,

$$\begin{aligned} I_{i,c} &= \text{The set of items of category } c \text{ that the user } u_i \text{ has rated.} \\ U_{item} &= \text{The set of users who have rated the item other than the user } u_i. \\ dist(u_i, u_j) &= \text{The distance of user } u_i \text{ and } u_j \text{ in the social network graph.} \end{aligned}$$

Motivation behind using this Formula We have estimated the Influence Score of each user u_i on the community with respect to items belonging to category c by analyzing u_i's rating activities and the activities of the other users towards the same category. In other words, for each user u_i, we aim to analyze all the items of category c, $I_{i,c}$, that are rated by u_i in order to understand the effect of this rating by u_i spreading all over the network, where every user is interested in the same category of items. We have also taken into account the set of users, U_{item}, who have an opinion about that item. Moreover, it can be noticed that, with the function $dist(u_i, u_j)$, we aim to take into account the distance in the social network between the persons. Note that we want to consider the whole social interaction graph, not the Category-specific Influence graph because it will allow the users to look at other item categories, which they may not have rated before. The higher the distance between the users, the less the effect of the considered person's rating on another. We believe that the influence of a friend of a friend cannot be considered of the same importance as of a direct friend of a user. The function $dist$ allows the assigning of weight to the flow of information. Consequently, the Influence Score is used for weighting the rating provided by the user.

To express the Influence Score of a user for a given category as a feature of the user,

we normalize the value based on the Influence Score of the other users to infer who has greater influence between two users regarding a particular type of item. Therefore, the normalized Influence Score of u_j about the product category c is denoted by $\mathcal{I}_j^c$ and can be expressed as follows:

$$\mathcal{I}_j^c = \frac{inf(u_{j,c}) - \min_{u_l \in U_c} inf(u_{l,c})}{\max_{u_l \in U_c} inf(u_{l,c}) - \min_{u_l \in U_c} inf(u_{l,c})} \in [0, 1], \tag{4.3.7}$$

where, $\min_{u_l \in U_c} inf(u_{l,c})$ is the minimum and $\max_{u_l \in U_c} inf(u_{l,c})$ is the maximum of all users' influence values $\{inf(u_{1,c}), inf(u_{2,c}), ..., inf(u_{n,c})\}$ in U for the item-category c.

4.3.4 Social-Influence-Aware Graph Embedding Generation

In this module, we infuse the Influence Score of the users in a Graph Neural Networks-based Recommender model so that the model is able to generate the embeddings in a social-influence aware manner. This module consists of two components, namely User Embedding Generation and Item Embedding Generation. The utility of a Graph Neural Network is that it finds Euclidean representations of nodes and edges of a graph, which is non-Euclidean in nature, preserving structural and feature information of the graph. As we have discussed earlier, in the social recommender system, two types of nodes are present, namely, user and item. Users interact with other users on social networks and also rate the items in the e-commerce system. Hence, to find the Euclidean representation of users, it is necessary to include information from social interaction as well as the user-item interaction. On the other hand, as items are connected with users, one must be aware of the following facts while finding item embedding: How many users have rated the item? Who are they? And how much they have rated?

4.3.4.1 User Embedding Generation

As user nodes interact with items as well as other users, the User Embedding Generation consists of two components, i.e., Item aggregation and Social aggregation.

Item Aggregation This component aims to capture a user's interaction with the items. Each user-item interaction contains the following types of information.

- which items the user has rated,
- how much rating he/she has given, and
- how much Influential he/she is when that particular kind of items is concerned.

To get the item-space embedding of the user using the Graph Neural Network framework, we start with two embeddings: q_a for the item and e_r for the opinion/rating. Both are initialized randomly by d'-dimensional vectors.
The framework for item-space embedding generation can be expressed as follows.

$$h_j^I = \sigma(W.Aggre_{items}(\{x_{ja}, \forall a \in \mathcal{D}(j)\}) + b). \tag{4.3.8}$$

where $\mathcal{D}(j)$ is the set of items in the User-Item graph, which the user u_j has already rated. W is a learnable matrix, b is a bias and σ is a non-linear activation function. $Aggre_{items}$ is an aggregation function to accumulate information from all of the neighboring nodes of u_j in the User-Item graph.
x_{ja} is a representation of the interaction between user u_j and item i_a, which capture the connection as well as the rating information.
However, as discussed earlier, in reality Influence of a user can play a crucial role and we bring Influence Score based on the item category to include a user's Trustworthiness in this task. For this purpose, MLP g_v is used to get a combined representation of the embedding of the item and the rating, weighted by users' User-Reliability-value. Therefore, the embedding generation for a rating relation between a user u_a and item i_j in our model can be expressed as,

$$x_{ja} = g_v([q_a \oplus e_r.\mathcal{I}_a^c]), \tag{4.3.9}$$

where, c is the category of the item i_j and $\mathcal{I}_a^c$ is the Influence Score of the user u_a regarding the c category items. Also, we have seen many choices of this aggregation

function available in the GNN literature. It can be a MAX function as of GraphSAGE [35], then the embedding generation step becomes,

$$h_j^I = \sigma(W.MAX(\{x_{ja}, \forall a \in D(j)\}) + b). \tag{4.3.10}$$

Another choice of $Aggre_{items}$ is $MEAN$ function as used in Graph Convolutional Networks (GCN) [9]. Accordingly, the Eq. 4.3.8 can be presented as follows,

$$h_j^I = \sigma(W.MEAN(\{x_{ja}, \forall a \in D(j)\}) + b). \tag{4.3.11}$$

However, to develop a maximally powerful GNN, Graph Isomorphism Network(GIN) in [36] has used SUM as an aggregation function, and this has been found to achieve the maximum discriminative power among all GNNs. So, in that case, $Aggre_{items}$ function will look like,

$$h_j^I = \sigma(W.SUM(\{x_{ja}, \forall a \in D(j)\}) + b). \tag{4.3.12}$$

All of these above-discussed aggregation functions give the same weightage to every connection with the user u_j. However, in the present scenario of social recommendation, this is not the case. Different connections may contribute differently to generate the representation vector for u_j. For example, connections with some items may be more informative about the user's preference than other connections. Similarly, a user may have a large number of neighbors in the graph, but every neighbor may not be equally revealing about the user. While aggregating, the Attention mechanism aims at modeling the relevance between connections corresponding to (u_j, i_a) pairs. Hence, to generate the representation vector for u_j, the attention mechanism is used as inspired by [104]. Here,

$$h_j^I = \sigma(W.\{\sum_{a \in D(j)} \alpha_{ja} x_{ja}\} + b) \tag{4.3.13}$$

where α_{ja} denotes the attention weight of the connection between the user u_j and the item i_a. To learn this attention weight α_{ja}, a two-layer neural network has been used,

which takes the concatenation of x_{ja} and the user embedding p_j of user u_j. The user embedding p_j is also initialized randomly and learned during the training of the model. Finally, the weights are normalized to get the attention weight.

The framework for item aggregation step in our model can be summarized as,

$$\begin{aligned} x_{ja} &= g_v([q_a \oplus e_r.\mathcal{I}_a^c]) \\ \alpha_{ja}^* &= w_2^T.\sigma(w_1.[x_{ja} \oplus p_j] + b_1) + b_2) \\ \alpha_{ja} &= \frac{exp(\alpha_{ja}^*)}{\sum_{a \in D(j)} exp(\alpha_{ja}^*)} \\ h_j^I &= \sigma(W.\{\sum_{a \in \mathcal{D}(j)} \alpha_{ja} x_{ja}\} + b). \end{aligned} \tag{4.3.14}$$

Here, c is the category of the item i_j and $\mathcal{I}_a^c$ is the Influence Score of the user u_a regarding the c category items. p_j is the randomly initialized user embedding vector, which is to be learned during the training of the model. A two-layer neural network has been used to find the attention weight using the concatenation of x_{ja} and p_j and the weights are normalized among all the items that user u_j has rated. Finally, the item-space embedding of a user u_a is generated using a learnable weight matrix W, non-linear function σ and the bias vector.

Social Aggregation This component focuses on a user's interactions with other users in social media. As we are interested in predicting whether a user will like or dislike a particular item, it is important to include the opinion of the user's friends opinion about that item because a user is usually influenced provoked by user's friends' recommendations. In the previous Item aggregation component, we have generated item-space embedding of a user based on the items he/she has rated, the amount of rating he/she has given and how influential he/she is with respect to various categories of the items. So, the most natural intuition will be to aggregate the item-space embedding of the friends of the user to encode their choice of items. As for, in the previous component, an attention mechanism is used to find the weightage at the time of aggregating the item-space embedding of the social friends of the user. Hence the social-space

embedding generation of a user u_j can be summarized as,

$$\begin{aligned} \beta^*_{jo} &= w_2^T.\sigma(w_1.[h^I_o \oplus p_j] + b_1) + b_2 \\ \beta_{jo} &= \frac{exp(\beta^*_{jo})}{\sum_{o \in \mathcal{N}(j) exp(\beta^*_{jo})}} \\ h^S_j &= \sigma(W.\{\sum_{o \in \mathcal{N}(j)} \beta_{jo} h^I_o\} + b). \end{aligned} \tag{4.3.15}$$

Embedding Of A User A l'-layer MLP has been used to accumulate the item-space embedding and social-space embedding for a user, that are generated in the previous components.

$$\begin{aligned} \gamma^{(0)}_j &= [h^I_j \oplus h^S_j] \\ \gamma^{(i)}_j &= \sigma(W_i.\gamma^{(i-1)}_j + b_i), \text{ for i} = 1, 2, ..., l' \\ h_j &= \gamma^{(l')}_j. \end{aligned} \tag{4.3.16}$$

Here, W is a learnable weight matrix, σ is a non-linear function and b is the bias vector.

4.3.4.2 Item Embedding Generation

For the set of items, as they interact only with the users, we gather information through social aggregation from the User-Item graph. We aim to generate item embedding for each item considering three aspects as follows.

- Which users have rated the item,
- How much influential the users are, and
- What are their opinions about that item.

As in our previous approach, for an item i_j, we use the embeddings of those users, who have rated the item i_j and the rating embedding for the opinion that the user has expressed about that item. Similar to the approach of item-space embedding generation of a user, we scale the rating based on the Category-specific Influence Score $\mathcal{I}^c_l$ of the user u_l for category c. Hence, the item embedding generation process can be

summarized as,

$$\begin{aligned} y_{jl} &= g_u([p_l \oplus e_r.\mathcal{I}_a^c), \\ \mu_{jl}^* &= w_2^T.\sigma(w_1.[f_{jl} \oplus q_j] + b_1) + b_2, \\ \mu_{jl} &= \frac{exp(\mu_{jl}^*)}{\sum_{l \in \mathcal{B}(j)} exp(\mu_{jl}^*)}, \\ z_j &= \sigma(W.\{\sum_{l \in \mathcal{B}(j)} \mu_{jl} y_{jl}\} + b). \end{aligned} \tag{4.3.17}$$

4.3.5 Rating Prediction Module

The rest of the work is to estimate the rating value, that are missing in the dataset using the embeddings of the user and the item. The concatenated vector of the user embedding and item embedding has been fed to a simple MLP with l' hidden layers. The steps of rating prediction module can be formalized as,

$$\begin{aligned} g^{(0)} &= [h_k \oplus z_j] \\ g^{(i)} &= \sigma(W_i.g^{(i-1)} + b_i), \text{for i} = 1, 2, ..., l' - 1. \\ r_{kj}^{'} &= w^T.g^{(l'-1)} \end{aligned} \tag{4.3.18}$$

Here, $r_{kj}^{'}$ is the predicted rating value for the user-item interaction between the user u_k and item i_j. The forward propagation steps of SInGER can be summarized as Algorithm 3.

Model Training To train the model, the square of the differences between the predicted rating r'_{kj} and actual rating r_{kj} given by the user u_k to the item i_j is taken as the loss function, i.e.,

$$Loss = \frac{1}{2|\mathcal{O}|} \sum_{<u_k, i_j> \in \mathcal{O}} (r'_{kj} - r_{kj})^2 \tag{4.3.19}$$

Algorithm 3 Prediction for the missing rating r_{jk} in SInGER (Forward propagation)

Input : user u_j; item i_k; a pre-trained l'- layer MLP model MLP; User-User graph U; User-Item graph I; a pre-trained Attention Model $Atten$; a pre-trained weight matrix W; User-Influence set$\{\mathcal{I}_a^c : u_a \in U\}$

Output : Predicted missing rating r'_{jk}

1: $h_j =$ User Embedding generation$(u_j, U, I, Atten, MLP, W, \{\mathcal{I}_a^c : u_a \in U\})$
2: $z_k =$ Item Embedding generation$(i_k, I, Atten, W, \{\mathcal{I}_a^c : u_a \in U\})$
3: $g = [h_j \oplus z_k]$
4: $r'_{jk} = MLP(g)$
5: **return** r'_{jk} ▷ The prediction for the missing rating r_{jk}

4.4 Experimental Results

This section gives the details of the datasets used for experiments, performance evaluation metrics, parameter settings and comparison of our model's performance against four state-of-the-art approaches. All experiments are performed on the computer with Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz $\times 8$ and 8 GB RAM. All implementations are done using Python3.

4.4.1 Datasets

The experimentation is carried out using two benchmark datasets of popular product review sites, namely *Epinions* and *Ciao* [96]. In both datasets, two sets of information are provided: rating information and trust network information. Rating information consists of tuples of the form $(1, 2, 3, 4, 5)$. It means user 1 has given item 2 of type 3 a rating of 4. The helpfulness of this rating is 5. On the other hand, the trust network information consists of tuples of the form $(1, 2)$, which means user 1 trusts user 2.

Ciao is a European-based online-shopping portal established in the United Kingdom, France, Spain, Germany, the Netherlands, Italy, and Sweden. In February 2008, the company launched an American version of the site, www.ciao.com. CIAO has 7376 number of users and 106798 number of items, respectively. The total number of rating connections is 28331, and the number of user-user interactions is 112365. The den-

sity of the User-Item graph and social interaction graph are 0.03596% and 0.2065%, respectively, which shows the fact that both graphs are very sparse in nature.

Epinion is a well-known general consumer review site established in 1999. The number of users and items in the EPINION dataset is 49290 and 139739, respectively. The total number of rating connections is 764353, and the number of user-user interactions is 502513. The density of the user-item graph and social interaction graph are 0.013% and 0.028%, respectively. The details of the datasets can be found in Table 4.2.

Table 4.2: Characteristics of the datasets

Datasets	CIAO	EPINION
# of users	7376	42290
# of items	106798	139739
# of ratings	283319	764352
Density of the User-Item graph	0.03596%	0.01293%
# of social connections	112365	502513
Density of social interaction graph	0.2065%	0.02809%
Number of product category	28	27

4.4.2 Performance Metrics

Two evaluation metrics, namely Mean Absolute Error (MAE) and Root Mean Square Error (RMSE), are used to assess the effectiveness of our model. These are defined as,

$$\begin{aligned} MAE &= \frac{\sum_{i=1}^{n} |y_i - x_i|}{n}, \\ RMSE &= \frac{\sqrt{\sum_{i=1}^{n} |y_i - x_i|^2}}{n}, \end{aligned} \tag{4.4.1}$$

where y_i and x_i are the predicted value by the model and true value, respectively. n is the total number of data points on which the model has been evaluated. Essentially, smaller values of MAE and RMSE mean that the ratings are more accurately predicted, and when it is exactly equal to the actual rating, MAE and RMSE will be 0.

4.4.3 Parameter Settings

We have used 80% of each dataset to train the model, 10% to validate and the rest 10% of the dataset to test the performance of the model. The details of hyper-parameters are as follows: $<$embedding size: 64, batch size: 256 and learning rate: 0.001 $>$. The experimental results of an empirical study on hyper-parameter settings can be found later in 4.4.5.2.

4.4.4 Baselines

We have compared our proposed model with four state-of-the-art neural network-based recommender system.

- NeuMF [94]: This model uses a neural network approach for the first time to generate users encoding using the User-Item interaction network only.
- GCMC+SN [96]: This model uses a graph auto-encoder model to generate user embedding and graph convolutional matrix completion approach to perform to social recommendation task.
- GraphRec [98]: This is a newly developed model which includes a User-User Interaction graph and User-Item Interaction graph to perform the rating prediction task using Graph Neural Network.
- RSGAN [99]: This recent work includes two modules, namely, generator and discriminator. Generative Adversarial Net (GAN) has been used to create friends of a user to correctly predict a user's preference and a discriminator to assess the generated friend's preferences.

4.4.5 Results and Discussion

4.4.5.1 Results of Comparative Study on Model Performance

The performance of our model on CIAO and EPINION datasets is summarized in Table 4.3, in comparison with the considered baselines. Our experimental findings can

Table 4.3: Performance Comparison with other Recommender System Models

Datasets		CIAO		EPINION	
Evaluation Metric		MAE	RMSE	MAE	RMSE
Models	NeuMF	0.8229	1.0850	0.9223	1.1510
	GCMC+SN	0.8169	1.1012	0.8802	1.0709
	GraphRec	0.7928	1.0812	0.8426	1.1232
	RSGAN	0.7492	0.9879	0.8398	1.0480
	SInGER	**0.6857**	**0.7592**	**0.5843**	**0.8961**

be summarized as follows.

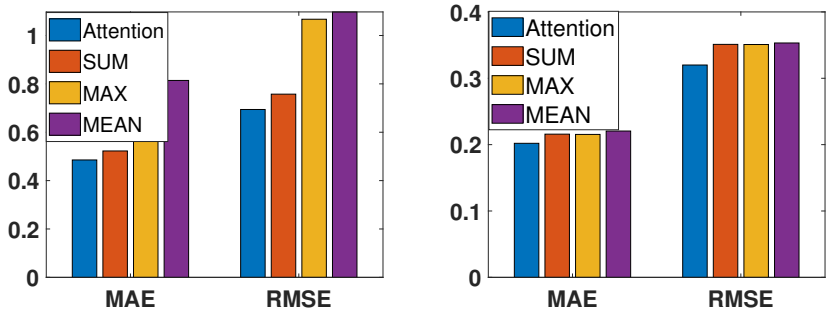
- The performance of NeuMF is poorer than all competing methods, which shows that to capture the user's preference completely, modelling the user-item interaction is not enough. Social interaction related information helps to improve the prediction performance.
- The two models GCMC+SN and GraphRec follow Graph Neural Network-based approaches and achieve better performance by utilizing the power of GNN to encode Non-Euclidean objects in low dimensional vector.
- RSGAN beats other models since penalizing the unreliable connections helps to improve the model's performance.
- Our model outperforms all other state-of-the-art approaches, showing that consideration of the user's Influence Score, in addition to the utilization of GNN to user-user interactions and user-item interactions, is important for predicting the missing rating.

4.4.5.2 Empirical Results for Different Parameter Settings

This empirical study is done on the choice of aggregation function and other hyper-parameters, including learning rate, batch size, and embedding size. Below we analyze our model performance while considering the effects of these parameters.

Effect of Aggregation Functions: We have performed experiments with four different functions, namely *Attention mechanism*, *SUM*, *MEAN*, and *MAX*. As can be interpreted from the figure Fig. 4.5, the *Attention Mechanism* as used in SInGER works best on both the datasets. *SUM* performs much better than *MEAN* and *MAX* on CIAO dataset. However *SUM* and *MEAN* work almost equivalently on EPINION dataset and outperform *MAX*.

Effect of Embedding Size: SInGER predicts the missing rating values in the dataset

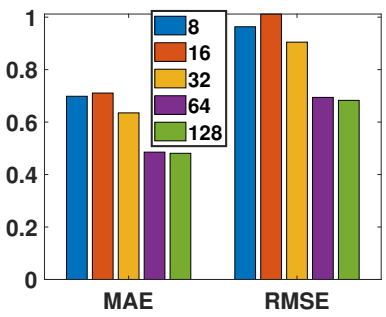


(a) CIAO (b) EPINION

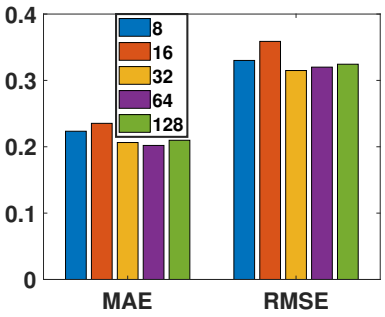
Figure 4.5: Effect of different aggregation function used for embedding generation for recommendation [4].

using three embeddings: User Embedding, Item Embedding, and Rating Embedding. The dimension of the space in which the users, items and ratings are to be embedded is another hyper-parameter.We have experimented with 5 different embedding sizes, namely $8, 16, 32, 64, 128$ as depicted in Fig. 4.6. As can be seen from the figures, with embedding sizes of 64 and 128, appeared to be performing the best. However, generating embedding in space $\mathbb{R}^{128}$ takes much more time than in $\mathbb{R}^{64}$. Hence, we use 64 as the dimension for embedding generation.

Effect of Different Batch Size: Batch size means the number of training examples that are used in each iteration. To speed up fetching the data from memory, batch size is normally taken as a power of 2. We have experimented with the batch sizes $\{32, 64, 128, 256, 512\}$. As shown in Fig. 4.7, our model SInGER works best for batch size of 256.

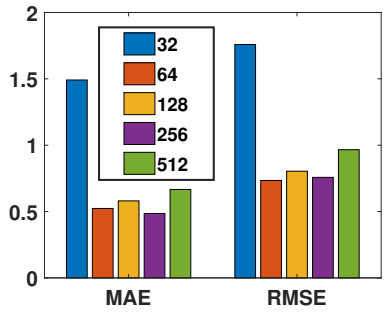


(a) CIAO

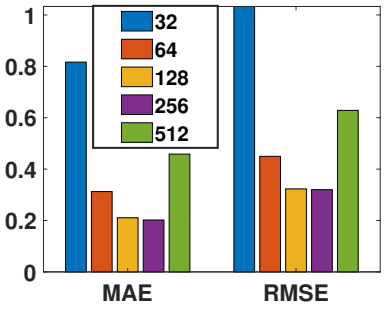


(b) EPINION

Figure 4.6: Effect of different embedding size used for user-embedding, item-embedding and rating-embedding [4].



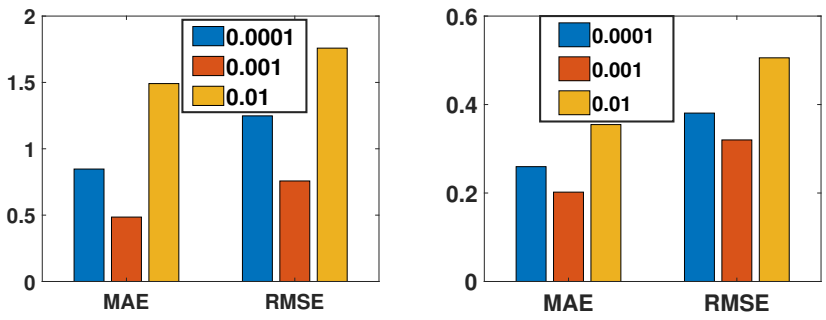
(a) CIAO



(b) EPINION

Figure 4.7: Effect of different batch size during training [4].

Effect of Learning Rate: The learning rate is a measure of the step size while moving to the minimum of the loss function [67]. There is a trade-off while setting the learning rate. If the learning rate is too high, the search may jump over the minima. On the other hand, if the learning rate is too low, the search may take a long time to converge. We have executed SInGER with the learning rates $\{0.0001, 0.001, 0.01\}$ and finally set the learning rate to be 0.001, since this leads to the best performance of our model, as can be seen from Fig. 4.8.



(a) CIAO (b) EPINION

Figure 4.8: Effect of learning rate during the training of the model [4].

4.5 Conclusion

In this chapter, a new framework, SInGER [4] for GNN-based social recommendation has been introduced, which integrates item category-based influence of a user to predict the missing rating values. The proposed model has been evaluated by experimenting with real-life benchmark datasets that proves its efficiency over the state-of-the-art models.

Although capturing the influence of a user in the way that has been defined in SInGER improves the rating prediction performance by a substantial amount, this kind of emotional aspect can not be captured perfectly by a pre-defined formula. Hence, in the next chapter, we propose a learnable approach to quantify the usability of a given rat-

ing, which we named as *Reliability of a user*.

Chapter 5

User-Reliability-Aware Social Recommendation Framework based on Graph Neural Network

5.1 Introduction

Influence, Reliability, and Trust: these are emotional aspects and, hence, difficult to capture using a predefined formula. They may vary from network to network. Therefore, it is better to make the model learn from the network data "How the Reliability of a user should be defined for that particular data." In order to ensure maximum user satisfaction, the traditional recommendation primarily exploits the User-Item-Rating information. However, there has been an increasing interest in developing Trust-aware recommendation systems. Li et al. in [105] have categorized the social trust in recommender systems into two categories: Trustworthiness of Recommender Systems and Trust-based Recommender Systems. The former deals with a series of noises that a recommender system faces due to attacks of spammers or other manual interference. In the latter category, trust has been formulated as a metric incorporating user

trust information extracted from the available social network or social interaction data. Typically, such Trust-based recommendation systems are based on the hypothesis that, while making decisions, a user usually likes to consult trusted friends' preferences rather than referring to those of others unknown to the user. Inferring the missing preference of a user by referring to the known ratings of friends of the user helps to address 'cold-start' issue in the recommendation system [105]. Our model, SInGER [4], discussed in Chapter-4, tries to quantify the user's trustworthiness as the Influence Score of a user, but it fails to deal with the 'cold-start' problem. It is a serious concern for the recommender system when insufficient information is available for some users or items. It is important to note here that, although GNNs have intrinsic ability to integrate node information and topological structure, this area of Trust-aware social recommendation is still primarily unexplored [98]. In this chapter, we aim to augment further the GNN-based trust-aware recommendation systems with the added facility of taking into account the *User-Reliability*, which aids in better evaluation of user trust and ultimately improves the quality of recommendation [84].

5.1.1 Motivation

The works SoURA [5] and CateReR are primarily motivated by two of the critical aspects of user trust in the social network, namely *trust propagation* and *trust composability*.

1) Trust propagation: In practice, a user's ratings are influenced not only by the ratings of the user's friends but also by the ratings of the friend's friends, to a certain extent. This happens primarily due to the propagative nature (sometimes called transitivity) of user trust, in which trust information can be transmitted from one user to another in a social network, creating chains of user trusts [106]. However, many of the social recommendation models, especially those based on GNNs, are found to pay little attention towards exploiting such trust propagation within a trust network of the users [98].

2) Trust composition: Due to the trust propagation in a trust network, whenever a user

receives multiple chains recommending different amounts of trusts, the user needs to compose the trust information to decide whether and how much he/she can trust the others [106]. The composability of trust also plays a vital role in effectively evaluating user trust in social networks. However, while making social recommendations, the existing models often ignore such trust composability traits of users in a trust network.

5.1.2 Contributions

This chapter first discusses User-Reliability-Aware Social Recommendation Framework based on Graph Neural Network, termed SoURA, developed in [5] that provides an effective means of implicitly accounting for trust propagation and composability while performing GNN-based analyses to accomplish the overall task of item rating prediction. In particular, we exploit the sequence modeling power of long short-term memory recurrent neural network (RNN-LSTM) to jointly learn the trust propagation and trust composability in terms of a new measure, termed as 'User-Reliability'. Subsequently, we utilize the 'User-Reliability' value to scale the respective ratings at the time of GNN-based learning of latent factors for both the user and the item. Note that, though the GNN-based analyses for rating prediction are influenced by the work of Fan et al. [98], SoURA has a more advanced approach to trust evaluation in terms of 'User-Reliability', which further improves the prediction performance. To the best of our knowledge, this is the first work on a social recommendation based on GNN approaches that deals with trust propagation and trust composability issues in user trust networks.

However, SoURA could not predict missing ratings based on categories of the items. Later in this chapter, we extend SoURA to overcome this drawback and discuss the proposed GNN-based Model for Category-wise Reliability-aware Recommendation (CateReR) that models the Reliability Value of a user based on the category of the items.

Thus, the major contributions of this chapter can be summarized as follows.

- A novel approach, proposed in [5], has been discussed to capture the *User-*

Reliability. User-Reliability has been formulated considering the user's social connectivity, which in turn helps to evaluate the user trust during social recommendation in a better way;

- SoURA [5] is extended to Category-wise Reliability-aware Recommendation (CateReR) to approximate the utility/usefulness of the user's ratings depending on the item's category;
- *User-Reliability* is incorporated during User-Item interaction and User-User interaction to generate user embedding and item embedding, respectively;
- The effectiveness of our proposed recommendation models is validated by using real-life benchmark datasets and in comparison with eleven state-of-the-art recommendation algorithms.

5.2 Related Works and their limitations

As we have already discussed existing models of recommender system and their drawbacks in the previous chapter (see Sec. 4.2), here we discuss the existing works on the Social Recommender System that capture the trustworthiness of the users in a supervised way.

Trust in Social Recommender System: Several models of social recommender systems capture the notion of trust from different viewpoints. For example, to develop a trustworthy recommender system, Adversarial dual autoencoder [101] uses one autoencoder as a predictor to infer the user preference and another as a discriminator that carries out cohort rating patterns. The work in [107] combines fake news propagation and recommender system to improve trust and transparency of the system. On the other hand, to capture the credibility or trustworthiness of individual users in a social network, in [108] the model processes location, time, weather, and user requests from the mobile device. In [109], trust has been defined using the clustering coefficient of the social network. However, these methods of capturing trust for a user differ from ours in two aspects. Firstly, these do not include the propagating and composite nature

of trust. Secondly, none of these except [101] exploit the embedding generation ability of GNN. Our model is also different from Adversarial dual autoencoder (ADA) [101] as ADA's focus is to improve the trustworthiness of a recommender system, but the proposed models capture the trust of the users at an individual level.

In this chapter, we design models in such a way that they have a unique capability of handling together the trust propagation and the trust composition, which are basically captured by our proposed measure of User-Reliability of a user. This helps our model better evaluate user trust from social interaction data and eventually improves the prediction quality.

5.3 SoURA: A User-Reliability-Aware Social Recommendation System based on Graph Neural Network

This section discusses the model architecture and methodological details of SoURA. SoURA is consists of *four* modules, allotted for *User-Reliability Modeling, User Embedding Generation, Item Embedding Generation,* and *Rating Prediction* (see Fig. 5.1). As we have two graph structures in the data, our job is to extract information from both graphs and use them to perform the rating prediction task. In the module for *User-Reliability Modeling*, we use the information about the user's social connectivity to model the Reliability Value for each user. The module for *User Embedding Generation* aggregates the Social Graph information and User-Item Graph information of the user to get an embedding for the user in Euclidean space. In the *Item Embedding Generation* module, we use the information about the various ratings of an item, given by the different users to model the Item Embedding. Finally, User Embedding and Item Embedding are used in *Rating Prediction* module to predict the missing rating values. A more detailed description of these four modules is given in the subsequent subsections. We follow the same notations as previous chapter, which is summarized in Table 4.1.

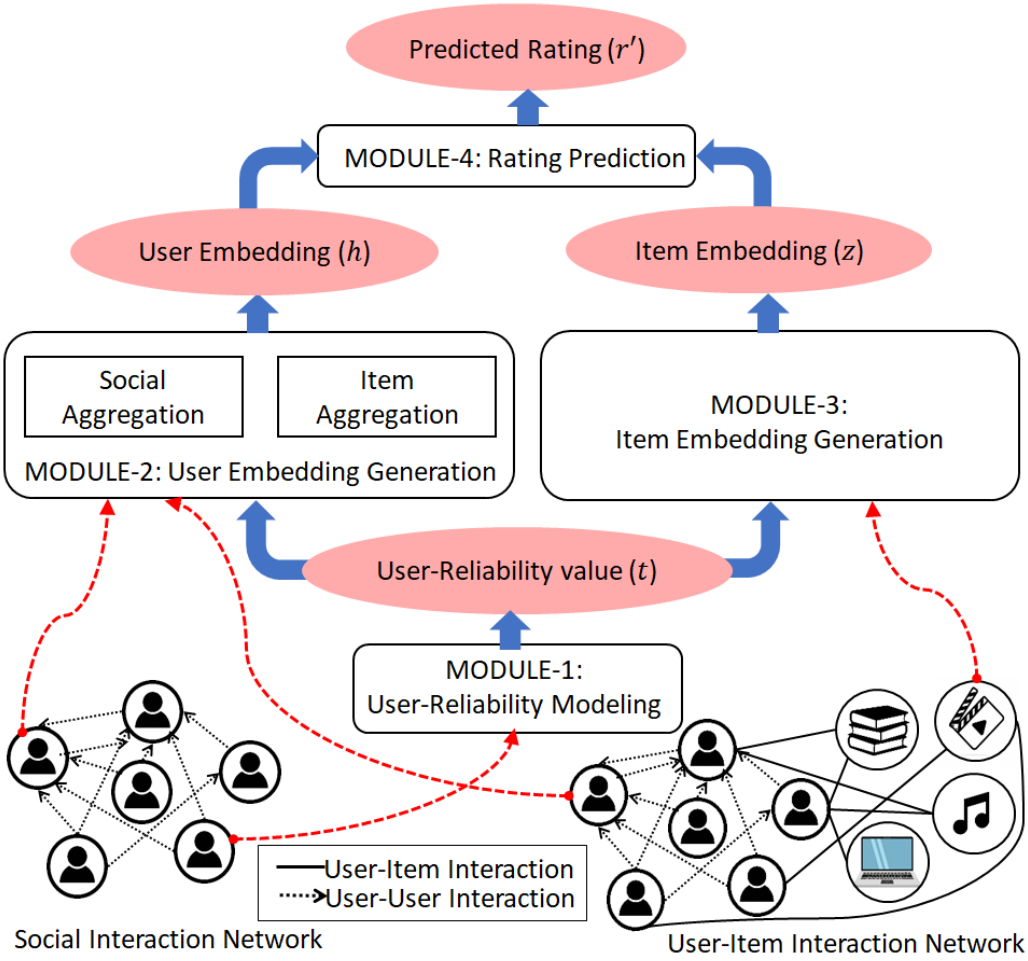


Figure 5.1: Overview of our proposed User-Reliability-Aware Social Recommendation Framework (SoURA)[5]

5.3.1 User-Reliability Modeling

Before explaining the User-Reliability modeling process, it is important to discuss what we mean by User-Reliability and why it is important in the context of social recommendation.

We define User-Reliability as a feature of the user that will measure the future helpfulness of his/her rating based on present connectivity, including direct and indirect trust relations, in the social network. Note that the success of any social recommendation system significantly depends on the appropriate extraction of such user-trust

information from the given social network and effective utilization of the same during predictive analytics. However, to evaluate trust between any two users, the majority of the existing models, including our previous work SInGER [4], only consider their direct interactions while ignoring the impact of trust-propagation and trust-composability [106].

In the last decade, due to the exponential increase of social media, modeling users' choices has become incomplete without studying his/her behavior in social media. In many cases, a user finds an item relevant in his/her life if any friend of the user, whom he/she trusts, uses the item. To make an item accessible to all, it is necessary to introduce the item to a set of users who are connected to a substantially good number of users in the social media and trace the spread of information in the network (also called contamination [86]). Information propagation and spread in networks have a similar analogy with the spread of a disease in a social environment [86]. A person is first susceptible to an infectious disease. When the person is in contact with an infected person, he/she becomes infected and turns out to be a potential carrier of the disease or infector. A similar analogy can be seen in the social recommender system. Every probable customer comes to buy an item if his/her friend or friends of friend use the item. A perfect strategy to promote an item will be to find potential customers who have substantially good connectivity in the social network site.

As already mentioned, none of the existing works on GNN-based social recommendation has incorporated the impact of trust propagation and trust-composability. In order to address this issue, in this section, we propose a novel scheme of augmenting the GNN-based social recommendation through explicit modeling of User-Reliability, while indirectly accounting for the propagation and composition properties of user trust. Our proposed User-Reliability measure is quantified by the number of neighbors (i.e., user node degree) and the number of neighbors' neighbors it has in the social network. The consideration of neighbors' neighbors aids in modeling trust propagation, whereas the use of the degree of a user helps us in composing the user trust.

We measure the amount of Reliability of a user using the Sequence-to-Sequence Encoder-

Decoder model consisting of two LSTM units and incorporate this into our model to improve the quality of the prediction task. Before introducing the User-Reliability modeling scheme, we give a required background of the Sequence-to-Sequence Encoder-Decoder model and the motivation behind using this model to extract User-Reliability value.

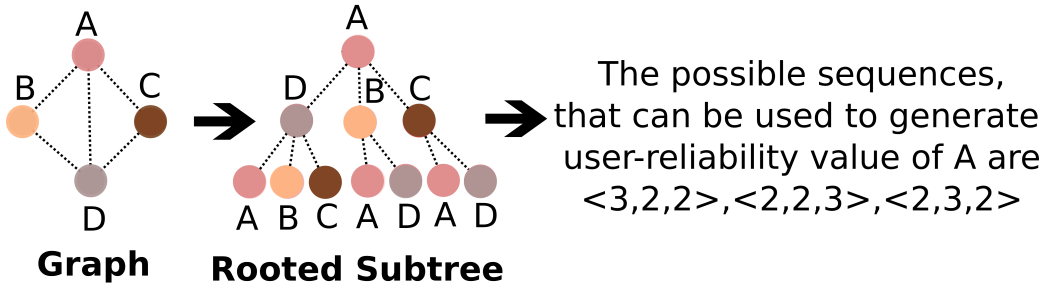


Figure 5.2: The sequence used for generating User-Reliability value. Left Panel: The graph contains four nodes A, B, C and D, representing four users. The node A has three neighbors B, C, D, with degrees (Number of first-order neighbors) $3, 2, 2$, respectively. Middle Panel: The rooted subtree structure at node A. Right Panel: Three possible permutations of the degree set are $< 3, 2, 2 >$, $< 2, 2, 3 >$, and $< 2, 3, 2 >$. One among these three sequences is selected randomly and used as the input to generate a User-Reliability value of A.

5.3.1.1 Sequence-to-Sequence Encoder-Decoder Architecture

Sequence-to-Sequence Encoder-Decoder architecture consists of two LSTMs: Encoder LSTM and Decoder LSTM. The encoder takes a sequence as input and generates a fixed-sized vector C, which we call the context of the input sequence. This context is given as input to the decoder LSTM to generate the output sequence. Thus, the context can be thought of as a summary of the input sequence.

5.3.1.2 Motivation behind the use of Sequence-to-Sequence Encoder-Decoder Architecture

The purpose of our User-Reliability Modeling is to assign a weight value to each of the ratings given by a user based on its own and its neighbors'/friends' social connectivity. one way to measure social connectivity is using the degree of the users. A summarized form of this degree sequence is used to model the Reliability of the user.As mentioned in the previous section, Sequence-to-Sequence encoder-decoder architecture is capable of summarizing a long sequence with variable length into a low dimensional vector and again generating the sequence from the summary vector. Unlike other N-D lattices (e.g., sentences, images, or 3-D volumes), a node in a graph does not have any fixed-sized neighborhood. To capture this arbitrariness, we prefer to use Sequence-to-Sequence Encoder-Decoder architecture over auto-encoder, as traditional auto-encoders can not process sequences with variable sequence lengths. However, it is important to note that the Sequence-to-Sequence Encoder-Decoder architecture is not inherently permutation invariant, as they process inputs in a sequential manner. But, we have modeled the User-Reliability generation problem on a similar notion of word ordering problem in a sentence [110]. Even if the words in a sentence are wrongly arranged, a recurrent neural network (RNN)-based Encoder-Decoder architecture is capable of performing any task on the input sentence and giving equally good results as that of a correctly organized sentence. The reason is, a recurrent Encoder-Decoder architecture does not start giving output right after seeing the first input word. It accepts the set of words and encodes the input data into a dense representation, which can be treated as the summary of the whole set of words and performs the task of learning based on this summary. Although the appropriate encoding method of input data remains an open question, existing methods can be decomposed into two categories:

1. Finding a representative of the set of elements using pooling techniques such as max, min, $mean$, sum etc.

2. A RNN-based approach to represent the set as a densely encoded vector.

The major difference between pooling and RNN-based methods lies in the fact that pooling methods treat the elements of the set independently, while RNN-based methods model the relationships between the entities by treating the input records as a sequence. In our case, two entries of a sequence are not independent. These stand for the number of nodes that share a common grandparent. Also, the length of the sequence eventually represents the number of child nodes of the particular node. As shown in [35], the LSTM-based Aggregator outperforms other pooling techniques while aggregating information from the neighbors of a node. Similar to our application, here also, no straight forward ordering is present in the neighbor set, but the promising result of the sequential model suggests that capturing the dependencies among the input records is helpful. So, inspired by this performance, we feed $10,000$ sequences with sequence lengths ranging from 1 to 100 and repeat each sequence 10 times with different orders of words during training. The model is trained to output the same sequence as the input. Also, to avoid a problem like "Link Farm" (A set of densely connected artificially created users in the system [111]), we use the connectivity strength of the first-order neighbors only to measure the User-Reliability Value of the user. We use this trained architecture to generate a User-Reliability value. The process of generating sequence for a node in a graph is demonstrated in Fig.- 5.2.

5.3.1.3 User-Reliability Value Generation

We want our model to capture the strength of the connectivity of a user through this Sequence-to-Sequence model. As shown in Algorithm 4, we construct a sequence of degrees of the neighbors of a node in the social graph and feed this to the Encoder unit. We want our Decoder unit to output the same sequence. After the training of the model, we extract the Encoder output as the context, which we scale into the interval $[0, 1]$ and use as the reliability value of the user trust. As shown in the Fig. 5.3, the intuition behind this idea is that the Encoder LSTM model will extract useful information about the degree sequence and express it in terms of the context so that the Decoder can again reconstruct the degree sequence from the context. To explain this idea mathematically, we formalize the concept as follows, Let the user u_j has neighbor

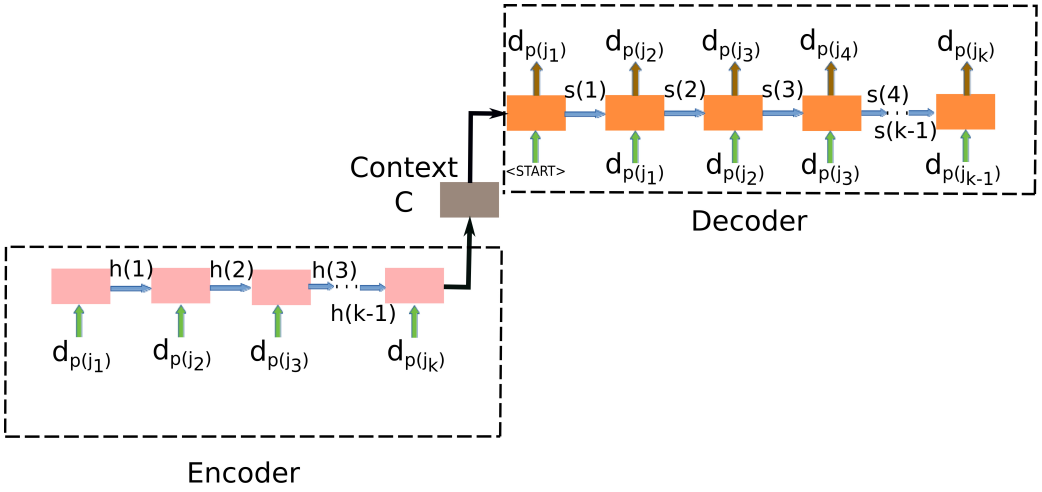


Figure 5.3: Encoder-Decoder system contains two LSTM units, functioning as Encoder [left box] and Decoder [right box], respectively. Every encoder unit takes one element of the degree sequence, say $d_{p(j_l)}$, and the hidden state h_{l-1} generated from the previous encoder unit. After the degree sequence finishes, Encoder outputs a context vector C, which is fed to the Decoder LSTM. Each decoder unit takes a hidden state, say s_{l-1} and the output $d_{p(j_l)}$, generated by the previous decoder unit at every timestamp and produces the next term of the output sequence and the hidden state for the next decoder unit. Our aim is to collect the context C and generate the User-Reliability value by scaling it.

Algorithm 4 Forward propagation of User-Reliability generation for user u_j through context learning

Input : Node u_j; neighborhood $\mathcal{N}(j)$; a pre-trained Encoder-Decoder Model $\mathcal{M} = [Enc, Dec]$,

Output : User-Reliability t_j

1: sequence=<>
2: **for** $v \in \mathcal{N}(j)$ **do**
3: sequence = CONCAT (sequence, degree (v))
4: **end for**
5: context$(u_j) = Enc$ (sequence)
6: $t_j = \frac{\text{context}(u_j) - Min_{u_k \in U}(\text{context}(u_k))}{Max_{u_k \in U}(\text{context}(u_k)) - Min_{u_k \in U}(\text{context}(u_k))}$

set $\mathcal{N}(j) = \{u_{j_1}, u_{j_2}, ..., u_{j_k}\}$.

Then, the sequence used for generating the Trust-Reliability value of u_j is $< d_{p(j_1)}, d_{p(j_2)}, ..., d_{p(j_k)} >$, where p is a permutation on $\{1, 2, ...k\}$ and d is the degree of a user. Accordingly, the context can be presented as,

$$\begin{aligned} c_j = Encoder(< d_{p(j_1)}, d_{p(j_2)}, ..., d_{p(j_k)} >) \text{ such that} \\ < d_{p(j_1)}, d_{p(j_2)}, ..., d_{p(j_k)} > = Decoder(c_j) \\ \text{Then, User-Reliability of } u_j = t_j = \frac{c_j - \min_k c_k}{\max_k c_k - \min_k c_k} \end{aligned} \tag{5.3.1}$$

where, $\min_k c_k$ is the minimum and $\max_k c_k$ is the maximum of all users' context set $\{c_1, c_2, ..., c_n\}$ in U. Hence, $t_j \in [0, 1]$.

5.3.2 User Embedding Generation

This module captures the latent factors of user preferences through a trust-aware analysis based on GNN. Though our User Embedding Generation approach is same as our previous work SInGER [4], this model exploits trust propagation and trust composability, as indirectly captured through the module for User-Reliability Modeling. As discussed earlier, User's item embedding includes information about the user from the User-Item graph, and social embedding includes information from the social (User-User) graph. Algorithm 5 presents the user embedding generation process.

5.3.3 Item Embedding Generation

As in our previous approach, for an item i_j, we use the user's embedding of neighbors of the item i_j in $\mathcal{B}(j)$, and the rating embedding for the opinion that the user has expressed about that item. Here also, we do not want to treat different ratings given by different users equally. These are scaled based on their User-Reliability values. The summary of this module can be found in the Algorithm 6.

Algorithm 5 Forward propagation of User-Embedding generation for the user u_j

Input : Node u_j; User-User graph U; User-Item graph I; a pre-trained Attention Model $Atten$; a pre-trained l- layer MLP model MLP; a pre-trained weight matrix W; User-Reliability set$\{t_k : u_k \in U\}$

Output : User embedding h_j.

function USER-ITEM SPACE EMBEDDING GENERATION($u_j, I, Atten, W, t_j$)
  for $a \in \mathcal{D}(j)$ **do**
    $x_{ja} = W([q_a \oplus e_r.t_j])$
  end for
  $h_j^I = Atten(\{x_{ja} : a \in \mathcal{D}(j)\})$
  return h_j^I ▷ The Item-space embedding of the user u_j
end function
function USER-SOCIAL SPACE EMBEDDING GENERATION($u_j, U, I, Atten, W, t_j$)
  for $o \in \mathcal{N}(j)$ **do**
    $h_o^I =$ User-Item Space Embedding generation$(o, I, Atten, W, t_j)$
  end for
  $h_j^S = Atten(\{h_o^I : o \in \mathcal{N}(j)\})$
  return h_j^S ▷ The Social-space embedding of the user u_j
end function
procedure USER EMBEDDING GENERATION($u_j, U, I, Atten, MLP, W, t_j$)
  $h_j^I =$ User-Item Space Embedding generation$(u_j, I, Atten, W, t_j)$
  $h_j^S =$ User-Social Space Embedding generation$(u_j, U, I, Atten, W, t_j)$
  $c_j^{(0)} = [h_j^I \oplus h_j^S]$
  for $i \in 1, 2, ..., l$ **do**
    $c_j^{(i)} = MLP(c_j^{(i-1)})$
  end for
  $h_j = c_j^l$
  return h_j ▷ The embedding of the user u_j
end procedure

Algorithm 6 Forward propagation of Item-Embedding generation for the item i_j

Input : Item i_j; User-Item graph I; a pre-trained Attention Model $Atten$; a pre-trained weight matrix W; User-Reliability set$\{t_k : u_k \in U\}$

Output : Item embedding z_j

for $a \in \mathcal{B}(j)$ **do**
    $f_{ja} = W([p_l \oplus e_r.t_a])$
end for
$z_j = Atten(f_{ja} : a \in \mathcal{B}(j))$
return z_j ▷ The embedding of the item i_j

5.3.4 Rating Prediction

This is the final module to finish the overall task of social recommendation. In this module, we focus on predicting the missing ratings using the embeddings of the user and the item. In order to accomplish this, we concatenate these two types of embeddings and feed this to an MLP.

$$\begin{aligned} g^{(0)} &= [h_k \oplus z_j] \\ g^{(i)} &= \sigma(W_i.g^{(i-1)} + b_i), \text{for i} = 1, 2, ..., l' - 1 \\ r'_{kj} &= w^T.g^{(l'-1)} \end{aligned} \tag{5.3.2}$$

where l' is the number of hidden layer in the MLP and r'_{kj} is the predicted rating of user u_k for the item i_j. The various steps in the Rating prediction module has been summarized in Algorithm 7.

5.3.5 Limitation of SoURA

As we have already seen, in a social recommender system, the trust relationships may not be limited to only the immediate neighbors [98, 106]. Higher-order neighbors of a person in the social network can also influence his/her perception of a product, which is termed as Trust Propagation [98]. The method, SoURA [5] introduced the notion of User-Reliability and utilized it for missing rating prediction. However, SoURA cal-

Algorithm 7 Prediction for the missing rating r_{jk} (Forward propagation)

Input : user u_j; item i_k; a pre-trained l'- layer MLP model MLP; User-User graph U; User-Item graph I; a pre-trained Attention Model $Atten$; a pre-trained weight matrix W; User-Reliability set$\{t_l : u_l \in U\}$

Output : Predicted missing rating r'_{jk}

1: $h_j =$ User Embedding generation$(u_j, U, I, Atten, MLP, W, t_j)$
2: $z_k =$ Item Embedding generation$(i_k, I, Atten, W, \{t_k : u_k \in U\})$
3: $g = [h_j \oplus z_k]$
4: $r'_{jk} = MLP(g)$
5: **return** r'_{jk} ▷ The prediction for the missing rating r_{jk}

culates the User-Reliability simply by assuming that "a user has greater connectivity because people find his/her review useful". This completely ignores the users' experience in reviewing items under specific categories and thus, produces a gross estimate of User-Reliability. However, this is not always the case. The trust may depend on several other factors, such as 'item category', in an e-commerce system. For example, a user knowing electronics items may not be well-informed about clothes or books. Thus, the reliability or trust value of a user may not be the same for every item category.

Following this intuition, instead of only taking into account the structural connections among users, we aim to discover the nature of their trust relationship based on their exchange of information about each specific item category. This can better evaluate the user-trust and ultimately improve the quality of recommendations.

5.4 CateReR: A Graph Neural Network-based Model for Category-wise Reliability-aware Recommendation

In the following subsections, we overcome the previously discussed limitation of SoURA and develop the next model for category-wise reliability-aware recommendation (CateReR). We aim to examine "how much one user influences other users" depending on the category of the products. For this purpose, we propose a technique to quantify the relia-

bility of a user in a category-specific manner to improve the missing rating prediction task.

Item-Category specific Reliability Value Generation We define category-wise User-Reliability value as the measure of the helpfulness of the user's rating given to a specific category of items. We use the sequence summarizing feature of the Encoder-Decoder LSTM model and express Category-specific User-Reliability value t_j^c of a user u_j as a real number in the range $[0, 1]$. We use the sequence of degrees of the neighbors of a user in the category-specific Trust propagation graph (see Sec. 4.3.3.1) instead of the original User-Item Interaction Network as input, assuming that a person who has rated more number of items under a particular category is more knowledgeable about that product category. Considering the degrees of the neighbors in the Category-specific Trust propagation graph helps to incorporate the trustworthiness of the user's friends as well. We formalize the idea as follows, Let a user u_j^c has a set of neighbors in the Category specific Trust propagation graph $G^c(U^c, E^c)$ $\{u_{j_1}^c, u_{j_2}^c, ..., u_{j_k}^c\}$, i.e., among all friends of u_j^c in social media, this set of friends have rated some items of the category c. Let the degree of the user $u_{j_l}^c$ is $d_{j_l}^c$ for $l \in \{1, 2, ..., k\}$. Then, we consider a permutation of the degree sequence of the neighbors of u_j^c as input to the Encoder to generate the Category-specific Reliability value of u_j^c for category c. The steps of the Category-specific Reliability value generation process can be presented as follows,

$$\mathcal{C}_j^c = Encoder(< d_{p(j_1^c)}, d_{p(j_2^c)}, ..., d_{p(j_k^c)} >) \text{ such that}$$
$$< d_{p(j_1^c)}, d_{p(j_2^c)}, ..., d_{p(j_k^c)} >= Decoder(\mathcal{C}_j^c).$$

Then, in order to express the Category-specific Reliability value as an attribute of the user, the value has been normalized based on the other users' value. It allows us to compare the users' reliability for a specific type of items. Hence, the User-Reliability value of u_j regarding c category turns out to be:

$$t_j^c = \frac{\mathcal{C}_j^c - \min_l \mathcal{C}_l^c}{\max_l \mathcal{C}_l^c - \min_l \mathcal{C}_l^c} \in [0, 1], \tag{5.4.1}$$

where, $\min_l \mathcal{C}_l^c$ is the minimum and $\max_l \mathcal{C}_l^c$ is the maximum of all users' context set $\{\mathcal{C}_1^c, \mathcal{C}_2^c, ..., \mathcal{C}_n^c\}$ in U for the item-category c.

The rest of the process flow within the rating prediction module of CateReR is same as that of our previous model SoURA model.

5.5 Experimental Results

In this section, we demonstrate the performance of our proposed models SoURA [5] and CateReR by comparing their performance on two real-world datasets from social networking websites against that of other baselines. As our proposed models are primarily based on GraphRec [98], the checking for effectiveness of the User Embedding Generation and Item Embedding Generation modules have been skipped. The details of the experimental study are discussed below.

5.5.1 Datasets

The experimentation is carried out using two benchmark datasets of popular product review sites, namely *Epinions* and *Ciao* [96]. The details of the datasets are provided in the Sec. 4.4.1.

5.5.2 Performance Metrics

Similar to SInGER [4], The quality of rating prediction is measured with respect to two Performance Metrics, namely Mean Absolute Error (MAE) and Root Mean Square Error (RMSE). The formal definitions of MAE and RMSE can be found in Sec. 4.4.2.

5.5.3 Parameter Settings

We have divided the dataset as follows: 80% of each dataset is used as training examples to train the model parameters, 10% as the validation set, and the remaining 10% to test the performance of the model. Three types of embeddings are generated in the

model, namely for users, items, and ratings. Dimension of the embedding is varied as per the following set: $\{8, 16, 32, 64, 128, 256\}$, and the size of the hidden layer is kept the same as the size of the embedding. The empirical analysis is also conducted using variants of batch size: $\{32, 64, 128, 256\}$ and considering different learning rates: $\{0.001, 0.01, 0.1\}$.

5.5.4 Baselines

Our proposed models are compared with state-of-the-art recommendation models as summarized below.

- PMF [87]: This is a probabilistic matrix factorization-based technique, which utilizes only the User-Item graph information to perform the prediction task.
- SoRec [88]: This variant of the recommendation algorithm employs a matrix factorization technique on both the User-Item interaction and the social interaction graphs.
- SocialMF [89]: This is a representative of a trust-aware recommendation system that extracts the user's trust information from the social network and incorporates the same while accomplishing the recommendation task.
- SoReg [93]: This variant of the social recommendation algorithm utilizes social interaction graph information for attaining regularized matrix factorization. This, in turn, helps the model to capture diversity in users' preferences.
- TrustMF [91]: This model achieves trust-aware recommendation by means of generating two latent vectors for each user, one for capturing the users whom he/she trusts and the other one to capture those users who trust him/her.
- NeuMF [94]: This is one of the pioneering recommendation models based on a neural network, where the user's representation is learned only from the rating network.

- GCMC+SN [96]: This model offers a graph convolutional matrix completion approach for the social recommendation while using a graph auto-encoder framework to generate the user embedding.

- DeepSoR [102]: This variant of the social recommendation algorithm is primarily attributed to its deep neural network-based approach to extract the hidden features of the users.

- GraphRec [98]: This is a recently proposed model for social recommendation, which is based on GNN analysis. The comparison with GraphRec is necessary since our proposed recommendation approach is influenced by this model.

- RSGAN [99]: This is another state-of-the-art approach that primarily includes two modules, namely, generator, which will create users' friends who will give a prediction about user's preference using Generative Adversarial Net (GAN) and discriminator to asses the generated friends preferences.

- Adversarial Dual Autoencoders(ADA) [101]: This is an extension work on GraphRec, which uses dual autoencoders to find embeddings capturing social trust and rating patterns.

While comparing with these baselines, we have used the same hyper-parameter settings as mentioned in their respective source files and manuscripts. Also, we report the performance of the model Adversarial Dual Autoencoders as mentioned in the manuscript.

5.5.5 Results and Discussions

The results of model performance with respect to MAE and RMSE are summarized in Table 5.1 Our interpretations of these results are discussed below.

5.5.5.1 Results of comparative study on model performance

The main observation from Table 5.1 are listed below,

Table 5.1: Comparative results on models performance

Name of the dataset		CIAO		EPINION	
Performance Metric		MAE	RMSE	MAE	RMSE
Recommendation Models	PMF	0.9292	1.1408	1.014	1.2139
	SoRec	0.8551	1.0727	0.9187	1.1436
	SoReg	0.9082	1.1090	0.9254	1.1679
	SocialMF	0.8386	1.0653	0.8909	1.1398
	TrustMF	0.7842	1.0645	0.8979	1.1624
	NeuMF	0.8229	1.0850	0.9223	1.1510
	DeepSoR	0.8819	1.0335	0.8518	1.1496
	GCMC+SN	0.8169	1.1012	0.8802	1.0709
	GraphRec	0.7928	1.0812	0.8426	1.1232
	RSGAN	0.7492	0.9879	0.8398	1.0480
	ADA	0.748	0.976	0.815	1.054
	SInGER(Proposed)	0.6857	0.7592	0.5843	0.8961
	SoURA (Proposed)	0.4854	0.6941	0.2020	0.3200
	CateReR (Proposed)	**0.2112**	**0.3550**	**0.1813**	**0.2572**

- The neural network-based recommendation models outperform the matrix factorization-based models in a majority of the cases. This emphasizes the fact that the neural networks are able to capture the intrinsic features of the user and item more accurately than the matrix factorization techniques.

- Among the various MF-based and NN-based baselines considered, the PMF and the NeuMF use only the User-Item interaction graph to predict the missing ratings. The poor performances of these models indicate that the mere consideration of the User-Item graph results in missing useful information about the users.

- The DeepSoR, GCMC+SN, and GraphRec, all of which take advantage of the deep network and utilize both rating and social network information, are found to perform notably better than the other baselines.

- The stronger performances of GCMC and GraphRec further reveal the power of GNN-based analysis.

- RSGAN beats other GNN-based models, which shows that penalizing unreliable connections and producing reliable friends help in better predicting users'

preferences.

- ADA improves the performance of the recommender system over GraphRec establishing the fact that introducing the trustworthiness of a recommender system benefits the rating prediction task.

- SInGER outmatches all other baseline approaches, proving that inclusion of user's influence score is an important task while predicting the missing rating.

- SoURA outperforms all the baseline approaches, which establishes that capturing User-Reliability enables the model to further improve its rating prediction ability. Also, it can be noted that SoURA beats ADA and SInGER as well proving the fact that rather than capturing the trustworthiness of a recommender system using autoencoder or unsupervised way, LSTM based encoder-decoder-based technique performs better. Also, it proves that trust does not only depend on the immediate neighbors of a node; higher order neighborhood also has effect on the reliability of a node. Therefore, considering a higher order neighborhood helps to capture the trust of a user in a composite and propagative manner and predict the missing rating value consequently.

- CateReR outperforms all the state-of-the-art approaches including SoURA as well, which demonstrates that the Reliability of a user highly depends on the product category.

5.5.5.2 Ablation Study for User-Reliability Module

In order to examine the effectiveness of the User-Reliability modules defined above for our three models, we have experimented with and without the said modules. It is evident from Table 5.2 that the modules have played a vital role in the encouraging performance of respective models.

Table 5.2: Effect of User-Reliability generation module on prediction performance

Name of the dataset			CIAO		EPINION	
Performance Metric			MAE	RMSE	MAE	RMSE
Variants	Without User-Reliability		0.7928	1.0812	0.8426	1.1232
	with User-Reliability	SInGER(Proposed)	0.6857	0.7592	0.5843	0.8961
		SoURA (Proposed)	0.4854	0.6941	0.2020	0.3200
		CateReR (Proposed)	0.2112	0.3550	0.1813	0.2572

5.5.5.3 Empirical Study on Effect of Different User-Reliability Computation Strategy

We have tried two other strategies for calculating the reliability value of the users: i) treating the scaled value of the degree of connectivity between 0 and 1, ii) keeping every user's Reliability as 1 together with our proposed strategies to generate User-Reliability, that we have discussed in the previous section. The experimental result (see Table 5.3) shows that our approaches outperform the other strategies by a large margin.

Table 5.3: Effect of different strategies for User-Reliability generation

Name of the dataset		CIAO		EPINION	
Performance Metric		MAE	RMSE	MAE	RMSE
Reliability Schemes	Degree/Popularity	0.5787	0.7872	0.5289	0.8956
	Equal Reliability Value	0.7928	1.0812	0.8426	1.1232
	Category specific unsupervised approach(SInGER)	0.4854	0.6941	0.2020	0.3200
	Encoder-Decoder-based(SoURA)	0.7928	1.0812	0.8426	1.1232
	Category specific Encoder-Decoder-based(CateReR)	0.7928	1.0812	0.8426	1.1232

5.6 Conclusion

In this chapter, two User-Reliability augmented GNN models have been introduced to predict the missing rating in a user-reliable manner. To be precise, we have proposed approaches to capture the User-Reliability using Encoder-Decoder architecture

respectively, which allow us to give weightage to the ratings based on the Reliability of the users. Our experimental section shows that the proposed methods far outperforms the state-of-the-art approaches on two real-life datasets. The results demonstrate the importance of incorporating User-Reliability for finding user-embedding and item embedding.

Chapter 6

Conclusions and Future Scope of Research

This chapter summarizes the research contributions of each contributory chapter. Additionally, it provides insight into future research directions related to the research work proposed in this thesis.

6.1 Research Contribution

The main focus of this thesis is to develop novel Graph Neural Network-based approaches to embed different types of graph data. Focusing on the goal of this thesis, four contributing chapters have been designed. The prediction performance of the proposed models, both for homogeneous and heterogeneous graphs, has been validated by corresponding experimental results available in the literature.

Chapter 2 describes a new interval aggregation scheme, having much better discriminative power than the existing interval aggregation functions. Using this newly developed aggregation operator as an AGGREGATE function, a Graph Neural Network-based architecture IV-GNN has been developed, relaxing the condition of the single-valued feature space. Despite being much more general in nature, the proposed method far outperforms the state-of-the-art approaches on several synthetic and real-life datasets

(comparable results for IMDB-BINARY and COLLAB).

As the embedding generation process of IV-GNN is primarily motivated by the Weisfeiler-Lehman Graph Isomorphism Test, it also accumulates information from the immediate neighborhood only. As a result, it fails to encode information from higher-order neighborhoods. To overcome this drawback, Chapter 3 presents a novel approach, GraMMy, that studies a graph from different levels of abstraction. Rather than stacking a number of GNN layers to capture information from distant nodes, it uses Locality Sensitive Hashing as a micro-macro scalar to see a graph as a set of clusters. Also, it aggregates information from the neighborhood of a node in a context-aware fashion.

In Chapter 4 and 5, two frameworks have been proposed for social recommendation using Graph Neural Networks. GNN exploits the User-User Interaction and User-Item Interaction to generate embeddings. Also, the novelties of the works lie in estimating the usefulness of the ratings given by the users based on the Influence Score (Chapter 4) and Reliability (Chapter 5). The metric Influence Score in Chapter 4 has been formulated by analyzing the user rating activities, the other users' activities, and the distance between the users in the social interaction graph. The inclusion of this Influence Score in the embedding generation process improves the quality of missing rating prediction tasks substantially. However, figuring out the influence of a user is not always straightforward. Therefore, in Chapter 5, we made the model to learn the Reliability of a user using an Encoder-Decoder architecture. As a result, the quality of missing rating prediction tasks has been improved further in Chapter 5.

6.2 Limitations and Future Scope

This thesis develops several GNN frameworks to generate embeddings for various graph data. However, there remain a few areas open with scope for future study. Such areas are enlisted as follows:

1. The aggregation function proposed in Chapter 2 is not continuous. Therefore, some situations may arise where a slight change in aggregating intervals may bring a significant change in the resultant interval, which is not expected. Hence,

designing a continuous aggregating function for intervals may greatly interest the GNN researcher.

2. There are architectures that have the summarizing ability of continuous data, such as LSTM. It would be an interesting direction for research to include the interval-valued feature as a summarized embedding and investigate the quality of the performance on the graph classification task.

3. An exciting direction for future work is exploring different interval aggregation methods according to the demand of the situation.

4. The process of mining all possible sequences as described in Chapter 3 to generate contextual information of a node becomes more expensive as the sequences become longer. In the future, we plan to handle this issue by adopting a parallel processing scheme.

5. Although GraMMy gives equal weightage to the embeddings from different abstraction levels, these weights can be learnable. Examining how attention mechanisms work to determine these weights would be interesting.

6. In the social recommender system, the interactions between User-User and User-Item are always evolving. Dealing with the time-varying nature of these interactions can be an interesting research area that we would like to explore in the future.

7. As all of these proposed models in this thesis use the Stochastic Gradient Descent approach, developing a Gradient-free algorithm for graph-embedding generation could be an interesting area for research.

Bibliography

[1] Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017.

[2] Sucheta Dawn and Sanghamitra Bandyopadhyay. IV-GNN: interval Valued data handling using graph neural network. *Applied Intelligence*, 53(5):5697–5713, 2023.

[3] Sucheta Dawn, Monidipa Das, and Sanghamitra Bandyopadhyay. GraMMy: Graph Representation Learning based on Micro-Macro Analysis. *Neurocomputing*, 2022.

[4] Sucheta Dawn, Monidipa Das, and Sanghamitra Bandyopadhyay. Singer: A recommendation system based on social-influence-aware graph embedding approach. In *2021 IEEE 18th India Council International Conference (INDICON)*, pages 1–6. IEEE, 2021.

[5] Sucheta Dawn, Monidipa Das, and Sanghamitra Bandyopadhyay. SoURA: a user-reliability-aware social recommendation system based on graph neural network. *Neural Computing and Applications*, 35(25):18533–18551, 2023.

[6] Sucheta Dawn and Monidipa Das. Graph representation learning for protein classification. In *Artificial Intelligence Technologies for Computational Biology*, pages 1–28. CRC Press, 2022.

[7] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 974–983, 2018.

[8] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in neural information processing systems*, pages 3844–3852, 2016.

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

[10] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[11] Kairanbay Magzhan and Hajar Mat Jani. A review and evaluations of shortest path algorithms. *Int. J. Sci. Technol. Res*, 2(6):99–104, 2013.

[12] Richard Bellman. On a routing problem. *Quarterly of applied mathematics*, 16(1):87–90, 1958.

[13] Robert Tarjan. Depth-first search and linear graph algorithms. *SIAM journal on computing*, 1(2):146–160, 1972.

[14] Uwe Schöning. Graph isomorphism is in the low hierarchy. In *Annual Symposium on Theoretical Aspects of Computer Science*, pages 114–124. Springer, 1987.

[15] Brendan L Douglas. The weisfeiler-lehman method and graph isomorphism testing. *arXiv preprint arXiv:1101.5211*, 2011.

[16] Stephen G Hartke and AJ Radcliffe. Mckay's canonical graph labeling algorithm. *Communicating mathematics*, 479:99–111, 2009.

[17] L. R. Ford and D. R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404, 1956.

[18] Jack Edmonds and Richard M Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM (JACM)*, 19(2):248–264, 1972.

[19] Joseph B Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical society*, 7(1):48–50, 1956.

[20] Robert Clay Prim. Shortest connection networks and some generalizations. *The Bell System Technical Journal*, 36(6):1389–1401, 1957.

[21] Sitao Luan, Chenqing Hua, Minkai Xu, Qincheng Lu, Jiaqi Zhu, Xiao-Wen Chang, Jie Fu, Jure Leskovec, and Doina Precup. When do graph neural networks help with node classification? investigating the homophily principle on node distinguishability. *Advances in Neural Information Processing Systems*, 36, 2024.

[22] ChunYan Meng and Hooman Motevalli. Link prediction in social networks using hyper-motif representation on hypergraph. *Multimedia Systems*, 30(3):123, 2024.

[23] Bingjun Li and Sheida Nabavi. A multimodal graph neural network framework for cancer molecular subtype classification. *BMC bioinformatics*, 25(1):27, 2024.

[24] Max Welling and Thomas N Kipf. Semi-supervised classification with graph convolutional networks. In *J. International Conference on Learning Representations (ICLR 2017)*, 2016.

[25] Jiaxuan You, Rex Ying, Xiang Ren, William Hamilton, and Jure Leskovec. Graphrnn: Generating realistic graphs with deep auto-regressive models. In *International conference on machine learning*, pages 5708–5717. PMLR, 2018.

[26] Martin Simonovsky and Nikos Komodakis. Graphvae: Towards generation of small graphs using variational autoencoders. In *Artificial Neural Networks and Machine Learning–ICANN 2018: 27th International Conference on Artificial Neural Networks, Rhodes, Greece, October 4-7, 2018, Proceedings, Part I 27*, pages 412–422. Springer, 2018.

[27] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2008.

[28] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. Computational capabilities of graph neural networks. *IEEE Transactions on Neural Networks*, 20(1):81–102, 2008.

[29] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*, 2013.

[30] Li Yi, Hao Su, Xingwen Guo, and Leonidas J Guibas. Syncspeccnn: Synchronized spectral cnn for 3d shape segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2282–2290, 2017.

[31] Xian-Hua Han, Boxin Shi, and Yinqiang Zheng. Ssf-cnn: Spatial and spectral fusion with cnn for hyperspectral image super-resolution. In *2018 25th IEEE International Conference on Image Processing (ICIP)*, pages 2506–2510. IEEE, 2018.

[32] Federico Monti, Karl Otness, and Michael M Bronstein. Motifnet: a motif-based graph convolutional network for directed graphs. In *2018 IEEE Data Science Workshop (DSW)*, pages 225–228. IEEE, 2018.

[33] Derek Allan Holton and John Sheehan. *The Petersen Graph*, volume 7. Cambridge University Press, 1993.

[34] Juergen Schmidhuber. Annotated history of modern ai and deep learning. *arXiv preprint arXiv:2212.11279*, 2022.

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

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

[37] Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4602–4609, 2019.

[38] Hongyang Gao and Shuiwang Ji. Graph u-nets. In *international conference on machine learning*, pages 2083–2092. PMLR, 2019.

[39] Gabriele Corso, Luca Cavalleri, Dominique Beaini, Pietro Liò, and Petar Veličković. Principal neighbourhood aggregation for graph nets. *Advances in Neural Information Processing Systems*, 33:13260–13271, 2020.

[40] Lucas Lacasa, Bartolo Luque, Fernando Ballesteros, Jordi Luque, and Juan Carlos Nuno. From time series to complex networks: The visibility graph. *Proceedings of the National Academy of Sciences*, 105(13):4972–4975, 2008.

[41] Lynne Billard and Edwin Diday. Regression analysis for interval-valued data. In *Data Analysis, Classification, and Related Methods*, pages 369–374. Springer, 2000.

[42] Eufrasio de A Lima Neto, Francisco AT de Carvalho, and Camilo P Tenorio. Univariate and multivariate linear regression methods to predict interval-valued features. In *Australasian Joint Conference on Artificial Intelligence*, pages 526–537. Springer, 2004.

[43] Jeongyoun Ahn, Muliang Peng, Cheolwoo Park, and Yongho Jeon. A resampling approach for interval-valued data regression. *Statistical Analysis and Data Mining: The ASA Data Science Journal*, 5(4):336–348, 2012.

[44] L Billard and E Diday. Symbolic data analysis: Conceptual statistics and data mining john wiley, 2006.

[45] Benjamín René Callejas Bedregal and Adriana Takahashi. The best interval representations of t-norms and automorphisms. *Fuzzy Sets and Systems*, 157(24):3220–3230, 2006.

[46] Soma Dutta, Benjamín RC Bedregal, and Mihir Kr Chakraborty. Some instances of graded consequence in the context of interval-valued semantics. In *Indian Conference on Logic and Its Applications*, pages 74–87. Springer, 2015.

[47] Tomasa Calvo, Anna Kolesárová, Magda Komorníková, and Radko Mesiar. Aggregation operators: properties, classes and construction methods. In *Aggregation operators*, pages 3–104. Springer, 2002.

[48] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991.

[49] Qingshan Liu, Jun Wang, and Finite-Time Convergent Recurrent Neural Network. With a hard-limiting activation function for constrained optimization with piecewise-linear objective functions. *IEEE Transactions on Neural Networks*, 22(4):601–613, 2011.

[50] Bernardo Llanas, Sagrario Lantarón, and Francisco J Sáinz. Constructive approximation of discontinuous functions by neural networks. *Neural Processing Letters*, 27(3):209–226, 2008.

[51] Antonio Muñoz San Roque, Carlos Maté, Javier Arroyo, and Ángel Sarabia. imlp: Applying multi-layer perceptrons to interval-valued data. *Neural Processing Letters*, 25(2):157–169, 2007.

[52] Kangjie Li, Yixiong Feng, Yicong Gao, and Jian Qiu. Hierarchical graph attention networks for semi-supervised node classification. *Applied Intelligence*, 50:3441–3451, 2020.

[53] Ling Chen, Jun Cui, Xing Tang, Yuntao Qian, Yansheng Li, and Yongjun Zhang. Rlpath: a knowledge graph link prediction method using reinforcement learning based attentive relation path searching and representation learning. *Applied Intelligence*, pages 1–12, 2021.

[54] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. *arXiv preprint arXiv:1806.03536*, 2018.

[55] Eugene L Lawler. *Combinatorial optimization: networks and matroids*. Courier Corporation, 2001.

[56] Geng Li, Murat Semerci, Bulent Yener, and Mohammed J Zaki. Graph classification via topological and label attributes. In *Proceedings of the 9th international workshop on mining and learning with graphs (MLG), San Diego, USA*, volume 2, 2011.

[57] Pinar Yanardag and SVN Vishwanathan. Deep graph kernels. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1365–1374, 2015.

[58] Asim Kumar Debnath, Rosa L Lopez de Compadre, Gargi Debnath, Alan J Shusterman, and Corwin Hansch. Structure-activity relationship of mutagenic

aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity. *Journal of medicinal chemistry*, 34(2):786–797, 1991.

[59] Karsten M Borgwardt, Cheng Soon Ong, Stefan Schönauer, SVN Vishwanathan, Alex J Smola, and Hans-Peter Kriegel. Protein function prediction via graph kernels. *Bioinformatics*, 21(suppl_1):i47–i56, 2005.

[60] Hannu Toivonen, Ashwin Srinivasan, Ross D King, Stefan Kramer, and Christoph Helma. Statistical evaluation of the predictive toxicology challenge 2000–2001. *Bioinformatics*, 19(10):1183–1193, 2003.

[61] Nikil Wale, Ian A Watson, and George Karypis. Comparison of descriptor spaces for chemical compound retrieval and classification. *Knowledge and Information Systems*, 14(3):347–375, 2008.

[62] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graphs over time: densification laws, shrinking diameters and possible explanations. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 177–187, 2005.

[63] Anshumali Shrivastava and Ping Li. A new space for comparing graphs. In *2014 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM 2014)*, pages 62–71. IEEE, 2014.

[64] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[65] Fei Yang, Huyin Zhang, and Shiming Tao. Simplified multilayer graph convolutional networks with dropout. *Applied Intelligence*, pages 1–16, 2021.

[66] Chih-Chung Chang and Chih-Jen Lin. Libsvm: A library for support vector machines. *ACM transactions on intelligent systems and technology (TIST)*, 2(3):1–27, 2011.

[67] Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[68] Zhiqiang Zhong, Cheng-Te Li, and Jun Pang. Hierarchical message-passing graph neural networks. *arXiv preprint arXiv:2009.03717*, 2020.

[69] Huaxiu Yao, Chuxu Zhang, Ying Wei, Meng Jiang, Suhang Wang, Junzhou Huang, Nitesh Chawla, and Zhenhui Li. Graph few-shot learning via knowledge transfer. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 6656–6663, 2020.

[70] Guohao Li, Matthias Muller, Ali Thabet, and Bernard Ghanem. Deepgcns: Can gcns go as deep as cnns? In *Proceedings of the IEEE International Conference on Computer Vision*, pages 9267–9276, 2019.

[71] Sebastian A Rios, Felipe Aguilera, J David Nuñez-Gonzalez, and Manuel Graña. Semantically enhanced network analysis for influencer identification in online social networks. *Neurocomputing*, 326:71–81, 2019.

[72] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864, 2016.

[73] Leonardo FR Ribeiro, Pedro HP Saverese, and Daniel R Figueiredo. struc2vec: Learning node representations from structural identity. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 385–394, 2017.

[74] Massimo Marchiori and Lino Possamai. Micro-macro analysis of complex networks. *PloS one*, 10(1):e0116670, 2015.

[75] Otmar Ertl. Probminhash–a class of locality-sensitive hash algorithms for the (probability) jaccard similarity. *IEEE Transactions on Knowledge and Data Engineering*, 2020.

[76] Duncan J Watts and Steven H Strogatz. Collective dynamics of 'small-world'networks. *nature*, 393(6684):440–442, 1998.

[77] Kim Dong-Young, Pengcheng Zhu, Xiao Wenli, and Lin Yen-Ting. Customer degree centrality and supplier performance: the moderating role of resource dependence. *Operations Management Research*, 13(1-2):22–38, 2020.

[78] Kurt Hornik, Maxwell Stinchcombe, Halbert White, et al. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[79] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[80] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. Learning convolutional neural networks for graphs. In *International conference on machine learning*, pages 2014–2023, 2016.

[81] Xiwang Yang, Yang Guo, Yong Liu, and Harald Steck. A survey of collaborative filtering based social recommender systems. *Computer communications*, 41:1–10, 2014.

[82] Eytan Adar and Lada A Adamic. Tracking information epidemics in blogspace. In *The 2005 IEEE/WIC/ACM International Conference on Web Intelligence (WI'05)*, pages 207–214. IEEE, 2005.

[83] David Kempe, Jon Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 137–146, 2003.

[84] Parham Moradi and Sajad Ahmadian. A reliability-based recommendation method to improve trust-aware recommender systems. *Expert Systems with Applications*, 42(21):7386–7398, 2015.

[85] Meeyoung Cha, Fabrício Benevenuto, Hamed Haddadi, and Krishna Gummadi. The world of connections and information flow in twitter. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, 42(4):991–998, 2012.

[86] Mario Cataldi and Marie-Aude Aufaure. The 10 million follower fallacy: audience size does not prove domain-influence on twitter. *Knowledge and Information Systems*, 44(3):559–580, 2015.

[87] Andriy Mnih and Russ R Salakhutdinov. Probabilistic matrix factorization. In *Advances in neural information processing systems*, pages 1257–1264, 2008.

[88] Hao Ma, Haixuan Yang, Michael R Lyu, and Irwin King. Sorec: social recommendation using probabilistic matrix factorization. In *Proceedings of the 17th ACM conf. on Information and knowledge management*, pages 931–940, 2008.

[89] Mohsen Jamali and Martin Ester. A matrix factorization technique with trust propagation for recommendation in social networks. In *Proceedings of the 4th ACM conference on Recommender systems*, pages 135–142, 2010.

[90] Tong Zhao, Chunping Li, Mengya Li, Qiang Ding, and Li Li. Social recommendation incorporating topic mining and social trust analysis. In *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*, pages 1643–1648, 2013.

[91] Bo Yang, Yu Lei, Jiming Liu, and Wenjie Li. Social collaborative filtering by trust. *IEEE transactions on pattern analysis and machine intelligence*, 39(8):1633–1647, 2016.

[92] Hao Ma, Irwin King, and Michael R Lyu. Learning to recommend with explicit and implicit social relations. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2(3):1–19, 2011.

[93] Hao Ma, Dengyong Zhou, Chao Liu, Michael R Lyu, and Irwin King. Recommender systems with social regularization. In *Proceedings of the 4th ACM international conference on Web search and data mining*, pages 287–296, 2011.

[94] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative filtering. In *Proceedings of the 26th international conference on world wide web*, pages 173–182, 2017.

[95] Wenqi Fan, Qing Li, and Min Cheng. Deep modeling of social relations for recommendation. In *Thirty-Second AAAI Conference on Artificial Intelligence (AAAI-18)*, pages 8075–8076. AAAI press, 2018.

[96] Rianne van den Berg, Thomas N Kipf, and Max Welling. Graph convolutional matrix completion. *arXiv preprint arXiv:1706.02263*, 2017.

[97] Wenqi Fan, Tyler Derr, Yao Ma, Jianping Wang, Jiliang Tang, and Qing Li. Deep adversarial social recommendation. *arXiv preprint arXiv:1905.13160*, 2019.

[98] Wenqi Fan, Yao Ma, Qing Li, Yuan He, Eric Zhao, Jiliang Tang, and Dawei Yin. Graph neural networks for social recommendation. In *The World Wide Web Conference*, pages 417–426, 2019.

[99] Junliang Yu, Min Gao, Hongzhi Yin, Jundong Li, Chongming Gao, and Qinyong Wang. Generating reliable friends via adversarial training to improve social recommendation. In *2019 IEEE International Conference on Data Mining (ICDM)*, pages 768–777. IEEE, 2019.

[100] Le Wu, Peijie Sun, Yanjie Fu, Richang Hong, Xiting Wang, and Meng Wang. A neural influence diffusion model for social recommendation. In *Proceedings of the 42nd international ACM SIGIR conference on research and development in information retrieval*, pages 235–244, 2019.

[101] Manqing Dong, Lina Yao, Xianzhi Wang, Xiwei Xu, and Liming Zhu. Adversarial dual autoencoders for trust-aware recommendation. *Neural Computing and Applications*, pages 1–11, 2021.

[102] Wenqi Fan, Tyler Derr, Yao Ma, Jianping Wang, Jiliang Tang, and Qing Li. Deep adversarial social recommendation. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pages 1351–1357. International Joint Conferences on Artificial Intelligence Organization, 7 2019.

[103] Bam Bahadur Sinha and R Dhanalakshmi. Dnn-mf: Deep neural network matrix factorization approach for filtering information in multi-criteria recommender systems. *Neural Computing and Applications*, pages 1–15, 2022.

[104] Chong Chen, Min Zhang, Yiqun Liu, and Shaoping Ma. Neural attentional rating regression with review-level explanations. In *Proceedings of the 2018 World Wide Web Conference*, pages 1583–1592, 2018.

[105] Chenhao Li, Jiyin Zhang, Amruta Kale, Xiang Que, Sanaz Salati, and Xiaogang Ma. Toward trust-based recommender systems for open data: A literature review. *Information*, 13(7):334, 2022.

[106] Wanita Sherchan, Surya Nepal, and Cecile Paris. A survey of trust in social networks. *ACM Computing Surveys (CSUR)*, 45(4):1–33, 2013.

[107] Oumaima Stitini, Soulaimane Kaloun, and Omar Bencharef. Towards the detection of fake news on social networks contributing to the improvement of trust and transparency in recommendation systems: Trends and challenges. *Information*, 13(3):128, 2022.

[108] Moon-Hee Park, Jin-Hyuk Hong, and Sung-Bae Cho. Location-based recommendation system using bayesian user's preference model in mobile devices. In *International conference on ubiquitous intelligence and computing*, pages 1130–1139. Springer, 2007.

[109] Weiwei Yuan, Donghai Guan, Young-Koo Lee, Sungyoung Lee, and Sung Jin Hur. Improved trust-aware recommender system using small-worldness of trust networks. *Knowledge-Based Systems*, 23(3):232–238, 2010.

[110] Lajanugen Logeswaran, Honglak Lee, and Dragomir Radev. Sentence ordering and coherence modeling using recurrent neural networks. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[111] Arnon Rungsawang, Komthorn Puntumapon, and Bundit Manaskasemsak. Unbiasing the link farm effect in pagerank computation. In *21st International Conference on Advanced Information Networking and Applications (AINA'07)*, pages 924–931. IEEE, 2007.