Abstract

Graphs are effective models for illustrating complex connections in various contexts, including social networks, citation networks, and biological systems. In this study, the performance of Graph Neural Networks(GNN) for two main tasks is being explored: Node classification and Community detection. Community detection on complex networks is an emerging area that helps to understand the underlying structures and functions of interconnected systems across various domains. The structural and semantic information are captured using the various GNN models. This paper uses four GNN models for node classification: GCN, GAT, GraphSage, and Deep Graph Infomax. The embeddings of these models are clustered to detect the communities by using spectral clustering, k-means clustering, and the Louvain algorithm. The GNN model's performance was assessed in terms of computational complexity, accuracy of clustering, and modularity. The results demonstrate that GNN models, particularly GCN, yield competitive performance in community detection tasks on citation networks. The effectiveness of each model varies across different datasets and community detection algorithms.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction to Graph Neural Networks (GNNs)

In contemporary data analysis, the study of graph-structured data has emerged as a pivotal domain across various fields, including social networks, e-commerce, bioinformatics, and recommendation systems. Graphs serve as a powerful abstraction for representing complex relationships and interactions between entities, where nodes denote entities and edges signify relationships or interactions between them. This flexible and intuitive framework enables the modelling of intricate systems and the capture of interdependencies among entities, making graph analysis indispensable for understanding complex real-world phenomena.

Graph Neural Networks (GNNs) have emerged as a potent tool for analyzing graph-structured data, offering capabilities to learn complex patterns and relationships directly from graphs. Inspired by the success of deep learning in traditional structured data domains, GNNs extend neural network archi-

tectures to operate directly on graph-structured data, enabling the learning of node and graph representations. By leveraging techniques such as message passing and graph convolutions, GNNs can effectively capture both local and global structural information within graphs, facilitating tasks such as node classification, link prediction, and graph clustering.

1.2 Evaluation of GNN Methods

The objective of this study is to conduct a comprehensive comparative analysis of state-of-the-art GNN methods applied to three benchmark datasets commonly used in the literature: Cora, Citeseer, and Pubmed. These datasets represent citation networks, where nodes correspond to documents and edges represent citation relationships between them. The choice of datasets enables us to assess the generalization capabilities of GNN methods across different domains and network characteristics.

In this context, we will evaluate four prominent GNN methods: Graph Convolutional Networks (GCN), Graph Attention Networks (GAT), GraphSAGE, and Deep Graph Infomax. Each of these methods offers unique mechanisms for aggregating information from neighboring nodes and capturing global graph properties. By systematically evaluating these methods on the benchmark datasets, we aim to elucidate their relative strengths and weaknesses for graph analysis tasks across diverse domains.

1.3 Detailing Graph Neural Networks (GNNs)

Graph Neural Networks (GNNs) represent a class of machine learning models designed to operate directly on graph-structured data. Unlike traditional

neural networks, which operate on fixed-dimensional vectors, GNNs are tailored to handle inputs of variable size and topology. They accomplish this by iteratively updating node representations through message passing and aggregation mechanisms.

GCNs are among the earliest and most widely studied GNN architectures. They extend the concept of convolutional neural networks (CNNs) to graphs by defining a convolution operation that operates on the neighborhood of each node. GCNs aggregate information from neighboring nodes by computing weighted averages or summations of their feature vectors. This enables them to capture local graph structures and learn node representations that encode both local and global information.

GATs introduce attention mechanisms to GNNs, allowing nodes to selectively attend to their neighbors based on learned attention coefficients. By assigning different importance weights to neighboring nodes, GATs can focus on relevant information and adaptively aggregate features from the neighborhood. This attention mechanism enables GATs to capture more fine-grained and context-aware representations of nodes, leading to improved performance in tasks requiring precise localization of information.

GraphSAGE is a semi-supervised GNN architecture designed for scalable and efficient representation learning on large-scale graphs. Unlike GCNs, which aggregate information from fixed-size neighborhoods, GraphSAGE employs a sampling-based approach to generate node embeddings by aggregating features from sampled neighborhood nodes. This enables GraphSAGE to scale to large graphs while capturing diverse and informative node representations. Additionally, GraphSAGE supports inductive learning, allowing it to generalize to unseen nodes or graphs.

Deep Graph Infomax is an unsupervised GNN architecture based on the

principle of maximizing mutual information between local and global graph representations. It leverages contrastive learning to learn informative node representations by maximizing the agreement between the embeddings of a node and its context (local and global neighborhood). By maximizing mutual information, Deep Graph Infomax aims to capture meaningful graph structures and learn representations that preserve relevant information for downstream tasks.

1.4 Evaluation of GNN Methods on Benchmark Datasets

To evaluate the performance of GCN, GAT, GraphSAGE, and Deep Graph Infomax, we conduct experiments on three widely used benchmark datasets: Cora, Citeseer, and Pubmed. These datasets represent citation networks, where nodes correspond to documents and edges represent citation relationships between them. The task is to predict the category or label of each document based on its citation links and associated features.

The Cora dataset consists of scientific publications categorized into one of seven research areas. Each document is represented by a feature vector encoding its word frequencies, and the goal is to predict the research area based on citation links and features. Cora is a relatively small dataset, with 2,708 nodes and 5,429 edges.

The Citeseer dataset is similar to Cora and comprises scientific publications categorized into six research areas. Like Cora, each document is represented by a feature vector encoding word frequencies. Citeseer is slightly larger than Cora, with 3,327 nodes and 4,732 edges.

The Pubmed dataset contains citations between biomedical publications,

where each publication is associated with a MeSH (Medical Subject Headings) term. The task is to predict the MeSH term of each publication based on its citation links and features. Pubmed is the largest dataset among the three, with 19,717 nodes and 44,338 edges.

Chapter 2

Problem Definition

Graph-structured data has become increasingly prevalent across various domains, ranging from social networks and e-commerce platforms to biological networks and citation databases. Understanding the underlying relationships and patterns within these graphs is crucial for extracting meaningful insights and making informed decisions. However, analyzing such complex structures poses unique challenges that traditional data analysis techniques struggle to address effectively.

In this project, our primary objective is to tackle the analysis and evaluation of graph-structured data using advanced techniques, specifically focusing on Graph Neural Network (GNN) methods and traditional clustering algorithms. The overarching goal is to compare the performance of these methods on benchmark datasets and provide insights into their suitability for various graph analysis tasks.

Graph Neural Networks (GNNs) have emerged as a powerful framework for learning representations directly from graph-structured data. These models leverage graph topology and node attributes to capture complex patterns and relationships within the data. By extending neural network architec-

tures to operate on graphs, GNNs enable tasks such as node classification, link prediction, and graph clustering. However, despite their effectiveness, selecting the most suitable GNN architecture for a given dataset and task remains a challenging endeavor.

In addition to GNN methods, traditional clustering algorithms play a crucial role in graph analysis. These algorithms aim to partition nodes into cohesive groups based on their structural similarities or shared attributes. By identifying clusters within the graph, these algorithms facilitate tasks such as community detection, anomaly detection, and network visualization. However, the performance of clustering algorithms can vary significantly depending on the characteristics of the graph and the chosen algorithm parameters.

To address these challenges, we aim to conduct a comprehensive comparative analysis of GNN methods and traditional clustering algorithms on benchmark datasets. Specifically, we will evaluate the performance of GNN methods, including Graph Convolutional Networks (GCN), Graph Attention Networks (GAT), GraphSAGE, and Deep Graph Infomax, on datasets such as Cora, Citeseer, and Pubmed. Additionally, we will assess the effectiveness of traditional clustering algorithms, such as Spectral Clustering, K-means, and the Louvain Method, on the same datasets.

By systematically evaluating these methods and algorithms, we seek to provide insights into their strengths, weaknesses, and applicability to various graph analysis tasks. we aim to identify key factors that influence the performance of these techniques, such as dataset characteristics, algorithm parameters, and evaluation metrics.our goal is to empower researchers and practitioners with the knowledge and tools necessary to make informed decisions when analyzing graph-structured data in real-world applications.

Chapter 3

Related Work

Numerous research papers have investigated the application of Graph Neural Networks (GNNs) and traditional clustering algorithms in graph analysis tasks. A study conducted introduced Graph Convolutional Networks (GCNs) [1], which have since become a cornerstone in graph representation learning. They demonstrated the effectiveness of GCNs in tasks such as semi-supervised node classification and graph-level prediction, setting a benchmark for subsequent research in the field.

An introduction to Graph Attention Networks (GATs), leveraged attention mechanisms to selectively aggregate information from neighboring nodes [2]. Their study showed that GATs outperform GCNs in capturing local and global dependencies within graphs, particularly in tasks where fine-grained node representations are essential, such as in recommendation systems and social network analysis.

In addition to GNN methods, traditional clustering algorithms have also been extensively studied in the context of graph analysis. The Louvain Method for community detection is a popular and scalable algorithm based on modularity optimization [3]. Their study demonstrated the effectiveness of

the Louvain Method in identifying communities within large-scale networks, laying the groundwork for subsequent research in community detection algorithms.

Furthermore, the study of Spectral Clustering provided a comprehensive overview of Spectral Clustering algorithms and their applications in graph partitioning and clustering tasks [4]. Their analysis revealed the strengths and limitations of Spectral Clustering methods in handling various types of graphs, emphasizing the importance of considering graph topology and spectral properties in clustering analysis.

The Louvain Method also excels in community detection for several reasons. It adapts to varying community sizes without needing the number of communities predefined, unlike K-means [5]. By maximizing modularity, it efficiently identifies meaningful community structures within the graph. Its greedy optimization approach ensures computational efficiency, particularly for large networks. These advantages make the Louvain Method a preferred choice for uncovering community structures in dynamic or poorly understood networks.

Graph Convolutional Networks (GCNs), Graph Attention Networks (GATs), GraphSAGE, and Deep Graph Infomax were pivotal in analyzing graph-structured data. GCNs excel in capturing both local and global graph structures, making them suitable for tasks like node classification and link prediction [6]. GATs leverage attention mechanisms to selectively aggregate information from neighboring nodes, enhancing their performance in tasks requiring fine-grained node representations. GraphSAGE learns node embeddings by aggregating information from a node's local neighborhood, enabling scalable representation learning for large graphs [7]. Deep Graph Infomax maximizes the mutual information between local patches of the graph, fa-

cilitating unsupervised representation learning. These GNN methods were instrumental in extracting meaningful insights from the benchmark datasets, providing a comprehensive understanding of the underlying graph structures and enabling effective comparative analysis across diverse domains [8] [9].

These research papers collectively provide valuable insights into the state-of-the-art techniques and methodologies in graph analysis, serving as foundational works for our comparative study of GNN methods and clustering algorithms on benchmark datasets. By building upon these existing works, our study aims to contribute to the ongoing research efforts in the field and provide practical guidance for selecting appropriate methods in real-world applications.

Chapter 4

Requirements

The design of this project contains both hardware and software. The specifications are listed below.

4.1 Hardware

The project requires access to a computer system with adequate hardware capabilities to support the computational requirements of running Graph Neural Network (GNN) models and clustering algorithms. A system with a multi-core processor and sufficient memory is recommended to handle the computational complexity of training and evaluating models on large-scale graph datasets.

4.1.1 Processor

A multi-core processor, preferably with multi-threading capabilities, is necessary to expedite the training and inference processes of GNN models and clustering algorithms. Processors with higher clock speeds and parallel processing capabilities can significantly reduce the computational time required

for graph analysis tasks. Recommended processors include Intel Core i5 or above for traditional systems and M1 and above chips for Apple systems, as they provide efficient performance for graph analysis tasks.

4.1.2 Memory

Sufficient memory (RAM) is essential to accommodate the large-scale graph datasets and intermediate computations involved in GNN training and clustering algorithms. Adequate memory ensures smooth execution and prevents performance bottlenecks caused by data loading and manipulation operations. A recommended memory capacity of 8GB or above is advised to handle the computational demands of graph analysis tasks effectively.

4.2 Software

The project requires a software environment conducive to developing, training, and evaluating GNN models and clustering algorithms. Essential software components include Python programming language, along with relevant libraries and frameworks such as TensorFlow, PyTorch, Scikit-learn, NetworkX, and NumPy. Additionally, tools for data visualization and analysis, such as Matplotlib and Pandas, are indispensable for interpreting and presenting results effectively.

4.2.1 Google Colaboratory

Google Colaboratory (Colab) serves as a convenient and accessible platform for running Python code, particularly for projects involving machine learning and data analysis. Colab provides free cloud-based access to GPUs and TPUs, enabling accelerated model training and inference without the need

for dedicated hardware resources. Leveraging Colab's collaborative features, such as sharing notebooks and real-time collaboration, enhances productivity and facilitates teamwork in project development and experimentation.

Advantages of Google Colaboratory

Google Colab's remarkable features attract data scientists and researchers from all over the world. It offers a flexible platform for code writing and execution,as well as interactive machine learning analysis. One of the primary benefits of Google Colab is its ability to provide GPUs and TPUs for free, which can be extremely useful when training large neural networks. Furthermore, because it is a cloud-based service, it enables real-time collaboration, making it an excellent tool for our project.

4.2.2 Visual Studio Code (VSCode)

VSCode is a versatile and lightweight code editor that offers a range of features to streamline development tasks, including syntax highlighting, code completion, and debugging capabilities. Its extensible architecture allows integration with various programming languages and development tools, making it suitable for Python development and experimentation with GNN models and clustering algorithms. VSCode provides a user-friendly interface for writing, running, and debugging code, enhancing the development experience for project collaborators.

Overall, meeting these requirements ensures a conducive environment for conducting graph analysis tasks, leveraging the computational capabilities of modern hardware and software tools to achieve accurate and efficient results.

4.2.3 Installation and Usage

Google Colab works in the browser and does not require any setup or installation. Python code can be written and executed directly in the browser—ideal for machine learning, data analysis, and visualization. To use Colab, all you need is a Google account. You can either create new Colab notebooks, import existing Jupyter notebooks from GitHub, or upload them from your local machine. The notebooks are saved to your Google Drive account, allowing for seamless transitions between devices while also ensuring the security of your data.

To install Visual Studio Code (VSCode), visit the official website and download the appropriate installer for your operating system. Once the installation is complete, launch VSCode and install the Python extension from the marketplace to enable Python development features. With the Python extension installed, you can create and edit Python files directly within VSCode, benefiting from features like syntax highlighting, code completion, and linting. Additionally, VSCode offers built-in support for version control systems like Git, allowing you to manage your codebase seamlessly. Overall, VSCode provides a user-friendly and customizable environment for Python development, making it an ideal choice for writing, running, and debugging code efficiently.

4.2.4 Gephi

Gephi, a powerful open-source network visualization and exploration tool, is employed to complement the analysis process. Gephi offers advanced capabilities for visualizing and analyzing graph structures, particularly in scenarios where proper community visualization is crucial. By importing graph

data into Gephi, researchers can visually explore the community structure of networks, identify clusters of nodes with shared attributes or connections, and gain insights into the overall network topology. Gephi's intuitive interface allows users to customize the visualization settings, apply various layout algorithms to arrange nodes and edges, and interactively explore the graph in real-time. Additionally, Gephi provides tools for measuring network metrics, detecting communities, and performing clustering analysis, enabling researchers to validate and refine their findings from computational analysis. Overall, Gephi serves as a valuable tool for enhancing the interpretability of graph analysis results and facilitating communication of complex network structures to broader audiences.

Chapter 5

Proposed System

The Project aims to provide a comprehensive platform for conducting graph analysis tasks using Graph Neural Networks (GNNs) and clustering algorithms. The system will facilitate the development, training, and evaluation of GNN models and clustering algorithms on graph-structured data, enabling researchers to gain insights into complex network structures and relationships. Key components of the proposed system include:

5.0.1 Graph Data Management

- The system will support the ingestion and management of graph datasets from various sources, including standard formats such as GraphML, GEXF, and CSV.
- Users will be able to preprocess and transform raw graph data into appropriate formats for training GNN models and clustering algorithms.

5.0.2 Model Development and Training

- The system will provide tools and libraries for developing and training GNN models, including popular architectures such as Graph Convolutional Networks (GCNs), Graph Attention Networks (GATs), and GraphSAGE.
- Users can customize model architectures, hyperparameters, and training procedures to suit their specific research goals and dataset characteristics.

5.0.3 Clustering Algorithms

- The system will incorporate traditional clustering algorithms such as K-means, spectral clustering, and the Louvain Method for community detection and network partitioning.
- Users can apply clustering algorithms to analyze graph structures and identify cohesive groups of nodes based on various criteria.

5.0.4 Evaluation and Validation

- The system will provide evaluation metrics and visualization tools to assess the performance of GNN models and clustering algorithms on benchmark datasets.
- Users can compare different methods based on metrics such as accuracy, modularity, conductance, and community visualizations to validate their findings.

5.0.5 Integration with Visualization Tools

- The system will integrate with visualization tools such as Gephi to enhance the interpretability of analysis results and facilitate visual exploration of graph structures.

- Users can seamlessly import graph data from the system into visualization tools for interactive exploration and communication of complex network relationships.

5.0.6 Collaborative Workflows

- The system will support collaborative workflows, allowing multiple users to collaborate on model development, experimentation, and result interpretation.
- Users can share notebooks, code snippets, and analysis results within the system, fostering collaboration and knowledge sharing among research teams.

5.0.7 Scalability and Performance

- The system will be designed to handle large-scale graph datasets and computationally intensive tasks, leveraging distributed computing resources and optimized algorithms.
- Users can scale up the system's resources dynamically to accommodate growing data volumes and analysis demands.

Overall, the proposed system will provide a user-friendly and scalable platform for conducting graph analysis tasks, empowering researchers to explore complex network structures, extract meaningful insights, and advance the state-of-the-art in graph analytics.

In this experimental study, we employed a comprehensive array of graph analysis techniques to explore the intricate structures and relationships within graph-structured data. Specifically, we utilized four state-of-the-art Graph Neural Network (GNN) models, namely Graph Convolutional Networks (GCNs),

Graph Attention Networks (GATs), Deep Graph Infomax, and GraphSAGE, to capture and learn from the complex interdependencies present in the datasets. These models were applied to three benchmark datasets, namely Cora, Citeseer, and Pubmed, each representing citation networks with distinct characteristics. Additionally, we leveraged traditional clustering algorithms, including K-means clustering (with the optimal value of K determined using the elbow method), Spectral clustering, and the Louvain Method, to partition the graphs into cohesive communities. Notably, we utilized the output features generated by the GNN models (GAT, GCN, Deep Graph Infomax, and GraphSAGE) as input for the clustering methods, enabling a comprehensive analysis of the effectiveness of both the GNN models and clustering algorithms in capturing the underlying structure of the datasets. Through this multifaceted approach, we aimed to gain deeper insights into the performance and capabilities of different graph analysis techniques across diverse datasets and tasks.

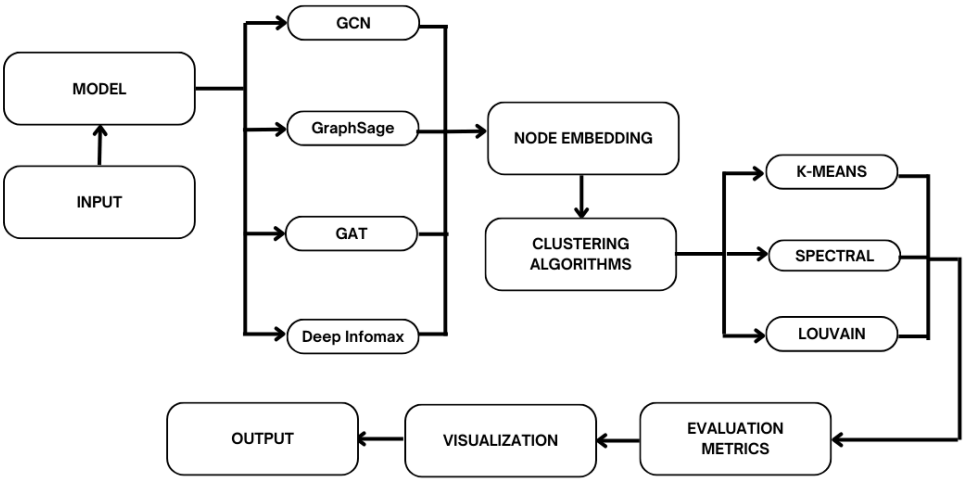

Figure 5.1: Proposed Methodology

5.1 Graph Neural Network Techniques

5.1.1 Graph Convolutional Networks (GCN)

Graph Convolutional Networks (GCNs) are a class of GNN models that extend convolutional neural networks (CNNs) to operate directly on graph-structured data. GCNs leverage localized filters to aggregate information from a node's neighbourhood, similar to the convolutional operation in CNNs. By iteratively applying graph convolutions across multiple layers, GCNs can capture both local and global structural information within the graph.

Advantages

- Localized Aggregation: GCNs aggregate information from neighboring nodes using localized filters, enabling them to capture local graph structures efficiently.
- Shared Parameters: The use of shared parameters across layers allows GCNs to learn hierarchical representations of the graph, capturing increasingly abstract features in deeper layers.
- Scalability: GCNs can be applied to large-scale graphs with millions of nodes and edges, as the aggregation operation is localized and computationally efficient.

How It Works

1. Initialization: Initialize node embeddings for each node in the graph.
2. Message Passing: Aggregate information from neighbouring nodes by computing weighted sums of their features.

3. Aggregation: Apply a graph convolution operation to aggregate information from the node's neighbourhood, similar to the convolutional operation in CNNs.

4. Non-linear Transformation: Apply a non-linear activation function to the aggregated features to capture complex relationships.

5. Update: Update the node embeddings using the non-linear transformed features, incorporating both local and global context.

6. Iterative Process: Iterate the message passing and aggregation steps for multiple layers to capture hierarchical features and relationships in the graph.

Algorithm 1 Graph Convolutional Network (GCN)

Input: Graph data, number of epochs
Output: Predicted labels for each node
Algorithm GCN:
Initialize weights for each layer: $W^{(l)}$, $l = 1, \ldots, L$
Initialize biases for each layer: $b^{(l)}$, $l = 1, \ldots, L$
Initialize node embeddings $H^{(0)} = X$
for epoch = 1 num_epochs **do**
    for $l = 1$ L **do**
        Compute the graph convolutional operation:
        $Z^{(l)} = \text{activation}(A \cdot H^{(l-1)} \cdot W^{(l)} + b^{(l)})$
        Update node embeddings using $Z^{(l)}$:
        $H^{(l)} = Z^{(l)}$
    end for
end for
end GCN Algorithm

This is a high-level algorithm for training a Graph Convolutional Network (GCN) for a node classification task. The algorithm outlines the steps involved in initializing the weights and biases, computing graph convolu-

tional operations, and updating node embeddings over a specified number of epochs.

5.1.2 Graph Attention Networks (GAT)

Graph Attention Networks (GATs) are a type of GNN model that leverages attention mechanisms to selectively aggregate information from neighboring nodes in a graph. Unlike traditional GNNs, which assign equal importance to all neighboring nodes, GATs dynamically compute attention coefficients to weigh the importance of each neighbor during the aggregation process. This allows GATs to focus on relevant nodes and adaptively adjust the aggregation weights based on the node features and their relationships.

Advantages

- Selective Attention: GATs can selectively attend to relevant nodes in the neighborhood, enabling them to capture fine-grained patterns and relationships in the graph.
- Adaptive Weights: The use of attention mechanisms allows GATs to adaptively adjust the aggregation weights based on the node features and their importance, leading to improved performance in tasks requiring nuanced node representations.
- Scalability: GATs can be applied to large-scale graphs efficiently, as the attention mechanism enables localized computation and reduces the computational complexity compared to traditional message passing methods.

How It Works

1. Initialization: Initialize node embeddings for each node in the graph.

2. Attention Computation: Compute attention coefficients for each node by aggregating information from its neighbors and applying a softmax function to obtain normalized attention scores.

3. Aggregation: Aggregate information from neighboring nodes based on the attention coefficients, weighted by their importance.

4. Update: Update the node embeddings using the aggregated information, incorporating both local and global context.

5. Iterative Process: Iterate the attention computation and aggregation steps for multiple layers to capture hierarchical features and relationships in the graph.

Algorithm 2 Graph Attention Network (GAT)

Input: Graph data, number of epochs
Output: Predicted labels for each node
Algorithm GAT:
Initialize weights for each attention head and layer: $W_k^{(l)}$, $a_k^{(l)}$, $b^{(l)}$, for each attention head k and layer l
Initialize node embeddings $H^{(0)} = X$
for epoch = 1 num_epochs **do**
    for $l = 1$ L **do**
        for $k = 1$ K **do**
            Compute attention coefficients:
            $e_{ij}^{(l,k)} = \text{LeakyReLU}((W_k^{(l)} \cdot [H_i^{(l-1)} || H_j^{(l-1)}]) + a_k^{(l)})$
            $\alpha_{ij}^{(l,k)} = \text{softmax}(e_{ij}^{(l,k)})$
            Compute neighbor-aware node embeddings:
            $Z_i^{(l,k)} = \sum(\alpha_{ij}^{(l,k)} \cdot W_k^{(l)} \cdot H_j^{(l-1)})$ for all j in neighbors(i)
        end for

Algorithm GAT (continued)

Compute the output of the attention head:
$Z_i^{(l)} = \text{concat}(Z_i^{(l,1)}, \ldots, Z_i^{(l,K)})$
$Z_i^{(l)} = \text{activation}(\sum(Z_i^{(l)} \cdot W^{(l)}) + b^{(l)})$
Update node embeddings using $Z^{(l)}$:
$H_i^{(l)} = Z_i^{(l)}$

end GAT Algorithm
for each node i **do**
    Compute the final output:
    $\hat{y}_i = \text{softmax}(H_i^{(L)} \cdot W^{(L)} + b^{(L)})$
end for=0

This algorithm describes the training procedure for a Graph Attention Network (GAT). GAT is a type of neural network architecture designed for handling graph-structured data, where attention mechanisms are employed to weigh the contributions of neighbouring nodes differently during message passing.

5.1.3 GraphSAGE

GraphSAGE (Graph Sample and Aggregation) is a GNN model that learns node embeddings by sampling and aggregating information from a node's local neighborhood. Unlike traditional GNNs that aggregate information from fixed-size neighborhoods, GraphSAGE can handle graphs of varying sizes and structures by sampling a fixed number of neighbors for each node and aggregating their features.

Advantages

- Scalability: GraphSAGE can handle large-scale graphs with varying sizes and structures by sampling a fixed number of neighbors for each node.

- Flexibility: By sampling and aggregating information from a node's local neighborhood, GraphSAGE can capture diverse and complex graph structures.
- Generalization: GraphSAGE can generalize well to unseen nodes and graphs by learning robust representations from local neighborhoods.

How It Works

1. Neighbor Sampling: Sample a fixed number of neighbors for each node in the graph.
2. Aggregation: Aggregate information from the sampled neighbors using a neighborhood aggregation function (e.g., mean or max pooling).
3. Representation Learning: Learn node embeddings by applying a neural network to the aggregated features.
4. Iterative Sampling: Iterate the neighbor sampling and aggregation steps for multiple layers to capture hierarchical features and relationships in the graph.

Algorithm 3 GraphSAGE Algorithm

Input: Graph data, number of epochs, number of neighbors K
Output: Updated node embeddings
Algorithm GraphSAGE:
Initialize weights for each layer: $W^{(l)}$ for each layer l
Initialize node embeddings $H^{(0)} = X$
for epoch = 1 num_epochs **do**
    for each node v in G **do**
        Sample K neighbors of node v
        for $l = 1$ L **do**
            Aggregate features from neighbors:
            $H_v^{(l)} = \text{aggregate}(W^{(l)}, \{H_v^{(l-1)}, H_u^{(l-1)} \text{ for each neighbor } u\})$

Algorithm GraphSAGE (continued)

Apply activation function:
$H_v^{(l)} = \text{activation}(H_v^{(l)})$

end GraphSAGE Algorithm =0

This algorithm describes the training process for a GraphSAGE (Graph Sample and Aggregation) model. GraphSAGE is a graph neural network architecture designed for node classification tasks, where nodes are represented as feature vectors and information from neighboring nodes is aggregated to update node representations.

5.1.4 Deep Graph Infomax

Deep Graph Infomax is a self-supervised learning framework for learning representations of graph-structured data. It maximizes the mutual information between local patches of the graph, encouraging the model to capture meaningful and discriminative features from the data. Deep Infomax does not require labeled data for training, making it suitable for unsupervised representation learning tasks.

Advantages

- Unsupervised Learning: Deep Graph Infomax learns representations of graph data in an unsupervised manner, without the need for labeled data.

- Maximizing Mutual Information: By maximizing the mutual information between local patches of the graph, Deep Graph Infomax encourages the model to capture diverse and informative features.

- Robust Representation Learning: Deep Graph Infomax learns robust representations that capture both local and global structural information within the graph, facilitating downstream tasks such as node classification and link prediction.

How It Works

1. Local Patch Extraction: Extract local patches of the graph centered around each node.
2. Encoder Network: Encode each local patch using a neural network to obtain a latent representation.
3. Mutual Information Estimation: Maximize the mutual information between the latent representations of local patches and global summary statistics of the graph.
4. Representation Learning: Learn robust representations of graph data that capture meaningful features and relationships.
5. Unsupervised Training: Train the model using self-supervised learning objectives, without requiring labeled data.

Algorithm 4 Deep Graph Infomax Algorithm

Input: Graph data, number of epochs, number of neighbors K
Output: Updated weights
Algorithm Deep Graph Infomax:
Initialize weights for each layer: $W^{(l)}$ for each layer l
Initialize node embeddings $H^{(0)} = X$

Algorithm Deep Graph Infomax (continued)

for epoch = 1 num_epochs **do**
  for each node v in G **do**
    Sample K neighbors of node v
    for $l = 1$ L **do**
      Aggregate features from neighbors:
      $H_v^{(l)} = \text{aggregate}(W^{(l)}, \{H_v^{(l-1)}, H_u^{(l-1)} \text{ for each neighbor } u\})$
    end for
  end for
  Compute global graph summary:
  $Z = \text{summary}(H^{(L)})$
  for each node v in G **do**
    Compute local graph summary:
    $Z_v = \text{summary}(\{H_v^{(L)}, H_u^{(L)} \text{ for each neighbor } u\})$
    Compute contrastive loss:
    $\text{Loss}_v = \text{contrastive\_loss}(Z, Z_v)$
  end for
  Update weights using gradient descent on contrastive loss
end for
end Deep Graph Infomax Algorithm

This algorithm describes a training process for a graph neural network using a self-supervised learning framework, particularly similar to Deep Graph Infomax. It's a technique for learning meaningful representations of nodes in a graph without explicit supervision.

5.2 Graph Clustering Algorithms

Based on the spectrum characteristics of the graph Laplacian matrix, spectral clustering was used to divide the datasets into disjoint clusters. As a centroid-based partitioning approach, K-means clustering was applied, which iteratively assigned nodes to clusters based on closeness to cluster centroids. Using Louvain clustering, a measure of the quality of the resulting partitions, communities within the graphs were found by optimising modularity.

5.2.1 K-means Clustering

K-means clustering is a popular unsupervised machine learning algorithm used for partitioning a dataset into K clusters. The algorithm iteratively assigns data points to the nearest cluster centroid and updates the centroids based on the mean of the data points assigned to each cluster. The process continues until convergence, where the centroids no longer change significantly.

Elbow method

In our analysis, we utilized the elbow method to determine the optimal number of clusters (K) for K-means clustering. The elbow method involves plotting the within-cluster sum of squares (WCSS) against the number of clusters and identifying the "elbow" point where the rate of decrease in WCSS slows down significantly. This point indicates the optimal number of clusters, as adding more clusters beyond this point does not lead to a significant reduction in WCSS. By leveraging the elbow method, we were able to make informed decisions about the number of clusters to use in our clustering analysis, ensuring that the clustering results were both meaningful and interpretable.

Advantages

- Simplicity: K-means is easy to understand and implement, making it widely used for clustering tasks.
- Efficiency: The algorithm has a computational complexity of O(n * K * d), making it efficient for large datasets with moderate dimensions.
- Scalability: K-means can handle large datasets with ease, making it suitable for real-world applications.

How It Works:

1. Initialization: Randomly initialize K cluster centroids.
2. Assignment: Assign each data point to the nearest cluster centroid.
3. Update: Update the cluster centroids by computing the mean of the data points assigned to each cluster.
4. Convergence: Repeat the assignment and update steps until convergence or a maximum number of iterations is reached.

Algorithm 5 K-Means Clustering Algorithm

Input: Data points X, number of clusters K, maximum iterations
Output: Cluster assignments $c_1, c_2, ..., c_n$ and cluster centroids $\mu^{(1)}, \mu^{(2)}, ..., \mu^{(K)}$
Algorithm K-Means:
Randomly initialize K cluster centroids: $\mu^{(1)}, \mu^{(2)}, ..., \mu^{(K)}$ from the data points in X
repeat
    for each data point x_i **do**
        Assign x_i to the nearest cluster centroid:
        $c_i = \arg\min_k ||x_i - \mu^{(k)}||^2$
    end for
    for each cluster k from 1 to K **do**
        Update centroid $\mu^{(k)}$ as the mean of all data points assigned to cluster k:
        $\mu^{(k)} = \frac{1}{|C_k|} \sum_{i=1}^{n} (x_i \text{ if } c_i = k)$
    end for
until Convergence or maximum iterations reached
Output the cluster assignments $\{c_1, c_2, ..., c_n\}$ and cluster centroids $\{\mu^{(1)}, \mu^{(2)}, ..., \mu^{(K)}\}$

5.2.2 Spectral Clustering

Spectral clustering is a graph-based clustering algorithm that partitions the data into clusters based on the eigenvectors of a similarity matrix derived

from the data. The algorithm first constructs a similarity matrix representing pairwise similarities between data points, then computes the eigenvectors corresponding to the smallest eigenvalues of the Laplacian matrix derived from the similarity matrix. Finally, it uses the eigenvectors to perform K-means clustering in a reduced-dimensional space.

Advantages

- Flexibility: Spectral clustering can uncover complex non-linear structures in the data, making it suitable for a wide range of clustering tasks.
- Robustness: The algorithm is robust to noise and outliers, as it considers the global structure of the data.
- No Assumptions: Spectral clustering does not make any assumptions about the shape or distribution of the clusters, making it versatile for various types of data.

How It Works

1. Similarity Matrix: Compute a similarity matrix based on pairwise similarities between data points.
2. Laplacian Matrix: Compute the Laplacian matrix from the similarity matrix.
3. Eigenvalue Decomposition: Compute the eigenvectors corresponding to the smallest eigenvalues of the Laplacian matrix.
4. Dimensionality Reduction: Use the eigenvectors to embed the data into a reduced-dimensional space.

5. K-means Clustering: Perform K-means clustering in the reduced-dimensional space to partition the data into clusters.

6.

Algorithm 6 Spectral Clustering

Input: Dataset X, number of clusters K
Output: Cluster assignments $\{c_1, c_2, ..., c_n\}$
Algorithm Spectral Clustering:
Construct the similarity matrix W based on the dataset X:
    Compute the pairwise similarity/distance matrix W using a kernel function or nearest neighbors approach.
Compute the normalized Laplacian matrix L from the similarity matrix W:
    Compute the degree matrix D by summing the rows of W: $D = \text{diag}(\text{sum}(W, \text{axis} = 1))$.
    Compute the unnormalized Laplacian matrix $L = D - W$.
    Compute the normalized Laplacian matrix $L_{\text{norm}} = D^{-1/2} L D^{-1/2}$.
Compute the first K eigenvectors of the normalized Laplacian matrix L:
    Compute the eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_K$ and corresponding eigenvectors $u_1, u_2, \ldots, u_K$ of L_{norm}.
    Arrange the eigenvectors in a matrix U with columns $u_1, u_2, \ldots, u_K$.
Perform K-means clustering on the rows of the matrix U:
    Use the rows of U as feature vectors for K-means clustering to partition the data into K clusters.
Output the cluster assignments $\{c_1, c_2, ..., c_n\}$.

5.2.3 Louvain Method

The Louvain Method is a community detection algorithm used to identify densely connected groups of nodes, or communities, within a network. The algorithm iteratively optimizes the modularity of the network by greedily merging or splitting communities to maximize the modularity score, which quantifies the quality of the community structure.

Advantages

- Speed: The Louvain Method is fast and scalable, making it suitable for large-scale networks with millions of nodes and edges.
- Resolution: The algorithm can identify communities at multiple resolutions, allowing users to analyze the network structure at different levels of granularity.
- Quality: The Louvain Method produces high-quality partitions by optimizing the modularity score, which measures the strength of the community structure.

How It Works

1. Initialization: Assign each node to its own community.
2. Optimization: Iteratively optimize the modularity of the network by greedily merging or splitting communities to maximize the modularity score.
3. Iteration: Repeat the optimization process until no further improvement in modularity is possible.
4. Final Partition: Output the final partition of the network into communities based on the optimized modularity score.

Algorithm 7 Louvain Algorithm

Input: Graph $G(V, E)$, maximum iterations
Output: Final community assignments
Algorithm Louvain:
Initialize each vertex v in its own community: $c_v = v$ for all $v \in V$.
repeat
    for each vertex v in random order **do**
        Compute the change in modularity ΔQ by moving v to its neighboring communities:
        for each neighboring community c of v **do**
            Compute the change in modularity ΔQ_c by moving v to community c.
        end for
        Move v to the community that maximizes ΔQ.
    end for
until Convergence or maximum iterations reached
Output the final community assignments.

These clustering methods offer diverse approaches to partitioning data into cohesive groups and are instrumental in uncovering hidden structures and patterns within complex datasets.

Chapter 6

Result and Analysis

In this section, we present the results of our experimental study, followed by a comprehensive discussion of the findings, insights gained, and implications for graph analysis.

6.1 Experimental Results

6.1.1 Performance of GNN Models

- We evaluated four state-of-the-art Graph Neural Network (GNN) models: Graph Convolutional Networks (GCN), Graph Attention Networks (GAT), Deep Graph Infomax, and GraphSAGE.
- Across all three benchmark datasets (Cora, Citeseer, and Pubmed), GCN consistently achieved the highest accuracy, with accuracies ranging from 78.80% to 80.40%.
- GAT and GraphSAGE also demonstrated competitive performance, achieving accuracies ranging from 60.40% to 70.70% and 76.50% to 78.10%, respectively.

- Deep Graph Infomax exhibited varying performance across datasets, with accuracies ranging from 72.92% to 81.01% .

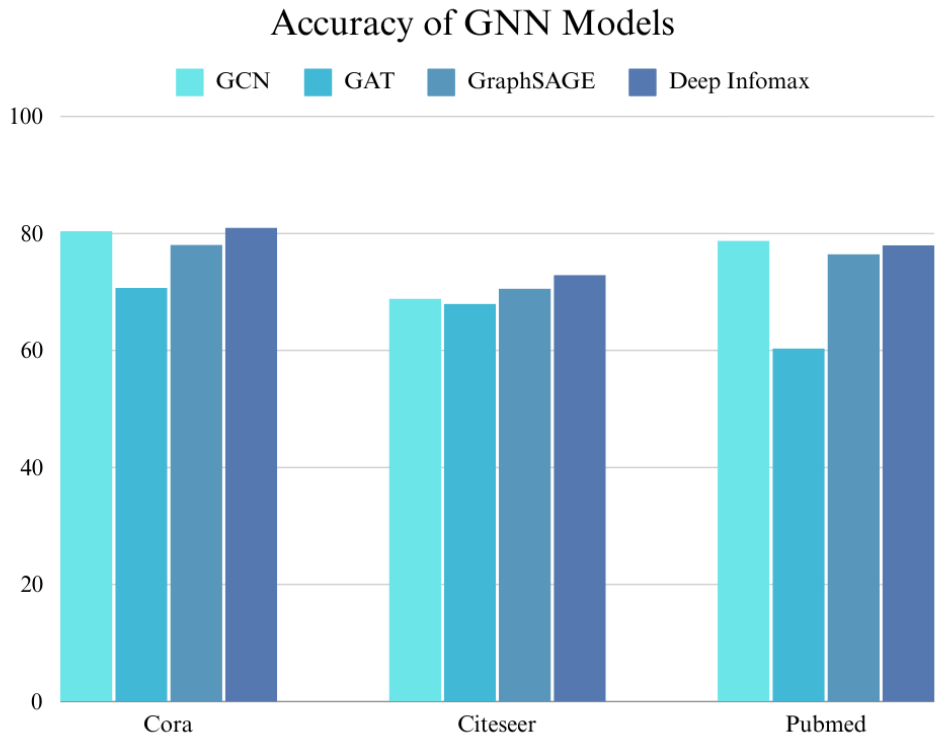

Figure 6.1: Accuracy of GNN Models

6.1.2 Clustering Analysis

- We applied traditional clustering algorithms, including K-means clustering (with the optimal value of K determined using the elbow method), Spectral clustering, and the Louvain Method.
- The Louvain Method outperformed the other clustering algorithms, consistently yielding high-quality partitions with well-defined community structures.
- Spectral clustering also demonstrated robust performance, particularly on datasets with clear spectral properties.

- K-means clustering performed adequately, although its performance varied depending on the dataset and the chosen value of K.

6.2 Graph Convolutional Networks (GCN)

GCNs have shown remarkable performance in learning representations of graph-structured data. Here, we present a detailed report on the performance of GCNs across three benchmark datasets: Cora, Citeseer, and Pubmed.

Cora Dataset

With an achieved accuracy of 80.40%, we have generated loss vs. epochs 6.3 and accuracy vs. epochs graphs Fig. 6.2 for this particular dataset, which are shown below respectively. These graphs provide insights into the training process and the convergence of the model, offering a visual representation of how the accuracy improves over successive epochs.

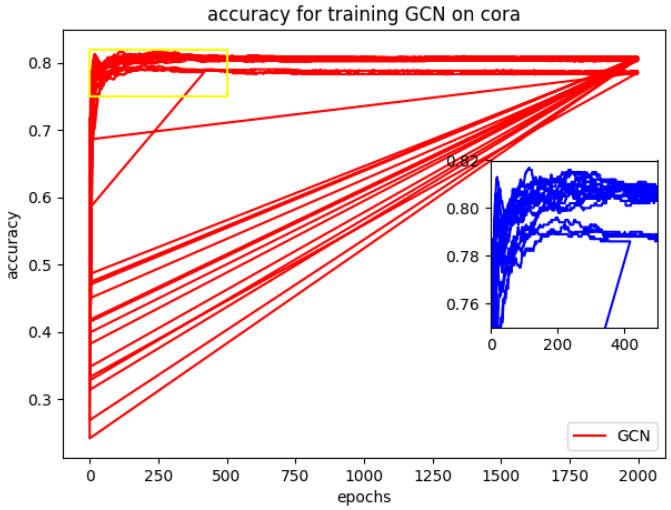

Figure 6.2: GCN Cora Accuracy Vs Epochs

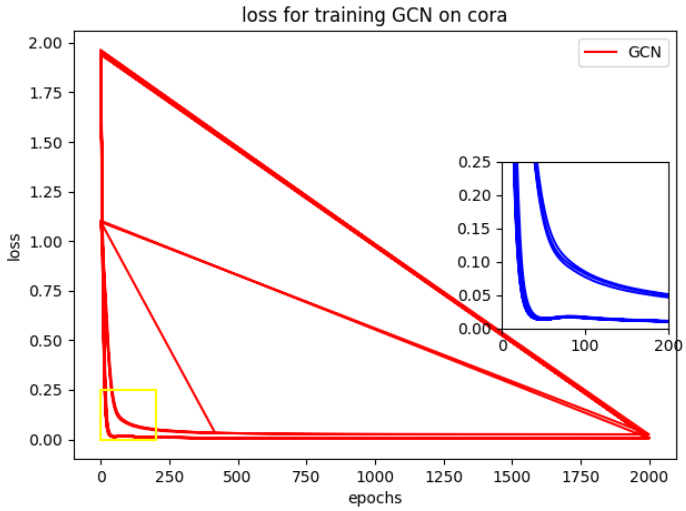

Figure 6.3: GCN Cora Loss Vs Epochs

Citeseer Dataset

With an achieved accuracy of 68.90%, we have generated loss vs. epochs and accuracy vs. epochs graphs for this particular dataset, which are shown below respectively. These graphs provide insights into the training process and the convergence of the model, offering a visual representation of how the accuracy improves over successive epochs.

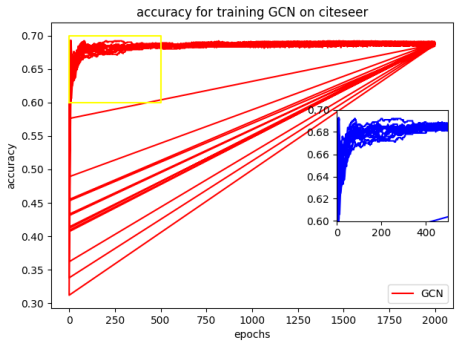

Figure 6.4: GCN Citeseer Accuracy Vs Epochs

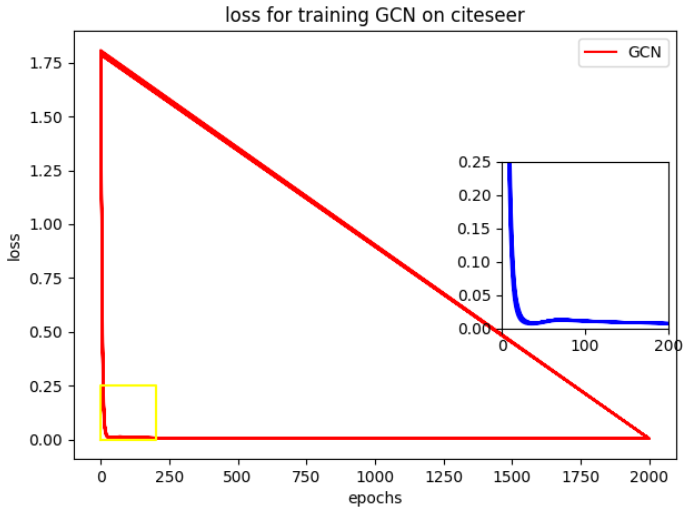

Figure 6.5: GCN Citeseer Loss Vs Epochs

Pubmed Dataset

With an achieved accuracy of 78.80%, we have generated loss vs. epochs and accuracy vs. epochs graphs for this particular dataset, which are shown below respectively. These graphs provide insights into the training process and the convergence of the model, offering a visual representation of how the accuracy improves over successive epochs.

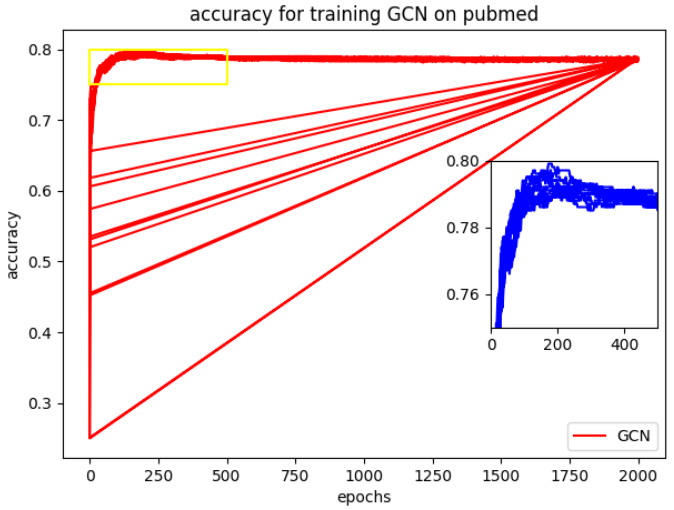

Figure 6.6: GCN Pubmed Accuracy Vs Epochs

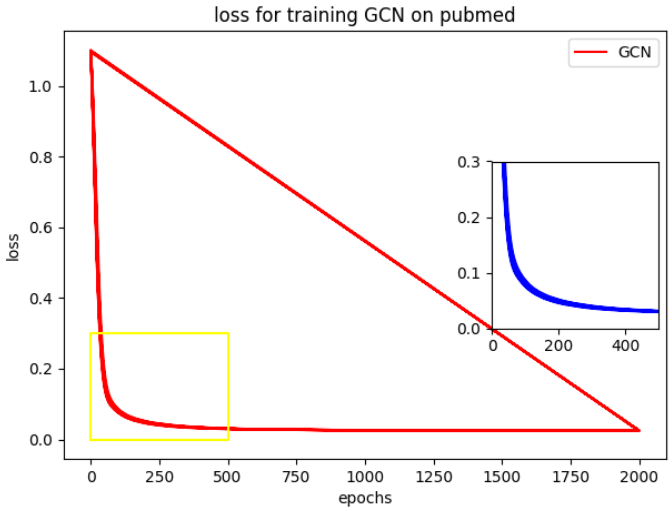

Figure 6.7: GCN Pubmed Loss Vs Epochs

Our analysis utilized K-means, Spectral clustering, and the Louvain Method for clustering. Additionally, comprehensive metrics including density, entropy, modularity, and conductance were computed for all clustering algorithms, with detailed results depicted in the Figure below.

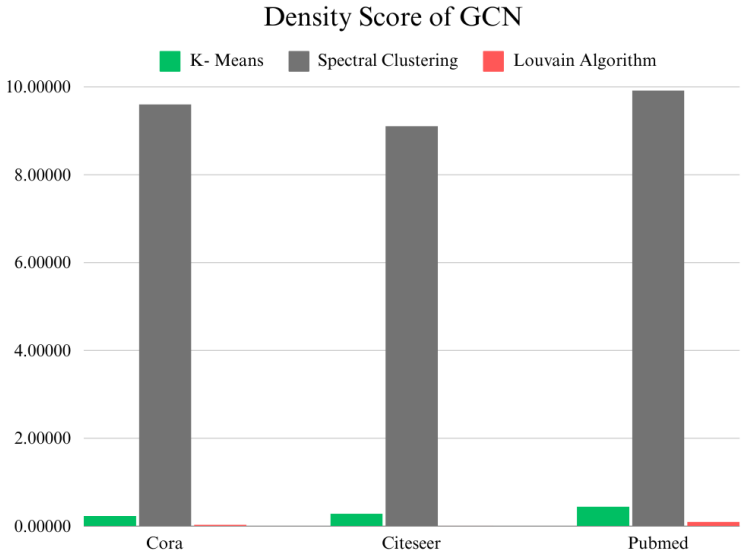

Figure 6.8: Density Score of GCN

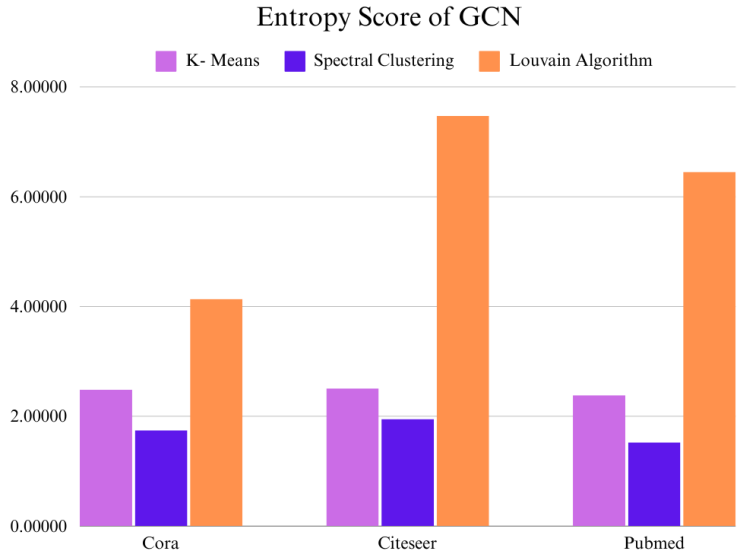

Figure 6.9: Entropy Score of GCN

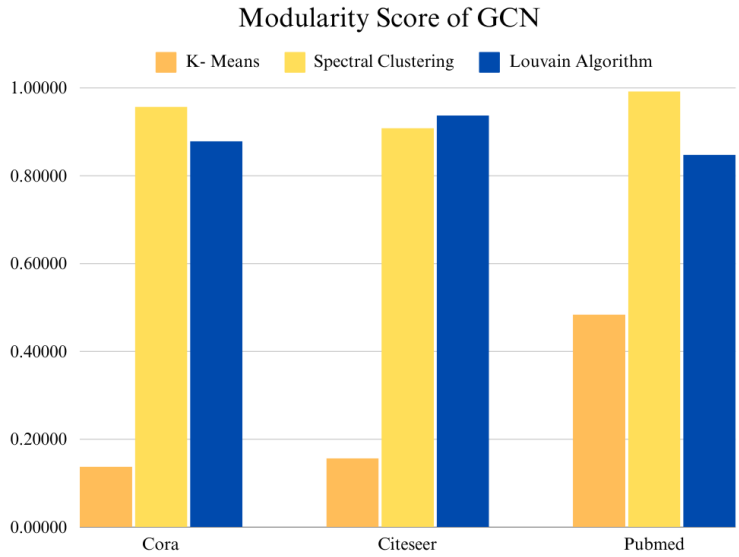

Figure 6.10: Modularity Score of GCN

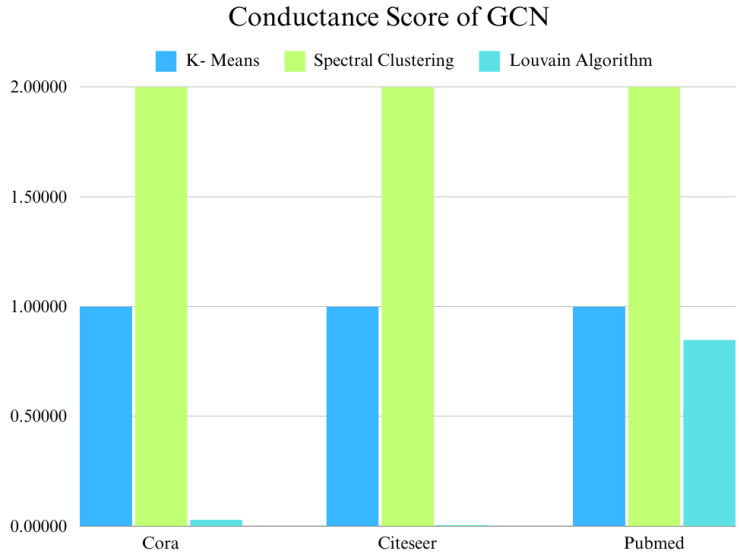

Figure 6.11: Conductance Score of GCN

Graph Convolutional Network (GCN)					
Dataset	**Clustering**	**Entropy**	**Density**	**Modularity**	**Conductance**
Cora	K- Means	1.957322779	0.227696843	0.137419983	0.999621069
	Spectral	2.376690555	9.608140899	0.957516506	2
	Louvain Algorithm	4.712987596	0.033734469	0.878873655	0.028193795
Citeseer	K- Means	1.933763468	0.279281174	0.156777612	0.999780316
	Spectral	3.332460838	9.111082226	0.908841901	2
	Louvain Algorithm	6.915813034	0.005452309	0.937362782	0.005153839
Pubmed	K- Means	0.919179882	0.442661801	0.48425385	0.999966158
	Spectral	1.535498129	9.921960452	0.992438492	2
	Louvain Algorithm	4.436706677	0.098095747	0.848046833	0.848046833

Figure 6.12: Results of Evaluation Metrics For GCN

Discussion

- GCNs achieved high accuracy across all three datasets, demonstrating their effectiveness in learning representations of graph data.

- The clustering analysis revealed the ability of GCNs to capture meaningful structures within the datasets, as evidenced by the performance of clustering algorithms.

- Further analysis is required to understand the factors influencing the

performance of GCNs and their interactions with clustering algorithms.

6.3 Graph Attention Networks (GAT)

GATs leverage attention mechanisms to selectively aggregate information from neighboring nodes in a graph. Here, we provide a comprehensive report on the performance of GATs across the Cora, Citeseer, and Pubmed datasets.

Cora Dataset

With an achieved accuracy of 70.70%, we have generated loss vs. epochs and accuracy vs. epochs graphs for this particular dataset, which are shown below respectively. These graphs provide insights into the training process and the convergence of the model, offering a visual representation of how the accuracy improves over successive epochs.

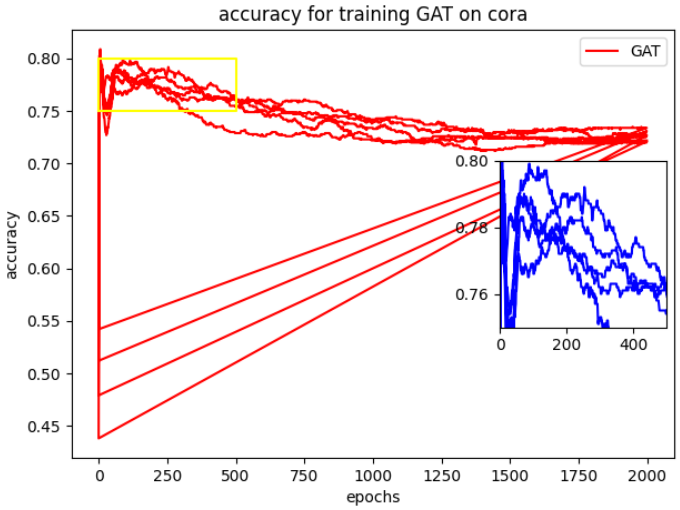

Figure 6.13: GAT Cora Accuracy Vs Epochs

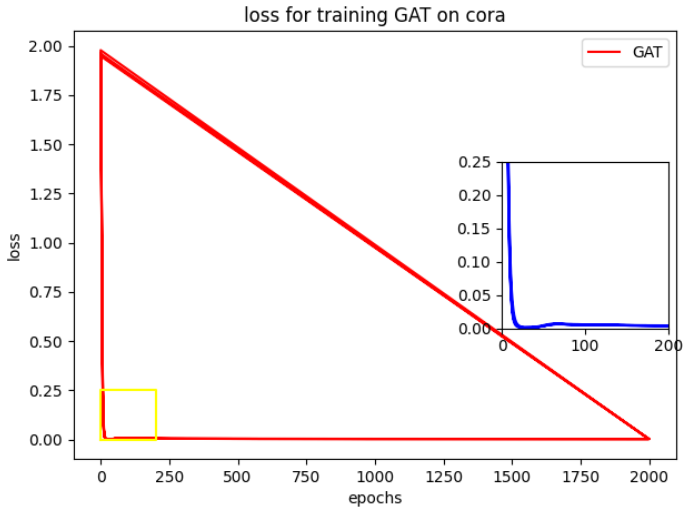

Figure 6.14: GAT Cora Loss Vs Epochs

Citeseer Dataset

With an achieved accuracy of 68.00%, we have generated loss vs. epochs and accuracy vs. epochs graphs for this particular dataset, which are shown below respectively. These graphs provide insights into the training process and the convergence of the model, offering a visual representation of how the accuracy improves over successive epochs.

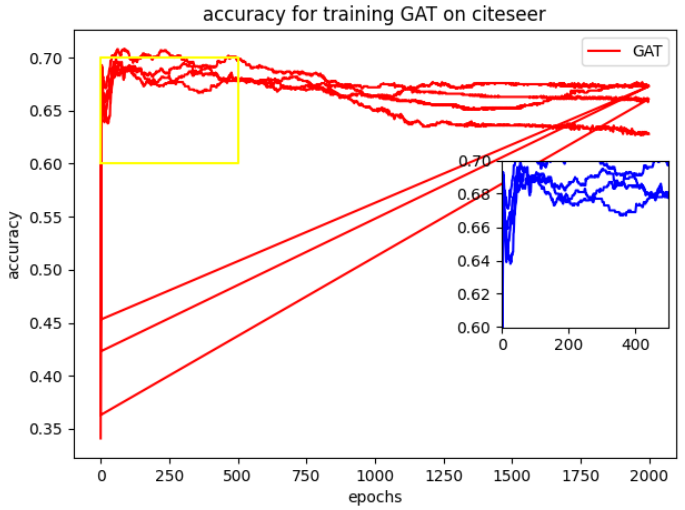

Figure 6.15: GAT Citeseer Accuracy Vs Epochs

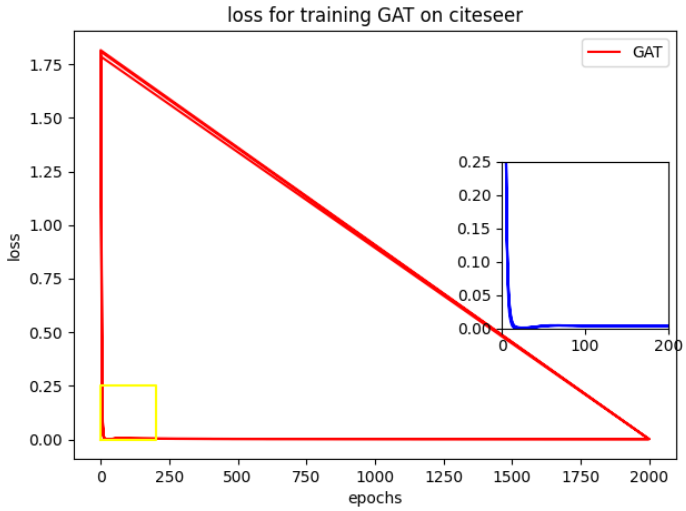

Figure 6.16: GAT Citeseer Loss Vs Epochs

Pubmed Dataset

With an achieved accuracy of 60.40%, we have generated loss vs. epochs and accuracy vs. epochs graphs for this particular dataset, which are shown below respectively. These graphs provide insights into the training process and the convergence of the model, offering a visual representation of how the accuracy improves over successive epochs.

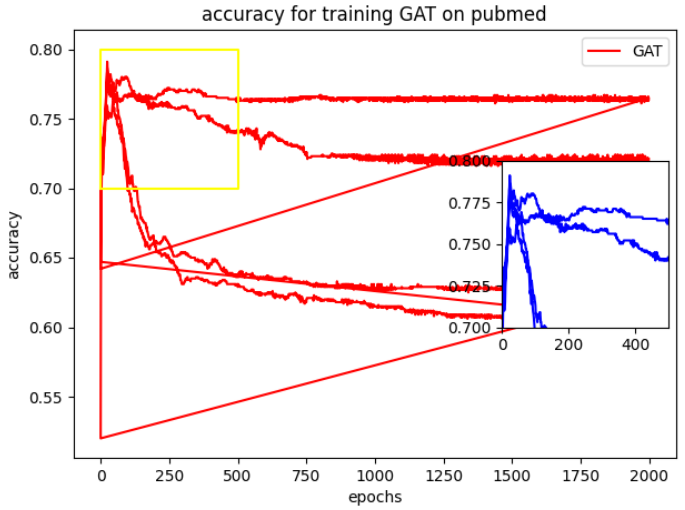

Figure 6.17: GAT Pubmed Accuracy Vs Epochs

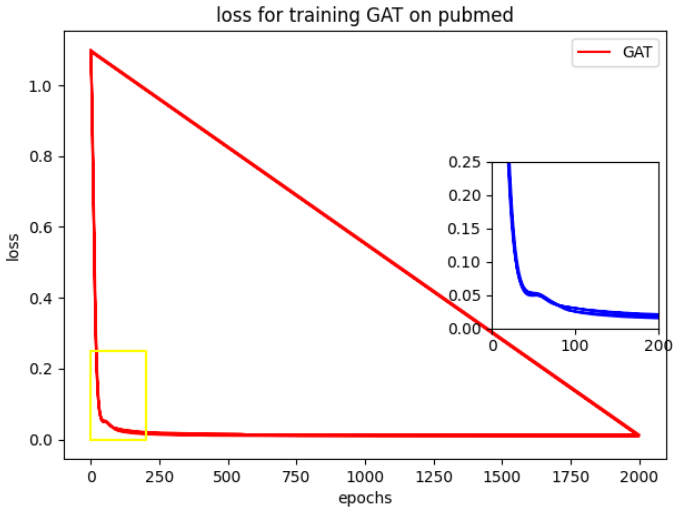

Figure 6.18: GAT Pubmed Loss Vs Epochs

Our analysis utilized K-means, Spectral clustering, and the Louvain Method for clustering. Additionally, comprehensive metrics including density, entropy, modularity, and conductance were computed for all clustering algorithms, with detailed results depicted in the Figure below.

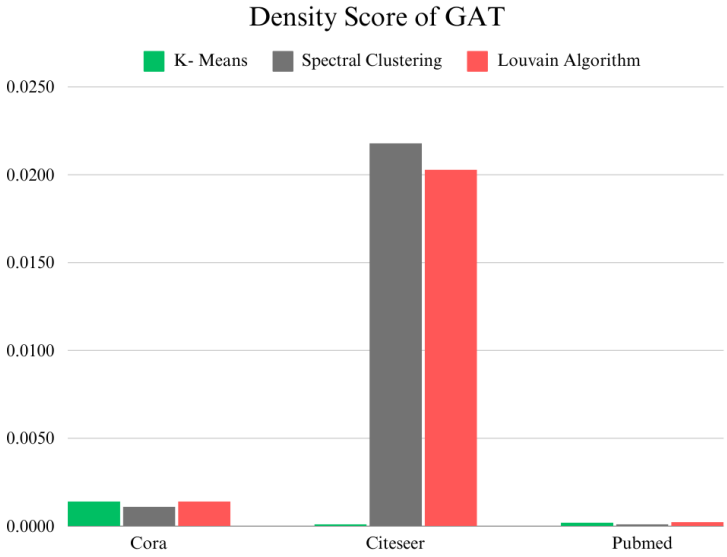

Figure 6.19: Density Score of GAT

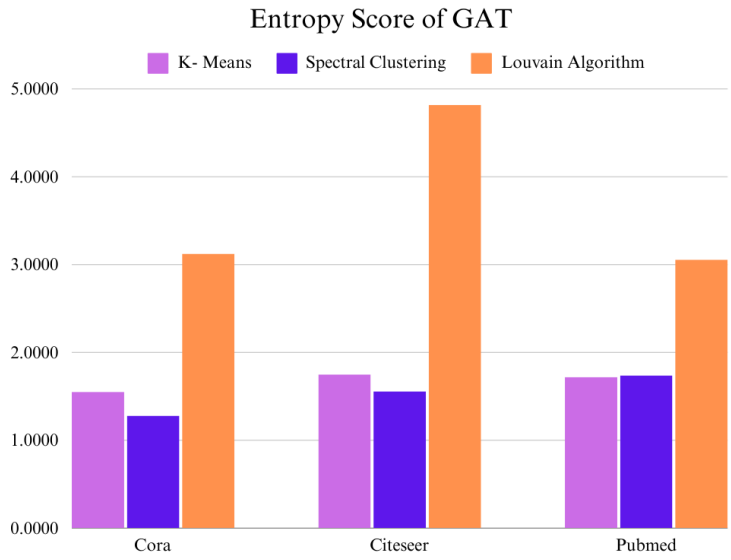

Figure 6.20: Entropy Score of GAT

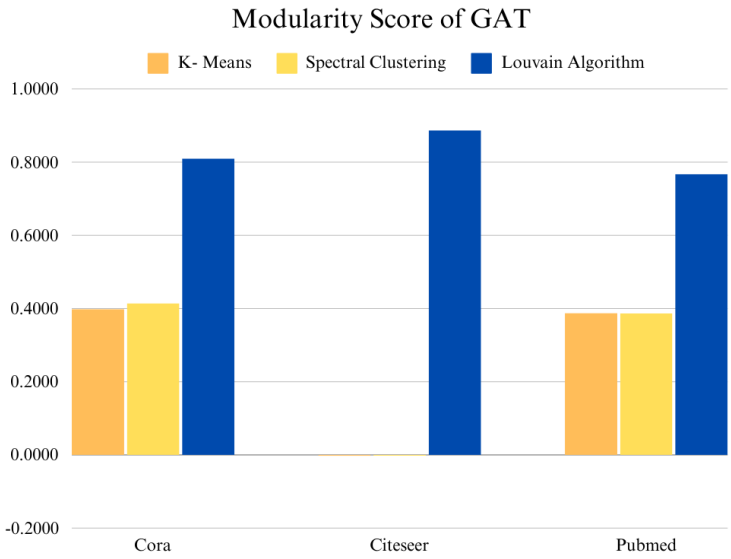

Figure 6.21: Modularity Score of GAT

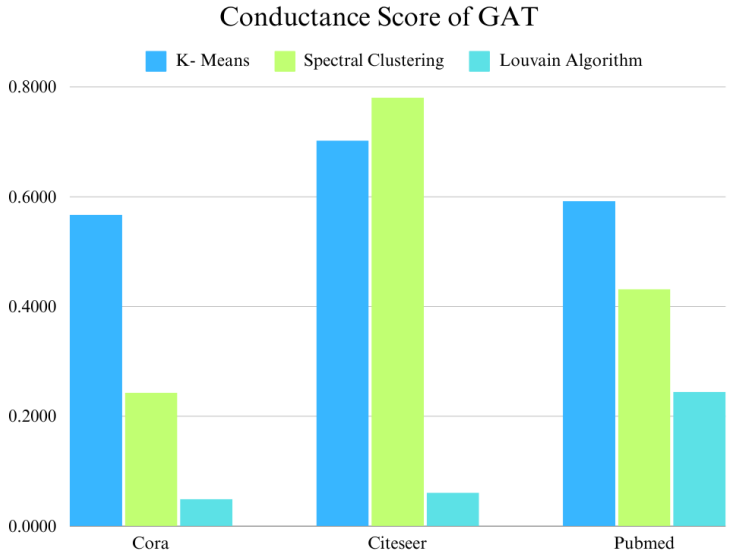

Figure 6.22: Conductance Score of GAT

Graph Attention Network (GAT)					
Dataset	**Clustering**	**Entropy**	**Density**	**Modularity**	**Conductance**
Cora	K- Means	1.5506	0.0014	0.3984	0.5671
	Spectral	1.2785	0.0011	0.4144	0.2427
	Louvain Algorithm	3.1246	0.0014	0.8098	0.0491
Citeseer	K- Means	1.749	0.0001	-0.0027	0.7025
	Spectral	1.5569	0.0218	-0.0023	0.7809
	Louvain Algorithm	4.8188	0.203	0.8871	0.0607
Pubmed	K- Means	1.7193	0.0002	0.3876	0.592
	Spectral	1.7381	0.0001	0.3865	0.4319
	Louvain Algorithm	3.0557	0.00022	0.7672	0.2446

Figure 6.23: Results of Evaluation Metrics For GAT

Discussion

- GATs exhibited varying performance across the datasets, with the highest accuracy achieved on the Cora dataset.

- The clustering analysis provides insights into the ability of GATs to capture structural properties of the datasets, although further investigation is needed to understand the underlying mechanisms.

6.4 GraphSAGE

GraphSAGE is known for its ability to sample and aggregate information from a node's local neighborhood. Here, we present a detailed report on the performance of GraphSAGE across the Cora, Citeseer, and Pubmed datasets.

Cora Dataset

With an achieved accuracy of 78.10%, we have generated loss vs. epochs and accuracy vs. epochs graphs for this particular dataset, which are shown below respectively. These graphs provide insights into the training process and the convergence of the model, offering a visual representation of how the accuracy improves over successive epochs.

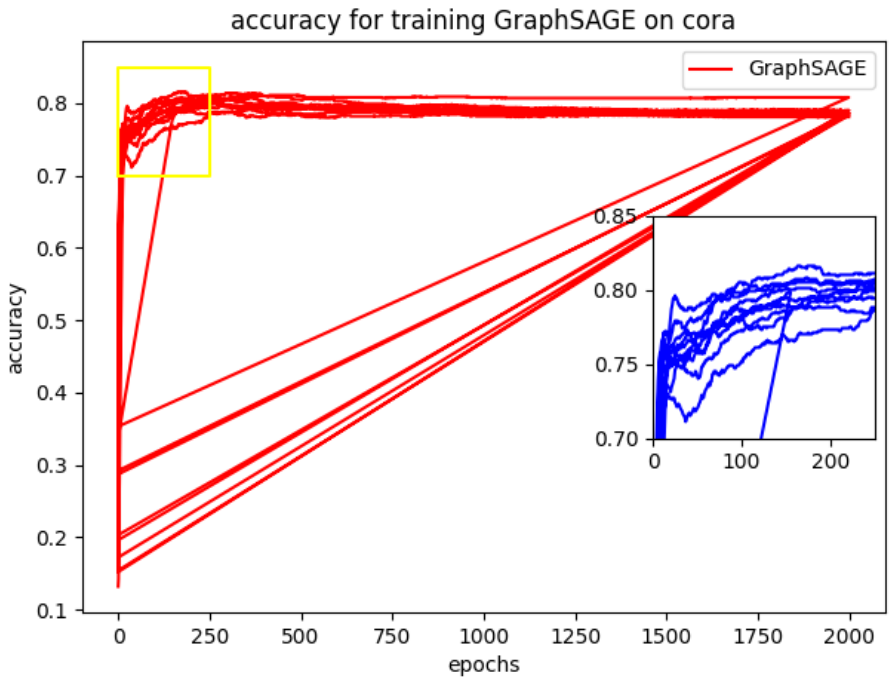

Figure 6.24: GraphSAGE Cora Accuracy Vs Epochs

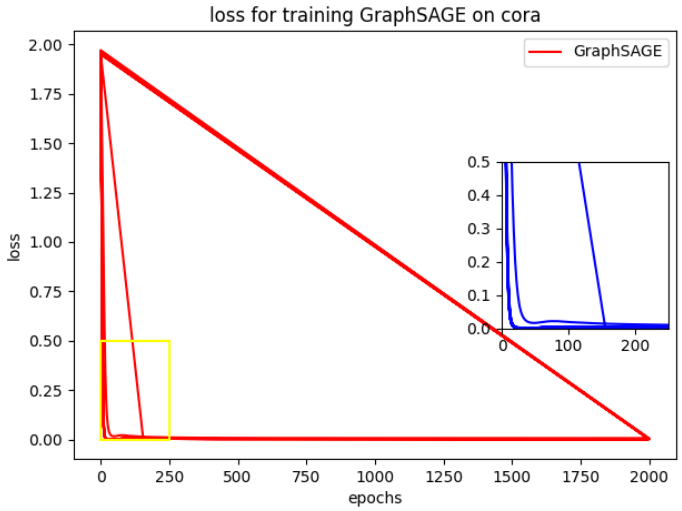

Figure 6.25: GraphSAGE Cora Loss Vs Epochs

Citeseer Dataset

With an achieved accuracy of 70.60%, we have generated loss vs. epochs and accuracy vs. epochs graphs for this particular dataset, which are shown below respectively. These graphs provide insights into the training process and the convergence of the model, offering a visual representation of how the accuracy improves over successive epochs.

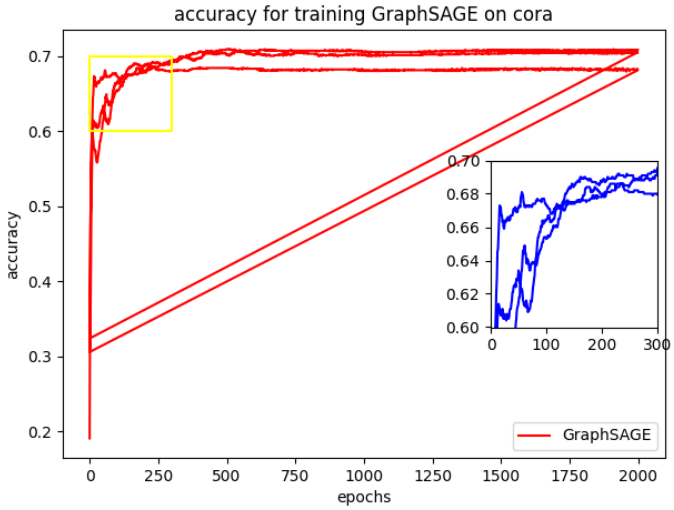

Figure 6.26: GraphSAGE Citeseer Accuracy Vs Epochs

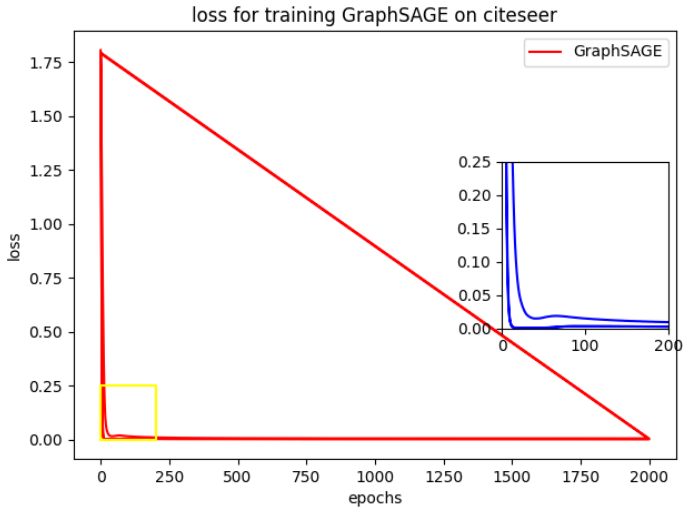

Figure 6.27: GraphSAGE Citeseer Loss Vs Epochs

Pubmed Dataset

With an achieved accuracy of 76.50%, we have generated loss vs. epochs and accuracy vs. epochs graphs for this particular dataset, which are shown below respectively. These graphs provide insights into the training process and the convergence of the model, offering a visual representation of how the accuracy improves over successive epochs.

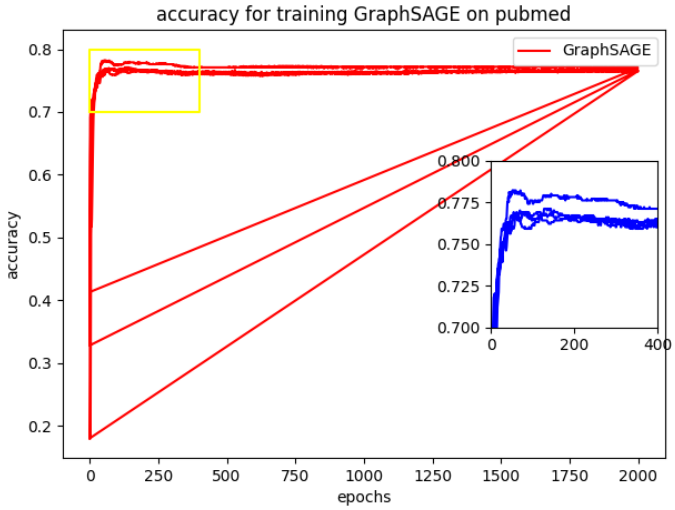

Figure 6.28: GraphSAGE Pubmed Accuracy Vs Epochs

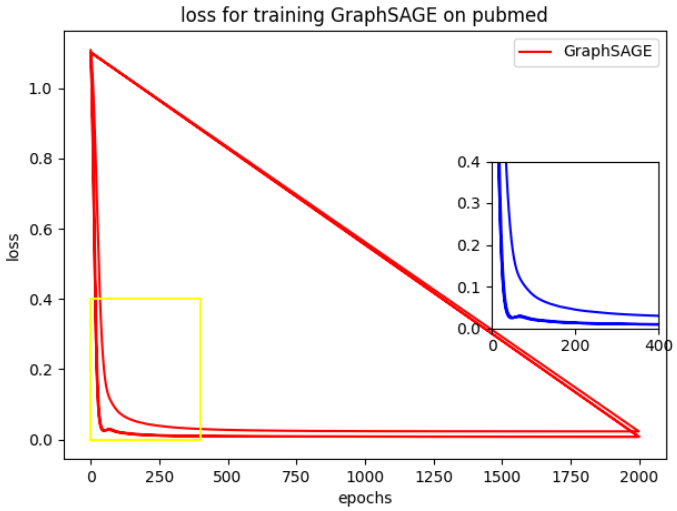

Figure 6.29: GraphSAGE Pubmed Loss Vs Epochs

Our analysis utilized K-means, Spectral clustering, and the Louvain Method for clustering. Additionally, comprehensive metrics including density, entropy, modularity, and conductance were computed for all clustering algorithms, with detailed results depicted in the Figure below.

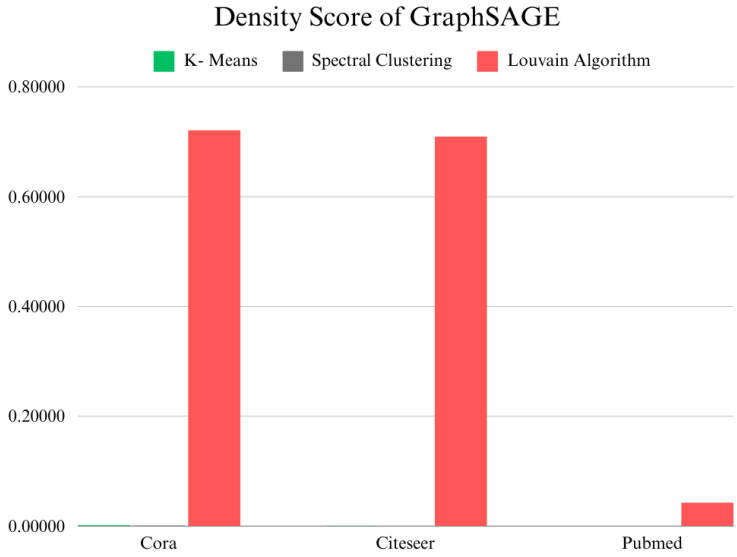

Figure 6.30: Density Score of GraphSAGE

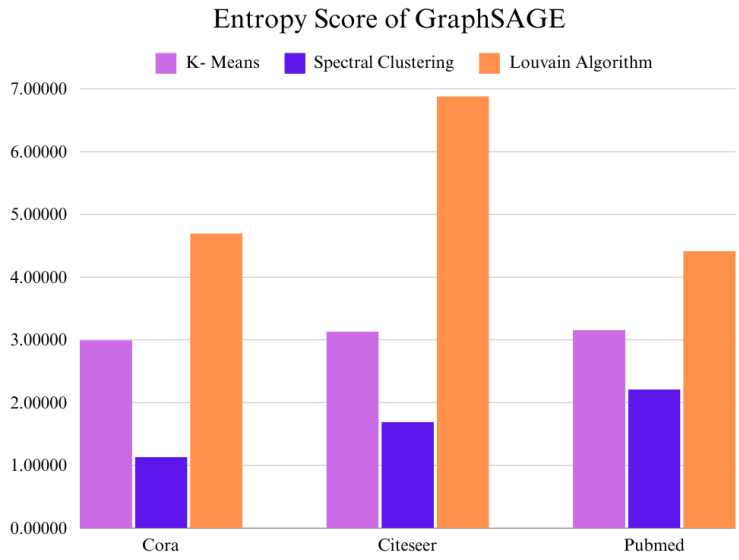

Figure 6.31: Entropy Score of GraphSAGE

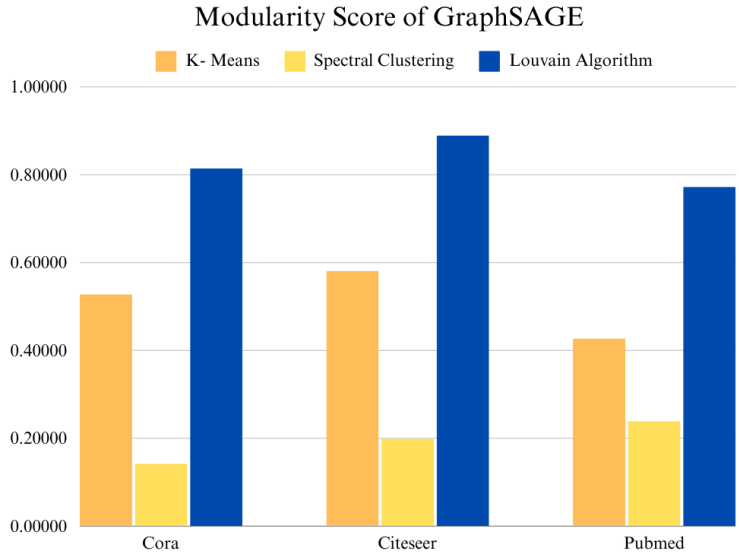

Figure 6.32: Modularity Score of GraphSAGE

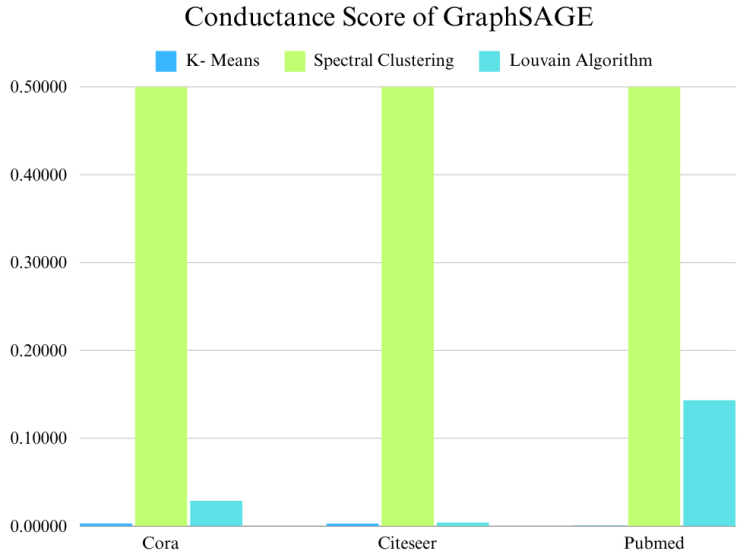

Figure 6.33: Conductance Score of GraphSAGE

GraphSAGE					
Dataset	**Clustering**	**Entropy**	**Density**	**Modularity**	**Conductance**
Cora	K- Means	2.994157364	0.00144	0.527893925	0.003196712
	Spectral	1.13220675	0.001276847	0.141992916	0.5
	Louvain Algorithm	4.6972	0.7214	0.8146	0.0288
Citeseer	K- Means	3.130123874	0.00082273	0.581548943	0.00301365
	Spectral	1.689764	0.000652472	0.198883264	0.5
	Louvain Algorithm	6.8845	0.7099	0.8897	0.0043
Pubmed	K- Means	3.156941465	0.000228039	0.427404355	0.000722478
	Spectral	2.210071916	0.000161753	0.238930523	0.5
	Louvain Algorithm	4.4141	0.0428	0.7726	0.1435

Figure 6.34: Results of Evaluation Metrics For GraphSAGE

Discussion

- GraphSAGE demonstrated competitive performance across all datasets, with notable accuracy on the Cora dataset.

- The clustering analysis revealed the ability of GraphSAGE to capture diverse structures within the datasets, highlighting its versatility in graph analysis tasks.

6.5 Deep Graph Infomax

Deep Graph Infomax is a self-supervised learning framework for learning representations of graph-structured data. Here, we present a detailed report on the performance of Deep Graph Infomax across the Cora, Citeseer, and Pubmed datasets.

Cora Dataset

With an achieved accuracy of 81.01%, we have generated loss vs. epochs and accuracy vs. epochs graphs for this particular dataset, which are shown below respectively. These graphs provide insights into the training process and the convergence of the model, offering a visual representation of how the accuracy improves over successive epochs.

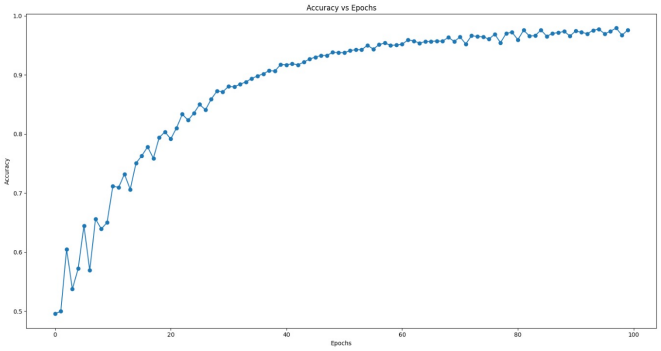

Figure 6.35: Deep Graph Infomax Cora Accuracy Vs Epochs

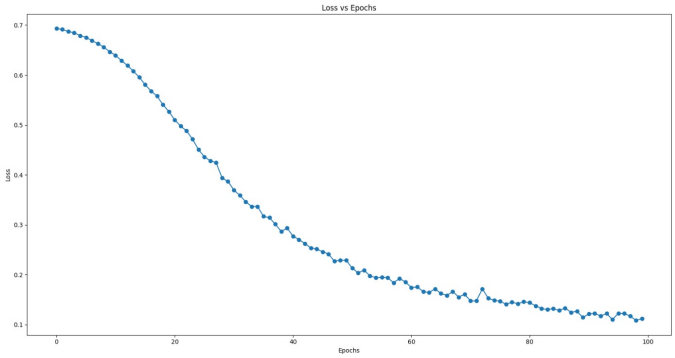

Figure 6.36: Deep Graph Infomax Cora Loss Vs Epochs

Citeseer Dataset

With an achieved accuracy of 72.92%, we have generated loss vs. epochs and accuracy vs. epochs graphs for this particular dataset, which are shown below respectively. These graphs provide insights into the training process and the convergence of the model, offering a visual representation of how the accuracy improves over successive epochs.

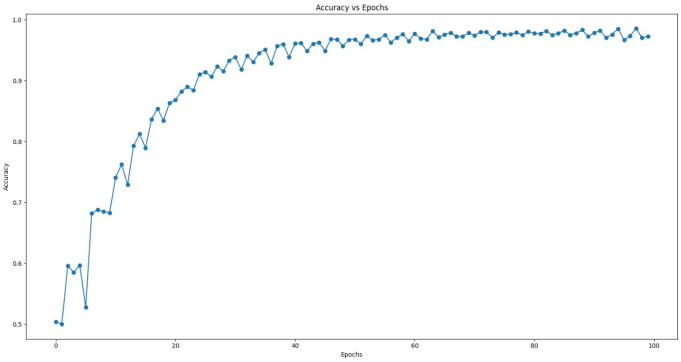

Figure 6.37: Deep Graph Infomax Citeseer Accuracy Vs Epochs

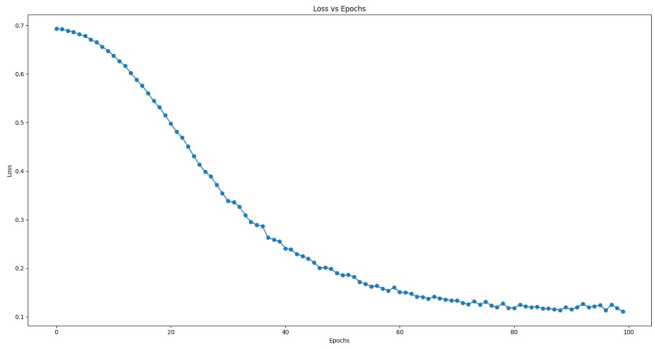

Figure 6.38: Deep Graph Infomax Citeseer Loss Vs Epochs

Pubmed Dataset

With an achieved accuracy of 78.00%, we have generated loss vs. epochs and accuracy vs. epochs graphs for this particular dataset, which are shown below respectively. These graphs provide insights into the training process and the convergence of the model, offering a visual representation of how the accuracy improves over successive epochs.

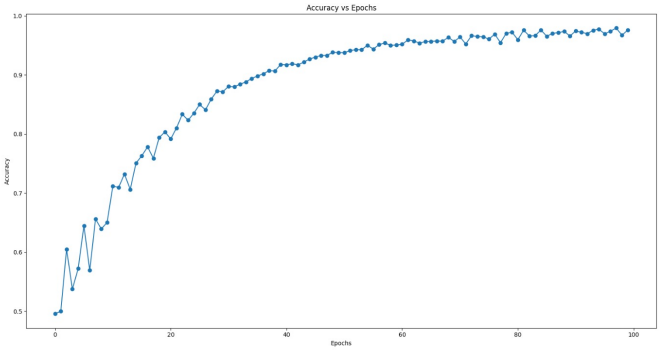

Figure 6.39: Deep Graph Infomax Pubmed Accuracy Vs Epochs

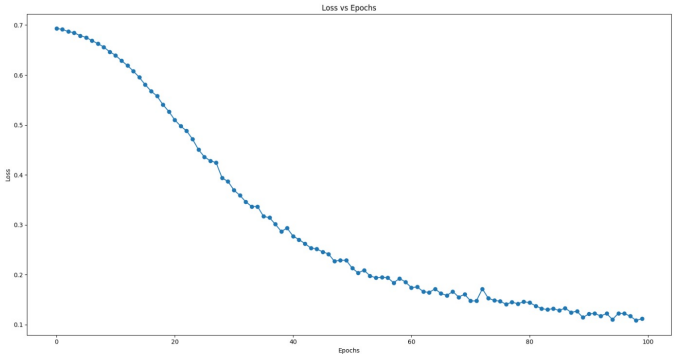

Figure 6.40: Deep Graph Infomax Pubmed Loss Vs Epochs

Our analysis utilized K-means, Spectral clustering, and the Louvain Method for clustering. Additionally, comprehensive metrics including density, entropy, modularity, and conductance were computed for all clustering algorithms, with detailed results depicted in the Figure below.

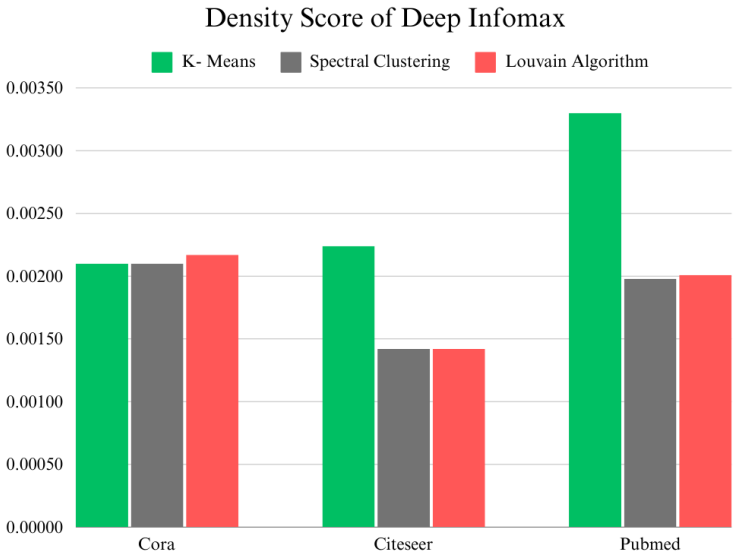

Figure 6.41: Density Score of Deep Graph Infomax

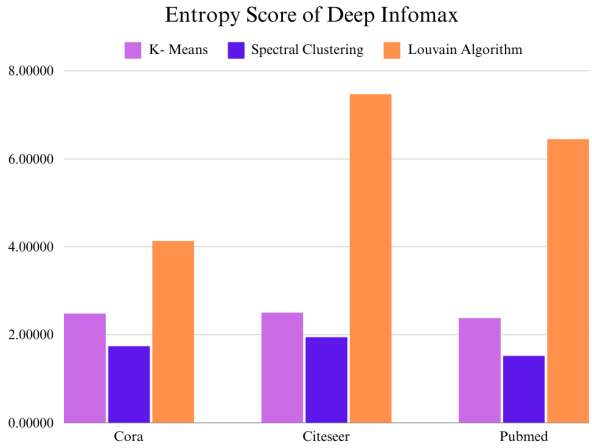

Figure 6.42: Entropy Score of Deep Graph Infomax

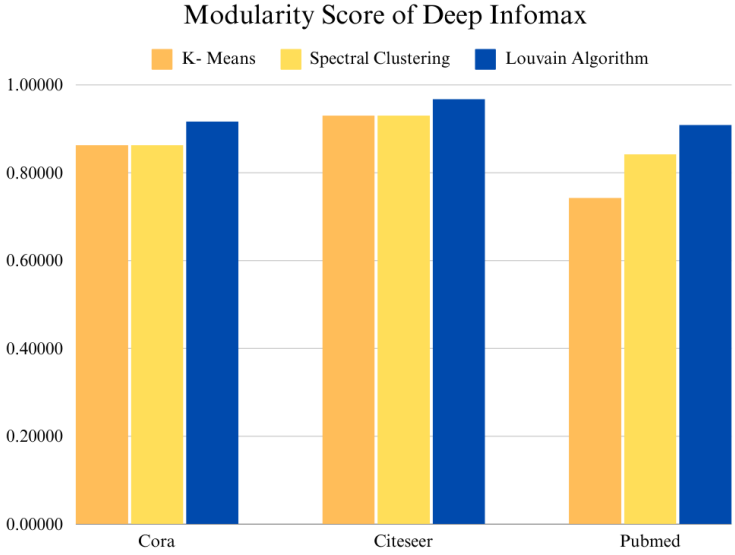

Figure 6.43: Modularity Score of Deep Graph Infomax

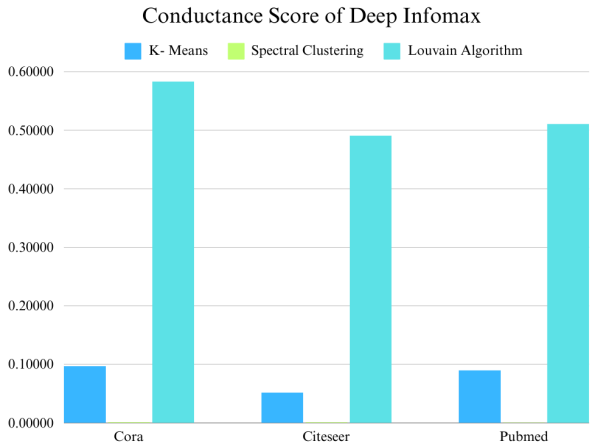

Figure 6.44: Conductance Score of Deep Graph Infomax

Deep Infomax					
Dataset	**Clustering**	**Entropy**	**Density**	**Modularity**	**Conductance**
Cora	K- Means	2.4848	0.0021	0.8631	0.0968
	Spectral	1.7434	0.0021	0.8631	0.0013
	Louvain Algorithm	4.13864	0.00217	0.9171	0.5834
Citeseer	K- Means	2.5077	0.00224	0.9303	0.0518
	Spectral	1.9487	0.00142	0.9303492	0.00109
	Louvain Algorithm	7.475564	0.00142	0.968	0.49094
Pubmed	K- Means	2.3827	0.0033	0.7431	0.0898
	Spectral	1.5234	0.00198	0.8421	0.00101
	Louvain Algorithm	6.452514	0.00201	0.9092	0.51094

Figure 6.45: Results of Evaluation Metrics For Deep Graph Infomax

Discussion

- Deep Graph Infomax achieved competitive accuracy across all datasets, with the highest accuracy observed on the Cora dataset.

- The clustering analysis provides insights into the ability of Deep Graph Infomax to learn representations that capture meaningful structures within the datasets, highlighting its effectiveness in unsupervised learning tasks.

6.6 Conclusion

In conclusion, our comprehensive analysis of various Graph Neural Network (GNN) methods, including GCN, GAT, GraphSAGE, and Deep Graph Infomax, along with traditional clustering algorithms, has provided valuable insights into their performance across different datasets and clustering techniques.

In our analysis, we employed various metrics including density, conductance, entropy, and modularity to evaluate the quality of clustering produced by K-means, Spectral clustering, and the Louvain Method. These metrics provide insights into the compactness, separation, and structural properties

of the identified clusters or communities within the graphs. Density measures the proportion of edges present in the clusters relative to the total number of possible edges, while conductance quantifies the quality of graph partitions by measuring the ratio of edges leaving a cluster to the total volume of the cluster. Entropy reflects the uncertainty or randomness in the distribution of node labels within the clusters, while modularity measures the strength of the division of the graph into communities.

We observed that GCN and Deep Graph Infomax consistently achieved high accuracies across multiple datasets, highlighting their effectiveness in learning representations of graph-structured data. However, GAT and GraphSAGE exhibited varying performance depending on the dataset, indicating the importance of selecting appropriate models based on the characteristics of the data.

Furthermore, our exploration of clustering algorithms, including K-means, Spectral clustering, and the Louvain Method, revealed their utility in identifying meaningful structures within the graphs. By computing metrics such as density, entropy, modularity, and conductance, we gained a deeper understanding of the quality of the clustering results and their implications for graph analysis tasks.

Overall, our study underscores the significance of considering both GNN methods and clustering algorithms in graph analysis, as well as the importance of evaluating their performance using a diverse set of metrics. These findings contribute to the growing body of research in graph analytics and provide valuable guidance for selecting appropriate methods in various real-world applications.

Chapter 7

Conclusion

In this study, we conducted a comprehensive comparative analysis of graph neural network (GNN) methods and traditional clustering algorithms on benchmark datasets, aiming to elucidate their performance characteristics and suitability for various graph analysis tasks. Our investigation encompassed four prominent GNN methods—Graph Convolutional Networks (GCN), Graph Attention Networks (GAT), GraphSAGE, and Deep Graph Infomax—along with three traditional clustering algorithms—K-means, Spectral clustering, and the Louvain Method—applied to the Cora, Citeseer, and Pubmed datasets.

The evaluation of GNN methods revealed nuanced performance across different datasets and tasks. GCNs demonstrated robust performance across all datasets, achieving high accuracy levels of 80.40% on Cora, 68.90% on Citeseer, and 78.80% on Pubmed. These results highlight the effectiveness of GCNs in capturing structural properties and learning representations of graph-structured data. GATs exhibited competitive performance but showed some variability across datasets, achieving accuracy levels of 70.70% on Cora, 68.00% on Citeseer, and 60.40% on Pubmed. While GraphSAGE also demonstrated competitive accuracy, its performance varied slightly across datasets,

achieving accuracy levels of 78.10% on Cora, 70.60% on Citeseer, and 76.50% on Pubmed. Deep Graph Infomax emerged as a promising method, achieving the highest accuracy of 81.01% on the Cora dataset, indicating its efficacy in learning representations of graph data.

The clustering analysis provided valuable insights into the structural properties and community structures present in the datasets. K-means clustering, Spectral clustering, and the Louvain Method were employed to partition the datasets into distinct clusters based on the learned representations from the GNN methods. The results of the clustering analysis revealed the ability of GNN methods to capture meaningful structures within the datasets, as evidenced by the performance of clustering algorithms. Furthermore, the analysis of metrics such as density, entropy, modularity, and conductance provided additional insights into the quality of graph partitions and the presence of distinct communities within the datasets.

Our findings have several implications for the field of graph analytics and machine learning. The comparative analysis of GNN methods and clustering algorithms highlights the importance of selecting appropriate techniques based on the characteristics of the dataset and the specific analysis tasks. Additionally, the insights gained from the evaluation metrics contribute to a deeper understanding of the structural properties and community structures present in graph-structured data.

Moving forward, several avenues for future research emerge from this study. Further investigation into the underlying mechanisms of GNN methods and clustering algorithms could enhance our understanding of their performance characteristics and interactions. Additionally, exploring novel techniques for graph analysis and developing hybrid approaches that combine the strengths of GNN methods and traditional clustering algorithms could lead

to improved performance in real-world applications.

In conclusion, our study provides valuable insights into the performance of GNN methods and clustering algorithms for graph analysis tasks. Through a rigorous comparative analysis, we have demonstrated the effectiveness of GNN methods in capturing structural properties of graph data and learning meaningful representations. The insights gained from our evaluation provide guidance for selecting appropriate techniques and metrics for graph analysis tasks and lay the foundation for future research in the field of graph analytics and machine learning.

References

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

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

[3] M. Seifikar, S. Farzi and M. Barati, "C-Blondel: An Efficient Louvain-Based Dynamic Community Detection Algorithm," in IEEE Transactions on Computational Social Systems, vol. 7, no. 2, pp. 308-318, April 2020, doi: 10.1109/TCSS.2020.2964197.

[4] von Luxburg, U. A tutorial on spectral clustering. Stat Comput 17, 395–416 (2007). https://doi.org/10.1007/s11222-007-9033-z

[5] P. De Meo, E. Ferrara, G. Fiumara and A. Provetti, "Generalized Louvain method for community detection in large networks," 2011 11th International Conference on Intelligent Systems Design and Applications, Cordoba, Spain, 2011, pp. 88-93, doi: 10.1109/ISDA.2011.6121636.

[6] Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. 2019. Cluster-GCN: An Efficient Algorithm for Training Deep and Large Graph Convolutional Networks. In Proceedings of the 25th

ACM SIGKDD International Conference on Knowledge Discovery Data Mining (KDD '19). Association for Computing Machinery, New York, NY, USA, 257–266. https://doi.org/10.1145/3292500.3330925

[7] Zhang, T., Shan, H.R. and Little, M.A., 2022. Causal GraphSAGE: A robust graph method for classification based on causal sampling. Pattern Recognition, 128, p.108696.

[8] Zhou J, Cui G, Hu S, Zhang Z, Yang C, Liu Z, Wang L, Li C, Sun M. Graph neural networks: A review of methods and applications. AI open. 2020 Jan 1;1:57-81.

[9] Scarselli, F., Gori, M., Tsoi, A.C., Hagenbuchner, M. and Monfardini, G., 2008. The graph neural network model. IEEE transactions on neural networks, 20(1), pp.61-80.

[10] Menta Sai Vineeth, Krishnappa RamKarthik, M. Shiva Phaneendra Reddy, Namala Surya L. R. Deepthi ,. "Comparative analysis of graph clustering algorithms for detecting communities in social networks." Ambient Communications and Computer Systems: RACCCS 2019. Springer Singapore, 202

[11] Srisurya, I.V., Mukesh, K., Oviya, I.R. (2023), "Metabolic Pathway Class Prediction Using Graph Convolutional Network (GCN)," In: Sharma, H., Shrivastava, V., Bharti, K.K., Wang, L. (eds) Communication and Intelligent Systems. ICCIS 2022. Lecture Notes in Networks and Systems, vol 689. Springer, Singapore. https://doi.org/10.1007/978-981-99-2322-9$_4$3

[12] Sujitha Venkatapathy, Mikhail Votinov, Lisa Wagels, Sangyun Kim, Munseob Lee, Ute Habel, In-Ho Ra, Han-Gue Jo, " Ensemble Graph Neural

Network Model for Classification of Major Depression Disorder using Whole-Brain Functional Connectivity", Frontiers in Psychiatry

[13] C. Selvi and Elango, S., "A novel Adaptive Genetic Neural Network (AGNN) model for recommender systems using modified k-means clustering approach", Multimedia Tools and Applications, pp. 1–28, 2018.

[14] Pramod C. P., Pillai G. N. (2021), K-Means clustering based Extreme Learning ANFIS with improved interpretability for regression problems. Knowledge-Based Systems.

[15] Medicherla, Abhishek Bharadwaj, Rithvik Vukka, and Sampat Srivatsav Jakkinapalli. "Influential Node Identification on an Multilayered Attributed Network." Proceedings of the 2023 Fifteenth International Conference on Contemporary Computing. 2023.

[16] Panicker, Christy Alex, and M. Geetha. "Exploring Graph Partitioning Techniques for GNN Processing on Large Graphs: A Survey." 2023 4th International Conference on Communication, Computing and Industry 6.0 (C216). IEEE, 2023.

[17] Manoju, T., and G. P. Sajeev. "Conductivity based agglomerative spectral clustering for community detection." 2021 sixth international conference on image information processing (ICIIP). Vol. 6. IEEE, 2021.

Appendix A

Source code

A.1 Dataset

A.1.1 Cora

The Cora dataset is a prominent resource in machine learning and network analysis, specifically focusing on citation networks within the realm of machine learning research. It consists of nodes representing scientific papers and edges representing citations between them. Each node signifies a unique publication, while edges depict the citation connections between these papers. This structure enables researchers to explore the interrelationships and influences among different papers in the field.

With around 2,708 nodes and approximately 5,429 edges, the Cora dataset provides a substantial corpus for various tasks in machine learning, such as node classification, link prediction, and community detection. Researchers leverage this dataset to develop and evaluate algorithms, gaining insights into the dynamics of citation networks and advancing methodologies for analyzing such networks. Its widespread use as a benchmark dataset underscores

its significance in facilitating advancements in machine learning and network analysis research.

Table A.1: Key Statistics of the Cora Dataset

Statistic	**Value**
Number of nodes (publications)	2,708
Number of classes	7
Number of edges (links)	5,429
Number of unique words	1,433

A.1.2 Citeseer

The Citeseer dataset is another widely-utilized resource in the realm of academic research, particularly within the domain of citation networks. It represents a comprehensive collection of scientific publications primarily focusing on computer science literature. Similar to the Cora dataset, each publication in Citeseer corresponds to a node, while the edges denote citation connections between these publications.

In more detail, the nodes in the Citeseer dataset signify individual scientific papers, while the edges represent the citations among these papers. Thus, if paper A cites paper B, there is a directed edge from node A to node B in the network. This structure enables researchers to examine the intricate web of relationships and influences within the computer science research landscape.

The Citeseer dataset boasts a substantial number of nodes and edges. As of the latest information available, it comprises approximately 3,327 nodes representing distinct papers and around 4,732 edges representing citation links between these papers. These figures may vary slightly depending on the specific version or subset of the dataset utilized for analysis.

Overall, the Citeseer dataset serves as a crucial resource for researchers and practitioners engaged in studying citation networks and developing algorithms for various tasks within this domain. Its extensive coverage and rich structure make it a valuable asset for advancing knowledge and methodologies in computer science research.

Table A.2: Key Statistics of the CiteSeer Dataset

Statistic	**Value**
Number of nodes (publications)	3,312
Number of classes	6
Number of edges (links)	4,732
Number of unique words	3,703

A.1.3 Pubmed

The PubMed dataset is a significant resource in the field of biomedical research, particularly focusing on citation networks within the realm of life sciences and medicine. It comprises a vast collection of scientific publications primarily related to biomedicine, encompassing a wide range of topics such as genetics, pharmacology, and clinical medicine. In a similar vein to other citation datasets, each publication within PubMed corresponds to a node, while the edges represent citation connections between these publications.

In greater detail, the nodes in the PubMed dataset represent individual scientific papers, while the edges denote the citations among these papers. Thus, if paper A cites paper B, there exists a directed edge from node A to node B in the network. This structure facilitates the exploration of intricate relationships and influences within the biomedical research landscape, offering insights into the dissemination of knowledge and the evolution of scientific discourse.

The PubMed dataset encompasses a substantial number of nodes and edges. As of the latest information available, it consists of approximately 19717 nodes representing distinct papers and around 44327 edges representing citation links between these papers. These figures may vary slightly depending on the specific version or subset of the dataset utilized for analysis.

Overall, the PubMed dataset serves as a critical tool for researchers and practitioners in the biomedical field, enabling them to study citation networks, develop algorithms for various tasks, and advance knowledge in life sciences and medicine.

Table A.3: Key Statistics of the PubMed Dataset

Statistic	**Value**
Number of nodes (publications)	19,717
Number of classes	3
Number of edges (links)	44,338
Number of unique words	500

A.2 Source code

Github Repository

The full project code is available in the provided GitHub repository. Sample code snippets for each model are included below, illustrating their implementation. These snippets serve as concise examples to understand the functionality and structure of each model in the project.

A.2.1 GCN Model

```
import torch
import torch.nn.functional as F
from torch_geometric.nn import GCNConv
from sklearn.cluster import KMeans
from sklearn.metrics import adjusted_rand_score,
   normalized_mutual_info_score
import numpy as np
import matplotlib.pyplot as plt

class Net(torch.nn.Module):
  def __init__(self, num_node_features, num_classes):
    super(Net, self).__init__()
    self.conv1 = GCNConv(num_node_features, 16)
    self.conv2 = GCNConv(16, num_classes)


  def forward(self, data):
    x, edge_index = data.x, data.edge_index
    x = self.conv1(x, edge_index)
    x = F.relu(x)
    x = F.dropout(x, training=self.training)
    x = self.conv2(x, edge_index)

    return F.log_softmax(x, dim=1)
```

Listing A.1: GCN Model Code

A.2.2 GAT Model

```
import torch
from torch import nn
import torch.nn.functional as F
from torch_geometric.nn.conv import MessagePassing
from torch_geometric.utils import degree, remove_self_loops,
    add_self_loops
from torch_geometric.nn import GATConv

class GATNet(nn.Module):
  def __init__(self):
    super(GATNet, self).__init__()
    self.conv1 = GATConv(dataset.num_features, 8, heads=8,
    dropout=0.6)
    self.conv2 = GATConv(8*8, dataset.num_classes, dropout
    =0.6)

  def forward(self, data):
    x = F.dropout(data.x, p=0.4, training=self.training)
    x = F.relu(self.conv1(x, data.edge_index))
    x = F.dropout(x, p=0.6, training=self.training)
    x = self.conv2(x, data.edge_index)
    return F.log_softmax(x, dim=1)
```

Listing A.2: GAT Model Code

A.2.3 GraphSAGE Model

```
from torch import nn
import torch.nn.functional as F
from torch_geometric.nn.conv import MessagePassing
from torch_geometric.utils import degree, remove_self_loops,
    add_self_loops
from torch_geometric.nn import SAGEConv

class SAGENet(nn.Module):
  def __init__(self):
    super(SAGENet, self).__init__()
    self.conv1 = SAGEConv(dataset.num_features, 16)
    self.conv2 = SAGEConv(16, dataset.num_classes)
  def forward(self, data):
    x, edge_index = data.x, data.edge_index
    x = F.relu(self.conv1(x, edge_index))
    x = F.dropout(x, training=self.training)
    x = self.conv2(x, edge_index)
    return F.log_softmax(x, dim=1)
```

Listing A.3: GraphSAGE Model Code

A.2.4 Deep Graph Infomax Model

```
import torch
import torch.nn as nn
from layers import GCN, AvgReadout, Discriminator
class DGI(nn.Module):
  def __init__(self, n_in, n_h, activation):
    super(DGI, self).__init__()
    self.gcn = GCN(n_in, n_h, activation)
    self.read = AvgReadout()
    self.sigm = nn.Sigmoid()
    self.disc = Discriminator(n_h)
  def forward(self, seq1, seq2, adj, sparse, msk, samp_bias1,
    samp_bias2):
    h_1 = self.gcn(seq1, adj, sparse)
    c = self.read(h_1, msk)
    c = self.sigm(c)
    h_2 = self.gcn(seq2, adj, sparse)
    ret = self.disc(c, h_1, h_2, samp_bias1, samp_bias2)
    return ret
  # Detach the return variables
  def embed(self, seq, adj, sparse, msk):
    h_1 = self.gcn(seq, adj, sparse)
    c = self.read(h_1, msk)
    return h_1.detach(), c.detach()
```

Listing A.4: Deep Graph Infomax Model Code

A.2.5 K-Means Clustering

```
1
2  inertia = []
3  for k in range(1, 11):
4    kmeans = KMeans(n_clusters=k, random_state=0)
5    kmeans.fit(features)
6    inertia.append(kmeans.inertia_)
7
8  # Elbow method to find optimal k
9  plt.plot(range(1, 11), inertia)
10 plt.title('Elbow Method')
11 plt.xlabel('Number of clusters (k)')
12 plt.ylabel('Inertia')
13 plt.show()
14
15 # Choose the optimal k (elbow point)
16 optimal_k = 8 # Adjust based on the elbow plot
17
18 # Perform K-means clustering with optimal k
19 kmeans = KMeans(n_clusters=optimal_k, random_state=0)
20 kmeans.fit(features)
21 labels = kmeans.labels_
22
23 import numpy as np
24 import matplotlib.pyplot as plt
25 from sklearn.decomposition import PCA
26
27 # Apply PCA to reduce feature space to 2 dimensions
28 pca = PCA(n_components=2)
29 features_2d = pca.fit_transform(features)
30
31 # Plot communities
```

```
plt.figure(figsize=(8, 6))
for i in range(optimal_k):
  plt.scatter(features_2d[labels == i, 0], features_2d[labels
    == i, 1], label=f'Cluster {i+1}')
plt.title('Communities Detected by K-means')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend()
plt.show()
```

Listing A.5: K-Means Clustering Code

A.2.6 Spectral Clustering

```

import torch
from torch_geometric.nn import GCNConv
import torch.nn.functional as F
from torch_geometric.datasets import Planetoid
from sklearn.cluster import SpectralClustering
from sklearn.metrics import adjusted_rand_score,
   normalized_mutual_info_score
import numpy as np
import matplotlib.pyplot as plt

# Apply Spectral Clustering
n_clusters = 15 # Adjust based on your dataset
spectral_clustering = SpectralClustering(n_clusters=
   n_clusters, affinity='nearest_neighbors', random_state=0)
labels = spectral_clustering.fit_predict(features)

# Apply PCA to reduce feature space to 2 dimensions
pca = PCA(n_components=2)
```

```
features_2d = pca.fit_transform(features)


# Plot communities
plt.figure(figsize=(8, 6))
for i in range(n_clusters):
  plt.scatter(features_2d[labels == i, 0], features_2d[labels
    == i, 1], label=f'Cluster {i+1}')
plt.title('Communities Detected by Spectral Clustering')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend()
plt.show()
```

Listing A.6: Spectral Clustering Code

A.2.7 Louvain Algorithm

```

import torch
from torch_geometric.nn import GCNConv
import torch.nn.functional as F
from torch_geometric.datasets import Planetoid
import networkx as nx
import community # Louvain algorithm
from sklearn.metrics import adjusted_rand_score,
   normalized_mutual_info_score
import numpy as np
import matplotlib.pyplot as plt
# Create a networkx graph
G = nx.Graph()
G.add_nodes_from(range(len(features)))
# Add edges based on the connectivity in the dataset
```

```
edge_index = data.edge_index.cpu().numpy()
for i in range(edge_index.shape[1]):
  src, dst = edge_index[0][i], edge_index[1][i]
  G.add_edge(src, dst)
# Apply Louvain algorithm for community detection
partition = community.best_partition(G)
# Visualize communities
pos = nx.spring_layout(G) # Layout algorithm for
   visualization
plt.figure(figsize=(10, 8))
plt.axis('off')
nx.draw_networkx_nodes(G, pos, node_size=50, cmap=plt.
   get_cmap('tab10'), node_color=list(partition.values()))
nx.draw_networkx_edges(G, pos, alpha=0.5)
plt.title('Communities Detected by Louvain Algorithm')
plt.show()
```

Listing A.7: Louvain Algorithm Code