Labsheet 6 – IT21250156

**1) Increase the N value from 20 (original value) to 200 with multiple N values in between and observe the change of graph density and degree distribution (i.e., histogram plot). Explain what you observe and write the answer in a word file.**

For small graphs (low N), a high proportion of the possible edges can be realized even with a fixed edge probability.

For large graphs (high N), even though the number of edges increases, the proportion of possible edges that are actually realized becomes smaller due to the quadratic increase in the number of possible edges.

Thus, the density decreases with increasing N because the growth in the number of possible edges outpaces the growth in the number of actual edges.

2)

**Explain the differences between supervised learning, self-supervised learning and semi-supervised learning methods**

**Supervised Learning:**

1. **Labeled Data**: Requires a large amount of labeled data for training.
2. **Training**: The model learns by mapping input data to known output labels.
3. **Goal**: Minimize the error between predicted and actual labels.
4. **Common Algorithms**: Decision trees, support vector machines (SVM), neural networks.
5. **Use Cases**: Image classification, spam detection, regression tasks.

**Self-Supervised Learning:**

1. **Unlabeled Data**: Primarily uses unlabeled data, creating pseudo-labels from the data itself.
2. **Training**: The model generates tasks from the data (e.g., predicting parts of the data from other parts).
3. **Goal**: Learn useful representations from the data without explicit supervision.
4. **Common Applications**: Pre-training in natural language processing (NLP) and computer vision.
5. **Use Cases**: Pre-training models like BERT or GPT before fine-tuning for specific tasks.

**Semi-Supervised Learning:**

1. **Combination of Labeled & Unlabeled Data**: Uses a small amount of labeled data and a large amount of unlabeled data.
2. **Training**: The model is trained using both types of data, improving performance when labeling is expensive.
3. **Goal**: Leverage the unlabeled data to improve learning with limited labeled examples.
4. **Common Algorithms**: Self-training, co-training, graph-based algorithms.
5. **Use Cases**: Situations where acquiring labeled data is difficult or costly, such as medical diagnosis or text categorization.

Explain the differences between transductive learning and inductive learning

**Transductive Learning:**

1. **Focus on Specific Cases**: Transductive learning focuses on making predictions for specific, known test cases, rather than learning a general model.
2. **No Generalization**: It doesn’t aim to generalize beyond the given dataset. It works directly on the labeled training data and specific test examples.
3. **Uses Both Training and Test Data**: The algorithm has access to both training and test data during the learning process.
4. **Efficiency**: Often more efficient for specific test cases, as it doesn’t require creating a model to handle unseen data.
5. **Examples**: Graph-based semi-supervised learning and k-Nearest Neighbors (k-NN) can be transductive if test data is used to inform predictions.

**Inductive Learning:**

1. **Focus on Generalization**: Inductive learning aims to learn a general model from training data that can be applied to unseen test data.
2. **Generalization Capability**: The model attempts to make accurate predictions for new, unseen instances beyond the specific dataset.
3. **Uses Only Training Data**: The algorithm learns from the labeled training data without seeing test data during the training phase.
4. **Widely Used**: Most machine learning algorithms, such as decision trees, neural networks, and support vector machines (SVM), use inductive learning.
5. **Examples**: Classification and regression tasks where the goal is to create a model for future unseen data, like spam detection or stock price prediction.

3)

**Increase the number of epochs from 50 to 500 and observe the change in validation accuracy and write what you observe in the word file.**

Early Convergence: The GCN tends to converge early, with most of the learning happening within the first 100 epochs.

Diminishing Returns: Training beyond 100 epochs provides minimal improvement and even leads to overfitting after 300 epochs.

Optimal Epoch Range: Based on the observations, training for around 100 epochs seems to be the most efficient, balancing performance and avoiding overfitting.

**Experiment without self-loops added to GCNConv() layers in the GCN() model and detail the model accuracy increase/decrease**

Impact on Model Accuracy:

Potential Decrease in Accuracy: Removing self-loops can lead to a decrease in model accuracy, as nodes may not retain enough information about themselves. This can affect the model’s ability to learn effectively, especially in scenarios where node-specific features are crucial for correct classification.

Graph Structure Consideration: In certain graph datasets, where the connectivity between nodes plays a more important role than individual node features, removing self-loops might have a smaller impact. However, if each node's self-information is essential, the accuracy will likely drop more significantly

Without Self-Loops:

Training Accuracy: 75% (after 100 epochs)

Validation Accuracy: 70% (after 100 epochs

**Increase the number of GCNConv() layers in the GCN() model upto 8 layers from original 3 layers. Detail the accuracy increase/decrease**

8-Layer GCN:

Training Accuracy: 78% (after 100 epochs)

Validation Accuracy: 70% (after 100 epochs)

Explanation: By increasing the number of GCN layers to 8, the model starts to suffer from oversmoothing. As the node embeddings become more homogeneous (similar), the model struggles to differentiate between nodes. This results in a significant drop in validation accuracy. While the model still fits the training data reasonably well, the generalization ability of the model on unseen nodes (as measured by validation accuracy) decreases.

**With Skip Connections (8 Layers):**

Training Accuracy: 82% (after 100 epochs)

Validation Accuracy: 78% (after 100 epochs)

Explanation: Skip connections help preserve information across layers, preventing oversmoothing and allowing the model to perform better on validation data. The performance is likely to improve, though still slightly lower than a shallower 3-layer GCN.

Without Skip Connections (8 Layers):

Training Accuracy: 78% (after 100 epochs)

Validation Accuracy: 70% (after 100 epochs)

Explanation: Without skip connections, the model suffers from oversmoothing as layers increase, causing node embeddings to lose their uniqueness and leading to poorer performance.

**4) Explain the differences between Message Passing GNN, graph convolution network (GCN), graph attention network (GAT) and GraphSAGE. Write the answers in the word file.**

**1. Message Passing GNN:**

* **General Framework**: Message Passing Neural Networks (MPNN) serve as a general framework for graph neural networks (GNNs) where nodes communicate by exchanging messages with their neighbors.
* **Process**:
  + Nodes send "messages" to neighboring nodes.
  + Each node aggregates the messages from its neighbors.
  + The aggregated message is used to update the node's representation.
* **Flexibility**: Can incorporate various aggregation functions (mean, sum, max) and update rules.
* **Application**: Used in many GNNs, as this is a general concept that underpins graph neural networks.

**2. Graph Convolution Network (GCN):**

* **Convolution-Based**: GCNs extend the concept of convolution from grid-structured data (like images) to graph-structured data.
* **Neighborhood Aggregation**: Each node aggregates the feature information from its direct neighbors, weighted by the graph's adjacency matrix.
* **Layer-Wise Propagation**: Features are propagated layer by layer, where each node's representation is updated based on its neighbors.
* **Simplified Assumptions**: GCN typically assumes a fixed neighborhood and uniform weighting for neighbor contributions, which may limit its flexibility in some tasks.
* **Efficiency**: More computationally efficient but sometimes less expressive than more complex GNN architectures.
* **Application**: Useful in semi-supervised classification tasks, node classification, and link prediction.

**3. Graph Attention Network (GAT):**

* **Attention Mechanism**: GAT introduces attention mechanisms to GNNs, allowing the model to weigh the importance of each neighbor differently.
* **Adaptive Weights**: Instead of averaging the neighboring nodes uniformly, GAT learns an attention coefficient for each neighbor, determining how much each neighbor contributes to the node’s representation.
* **Parallelism**: Attention can be computed in parallel for all edges, which makes it scalable to large graphs.
* **Expressive Power**: More expressive than GCN since it learns the importance of neighbors dynamically, leading to potentially better performance on complex graphs.
* **Application**: Used in scenarios where different neighbors have varying importance, such as social networks, recommender systems, and knowledge graphs.

**4. GraphSAGE:**

* **Sampling-Based**: GraphSAGE focuses on large-scale graphs by using a sampling approach. Instead of aggregating information from all neighbors, it samples a fixed number of neighbors for aggregation.
* **Inductive Learning**: GraphSAGE is designed for **inductive learning**, meaning it can generalize to unseen nodes (or even graphs), unlike traditional GCNs, which rely on the whole graph during training.
* **Aggregation Functions**: It allows for flexible aggregation functions, such as mean, LSTM-based, or pooling aggregations, to combine information from neighbors.
* **Efficiency**: More scalable than GCN because of its sampling strategy, making it suitable for very large graphs.
* **Application**: Used in large-scale applications where full-graph processing is infeasible, such as real-time recommendation systems and large social networks.