RAJALAKSHMI ENGINEERING COLLEGE

RAJALAKSHMI NAGAR, THANDALAM – 602 105

RAJALAKSHMI ENGINEERING COLLEGE
An AUTONOMOUS Institution
Affiliated to ANNA UNIVERSITY, Chennai

CP23211 ADVANCED SOFTWARE ENGINEERING LAB

LAB MANUAL

Name :	……………….HEMANTH KUMAR R…………….…………………...
Year / Branch / Section :	………..1st YEAR/ME-CSE ……………………….
University Register No. :	…………2116230711002…..……………………
College Roll No. :	…………230711002………………..………………
Semester :	………………2…………………….……………………………
Academic Year :	……………2023-2024…………………………………….

RAJALAKSHMI ENGINEERING COLLEGE

RAJALAKSHMI NAGAR, THANDALAM – 602 105

BONAFIDE CERTIFICATE

Name: HEMANTH KUMAR R

Academic Year: 2023-2024 Semester: 2 Branch: ME-CSE

Register No: 2116230711002

Certified that this is the bonafide record of work done by the above student in the CP23211-Advanced Software Engineering Laboratory during the year 2023- 2024

Signature of Faculty-in-charge

Submitted for the Practical Examination held on 22/06/2024

Internal Examiner External Examiner

INDEX

SELF SUPERVISED GRAPH NEURAL NETWORKS FOR JOINT EMBEDDING REPRESENTATION AND IMPUTATION

OVERVIEW OF THE PROJECT:

Many real-world datasets suffer from missing data, which makes analysis of data and machine learning very difficult. To keep data intact and make sure downstream analyses work, imputation, or filling in values that are missing, is essential. It is possible that the underlying data structure is not completely captured by traditional imputation methods because they depend on oversimplified assumptions or statistical methodologies. Using self-supervised graph neural networks, we provide a new method for filling in missing data in this research. When it comes to missing values in a graph representation, graph neural networks shine because to their exceptional skills in learning complicated relationship structures in data. A graph structure is used to encode the data in our technique. Nodes in this structure stand for data instances, and edges indicate the links between them. A GNN model may be taught to anticipate missing values using graph topology and node attributes via self-supervised learning, eliminating the need for explicit supervision. When compared to state-of-the-art imputation approaches, our experimental findings show how the self-supervised GNN methodology is more resilient and achieves better imputation accuracy across various data types and patterns of missing data. In conclusion, our research proves that self-supervised graph neural networks are a powerful tool for solving missing data imputation problems across many different industries.

SOFTWARE REQUIREMENTS SPECIFICATION (SRS)

EXP.NO: 1 **DATE: 05/03/2024**

CONTENTS

SELF SUPERVISED GRAPH NEURAL NETWORKS FOR JOINT EMBEDDING REPRESENTATION AND IMPUTATION

1. Introduction

1.1 Purpose

The Purpose of embedding imputation is to fill in missing data in a dataset. In the realm of data analysis and machine learning, missing data is a ubiquitous challenge that can significantly hinder the accuracy and reliability of analyses and predictions. Whether it's medical records, financial transactions, or social network interactions, incomplete datasets are a common occurrence, stemming from various reasons such as measurement errors, equipment malfunction, or intentional omission. Addressing missing data is thus paramount for extracting meaningful insights and making informed decisions from the available information.

1.2 Scope

Self-supervised graph neural networks for joint embedding representation and imputation can be applied across diverse domains such as healthcare, finance, social networks, e-commerce, transportation, NLP, and manufacturing. These technologies enhance data analysis by learning meaningful representations from graph structures.

2. Overall Description

2.1 Product Perspective

The project on self-supervised graph neural networks offers a versatile solution adaptable to diverse data environments. By autonomously learning embeddings from graph-structured data, it caters to various domains like healthcare, finance, e-commerce and NLP, ensuring robust predictive capabilities and decision support tools that continuously improve with user feedback.

2.2 Features

2.2.1 Joint Embedding and Imputation

- Utilizes self-supervised graph neural networks (GNNs) to autonomously learn and enhance joint embeddings from complex graph-structured data, accommodating missing information and optimizing predictive accuracy across various domains.

- Integrates adaptive learning mechanisms to refine embeddings based on extensive datasets, ensuring robust representation and imputation capabilities suitable for diverse applications.

- Provides flexibility to customize embedding representations to specific application needs, facilitating tailored solutions for healthcare diagnostics, financial predictions, social network analysis, and more.

2.2.2 Predictive Accuracy Enhancement

- Utilizes learned patterns to automatically impute missing data within graph structures, enhancing the completeness and reliability of predictive models.

- Incorporates feedback loops to continuously refine imputation algorithms, ensuring adaptability and reliability in dynamic data environments.

- Enables scalable application across large-scale datasets and complex networks, supporting efficient decision-making and data-driven insights in real-time scenarios.

2.3 User Classes and Characteristics:

- Utilizes the system for advanced data analysis and modeling tasks, prioritizing accuracy and scalability in handling complex relational data.

- Leverages the technology for comprehensive research and development initiatives, benefiting from enhanced predictive capabilities and robust decision support tools.

- Applies the framework in specialized fields such as healthcare, finance, and social sciences, relying on accurate embeddings and imputation for critical insights and strategic planning.

3. Specific Requirements

3.1 Functional Requirements:

3.1.1 Joint Embedding and Imputation

- Users should have the capability to generate joint embeddings from graph-structured data.

- Implement algorithms to effectively handle missing data within the graphs.

- Provide an intuitive interface for users to interact with and visualize the embeddings and imputed data.

3.1.2 Predictive Modeling

- Allow users to train predictive models using the generated embeddings.

- Provide tools to evaluate the performance of the trained models in terms of accuracy, precision, recall, etc.

- Ensure seamless integration with existing data analysis pipelines and frameworks.

3.1.3 Application Flexibility

- Enable customization of embedding generation and imputation techniques based on specific user requirements and domain characteristics.

- Support real-time processing capabilities for timely decision-making and data-driven insights.

- Include visualization tools to aid in the interpretation and understanding of complex graph data structures and their embeddings.

3.2 Non-Functional Requirements:

3.2.1 Usability

- The interface should be user-friendly and intuitive.

- Should be usable across all the platforms.

3.2.2 Performance

- Optimize processing speed and efficiency to handle large-scale graph data and complex computations effectively

- Scalability to accommodate a growing user base without compromising performance.

3.2.3 Security

- Implement secure user authentication and authorization.

- Implement robust security measures to protect sensitive data, including encryption during transmission and storage.

4. External Interface Requirements

4.1 User Interfaces:

- Intuitive design with easy navigation.
- Consistent theme and layout for a seamless user experience.

4.2 Hardware Interfaces:

- Compatible with Windows, Mac, Android and iOS devices.

4.3 Software Interfaces:

- Integration with database for continuous model training.
- File storage integration for storing and accessing enhanced embeddings and imputed data.

5. Conclusion

In conclusion, our proposed approach for imputing missing data in graph-structured datasets using self-supervised graph neural networks offers a promising solution to a pervasive problem in data analysis and machine learning. By leveraging the expressive power of graph neural networks and the self-supervised learning framework, our method effectively captures complex relational dependencies and accurately predicts missing values, thereby enhancing the integrity and reliability of analyses conducted on incomplete datasets.

SCRUM METHODOLOGY

EXP.NO: 2 **DATE: 14/03/2024**

1. Introduction

The self-supervised graph neural network aims to provide an intuitive and comprehensive solution for users to enhance their datasets by predicting and imputing missing values. The Agile Scrum framework will guide the development process, ensuring iterative enhancements and timely delivery of key features.

2. Objectives

- Implement iterative development with sprints to continuously enhance self-supervised graph neural network algorithms.
- Form cross-functional teams to integrate diverse skills necessary for effective embedding and imputation techniques.
- Utilize Scrum's adaptability to respond swiftly to emerging research and project requirements throughout development.

3. Product Backlog Introduction

The product backlog is a dynamic list of features, enhancements, and fixes prioritized by the product owner. It serves as a road-map for the development team.

4. Product Backlog

4.1 For Users

- **Graph Embedding and Imputation Framework**
  - Develop and implement a self-supervised GNN framework for joint embedding representation and imputation.
  - Ensure the framework supports diverse graph data structures and scales effectively with data size.

- **Embedding Quality Evaluation**
  - Provide users with tools to quantify embedding quality metrics such as node similarity preservation and embedding stability.
  - Enable visualization of embeddings in low-dimensional spaces to aid in interpretability and analysis.

- **Imputation Accuracy Enhancement**

- Research and integrate advanced GNN techniques to enhance imputation accuracy for missing node attributes.

- **User interface for input and output**
  - Design a user-friendly interface allowing seamless uploading and preprocessing of diverse graph datasets.

5. User Stories

- As a data scientist, I want to use a self-supervised GNN framework to generate joint embedding representations of graph data, enabling more accurate imputation of missing attributes.
- As a researcher, I need tools to evaluate the quality and interpretability of graph embeddings produced by the neural network, facilitating deeper insights into underlying data structures.
- As a machine learning engineer, I require advanced GNN techniques that enhance imputation accuracy, particularly for datasets with complex graph topologies and missing data patterns.
- As a user, I want an intuitive interface where I can easily input graph data, visualize embedding results, and validate imputation outcomes to support decision-making processes.

6. Sprint

- A time-boxed iteration during which a set of user stories is implemented and tested.

7. Sprint Backlog

The sprint backlog is a list of tasks selected from the product backlog for a specific sprint. In other terms Sprint Backlog could also be defined as the subset of Product backlog which is chosen for a specific sprint. In general a sprint backlog allows the development team to work on the tasks necessary to implement the User Stories within the selected sprint.

8. Sprint Review

A meeting held at the end of each sprint to review and demonstrate the completed work.

Sprint 1 Review:

- The sprint review is a meeting that includes the demonstration of implemented features at that particular sprint that is conducted at the end of each sprint.

- In the Underwater image enhancement app the sprint backlog for the first week is the Use case Diagram and the sprint backlog for the second use case is Software Requirement Specification document.

9. Software Used

- **Development Platform:** Jupyter Notebook, PyCharm

10. Conclusion

In conclusion, our proposed approach for imputing missing data in graph-structured datasets using self-supervised graph neural networks offers a promising solution to a pervasive problem in data analysis and machine learning. By leveraging the expressive power of graph neural networks and the self-supervised learning framework, our method effectively captures complex relational dependencies and accurately predicts missing values, thereby enhancing the integrity and reliability of analyses conducted on incomplete datasets.

USER STORIES

EXP.NO: 3 **DATE: 26/03/2024**

1. Data Acquisition and Preprocessing

User Story: As a data scientist, I want to upload raw data from various sources to prepare it for the graph neural network model.

Acceptance Criteria:

- The system allows uploading data files in various formats (e.g., CSV, TXT).
- The user can browse and select files from their local storage.
- The system provides options to specify data format details (e.g., delimiters, header row).

2. Organize Study Materials

User Story: As a data scientist, I want to explore and understand the structure of the uploaded data to identify nodes, edges, and their attributes.

Acceptance Criteria:

- The system displays a preview of the uploaded data, including a sample of rows and columns.
- The user can identify columns containing node IDs, edge information, and node features.
- The system provides basic data statistics.

3. Graph Construction and Preprocessing

User Story: As a data scientist, I want to define how nodes and edges are represented in the graph based on the uploaded data.

Acceptance Criteria:

- The system allows specifying which columns in the data represent nodes and edges.
- The user can define how edge attributes (weights, directions) are extracted from the data.
- The system offers options to handle missing edges or create them based on heuristics.

4. Model Training and Evaluation

User Story: As a data scientist, I want to train the self-supervised graph neural network model with appropriate hyperparameters.

Acceptance Criteria:

- The system allows specifying the model architecture.
- The user can define training hyperparameters.

- The system provides options for selecting an appropriate loss function for the self-supervised learning task.

5. Embedding Visualization and Analysis

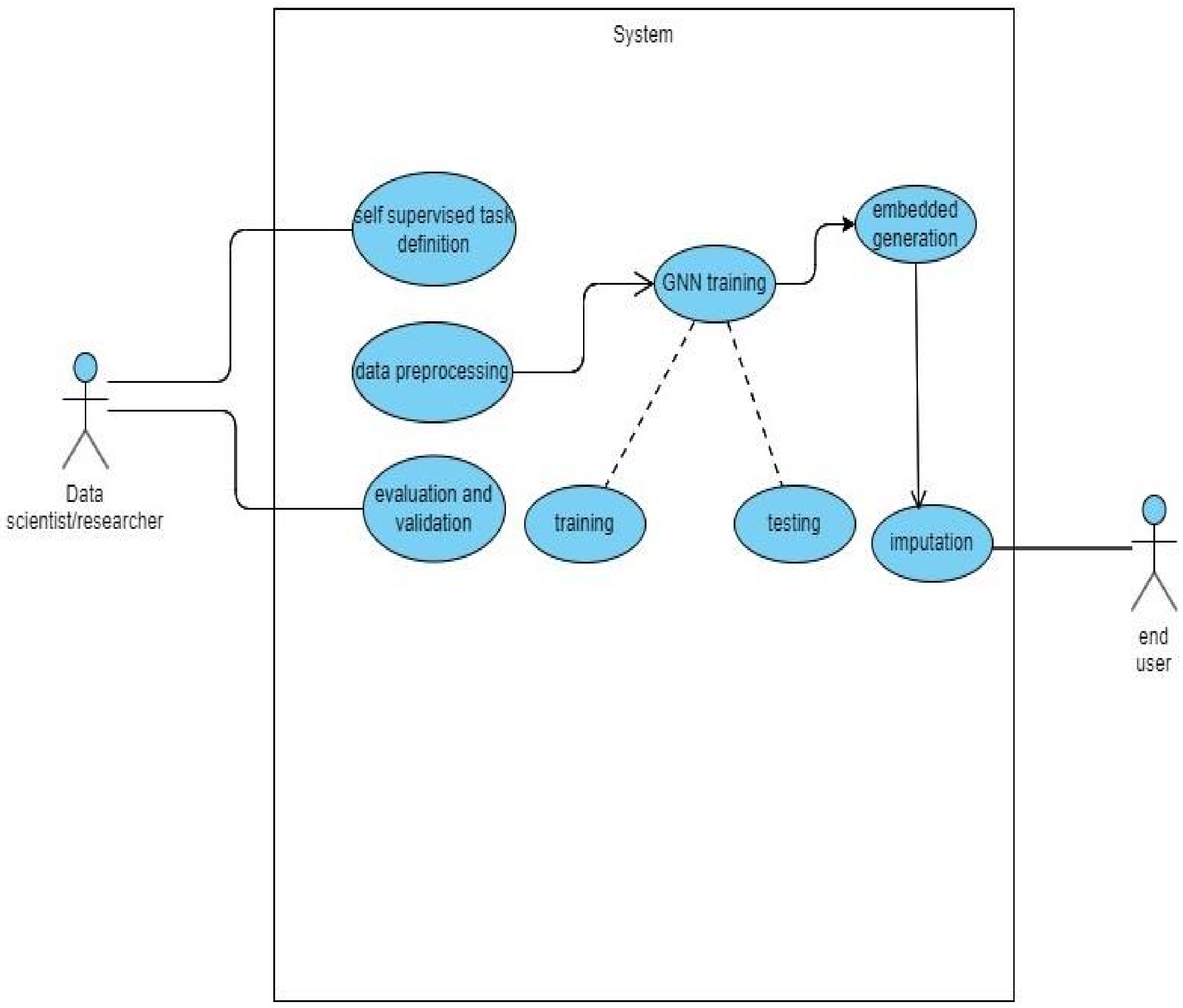
User Story: As a data scientist, I want to visualize the learned node embeddings from the trained model.

Acceptance Criteria:

- The system allows visualizing node embeddings in a low-dimensional space.
- The user can explore how nodes are grouped based on their learned representations.

USE CASE DIAGRAM

EXP.NO: 4 **DATE: 04/04/2024**



NON-FUNCTIONAL REQUIREMENTS

EXP.NO: 5 **DATE: 16/04/2024**

1. Performance

The Self Supervised GNN should be highly responsive, with all major functions taking no more than 2 seconds to load. The GNN should handle up to 500 concurrent users without performance degradation, ensuring smooth and efficient user experience during peak times such as thebeginning of a semester.

2. Security

The GNN must implement robust security measures to protect user data. All user information, including login credentials, personal details, and stored notes, must be encrypted both in transit and at rest. Access to the GNN should be controlled via secure authentication mechanisms, and administrators should have the ability to set and enforce password policies. The app should also include role-based access controls to ensure thatusers only have access to the features and data appropriate to their role.

3. Usability

The Self Supervised GNN should provide an intuitive and user-friendly interface that is easy to navigate for all users, including those with limited technical skills. It should adhere to common usability principles, providing clear labels, logical flow, and accessible help documentation. The GNN should support both desktop and mobile platforms, ensuring a consistent and pleasant user experience across devices.

4. Reliability

The GNN must be reliable, with an uptime of at least 99.9%. It should include mechanisms for automatic failover and recovery to minimize downtime and ensure continuous availability. Regular backups should be performed to prevent data loss, and in the event of a failure, the system should recover without data corruption or significant downtime.

5. Scalability

The Self Supervised GNN should be scalable to accommodate future growth in the number of users and the volume of data. The system architecture should support the addition of new features and integration with other educational tools without requiring significant redesign. The app should be able to handle an increasing number of simultaneous users and data entries with minimal impact on performance.

6. Maintainability

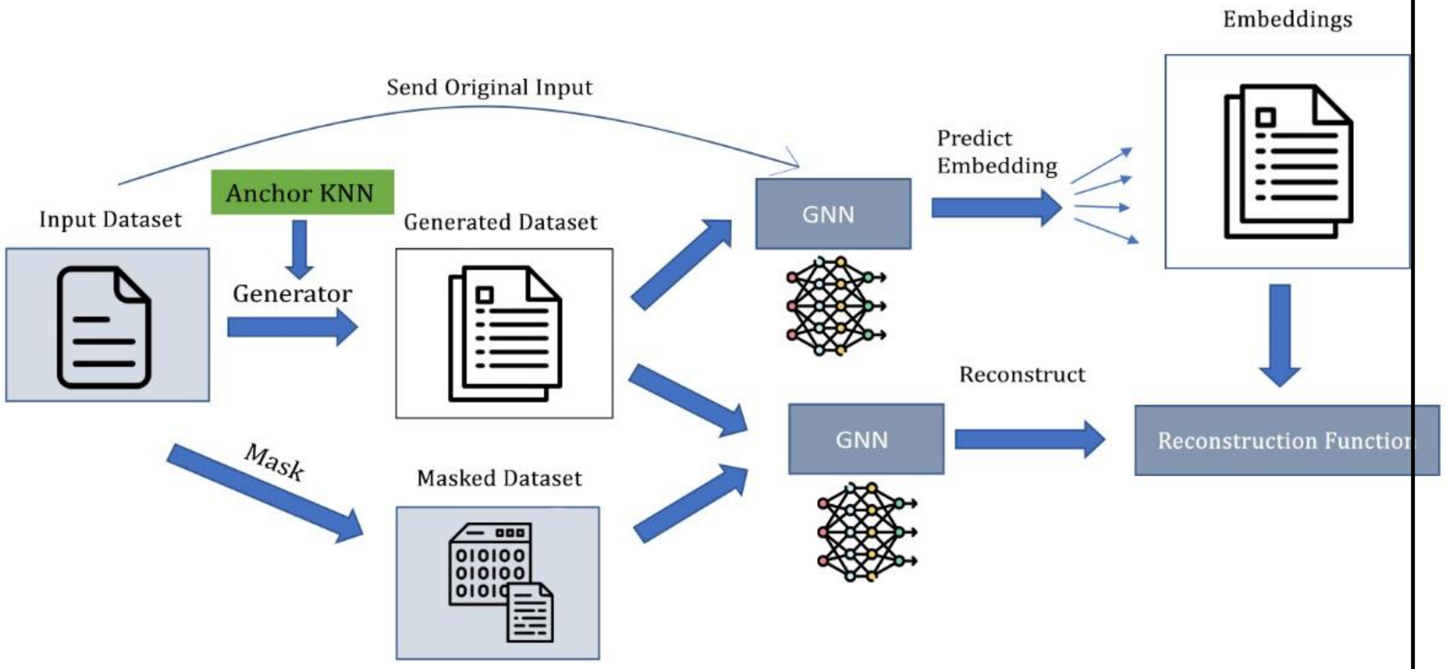
The GNN should be designed with maintainability in mind, enabling easy updates and bug fixes. The codebase should follow standard coding practices and be well-documented to facilitate understanding and modifications by different developers. Modular design principles should be applied, allowing individual components to be updated or replaced without affecting the entire system.

7. Compliance

The Self Supervised GNN must comply with relevant regulations and standards, including data privacy laws such as GDPR for users in Europe. The GNN should ensure that all data handling practices meet legal requirements for data protection and user privacy. Regularaudits should be conducted to verify compliance, and any necessary adjustments shouldbe made promptly to address new regulatory changes.

OVERALL PROJECT ARCHITECTURE

EXP.NO: 6 **DATE: 25/04/2024**



Data Preprocessing

Transform the data set provided into a graph style representation, with nodes representing entities and edges representing relationships. Identify important properties and traits of each object by extracting node characteristics from the observed data. Make the graph more realistic by adding missing values, which represent situations in the real world when certain data points are either missing or not accessible.

Model Architecture

Create an architecture for a graph neural network that can process data that is organized in a graph. Specify the GNN model's architecture, including its activation functions, number of layers, and message carrying mechanism type (e.g., Graph Attention Networks, Graph Convolutional Networks). Build in ways to deal with missing values into the GNN design so the model can successfully fill them in using the data that is available and the graph's relational structure.

Self-Supervised Learning

Create a self-supervised learning job within the graph architecture to fill in missing data. Make sure the GNN model is motivated to learn accurate depictions given the input data by defining a pretext task. In the pretext task, which aims to train the model by predicting missing values from observed data and graph topology, the model is trained. Iteratively update the GNN model's parameters using backpropagation to minimize a predetermined loss function and optimize the forecasting of missing data.

Training Procedure

Make three separate sets, one for training, one for validation, and one for testing. Make use of the training set and the self-supervised learning structure to train the GNN-based restoration model. Check the model's accuracy and performance using important metrics like loss and imputation on the validation set. Adjust the model's hyperparameters as needed according to the validation results.

Imputation Process

Complete the dataset by impinging missing values using the learned GNN-based imputation model on additional, unseen data. Use the network topology and the learnt representations to generalize to new cases and deal with missing data in different ways. Make sure the data remains accurate and trustworthy after the imputation process by validating the imputed dataset.

Evaluation Metrics

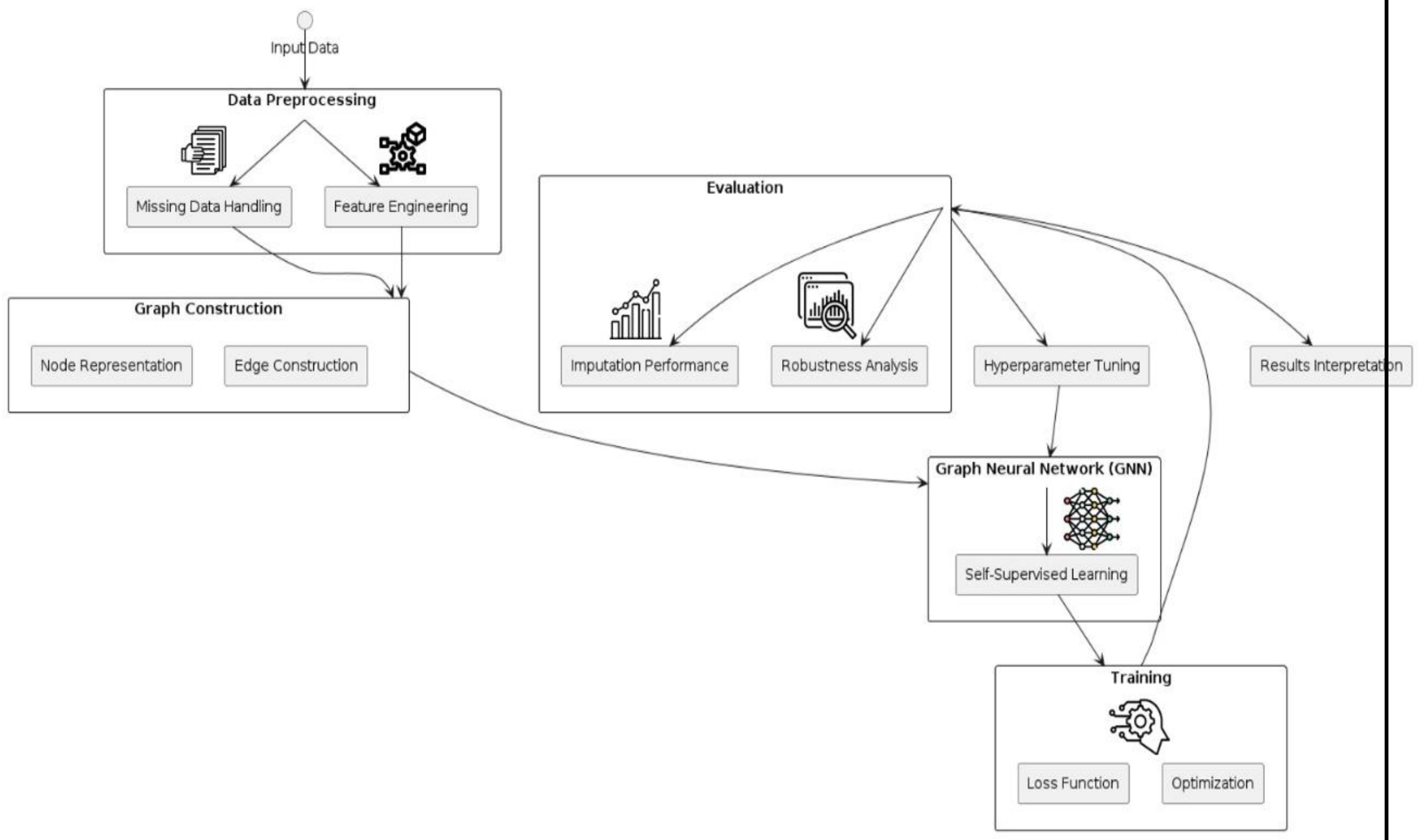
To correctly evaluate the imputation model's performance, suitable assessment measures must be defined. The most common metrics include imputed accuracy, R^2, mean absolute error, and root mean squared error (RMSE). To measure how well the model handles missing data, these metrics compare the imputed values to the ground truth and provide an idea of how accurate and reliable they are.

Robustness Analysis

Test the model's resilience by running it through a series of simulations with varied missing data levels, dataset sizes, and missing data procedures (MCAR, MAR, MNAR). By evaluating the model's predictive stability and generalizability over various datasets and situations, robustness analysis sheds light on the model's possible limits and its usefulness in real-world applications.

BUSINESS ARCHITECTURE DIAGRAM

EXP.NO: 7 **DATE: 02/05/2024**



Actor:

- **Data engineers and scientists**: Represents the primary users who Responsible for handling the preprocessing of the data, including missing data handling and feature engineering.

Frontend Use Cases

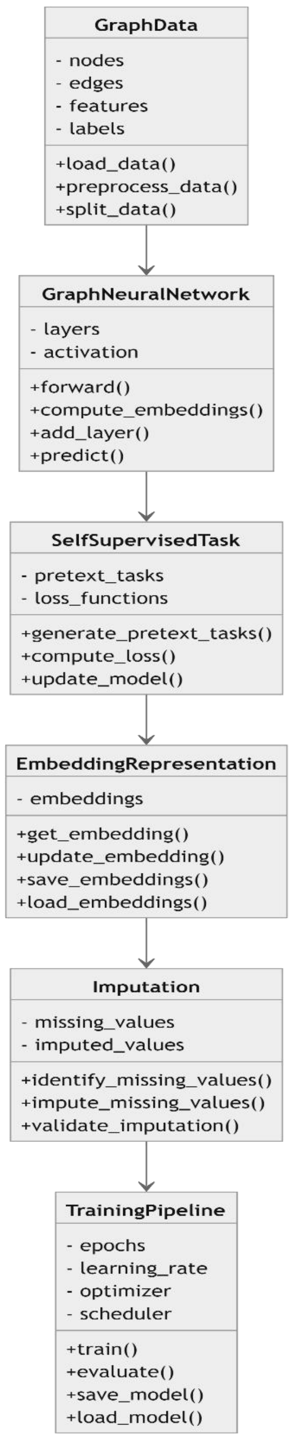
- **Data Visualization Tools:**Interfaces to visualize the input data, feature engineering processes, and results interpretation.
- **Dashboards for Performance Monitoring:**Display metrics related to imputation performance, robustness analysis, and training progress.
- **Interactive Graphs:**Allow users to explore the constructed graph, examining node representations and edge connections.

Backend Use Cases

- **Data Preprocessing Pipelines:**Automated pipelines to handle missing data and perform feature engineering.
- **Graph Construction Algorithms:**Backend services to define node representations and construct edges based on data relationships.
- **Model Training Infrastructure:**Scalable infrastructure for training the GNN, including distributed computing resources for large datasets.
- **Evaluation Metrics Computation:**Backend processes to compute imputation performance and conduct robustness analysis.
- **Hyperparameter Tuning Services:**Automated services to perform hyperparameter tuning using techniques like grid search or random search.
- **Result Storage and Retrieval:**Databases to store model outputs and evaluation results, accessible for further analysis and reporting.

CLASS DIAGRAM

EXP.NO: 8 **DATE: 07/05/2024**



Classes:

1. **Graph Data:** The Graph Data class is the foundation of the pipeline, representing the raw data in graph format. It includes attributes such as nodes, edges, features, and labels, which define the structure and properties of the graph.

2. **Graph Neural Network:** The Graph Neural Network class leverages the data prepared by Graph Data. It contains layers, which represent the architecture of the neural network, and activation, which defines the activation functions used within the network.

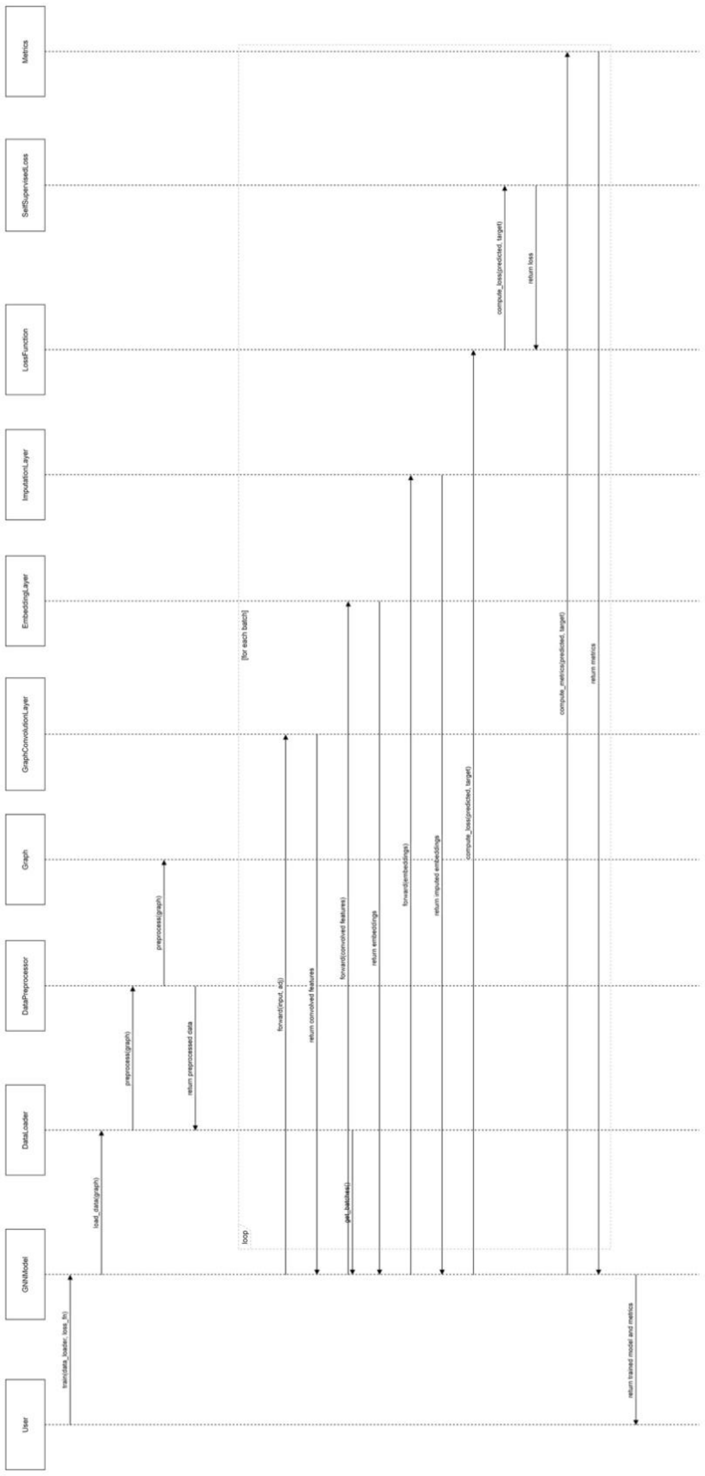
3. **Self-Supervised Task**: The Self Supervised Task class is crucial for implementing self-supervised learning within the pipeline. It includes attributes like pretext_tasks and loss_functions, which define the tasks and corresponding loss functions used to train the model without labeled data.
4. **Embedding Representation**: The Embedding Representation class manages the node embeddings created by the GNN. It has an attribute embeddings, which stores these embeddings. Methods in this class include get_embedding(), to retrieve embeddings for individual nodes, update_embedding(), to modify existing embeddings, and save_embeddings() and load_embeddings(), which handle the storage and retrieval of embeddings from files.
5. **Imputation**: The Imputation class addresses the problem of missing values in the data. It includes attributes missing_values and imputed_values, which store the original missing data points and the values imputed during processing. The methods identify_missing_values(), impute_missing_values(), and validate_imputation() are used to detect missing values, perform imputation, and validate the results of the imputation process.
6. **Training Pipeline**: The Training Pipeline class orchestrates the overall training process of the GNN. It includes attributes such as epochs, learning_rate, optimizer, and scheduler, which define the training parameters. The methods train(), evaluate(), save_model(), and load_model() manage the training loop, evaluate the model performance, save the trained model to disk, and load a pre-trained model, respectively.

Relationships:

The Graph Data class provides preprocessed data, including nodes, edges, and features, to the Graph Neural Network class, which uses this data to construct and train the GNN model. The Graph Neural Network class then interacts with the Self Supervised Task class to generate pretext tasks and compute losses necessary for self-supervised learning. The Self Supervised Task class relies on the Embedding Representation class to manage and update the node embeddings during training. These embeddings are utilized by the Imputation class to handle missing data, where the learned representations help in accurately imputing missing values. The Imputation class, in turn, interacts with the Training Pipeline class to validate the imputation results and ensure the training process incorporates the imputed data effectively.

SEQUENCE DIAGRAM

EXP.NO: 9 **DATE: 16/05/2024**



Actor:

- **Data engineers and scientists**: Represents the primary users who Responsible for handling the preprocessing of the data, including missing data handling and feature engineering.

System Components

- **Actor (User)**: Represents the external user interacting with the system.
- **Frontend**: The user interface through which the actor interacts with the system.
- **Backend**: The server-side application logic that processes requests from the frontend.
- **DataLoader**: Component responsible for loading data into the system.
- **Preprocessor**: Handles preprocessing tasks like cleaning and transforming the data.
- **GraphConstructor**: Constructs the graph structure from the preprocessed data.
- **GNNTrainer**: Manages the training process of the Graph Neural Network (GNN).
- **SelfSupervisedTask**: Handles self-supervised learning tasks.
- **EmbeddingManager**: Manages the embeddings generated by the GNN.
- **ImputationModule**: Performs data imputation tasks using the embeddings.
- **Evaluator**: Evaluates the performance of the trained model.
- **ModelSaver**: Saves the trained model for future use.

Sequence of Events

1. **Initialization**

- The sequence begins with the user initiating a process through the frontend.
- The frontend sends a request to the backend to start the process.

2. **Data Loading**

- The backend calls the DataLoader to load the necessary data.
- The DataLoader loads the data and returns it to the backend.

3. **Preprocessing**

- The backend forwards the loaded data to the Preprocessor.
- The Preprocessor cleans and transforms the data and then sends it back to the Backend.

4. **Graph Construction**

- The backend sends the preprocessed data to the GraphConstructor.

- The GraphConstructor constructs the graph structure and returns it to the backend.

5. **Training GNN**

- The backend calls the GNNTrainer to train the Graph Neural Network.
- The GNNTrainer initializes the training process and performs the forward pass, computes the loss, and updates the model parameters iteratively.

6. **Self-Supervised Learning**

- During the training, the GNNTrainer interacts with the SelfSupervisedTask component.
- The SelfSupervisedTask generates pretext tasks and computes the loss for self-supervised learning.

7. **Managing Embeddings**

- The GNNTrainer uses the EmbeddingManager to manage node embeddings.
- The EmbeddingManager retrieves and updates embeddings as needed during the training process.

8. **Data Imputation**

- The ImputationModule is used to perform imputation on the data using the learned embeddings.
- The ImputationModule identifies missing values, imputes them, and validates the imputed values.

9. **Evaluation**

- The backend sends the trained model and imputed data to the Evaluator.
- The Evaluator assesses the model's performance and returns the evaluation results to the backend.

10. **Saving the Model**

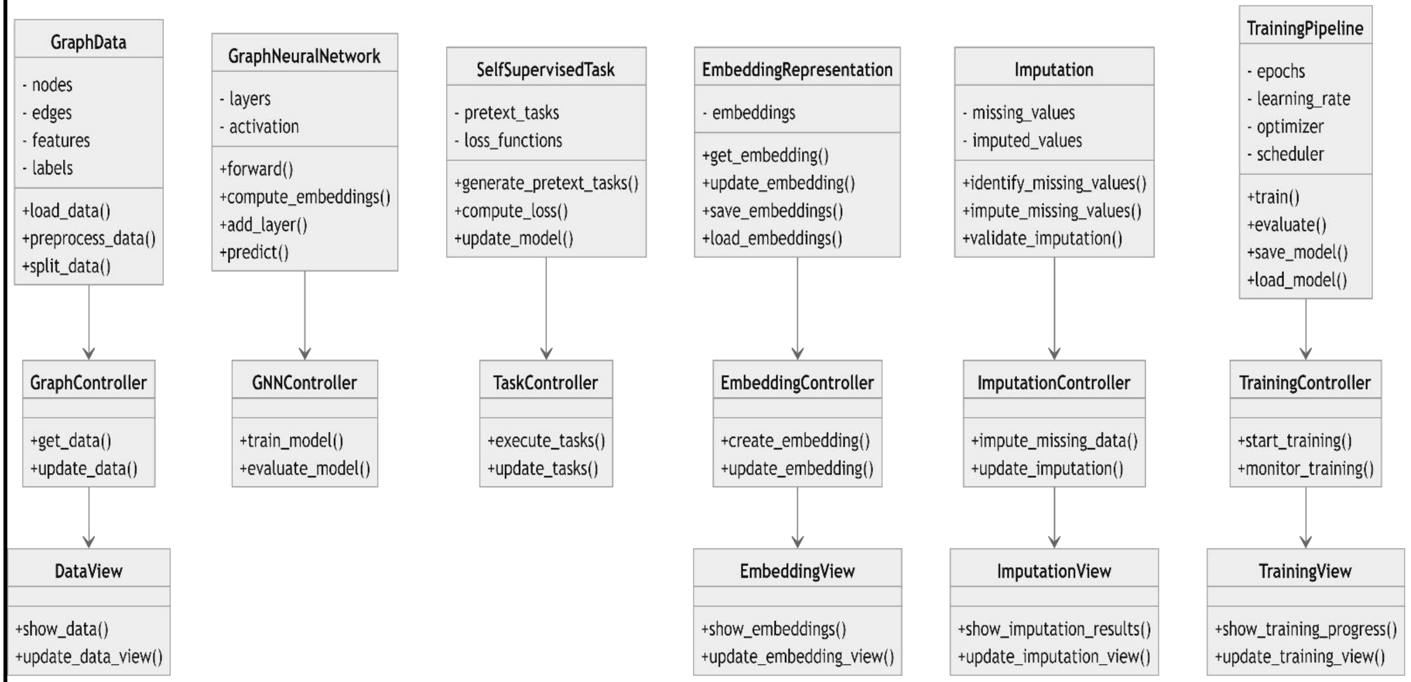
- The backend calls the ModelSaver to save the trained model.
- The ModelSaver stores the model, making it available for future use.

11. **Completion**

- The backend sends the results and any additional information back to the frontend.
- The frontend displays the results to the user, completing the process.

ARCHITECTURAL PATTERN (MVC)

EXP.NO: 10 **DATE: 28/05/2024**



Model:

- **GraphData:** Handles graph-related data including nodes, edges, features, and labels. It includes methods for loading, preprocessing, and splitting data.
- **GraphNeuralNetwork:** Represents the neural network model used for graph data. It includes methods for forwarding data, computing embeddings, adding layers, and making predictions.
- **SelfSupervisedTask:** Manages tasks and loss functions for self-supervised learning. It includes methods for generating pretext tasks, computing loss, and updating the model.
- **EmbeddingRepresentation:** Manages embeddings of the data. It includes methods for retrieving, updating, saving, and loading embeddings.
- **Imputation:** Deals with identifying and imputing missing values in the data. It includes methods for identifying, imputing, and validating missing values.
- **TrainingPipeline:** Manages the training process including epochs, learning rate, optimizer, and scheduler. It includes methods for training, evaluating, saving, and loading the model.

Controller:

- **GraphController:** Manages data operations for the GraphData model. It includes methods for getting and updating data.
- **GNNController:** Manages operations for the GraphNeuralNetwork model. It includes methods for training and evaluating the model.
- **TaskController:** Manages tasks related to the SelfSupervisedTask model. It includes methods for executing and updating tasks.
- **EmbeddingController:** Manages operations related to embedding representations. It includes methods for creating and updating embeddings.
- **ImputationController:** Manages operations related to data imputation. It includes methods for imputing missing data and updating imputation.
- **TrainingController:** Manages the training process. It includes methods for starting and monitoring training.

View:

- DataView: Displays data managed by GraphController.
- EmbeddingView: Displays embeddings managed by EmbeddingController.
- ImputationView: Displays imputation results managed by ImputationController.
- TrainingView: Displays training progress managed by TrainingController.

Relationships:

- The Model, View, and Controller components interact as follows:

- The system architecture employs a clear separation of concerns by dividing responsibilities among Models, Controllers, and Views, enhancing maintainability and scalability.

- Each Model has a corresponding Controller that manages specific operations, such as GraphController with GraphData for graph-related tasks and GNNController with GraphNeuralNetwork for neural network training.

- Controllers also update Views, like GraphController updating DataView and EmbeddingController updating EmbeddingView, ensuring real-time data visualization and interaction.

- This structure ensures organized, consistent interactions between user actions and data processing, maintaining a clean, scalable, and user-responsive system.