Assignment: Entity Feature Initialization with LLMs and Graph Representation Learning with GNNs on CoDEx Dataset

Objective

To design a knowledge graph representation model that leverages both structured graph data from the **CoDEx** dataset and unstructured text-based features extracted from a large language model (LLM). You will initialize entity features using the LLM and train a GNN to predict missing links based on these enhanced representations.

Total Marks: 200

Task Overview

1. **Data Preprocessing (30 marks)**:
    - **Download and Parse the CoDEx Dataset**:
        - Clone the CoDEx dataset repository from GitHub: CoDEx on GitHub.
        - Load and parse the CoDEx dataset files. Specifically, extract the triples from `codex-s`, `codex-m`, or `codex-l` datasets (you may choose one based on your computational resources).
    - **Extract Entity Descriptions**:
        - For each entity in the dataset (that you chose earlier), extract any available textual descriptions, names, or labels that can serve as input text for an LLM.
        - If direct descriptions are not available, use the entity names as input for the LLM to generate feature embeddings.
    - **Preprocess Relation Information**:
        - Identify and process the relations in the dataset. Use these relationships to structure the input graph and create an adjacency list or matrix as required by your GNN model.
2. **Entity Feature Initialization with LLM (40 marks)**:
    - **LLM-based Embedding Generation**:
        - Use a large language model (e.g., BERT, RoBERTa, Llama, or GPT) to generate embeddings for each entity based on its textual description or name.
    - **Feature Extraction and Embedding Storage**:
        - Store the entity embeddings as feature vectors. Ensure that these feature vectors are correctly indexed and prepared for input into your GNN.

        - Evaluate the feature embeddings for quality by visualizing them using dimensionality reduction techniques (e.g., PCA or t-SNE).
3. **Graph Neural Network Model Design (50 marks)**:
    - **GNN Architecture Selection**:
        - Choose a suitable GNN model architecture (such as Graph Convolutional Network, Graph Attention Network, or R-GCN) that can handle multi-relational data.
        - Describe your choice of architecture (in the report), specifying how you plan to handle the heterogeneous nature of the relations in the dataset.
    - **Model Implementation**:
        - Implement the chosen GNN architecture, using the LLM-based embeddings as initial node features and the adjacency matrix derived from the CoDEx triples.
        - Ensure that your model is capable of learning both entity and relation representations to enable effective link prediction.
4. **Knowledge Base Completion Task (50 marks)**:
    - **Training and Testing Setup**:
        - **KG completion**: Given a set of training triples, predict a set of new test triples. For evaluation, for each test triple, (`head`, `relation`, `tail`), the model is asked to predict `tail` entity from (`head`, `relation`). Please assume that every validation/test triple is not present during training, so training triples should not be predicted even if they are correct. More specifically, for each (`head`, `relation`), the model is asked to predict the top 10 tail entities that are most likely to be positive.
        - Split the dataset into training, validation, and test sets based on the existing splits provided by CoDEx or by creating your own split.
        - Use the GNN model to perform link prediction, where the model is trained to predict missing or unseen triples in the knowledge graph (as mentioned above).
    - **Evaluation and Metrics**:
        - Report on performance using metric such as MRR. The goal is to rank the ground-truth tail entity as high in the rank as possible within the top 10, which is measured by Mean Reciprocal Rank (MRR). Also, use Hits@k (for k = 1, 3, and 10) as your metric.
        - Compare the performance of the GNN model with random feature initialization versus LLM-based initialization.
    - **Hyperparameter Tuning**:
        - Experiment with different hyperparameters for the GNN, such as learning rate, number of layers, hidden dimension size, and dropout.
        - Document the tuning process and report on the combination of hyperparameters that provides the best performance.
5. **Analysis and Report (30 marks)**:
    - **Analysis of LLM Embedding Impact**:

        - Analyze the impact of using LLM-based embeddings versus random initialization on the GNN's performance. Identify specific relations or entity pairs where LLM embeddings make a notable difference.
    - **Report Findings and Insights**:
        - Provide a comprehensive report detailing your approach, including the data preprocessing, model architecture, and training process.
        - Discuss the challenges encountered and the solutions implemented. Highlight any interesting observations, such as how LLM embeddings enhance or fail to enhance specific types of relationships.
    - **Conclusion and Future Work**:
        - Summarize the overall findings and propose potential extensions or improvements for future work, such as additional tuning or architectural changes that could further leverage LLM embeddings for graph-based tasks.

Note:

1. For suggestions on the implementation of GNN models, one can use the codex implementations (should be implemented by oneself and not merely a function call from the codex library) or any of the *GNN-based non-ensemble* methods (again, own implementation) in https://ogb.stanford.edu/docs/lsc/leaderboards/#wikikg90mv2.