Knowledge Graph and Inference for Drug-Disease-Gene Interactions

Project Summary

This project integrates biomedical data (drugs, diseases, proteins, genes) into a knowledge graph and applies inference techniques to generate new insights. It reads input files of drug/disease features and known interactions, extracts relational triples, and constructs an RDF-like graph using NetworkX. Rule-based logic is then applied to infer possible new drug–disease associations (for example, inferring that Drug A may treat Disease Y if Drug A treats Disease X and X and Y share a common gene). Finally, Bayesian networks (via pgmpy) are built to perform probabilistic inference on drug efficacy or gene involvement given observed data. This end-to-end workflow, implemented in a Jupyter Notebook, demonstrates how knowledge engineering methods can support hypothesis generation in bioinformatics.

Environment Setup

- Use a Python 3.8+ environment (Anaconda recommended). Create and activate a new environment, e.g. conda create -n kg_env python=3.8 and conda activate kg_env.
- Install the required libraries: run pip install -r requirements.txt (or use conda install for listed packages).
- Ensure Jupyter Notebook is installed to execute the notebook.

Input Files

The notebook expects three input files in the working directory:

- **kg.csv**: A CSV file containing knowledge graph data, typically with columns for subject, predicate, object (e.g., drug–disease–gene relationships).
- **disease_features.tab**: A tab-separated file with numerical features or gene association scores for diseases.
- **drug_features.tab**: A tab-separated file with numerical features or target gene indicators for drugs.
  These files are read and processed in the preprocessing module to extract triples and normalize features, forming the basis of the graph.

Module Descriptions

Data Preprocessing Module

- Load **kg.csv** into a panda DataFrame and parse each row as a triple: (subject, predicate, object).
- Read **disease_features.tab** and **drug_features.tab** with pandas (sep='\t') to obtain feature tables.
- Normalize or standardize numerical features (e.g., using min-max scaling or z-score) so that different measurements are on comparable scales.
- Build mapping dictionaries that assign a unique integer ID to each entity (drug, disease, protein, gene) and each relation/predicate encountered. This ID mapping simplifies downstream processing.

- Output cleaned triple lists and mapped feature data structures for use in subsequent modules.

Knowledge Graph Construction Module

- Initialize a directed (or multi-) graph using NetworkX (e.g. nx.MultiDiGraph to allow multiple edge types).
- For each triple from preprocessing, add nodes for the subject and object if they don't already exist. Tag nodes by entity type (drug, disease, gene, etc.) in the node attributes.
- Add a directed edge for each triple, using the predicate as the edge label. Store any additional metadata (e.g., evidence scores or weights) as edge attributes.
- After graph construction, compute and output basic statistics (such as the number of nodes and edges, or counts by entity type). The graph structure can also be visualized or exported if needed.

Rule-Based Inference Module

- **Co-treatment rule**: If *Drug A* Treats *Disease X* and *Disease Y* shares a common gene *Z* with *Disease X*, then infer that *Drug A* may potentially treat *Disease Y*. (This leverages the idea that shared genes imply related disease mechanisms.)
- **Co-expression rule**: If *Drug A* affects *Gene G* and *Gene H* is co-expressed or functionally linked with *Gene G*, then infer that *Drug A* might influence pathways involving *Gene H*.
- The code implements these rules by scanning the graph for matching patterns. For each match, it generates a new inferred triple, for example (DrugA, 'potentially_treats', DiseaseY), along with the supporting rationale (e.g., shared gene Z).
- The notebook lists these inferred relations. For instance, it might output:
    - "Inferred: DrugA may treat DiseaseY (shared gene: GeneZ)."

Bayesian Modeling Module

- Define a Bayesian Network using pgmpy.models.BayesianModel. Nodes represent variables such as *drug effectiveness*, *presence of a gene*, or *disease response*. Edges represent conditional dependencies (e.g., a gene node influencing a drug efficacy node).
- Specify Conditional Probability Distributions (TabularCPDs) for each node, encoding hypothetical probabilities. For example, a CPD might specify P(DrugEffective | GeneG).
- Use VariableElimination (from pgmpy.inference) to perform probabilistic queries. For example, given evidence that GeneZ = 1 (gene is present/active), compute P(DrugX_effective | GeneZ=1).
- Run example queries in the notebook and display the results. This might print outputs like:

    P(DrugX_effective = True | GeneZ = 1) = 0.78

Such results illustrate how the model estimates drug efficacy probabilities under observed conditions.

Sample Outputs and Use Cases

- **Graph Summary**: The notebook prints statistics such as *"Graph has 120 nodes (Drugs: 30, Diseases: 50, Genes: 40) and 250 edges."*
- **Extracted Triples**: It shows sample rows from preprocessing, e.g., ('DrugA', 'treats', 'DiseaseX').
- **Inferred Relations**: Example output from the rule module, such as **"Inferred: DrugA may treat DiseaseY (shared gene: GeneZ)"**.
- **Probabilistic Inference**: Sample Bayesian query result, e.g., P(DrugA_effective | GeneZ=1) = 0.85.
- **Use Cases**: This project can be used to explore drug repurposing by identifying candidate drugs for diseases with similar genetic markers. It also serves as an educational example of combining knowledge graphs, logical inference, and probabilistic reasoning in bioinformatics. Researchers or students can adapt the notebook to their own datasets to discover new hypothesized drug–disease–gene relationships.

Running the Notebook

To run the notebook:

1. Ensure the Python environment is set up and all dependencies are installed.
2. Place the three input files (kg.csv, disease_features.tab, drug_features.tab) in the project directory.
3. Open a terminal, activate the environment, and launch Jupyter Notebook (jupyter notebook).
4. In the Jupyter interface, open the project notebook (e.g., knowledge_graph_inference.ipynb).
5. Run the cells in order (or run entire notebook). The notebook is organized by section (Preprocessing, Graph Construction, etc.), so each block can be executed sequentially. The outputs and figures will appear in-line as you run each cell.

Dependencies

This project relies on the following Python libraries (which are listed in requirements.txt):

- **pandas** (for data handling)
- **numpy** (numerical operations)
- **networkx** (graph construction and manipulation)
- **pgmpy** (probabilistic graphical models for Bayesian inference)
- **matplotlib** (plotting graphs and results)
- **tqdm** (progress bars during processing)
- **jupyter** (notebook interface)

Author
Mohammad Arsalan