VishGraphs and CoreRec Manual

VishGraphs and CoreRec Manual

link to published medium story : Dare you to click

Welcome to VishGraphs - your go-to Python library for making random graphs.

Introduction

VishGraphs is a versatile Python library designed to simplify graph visualization and analysis tasks. Whether you're a data scientist, researcher, or hobbyist, VishGraphs provides intuitive tools to generate, visualize, and analyze graphs with ease.

Features

- Generate random graphs with customizable parameters.
- Visualize graphs in both 2D and 3D.
- Analyze graph properties such as connectivity and centrality.
- Export graphs to various formats for further analysis or presentation.

Installation

You can install VishGraphs via pip:

pip install vishgraphs

VishGraphs Manual

Table of Contents

Introduction

VishGraphs is a Python library for graph visualization and analysis. It provides tools for generating random graphs, drawing graphs in 2D and 3D, and analyzing graph properties.

Installation

To install VishGraphs, you can use pip:

```
pip install vishgraphs
```

Usage

Generating Random Graphs

To generate a random graph, you can use the `generate_random_graph` function:

```
import vishgraphs

graph_file = vishgraphs.generate_random_graph(10, "random_graph.csv")
```

The use cases are:-

Graph Analysis and Recommendation System

Welcome to the Graph Analysis and Recommendation System! This repository contains Python code for analyzing and visualizing graph data, as well as recommending similar nodes within a graph. Let's explore the main functionalities of this codebase:

vishgraphs

```
import vishgraphs as vg
```

1. `generate_random_graph(num_people, file_path="graph_dataset.csv", seed=None)`

This function generates a random graph with a specified number of people and saves the adjacency matrix to a CSV file.

Use case: Generating synthetic graph data for testing algorithms or simulations.

2. `draw_graph(adj_matrix, nodes, top_nodes)`

Draws a 2D visualization of a graph based on its adjacency matrix, highlighting top nodes if specified.

Use case: Visualizing relationships within a graph dataset.

3. `find_top_nodes(matrix, num_nodes=10)`

Identifies the top nodes with the greatest number of strong correlations in a graph.

Use case: Identifying influential or highly connected nodes in a network.

4. `draw_graph_3d(adj_matrix, nodes, top_nodes)`

Creates a 3D visualization of a graph based on its adjacency matrix, with optional highlighting of top nodes.

Use case: Visualizing complex network structures in a three-dimensional space.

5. `show_bipartite_relationship_with_cosine(adj_matrix)`

Visualizes bipartite relationships in a graph using cosine similarity and community detection algorithms.

Use case: Analyzing relationships between different sets of nodes in a bipartite graph.

6. `bipartite_matrix_maker(csv_path)`

Reads a CSV file containing a bipartite adjacency matrix and returns it as a list.

Use case: Preparing data for analyzing bipartite networks.

core_rec

```
import core_rec as cs
```

1. `recommend_similar_nodes(adj_matrix, node)`

Recommends similar nodes based on cosine similarity scores calculated from the adjacency matrix.

Use case: Providing recommendations for nodes based on their similarity within a graph.

2. `GraphTransformer(num_layers, d_model, num_heads, d_feedforward, input_dim)`

Defines a Transformer model for graph data with customizable parameters.

Use case: Training machine learning models for graph-related tasks, such as node classification or link prediction.

3. `GraphDataset(adj_matrix)`

Defines a PyTorch dataset for graph data, allowing easy integration with DataLoader for model training.

Use case: Preparing graph data for training machine learning models.

4. `train_model(model, data_loader, criterion, optimizer, num_epochs)`

Trains a given model using the provided data loader, loss function, optimizer, and number of epochs.

Use case: Training machine learning models for graph-related tasks using graph data.

In the `test.py` file, various functionalities from `vishgraphs.py` and `core_rec.py` are utilized and demonstrated:

- Random graph generation (`generate_random_graph`).
- Identification of top nodes in a graph (`find_top_nodes`).
- Training a Transformer model for graph data (`GraphTransformer`, `GraphDataset`, `train_model`).
- Recommending similar nodes using a trained model (`recommend_similar_nodes`).
- Visualization of a graph in 3D (`draw_graph_3d`).

Feel free to explore the codebase and utilize these functionalities for your graph analysis and recommendation tasks! If you have any questions or need further assistance, don't hesitate to reach out. Happy graph analyzing! 📊 🔍

Drawing Graphs

VishGraphs supports drawing graphs in both 2D and 3D:

```
adj_matrix = vishgraphs.bipartite_matrix_maker(graph_file)
nodes = list(range(len(adj_matrix)))
top_nodes = [0, 1, 2]  # Example top nodes
vishgraphs.draw_graph(adj_matrix, nodes, top_nodes)
```

Examples

Example 1: Generating and Drawing a Random Graph

```
import vishgraphs

graph_file = vishgraphs.generate_random_graph(10, "random_graph.csv")
adj_matrix = vishgraphs.bipartite_matrix_maker(graph_file)
nodes = list(range(len(adj_matrix)))
top_nodes = [0, 1, 2]  # Example top nodes
vishgraphs.draw_graph(adj_matrix, nodes, top_nodes)
```

Troubleshooting

If you encounter any issues while using VishGraphs, please check the documentation or open an issue on the GitHub repository.

Contributing

Contributions to VishGraphs are welcome! You can contribute by reporting bugs, submitting feature requests, or creating pull requests.

License

VishGraphs is distributed following thought.

```
The library and utilities are only for research purpose please do not
use it commercially without the authors(@Vishesh9131) consent.
```