VishGraphs and CoreRec Manual

link to published medium story : Dare you to click

Welcome to CoreRec & VishGraphs - your go-to Python library for Training Recomendation models & 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

Feature Summary

core_rec.py

- `GraphTransformer(num_layers, d_model, num_heads, d_feedforward, input_dim)`
  - Defines a Transformer model for graph data with configurable parameters.
- `GraphDataset(adj_matrix)`
  - Creates a PyTorch dataset for graph data to facilitate model training.
- `train_model(model, data_loader, criterion=False, optimizer=False, num_epochs=False)`
  - Trains a given model using the provided data loader and parameters.
- `predict(model, graph, node_index, top_k=5)`
  - Predicts similar nodes based on a trained model and graph data.
- `aaj_accuracy(graph, node_index, recommended_indices)`
  - Calculates accuracy metrics for recommended nodes based on graph data.

vish_graphs.py

- `generate_large_random_graph(num_people, file_path="large_random_graph.csv", seed=None)`

    - Generates a large random graph and saves it to a CSV file.

- `draw_graph(adj_matrix, top_nodes=None, recommended_nodes=None, node_labels=None, transparent_labeled=True, edge_weights=None)`
    - Draws a 2D visualization of a graph with optional features.

- `draw_graph_3d(adj_matrix, top_nodes=None, recommended_nodes=None, node_labels=None, transparent_labeled=True, edge_weights=None)`
    - Creates a 3D visualization of a graph with customizable properties.

- `show_bipartite_relationship(adj_matrix)`
    - Visualizes bipartite relationships in a graph.

These libraries provide essential functionalities for graph analysis, visualization, and machine learning tasks.

Installation

For Security reasons, we are not providing the pip install command for now.

just copy the files "`vish_graphs.py`" ,"`core_rec.py`" and "`common_imports.py`" to your project folder and import them in your project.

Or;

For Windows Users

For Windows Users You can set the python path by `set` command or just copy paste this in cmd prompt `set PATH "%PATH%;C:\path\to\your\CoreRecRepo`

For MAC OS Users

For Windows Users You can set the python path by `nano ~/.zshrc` command or just copy paste this `export PYTHONPATH=$PYTHONPATH:\path\to\your\CoreRecRepo`

CoreRec Manual

Table of Contents

Introduction

- CoreRec :
  CoreRec is a recommendation engine designed for analyzing and visualizing graph data. It offers functionalities for recommending similar nodes based on graph structures, training machine learning models for graph-related tasks, and visualizing complex network structures.
- VishGraphs :
  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.

Directory Structure

To install VishGraphs, you can use pip: ## Directory Structure

Directory/File	Files	Description
CoreRec	REPO_UTIL/	Contains CoreRec related files and modules
	coreRec.svg	
SANDBOX/	usecases.md	Contains sandbox files for testing and experimentation
	vish_graphs.py	
USECASES/	vish_graphs.py	Contains specific use case files
	weightG.py	
UPDATES/		Directory for update-related files
BACKUP/		Directory for backup files
vish_graphs/	vish_graphs.py	Directory for VishGraphs related files and modules
ROADMAP/		Directory for roadmap and future updates

Usage

Generating Random Graphs

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

```
import vish_graphs as vg

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

The use cases are:-

🔍 Delve into Advanced Graph Analysis and Recommendation with VishGraphs and CoreRec! 🚀

Welcome to a world of cutting-edge graph analysis and recommendation tools brought to you by VishGraphs and CoreRec. Uncover the potential of data visualization and machine learning in a sophisticated manner.

🔗 Explore Detailed UseCases Here 🔗

CoreRec

```
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`).

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.

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)
```

Troubleshooting

Troubleshooting Guide

For issues with CoreRec and VishGraphs:

1. **Check Documentation:** Ensure you're following the library's guidelines and examples correctly.
2. **GitHub Issues:** Report bugs or seek help by creating an issue on the GitHub repository.
3. **Verify Data:** Confirm that your input data is correctly formatted and compatible.
4. **Model Parameters:** Double-check model configurations and training parameters.
5. **Visualization Inputs:** Ensure correct parameters for graph visualization functions.
6. **Community Help:** Utilize community forums for additional support.

This streamlined approach should help resolve common issues efficiently.

Contributing

We welcome contributions to enhance the functionalities of our graph analysis and recommendation tools. If you're interested in contributing, here are a few ways you can help:

- **Bug Fixes:** Identify and fix bugs in the existing code.
- **Feature Enhancements:** Suggest and implement improvements to current features.
- **New Features:** Propose and develop new features that could benefit users of the libraries.
- **Documentation:** Help improve the documentation to make the libraries more user-friendly.

To contribute, please follow these steps:

1. Fork the repository.
2. Create a new branch for your feature or fix.
3. Develop your changes while adhering to the coding standards and guidelines.
4. Submit a pull request with a clear description of the changes and any relevant issue numbers.

Your contributions are greatly appreciated and will help make these tools more effective and accessible to everyone!

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.
```