CSC591GraphP4

This project implementation is done towards fulfillment of Project 4 Anomaly Detection in Time Detection of CSC 591 Graph Data Mining.

Research Paper

The paper implemented is given under the research_paper folder. The paper is DELTACON: A Principled Massive-Graph Similarity Function

Goal

To implement the given community detection algorithm for real world graphs. Objective is stated here. To do that, you will need to:

- Read and understand the scientific publication assigned to you. If your publication has other emphases besides anomalous time point detection, then your implementation should focus only on the anomalous time point detection part.
- Implement the algorithm described in the publication using either R or Python. Your code must contain detailed comments and a README file that specifies any software that needs to be installed (using python's pip or or R's install.packages). Your main file should be called anomaly.py or anomaly.R and should take one argument, the data directory, as input. Make sure your code includes detailed comments.
- Have your code output a time series of your algorithms output to the file time_series.txt . If your paper uses a similarity score then output the similarity value time series. If it uses a distance metric then output the distance value time series. Let each line represent a single time step (i.e., one number per line).
- Finally, generate a plot of the above time series. This does not have to be performed in the program and can simply be created manually from the results file. On the plot, indicate the threshold value (for determining an anomaly) with a horizontal line.

Datasets

A collection of time evolving graph data has been provided to you. This data was primarily taken from the Stanford Large Network Dataset Collection. There are four time evolving graph from different domains. For example, the `enron_by_time` graph represents an email network and the `p2p-Gnutella` graph represent a peer to peer network. Each time evolving graph is represented as a series of text files which define the graph at a given time point. Each text file is given as an edge list but with the first line stating the number of nodes and edges. The 4 datasets are-

- `autonomous`
- `enron_by_day`
- `voices`
- `p2p-Gnutella`

Getting Started

Installation

- Install Python3 from here and finish the required setup in the executable file.

Install pip package manager for future downloads-

```
$ python -m ensurepip --upgrade
```

-

Upgrade the version of pip-

```
$ python -m pip install --upgrade pip
```

-

Install SciPy for scientific computing or follow instructions from here-

```
$ python -m pip install scipy
```

-

Install matplotlib for plotting data or follow instructions from here-

```
$ python -m pip install -U matplotlib
```

-

Install NumPy for plotting data or follow instructions from here-

```
$ pip install numpy
```

-

Create working directory named `Anomaly_detection_P4` and go inside it

```
$ mkdir Anomaly_detection_P4
$ cd Anomaly_detection_P4
```

-

Clone this repository from here or use the following in GitBash

```
$ git clone https://github.com/tusharkini/CSC591_Anomaly_detection
```

-

Running the Algorithm Code

Run the algorithm code in `anomaly.py` using-

```
$ cd code
$ python anomaly.py <name_of_dataset>
```

For example to run algorithm on `enron_by_day` use the following code-

```
$ python anomaly.py enron_by_day
```

- This will create an output file named `results/enron_by_day_time_series.txt` file containing all the similarity values, 1 in each line.
  It will also create a plot named `results/enron_by_day_time_series.png` for all these similarity values and the lower and upper threshold.

Note on running time

Depending on the size of the dataset and the size of each graph screenshot in the dataset, the program may take quite some time to execute. All graphs except the `p2p-Gnutella` and the `autonomous` graph take a very small time (about 10 seconds) to execute. The `autonomous` graph takes about 5 minutes to run and the `p2p-Gnutella` takes the highest running time(about 60 minutes).

Authors

- Tushar Kini Github
