**HW2 – Community detection**

**Submission Deadline:** 13.12.2020, 23:00

For the first part of the HW, you need to have the [python package](https://github.com/taynaud/python-louvain) that implements the [louvain method](https://en.wikipedia.org/wiki/Louvain_method) installed.

**Submission guidelines:**

You are required to submit your solution as a zip file:

* Python file with your functions implemented, the file’s name should be ID.py where ID is the student’s ID.
* A PDF file with answers to open questions (these are marked in a blue font).
* The zip file’s name should be ID.zip where ID is the student’s ID. For example, the student Moshe Moshe with an ID of 1234567, should submit a zip file “1234567.zip”, containing his implemented solution “1234567.py” and a pdf file.
  + Do not zip the directory where your solution is stored, only zip the required files!
* You are required to implement a function called ‘get\_name’ that returns your full name in English.
* You are required to implement a function called ‘get\_id’ that returns your ID number.
* You have to follow **the exact API described in the HW** (exact function names, parameters and returned types). Please avoid typos.
* Your code **should not contain any part of loading data**. You may include a ‘main part’ code block (using the if \_\_name\_\_ == "\_\_main\_\_": syntax). Within this code block you can load data and test your implementation.

Objectives:

* Get familiar with a number of community detection algorithms.
* Learn to work and analyse Twitter data
* Apply an end-to-end community detection solution with real data.
* Learn how to analyze results of a community detection algorithm in a quantitative and qualitative way.

1. **Community detection wrapper**
2. Use the networkX package and implement community\_detector - a function wrapping different community detection algorithms. The function should run a specific community detection algorithm (e.g., clique\_percolation) with the aim of optimizing the modularity value:

|  |  |  |  |
| --- | --- | --- | --- |
| **Parameter name** | **Parameter type** | **Explanation** | **Default value** |
| ‘algorithm\_name’ | str | Name of the algorithm to run. Can be either ‘girvin\_newman’, ‘louvain’ or ‘clique\_percolation’ |  |
| ‘network’ | networkX object | The network to run the detection over |  |
| ‘most\_valualble\_edge’ | function (or None) | A parameter that is used only by the ‘girvin\_newman’ algorithm (see further explanation about the parameter [here](https://networkx.org/documentation/stable/_modules/networkx/algorithms/community/centrality.html#girvan_newman)) | None |

The function returns **a dictionary** with the following key/values:

|  |  |  |
| --- | --- | --- |
| **Key** | **Type** | **Explanation** |
| ‘num\_partitions’ | int | Number of partitions the network was divided to |
| ‘modularity’ | float | The modularity value of the partition. In the case of the ‘clique\_percolation’ model - think about a simple way to calculate the modularity (it is not simple as the other algorithms since some nodes share different communities). |
| ‘partition’ | List of lists | The partition of the network. Each element in the list is a community detected (with node names). |

1. Take a look at the [documentation of the ‘girvin\_newman’ algorithm](https://networkx.org/documentation/stable/_modules/networkx/algorithms/community/centrality.html#girvan_newman). One of the options in this function, is to define a special parameter function as input (‘most\_valualble\_edge’).

You are required to write your own ‘most\_valualble\_edge’ function to be used by the ‘community\_detector’ function. The function name should be ‘edge\_selector\_optimizer’

1. Use sections (i) + (ii) and run your function over the ‘Les Miserables’ network, which is a network of interactions between central characters of the [famous novel book by Victor Hugo](https://en.wikipedia.org/wiki/Les_Mis%C3%A9rables). The network can be found in the ‘networkX’ package. Note that the edges of this network are weighted.

Report the output of your function over the three algorithm options. Which setting yields the best modularity value? Can you identify any issue with one of the partitions achieved by any of the algorithms? Did your ‘edge\_selector\_optimizer’ achieve the best ‘modularity’ value? If not - try to improve it or explain why it did not achieve any improvement.

1. **Community detection - Twitter data**

In this part of the assignment, you will analyse Hebrew Twitter data. Specifically, you will analyse Hebrew political tweets. You will explore how well the various community detection algorithms capture the political community structures that correspond to different political agendas.

[Here](https://drive.google.com/file/d/1fMxHOD61IuIDqx9gIct6ErqJW0EXeIdQ/view?usp=sharing)  you can find 90 .txt files containing Twitter data. Each .txt file contains a sample of the Hebrew tweets posted on a particular day.

Before approaching this task, read a bit about [Tweets json format](https://developer.twitter.com/en/docs/twitter-api/v1/data-dictionary/overview/intro-to-tweet-json). Note that each user has a distinct user\_id (that will be used in this exercise as a unique identifier for each node).

In this question, we will use information about [**retweets**](https://en.wiktionary.org/wiki/retweet)**.** As an example, you can see below a financial news tweet that was retweeted by 1.8K users (so far):



Viewing each of the users as a node, if user *u* retweets a tweet *t,* originally posted by user *v*, we consider it a directed edge from *u* to *v*. The edge weight is determined by the number of tweets by *v* that *u* has retweeted.

* 1. Implement a function (called “construct\_heb\_edges”) that will be used to process the zipped files, building an edge dictionary based on the retweeting relations.

The function parameters are:

|  |  |  |  |
| --- | --- | --- | --- |
| **Parameter name** | **Parameter type** | **Explanation** | **Default value** |
| ‘files\_path’ | str | Location of **all** files the function requires. These are the 90 txt files and the *central\_political\_players.csv* file (see further explanation below) |  |
| ‘start\_date’ | str | First day to include (format: YYYY-MM-DD) | '2019-03-15' |
| ‘end\_date’ | str | Last day to include (format: YYYY-MM-DD) | '2019-04-15' |
| ‘non\_parliamentarians\_nodes’ | int | See section (ii) | 0 |

The function should return an edge dictionary. The dictionary keys are tuples of size two, each with two user\_id values (e.g., (USER\_X\_ID, USER\_Y\_ID)). The value of each key is a **retweets counter** (e.g., number of times USER\_X\_ID retweeted USER\_Y\_ID). **Note that the tuple’s order is important!**

We are mainly interested in the central players that are active in the political Twitosphere. Hence, your analysis should focus on these. Your “construct\_heb\_edges” should only contain the central political players. The rest of the nodes (as well as their edges) should be removed. The details of those members are provided in [this csv file](https://drive.google.com/file/d/1f2vi-wj-kjpUNDu8S7k2Vz5PWPtktovr/view?usp=sharing) (you can see their tweeter\_id in the first column).

* 1. Removing most of the nodes and considering only the central political members (as we did in section (i)) is an aggressive manipulation.

The last parameter of the function (‘non\_parliamentarians\_nodes’ int, defaulted to zero) controls the number of nodes **that are not deleted** from the original network (on top of the central political players listed in [this csv file](https://drive.google.com/file/d/1f2vi-wj-kjpUNDu8S7k2Vz5PWPtktovr/view?usp=sharing)). You need to think about a smart (and an efficient) method to choose these extra node, so their inclusion will provide an added value in analyzing the network.

* 1. Implement a function (called “construct\_heb\_network”) that takes as input a single parameter called “edge dictionary” and returns a networkX object (consisting of the input nodes and edges). The networkX object should be a directed and weighted network. Naturally, the input value of this function is the dictionary returned by the “construct\_heb\_edges” function.
  2. Apply the Girvin\_Newman over the network you built in (iii) (over the two options described in sections (i) and (ii) and focus on the modularity of each setting. You can use your implementation from Q1.

Compare the communities detected at the two networks. Do you see any striking difference? Do the differences/similarities make sense? Can you identify Right/Left/Center blocks? Can you identify political players that have the potential to “desert” to the other side of the political map?

Has allowing the extra nodes, not listed in the [predefined list](https://drive.google.com/file/d/1f2vi-wj-kjpUNDu8S7k2Vz5PWPtktovr/view?usp=sharing), proved useful in finding an optimal (better?) partition? Why?

**Notes:**

Please make sure you install the required packages using the latest version.

If you use any package that is not mentioned in the list below, please include a “requirements.txt” file, stating which packages you used for your solution.

List of packages:

* Pandas
* NetworkX
* Numpy
* Scipy
* tqdm
* Pickle
* python-louvain