MACHINE LEARNING IN NETWORK SCIENCE
CENTRALESUPÉLEC

Lab 6: Propagation on Graphs & Influence Maximization

Instructor: Fragkiskos Malliaros
TA: Alexandre Duval

March 24, 2023

Description

The purpose of the lab is to introduce the basic models used to simulate epidemics over networks, to conduct experiments with them and to compare various models. We will examine how to detect influential nodes by combining those models with node centrality. Afterwards, we will work on influence maximization problem in social networks and we will implement the *greedy algorithm* in order to identify a set of nodes that maximizes the spread of influence. In our experiments, we are going to work on the largest connected component of the `NetScience` co-authorship network[1] and use the *NDlib*[2][3] package to simulate the diffusion models.

Part I: Susceptible-Infected-Recovered (SIR) Model

The typical models utilized for diffusion simulation in the field of epidemics are the *Susceptible-Infected-Recovered (SIR)* and *Susceptible-Infected-Susceptible (SIS)* models. Both models assume a population of N individuals, divided into the following three states. Susceptible (S): the individual has not yet infected, thus being susceptible to the epidemic; Infected (I): the individual has been infected with the disease and is capable of spreading the disease to the susceptible population; Recovered (R): after an individual has experienced the infectious period, it is considered immune to the disease and it is not able to be infected again or to transmit the disease to others. Every node that is on the state I can infect its susceptible neighbors with probability β (called infection rate) and afterwards it can recover with probability γ (called recovery rate). The difference between SIS and SIR, as you can infer, is that a node in SIS can be infected again. This can be the case in many real world applications of epidemiology, such as political campaigns or rumour spreading and reasonably induces important changes in the behavior of the model. Here we will specifically focus on the *Susceptible-Infected-Revovered (SIR)* model. The following state diagram represents the behavior of each node of the network:

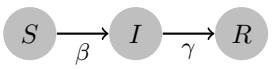

Figure 1: State diagram of the SIR model

[1] http://vlado.fmf.uni-lj.si/pub/networks/data/collab/netscience.htm
[2] https://ndlib.readthedocs.io/
[3] If you encounter any difficulties with the installation of the package, please look at the web page `https://ndlib.readthedocs.io/en/latest/installing.html` and let us know.

Initially, all the nodes of the network are set at the susceptible state S, except the one whose spreading properties are to be studied, which is set at the infected state I. Then, we let the process evolve as described above. Note that at a given time step a node can only recover after he was given a chance to infect its neighbors.

Exercise 1: Simulation of the SIR Model

In the first experiment, we will simulate the *SIR Model* by randomly choosing an initially infected set of nodes. You can define your model by setting the parameters of the `SIR(graph, beta, gamma, seed_set)` function, which was implemented for you by using various functionalities of the `NDlib`[4] package.

1. Simulate *SIR Model* by setting the parameter γ=0.1 and the probability $\beta = 1/\lambda_1$ for randomly sampled 100 seed nodes where λ_1 is the largest eigenvalue of the adjacency matrix of the network.

   Note that you can use the `eigh(a)` method of the *SciPy*[5] library to find the eigenvalues of the adjacency matrix. Keep in mind that you have to transform the sparse matrix to an array before feeding it to `eig()` since `adjacency_matrix(G, nodelist=None, weight='weight')` method of the *NetworkX* package returns a sparse matrix.

   To see how many nodes were infected or removed at the end of the each iteration, we can use the following built-in method of `NDlib`:

   ```
   SIRModel.iteration_bunch(bunch_size)
   ```

   It returns a list, let's call it as `sir_iterations`, which contains a dictionary for each step. For instance, its *node_count* attribute at the last time step, `sir_iterations[-1]['node_count']`, gives a dictionary where the values corresponding to the keys 0, 1 and 2 indicate the number of *susceptible*, *infected* and *recovered* nodes, respectively.

2. In order to plot the progression of the simulation, we use the built-in methods of the `NDlib` library. Initially, `build_trends` function is used with the result of the simulation and then the `DiffusionTrend`. Finally, the `show` function generates an interactive plot signifying the course of the number of *infected*, *susceptible* and *recovered* nodes. What do you observe?

3. You can also draw the nodes with different colors with respect to their status, using the function `visualize_status(graph, iterations, t=5, node_size=20)` as it is illustrated in Figure 2.

Part II: The Independent Cascade Models

Exercise 2: Independent Cascade Model

The independent cascade model is probably the most used model in computer science, as most algorithmic works are based on it. This model assumes that a node v has only one chance to influence each of its neighbors u based on the probability $p_{v,u}$. Moreover, a node v can infect its neighbor u depending only on a probability $p_{v,u}$ as shown in the figure below.

In this model, we will use the `independent_cascade(graph, threshold, seed_set)` for specifying the model, and the threshold parameter denotes the probability of a node influencing its neighbor.

[4] `https://ndlib.readthedocs.io/en/latest/installing.html`
[5] https://docs.scipy.org/doc/

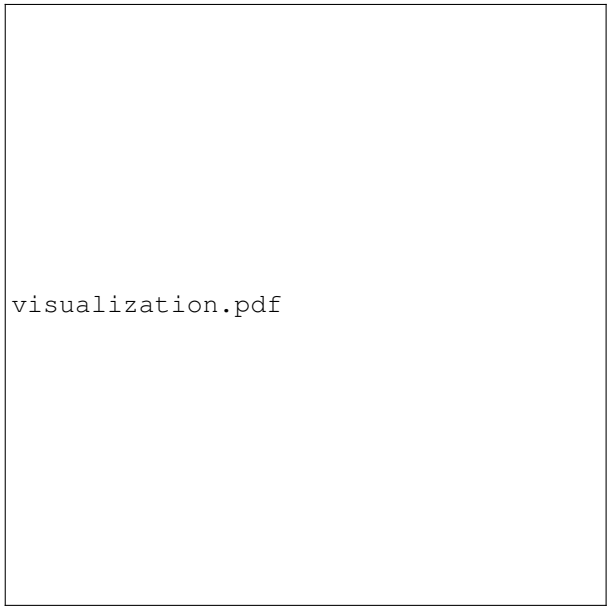

Figure 2: Visualization of the status of nodes.

Figure 3: Illustration of the independent cascade model.

1. Simulate the experiment for the *threshold* or *influencing probability* equal to 0.5.
2. Plot the progression of the simulation. What do you observe?
3. What happens if you increase or decrease the threshold value?

Exercise 3: Model Comparison (Optional)

In this exercise, we will perform various experiments to compare the models described in the previous parts.

1. As a final step, plot the number of infected nodes retrieved in each iteration of all three models in different colors. To do this you will have to extract the number of infected and recovered (if the model has recovered nodes) in each iteration, sum them up, and store it in a list. Then, you can plot the values with the `plot()` method.
2. Which model provides the most infected nodes?

Part III: Detection of Influential Spreaders

In the last part, we will evaluate the effect of initially chosen set of infected nodes for different diffusion models. In the end, fixing the epidemic model, we will be able to examine which nodes are more influential in the network.

Exercise 4: The Effect of Seed Sets

By now, we are familiar with certain centrality measures that are correlated with influence and they can also be used to identify influential node. One way to evaluate the effectiveness of the chosen initial inflected nodes or the seed set is to start diffusion simulations from those nodes and to measure how

many nodes will get infected during these processes. In this exercise, we will apply three centrality criteria to evaluate their effectiveness in the task of information propagation: i) *Degree Centrality*, ii) *Pagerank*, iii) *k-core* and iv) *neighborhood coreness*. You can utilize the built-in functions of *NetworkX* to compute the centrality values of nodes.

1. Simulate the models described in the previous parts for the seed sets of size 100 using different centrality criteria.
2. Which measure of centrality leads to the largest number of infected nodes? Can you hypothesize why?
3. Examine the total number of infected nodes for seed sets of size $10, 20, 30, 40, 50$. Plot the results.

Part IV: Influence Maximization

The problem of *influence maximization* addresses how to find an optimum set of nodes in a given network such that, if a diffusion cascade starts from this set, the number of infected nodes when the diffusion is over would be maximized.

To understand why we may need an algorithm instead of using simple metrics like the ones introduced in the previous part, we can consider the simple network in Figure 4 for an illustration. In this graph, nodes w, u and v have the same outdegree number, so they have the same number of followers, but u and v posses more central positions in the network – so their followers can affect more nodes than the ones of w. Moreover, if we initialize a diffusion from w and u, the infected nodes induced by each one will highly overlap because of the common followers – so their influence might not be as effective as in the previous case.

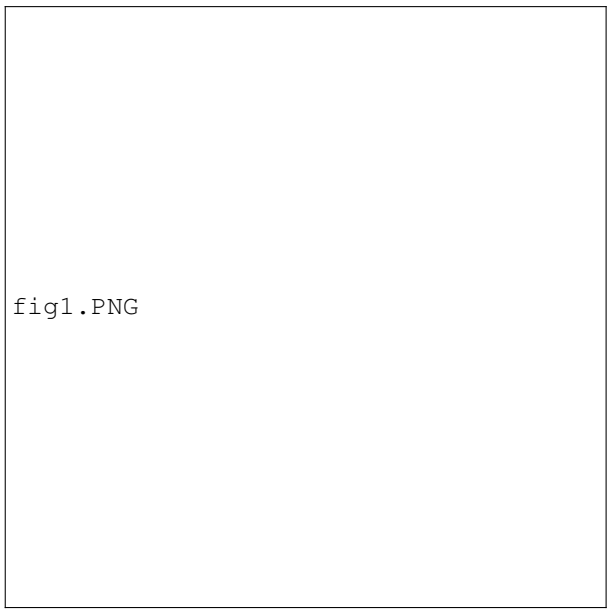

Figure 4: Example of seed set for influence maximization.

More formally, the *influence* of a given set of nodes S, denoted by $\sigma(S)$, is defined to be the expected number of infected nodes at the end of the diffusion process when S is chosen as initial active nodes. Therefore, the *influence maximization* problem aims at finding a set of k nodes having maximum influ-

ence for a given value k. However, the optimal solution is NP-hard for various diffusion models and the work [3] proves that the problem of selecting the most influential nodes is also NP-hard for the *Independent Cascade* and *Linear Threshold* models.

Fortunately, the papers [1, 2] demonstrate that the optimal solution can be approximated by using the greedy Algorithm 1 within a factor of $(1 - 1/e)$. The idea behind the algorithm is that we begin with an empty set, and a node providing the maximum marginal gain is selected at each step until we obtain a set of k nodes.

Algorithm 1 The Greedy Algorithm

1: **Input:** A graph $G = (V, E)$ and a parameter k
2: **Output:** A set of nodes S
3: Initialize $S \leftarrow \{\}$
4: **for** i=1 **to** k **do**
5: $v = \arg\max_{u \in V \setminus S} \sigma(S \cup \{u\}) - \sigma(S)$
6: $S \leftarrow S \cup \{v\}$
7: **end for**

Note that, we have to compute the influence function $\sigma(A)$ at each step and it can be rewritten as follows:

$$\sigma(A) = \sum_{c \in \mathcal{C}} Prob[c] \cdot \sigma_c(A) \tag{1}$$

where $c \in \mathcal{C}$ is a mapping defined on the edge set E such that $c((u, v))$ indicates whether the node u activates v with probability p_{uv} during the diffusion process for $(u, v) \in E$, so the set $\mathcal{C}$ consists of all possible mappings and $\sigma_c(A)$ denotes the total number of infected nodes under the realization $c \in \mathcal{C}$ when A is initially chosen as the *seed* set. However, exactly computing the influence function at every step is computationally infeasible so we can repeat the simulation of the model for different realizations only limited times and consider the average of the marginal gains of adding a candidate node u to the set A.

Exercise 5: Implementation of the Greedy Algorithm

In this exercise, we will implement Algorithm 1 for the *Independent Cascade (IC)* model and perform various experiments working with the largest connected component of the network.

1. Implement Algorithm 1 by completing the following function:

```
def greedy_algorithm(graph, k, repetition, ic_threshold, ic_num_steps):
    '''
    :param graph: given graph
    :param k: seed set size
    :param repetition: the number of times to perform the simulation
    :param ic_threshold: threshold value for independent cascade model
    :param ic_num_steps: number of steps for independent cascade model
    :return seed_set: a list containing k-nodes
    '''
```

```
    ...
    ...
    ...


    return seed_set
```

The function `simulate_ic_model` has already been implemented – it simulates the *IC* model on the given graph and returns the number of infected nodes at the end of the process. Thus, you can use it in order to run the *IC* model and get the number of infected nodes at the last step of the simulation.

Note that, evaluating the influence function $\sigma(A)$ in Equation 1 requires high computational cost so we can estimate it by running the model for `repetition` number of times. In other words, it can be estimated as follows:

$$\widehat{\sigma}(A) = \frac{1}{N} \sum_{i=1}^{N} \sigma_{c_i}(A) \tag{2}$$

where c_i's are realizations obtained by running the model N independent times. We can choose small `repetition` or N values in our experiments.

2. Find seed sets of size k, running the implementation of the greedy algorithm for different values of k (set the *threshold* parameter as 0.5).
3. Perform the same experiment for different *threshold* values. For instance, you can choose 0.1, 0.5 and 0.9. What do you observe? Is there any difference?

Exercise 6: Visualization and comparison

In this exercise, we will compare different seed sets, visualizing them with different node colors.

1. Complete the required part in the `visualize_seedsets` function in order to find the intersection between seed sets. We will use green color to indicate the nodes existing in both of the sets.
2. Similar to the experiments in the previous lab, use the *degree centrality* and *k-core* to extract seed sets of different sizes, and visualize them using the `visualize_seedsets` function.
3. Now, compute seed sets with the `greedy_algorithm`, and compare them with the ones obtained by using centrality measures by visualizing over the network.
4. For various values of seed set size k, compare the number of influenced nodes obtained by the greedy algorithm vs. and degree and k-core heuristics. What do you observe?

References

[1] Gerard Cornuejols, Marshall L. Fisher, and George L. Nemhauser. Location of bank accounts to optimize float: An analytic study of exact and approximate algorithms. *Management Science*, 23(8):789–810, 1977.

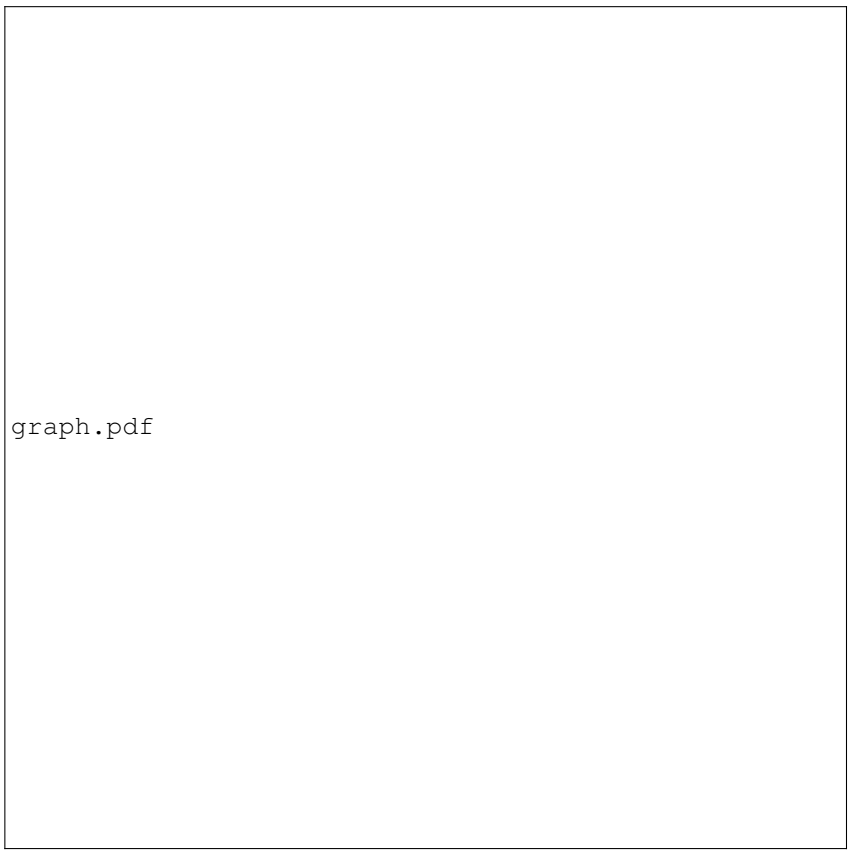

Figure 5: The seed sets of blue and red colors are respectively obtained by choosing 10 nodes having the highest degree and k-core centralities. The green color denotes the nodes existing in both of the seed sets.

[2] M.L. Fisher, Â G.L. Nemhauser, and L.A. Wolsey. An analysis of approximations for maximizing submodular set functions. CORE Discussion Papers RP 341, Université catholique de Louvain, Center for Operations Research and Econometrics (CORE), 1978.

[3] David Kempe, Jon Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '03, pages 137–146. ACM, 2003.