ADVANCED LEARNING FOR TEXT AND GRAPH DATA
MASTER DATA SCIENCE - MVA
Lab 8: Influence Analysis

Lecture: Prof. Michalis Vazirgiannis
Lab: George Panagopoulos, Giannis Nikolentzos and Maria Rossi

January 9, 2019

1 Description

The goal of this lab is to examine and experiment with several techniques related to influence analysis in complex networks. Influence is a directed measure defined between two users and represents how possible is for a user to adapt the behavior or copy the action of another user. We will first explore how influence is depicted in the structure of the network, by identifying influencers based on different structural characteristics, in a heuristic manner. Subsequently, we will proceed to introduce a diffusion model which can be used to simulate a spreading process that takes place over the network, and utilize it in evaluating our chosen influencers from the previous step. Then, we will start working with real information diffusion cascades and define new, more realistic measures of influence for evaluation. Finally, we will examine the algorithmic solution to the problem of influence maximization, which is choosing the optimum nodes to maximize the spread of information, and compare it with the aforementioned heuristic approaches in two different manners. You are given a python script (`lab_influence_maximization.py`) which you will fill throughout the lab.

2 Identifying Influential Nodes

As a first step, we will familiarize with basic influence metrics in the follower graph of the Digg dataset [1], which is the graph we will work with throughout this lab. This network comes from the news sharing website Digg, which used to have close to 3 million users that could share, vote or rate on news stories. The dataset, which was crawled in June 2009 [1], includes the directed follower network and vote logs which can be used to track who influenced whom by voting for similar news. Unfortunately, the whole dataset is too big to be handled in the scope of this lab. Thus we have created a subset that includes only the nodes with 100 or more followers, which roughly amounts to the top 1.5% of the network and outputs this graph:

```
Number of nodes: 3944
Number of edges: 565286
Average degree: 286
```

[1] https://www.isi.edu/ lerman/downloads/digg2009.html

1. Degree centrality is the simplest index to identify influencers: the more connections a node has (followers and following), the greater its influence. Load the given graph and compute the top 10 nodes based on the degree. To do this, sort the dictionary given with the `degree` function based on its values in descending order and take the first 20. Do the same thing for the in degree, which resembles the number of followers a node has.

2. The number of followers can be too limited, as it fails to capture the importance of the node's position in the network. To do this, we can use the famous pagerank algorithm [2], which measures a node's strength based on the strength of its neighbors. Calculate the top 20 nodes based on pagerank. Do this for both, directed and undirected versions of the network. The procedure is similar to the previous step using `pagerank` and `to_undirected`. Also, bare in mind that pagerank function returns a dictionary, unlike degree which returns a `DegreeView`, so they need different handling. In addition, pagerank might take more time to be computed (up to 1 minute).

3. The aforementioned metrics can assign strong values in a node that is well connected but resides in the periphery of the network. Such position though is not fit for an influencer, because the more central a node is, the more its opinion can spread. To overcome this, we can use another graph theoretic metric, the core number [7]. Given our undirected network G, C_k is defined as the k-core subgraph of G if it is a maximal connected subgraph in which all nodes have degree at least k. Then, each node $v \in V$ has a core number $c_{v_i} = k$, if it belongs to a k-core but not to a $(k+1)$-core. Figure 1 illustrates the k-core decomposition of a graph. It is clear that the cohesion of subgraphs increases as k increases. Let us denote as C the set of nodes with the maximum core number k_{max}. To compute the top 20 nodes in terms of the k-core they belong to, we will utilize

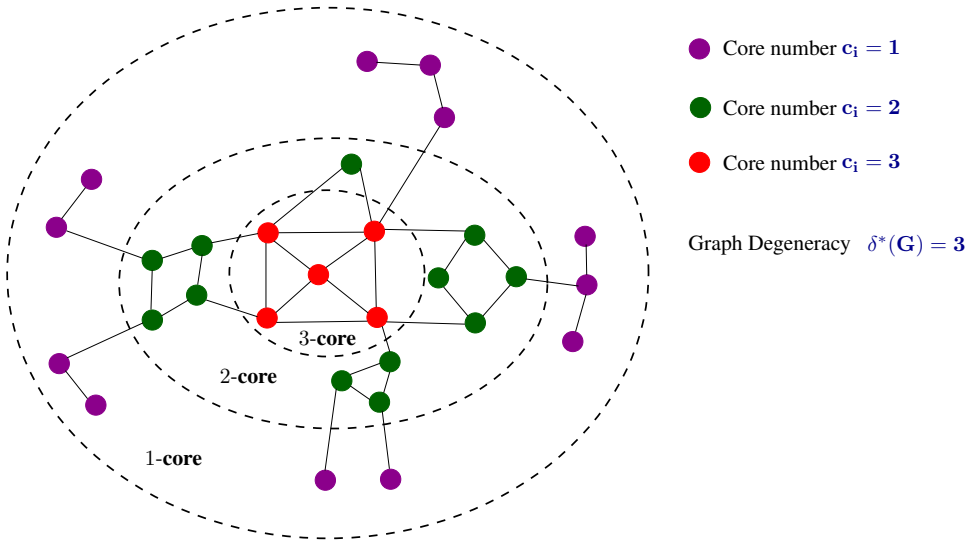

Figure 1: *k-core* decomposition illustration.

the function `core_number` in a similar way with the previous ones.

4. It is time to visualize the top nodes that we found. Unfortunately, trying to visualize a network with more then half a million edges will result in a hairball plot. Thus we first have to create a subgraph. To do this, we will construct a function called `clean_graph` which can remove nodes that have a degree of under a threshold.

[2] `https://en.wikipedia.org/wiki/PageRank`

```
def clean_graph(G, threshold):
    """
    Remove the nodes with degree less then threshold
    """
    to_plot = G.copy()
    #------ Put the nodes that have degree less then threshold to a variable "to_remove"

    #------ Remove them from the graph

    print("Removed "+str(len(to_remove))+" of "+str(len(G.nodes()))+" nodes")
    return to_plot
```

Once we have this function, we can threshold at $degree < 300$, which is close to the average degree, and plot it. Keep in mind that some of the top 20 nodes retrieved might have been removed fromt his process, so you have to remove them from the seed set before plotting them. Also, there should be some overlapping between different measures, so we do not expect to see 10 nodes for each set.

3 Evaluating Influence

Now that we have identified a set of influencers, we need a way to evaluate our choices. To this end, we will rely on two different approaches. The first one is based on epidemiology and relies on simulating spreading processes over the network, to identify how many people would be influenced if the process started from our chosen seeds. The second approach is more data-driven and relies on actual diffusion cascades that have taken place over the network.

3.1 Linear Threshold model

In this type of evaluation, the spreading capabilities of each node is calculated after starting a diffusion process from the node in question. The models that are most commonly used in order to simulate a spreading process are borrowed from the field of epidemics [2, 3]. Here we will specifically work with the *Linear Threshold (LT)* model, because it has been thoroughly studied and proved to exhibit some elegant theoretical properties which allow it to be used in influence maximization algorithms [8]. In this model, a node v is influenced when a sufficient number of its neighbors u are infected. From the perspective of computational sociology, the threshold of each node resembles how much cumulative influence needs to be exerted on an individual from its social surrounding in order to effect her behavior. Typically each node v chooses a threshold θ_v uniformly at random from the interval $[0, 1]$. It represents the weighted fraction of v's neighbors that must become active in order for v to become active: where $\mathcal{N}(v)$ is the set of neighbors of v. Given a random choice of thresholds and an initial set of active nodes (with all other nodes being inactive), the diffusion process unfolds in discrete steps as shown in figure 2.

Normally, each neighbor can exert different amount of influence to a node. Thus, the directed edges can be accompanied with a weight $b_{v,u}$ that captures how much each node influences another, in which case the criteria becomes:

$$\sum_{u \in \mathcal{N}(v) \cap I} b_{v,u} \geq \theta_v \tag{1}$$

Since assigning these weights is an open research problem and it calls for an of-the-shelf implementation of the model as well as more computational time, we will rely on the simple unweighted

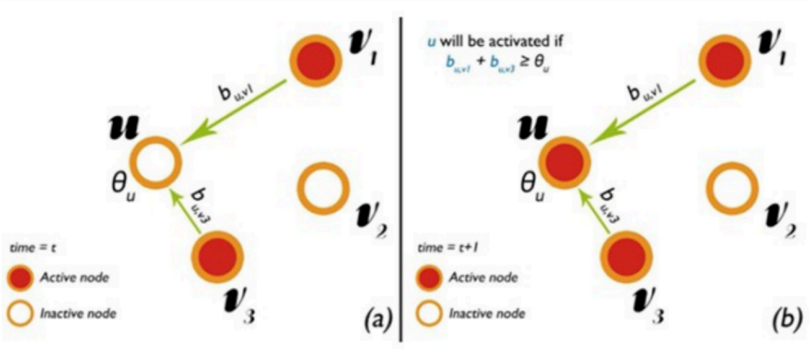

Figure 2: Linear Threshold model illustration.

approach for the lab. However, you are encouraged to perform the analysis with different types of weights, such as based on weighted cascade [8] or in activity history [10], and explore the different outcomes.

To do this we will utilize the NDlib[3] python library. We will define a function *simulate_spreading* that performs the simulation using the `ThresholdModel` and `ModelConfig` modules to define the Linear Threshold model.

```
def simulate_spreading(G, seed_set, num_steps=3, threshold = 0.01):
    """
    Given the graph and the seed set, compute the number of infected nodes
    after the end of a spreading process
    """
    th_model = th.ThresholdModel(G)
    config = mc.Configuration()
    #------ set thresholds

    #------ add initial infected nodes

    #------ run simulations and put the result in a variable called "result"
    return result[num_steps-1]["node_count"][1]
```

The functions needed to fill the blanks are *add_model_initial_configuration*, *set_initial_status* and *iteration_bunch*.We will add a threshold to each node equal to 0.01 using the function *add_node_configuration*. Remember this is a very dense network with a high average degree and the infection starts only with 20 nodes, thus we need the nodes to be effected by a small portion of their neighbors for the spreading to actually happen. We will let the simulation run for 3 steps, and each simulation will use a different seed set of the 5 we extracted in the previous section.

Once you run the experiments, what do you observe? Which metric prevails?

3.2 Diffusion Cascades

In this part, we will utilize logs of activity to evaluate which nodes are actually influenced by the seed sets we found. The initial logs consisted of votes of users to different posts. We used this to construct the diffusion cascades, which are a series of related events in time that happen throughout the network. A common example of a diffusion cascade is a tweet and its retweets. Note that as we subsetted the initial Digg network, these cascades are also subset such that they include only the nodes that are retained

[3] `https://ndlib.readthedocs.io/en/latest/installing.html`

in the network. Generally, the cascades can be used to trace how information flows throughout the network. This information provides a new dimension in influence estimation, while combining with its structural counterparts is an open and very popular problem.

For the purpose of our exercise, you have to load the cascades found in the file $digg\_votes\_subset.txt$ into a dictionary $node\_cascades$ where the key is the initiator of the cascade and the value is a list containing the cascades it initiated. Each cascade is simply a series of nodes. Here, for simplicity, we overlook the order of the nodes and the time they got "influenced", but you can experiment with them further using the initial Digg votes dataset. Once you have this dictionary, you will define the function `dni`, which stands for distinct nodes influenced. Keep in mind that some seeds might have not started any cascades.

```
def dni(node_cascades, seed_set):
    """
    Measure the number of distinct nodes in the cascades started by the given seed set
    """
    influence_spread = set()
    #------ retrieve the cascades of each seed

    #------ add the nodes in the cascade into influence spread
    return len(influence_spread)
```

This is a measure to evaluate how many nodes participated in diffusion cascades started by the nodes in our seed sets. Note that we do not really care about how many times a node participates in the cascades, as we focus on how many distinct people can be effected by the spreading of our seeds, which is why we are using a simple set. Run this evaluation in a similar manner you did with $simulate\_spreading$ and explain your observations. Do these results agree with the previous ones?

4 Influence Maximization

Influence Maximization (IM) is the problem that lies in the heart of influence analysis and addresses how to find a set of nodes, such that if they start a diffusion, the number of infected nodes in the network (influenced spread) will be maximized. It has a broad range of applications, from viral marketing, which was the initial motivation for the problem, to epidemiological containment and political campaign management. The problem can be formulated as follows: given a social network, a diffusion model with some parameters and a number *k*, find a seed set S $\subset$ V of size *k* such that the influence spread is maximized. In their seminal work, Kempe et al. [8] proved that *Influence Maximization* is NP-complete, under the *Linear Threshold (LT)* and the *Independent Cascade (IC)* diffusion models. Moreover, they proved that the function of the influence spread under both models is monotone non-decreasing and submodular. A monotone non-decreasing submodular function can be maximized by a greedy algorithm with a $(1-1/e)$ approximation ratio. Hence, they provide a greedy algorithm that can approximate an optimal solution with a theoretical guarantee.

What is a submodular function?
If V is a finite set, a submodular function is a function $f : 2^V \rightarrow \mathbb{R}$ which satisfies the following:

- $\forall S \subseteq T \subseteq V \backslash v$, we have that: $f(S \cup \{v\}) - f(S) \geq f(T \cup \{v\}) - f(T)$ (**diminishing returns**)

A function f is monotone non-decreasing if and only if:

- for all $S \subseteq T \subseteq V$, $f(S) \leq f(T)$

Algorithm 1 illustrates the main steps of the greedy algorithm.

Algorithm 1 The Greedy Algorithm

Input: A graph $G = (V, E)$ and parameter l
Output: A set of vertices S
Initialize $S = \emptyset$
for $i = 1$ to l **do**
  $v = \arg\max_{u \in V \setminus S} f(S \cup \{u\}) - f(S)$
  $S = S \cup \{v\}$
end for

Note that f is the function that calculates the influence spread of a node or a set of nodes. In this case, to calculate the influence spread we simulate a spreading process via the *LT* model starting from the nodes of interest. Keep in mind that since this is a stochastic model, we will have to take into account all possible realizations of the influence probabilities in a network's edges. This means that we have to estimate the spread of a set of nodes under all possible combinations of existing and non-existing edges, which has proven to be #P-hard [9]. Thus, we resort in sampling randomly possible realizations of the graph in a Monte Carlo fashion. The final influence spread of a seed set according to the live-edge model [8] is given by:

$$E[\sigma(S)] = \sum_{g \subseteq G} P(g)\sigma_g(S) \tag{2}$$

where $P(g)$ is the probability of the specific realization g of the influence network, and $\sigma_g(S)$ is the influence spread of S under this realization. The probability of the realization is derived by the joint probability of the influence edges sampled at this realization.

To simplify and speed up the implementation, we are going to use only one simulation. Moreover, we will reduce the search space of the algorithm by giving as an input a set of selected nodes to search on. To find a possible set of such nodes, you can use the `clean_graph` function as defined above, and set a $threshold = 850$. We utilize these shortcuts because we need to make the algorithm run in time for the lab. You are encouraged to experiment further in your own time with an implementation that matches more the original algorithm, mainly in order to get a firm understanding of the computational demand of influence maximization. The current implementation should not take more the 5 minutes for a seed set of $size = 10$. The function of the algorithm will be called `greedy_algorithm` and will follow the format below. Run it using the above parameters and store the returned seed set. While this runs, develop the code to derive the top 10 seeds based on the heuristics that we defined above and compare them with the set from the greedy algorithm using both, `simulate_spreading` and `dni`.

```
def greedy_algorithm(G, selected_nodes, size):
    """
    Run the greedy algorithm for influence maximization using the set of preselected nodes
    """
    s = set()
    while(len(s)<size):
        #------ Loop through all selected nodes that have not been added to s

        #------ Create a temporary seed set with s and the node of the current loop

        #------ Run simulation using the function "simulate_spreading" from above

        #------ Keep the selected node with the maximum number of infected nodes

    return s
```

What do you observe? Does the greedy outperform the rest?

References

[1] Hogg, Tad, and Kristina Lerman. "Social dynamics of digg." EPJ Data Science 1.1 (2012): 5.

[2] William O. Kermack, and Anderson G. McKendrick. "A contribution to the mathematical theory of epidemics." Proceedings of the Royal Society of London A: mathematical, physical and engineering sciences. Vol. 115. No. 772. The Royal Society, 1927.

[3] Mark EJ. Newman. "Spread of epidemic disease on networks." Physical review E 66.1 (2002): 016128.

[4] Garimella, Kiran, et al. "Balancing information exposure in social networks." Advances in Neural Information Processing Systems. 2017.

[5] Deepayan, C., Yang, W., Chenxi, W., Jurij, L. & Christos, F. Epidemic thresholds in real networks. ACM Trans. Inf. Syst. Secur. 10(4), 1:11:26 (2008)

[6] Maksim Kitsak, Lazaros K. Gallos, Shlomo Havlin, Fredrik Liljeros, Lev Muchnik, H. Eugene Stanley, and Hernn A. Makse. "Identification of influential spreaders in complex networks." Nature physics 6.11 (2010): 888-893.

[7] Vladimir Batagelj and Matjaz Zaversnik. "An O (m) algorithm for cores decomposition of networks." arXiv preprint cs/0310049 (2003).

[8] David Kempe, Jon Kleinberg, and va Tardos. "Maximizing the spread of influence through a social network." Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2003.

[9] Chen, Wei, Yifei Yuan, and Li Zhang. "Scalable influence maximization in social networks under the linear threshold model." Data Mining (ICDM), 2010 IEEE 10th International Conference on. IEEE, 2010.

[10] Panagopoulos, George, Fragkiskos D. Malliaros, and Michalis Vazirgiannis. "DiffuGreedy: An Influence Maximization Algorithm Based on Diffusion Cascades." International Workshop on Complex Networks and their Applications. Springer, Cham, 2018.