Controlling Epidemics on Networks

Final Project – 1.022: Introduction to Network Models, Fall 2018

In this project we will consider the problem of epidemic spreading on networks and how to prevent their outbreaks. The project will consist of three main parts, dealing with both theoretical and numerical analysis of epidemics on networks.

Parts I and II will consider the theoretical and computational analysis of some epidemic models. For these parts we will focus on a real-world network of global flight connections as our main dataset. Specifically, in Part I you will need to get the main dataset that will be used in the subsequent parts of the project. This is similar to the homework that you have been doing for the course, and will ensure that there are no errors with the implementation of your vaccination strategy later. Part II will then guide you through designing simple spread models on networks that you will use in order to test various immunization strategies for the airport network from a theoretical and practical viewpoint.

Finally, in Part III you will have to design a general immunization strategy for a set of (generic) networks on which an SIS epidemic is simulated. You will have to write an algorithm that, given the adjacency matrix of a network and some further parameters, provides a list of the nodes to vaccinate. We will test your code on set of test networks, to which you will not have access a priori. You will have to describe the strategy you developed in your report and hand in your code.

Timeline

Each group will have to write a report, and submit the code written in form of an ipython notebook (like in your homework). There will be several deadlines associated with the individual tasks, so make sure you keep track of those deadlines in order not to lose points. As Part II builds up on Part I, you will have to hand in your results for this part earlier. We will provide you with a correct solution for the first part afterwards to ensure that there are no follow-up errors in Part II.

Timeline

There will be the following deadlines.

- **Lecture 22:** Get familiar with the data and hand in your results for Part I.
- **Lecture 25:** Oral Presentation (15 minutes).
- **One week after Lecture 25:** Final project report (including all code).

1 Part I – Getting familiar with the data

Throughout the first two parts of the project we will consider a real world network as our main example, namely, the global network of flight connections. In this first task, you will have to read in the data, construct and visualize the network, and perform simple computations on the data. Many of these tasks should be familiar to you from your homework.

The original dataset was made available by Tore Opsahl and is discussed in the blog post

> Opsahl, T., 2011. *Why Anchorage is not (that) important: Binary ties and Sample selection.* Available at http://wp.me/poFcY-Vw.

We have provided you with 2 `csv` files, that contain all the information of all the airports (nodes), and flights (links). In this first part you should familiarize yourself with this dataset, and conduct a first analysis which will also be useful for the later tasks. Moreover, the airport dataset will be one of the networks on which your algorithm in Part III of this project will be evaluated on, so it might be useful to study it carefully.

Your tasks for this part of the project are the following.

Task 1 Construct a network from the `csv` data (5 points)

In the network data, the edges are directed and correspond to the number of flights from airport A to airport B. You should construct a network G from this data as follows.

- Read in the edge data and create an initial (directed) network.
- We will work with undirected graphs in the first two parts of this project, so form the undirected graph of this network by symmetrizing its adjacency matrix as follows

$$A_{sym} = (A + A^\top)/2.$$

- If the undirected graph obtained is not connected, keep its giant component as your new graph and work with it from here onwards.

Hint: Note that command `networkx.DiGraph.to_undirected` returns an undirected graph with the same name and nodes and with edge $(u, v, data)$ if either $(u, v, data)$ or $(v, u, data)$ is in the digraph. If both edges exist in the digraph and their edge data is different, **only one edge is created with an arbitrary choice of which edge data** to use. You must check and correct for this manually if desired, or use other methods to obtain the symmetry operation required above.

Task 2 Analysis of the airport network (10 points)

Now that you have obtained the network, perform some initial analysis on it.

- Compute the degree, closeness, betweenness, eigenvector, page-rank, and Katz centrality measures. For page-rank and Katz (that depend on the choice of a parameter) select any *valid* parameter.
- Describe the results that you have obtained from this analysis. Are there some airports that are consistently ranked as important throughout?

- Plot the flight network, using some of the extra information for the node files: use the longitude and latitude coordinates provided when drawing the nodes; color the nodes according to their country labels, and scale the node size according to its eigenvector centrality. You may find it useful as well to adjust the transparency of the edges for visual clarity.

Task 3 Produce a simple vaccination function (5 points)
In Part III of this project you will have to develop an algorithm for efficiently vaccinating a network to prevent an epidemic from spreading. In this last task of Part I, you will write a simple vaccination function that chooses the nodes to be vaccinated according to degree (i.e., the highest degree nodes are to be vaccinated). Your function should take as inputs a graph adjacency matrix `A` (in the form of a sparse scipy matrix), infection rate `beta` (float), recovery rate `gamma` (float), and the number of nodes to immunize `M` (integer), and return an array `nodeList` (numpy array) containing the ids of the `M` nodes to be vaccinated. Your function should thus be of the following form:

```
def vaccinateNetwork(A,beta,gamma,M):
    # do some calculations..
    # insert your code...

    return nodeList
```

Make sure you hand in your results for Part I in electronic form as ipython notebook and pdf (exported from the notebook) on or before **Lecture 22**. Late submissions will not be accepted.

2 Part II – Analysis and control of epidemics

2.1 Networked susceptible-infected-susceptible model

In this part, we study in more detail the model that you have seen in the course for the spread of a virus over networks. We denote the set of the vertices with $V = \{v_1, \ldots, v_n\}$ and the adjacency matrix with $A = [a_{ij}]$. The state of node v_i at time $t \geq 0$ is a binary random variable $X_i(t) \in \{0, 1\}$. The state $X_i(t) = 0$ (resp., $X_i(t) = 1$) indicates that node v_i is in the susceptible (resp., infected) state. We define the vector of states as $X(t) = (X_1(t), \ldots, X_n(t))^\top$. The state of a node can experience two possible stochastic transitions:
1) Assume node v_i is in the susceptible state at time t. This node can switch to the infected state during the time interval $[t, t + \Delta)$ with a probability that depends on: (i) an infection rate β, (ii) the probability of contact with its neighboring nodes $N_i = \{j \mid a_{ij} \neq 0\}$ and their states. We can write the probability of this transition as

$$\text{Prob}[X_i(t + \Delta) = 1 | X_i(t) = 0, X(t)] = 1 - \Pi_{j \in N_i \wedge X_j(t) = 1}(1 - \beta a_{ij}). \quad (1)$$

We usually have $\beta \ll 1$. Therefore, instead of (1), we use the following first-order approximation:

$$\text{Prob}[X_i(t+\Delta) = 1 | X_i(t) = 0, X(t)] = \sum_{j \in N_i} \beta a_{ij} X_j(t). \tag{2}$$

2) Assuming that node v_i is infected, the probability of v_i recovering back to the susceptible state in the time interval $[t, t+\Delta)$ is given by

$$\text{Prob}[X_i(t+\Delta) = 0 | X_i(t) = 1] = \gamma, \tag{3}$$

where $0 \leq \gamma \leq 1$ is the curing rate.

Task 1 First-order approximation of the transition probabilities (5 points)
Show that the first-order approximation in (2) is in fact an upper bound for the original transition probabilities in (1).

The spread model characterized by (2) and (3) may be hard to analyze for large-scale networks. One standard approach is to use a *mean-field approximation* of the model: define $p_i(t) = \text{Prob}[X_i(t) = 1] = \mathbb{E}[X_i(t)]$, i.e., the marginal probability of node v_i being infected at time t. We can use (2) and (3) to approximate the dynamics of $p_i(t)$:

$$\frac{dp_i(t)}{dt} = \beta(1 - p_i(t)) \sum_{j=1}^{n} a_{ij} p_j(t) - \gamma p_i(t). \tag{4}$$

This approximation is widely used in the field of epidemic analysis and control, since it performs numerically well for many realistic network topologies.

Task 2 Simulating the spread dynamics using the mean-field approximation (20 points)
Suppose that the first 20 nodes (according to their ids in the dataset) are initially infected.

a) Use (4) to simulate the evolution of the expected size of the infection defined as $\bar{p}(t) = \sum_{i=1}^{n} p_i(t)$ over time, assuming a recovery rate $\gamma = 0.1$ and an infection rate $\beta = 0.1$. (Hint: you can use $\frac{dp_i(t)}{dt} \approx \frac{p_i(t+\Delta) - p_i(t)}{\Delta}$ where $\Delta = 0.05$ and time horizon $[0, 5]$.)

b) Use simulations to find the smallest value of β that results in an outbreak (i.e., the infection never dies out).

Task 3 A criteria for the extinction of the epidemic (15 points)
Considering only the linear terms in (4) gives an upper bound on the dynamics of $p_i(t)$:

$$\frac{dp_i(t)}{dt} \leq \beta \sum_{j=1}^{n} a_{ij} p_j(t) - \gamma p_i(t). \tag{5}$$

The objective of this task is to use this upper bound to find conditions that ensure virus extinction, that is to have $\lim_{t \to \infty} \bar{p}(t) = 0$.

a) Define $q_i(t) = e^{\gamma t} p_i(t)$. Use (5) to show that

$$\frac{dq_i(t)}{dt} \leq \beta \sum_{j=1}^{n} a_{ij} q_j(t).$$

b) Let $\lambda_{\max}(A)$ denote the largest eigenvalue of A. Use part (a) to show that

$$\bar{q}(t) \leq K e^{\beta \lambda_{\max}(A) t} \bar{q}(0),$$

for some $K > 0$, where $\bar{q}(t) = \sum_{i=1}^{n} q_i(t)$. Use this to prove the following upper bound on the expected size of the infection:

$$\bar{p}(t) \leq K e^{(\beta \lambda_{\max}(A) - \gamma) t} \bar{p}(0).$$

c) Show that a sufficient condition for virus extinction, that is to ensure that the virus will eventually die out, is to have

$$\lambda_{\max}(A) < \frac{\gamma}{\beta}. \tag{6}$$

Effective immunization strategies: An immunization strategy is to choose a subset of nodes $I \subseteq V$ and immunizing them against the virus. An immunized node can neither get infected nor pass the infection. We call an immunization strategy "effective" if it results in the eventual extinction of the virus (almost surely), *no matter how widespread the initial infection is*. A potential idea for designing an effective immunization strategy is to rank the nodes based on some centrality measure and immunize them in order of their centralities until the condition (6) is satisfied. Note that immunizing node v_i is equivalent to removing the i-th row and column of A.

Task 4 Centrality-based effective immunization strategies (15 points)

a) Assume that $\beta = 0.01$ and $\gamma = 0.4$. (i) What is the minimum number of immunizations required to satisfy condition (6) if you use degree centrality to sort the nodes? (ii) What is this minimum number when you use eigenvector centrality instead?

b) Design a better immunization strategy that requires to immunize at least (i) 1 node less, and (ii) 9 nodes less, than the best of the two strategies in part (a)

3 Part III – Design your own immunization scheme

Task 1 Strategy description (10 points + 10 bonus points)

In the final part of this project, you have to design an efficient strategy for vaccinating a network. Your function should be of the same form as indicated in Task 3 of Part I of this project (c.f. Task 3 in Part I).

```
def vaccinateNetwork(A,beta,gamma,M):
    # do some calculations..
    # insert your code...

    return nodeList
```

You will have to hand in a vaccination function together with your final report explaining the reasoning and, where applicable, any addional calculations made to justify this strategy. Your vaccination strategy should be well motivated and described in your report; mereley submitting the code will not be enough to gain full points.

Your algorithm will be evaluated on 4 test networks. The first test network is the airport network used in Parts I and II of this project. There will be three other networks on which your algorithm will be evaluated, to which you will however not have access. The only additional information is that these networks have been generated according to a) a configuration model, b) a Barabasi-Albert model, and c) a stochastic blockmodel. We may reveal some further information about these networks within the duration of the project.

The evaluation of your algorithm will be according to a full stochastic simulation of an SIS dynamics. For each network we will report the mean performance in terms of the fraction of the nodes in the network that is infected $\pm$ the standard deviation, after a sufficiently long simulation time (which will in general be different for each network, but reported back to you with the results). The lower the fraction of infected nodes, the better the score of your algorithm.

In terms of computational complexity, we will run your function for a fixed amount of time and, if no `nodeList` is returned by that time then no node will get immunized. This upper bound on the running time will of course depend on the number of nodes n in the network considered and will be equal to $\sqrt{n}\, t_{\text{inv}}(n)$, where $t_{\text{inv}}(n)$ is the amount of time required to invert a full $n \times n$ matrix.

You may want to test your algorithm on some further networks as well. To this end, you can employ some of your own simulations and approximations developed above, or use a third party python package. For our evaluation we will make use of the companion python package to the book *Mathematics of Epidemics on Networks* by Kiss, Miller, and Simon (Springer, 2017), which contains plenty of addional information and resources on epidemic spreading on networks.
https://github.com/springer-math/Mathematics-of-Epidemics-on-Networks.

Note that you should *not* employ the functionality of this or any other simulation package for any of the earlier parts of this project.

1.022 Introduction to Network Models

Fall 2018