Lab 6: Diffusion, Infection, and Influence Models (Newman 6.13, 17-17.4, 17.6-17.11*)

I have provided some stub code in `lab6_stub.ipynb` located online. At the top of the notebook change the name and NetID listed there. Please make sure you label your print statements, so we know what you are printing. In the end you are submitting one `ipynb` file and one `pdf` file, both named with your NetID: `lab4_abc123456`.

Data

In this lab you will use the three networks listed below. Do all your analysis first with the dolphin network, and then run your code again for the terrorist and monkey networks. In your PDF submission, discuss briefly the different patterns you see in the evolution of the diffusion and infection processes and relate this to the structure of the network (so your PDF should summarize the results you get using different networks and different parameter choices for the diffusion and infection processes). Technically the dolphin and monkey networks are directed, but we will use them as undirected for this module. When you submit your code, return it to analyzing the dolphin network.

> **dolphins.edgelist** This is a directed social network of bottlenose dolphins. The nodes are the bottlenose dolphins (genus Tursiops) of a bottlenose dolphin community living off Doubtful Sound, a fjord in New Zealand (spelled fiord in New Zealand). An edge indicates a frequent association. The dolphins were observed between 1994 and 2001.
>
> *Citation:* D Lusseau, et al. The bottlenose dolphin community of Doubtful Sound features a large proportion of long-lasting associations. Behavioral Ecology and Sociobiology, 54:396-405, 2003.
>
> **train.edgelist** This undirected network contains contacts between suspected terrorists involved in the train bombing of Madrid on March 11, 2004 as reconstructed from newspapers. A node represents a terrorist and an edge between two terrorists shows that there was a contact between the two terroists. The edge weights denote how 'strong' a connection was. This includes friendship and co-participating in training camps or previous attacks.
>
> *Citation:* Brian Hayes. Connecting the dots: can the tools of graph theory and social-network studies unravel the next big plot? American Scientist, 94(5):400-404, 2006.
>
> **macaque.edgelist** This directed network contains dominance behaviour in a colony of 62 adult female Japanese macaques (Macaca fuscata fuscata). A node represents a macaque and an edge represents dominance of the left node over the right node.
>
> *Citation:* Yukio Takahata. Diachronic changes in the dominance relations of adult female Japanese monkeys of the Arashiyama B group. The Monkeys of Arashiyama.

> State University of New York Press, Albany, pages 123-139, 1991.

Section 6.13 Diffusion

Diffusion is a process that we are all familiar with. It is the process that describes how a drop of dye spreads out through a glass of water. Fundamentally, diffusion relies on a "pressure" difference (sometimes called a pressure gradient). In class we discussed how this process can be adapted for use with networks. Specifically we model diffusion processes on all the edges of the network, where quantity flows along an edge whenever there is a difference in the values of the nodes on each end.

Processes by definition operate over time, so our simulation is one that will evolve over time. There are many ways of calculating such evolution, but for now we'll use a very simple tactic. Our approach is to break this process down into very small steps where we assume the derivative is constant. In general, the derivative won't be constant, but we can make the steps we choose small enough that this assumption is reasonable. This isn't the most efficient strategy, but for these purposes it will work fine for demonstration. If the system obeys a differential equation of the form,

$$\frac{dx}{dt} = f(x,t),$$

where f is any arbitrary function that represents the dynamics, then $f(x,t)$ gives the value of the derivative of the state at x and t. If we assume that this derivate stays constant over the small time step dt, then $dx = f(x,t)dt$ is the small amount that x changes over this time step. Therefore, the new state value is the old state value plus this update,

$$x(t+dt) \approx x(t) + f\left(x(t), u(t)\right) dt.$$

Notice that the accuracy of this approximation is determined by how small dt is chosen (smaller steps make the approximation more exact). In the stub code, you'll see I have already set the step size `dt = 0.02` and the total length of time to simulate is `T = 6`. The array `time` is the vector of times from `0` to `T`, each separated by a step size of `dt`. Beneath that, I have initialized the state variable `x` as a vector of all zeros, except for one node which is set to a value of 1 (you can/should change this later to see what happens when different or multiple nodes are activated). I have also created a loop that steps through the `time` values. Your job is to calculate the new value of the state `x` at each time step and update the network visualization. The network visualization is already updated by the custom function I have implemented `called color_by_value(...)`. The loop uses a Python command called `enumerate(...)`. This function takes in a list (or array) and outputs a list of tuples; each of the tuples has the index of the element in the list and then the value of the element in the list. This may become useful for what we do next, depending on how you implement it. Once you have the state updating properly, don't forget to watch the visualizer to see what's going on! Start with a diffusion constant of 1, but then you can change it to see how it effects the evolution – make a comment about this in your report.

In class we discussed that the diffusion process approaches an equilibrium when time gets large and we specifically described how that equilibrium was related to the eigenvectors of the Laplacian matrix. You can reuse your code from the module on centrality to calculate the appropriate eigenvector(s) for this module. To show that the system reaches this equilibrium, make a plot of the

distance between the state `x` and the equilibrium you've calculated over time (this is where the `enumerate` might come in handy since you'll need to store your value of distance at each time step). Don't forget that some normalization might be necessary! Also note that "distance" can be measured in several different ways, I leave it to you to decide which one you want to use (but in the end distance is a single scalar number). The stub code has the code to make this plot, you just need to replace the `xvalues` and `yvalues` and update the axis labels appropriately. Properly label your graph, making it clear what choice of distance function you used.

Sections 17-17.11: Infection Models

We also talked about infection models as a process that evolves on networks. We're now going to simulate both a SI and SIR model on this same network (note that to include network effects you should use the networked, *not the population-level*, SI and SIR models). We will take the same approach we used to simulate the diffusion process above - this means you can reuse a lot of your code, just update the dynamics to the appropriate SI and then SIR dynamics. Note that you don't need to change the `dt`, `T`, or `time` definitions, but it is important to reinitialize the state variable `x` setting it back to your chosen initial starting state. Make a plot of the expected fraction of infected people over time using a SI infection model. To plot multiple lines on the same plot, just add more of the `plt.plot(x,y)` commands. You can differentiate the curves by giving a color as a third parameter: `plt.plot(x,y,'r')`, you can pick from: b, r, g, k (there are more, but this should do). First take the probability of infection $\beta = 1$ and then you can adjust it to see how it effects the evolution.

Now make a plot of the expected fractions of susceptible, infected, and removed people over time using a SIR infection model. In class we identified an interesting relationship between the largest eigenvalue of the adjacency matrix and the parameters of the SIR model. For the SIR model (not the SI model) investigate how changing the value of γ (recall γ is the rate at which people are removed/recovered) changes the shape of your plots - you should calculate this eigenvalue so you can pick your values for γ appropriately. Describe this briefly and explain why we see this effect. Again, start with a value $\gamma = 1$ and then vary it from there.

Influence Models

We concluded our discussion of diffusion-type processes talking about the independent cascade and linear threshold models. Let's take a closer look at some of the claims we made for the independent cascade (IC) model. In the stub code, I have implemented an "online" version of the IC model. The function `independent_cascade(...)` assumes a uniform probability of activation for all the edges of the graph. Although the IC model is more general (works for edge dependent probabilities), we will do this so that we can compare with the results of the SIR model above. This function propagates the cascade once, however, because the activation happens in a stochastic manner, it is necessary to repeat this many times and average the results to determine the expected probability of a node being activated. Do this averaging over over 1000 repetitions.

In class we also mentioned that these models can be efficiently implemented as a percolation process: by computing all the probabilities ahead of time rather than online. To do this,

programatically flip a coin for every edge to determine if it will propagate the activation, if activation reaches one of the adjacent nodes. Perform a percolation of the network based on these coin flips. The activated portion of the network is the set of nodes in the same component as the initial seed set. You can either use set intersection to compute this from the components of the remaining graph or notice you can use (abuse) the `independent_cascade(...)` function with a probability `p=1` to find the nodes connected to the initial seed set. Notice again, that you must average this activation over many repetitions of the percolation process. This percolation process changes the graph, so you'll want to make a copy of it each time before you do this. The network object we use for the visualizer `D3Graph` requires a full copy, which can be achieved by using `G2 = deepcopy(G)`. The function `deepcopy(...)` is already imported at the top of the stub code (you can see this function used in the `independent_cascade(...)` function).

Compare the two vectors of *average* activation by plotting them both on the same figure. Plot the the average activation against the node indices. They should match!

Your final task is to observe that the late time infection of the SIR model can be predicted by the independent cascade model. The probability used in the IC model must be chosen appropriately to make an equitable comparison with the SIR process. Thus the IC probability should capture the infection and recovery rates of the SIR model. In an infinitesimal interval of time dt, the probability that the SIR infection spreads across an edge is βdt. Using similar logic to our derivation of the distribution of recovery times in the population SIR model, we can observe that the probability of infecting a neighbor is

$$p = 1 - e^{-\beta\langle\tau\rangle},$$

where $\langle\tau\rangle$ is the expected time that the infectious node will stay infectious. This value $\langle\tau\rangle$ is determined by the rate of recovery, γ, and is found similar to our calculation of the SIR reproduction number:

$$\langle\tau\rangle = \int_0^\infty \tau \cdot \gamma e^{-\gamma\tau}\, d\tau.$$

Using this probability, compute the independent cascade models above. On your plot of average activation, now plot the final recovered probability for each node from the SIR model. All three lines should match quite well! For some choices of β and γ the match is not perfect. Test your code for $\beta = 1$ and $\gamma = 1$ and once it is working you can test other values.

In [ ]: