Markov Chains and NetworkX

Daryl DeFord

June 19, 2019

1 Introduction

This document provides a set of examples for getting used to the main python package for working with graphs: NetworkX. It should also help reinfocce some of the ideas about Markov chains and MCMC from yesterday.

2 Constructing Graphs with Annotations

The first task is to construct some graphs. One easy way is to start with an empty graph and then add nodes and edges sequentially. The node labels can be either integers or strings and edges are represented as tuples of nodes. As a first step, you should generate your ego graph from yesterday.

```
>>>g2 = nx.Graph()
>>>g2.add_node("me")
>>>g2.add_node("cat1")
>>>g2.add_node("othercat")
>>>g2.add_edge("me","cat1")
>>>g2.add_edge("me","othercat")
>>>g2.add_edge("me","sibglings")
>>>g2.add_edges_from([("mom","dad"),("mom","siblings"),("dad","siblings")])
>>>print(list(g.neighbors("me")))
s
```

NetworkX also has a bunch of prebuilt graph families like grid graphs, complete graphs, etc.

3 Plotting Graphs

NetwrokX provides functionality for plotting networks in the plane. The main function is nx.draw which takes a long list of possible arguments.

```
>>> g = nx.grid_graph([4,4])
>>> node_colors = {x:random.random() for x in g.nodes()}
>>> edge_colors = {x:random.random() for x in g.edges()}
>>> nx.draw(g,pos = {x:x for x in g.nodes()}, node_color = [node_colors[x] for x in g.nodes()],
    edge_color = [edge_colors[x] for x in g.edges()],
    width =3, cmap="jet")
```

Try our plotting your ego graph and changing the node and edge colors. You can also adjust the layouts of the nodes by passing a dictionary that maps points to (x, y) coordinates in the plane.

4 Markov Chains on Graphs

Our next step is to implement a simple random walk on our graph. We will make a lot of use of the randon.choice() function which makes a uniformly random selection from any list of objects. Our plan here is to move to an adjacent node at every step. What statistics can you compute for your chain? You should also plot a histogram of the visited nodes to see how close the chain has gotten to the steady state distribution.

```
>>>state = random.choice(list(g.nodes()))
>>> states = []
>>>for i in range(5):
>>>    node_colors = {x:0 for x in g.nodes}
>>>    edge_colors = {x:0 for x in g.edges}
>>>    old_state = state
>>>    states.append(state)
>>>    state = random.choice(list(g.neighbors(state)))
>>>    edge_colors[(old_state, state)]= 1
>>>    edge_colors[(state, old_state)]= 1
>>>    node_colors[old_state] = 2
>>>    node_colors[state]=1
>>>    plt.figure()
>>>    nx.draw(g,pos = {x:x for x in g.nodes()}, node_color = [node_colors[x] for x in g.nodes()],
        edge_color = [edge_colors[x] for x in g.edges()],
        width =3, cmap="jet")
>>>    plt.show()
```

Other thoughts: It is an interesting exercise to form the transition matrix for this simple walk. Once you build the matrix, you should extract the leading eigenvector and check that the steady state is proportional to the degrees of the nodes in the graph.

5 MCMC on Graphs

To implement MCMC we need a score function on our nodes. One easy way to implement this is to create a score dictionary that maps the node labels to scores.

```
>>> uniform_scores = {x: 1 for x in graph.nodes()}
>>> length_scores = {x: len(x) for x in graph.nodes()}
```

Now we need to implement an accepatance function that uses the α acceptance ratio that we talked about yesterday

$$\alpha = \min(1, \frac{s(\hat{y})}{s(y)}) \frac{Q_{\hat{y},y}}{Q_{y,\hat{y}}}$$

and remain in place and add the current state to our chain again if we don't accept the new state.

6 MCMC on (discontiguous) Partitions

Finally, we can start to think about how to implement Markov chains on partitions by borrowing a model called Galuber Dyanmics from physics. We can start with a random partition:

```
>>> for n in graph.nodes():
>>>     graph[n]["partition"] = random.choice([0,1])
```

and at every step we choose a random node and change it to the color of one of its neighbors.