Lab 10: Network Graphs

Contents

Objectives

By the end of this lab you will understand:

- How graphs are used to model various networks

By the end of this lab you will be able to:

- Create graphs with `networkx`
- Compute various metrics about graphs

Feedback

Since this is our last lab, it's time to get your feedback about how the module has gone. This will be invaluable in improving both this module and the overall programme in future years.

> **Task 1.** Spend 10-15 minutes filling out the feedback form at `https://www.surveymonkey.co.uk/r/TCZFJRM`. Your participation will be anonymous, and we would really appreciate constructive and thoughtful feedback.

1 networkx

The most commonly used module for handling graphs in Python is `networkx`.[1] This module has great documentation, and in particular has a nice tutorial to introduce you to its functions and features.[2] We thus do not replicate the basic information on setting up graphs here, but do present the relevant commands in the discussion that follows. This module is often imported using the abbreviation `nx`.

[1] `https://networkx.github.io/documentation/stable/index.html`
[2] `https://networkx.github.io/documentation/stable/tutorial.html`

2 Graphs

In mathematics and computer science, the study of graphs is a rich and longstanding area of research. In crime science, it is important to understand graphs, as they can be used to represent any number of networks in which criminals might act, such as the layout of roads or the connections between members of a community.

2.1 Basic definitions

A graph G is a pair (V, E), where values in V represent *nodes* or *vertices*. In a social network, for example, each node v would represent a different user of the service. Values in E represent *edges*, which can be either *directed* or *undirected*. In Facebook, for example, an edge (u, v) could indicate that u and v are friends (which is a mutual relationship, so the edge would be undirected). In Twitter, an edge (u, v) could indicate that u is a follower of v (which is not a mutual relationship, so a second edge (v, u) would be needed to show that v follows u back). In a road network, edges would also likely be directed in order to indicate the direction of travel (with two-way streets represented by two edges).

An undirected graph can be initialized using `networkx` as `graph = nx.Graph()`. We can add a node `u` to the graph using `graph.add_node(u)`, and can add an edge (u, v) to it using `graph.add_edge(u, v)`, where `u` and `v` are already nodes in the graph.

2.2 Decorations

Both nodes and edges can be *decorated* with different values or *attributes* associated with those objects. In Facebook, for example, a node might be decorated with the user's name and other information about them, and an edge might be decorated with the date on which the two users became friends.

To decorate a node `u` with some value `val` for an attribute `attr`, in a graph `graph`, we can use `graph.node[u][attr] = val`. For example, if we identify members of a social network in a graph using some numeric identifier but also want to store their name, we could use `graph.node[123]['name'] = 'sarah'`. To decorate an edge we can use `graph[u][v][attr] = val`.

2.3 Graph representations

One of the most common ways to represent graphs is using an *adjacency matrix*.[3] This is a square matrix with each node in the graph representing both a row and column. The entry in the i-th row and the j-column indicates whether or not those two nodes are connected (i.e., share an edge) in the graph. In a traditional adjacency matrix, the value is 0 if there is no edge in between them and 1 if there is. It is also possible for the value in the matrix to be the decoration on the edge, if one exists, or to take on some other non-zero value to provide additional meaning.

To create an adjacency matrix given a graph `graph`, we can use `nx.adjacency_matrix(graph)`, which returns a NumPy matrix. We can also use `nx.to_dict_of_dicts(graph)` or `graph.adj` to return a "pure" Python representation, which is a dictionary in which keys are nodes, and values are themselves dictionaries in which keys are the nodes to which the original node is connected. To go the other way, it is possible to construct a graph from an adjacency matrix represented in this dictionary of dictionaries format by running `graph = nx.Graph(dict_of_dicts)`. (It may be easier, however, to simply go through the matrix using nested `for` loops and add the edges one-by-one using `graph.add_edge`.)

2.4 Node degree

The *degree* of a node in the graph is the number of other nodes to which it is connected. In Facebook, for example, the degree of a node would be the number of friends that user has. In a directed graph (i.e., a graph with directed edges), this could be further broken down into *in-degree* and *out-degree*. In Twitter, for example, the in-degree would represent the number of followers a user has, and the out-degree would represent the number of people they are following.

To get the degree of all the nodes in a graph, we can run `graph.degree`. This is a dictionary-like object, so we can index into it to get the degree for a specific node using `graph.degree[u]`.

[3] `https://en.wikipedia.org/wiki/Adjacency_matrix`

2.5 Paths

A *path* between two nodes u and v in a graph is a list of edges connecting them. For example, if Alice is friends with Bob, Bob is friends with Charlie, and Charlie is friends with Dora, then one path from Alice to Dora is ((Alice,Bob), (Bob, Charlie), (Charlie, Dora)). This path is of length three, since it consists of three edges. If Alice and Dora have a mutual friend, Edgar, then ((Alice,Edgar), (Edgar,Dora)) is also a path. Since this path is of length two, assuming Alice and Dora aren't friends it is the *shortest* path between Alice and Dora, since it has the fewest number of edges. In a road network, a path represents the set of directions we would follow to get from one location to another.

To see if any path exists between two nodes, we can run `nx.has_path(graph, u, v)`. To find the shortest path between all pairs of nodes in a graph, we can run `nx.shortest_path(graph)`. To find the shortest path between a specific pair, we can run `nx.shortest_path(graph, u, v)`. This returns a list whose first element is `u` and whose last element is `v`, and where every pair of neighbors is connected by an edge in the graph. Crucially, it is the shortest such path.

2.6 Centrality metrics

One important question within a graph is how *central* a given node is. It turns out that there are many possible ways to define the centrality of a node, but we stick with the following three concrete examples:

- *Degree* centrality is higher for nodes that are connected to a higher proportion of other nodes in the graph (meaning nodes with higher degree). It can be computed for every node in the graph using `nx.degree_centrality(graph)`, or for a specific node `u` using `nx.degree_centrality(graph, u)`. In a social network like Facebook, for example, this would rank as more important those users who have more friends.
- *Closeness* centrality is higher for nodes where the shortest paths between them and all other nodes are shorter than they are for other nodes (meaning they are a short distance away from, or close, to other nodes). It can be computed using `nx.closeness_centrality`.
- *Betweenness* centrality is higher for nodes that lie along more shortest paths in the graph (meaning they sit along the shortest path between the most pairs of nodes). It can be computed using `nx.betweenness_centrality`. In a transportation network, for example, this would rank as more important those areas (intersections, airports, train stations, etc.) that act as a "hub" of sorts, through which many people must travel.

Each of these centrality metrics has uses cases in which they are appropriate, so it is worth taking some time to think about what each one provides. Degree centrality, for example, is simple and easy to think about, but it also has some limitations: if a road intersection is very well-connected within its own local network, for example, but is on an island that is disconnected from the rest of the world, then it could be mistakenly considered a central area due to its high degree. Similarly, closeness centrality might rank as very important airports like the one in Vienna, given its physical proximity to many other airports within Europe. Few international flights transit through there, however, so if the goal was to rank the world's busiest airports (a list which is dominated by hubs in the US and Asia) then something like betweenness centrality would better capture the desired measurement.

3 Putting It All Together

Today, we'll look at a well-known dataset concerning a London-based gang that has been in operation since 2005. All members of the gang identify as black males, but are from different parts of the world. You can find this data in the `data/` directory as `london_gang.csv` and `london_gang_attr.csv`. The attribute file, `london_gang_attr.csv`, has nine columns:

1. `ID`, which is a numeric identifier for this gang member.
2. `Age`, which is their age.
3. `Birthplace`, which indicates their place of birth and is used as a proxy for their ethnicity. It codes locations as 1 = West Africa, 2 = Jamaica, 3 = UK, and 4 = Somalia.

4. `Residence`, which we won't use.
5. `Arrests`, which is the number of times they have been arrested.
6. `Convictions`, which is the number of times they have been convicted of a crime.
7. `Prison`, which is either a 0 (meaning they haven't been to prison) or a 1 (meaning they have).
8. `Music`, which we won't use.
9. `Ranking`, which indicates on a scale from 1 to 5 how high up that member is in the gang's hierarchy.

The connection file, `london_gang.csv`, is the adjacency matrix for the gang, representing connections between each pair of members. The top row and leftmost column both contain identifiers `ID` for gang members, and the entry in the matrix corresponding to a given pair of members is 0 if they have no relationship, 1 if they hang out together, 2 if they have co-offended, 3 if they have co-offended on a serious crime, and 4 if they have co-offended on a serious crime and are related (i.e., are kin).

This dataset was created by researchers in Manchester, who used it to support a theory around *homophily*; namely, that gang members of the same ethnic group co-offend more than gang members of different ethnic groups [1, 2]. The analysis done by the researchers relied crucially on graph theory, and today we'll also be using graphs to analyze some preliminary aspects of the dataset.

Task 2. First, in a file `stats.py`, write code to compute some basic descriptive statistics about the gang. In particular, compute and print out (1) the mean age; (2) the age range; (3) the total number of arrests across all gang members; (4) the total number of convictions; and (5) the total number of members who have spent time in prison.

Task 3. Now, in a file `graph.py`, populate a graph with the information in the adjacency matrix; i.e., create a `networkx` graph where nodes are gang members and an edge exists between two members if they have co-offended (on any type of crime). Nodes should be decorated with the ranking of the gang member.

In this graph, compute the centrality of each gang member, using any metric that you can justify (in a comment). What would you guess is the correlation between the centrality of a gang member, in terms of their offending behavior, and their ranking in the gang? Write this hypothesis in a brief comment and print the outcome of a statistical test for it. In another brief comment, did this test support or reject your hypothesis?

4 Handing In

As always, feel free to continue trying out the various concepts we've covered in this lab. Once you are done, add the files into a `lab10` directory; this should mean adding:

- `stats.py`
- `graph.py`

Push these files, along with the usual `partner.txt`, to the remote repository, and you're done! Please check with one of us on your way out.

References

[1] Thomas U. Grund and James A. Densley, Ethnic heterogeneity in the activity and structure of a Black street gang. *European Journal of Criminology*, 9(4):388–406, 2012.

[2] Thomas U. Grund and James A. Densley, Ethnic Homophily and Triad Closure: Mapping Internal Gang Structure Using Exponential Random Graph Models. *Journal of Contemporary Criminal Justice*, 31(3):354–370, 2015.