Lab 5: Modularity & Assortativity, Partitioning & Community Detection

Download the stub code and rename it as your own for the following exercises. At the top of the file create a commented out section (using #) and place your names and NetIDs there. Please make sure you label your print statements, so we know what you are printing.

Modularity & Assortative Mixing

We've now looked at how connectivity tells us which nodes are similar (cocitation and bibliographic coupling), which nodes are important (centrality), and now we will look at how connectivity implies communities in a network. As discussed in class, the key idea (and assumption) is that communities, or modules, have more connections to nodes within the community than they do to nodes outside the community. The intuition is that if you want to find, for example, groups of friends in a social network, the friends are more likely to be connected to each other than to other people in the network. This relates to small scale structure of the network - cliques are an exaggerated example of high connectivity in a group (exaggerated because every node is connected to every other node).

There is a built-in function in `zen` to calculate modularity, `zen.modularity`, for undirected graphs. In the stub code I have also implemented a version that works with directed graphs. In my directed version, the function takes the graph and a dictionary of groups and returns the modularity value `Q` and also the maximum value this can take `Qmax` for the given connectivity and groupings. The network that we are analyzing is a soccer (football) network `fifa1998.edgelist` defined as follows:

> Our network example describes the 22 soccer teams which participated in the World Championship in Paris, 1998.
>
> Players of the national team often have contracts in other countries. This constitutes a players market where national teams export players to other countries. Members of the 22 teams had contracts in altogether 35 countries.
>
> Counting which team exports how many players to which country can be described with a valued, asymmetric graph. The graph is highly unsymmetric: some countries only export players, some countries are only importers.
>
> Citation: Dagstuhl seminar: Link Analysis and Visualization, Dagstuhl 1-6. July 2001(part of the Pajek datasets)

This means there is an edge from country A to country B if a player originally from country A plays on a club team in country B. I have defined two groups, one based on geography and the other by whether the teams stereotypically import players from other countries for their club teams or typically export players for club teams in other countries (feel free to try other options!). The stub code will already run and print out the modularity of each choice of grouping. Write a few sentences describing why you see the values you get. Notice the modularity values are not tremendously large, so the effects we're seeing aren't huge, but they are noticeable. If you're not "up" on your soccer (football), in my experience, the people who know things about soccer are very willing to talk about it, so ask your buddies!

Next, let's explore the scalar-valued extension of modularity: scalar assortativity. We use scalar assortativity when the nodes do not fall into types or classes, but rather have a real-valued number that describes them. We are interested in determining assortative or disassortative mixing according to, for example, age, income, height, etc. In this case, we will explore this concept through a network that captures the international import/export activity of countries.

> This network captures the international trade between countries during the years 2012-2014. A weighted edge connects country A to country B if A exports materials and/or services to B. The weight of this edge is given by the net trade value in USD. The data was aggregated using the http://comtrade.un.org API.

In addition to the network, we load several different statistics about the countries involved, namely the GDP of each country, the life expectancy, and the average taxation on traded goods. We will use these statistics to determine if countries who have similar GDPs tend to trade together more or less than countries with different GDPs (likewise for life expectancy and taxation). We collect this information from the following sources:

> GDP statistics for 2014 come from the World Bank and quantify the GDP in millions of USD.
>
> Life expectancy statistics are drawn from the CIA World Factbook.
>
> Tariff information quantifies the simple mean applied tariff, which is the unweighted average of effectively applied rates for all products subject to tariffs calculated for all traded goods. This information is available from the World Bank.

Incidentally, we have loaded these using a Python module amusingly called `pickle`. A pickle file essentially stores a Python dictionary in a file. It is a convenient way to easily bring back a dictionary for immediate use. Later in this lab you'll get the chance to do this more manually. The stub code also automatically loads the network and the scalar values and runs scalar assortativity. Again, write a few sentences to describe what these values tell you about the world trade system and reason why these attributes might be assortative, disassortative, or neither.

In our derivation of the assortativity statistic R, we left out a step, namely simplifying the expression involving μ. Do this algebraic activity to show the following simplification (and in the process hopefully helping your understanding of our derivation):

$$R = \frac{1}{2m}\sum_{i=1}^{n}\sum_{j=1}^{n} A_{ij}(x_i - \mu)(x_j - \mu) = \cdots = \frac{1}{2m}\sum_{i=1}^{n}\sum_{j=1}^{n} A_{ij}x_i x_j - \mu^2 = \frac{1}{2m}\sum_{i=1}^{n}\sum_{j=1}^{n}\left(A_{ij} - \frac{k_i k_j}{2m}\right)x_i x_j$$

where

$$\mu = \frac{1}{2m}\sum_{\ell=1}^{n} k_\ell x_\ell$$

Partitioning & Community Detection

In class we discussed several ways to identify groupings of nodes. Both partitioning and community detection admitted a "greedy" approach, with some restrictions, that could be used to find partitions or communities. In the Kernighan-Lin algorithm, one of the restrictions (deviations from a simple greedy approach) was that the cut set size might not necessarily decrease with each pair of nodes swapped. In fact, the algorithm requires us to swap a pair of nodes even if it *increases* the cut set size. To understand why this is an important step in the algorithm, diagram an example in which the cut set size must increase before it decreases to a value smaller than the original.

In our derivation of spectral modularity maximization we left out the step that simplifies the statistic Q after we introduce the variable s. Show that the modularity matrix satisfies the following summation property:

$$\sum_{j=1}^{n} B_{ij} = 0$$

where

$$B_{ij} = A_{ij} - \frac{k_i k_j}{2m}$$

Community Detection in Practice

In this section of the lab, we'll be using community detection to understand hashtags on Twitter – specifically finding hashtags with related meanings. Twitter is a microblogging platform – users write short 140 character messages that others can read. Hashtags are a major way that people quickly flag what their post is topically about. For example, if I tweeted "Texted my boss instead of my wife. #embarrassed" – I'd be flagging my feeling of embarrassment as really salient. The challenge with hashtags is that people make up new ones all the time – and often there are quite a few that are related being used at any given instant. So how do we know that "#embarrassed" and "#fml" carry a similar meaning? One way is to use network models!

We're going to do this by finding communities of hashtags in a hashtag network. To get started, download the `raw_twitter.json.zip` file onto your computer. Expanding it will leave you with the file `raw_twitter.json`. By way of context, these tweets represent 10% of all the content that was generated on Twitter around Thanksgiving last year over a time period of approximately 5 minutes. If you look through it, you'll see that it contains a whole bunch of tweets, each one in JSON format.

At this point, you might be wondering – where's the network? It's in the JSON file! We're going to work through a fairly complete network analysis process. And since it's rare to find network data just sitting around, we're going to have to build our network from scratch. In the case of this lab, we're going to build a network in which nodes are hashtags and edges connect hashtags that have appeared together in the same tweet. Think through why linking hashtags that co-occur in tweets could give us a way of finding groups of hashtags with similar meanings. Find two tweets in the provided dataset that support this idea. Find one that doesn't (and explain why). The JSON file is big, so

to do this without crashing your editor, you should use a command like `more`, which is available on both the Windows and Unix command lines. Type `more raw_twitter.json`. This will present the file screen-by-screen on the command line. Hit enter to scroll and hit `q` to exit.

Our goal is to build a hashtag co-occurrence network from the data file provided. This process is going to involve several high-level steps. It's really critical to recognize that, while some of these steps will feel tedious at times, massaging data into network form is a really important skill to develop. It also exposes you to the many decisions that have to be made as you build a network. We'll see a number such decisions in this lab. The major steps are:

1. Identify co-occurring hashtags in the data
2. Build a network from the co-occurring hashtags
3. Detect communities in the network
4. Determine whether the decisions we made in 2 and 3 have given us meaningful communities. (The answer here is: probably not at first. In which case, we'll need to redo steps 2 and 3, trying different settings each time)

1. Identifying co-occurring hashtags in the data

We are going to read in the content of `raw_twitter.json` and output a file that contains only the hashtags. The most sequential approach to doing this is to first create a file `raw_tweets.txt` that contains the text of a tweet on each line (getting rid of the extra JSON data). To do this, you'll need to use the `json` Python library. In particular, look at the `json.loads(…)` function.

Next we will produce the file `hashtag_sets.txt` from `raw_tweets.txt`. Each line of the `hashtag_sets.txt` file should contain a space-delimited list of hashtags, which correspond to all the hashtags that appeared in one tweet. What does it mean if this file has empty lines?

Note that it is also possible (and permitted) for you to do both of these things in one step, omitting the middle file `raw_tweets.txt`. Reading and writing files is quite simple in Python. The command

```
f = open('filename', 'r')
```

opens the file for reading, where the `'r'` parameter can be replaced by `'w'` in order to open the file for writing (be careful – if you open an existing file with the `w' parameter, its previous content will be deleted). The variable `f` represents the file itself and Python makes it easy to do some useful things with it. For example:

```
for line in f:
    print line.strip()
```

will print the file line-by-line. The function `strip()` is a very handy function that eliminates whitespace characters (spaces, tabs, new line) on both ends of a string. Writing to a file is also simple:

```
f.write('writing this out to a file\n')
```

Notice that writing to a file simply writes the string. If you want to insert a new line, you use a special character `'\n'`. Similarly, to insert a tab, you would use `'\t'`. These are called escape characters and are useful to format your output (they can be used in your print statements as well!).

In between reading from and writing to a file, you're going to need to do some string manipulation. There are some very powerful tools for text manipulation (e.g., regular expressions), but we will stick to just a handful of standard python functions. With these you can do pretty much anything you want to do. We've already seen `strip()`. It is often useful (hint, hint) to work in all lowercase letters so that "Bob" and "bob" become the same strings; to do this, use `s.lower()` to convert the text to lowercase. Other useful functions include:

`s.find(c)`	finds the location of a character or string `c` within another string `s`
`s.split(c)`	uses `c` as a delimiter and splits the string `s` into a list of strings, effectively cut at the appearance of `c`
`c.join(a)`	like the opposite of `split()`, it combines the elements of list `a` into a string, inserting `c` in between each element
`s.replace(a,b)`	replaces occurrences of string `a` in string `s` by `b`
`s.startswith(c)`	returns True if `s` starts with the string `c`, otherwise returns False

Also note that strings are actually arrays of characters. So just like lists/arrays, you can index through them: `s[0]`, for example, returns the first element (character) of the string `s`.

When you are done reading or writing a file you close the file with: `f.close()`.

You are working with real data in this case and people do some crazy things when they generate data (in this case when they tweet). They might have autocorrect fails, they may decide to omit spaces in usual places (e.g., after a period) to save characters, or they may try their hand at ascii art. Some of your assumptions and decisions in parsing the data will need to handle such cases. Another challenge is languages. You'll notice we have a lot of languages represented in this Twitter dataset. ASCII is the collection of 128 standard characters in the Roman/English alphabet and number systems. There are other character encodings that allow for more letters – UTF-8 is the most widely used. When we write text to file that has such extra characters, we need to be sure to indicate that there could be other characters other than what is supported by ASCII. To do this we encode the text with an expanded character set, such as UTF-8:

```
f.write((‘writing this out to a file\n’).encode(‘utf-8’))
```

2. Building a network from the co-occuring hashtags

Now that we have the hashtag co-occurrences, we need to turn them into a network. Technically the `hashtag_sets.txt` is a network already. It's a hypergraph. How do we interpret this file as a (hypergraph) network? Why don't we just analyze the hypergraph? We don't want a hypergraph, we want a regular, undirected graph. Read in the file `hashtag_sets.txt` line-by-line and construct the undirected graph in which hashtags are connected by an edge when the pair of hashtags appear in the same tweet. Make sure to think about and properly handle the weight of the edges between hashtags. Save your network in edgelist format:

```
zen.io.edgelist.write(G,'hashtags.edgelist', use_weights=True)
```

3. Detecting communities in the network

We'll use the Louvain method, which is a heuristic that tries to find communities (node sets) that approximately maximize modularity. Given undirected graph, G:

```
cset = zen.algorithms.community.louvain(G)
```

While we could (and usually would at least try to) run community detection on the original graph, in this case, it would take a while (feel free to give it a try – running it overnight if you get time). As it turns out, the method to make the computation time more reasonable is also the same approach we will use to find meaningful communities.

4. Finding the most meaningful communities

In many respects, community detection is more of an art than a science. Let's see what we can do to find more meaningful communities. One quick trick is to focus in on the parts of the network that are big enough to be interesting. One way to do this is to remove small graph components – recall function `zen.components(G)`. Why would it make sense to remove small graph components from our community analysis? What might we miss by doing this? Edit your community detection code to first remove components consisting of fewer than 10 nodes.

One other big thing we can tweak is the network we're giving to the community detection algorithm. Let's try this. We can impose a threshold on the minimum weight of an edge that gets added to the network. What does it mean to ignore low weighted edges (in terms of tweets/hashtags)? As we raise the threshold higher and higher, what does the network represent? Impose a weight threshold of 10.

Now find the communities on the new graph. Print out the number of communities you've detected. Plot a histogram of the distribution in community sizes. To make a histogram just use the following commands:

```
plt.hist(comm_sizes,20)
plt.xlabel('community sizes')
plt.show()
```

where `comm_sizes` is the list of community sizes and `20` is the number of bins you want to use. Write the communities you've detected to htag_communities_w10_c10.txt. When you write the communities to a file you may or may not need to encode the string (it depends on how you have read in the text). If the graph was read in as an edgelist, then no encoding is needed.

Find a community that "makes sense." Interpret the cluster. Find a community that doesn't make sense. Explain why it doesn't make sense.

Now explore the relationship of how the quality of the communities relates to the weight threshold you impose and the component size you impose by trying a variety of these values (note the weight threshold and component threshold don't have to be the same). Each time write the communities to a text file and examine the communities you are finding – the ones that make sense and the ones that don't. Use these observations to generalize the advantages and disadvantages of these threshold settings.

		weight threshold	
		Low	High
component threshold	Low	Q1	Q2
	High	Q3	Q4

Consider the table above. Clearly there is a whole range of values that each threshold can take, but let's just consider the relatively extreme cases. In your PDF, describe the strengths and weaknesses of each quadrant of the table. Present example communities from each quadrant that support your comments.

Finally, pick a topic/theme that you see in the data. For each threshold, find the communities that seem to correspond best to that topic. Which threshold has communities that best represent that topic? Why? Is there a threshold choice that is clearly the best? Why or why not?