Assignment 2

Eduardo López

1 Creation of histograms

A key task in analyzing networks is generating histograms of the properties of the network. Clearly, histograms only apply when the property being measured is associated with multiple values throughout the network, leading to many numbers to track. For example, each node has a degree, and thus there are n values of degree for the entire network.

In the lectures, the creation of histograms and probability distributions of degree has been discussed. How does this change, for instance, for quantities such as triangles, v-shapes, or local clustering? Please review your notes in order to make sure the following discussion makes sense to you. Take `E` as the Enron email network

```
>>> Ht={} # Define a dictionary to hold local triangle counts
>>> for i in E.nodes(): # Network already loaded to G
...     t=nx.triangles(E,i)
...     Ht[t]=Ht.get(t,0)+1
...
>>> import matplotlib.pyplot as plt # Library to plot
>>> plt.xscale('log') # Log scale horizontal axis
>>> plt.yscale('log') # Log scale vertical axis
>>> plt.xlabel('Triangle count t') # Horizontal axis label
>>> plt.ylabel('H(t)') # Vertical axis label
>>> triangNumList=Ht.keys() # Numbers of triangles found on
    nodes
>>> triangFreqList=Ht.values() # Frequencies of those numbers
>>> plt.plot(triangNumList,triangFreqList,'o') # Plots each
    element in triangNumList against corresponding
    triangFreqList
>>> plt.show() # Not everybody needs this command.
```

The result of this code if the histogram of triangles of the Enron network (Fig. 1). The figure has minimum format (labels but missing font

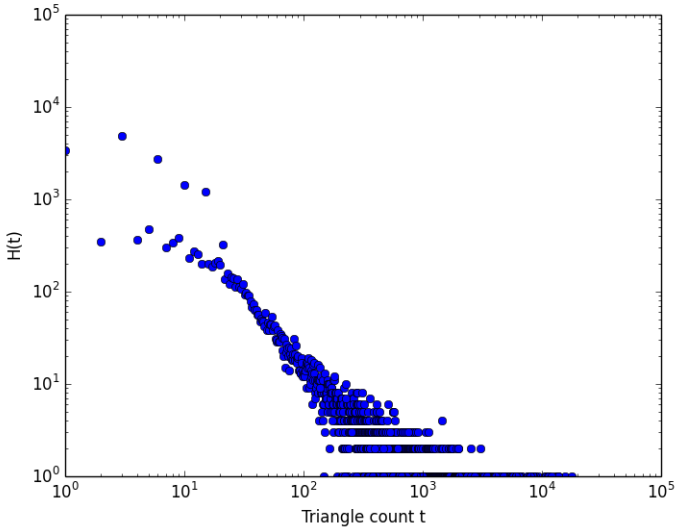

Figure 1: Plot of $H(t)$ for the Enron network produced by `matplotlib`

size control, etc.) but it is our first use of `matplotlib`. The function `plt.plot()` takes elements of two lists (in our case `triangNumList` and `triangFreqList`) and matches them term by term to find the horizontal and vertical coordinates of each point in the plot. The first list provides the horizontal coordinates, and the second list the vertical coordinates. The third argument we used was `'o'`, which is one of a set of built-in symbols that `matplotlib` understands as allowed symbols for a plot; this one in particular produces a small filled circle.

Note in addition, that I have not indicated any instructions on how to save the plot, and this is because the plot can be saved from its screen display in most cases, where a symbol for saving guides us through the steps. Alternatively, before running the command `plt.show()`, you can execute

```
>>> plt.savefig("Hist-t-Enron.png") # Specifies name and figure
    format
```

Either the dialog on the displayed figure or the line commands lead to the same output. Note: you may want to run `plt.show()` after saving the figure because the figure can stay in a buffer and show up as part of the next plot.

Let us now switch gears from worrying about displaying plots, to thinking about why we are making them in a particular way. An important point

about the use of dictionaries in making histograms is that they work well when the quantity being tracked is an integer. Thus, degree, number of local triangles, and number of local v-shapes, to name a few, fit this perfectly. However, when the quantity becomes a real number, dictionaries become more subtle to use. The reason is that, as in all analysis of statistical data of real (decimal) numbers, there is a need to create *bins*. Thus, one does not make a histogram of the number of nodes with local clustering *exactly* equal to c, but rather, of local clustering falling between c and $c + \Delta c$. The histogram is then made by counting all samples that fall in each of the bins, and then plotting the frequencies against some descriptor of the bin, usually either its starting or its middle value.

`matplotlib` has some automated facilities for generating histograms automatically, and being able to plot the vertical direction in logarithmic scale. This can take care of some situations, but when double logarithmic scales are needed (and this is common), this function is not very practical.

Let us generate some clustering values

```
>>> clist=[] # List to hold result of local clustering
>>> def vshape(G,i): # Define a v-shape count function
...     k=G.degree(i)
...     if k<2:
...         return(0)
...     else:
...         return(k*(k-1)/2)
...
>>> for i in E.nodes(): # Loop over nodes to find all
    clustering values
...     t=nx.triangles(E,i)
...     vi=vshape(E,i)
...     clist.append(float(t)/v)
...
>>> clist2=nx.clustering(E) # This is the networkx shortcut
>>> clist2=clist2.values()
>>> plt.plot(clist,'o')
>>> plt.show()
>>> plt.plot(clist2,'o')
>>> plt.show()
```

Now, we can make some potential useful plots to see what we are dealing with. First, we can plot this list in a very trivial way. When `matplotlib` receives a single list, `plt.plot()` does not require two lists. It would plot one list assuming its values should be displayed along the vertical axis, and uses a counter for the positions along the horizontal axis. We can see the

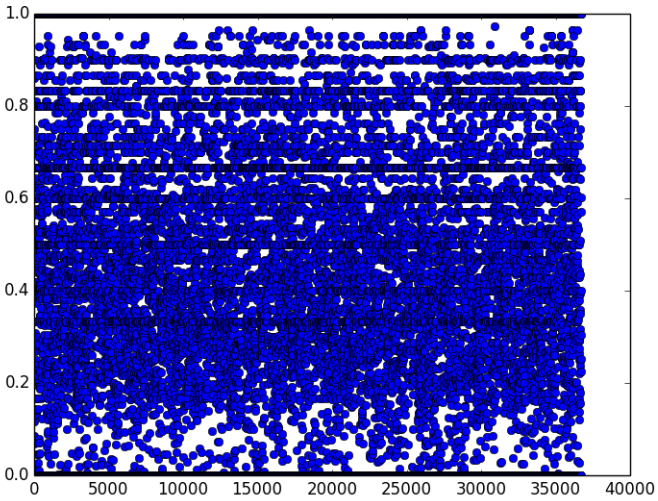

Figure 2: The list of values of c_i for all the nodes of the Enron network. The horizontal axis is merely a counter with no relation to the labels of nodes. The same plot emerges if we use `clist` of `clist2`.

result of the plot in Fig. 2. Looking carefully, you may notice a population of points sitting either along $y = 0$ and $y = 1$ (with y really meaning c_i), and everything else is between these values.

To create histograms of these data we can first make use of the `matplotlib` function `hist`. This is applied as follows

```
>>> plt.hist(clist,30,log=True)
>>> plt.show()
```

The outcome is the plot in Fig. 3. The argument `30` specifies the number of bins to be used, and `log=True` ensures that the vertical axis is logarithmic in scale. How could we generate this ourselves?

The difficult part is in creating our bins. There are in fact various techniques by which one can bin data. The plot in Fig. 3 is binned linearly, which means that Δc is the same no matter what the value of c is. This simplest type of binning is typical and we should always do it so that we make sure we know our histograms as well as possible.

To create a binning for our data, we require some key parameters. First, we want to know the smallest and largest values among the numbers we are binning. These can be found through various means. One of the most

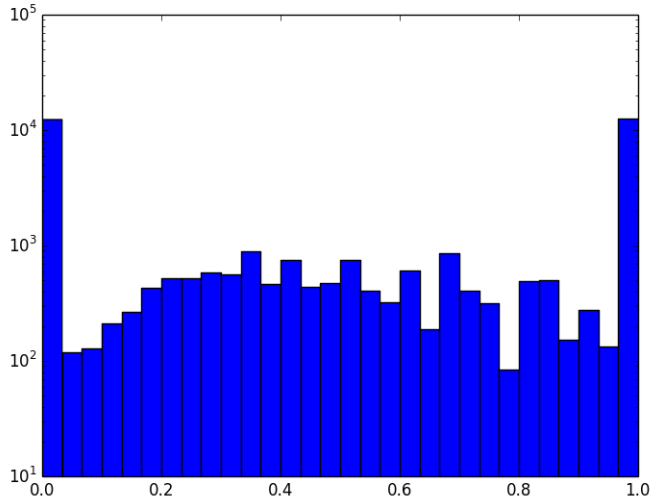

Figure 3: A histogram in vertical logarithmic scale of c_i for the nodes in the Enron network.

straightforward is by using the default functions of `python`. Thus

```
>>> cmin=min(clist)
>>> cmax=max(clist)
```

Subsequently, we need to break up the range between these minimum and maximum numbers into equally-sized bins. But first we need to decide on the number of bins we would like to use. We can define a variable such as `Nbin` and provide it with a value. The key step is to then define ranges of values of c according to this number of bins. The bins then become defined by a counter that determines which of the various bins a value of c falls into. Let us look at the following code

```
>>> Nbin=30 # Repeating choice for plt.hist()
>>> dc=(cmax-cmin)/Nbin # These are the sizes of the bins
>>> dc # Since cmax=1. and cmin=0., then dc=1/30. Check
0.033333333333333333
```

Now, say node i has local clustering c_i. The bins in our problem have the

values

$$\begin{array}{ll} \text{first bin:} & 0 \leq c < dc, \\ \text{second bin:} & dc \leq c < 2 \times dc, \\ \vdots & \\ \text{30th bin:} & 29\times \leq c \leq 30 \times dc, \end{array} \qquad [dc = 1/\text{number of bins}]. \tag{1}$$

Thus, that c_i value needs to be placed in the correct bin of the list above. How do we do it? Note that what identifies the bin, in essence, is a counter (first bin, second bin, etc.). Thus, what we need is an automated way to get the counter. If we introduce the notation q to represent this counter, we see that if c_i belongs to the qth bin, it satisfies

$$c_{\min} + (q-1) \times dc \leq c_i < c_{\min} + q \times dc. \tag{2}$$

This is really a critical point about binned data. It means that we are basically approximating the number c_i to be greater or equal to $c_{\min} + (q-1) \times dc$ and less than $c_{\min} + q \times dc$.

With a bit of algebra, we can rewrite this in a couple of steps to solve for q

$$(q-1) \leq \frac{c_i - c_{\min}}{dc} < q \;\; \Rightarrow \;\; q - 1 = \left\lfloor \frac{c_i - -c_{\min}}{dc} \right\rfloor$$

$$\Rightarrow \;\; q = \left\lfloor \frac{c_i - c_{\min}}{dc} \right\rfloor + 1. \tag{3}$$

Computationally, the floor function $\lfloor\rfloor$ is identical to `int()`, extracting the integer part of a number.

Equation 3 works well for all q values except for $c_{\max}$. This is because in Eq. 1, all the intervals are defined with two relationship symbols, a $\leq$ and then a $<$ *except* for the last one where both relationships are $\leq$ because otherwise the largest value would be excluded. In that case, since $c_{\max}$ strictly satisfies the equality on the *right hand side*, we have to "manually" assign $c_{\max}$ to a bin of our choice. The most logical thing is to assign it to the last bin. If we didn't choose to put $c_{\max}$ in the last bin but in the next one, we would end up with one more bin than what we specified.

By "classifying" every value of c_i into a bin, we can return to using integers in dictionaries to create histograms. Let us write a short function to do this. We assume that what is supplied is a list of values and a number of bins. The output is a histogram and the locations along the vertical axis where each of the frequencies needs to be placed.

```
>>> def histogram(InList,Nbin):
...     xmin=min(InList) # Find the minimum value of InList
...     xmax=max(InList) # Find the maximum value of InList
...     dx=float(xmax-xmin)/Nbin # Find bin widths
...     H={} # Define the histogram
...     for x in InList: # Loop over data
...         if x==xmax: # If x==xmax, store in last bin
...             q=Nbin
...         else: # Else, find bin
...             q=int((x-xmin)/dx)+1
...         H[q]=H.get(q,0)+1 # Populate histogram
...     xcoord={} # Create horizontal coord dictionary
...     for q in H.keys():
...         xcoord[q]=xmin+(q-1)*dx # Using left end of bin.
...         #xcoord[q]=xmin+(q-0.5)*dx # This would be mid-bin.
...     return(xcoord,H)
```

The output of this function can be easily applied to the list `clist`. It now also has the flexibility to be used to create double logarithmic scales. But before we do this, let us check how it compares to `hist` from `matplotlib`.

```
>>> x,h=histogram(clist,40) # make 40 bins
>>> plt.yscale('log') # vertical logarithmic scale
>>> plt.plot(list(x.values()),list(h.values()),'o',markersize
    =10)
>>> plt.hist(clist,40,log=True)
>>> plt.show()
```

The result of this code is Fig. 4.

As a final point, we can now also create a probability distribution from our histograms. Thus,

```
>>> pr={} # Create a distribution
>>> sum=0. # Sum the histogram to normalize it to get pr.
>>> for q in h.keys(): # Go through all bins
...     sum=sum+h[q] # and add up the contents
...
>>> print(sum) # Should give number of nodes
36692.0
>>> E.order() # and it does
36692
>>> for q in h.keys(): # Now create the distribution
...     pr[q]=h[q]/sum
...
>>> plt.yscale('log')
```

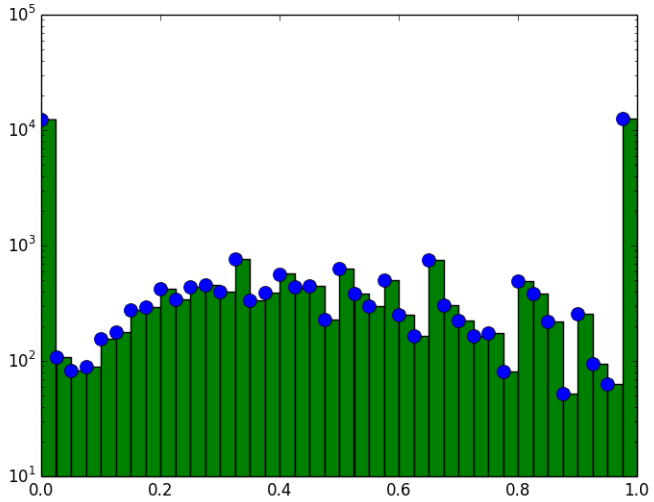

Figure 4: Comparison of our code output and that of `matplotlib`'s `hist`. With our code, we can plot using functions other than `hist` and thus we can control other aspects of the plot.

```
>>> plt.xlabel('clustering c')
>>> plt.ylabel('pr[c]')
>>> plt.plot(list(x.values()),list(pr.values()),'o',markersize
    =10)
>>> plt.show()
```

The result is as shown in Fig. 5

Unfortunately, after this much work, we will have to postpone actually using `histogram` is a broad way because local clustering in the networks we are studying is not too interesting from the standpoint of the distributions. However, we are now ready to use it on some data and develop familiarity.

As a final computational topic helping us analyze data, let us spend a small amount of time making scatter plots. We will only make one plot here, using the Enron dataset to compare degree and triangles attached to a node. The code needed is

```
>>> KTlist=[] # Create a list, although a dictionary could do
    too.
>>> for i in E.nodes():
...     k=E.degree(i)
```

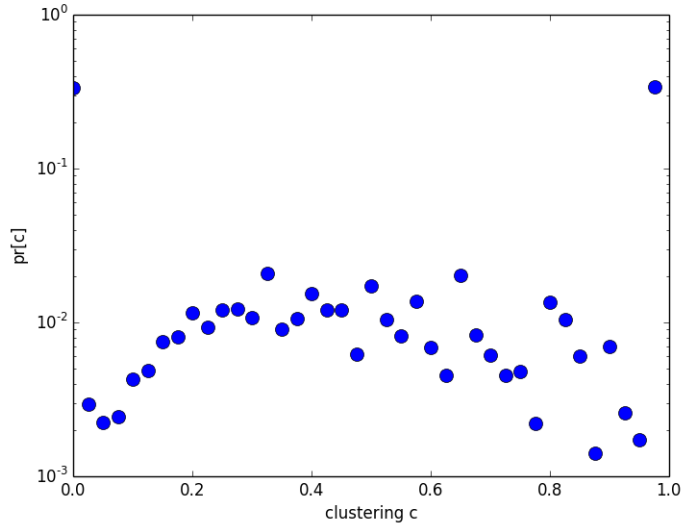

Figure 5: Distribution $\Pr(c)$ versus c for the Enron network.

```
...     t=nx.triangles(E,i)
...     KTlist.append((k,t))
...
>>> plt.xscale('log')
>>> plt.yscale('log')
>>> plt.xlabel('degree k')
>>> plt.ylabel('triangle count t')
>>> plt.plot([x for (x,y) in KTlist],[y for (x,y) in KTlist],'o
    ')
>>> plt.show()
```

This generates Fig. 6.

2 Assignment

Instructions

1. In each problem or assignment, carefully pay attention to what you are allowed to use directly from `networkx` and what you will need to create from scratch.

2. When presenting the assignment, please have your laptop ready with

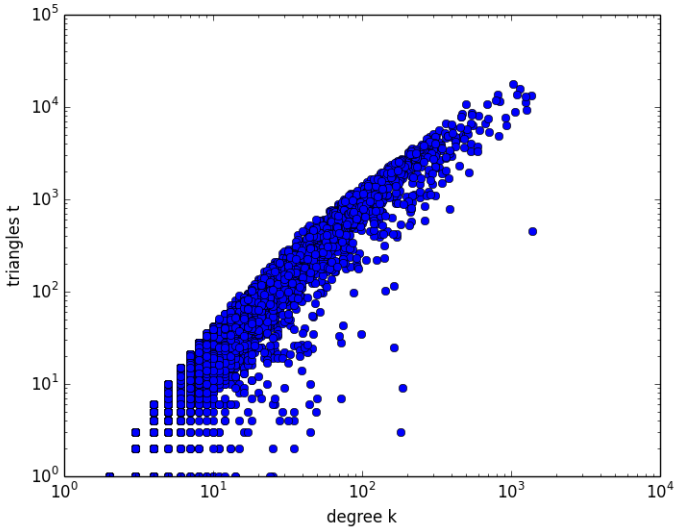

Figure 6: Scatter plot in double logarithmic scale of the degrees and numbers of triangles for each node i of the network.

all the necessary libraries, data, and code loaded, and the interface for `python` ready so that we can see on screen the code for the functions.

3. In general, the code that you create must be such that it can receive *any* new network `G` as input and produce the output needed. If, for instance, the code is supposed to produce the number of links of a network, then it should be able to receive any network (of the right type) as input, and provide as output the number of links.

4. You will quickly notice that some of the work is repetition. This is to encourage you to work through your prior problems and improve your code. But don't worry... new tasks with their own challenges are here too.

Tasks

Now, the work

1. Create a function that takes as input a network of `networkx` type `Graph()` and a node i and returns the number of v-shapes visiting node `i`. You are free to use the `degree` method.

2. Create a function that takes as input a network of `networkx` type `Graph()` and a node i and returns the number of triangles visiting node `i`. You are free to use the `degree` and `neighbors` methods.

3. Create a function that takes as input a network of `networkx` type `Graph()` and a node i and returns the local clustering of `i`.

4. Copy the function from the lecture notes that generates WS networks (note that what I have provided is the $r = 0$ case, i.e. no disorder, which we use in this problem). Taking a network of $n = 100$ and $k = 8$, use the code above to create

   (a) a list `VslistWS` of all the local v-shape counts for each and every node in the network,

   (b) a list `TrilistWS` of all the local triangle counts for each and every node in the network,

   (c) a list `clistWS` of all the clustering coefficients for each and every node in the network,

   (d) using only a `python` dictionary and **not** the `histogram` function above, create a histogram of the values in `VslistWS` and describe what you find,

   (e) using only a `python` dictionary and **not** the `histogram` function above, create a histogram of the values in `TrilistWS` and describe what you find,

   (f) using the `histogram` function above, create a histogram of the values in `clistWS` and describe what you find.

   (g) **(Stop! Do the previous three subitems before this one)** Construct three scatter plots, each one pairing two out of the three quantities k_i, t_i, c_i.

5. Now, increase k in Prob. 4 in increments of 2 until $k = 2\lfloor (n-1)/2 \rfloor$. Plot as a function of k

   (a) the count of v-shapes for a node,

   (b) the count of triangles for a node, and

   (c) value of the local clustering of a node.

   What is the value of k_c? Can you spot it in the plots?

6. Using the instructions provided in the lecture notes create two networks from data, one from the Enron dataset and the other from the condensed matter coauthorship dataset. Download these datasets either from my site (`https://sites.google.com/site/edlopez72/teaching-datasets`) or go to `https://snap.stanford.edu/data/` (caution: I've preprocessed the coauthorship a bit so if you download it from here, you have to check the data for self-loops). You can name the networks as you like. In this assignment, I label the Enron network as `E` and the coauthorship network as `C`.

7. Repeat Prob. 4 for both `E` and `C`, however **do not worry about describing your findings in words**. Instead, plot all histograms using

   (a) linear scales for both horizontal and vertical axes (hint: you don't need to the `plt.xscale('log')` statement),

   (b) logarithmic scales for both horizontal and vertical axes,

   The outcomes of these plots can be compared with the lecture notes for accuracy.

8. Given that m is the number of links in a network of n nodes, write a small equation that provides the number of links that would be changed if a fraction r of the links are modified (such as being rewired) by some algorithm. Hint: it's really simple... not a trick question.

9. **(Optional but useful)** If you are already thinking about the network(s) that will be part of your final project, please repeat Probs. 6 and 7 above.