Web Structure Mining

Introduction

In this practical work, you will learn some basics on network data analysis. In Python, the 3 most popular libraries for network analysis are networkx, igraph (also available in R), and graph-tool. In this practical work, we will illustrate network analysis using the igraph library since it is easy-to-use, offers a broad range of functionalities, and offers good performances on relatively large networks.

Note that this tutorial is guided step by step. Its objective is not showing you all the functionalities of this package but showing the most useful instead.

The network we will use today is the Political blogs network. It provides a representation of the political blogosphere in February 2005. The vertices of a network are political blogs; there is an (directed) edge between from vertex A to vertex B if there is a link to B on the home page of A. The network has n = 1490 vertices and m = 19090 edges. There is no loop in the network.

The documentation of the package is available at: http://igraph.org/python/.

Important note. In this practical work, you are manipulating non native libraries (pandas, seaborn, igraph, and scipy). To import these libraries, download them and install them first. Depending on your Operating System (Windows, Linux, Mac OS X, ...) the installation process differs. Please refer to http://igraph.org/python/#startpy for details. Once installed, you must import the library in your code to get all the functionalities provided by the library. To do so, you must copy and paste the following lines at the beginning of your python file.
In this practical work, you will achieve the following steps:

1. Load an external file and map it as a graph object
2. Plot its degree distribution
3. Check if the power-law distribution holds
4. Check if the small-world hypothesis holds
5. Check if the strong community structure holds

Data loading

First, download the dataset by following this link: www.irit.fr/~Yoann.Pitarch/Docs/M2Stats/WebMining/polblogs.gml. Once you have downloaded the data, they need to be read in python. Since the data is stored in a particular format (GML format), a particular reading function is needed. Please refer to the documentation of the Graph class (http://igraph.org/python/doc/igraph.Graph-class.html) to find the appropriate function. Once loaded, it is possible to access to vertices as follows.

For information related to edges, you can proceed in a similar way using es instead of vs. Try these methods to feel confortable with them.
Plotting degree distribution

Plotting functionalities are always useful to get a better understanding of the data. In this section, we will focus on plotting the degree distribution of the network we are analyzing. We will then check if the distribution follows a power-law distribution as it is the case in almost every complex networks. Since we are considering a directed network, in the following we will consider the degree as the sum of in and out degrees. For doing so, please execute these following steps.

1. Get the degree distribution using the `degree` function;

2. Transform the obtained list into a pandas series using the `Series` constructor (see [http://pandas.pydata.org/pandas-docs/stable/generated/pandas.Series.html](http://pandas.pydata.org/pandas-docs/stable/generated/pandas.Series.html)). The pandas library offers very convenient functionalities, e.g., discretization, sampling, or percentile calculation, in manipulating series (lists) and data frame (matrices);

3. Use the `distplot` function of the seaborn library to plot the degree distribution. You must specify the data, use 20 bins, add a label, and do not plot the gaussian kernel density estimate (see [http://seaborn.pydata.org/generated/seaborn.distplot.html#seaborn.distplot](http://seaborn.pydata.org/generated/seaborn.distplot.html#seaborn.distplot)).

4. Display the plot with the command `sns.plt.show()`;

5. Modify the call to the function `distplot` to check if the distribution follows a power-law distribution.

Repeat the same methodology with in-degree and out-degree separately.

Small-world hypothesis

In this section, we will check if the small-world hypothesis holds on this dataset. To do so, we will calculate the shortest path length between pairs of vertices and average these lengths. Of course, it is not realistic to calculate the shortest path for every pairs of vertices. We will thus consider only a subset of vertices. This subset contains 200 randomly chosen vertices. The sampling can be performed with the `sample` function of the series object. It is then needed to iterate over all possible pairs and calculate the shortest paths. Iterating over all possible pairs can be done using a nested loop whereas calculating the shortest path
length can be performed by the `shortest_paths_dijkstra` function associated to the graph instance (see `http://igraph.org/python/doc/igraph.Graph-class.html#shortest_paths_dijkstra`).

**Instructions.** Calculate the average shortest path length of 200 randomly chosen vertices. In case the shortest path equals `math.inf` you will consider that the shortest path length equals 10.

**Strong community structure**

The third characteristic of complex networks is that they exhibit a strong community structure. A typical methodology for checking if this characteristic is to calculate the average local clustering coefficient as seen in the lecture. This average local clustering coefficient is then compared to the average local clustering coefficient of a random having the same numbers of vertices and edges.

**Instructions.** Apply the above mentioned strategy to check if the strong community structure property holds on this network.

**Tips.** Here are some practical information to help you to reach your objective:

- A random graph can be generated using the `Erdos_Renyi` function (see `http://igraph.org/python/doc/igraph.GraphBase-class.html#Erdos_Renyi`);

- The average local clustering coefficient can be calculated with the function `transitivity_avglocal_undirected` (see `http://igraph.org/python/doc/igraph.Graph-class.html#transitivity_avglocal_undirected`). Note that this function is dedicated to undirected graph. However, it is of little importance since both graph (the original one and the random one) are both considered as undirected.