Distancia_e_Correlacoes_Questionario_BrunoFBessa_5881890

May 15, 2021

0.1 SME0130 - Redes Complexas

Structure of networks: Distance and Correlation

Professor: Francisco Aparecido Rodrigues, francisco@icmc.usp.br. Estudante: Bruno F. Bessa (num. 5881890), bruno.fernandes.oliveira@usp.br Universidade de São Paulo, São Carlos, Brasil.

[1]:
```
import numpy as np
import networkx as nx
import matplotlib.pyplot as plt
import math
from scipy import stats
```

[2]:
```
def get_graph_from_data_file(file_name='lesmis.txt', ncols=3):

    '''
    Defines a NetworkX graph based on data from file.
    Plots a visual representation of the graph
    '''

    file_path = 'data/' + file_name

    if ncols == 2:
        G = nx.read_edgelist("data/powergrid.txt", nodetype=int)
    else:
        G = nx.read_edgelist(file_path, nodetype=int, data=(('weight', float),))

    pos = nx.spring_layout(G)
    nx.draw(G, pos, node_color='b', node_size=50, with_labels=False)

    G = G.to_undirected()
    G.remove_edges_from(nx.selfloop_edges(G))
    Gcc = sorted(nx.connected_components(G), key=len, reverse=True)
    G = G.subgraph(Gcc[0])
    G = nx.convert_node_labels_to_integers(G, first_label=0)

    return G
```

[3]:
```
# Distance
def graph_aspl(G):
    if nx.is_connected(G) == True:
        l = nx.average_shortest_path_length(G)
        return l
    else:
        return None

def spl_distribution(G):
    N = len(G)
    if nx.is_connected(G) == True:
        D = np.zeros(shape=(N,N)) # D is the matrix of distances
        vl = []
        for i in np.arange(0,N):
            for j in np.arange(i+1, N):
                if(i != j):
                    aux = nx.shortest_path(G,i,j)
                    dij = len(aux)-1
                    D[i][j] = dij
                    D[j][i] = dij
                    vl.append(dij)
    return vl

def get_slp_dist_prob(G):
    vk = np.array(spl_distribution(G))
    maxk = np.max(vk)
    mink = np.min(vk)
    kvalues= np.arange(0,maxk+1) # possible values of k
    Pk = np.zeros(maxk+1) # P(k)

    for k in vk:
        Pk[k] = Pk[k] + 1
    Pk = Pk/sum(Pk) # the sum of the elements of P(k) must to be equal to one
    return kvalues,Pk

def get_spl_distribution_plot(G):
    N = len(G)
    if nx.is_connected(G) == True:
        D = np.zeros(shape=(N,N)) # D is the matrix of distances
        vl = spl_distribution(G)
        d = nx.diameter(G)
        x = range(0,d+1)
        plt.hist(vl, bins = x, density=True)

        avg_spl = np.mean(vl)
        var_spl = np.var(vl)
```

```
        plt.title("Distribution of the geodesic distances (mean: {:.2f},␣
↪variance: {:.2f})".format(avg_spl, var_spl), fontsize=20)
        plt.ylabel("P(l)", fontsize=15)
        plt.xlabel("Shortest path length (l)", fontsize=15)
        #plt.grid(True)
        plt.savefig('av_short_path.eps')
        plt.show(True)
    else:
        None

def pearson_correlation(G):
    ki = []
    kj = []
    for i in range(0,len(G.nodes())):
        for j in range(0, len(G.nodes())):
            if(G.has_edge(i,j) == True):
                ki.append(G.degree(i))
                kj.append(G.degree(j))

    from scipy.stats import pearsonr
    # calculate Pearson's correlation
    corr, _ = pearsonr(ki, kj)
    return corr

def graph_knn(G):
    knn = []
    for i in G.nodes():
        aux =  nx.average_neighbor_degree(G, nodes = [i])
        knn.append(float(aux[i]))
    knn = np.array(knn)
    return knn


def get_graph_k_knn_corr(G):
    vk = dict(G.degree())
    vk = list(vk.values())
    knnk = list()
    ks = list()
    knn = graph_knn(G)
    for k in np.arange(np.min(vk), np.max(vk)+1):
        aux = vk == k
        if(len(knn[aux]) > 0):
            av_knn = np.mean(knn[aux]) #average clustering among all the nodes␣
↪with degree k
            knnk.append(av_knn)
            ks.append(k)
    fig= plt.figure(figsize=(10,6))
```

```
    plt.plot(ks, knnk, '-o', color='gray',markersize=10, linewidth=0,
             markerfacecolor='lightgray',
             markeredgecolor='black',
             markeredgewidth=2)

    corr = np.corrcoef(ks, knnk)[0,1]
    s = stats.spearmanr(ks, knnk)
    spearman_corr = s[0]
    spearman_pval = s[1]

    plt.title("Corr. plot for k and knn (corr.: {:.2f}, spearman: {:.2f}, p-val:
↪ {:.3f})".format(corr,
↪                 spearman_corr,
↪                 spearman_pval), fontsize=20)
    plt.ylabel("knn(k)", fontsize=15)
    plt.xlabel("k", fontsize=15)



    #plt.loglog(ks,knnk,'bo',basex=10,basey=10)
    #plt.title("Average neighborhood connectivity vs degree")
    plt.ylabel("knn(k)", fontsize = 20)
    plt.xlabel("k", fontsize = 20)
    #plt.savefig('knnk.eps')

    # determine best fit line
    par = np.polyfit(ks, knnk, 1, full=True)
    slope=par[0][0]
    intercept=par[0][1]
    xl = [min(ks), max(ks)]
    yl = [slope*xx + intercept  for xx in xl]
    plt.plot(xl, yl, '--', linewidth=3, color='black')
    plt.xticks(fontsize=15)
    plt.yticks(fontsize=15)
    plt.savefig('knn.eps') #save the figure into a file
    plt.show(True)

def shannon_entropy(G):
    k,Pk = get_slp_dist_prob(G)
    H = 0
    for p in Pk:
        if(p > 0):
            H = H - p*math.log(p, 2)
    return H
```

0.2 Questions

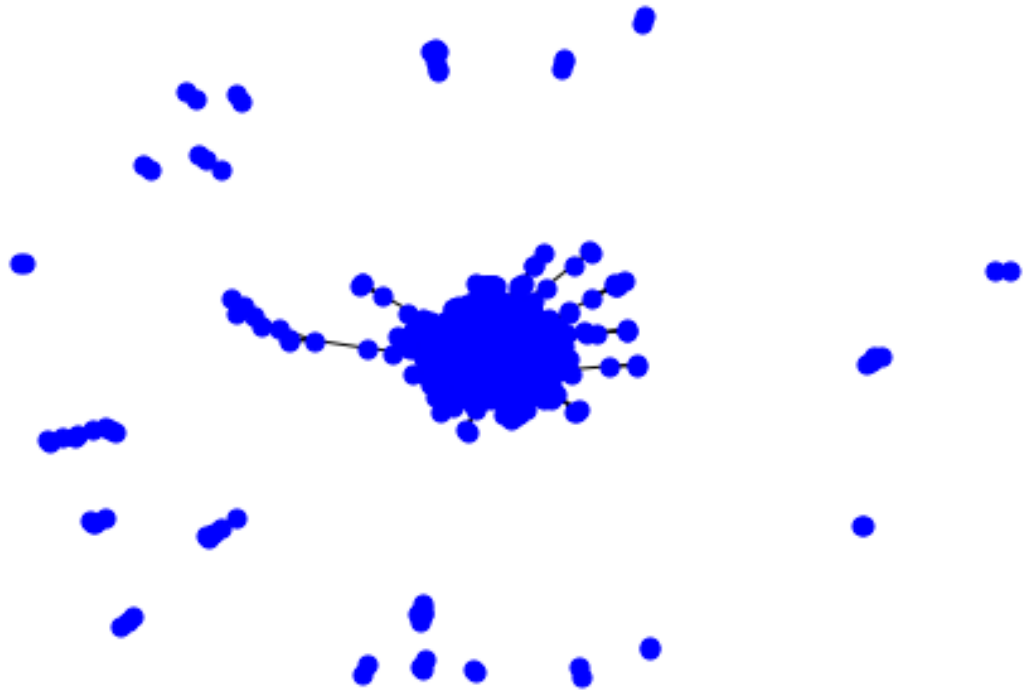
1 - Para a rede “Hamsterster”, calcule a média dos menores caminhos e o diâmetro. Use apenas o maior componente da rede e remova ciclos ou auto-conexões.

Usaremos as funções graph_aspl e nx.diameter para obter os valores desejados.

[4]:
```
G = get_graph_from_data_file('hamsterster.txt')

print('Média dos menores caminhos: {:.2f}'.format(graph_aspl(G)))
print('Diâmetro da rede: {:.2f}'.format(nx.diameter(G)))
```

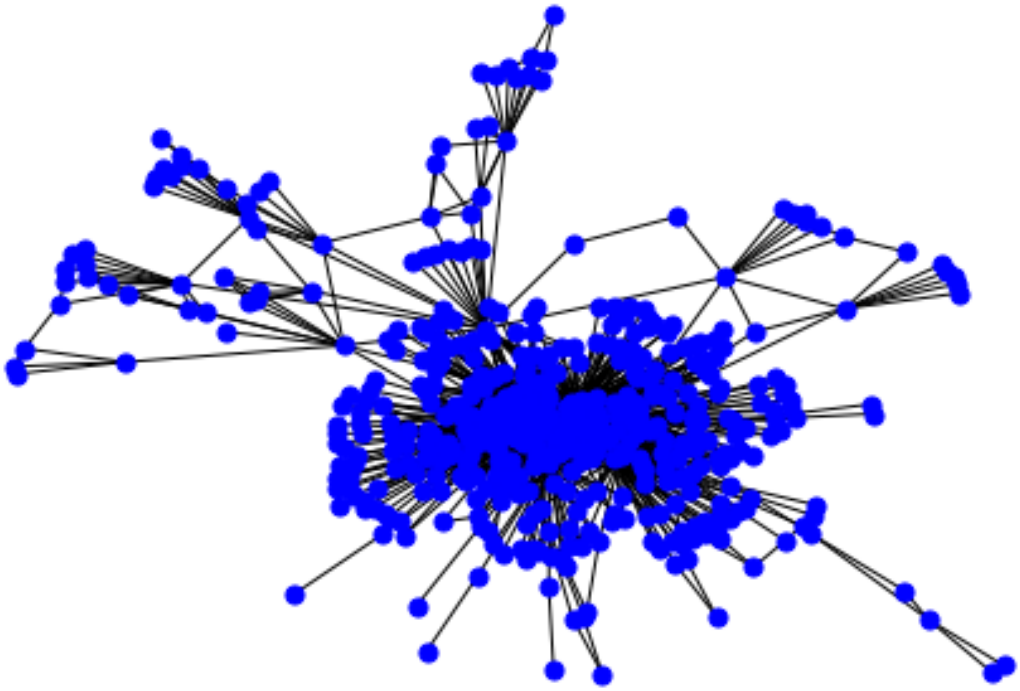
```
Média dos menores caminhos: 3.45
Diâmetro da rede: 14.00
```



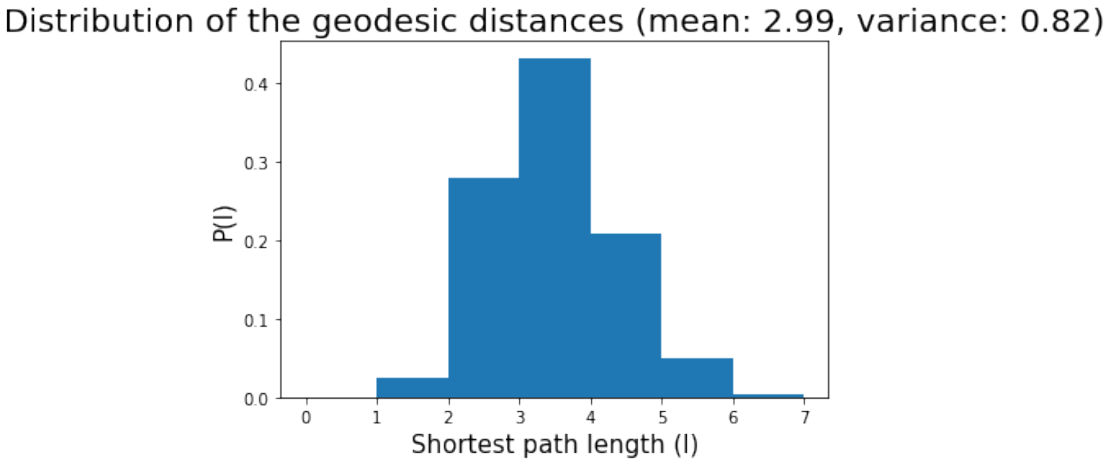
2 - Considere a rede “USairport500” e calcule a média e variância dos menores caminhos. Use apenas o maior componente da rede e remova ciclos ou auto-conexões.

Usaremos para a solução as informações de distribuição dos menores caminhos implementada na função “get_spl_distribution”.

[5]:
```
G = get_graph_from_data_file('USairport500.txt')
```



[6]:
```
get_spl_distribution_plot(G)
```



3 - Para a rede “USairport500”, calcule a entropia de Shannon da distribuição dos menores caminhos. Use logaritmo na base 2 e considere apenas o maior componente da rede.

A entropia de Shannon, dada por

$$H = -\sum_{k=0}^{\infty} P(k) log P(K)$$

é uma medida da previsibilidade de uma medida. Está implementada na função "shannon_entropy" e neste caso usamos a distribuição dos menores caminhos em vez do grau.

[7]:
```
print('A entropia de Shannon da distribuição dos menores caminhos da rede␣
↪USairport500: {:.4f}'.format(shannon_entropy(G)))
```

```
A entropia de Shannon da distribuição dos menores caminhos da rede USairport500:
1.8837
```

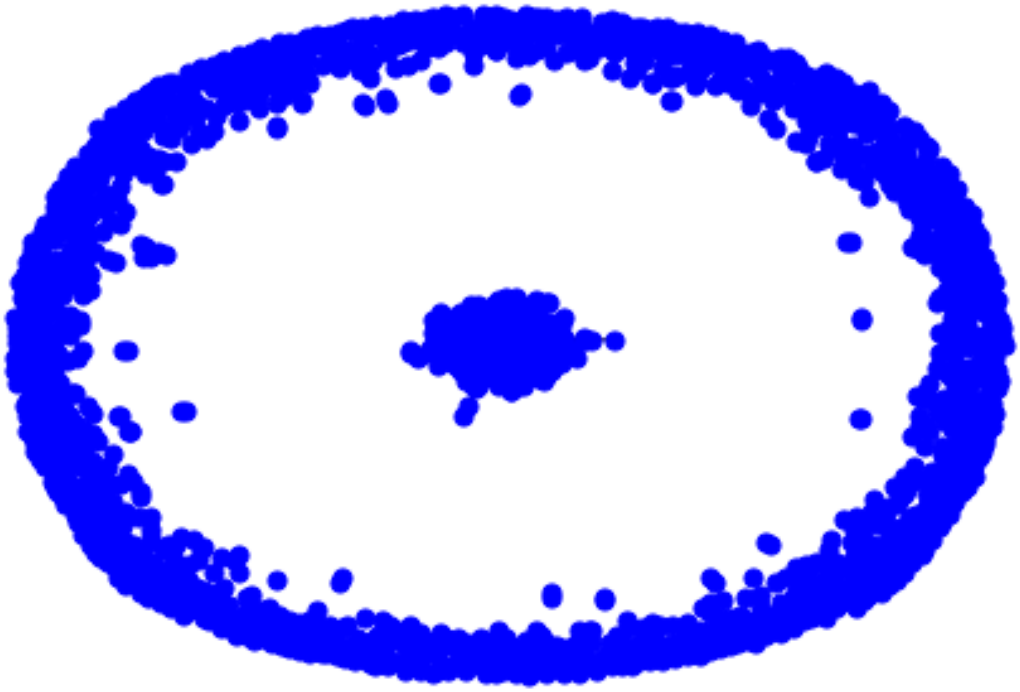
4 - Calcule o coeficiente de assortatividade da rede Advogato. Considere apenas o maior componente.

O coeficiente de assortatividade de uma rede mede a correlação entre os graus de seus nós. Está implementado na função "degree_assortativity_coefficient" da biblioteca NetworkX.

[8]:
```
G = get_graph_from_data_file('advogato.txt')

print("Coeficiente de assortatividade da rede Advogato: {:.4}".format(nx.
↪degree_assortativity_coefficient(G)))
```

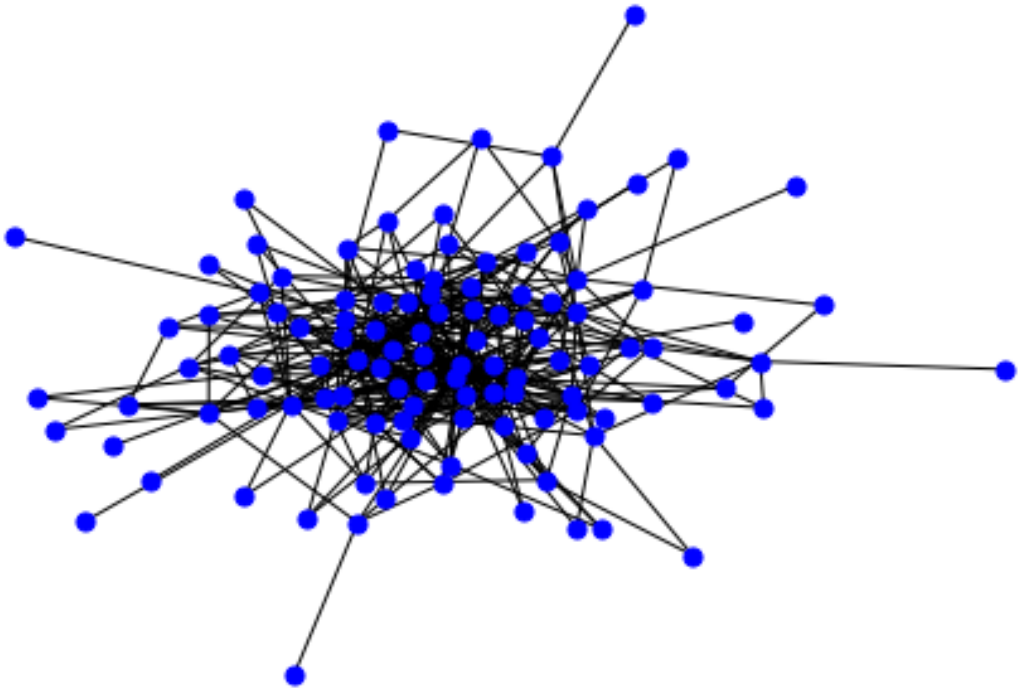
```
Coeficiente de assortatividade da rede Advogato: -0.09572
```



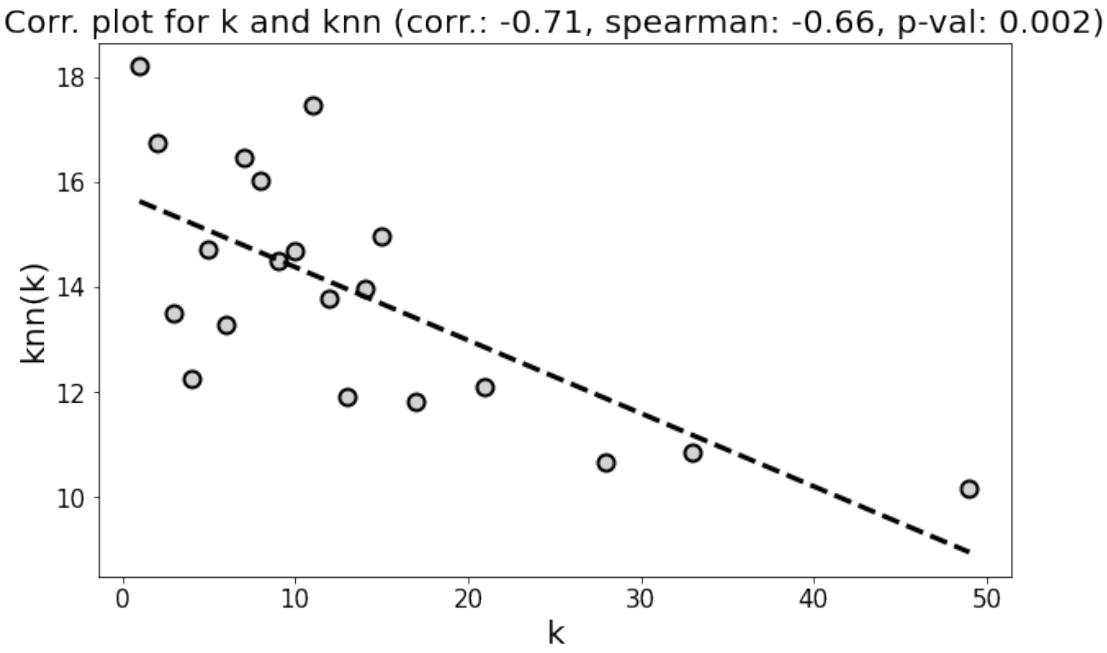
5 - Calcule o coeficiente de correlação de Pearson entre o grau médio dos vizinhos e o grau de cada vértice para a rede "word_adjacencies". Isto é, entre k e knn(k). Use apenas o maior componente. Considere o exemplo da aula.

Calcularemos utilizando a função "get_graph_k_knn_corr" definida acima.

[9]:
```
G = get_graph_from_data_file('word_adjacencies.txt')
```



[10]:
```
get_graph_k_knn_corr(G)
```


Corr. plot for k and knn (corr.: -0.71, spearman: -0.66, p-val: 0.002)
knn(k)
k
18
16
14
12
10
0
10
20
30
40
50