BrunoFBessa_5881890_P1_codigo

May 23, 2021

0.1 SFI5904 - Redes Complexas

Projeto Prático 1: Modelo de redes de Erdos-Renyi Primeiro Semestre de 2021

Docente: Luciano da Fontoura Costa (luciano@ifsc.usp.br) Estudante: Bruno F. Bessa (num. 5881890, bruno.fernandes.oliveira@usp.br) Universidade de São Paulo, São Carlos, Brasil.

Escopo do projeto:

Implementar redes de Erdos-Renyi a partir de um determinado número de nós N e um grau médio desejado (a partir do qual se obtém a probabilidade de conexão p).

Visualizar algumas das redes geradas. Apresentar: - os histogramas de frequência relativa dos graus, - coeficientes de aglomeração e distâncias mínimas, identificando nas respectivas legendas a média e o desvio padrão.

[62]:
```
# Importação de bibliotecas necessárias para o processamento e visualização

import random
import numpy as np
import scipy
import math
import networkx as nx
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_theme()
from IPython import display
```

[60]:
```
# Definição das redes e rotinas para cálculo de métricas

class Point():

    """
    Define os pontos (nodos) do grafo.
    A rede de Erdos-Renyi será formada por um conjunto de objetos desta classe.
    """

    def __init__(self, index: str) -> None:
        self.index = index
        self.neighbors = []
```

```
    def __repr__(self) -> None:
        return repr(self.index)


def erdos_renyi(N: int,
                p: float,
                plot: bool = True,
                file_name: str = None) -> nx.classes.graph.Graph:

    """
    Define as conexões (i,j) = (j,i) para todos os pares de pontos com base em␣
↪um
    evento medida aleatória para probabilidade p, recebida como parâmetro na␣
↪construção da rede.
    """

    G = nx.Graph()
    nodes = [Point(i) for i in range(N)]
    edges = [(i, j) for i in range (N) for j in range(i) if random.random() < p]

    # Configura e adiciona arestas (edges) na rede
    for (i, j) in edges:
        nodes[i].neighbors.append(nodes[j])
        nodes[j].neighbors.append(nodes[i])
    for edge in list(edges):
        G.add_edge(list(edge)[0], list(edge)[1])

    # Para calcularmos medidas de distância precisaremos remover nós não␣
↪conectados
    # No trecho abaixo mantemos somente o maior componente conctado da rede.
    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)

    # Opção de visualização da rede gerada (não utilizar para séries grandes de␣
↪experimentos)
    if plot:
        pos = nx.spring_layout(G)
        fig_net = nx.draw(G, pos, node_color='w', node_size=1,␣
↪with_labels=False)
        plt.suptitle("Erdos-Renyi Network (N={}, p={})".format(N, p),␣
↪fontsize=15)
        plt.show(fig_net)
```

```
    if file_name != None:
        pos = nx.spring_layout(G)
        fig_net = nx.draw(G, pos, node_color='w', node_size=1,␣
↪with_labels=False)
        plt.suptitle("Erdos-Renyi Network (N={}, p={})".format(N, p),␣
↪fontsize=15)
        plt.savefig("images/"+file_name)
        plt.close(fig_net)

    return G

def degree_distribution(G: nx.classes.graph.Graph) -> list:

    """
    Retorna a lista de valores de grau (k) para todos os nós da rede.
    """

    dict_degree = dict(G.degree())
    list_k = []
    for node, k_value in dict_degree.items():
        list_k.append(k_value)

    return list_k

def clustering_coef_distribution(G: nx.classes.graph.Graph) ->list:

    """
    Retorna a lista de valores de cluster coefficient (cc) para todos os nós da␣
↪rede.

    """

    list_cc_nodes = []
    for node in G.nodes():
        list_cc_nodes.append(nx.clustering(G, node))

    return list_cc_nodes

def spl_distribution(G: nx.classes.graph.Graph) ->list:
    """
    Retorna a lista de valores de shortest path length (spl) para todos os nós␣
↪da rede.

    """

    N = len(G)
    if nx.is_connected(G) == True:
```

```
        distance_matrix = np.zeros(shape=(N,N))
        diameter = nx.diameter(G)
        slp_values = []
        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
                    distance_matrix[i][j] = dij
                    distance_matrix[j][i] = dij
                    slp_values.append(dij)
        return slp_values
    else:
        pass

def shannon_entropy(G: nx.classes.graph.Graph) ->float:

    """
    Calcula a entropia de Shannon para um grafo G recebido como parâmetro.
    """
    list_k = degree_distribution(G)
    min_k = np.min(list_k)
    max_k = np.max(list_k)

    k_values= np.arange(0,max_k+1)
    k_prob = np.zeros(max_k+1)
    for k in list_k:
        k_prob[k] = k_prob[k] + 1
    k_prob = k_prob/sum(k_prob)

    H = 0
    for p in k_prob:
        if(p > 0):
            H = H - p*math.log(p, 2)
    return H

def distribution_plot(list_values: list,
                      plot_title: str = "Histograma de densidade",
                      var_name: str = "Variável",
                      file_name: str = None) -> None:
    """
    Produz histgrama de uma medida recebida na forma de lista.
    """

    avg_value = np.mean(list_values)
    var_value = np.var(list_values)
```

```
    fig, ax = plt.subplots()
    n, bins, patches = ax.hist(list_values, density=True)
    ax.set_xlabel(var_name)
    ax.set_ylabel("Densidade de probabilidade")
    ax.set_title("{} de {}: média={:.2f}, var={:.2f}".format(plot_title,
                                                            var_name,
                                                            avg_value,
                                                            var_value),
                                                            fontsize=15)
    plt.show(True)
    if file_name != None:
        fig.savefig("images/"+file_name)

def correlation_plot(x: list,
                         y: list,
                         x_label: str = "x",
                         y_label: str = "y",
                         file_name: str = None) -> None:
    """
    Produz gráfico de dispersão de duas variáveis x e y recebidas na forma de␣
 ↪listas.
    Calcula correlação de Pearson e Spearman para x e y.
    """

    pearson_corr = np.corrcoef(x, y)[0,1]
    spearman_corr, spearman_pval = scipy.stats.spearmanr(x, y)

    fig, ax = plt.subplots()
    ax.scatter(x, y)
    ax.set_xlabel(x_label)
    ax.set_ylabel(y_label)
    ax.set_title("Dispersão de {} e {}: Pearson: {:.2f}, Spearman: {:.2f}␣
 ↪(p-val: {:.3f})".format(x_label,
                                                                                ␣
 ↪           y_label,
                                                                                ␣
 ↪           pearson_corr,
                                                                                ␣
 ↪           spearman_corr,
                                                                                ␣
 ↪           spearman_pval),
                                                                                ␣
 ↪           fontsize=15)
    plt.show(True)
    if file_name != None:
        fig.savefig("images/"+file_name)
```

```
def simple_plot2d(x: list,
                  y: list,
                  x_label: str = "x",
                  y_label: str = "y",
                  file_name: str = None) -> None:
    """
    Produz gráfico simples com associação entre suas variáveis x e y recebidas␣
 ↪na forma de listas.
    """
    fig, ax = plt.subplots()
    ax.plot(x, y)
    ax.set_xlabel(x_label)
    ax.set_ylabel(y_label)
    ax.set_title("Dispersão de {} e {}".format(x_label, y_label, fontsize=15))
    plt.show(True)
    if file_name != None:
        fig.savefig("images/"+file_name)
```

Vamos agora reproduzir alguns experimentos com as redes de ER. Para conseguirmos visualizar as conexões da rede, usaremos diferentes valores de número de nós (10 50 100, 500). Para cada valor de N vamos variar as propabilidades de conexão p e observar o efeito sobre o grau médio.

[ ]:
```
list_N = [10, 50, 100, 500]
list_p = [0.05, 0.25, 0.5, 0.75, 1]
network_model = "erdosrenyi"

for _n in list_N:
    for _p in list_p:
        G = erdos_renyi(_n, _p, True, "graph_{}_n{}_p{}.jpg".
 ↪format(network_model, str(_n), str(_p)))
        degree_dist = degree_distribution(G)
        cluster_dist = clustering_coef_distribution(G)
        spl_dist = spl_distribution(G)
        distribution_plot(degree_dist, var_name="Grau",␣
 ↪file_name="degree_dist_{}_n{}_p{}.jpg".format(network_model, str(_n),␣
 ↪str(_p)))
        distribution_plot(cluster_dist, var_name="Cluster␣
 ↪Coefficient",file_name="cc_dist_{}_n{}_p{}.jpg".format(network_model,␣
 ↪str(_n), str(_p)))
        distribution_plot(spl_dist, var_name="SPL",␣
 ↪file_name="spl_dist_{}_n{}_p{}.jpg".format(network_model, str(_n), str(_p)))
```

Vemos que a rede de Erdos-Renyi é muito fortemente caracterizada pela pobabilidade de conexão dos nós. Fixado N = 500, vamos obsevar como variam as medidas médias da rede em função de p.

[ ]:
```
list_N = [500]
list_p = [0.05, 0.25, 0.5, 0.75, 1]
list_mean_k = []
list_mean_cc = []
list_mean_spl = []
list_shannon_entropy = []
network_model = "erdosrenyi"

for _n in list_N:
    for _p in list_p:
        G = erdos_renyi(_n, _p, False)
        degree_dist = degree_distribution(G)
        cluster_dist = clustering_coef_distribution(G)
        spl_dist = spl_distribution(G)
        list_mean_k.append(np.mean(degree_dist))
        list_mean_cc.append(np.mean(cluster_dist))
        list_mean_spl.append(np.mean(spl_dist))
        list_shannon_entropy.append(shannon_entropy(G))

simple_plot2d(list_p, list_mean_k, "p", "Grau médio", "plot2d_p_mean_k_{}_n500.
 ↪jpg".format(network_model))
simple_plot2d(list_p, list_mean_cc, "p", "Cluster Coef.",␣
 ↪"plot2d_p_mean_cc_{}_n500.jpg".format(network_model))
simple_plot2d(list_p, list_mean_spl, "p", "SPL médio",␣
 ↪"plot2d_p_mean_spl_{}_n500.jpg".format(network_model))
simple_plot2d(list_p, list_shannon_entropy, "p", "Entropia de Shannon",␣
 ↪"plot2d_p_shannon_{}_n500.jpg".format(network_model))
```

[ ]:
```

```