SME0130 - Redes Complexas

Projeto Prático: Classificação de redes

Professor: Francisco Aparecido Rodrigues, francisco@icmc.usp.br (mailto:francisco@icmc.usp.br).
Estudantes:

- Bruno F. Bessa (Nº USP. 5881890), bruno.fernandes.oliveira@usp.br (mailto:bruno.fernandes.oliveira@usp.br)
- Daniel Torkomian Joaquim (Nº USP. 12420860), danieltorkjoaquim@ifsc.usp.br (mailto:danieltorkjoaquim@ifsc.usp.br)

Universidade de São Paulo, São Carlos, Brasil.

Selecione 3 redes biológicas, 3 redes sociais e 3 redes tecnológicas desses endereços:

- https://networks.skewed.de (https://networks.skewed.de)
- http://konect.cc/networks/ (http://konect.cc/networks/)
- https://icon.colorado.edu/ (https://icon.colorado.edu/)

Faça a classificação das redes usando os modelos e medidas que aprendemos na aula. Não se esqueça de selecionar o mesmo N e grau médio que a rede original na construção dos modelos.

Verifique qual o modelo mais adequado para cada rede.

Hipótese: redes do mesmo tipo seguem o mesmo modelo.

Verifique se essa hipótese é verdadeira. Bônus (não é obrigatório): identifique as principais diferenças entre os modelos. Isto é, quais medidas mais contribuem para que uma rede seja classificada como sendo do modelo BA.

Metodologia

Para o desenvolvimento do projeto prático selecionou-se as redes abaixo conforme a indicação:

Social:

- Slavko: http://wwwlovre.appspot.com/support.jsp (http://wwwlovre.appspot.com/support.jsp)
- Physicians: http://konect.cc/networks/moreno_innovation/ (http://konect.cc/networks/moreno_innovation/)
- Jazz Collab: https://networks.skewed.de/net/jazz_collab (https://networks.skewed.de/net/jazz_collab)

Biological:

- malaria: https://github.com/dblarremore/data_malaria_PLOSCompBiology_2013/blob/master/HVR_2.txt (https://github.com/dblarremore/data_malaria_PLOSCompBiology_2013/blob/master/HVR_2.txt)
- BPmaps: http://interactome.dfci.harvard.edu/C_elegans/index.php?page=download (http://interactome.dfci.harvard.edu/C_elegans/index.php?page=download)
- Genetic: http://interactome.dfci.harvard.edu/C_elegans/index.php?page=download (http://interactome.dfci.harvard.edu/C_elegans/index.php?page=download)

Technological:

- Unicode: http://konect.cc/networks/unicodelang/ (http://konect.cc/networks/unicodelang/)

- Guava: http://wwwlovre.appspot.com/support.jsp (http://wwwlovre.appspot.com/support.jsp)
- Air Traffic: http://konect.cc/networks/maayan-faa/ (http://konect.cc/networks/maayan-faa/)

Para cada uma destas redes treinou-se o modelo de predição (Knn e LDA) com uma coleção de redes sintéticas dos seguintes modelos:

- Erdos-Renyi
- Watts-Strogatz
- Barabási-Albert
- Waxman
- Geométrica

sendo o grau médio e o número de nós da rede a ser classificada um parâmetro passado para o ajuste na criação das redes sintéticas.

A função **measures** calcula para uma rede as seguintes medidas que serão usadas nos classificadores, por exemplo, para o cálculo das distâncias no classificador KNN:

- k1: primeiro momento do grau
- k2: segundo momento do grau
- variance: variância do grau
- av_cl: coeficiente de clustering médio
- l: caminho mínimo médio
- r: coeficiente de assortatividade
- Cx: coeficiente de complexidade
- CC: coeficiente de transitividade
- av_EC: eigenvector-ccentrality médio
- Mod: modularidade
- H: entropia de Shannon do grau
- D: Diâmetro

Após a carga das redes (em arquivos de texto) pela função **get_graph_from_data_file**, utilizamos as funções **train_model** e **test_model** (a primeira é invocada por meio da segunda) para geração do algoritmos classificadores KNN e LDA e produzir a classificação desejada.

Como em alguns casos os classificadores apresentaram resultados diferentes, fez-se também o plot utilizando-se de PCA para ter-se uma melor "visualização e ideia". Nesses plots, a rede a ser classificada foi representada pela bolinha rosa.

Os resultados foram os seguintes:

Redes sociais:

- Slavko: Geométrica (KNN) e Watts-Strogatz (LDA)
- Physicians: Erdos-Renyi (KNN e LDA)
- Jazz Collab: Barabási-Albert (KNN e LDA)

Biological:

- malaria: Geométrica (KNN e LDA)
- BPmaps: Barabási-Albert (KNN e LDA)
- Genetic: Barabási-Albert (KNN) e Erdos-Renyi (LDA)

Technological:

- Unicode: Watts-Strogatz (KNN e LDA)

- Guava: Watts-Strogatz (KNN) e Erdos-Renyi (LDA)
- Air Traffic: Barabási-Albert (kNN) e Erdos-Renyi (LDA)

Assim, embora a pequena quantidade de amostras, é possível dizer que não verificou-se a hipótese de que redes do memso tipo seguem a mesma topologia, já que mesmo entre poucas redes verificou-se a classificação em tipos diferentes. Apesar de pertencerem ao mesmo tipo, muitas vezes as redes divergem muito entre sí no que diz respeito as características topológicas, inclusive no própio tamanho, em número de nós e arestas. Vale ressaltar que aqui foi veita uma classificaão simples, para maior precisão, seria interessante trabalhar mais tanto nos modelos para treino quanto nos modelos de predição, tanto que para alguns casos os resultados divergiram entre os modelos de predição.

Código

Os códigos utilizados para a geração dos resultados constam abaixo.

In [4]:

```
import networkx as nx
import matplotlib.pyplot as plt
import numpy as np
from networkx.algorithms.community import greedy_modularity_communities
import math
from scipy.stats import pearsonr
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
```

In [5]:

```
def measures(G):
    def momment_of_degree_distribution(G,m):
        M = 0
        N = len(G)
        for i in G.nodes:
            M = M + G.degree(i)**m
        M = M/N
        return M

    N = len(G)
    M = G.number_of_edges()
    vk = dict(G.degree())
    vk = list(vk.values())
    vk = np.array(vk)


    def degree_distribution_Pk(G):
        maxk = np.max(vk) #valor máximo
        mink = np.min(vk) #valor mínimo
        kvalues= np.arange(0,maxk+1) # valores possíveis de k
        Pk = np.zeros(maxk+1) # P(k)
        for k in vk:
            Pk[k] = Pk[k] + 1
        Pk = Pk/sum(Pk) # Normalizando
        return kvalues,Pk

    k, Pk = degree_distribution_Pk(G)

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


    c = list(greedy_modularity_communities(G))
    communities = np.zeros(len(G.nodes()))
    nc = 0
    for k in range(0,len(c)):
        communities[sorted(c[k])]=nc
        nc = nc+1

    def modularity(G, c):
        A = nx.adjacency_matrix(G)
        N = len(G)
        Q = 0
        for i in np.arange(0,N):
            ki = len(list(G.neighbors(i)))
            for j in np.arange(0,N):
                if(c[i]==c[j]):
                    kj = len(list(G.neighbors(j)))
                    Q = Q + A[i,j]-(ki*kj)/(2*M)
        Q = Q/(2*M)
        return Q

    k1 = momment_of_degree_distribution(G,1)
    k2 = momment_of_degree_distribution(G,2)
```

```
    variance = momment_of_degree_distribution(G,2) - momment_of_degree_distribution(
    av_cl = nx.average_clustering(G)
    l = nx.average_shortest_path_length(G)
    r=nx.degree_assortativity_coefficient(G)
    Cx = momment_of_degree_distribution(G,2)/momment_of_degree_distribution(G,1)
    CC = (nx.transitivity(G))
    D = nx.diameter(G)

    #corr, _ = pearsonr(ks, knnk)

    EC = dict(nx.eigenvector_centrality(G, max_iter = 10000))
    EC = list(EC.values())
    av_EC = np.mean(EC)

    Mod = modularity(G,communities)

    H = shannon_entropy(G)


    return  k1,k2,variance,av_cl,l,r,Cx,CC,D,av_EC,Mod,H
```

In [6]:

```
def get_graph_from_data_file(file_name=None, plot=False):

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

    file_path = file_name

    G = nx.read_edgelist(file_path, nodetype=None, data=(("weight", float),))


    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)

    N = len(G)
    M = G.number_of_edges()

    print('Number of nodes:', N)
    print('Grau médio:', 2*(M/N))

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

    return G
```

In [7]:

```
def train_model(G, n_nets):

    X = []
    y = []

    avg_k = measures(G)[0]
    N = len(G)

    #ER networks

    av_degree = avg_k
    p = av_degree/(N-1)
    for i in range(0,n_nets):
        GER = nx.gnp_random_graph(N, p, seed=None, directed=False)
        Gcc = sorted(nx.connected_components(GER), key=len, reverse=True)
        GER = GER.subgraph(Gcc[0])
        GER = nx.convert_node_labels_to_integers(GER, first_label=0)
        k1,k2,variance,av_cl,l,r,Cx,CC,D,av_EC,Mod,H = measures(GER)
        x = [k1,k2,variance,av_cl,l,r,Cx,CC,D,av_EC,Mod,H]
        X.append(x)
        y.append(0.0)

    #WS networks (p = 0.1)
    k = int(av_degree)
    p = 0.1 #probability of rewiring
    for i in range(0,n_nets):
        GWS1 = nx.watts_strogatz_graph(N, k, p, seed=None)
        Gcc = sorted(nx.connected_components(GWS1), key=len, reverse=True)
        GWS1 = GWS1.subgraph(Gcc[0])
        GWS1 = nx.convert_node_labels_to_integers(GWS1, first_label=0)
        k1,k2,variance,av_cl,l,r,Cx,CC,D,av_EC,Mod,H = measures(GWS1)
        x = [k1,k2,variance,av_cl,l,r,Cx,CC,D,av_EC,Mod,H]
        X.append(x)
        y.append(1.0)

    # BA networks
    m = int(av_degree/2)
    for i in range(0,n_nets):
        GBA = nx.barabasi_albert_graph(N, m)
        Gcc = sorted(nx.connected_components(GBA), key=len, reverse=True)
        GBA = GBA.subgraph(Gcc[0])
        GBA = nx.convert_node_labels_to_integers(GBA, first_label=0)
        k1,k2,variance,av_cl,l,r,Cx,CC,D,av_EC,Mod,H = measures(GBA)
        x = [k1,k2,variance,av_cl,l,r,Cx,CC,D,av_EC,Mod,H]
        X.append(x)
        y.append(2.0)


    # Waxman networks
    for i in range(0,n_nets):
        GWX = nx.waxman_graph(N, 0.45, 0.2)
        Gcc = sorted(nx.connected_components(GWX), key=len, reverse=True)
        GWX = GWX.subgraph(Gcc[0])
        GWX = nx.convert_node_labels_to_integers(GWX, first_label=0)
        k1,k2,variance,av_cl,l,r,Cx,CC,D,av_EC,Mod,H = measures(GWX)
        x = [k1,k2,variance,av_cl,l,r,Cx,CC,D,av_EC,Mod,H]
        X.append(x)
        y.append(3.0)
```

```
    # Geometric networks
    for i in range(0,n_nets):
        GG = nx.random_geometric_graph(N, 0.18)
        Gcc = sorted(nx.connected_components(GG), key=len, reverse=True)
        GG = GG.subgraph(Gcc[0])
        GG = nx.convert_node_labels_to_integers(GG, first_label=0)
        k1,k2,variance,av_cl,l,r,Cx,CC,D,av_EC,Mod,H = measures(GG)
        x = [k1,k2,variance,av_cl,l,r,Cx,CC,D,av_EC,Mod,H]
        X.append(x)
        y.append(4.0)

    return X, y
```

In [8]:

```
def test_model_Knn(G, n_nets):

    k1,k2,variance,av_cl,l,r,Cx,CC,D,av_EC,Mod,H = measures(G)
    X_net = [k1,k2,variance,av_cl,l,r,Cx,CC,D,av_EC,Mod,H]

    X, y = train_model(G, n_nets)

    X = np.array(X)
    y = np.array(y)
    XK = X
    yK = y

    scaler = StandardScaler().fit(XK)
    XK = scaler.transform(XK)

    X_net = np.array(X_net)
    X_net = X_net.reshape(1,len(X_net))
    X_net = scaler.transform(X_net)

    k = 5
    model = KNeighborsClassifier(n_neighbors=k, metric = 'euclidean')
    model.fit(XK,yK)
    # faz a predição no conjunto de teste

    cl = ['ER','WS(p=0.1)','BA','Waxman','Geometric']
    y_pred = model.predict(X_net)
    print('Classe:', cl[int(y_pred)])
```

In [9]:

```
def test_model_LDA(G, n_nets):

    k1,k2,variance,av_cl,l,r,Cx,CC,D,av_EC,Mod,H = measures(G)
    X_net = [k1,k2,variance,av_cl,l,r,Cx,CC,D,av_EC,Mod,H]

    X_net = np.array(X_net)
    X_net = X_net.reshape(1,len(X_net))

    X, y = train_model(G, n_nets)

    X = np.array(X)
    y = np.array(y)
    XLDA = X
    yLDA = y

    scaler = StandardScaler().fit(XLDA)
    XLDA = scaler.transform(XLDA)
    X_net = scaler.transform(X_net)

    model = LinearDiscriminantAnalysis()
    model.fit(XLDA,yLDA)

    # faz a predição no conjunto de teste

    cl = ['ER','WS(p=0.1)','BA','Waxman','Geometric']
    y_pred = model.predict(X_net)
    print('Classe:', cl[int(y_pred)])
```

In [10]:

```
from sklearn.decomposition import PCA
import pandas as pd

def test_model_PCA(G, n_nets):

    k1,k2,variance,av_cl,l,r,Cx,CC,D,av_EC,Mod,H = measures(G)
    X_net = [k1,k2,variance,av_cl,l,r,Cx,CC,D,av_EC,Mod,H]

    X_net = np.array(X_net)
    X_net = X_net.reshape(1,len(X_net))

    X, y = train_model(G, n_nets)

    X = np.array(X)
    y = np.array(y)
    XPCA = X
    yPCA = y

    XPCA = pd.DataFrame(XPCA)
    X_netPCA = pd.DataFrame(X_net)
    PCATESTEX = pd.concat([XPCA, X_netPCA])
    yPCA = pd.DataFrame(y)
    yPCA.loc[150] = [5.0]

    scaler = StandardScaler().fit(PCATESTEX)
    PCATESTEX = scaler.transform(PCATESTEX)


    pca = PCA(n_components=2)
    principalComponents = pca.fit_transform(PCATESTEX)
    principalDf = pd.DataFrame(data = principalComponents
                , columns = ['principal component 1', 'principal component 2'])

    yPCA = pd.DataFrame(yPCA)
    finalDf = pd.concat([principalDf, yPCA], axis = 1)
    finalDf.columns = ['principal component 1', 'principal component 2', 'Modelo']

    cl = ['ER','WS(p=0.1)','BA','Waxman','Geometric']

    fig = plt.figure(figsize = (8,8))
    ax = fig.add_subplot(1,1,1)
    ax.set_xlabel('Principal Component 1', fontsize = 15)
    ax.set_ylabel('Principal Component 2', fontsize = 15)
    ax.set_title('2 component PCA', fontsize = 20)
    targets = [0.0,1.0,2.0,3.0,4.0,5.0]
    colors = ['r', 'g', 'b', 'orange', 'yellow', 'pink']
    for target, color in zip(targets,colors):
        indicesToKeep = finalDf['Modelo'] == target
        ax.scatter(finalDf.loc[indicesToKeep, 'principal component 1']
                   , finalDf.loc[indicesToKeep, 'principal component 2']
                   , c = color
                   , s = 30)
    ax.legend(cl)
    ax.grid()
```

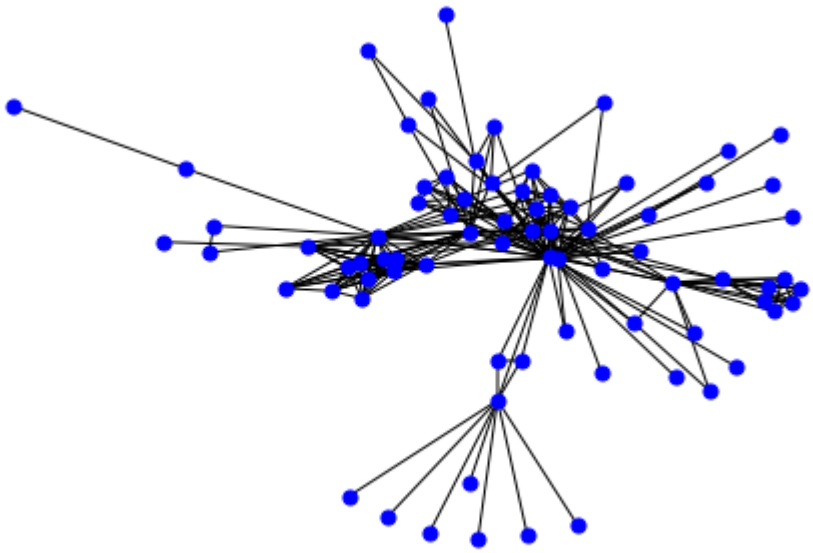
##Classificação

###Teste Rede Lesmis

In [11]:

```
Gteste = get_graph_from_data_file("lesmis.txt", plot=True)
```

```
Number of nodes: 77
Grau médio: 6.597402597402597
```



In [18]:

```
test_model_Knn(Gteste,40)
```

```
Classe: Geometric
```

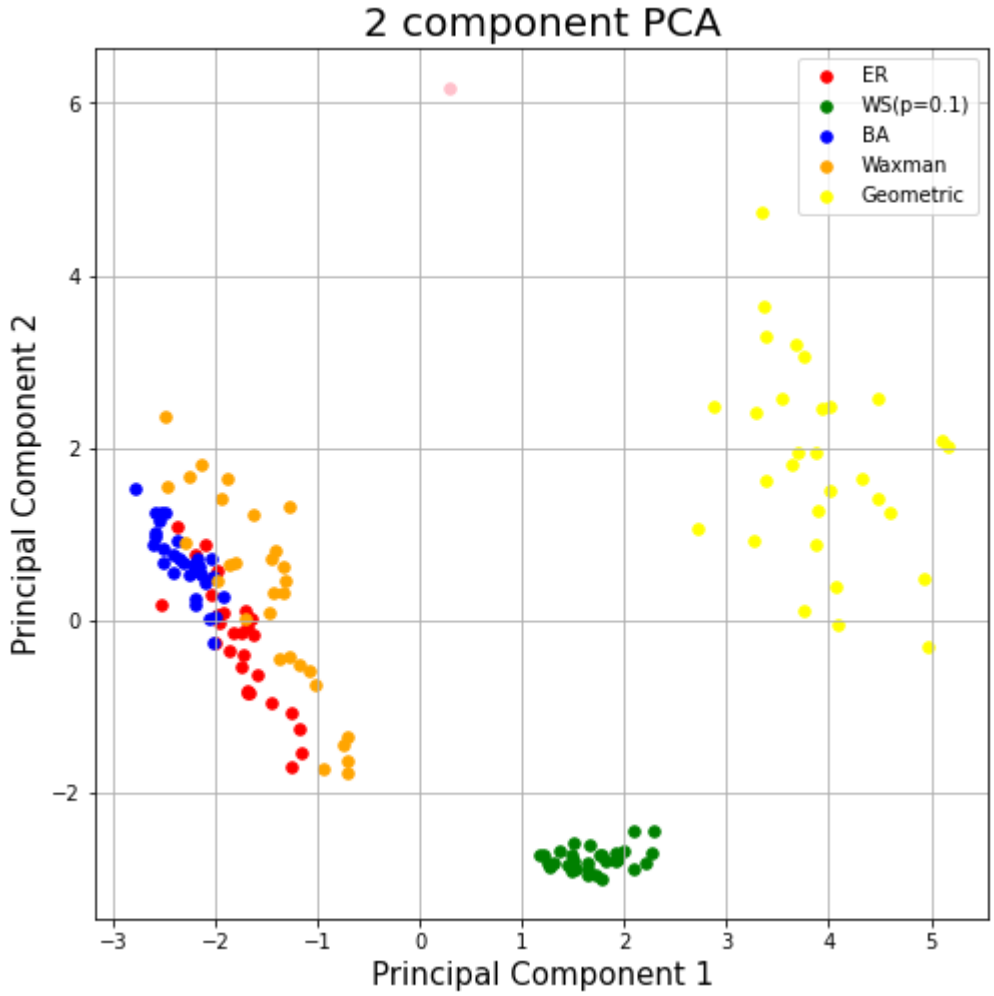
In [19]:

```
test_model_LDA(Gteste,40)
```

```
Classe: WS(p=0.1)
```

In [18]:

```
test_model_PCA(Gteste,30)
```



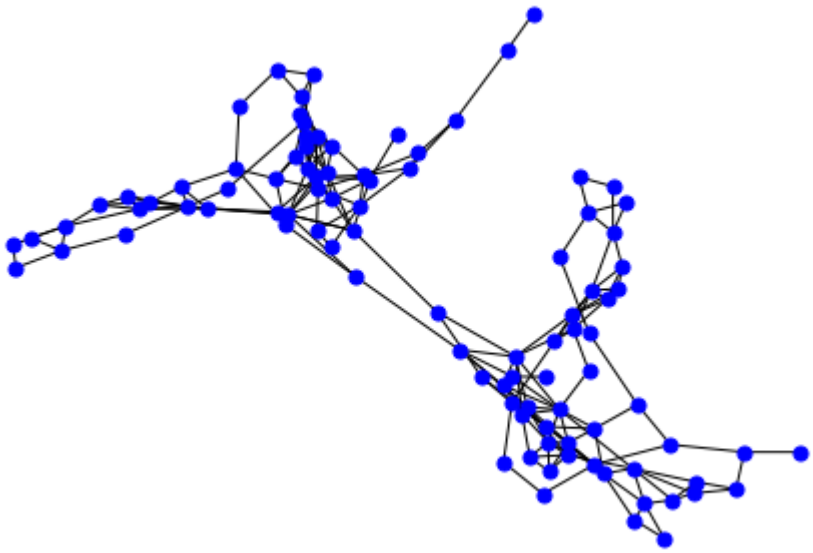
###Biológicas

###Malária

In [19]:

```
GColi = get_graph_from_data_file("ColiNet.txt", plot=True)
```

```
Number of nodes: 97
Grau médio: 4.371134020618556
```



In [37]:

```
test_model_Knn(GColi,40)
```

```
Classe: Geometric
```

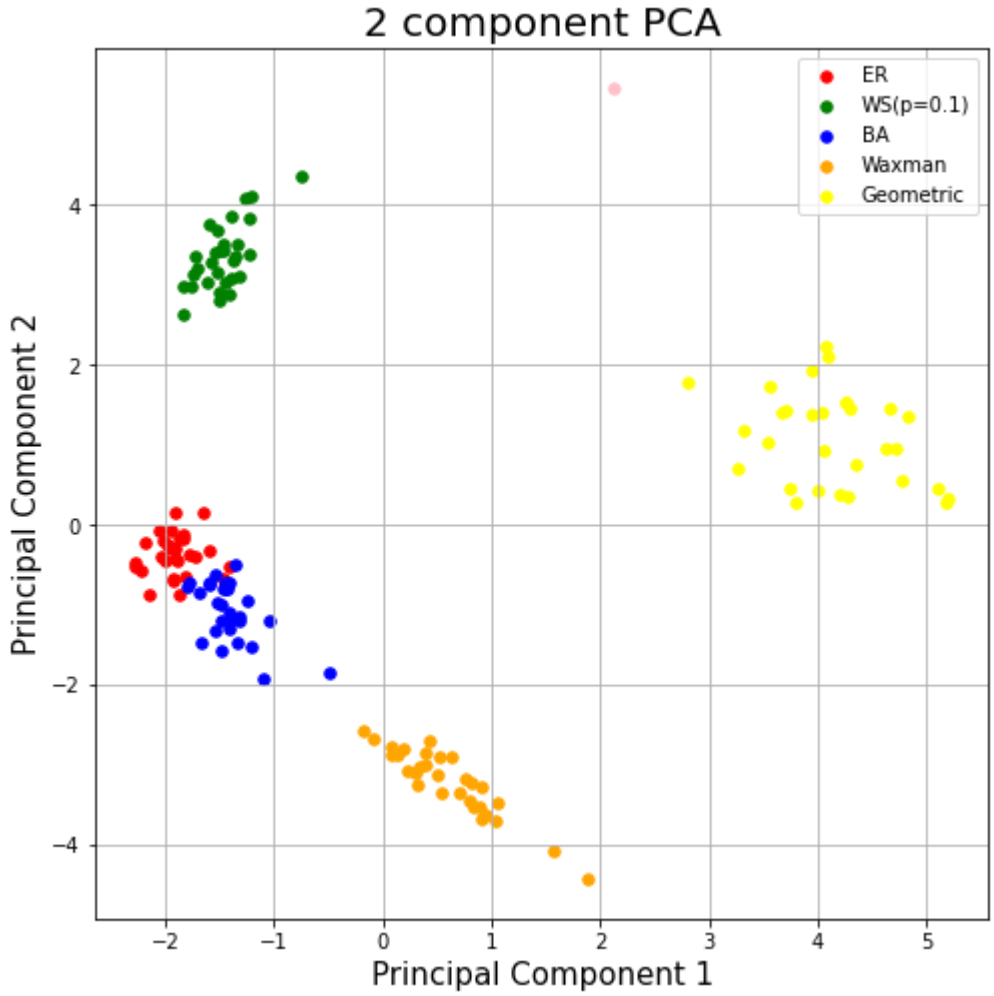
In [38]:

```
test_model_LDA(GColi,40)
```

```
Classe: Geometric
```

In [20]:

```
test_model_PCA(GColi,30)
```

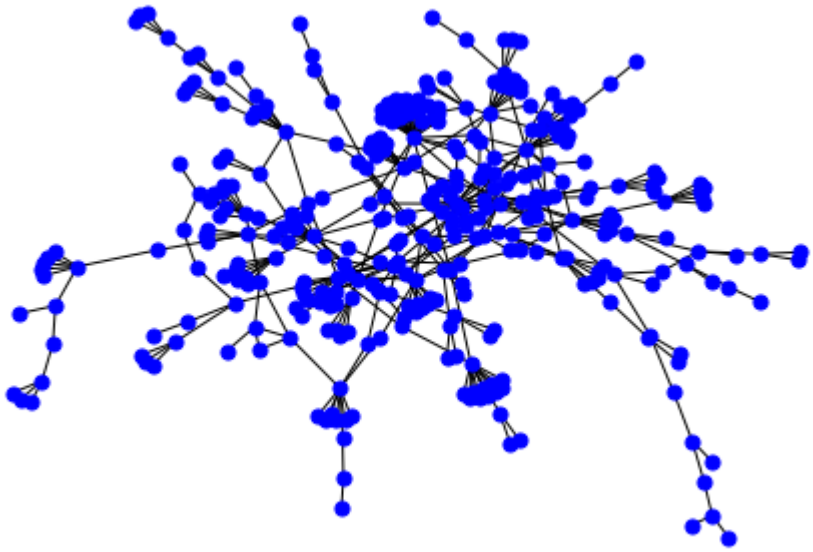


###BPMaps

In [23]:

```
Gbio = get_graph_from_data_file("BPMaps.txt", plot=True)
```

```
Number of nodes: 345
Grau médio: 2.318840579710145
```



In [18]:

```
test_model_Knn(Gbio,40)
```

```
Classe: BA
```

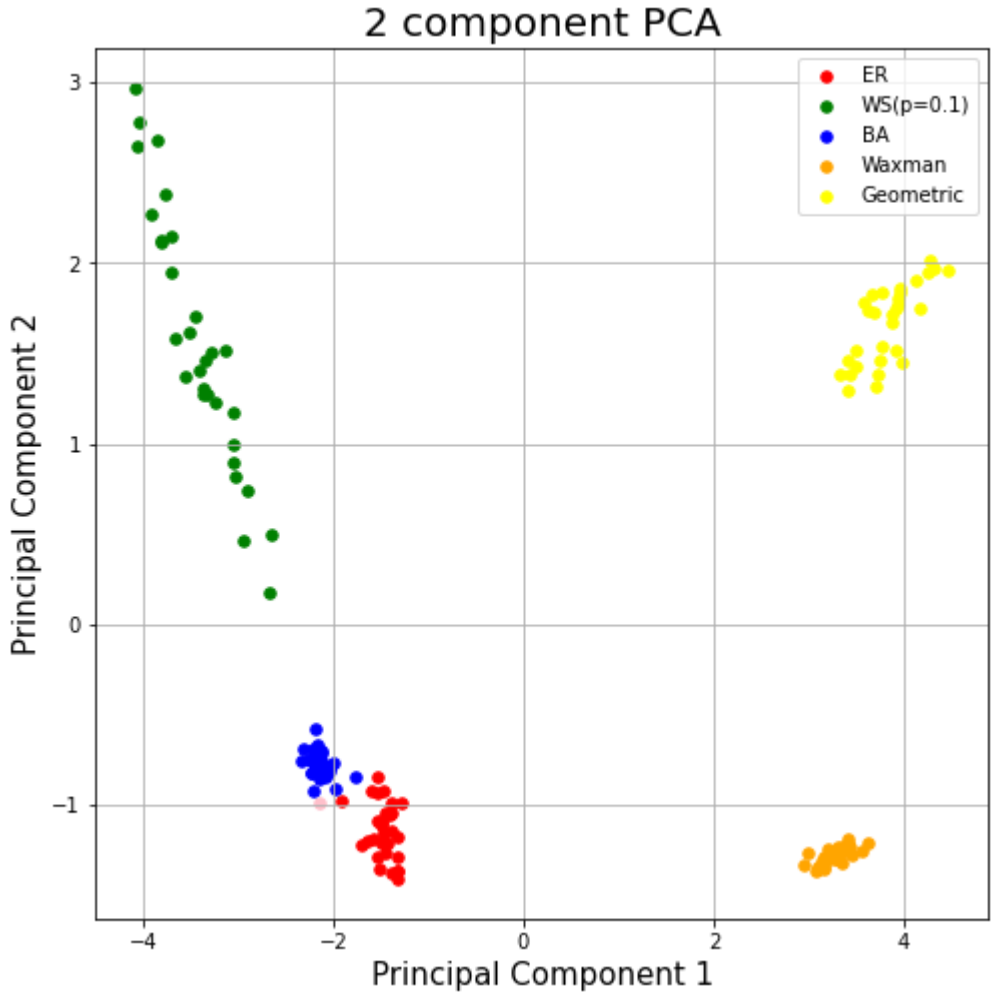
In [25]:

```
test_model_LDA(Gbio,40)
```

```
Classe: BA
```

In [23]:

```
test_model_PCA(Gbio,30)
```

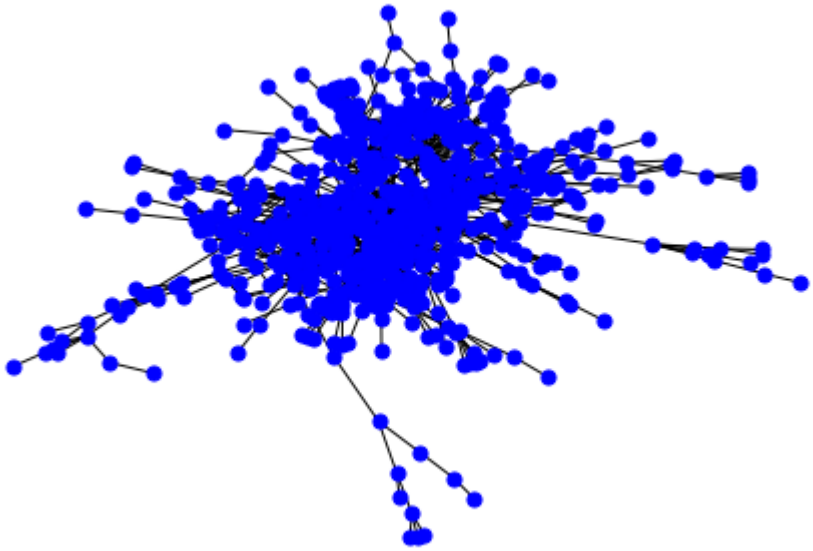


###Genética

In [25]:

```
Ggen = get_graph_from_data_file("Genetic.txt", plot=True)
```

```
Number of nodes: 683
Grau médio: 4.5183016105417275
```



In [10]:

```
test_model_Knn(Ggen,40)
```

```
Classe: BA
```

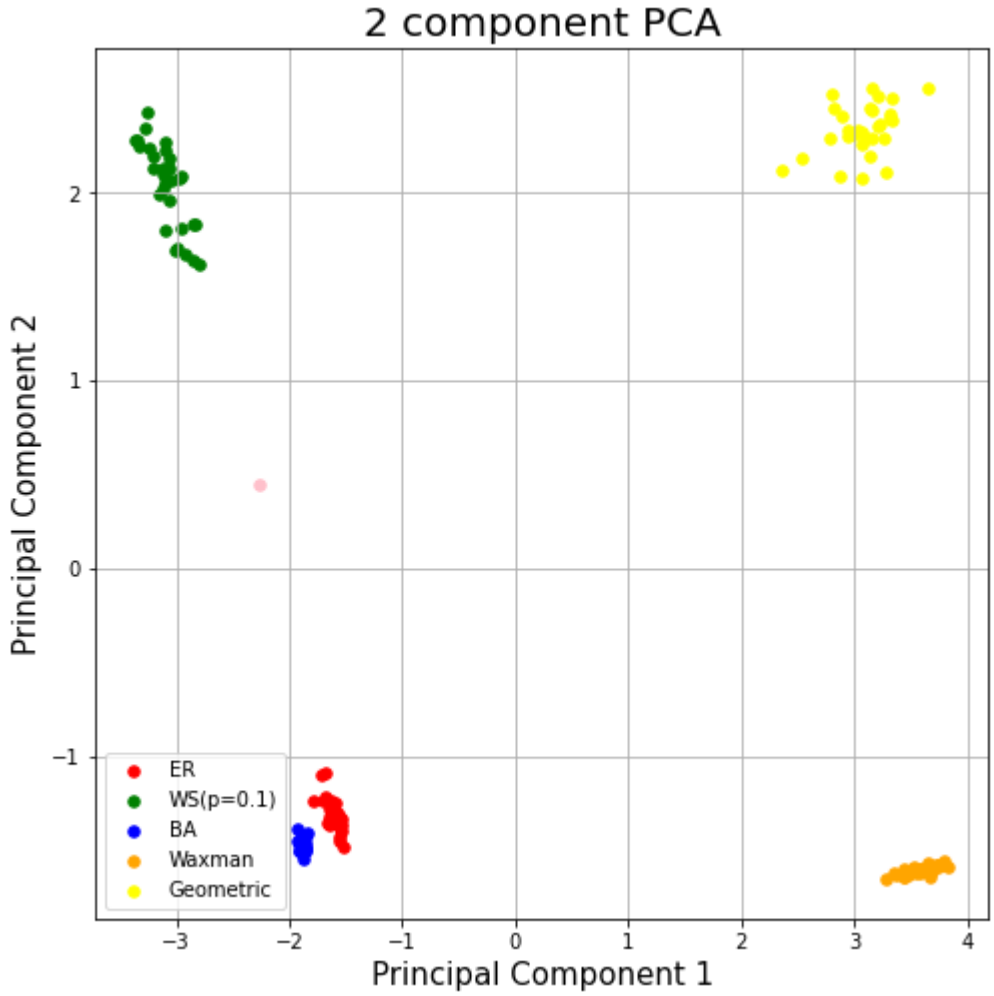
In [15]:

```
test_model_LDA(Ggen,40)
```

```
Classe: ER
```

In [25]:

```
test_model_PCA(Ggen,30)
```



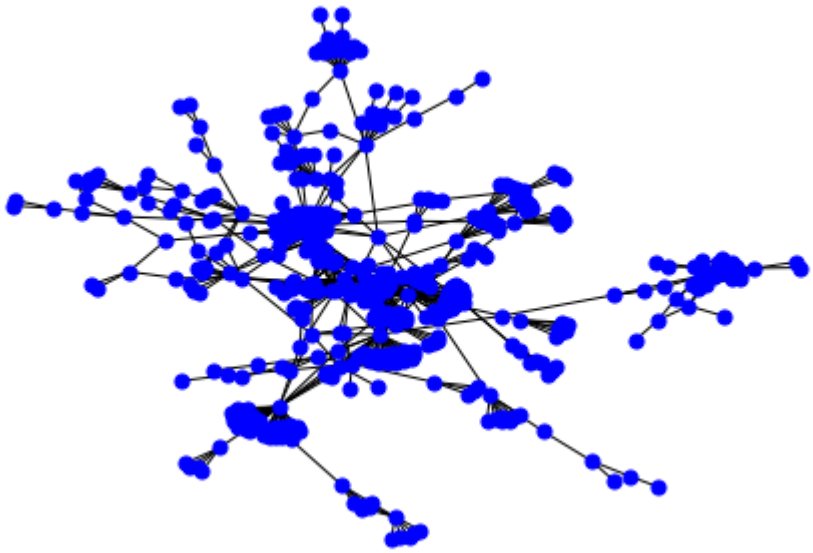
###Tecnológicas

###Guava

In [26]:

```
GGuava = get_graph_from_data_file("GuavaTec.txt", plot=True)
```

```
Number of nodes: 457
Grau médio: 4.052516411378556
```



In [9]:

```
test_model_Knn(GGuava,40)
```

```
Classe: WS(p=0.1)
```

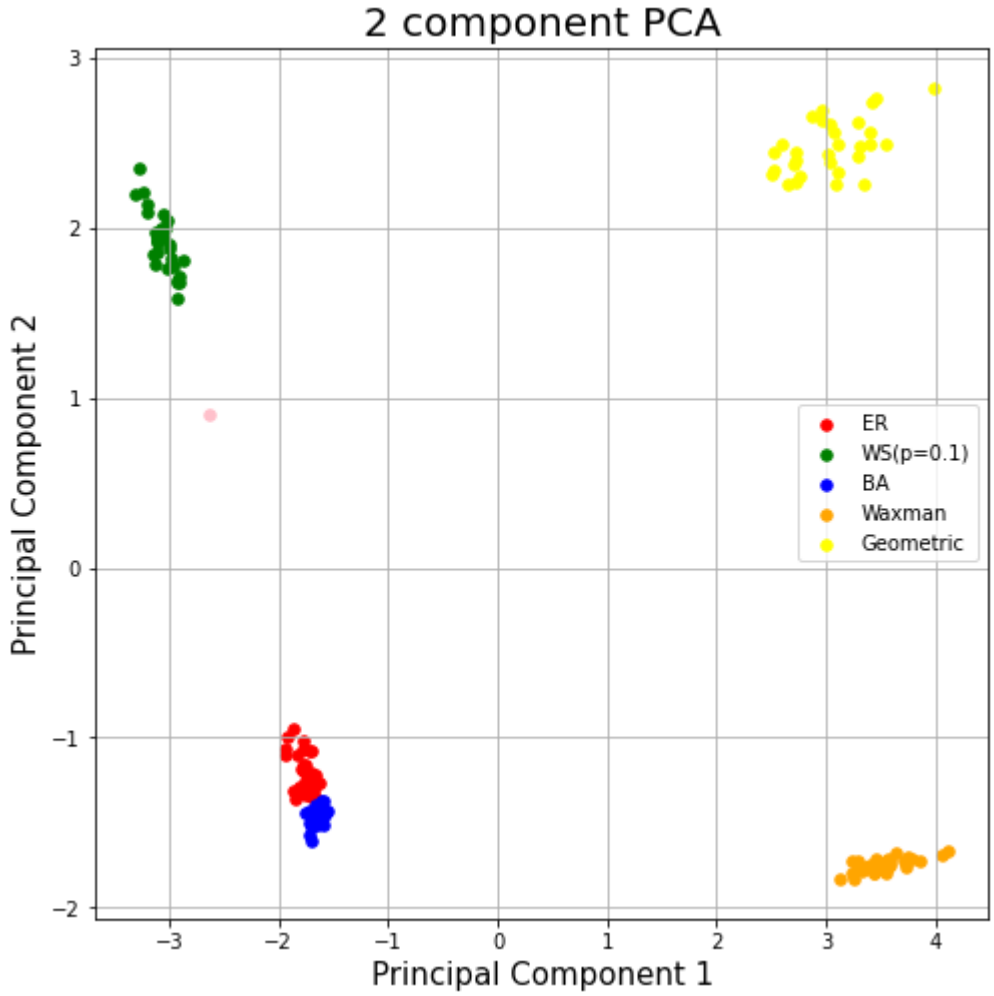
In [10]:

```
test_model_LDA(GGuava,40)
```

```
Classe: ER
```

In [27]:

```
test_model_PCA(GGuava,30)
```

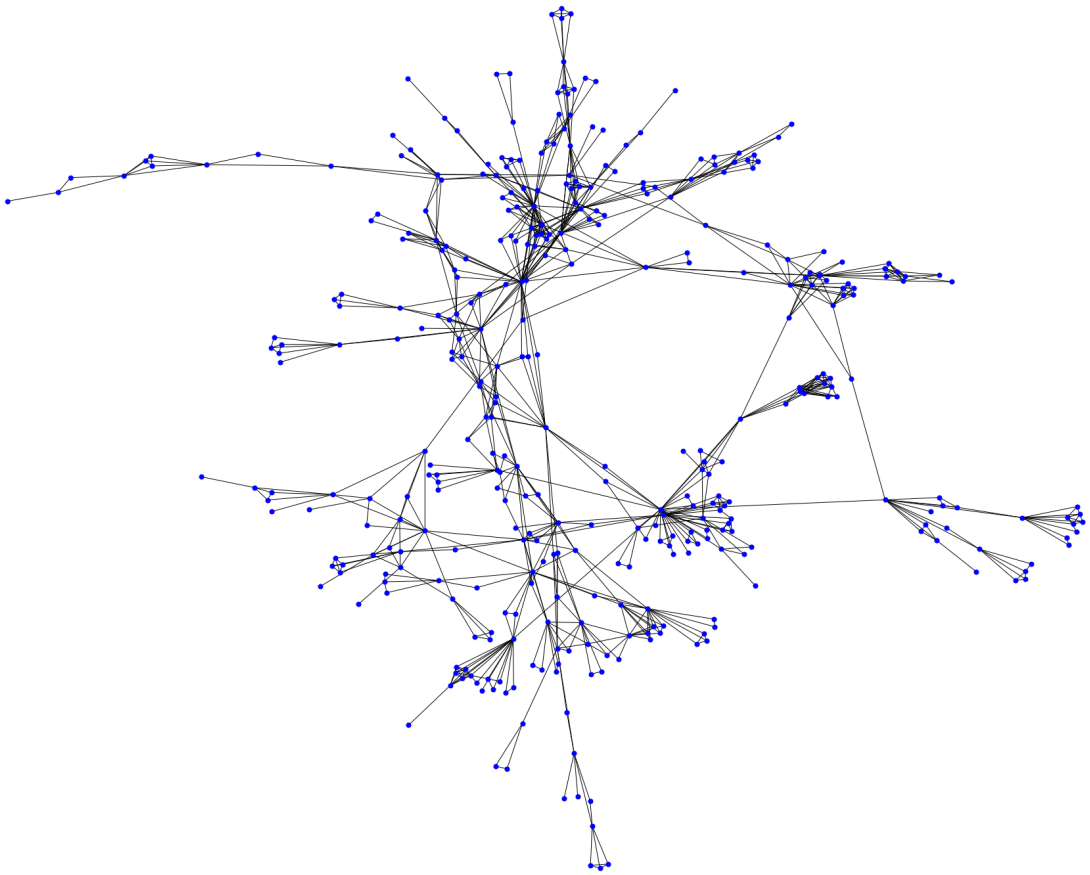


###Unicode

In [33]:

```
GUnicode = get_graph_from_data_file("Unicode.txt", plot=True)
```

```
Number of nodes: 379
Grau médio: 4.823218997361478
```



In [22]:

```
test_model_Knn(GUnicode ,40)
```

```
Classe: WS(p=0.1)
```

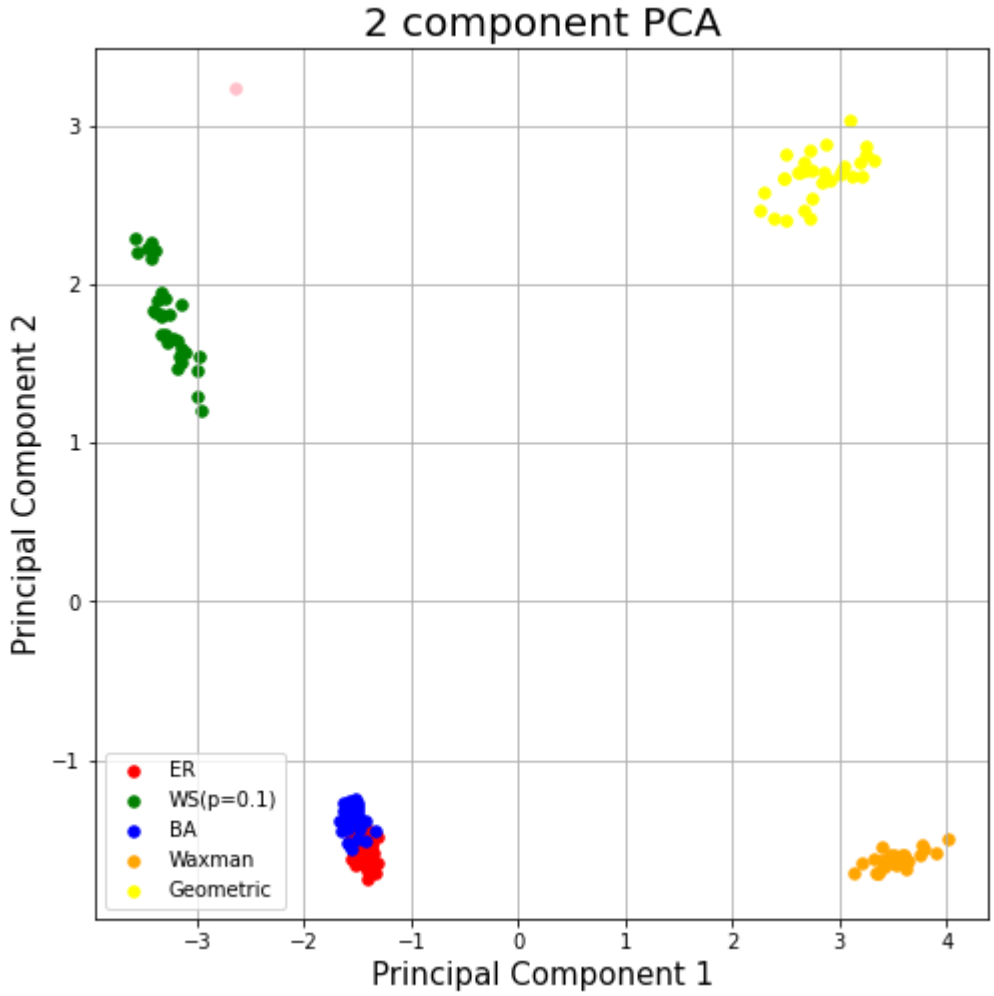
In [23]:

```
test_model_LDA(GUnicode ,40)
```

```
Classe: WS(p=0.1)
```

In [29]:

```
test_model_PCA(GUnicode ,30)
```

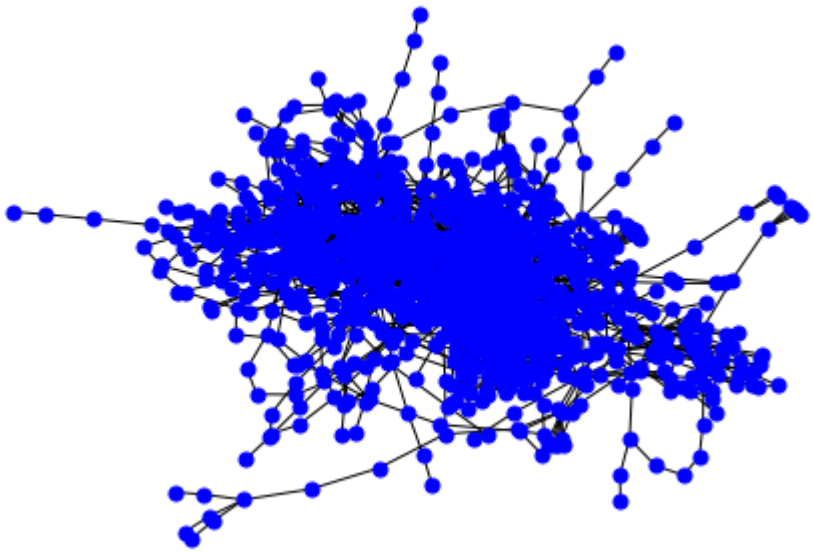


###FAA

In [30]:

```
GFAA = get_graph_from_data_file("TecFAA.txt", plot=True)
```

```
Number of nodes: 1226
Grau médio: 3.9282218597063623
```



In [24]:

```
test_model_Knn(GFAA,40)
```

```
Classe: BA
```

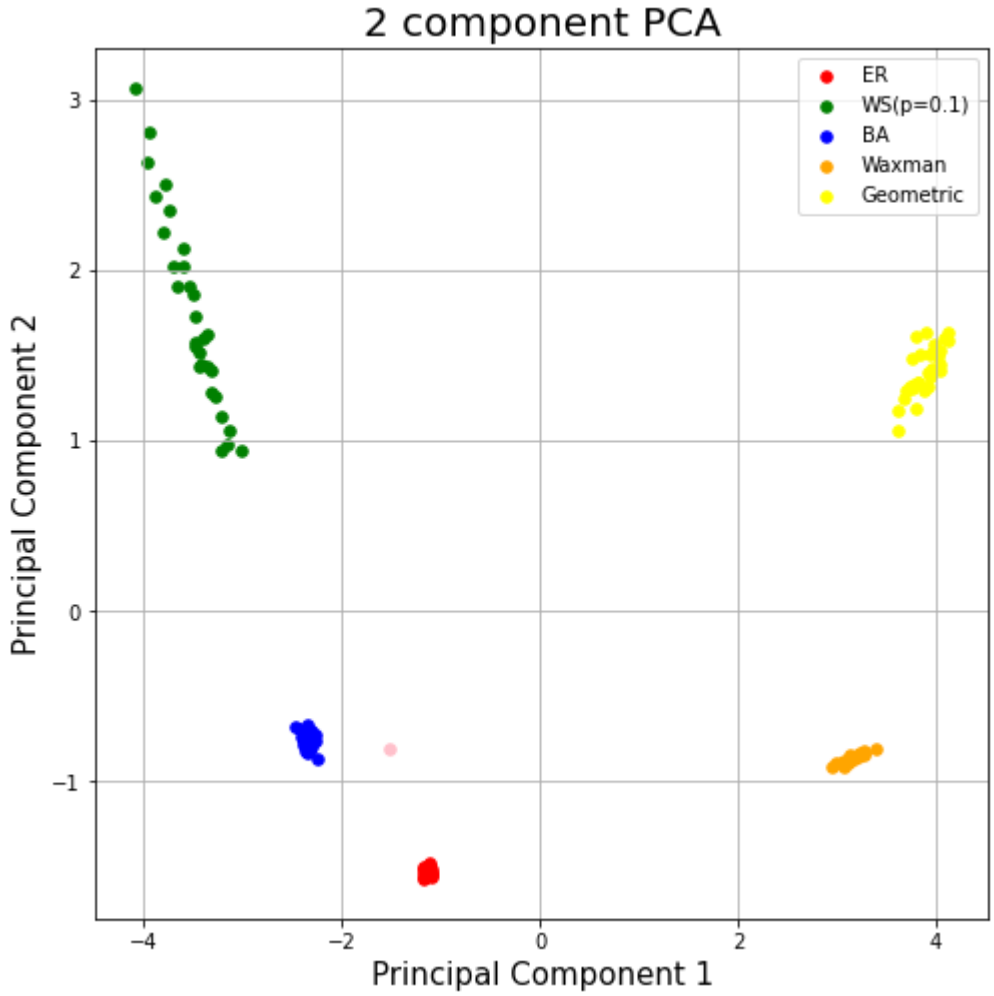
In [25]:

```
test_model_LDA(GFAA,40)
```

```
Classe: ER
```

In [31]:

```
test_model_PCA(GFAA,30)
```



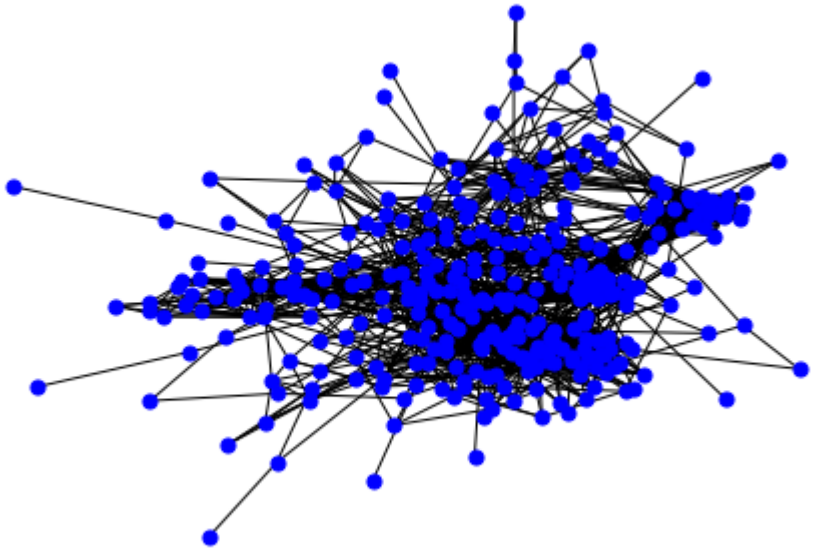
###Sociais

###Slavko

In [32]:

```
GSla = get_graph_from_data_file("Slavko.txt", plot=True)
```

```
Number of nodes: 324
Grau médio: 13.691358024691358
```



In [12]:

```
test_model_Knn(GSla,40)
```

```
Classe: Geometric
```

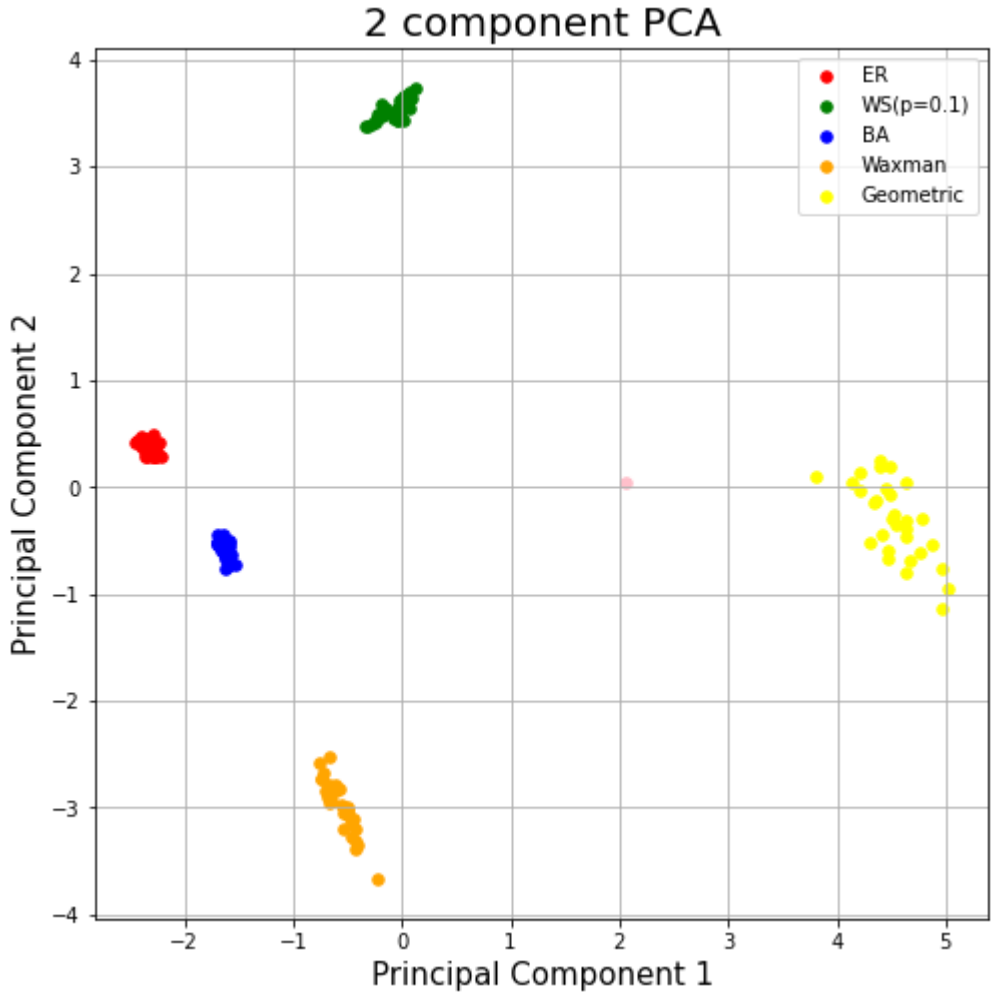
In [13]:

```
test_model_LDA(GSla,40)
```

```
Classe: WS(p=0.1)
```

In [33]:

```
test_model_PCA(GSla,30)
```

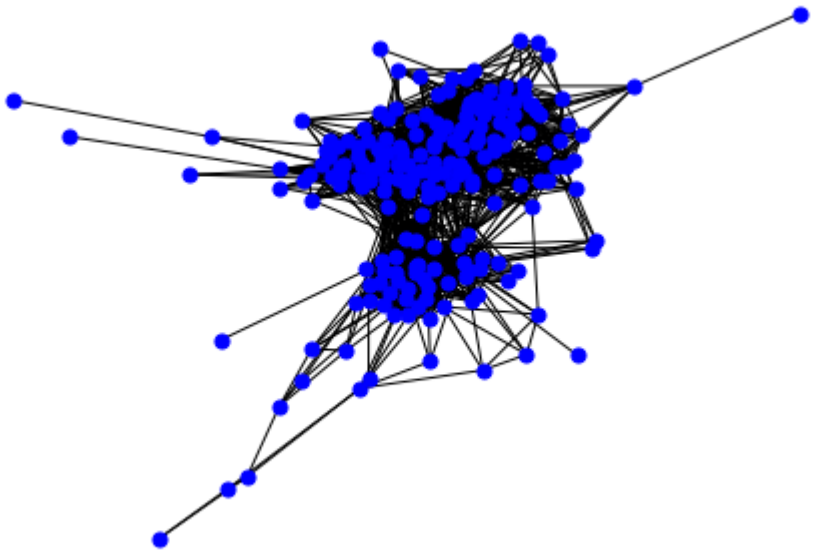


###JAZZ

In [11]:

```
GJazz = get_graph_from_data_file("jazz.txt", plot=True)
```

```
Number of nodes: 198
Grau médio: 27.696969696969695
```



In [27]:

```
test_model_Knn(GJazz,40)
```

```
Classe: BA
```

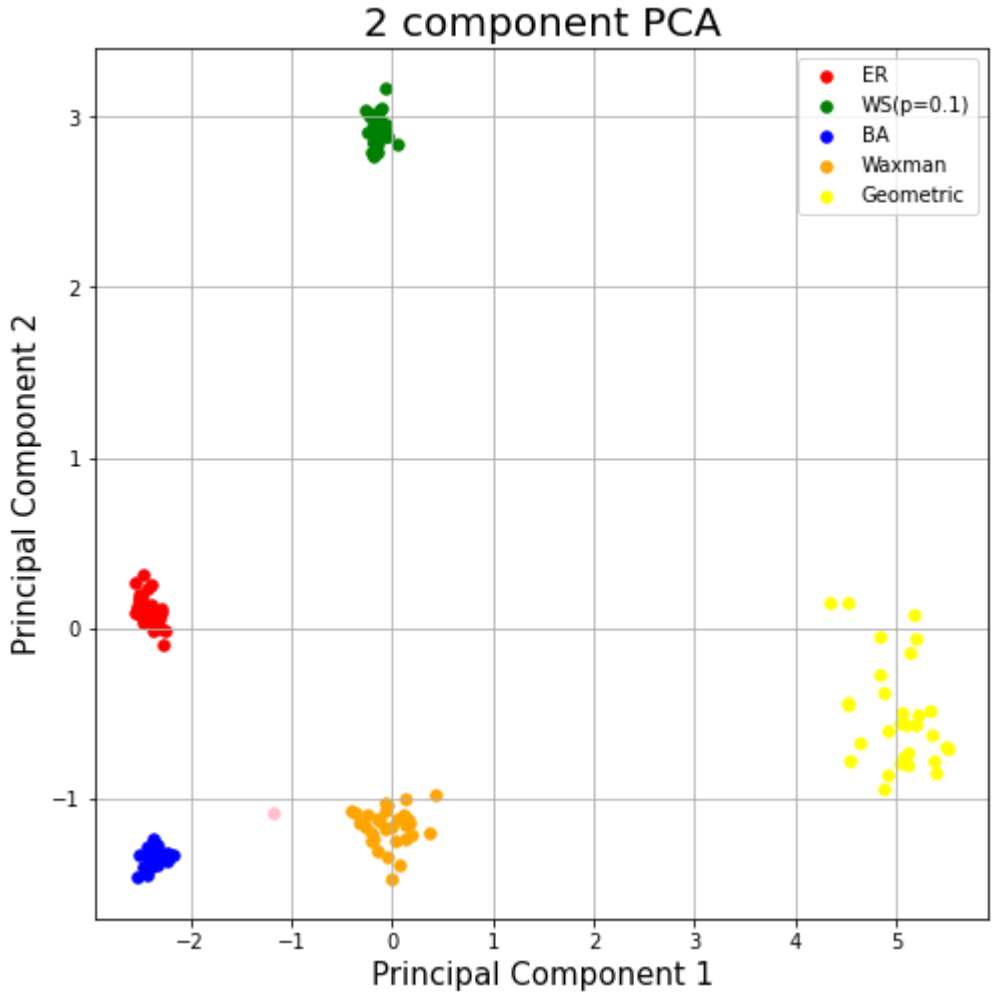
In [28]:

```
test_model_LDA(GJazz,40)
```

```
Classe: BA
```

In [35]:

```
test_model_PCA(GJazz,30)
```

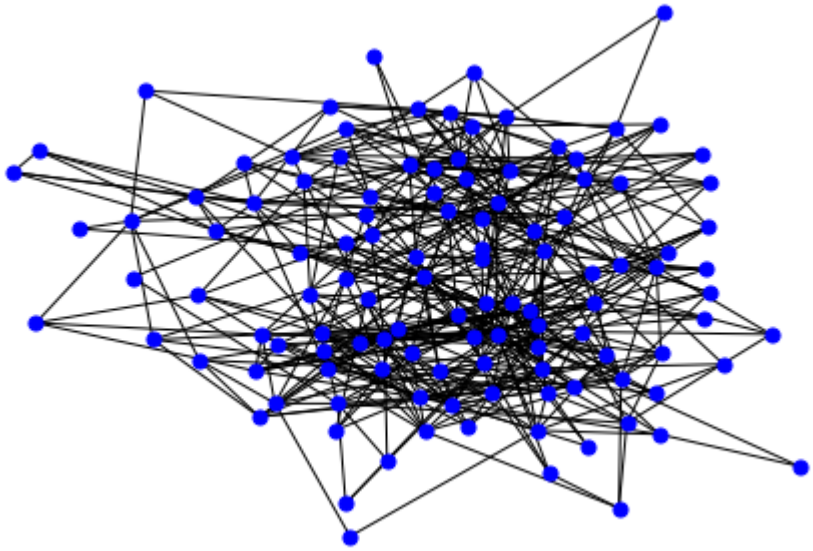


###Physicians

In [36]:

```
GPhys = get_graph_from_data_file("Physicians.txt", plot=True)
```

```
Number of nodes: 117
Grau médio: 7.948717948717949
```



In [30]:

```
test_model_Knn(GPhys,40)
```

```
Classe: ER
```

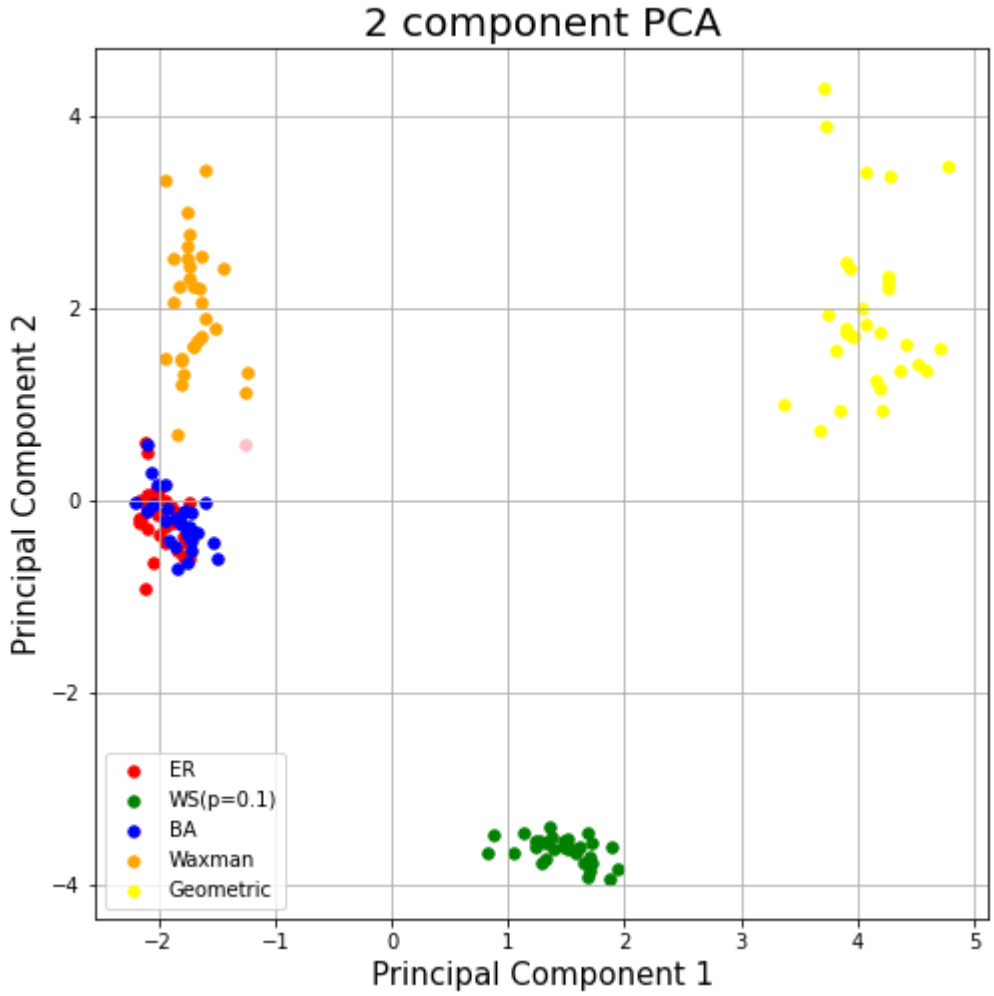
In [31]:

```
test_model_LDA(GPhys,40)
```

```
Classe: ER
```

In [37]:

```
test_model_PCA(GPhys,30)
```



##**Desafio**

In [13]:

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt
```

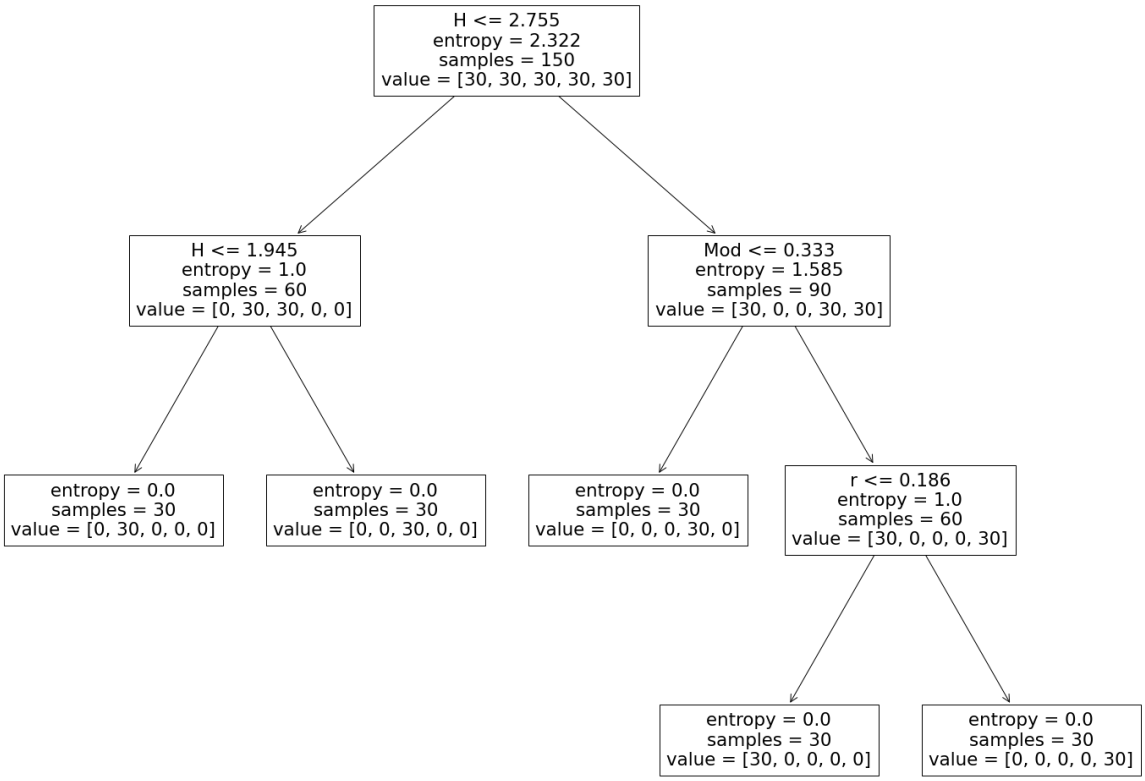
In [49]:

```
def test_model_dt(G, n_nets):

    k1,k2,variance,av_cl,l,r,Cx,CC,D,av_EC,Mod,H = measures(G)
    X_net = [k1,k2,variance,av_cl,l,r,Cx,CC,D,av_EC,Mod,H]

    X_net = np.array(X_net)
    X_net = X_net.reshape(1,len(X_net))

    X, y = train_model(G, n_nets)

    X = np.array(X)
    y = np.array(y)
    X_train = X
    y_train = y

    X_train = pd.DataFrame(X_train)
    y_train = pd.DataFrame(y_train)

    X_train.columns = ['k1','k2','variance','av_cl','l','r','Cx','CC','D','av_EC','M

    dt = DecisionTreeClassifier(criterion="entropy")
    dt.fit(X_train, y_train)

    for col, importance in zip(X_train.columns, dt.feature_importances_):
        print(f"Atributo '{col}' tem {importance} de importância")

    # faz a predição no conjunto de teste

    # cl = ['ER','WS(p=0.1)','BA','Waxman','Geometric']
    # y_pred = dt.predict(X_net)
    # print('Classe:', cl[int(y_pred)])

    plt.rcParams["figure.figsize"] = (25, 20)
    _ = plot_tree(dt, feature_names=X_train.columns)
```

In [50]:

```
test_model_dt(GUnicode, 30)
```

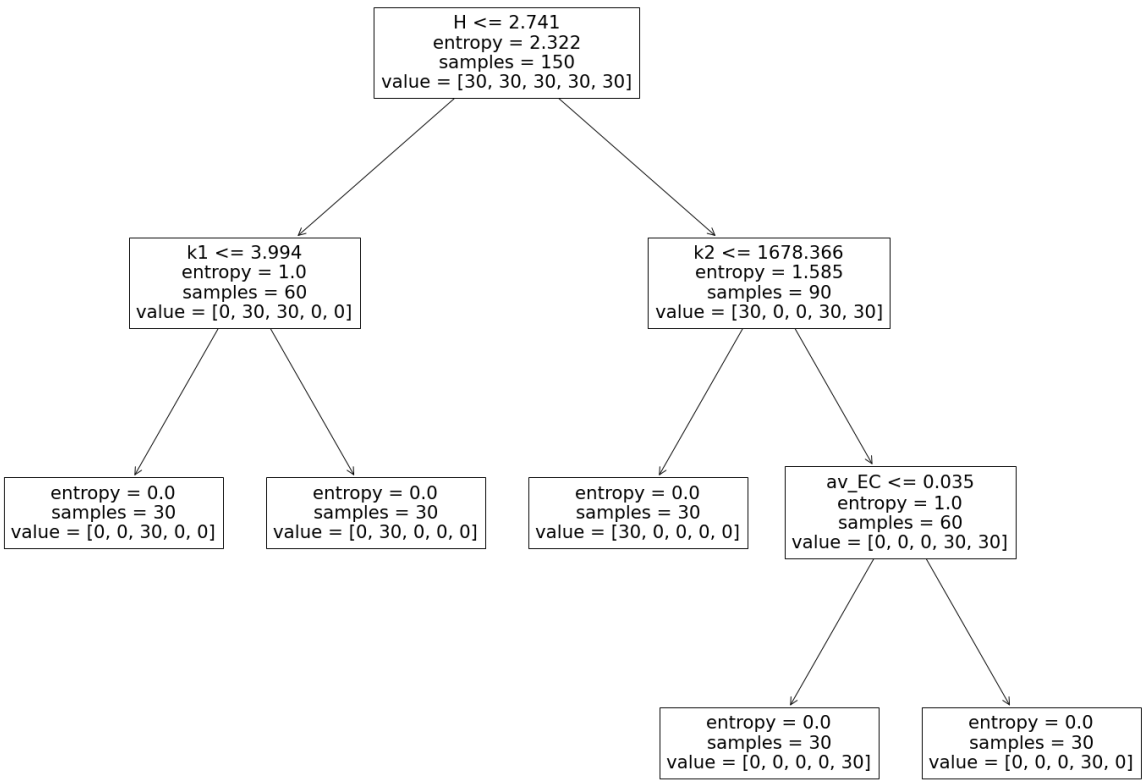
```
Atributo 'k1' tem 0.0 de importância
Atributo 'k2' tem 0.0 de importância
Atributo 'variance' tem 0.0 de importância
Atributo 'av_cl' tem 0.0 de importância
Atributo 'l' tem 0.0 de importância
Atributo 'r' tem 0.17227062322935724 de importância
Atributo 'Cx' tem 0.0 de importância
Atributo 'CC' tem 0.0 de importância
Atributo 'D' tem 0.0 de importância
Atributo 'av_EC' tem 0.0 de importância
Atributo 'Mod' tem 0.23729309346223396 de importância
Atributo 'H' tem 0.5904362833084088 de importância
```



In [51]:

```
test_model_dt(Ggen, 30)
```

```
Atributo 'k1' tem 0.17227062322935724 de importância
Atributo 'k2' tem 0.23729309346223396 de importância
Atributo 'variance' tem 0.0 de importância
Atributo 'av_cl' tem 0.0 de importância
Atributo 'l' tem 0.0 de importância
Atributo 'r' tem 0.0 de importância
Atributo 'Cx' tem 0.0 de importância
Atributo 'CC' tem 0.0 de importância
Atributo 'D' tem 0.0 de importância
Atributo 'av_EC' tem 0.17227062322935724 de importância
Atributo 'Mod' tem 0.0 de importância
Atributo 'H' tem 0.41816566007905154 de importância
```



Discussão

Como é possível observar, o atributo que mais contribuí depende muito da topologia da rede a ser classificada, e com isso dos modelos de redes com base nessa rede alvo.

Entretando, vale dizer que para uma análise mais completa outros modelos de árvores de decisão deveriam ser testados, assim como os parâmetros e hiperparâmetros deveriam ser melhor estudados.