クレジット:
UTokyo Online Educationデータマイニング入門 2018 森　純一郎



東京大学
THE UNIVERSITY OF TOKYO

CC BY NC ND

# 課題5 ネットワーク分析

In [ ]:
```
# モジュールのインポート
import networkx as nx
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
```

## Q1

以下では、networkxモジュールに含まれるデータセット、空手クラブのソーシャルネットワーク、からグラフオブジェクトを作成し、ネットワーク全体を可視化しています。ネットワークは無方向で重みなしです。また自己ループやノード間の多重リンクはありません。

In [ ]:
```
G = nx.karate_club_graph() # グラフの作成
pos = nx.spring_layout(G) # 可視化のレイアウト
plt.figure(figsize=(10,10)) # 可視化のサイズ
nx.draw(G, pos=pos, with_labels=True) # 可視化
```

このネットワークには34のノードと78のリンクが含まれ、各ノードには0から33の数字がふられています。以下ではこの数字をノードのラベルとして用います。

In [ ]:
```
print(len(G)) # ノード数
print(G.size()) # リンク数
```

このネットワークの隣接行列 `adj_matrix` は、以下の通りです。

In [ ]:
```
adj_matrix=nx.adjacency_matrix(G).toarray() # 隣接行列の取得
for row in adj_matrix:
    print(row)
```

## Q1.1

以下の辞書 adj_list は、上記にネットワークのノードをキー、そのノードに隣接しているノードのリストを値として持つ辞書です。例えば、キーが31の値のリストを見ることで、ノード31には0, 24, 25, 28, 32, 33のノードが隣接していることがわかります。このような各ノードの隣接ノードリストの辞書を隣接リストと呼びます。

隣接行列 adj_matrix を受け取り、隣接リストとなる辞書 adj_list を作成して返す get_adj_list() 関数を実装してください。

```
adj_list={0: [1, 2, 3, 4, 5, 6, 7, 8, 10, 11, 12, 13, 17, 19, 2
1, 31],
                1: [0, 2, 3, 7, 13, 17, 19, 21, 30],
                2: [0, 1, 3, 7, 8, 9, 13, 27, 28, 32],
                  ...
                31: [0, 24, 25, 28, 32, 33],
                32: [2, 8, 14, 15, 18, 20, 22, 23, 29, 30, 31,
33],
                33: [8, 9, 13, 14, 15, 18, 19, 20, 22, 23, 26,
27, 28, 29, 30, 31, 32]}
```

In [ ]:
```
def get_adj_list(adj_matrix):

```

get_adj_list 関数が完成したら、以下のセルを実行して、 adj_list[0] や adj_list[33] を表示し、上記に示されている各ノードの隣接ノードリストが作成できているか確認してください。

In [ ]:
```
adj_list=get_adj_list(adj_matrix)
print(adj_list[0])
print(adj_list[33])
```

## Q1.2

講義資料の幅優先探索のアルゴリズムを参考に、第1引数として隣接リスト、第2引数として始点ノードのラベルを受け取り、始点から各ノードへの距離の辞書 d を返す以下の bfs 関数を完成させてください。隣接リストは上記で作成した adj_list を用います。 d は、キーがノードのラベル、値が始点からそのノードへの距離となる辞書です。

講義資料の Q.enqueue , Q.dequeue はそれぞれコード中の Q.append , Q.popleft に対応します。

In [ ]:
```
from collections import deque # キューのインポート
def bfs(g, s):

```

bfs 関数が完成したら、以下のセルを実行して動作を確認してください。ノード0からノード26への長さは3、ノード11からノード26への長さは4となります。

In [ ]:
```
d1=bfs(adj_list, 0)
print(d1[26])

d2=bfs(adj_list, 11)
print(d2[26])
```

# Q2

以下では、Q1で用いた空手クラブのソーシャルネットワークデータについて各ノードの次数中心性を計算し、次数中心性の上位のノードを表示しています。ノード33が最も次数中心性が高いことがわかります。

In [ ]:
```
G = nx.karate_club_graph()
plt.figure(figsize=(10,10)) # 可視化のサイズ
nx.draw(G, pos=pos, with_labels=True) # 可視化
```

In [ ]:
```
# 次数中心性 (次数/最大次数 (N-1))
degree=nx.degree_centrality(G)
print(pd.Series(degree).sort_values(ascending=False).head(5))
```

## Q2.1

近接中心性、媒介中心性、固有ベクトル中心性の各中心性について、上記の次数中心性と同様に各ノードの中心性を計算し、ネットワーク図の各ノードの位置を参照しながらそれぞれの中心性で上位のノードがどのような位置にあるか観察してください。

In [ ]:
```
closeness=
betweenness=
eigenvector=
```

## Q2.2

以下では各ノードのPageRankを計算する関数を実装します。 `pagerank_centrality` 関数は、第1引数としてネットワークの隣接行列、第2引数としてダンピング係数（通常移動とテレポート移動の割合を指定）、第3引数としてベキ乗法の繰り返し回数を受け取り、各ノードのPageRankを要素とするベクトルを返します。ベクトルの各要素のインデックスはノードのラベルに対応します。

PageRankを更新するための行列$B$はダンピング係数を$\alpha$、推移確率行列を$A$として以下のように定義されます。行列$A$はネットワークの隣接行列について、すべての要素が0の列があればその列のすべての要素を1とした上で、隣接行列の各要素についてその列の和で割ったものです。

$$B = \alpha A + \frac{1-\alpha}{\text{ノード数}}$$

行列$B$を用いてPageRankのベクトル$x = (x_1, x_2, \ldots, x_n)$は以下のように更新されます。

$$x = Bx$$

$$x = x/\Sigma_{i=1}^{n} x_i$$

In [ ]:
```
def pagerank_centrality(adj_matrix, alpha=0.85, t=100):

```

`pagerank_centrality` 関数が完成したら、以下のセルを実行して動作を確認してください。最もPageRankが高いノードは33でそのPageRankは$\simeq 0.10$、次にPageRankが高いノードは0でそのPageRankは$\simeq 0.09$となります。

In [ ]:
```
adj_matrix=nx.adjacency_matrix(G).toarray()
pagerank=pagerank_centrality(adj_matrix, 0.85, 100)
print(pagerank)
print(pd.Series(pagerank).sort_values(ascending=False).head(5))
```

## Q3

以下では、空手クラブのソーシャルネットワークデータを用いてコミュニティ抽出を行います。まず、コミュニティ抽出前のネットワークを可視化します。

In [ ]:
```
G = nx.karate_club_graph()
pos = nx.spring_layout(G)
plt.figure(figsize=(10,10))
nx.draw_networkx(G, pos=pos)
_ = plt.axis('off')
```

### Q3.1

エッジ媒介中心性を用いたコミュニティ抽出（Girvan-Newman法）を行う以下の `girvan_newman` 関数を完成させてください。 `girvan_newman` 関数は引数としてnetworkxのグラフオブジェクトを受け取り、ネットワークを2つのコミュニティに分割し、各コミュニティのノード集合を要素としたリストを返します。

In [ ]:
```
# グラフGにおいてエッジ媒介中心性が一番大きいエッジの両端ノードをタプルとして返す関数
def find_best_edge(G):
    edge_between = nx.edge_betweenness_centrality(G)
    return max(edge_between, key=edge_between.get)

def girvan_newman(G):

```

`girvan_newman` 関数が完成したら、以下のセルを実行して動作を確認してください。取り除かれるエッジのリスト `removed_edges` は以下になります。

```
[(0, 31), (0, 2), (0, 8), (13, 33), (19, 33), (2, 32), (1, 30)
, (1, 2), (2, 3), (2, 7), (2, 13)]
```

In [ ]:
```
communities, removed_edges = girvan_newman(G)
print(removed_edges)
```

以下のセルを実行してネットワークを可視化し、先の処理で抽出されたコミュニティを確認してください。取り除かれたエッジは破線で示されています。

In [ ]:
```
keeped_edges = set(G.edges()) - set(removed_edges)
colors = ['red', 'yellow']

plt.figure(figsize=(10,10))
nx.draw_networkx_edges(G, pos, edgelist=keeped_edges)
nx.draw_networkx_edges(G, pos, edgelist=removed_edges, style='dashed')
for community, color in zip(communities, colors):
    nx.draw_networkx_nodes(G, pos=pos,
                           nodelist=community, node_color=color)
nx.draw_networkx_labels(G, pos=pos)
_ = plt.axis('off')
```

## Q3.2

以下の modularity 関数は、第1引数としてグラフオブジェクト、第2引数として各コミュニティのノード集合を要素としたリスト（上記のgirvan_newman関数の出力と同じ形式のリスト）、を受け取り、ネットワークのモジュラリティを計算して返します。 modularity 関数を完成させ、Q3.1でコミュニティを抽出した時のネットワークのモジュラリティを求めてください。

モジュラリティ$Q$は、ネットワークのリンク数を$m$、隣接行列を$A$、ノード$i$の次数を$k_i$とした時、以下で定義されます。

$$Q = \frac{1}{2m} \sum_{ij} \left( A_{ij} - \frac{k_i k_j}{2m} \right) \delta(c_i, c_j)$$

ノード$i$のコミュニティを$c_i$とすると$\delta(c_i, c_j)$はノード$i$と$j$が同じコミュニティの時1、異なるコミュニティの時は0となります。

In [ ]:
```
def modularity(G, communities):

```

## 参考

In [ ]:
```
# networkxのGirvan–Newman法の関数によるコミュニティ抽出
community_generator = nx.community.girvan_newman(G)

# community_generatorはジェネレーターとなっており、
# for文を繰り返すごとにコミュニティ分割を行う
# 最終的には各ノードが1つのコミュニティの状態となる
for communities in community_generator:
    print(communities)
```