네트워크 분석 및 시각화 과제2

2019311195 김지유

1 그래프 설명

Football_graph:
https://networkx.org/documentation/stable//auto_examples/graph/plot_football.html

이번 과제에서 분석한 그래프는 American football games between Division I-A colleges during regular season Fall 2000이다. 그래프의 이해를 돕기 위해 미국 대학 풋볼 리그를 간단히 설명하자면 NCAA(전미 대학 체육 협회)의 주관 하에 8월이나 9월에 시즌이 시작되고, 한 시즌이 정규 게임 기준 12 경기이다. 이 12 정규 경기들 중 8 경기는 같은 컨퍼런스 팀과, 4 경기는 다른 컨퍼런스 팀과 치르게 된다. 여기서 8 개의 컨퍼런스 경기 성적을 갖고 컨퍼런스 챔피언십 진출팀을 가린다.

1.1 Node

해당 그래프는 총 115개의 node로 구성되어 있으며, 각 노드는 Division I-A에 속한 대학교 풋볼팀을 의미한다.

1.2 Edge

해당 그래프는 총 613개의 edge로 구성되어 있으며, 각 edge는 경기를 의미한다. Average of degree는 약 10.6으로, 이 수치는 컨퍼런스 챔피언십에 진출하지 못한 팀은 8 경기, 컨퍼런스 챔피언십에 진출한 팀은 12 경기를 뛴다는 사전 정보에 어느 정도 부합하는 수치이다.

1.3 Node feature

해당 그래프의 node feature는 0이상 11 이하의 정수 값이며, 이 정수 값은 대학교 풋볼팀이 속한 컨퍼런스를 의미한다. 2000년에는 총 12개의 컨퍼런스가 존재했으며, 각 컨퍼런스의 정수 값은 다음과 같다.

0 = Atlantic Coast

1 = Big East

2 = Big Ten

3 = Big Twelve

4 = Conference USA

5 = Independents

6 = Mid-American

7 = Mountain West

8 = Pacific Ten

9 = Southeastern

10 = Sun Belt

11 = Western Athletic

1.4 Edge feature

해당 그래프는 edge feature를 갖지 않는다.

2 네트워크 분석 및 그래프 시각화

2.1 Small world

Small world는 특별히 높은 clustering 값을 가지면서 동시에 특별히 짧은 average shortest path length를 갖는 네트워크를 가리킨다. 이번 과제에서는 sigma 지표를 사용하여 해당 그래프가 얼마나 small world의 특성 가졌는지 평가하였다. 보통 sigma가 1 보다 크면 그래프가 small world의 특성을 가졌다고 판단한다.

```
g_random = sw.random_reference(g, niter=20, seed=2021)
```

```
C = nx.average_clustering(g)
L = nx.average_shortest_path_length(g)
```

```
Cr = nx.average_clustering(g_random)
Lr = nx.average_shortest_path_length(g_random)
```

```
A = (C / Cr)
B = (L / Lr)
```

```
sigma =  A / B
sigma
```

```
4.643321690024574
```

위와 같이 직접 equivalent random graph를 생성한 후, 해당 그래프와 equivalent random graph 간의 비교를 통해 sigma 값을 구한 결과, 약 4.64이 나왔다.

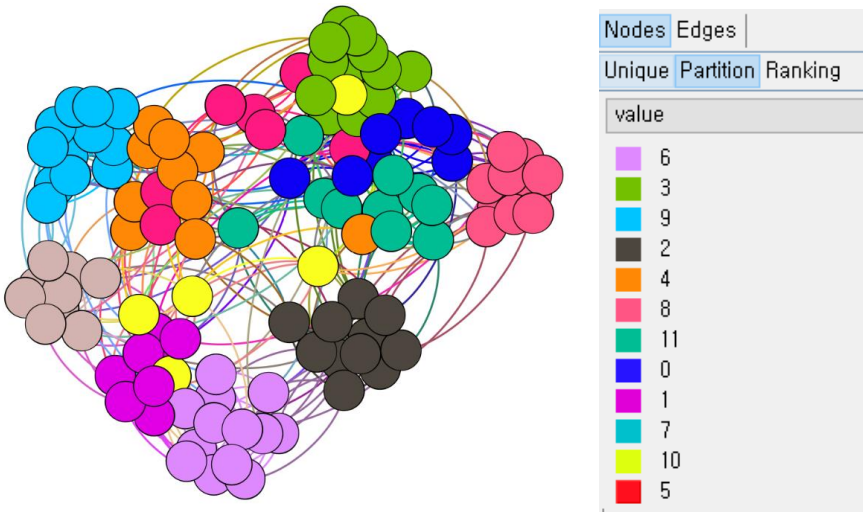
```
from networkx.algorithms import smallworld as sw
```

```
sw_sigma = sw.sigma(g, niter=20, seed=2021)
sw_sigma
```

```
4.539317382484467
```

위와 같이 직접 sigma 값을 구하지 않고 networkx에서 제공하는 small world를 구해주는 알고리즘을 이용해 sigma 값을 구한 결과, 약 4.53이 나왔다.

sigma 값이 1이 넘으면 equivalent random graph와 비교했을 때 해당 그래프가 특별히 높은 clustering 값을 가지면서 동시에 특별히 짧은 average shortest path length를 가지고 있는 것이기 때문에 small world의 특성을 갖는다고 판단한다. 해당 그래프의 sigma 값이 대략 4.5 ~ 4.6 사이이기 때문에 small world의 특성을 굉장히 높게 갖는다고 할 수 있다. 해당 그래프의 sigma 값이 높게 나온 원인을 분석하자면 해당 그래프가 미국 대학 풋볼 '리그'를 표현한 그래프이기 때문이다. 리포트 서두, 그래프 설명에서 언급한 바와 같이 이 그래프는 미국 대학 풋볼 리그 그래프이며, 미국 대학 풋볼 리그의 한 시즌은 정규 게임 기준 12 경기로 이루어졌는데, 이 12 경기 중 8 경기는 컨퍼런스 내에서 치러지며, 8 경기에서 우수한 성적을 기록한 팀이 본인이 소속된 컨퍼런스의 대표팀 지위로서 다른 컨퍼런스의 대표팀과 나머지 4 경기를 치룬다. 전자에서 모든 팀이 컨퍼런스 내에서 치르는 8 경기는 이 그래프의 clustering 값을 높이는 요인이고, 후자에서 컨퍼런스 대표팀이 컨퍼런스 외에서 치르는 4 경기는 이 그래프의 average shortest path length를 짧게 만드는 요인이다.



위의 그래프는 node feature인 대학교 풋볼팀이 속한 컨퍼런스를 기준으로 각 node를 색칠한 그래프이다. 이 그래프에서도 알 수 있듯이 node는 conference를 기준으로 모여 지역성을 띄고 있다. 하지만 각 conference에 속한 팀 중 우수한 팀끼리 치른 경기로 인해 average shortest path length는 짧아질 수 있었다.

2.2 Motif

```
# https://pypi.org/project/netsci/
! pip install netsci
```

```
import numpy as np #netsci가 그래프를 넘파이 매트릭스 형태로 받는다
import netsci.visualization as nsv #시각화 해주는 모듈
import netsci.metrics.motifs as nsm #motif 구해주는 모듈
```

```
# networkx to numpy
#netsci가 그래프를 넘파이 매트릭스 형태로 표현하기 때문에 바꿔준다
g_np = nx.convert_matrix.to_numpy_matrix(g,dtype=np.integer)
g_r_np = nx.convert_matrix.to_numpy_matrix(g_random,dtype=np.integer)
```

```
f_real = nsm.motifs(g_np)[3:]
f_random = nsm.motifs(g_r_np)[3:]
f_real = np.concatenate((np.array([0, 0, 0]), f_real), axis=None)
f_random = np.concatenate((np.array([0, 0, 0]), f_random), axis=None)
```

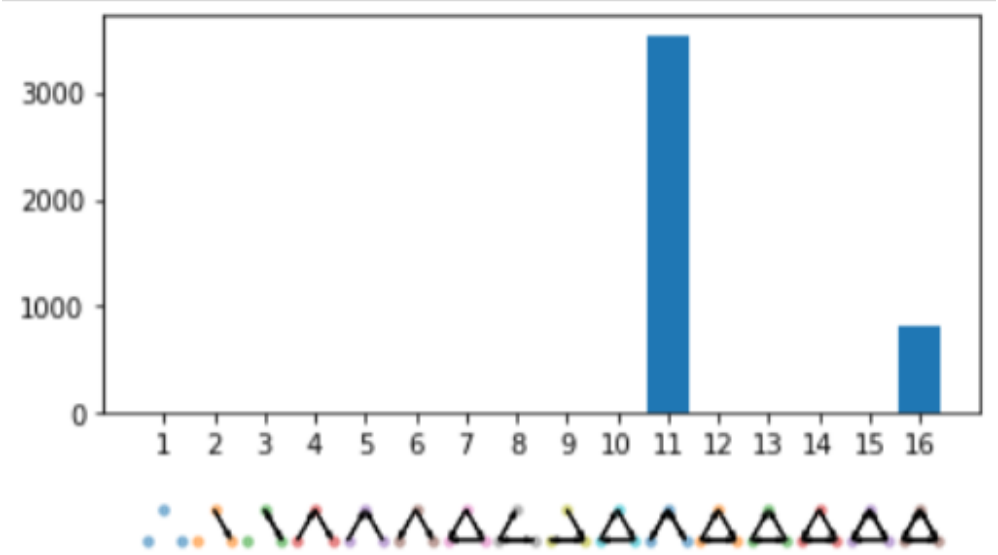
```
[   0    0    0    0    0    0    0    0    0    0 3537    0    0    0
    0  810]
[   0    0    0    0    0    0    0    0    0    0 5499    0    0    0
    0  156]
```

netsci에서는 3개의 노드로 이루어진 총 16가지의 subgraph가 특정 그래프에서 각각 몇 번 등장하는지 구해주는 기능을 제공하고 있다. 이 기능을 이용해 위와 같이 미국 대학교 축구 리그 그래프(실제 그래프)의 motif와 equivalent random graph(랜덤 그래프)의 motif를 각각 알아보았다. netsci에서는 그래프를 넘파이 매트릭스 형태로 다루기 때문에 위에서 만든 실제 그래프와 랜덤 그래프를 넘파이 매트릭스 형태로 변환시켜준 후, nsm.motifs 함수의 인자로 전달해주었다. netsci에서 사용하는 16가지의 subgraph 중 1번째부터 3번째는 수업시간에 배우지 않은 subgraph라 일단 제외하였다가, 추후 subgraph별 등장 회수를 시각화하는 작업을 위해 [0, 0, 0]을 추가해주었다. 그 결과, 실제 그래프에서는 11번째 subgraph가 3537번, 16번째 subgraph가 810번 나타났고, 랜덤 그래프에서는 11번째 subgraph가 5499번, 16번째 그래프가 156번 나타났다.

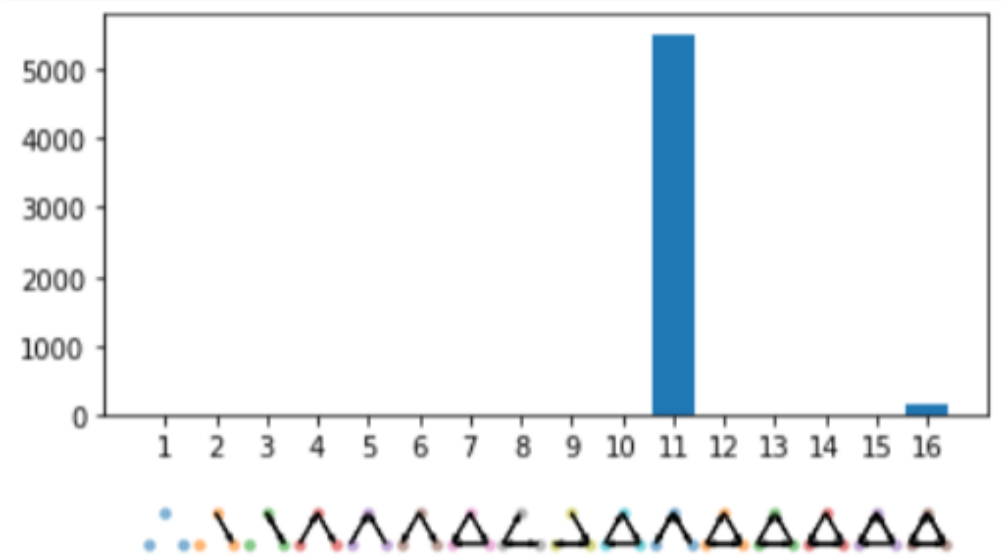
```
# visualization of motifs
nsv.bar_motifs(f_real)
nsv.bar_motifs(f_random)
```

위의 결과를 시각화화면 다음과 같은 그래프가 나온다.

<실제 그래프의 결과>



<랜덤 그래프의 결과>



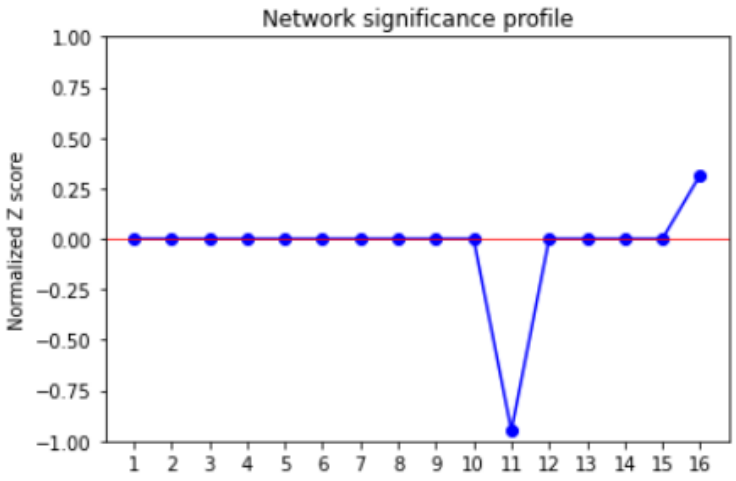
실제 그래프의 motif를 구하기 위해 netsci에서 구해준 결과를 바탕으로 아래와 같이 network significance profile을 구하여 시각화를 하였다.

```
import math

#significant: more frequent than expected
z_list = []
for idx, n_real in enumerate(f_real):
  n_rand = f_random[idx]
  z = (n_real-n_rand) / np.std(f_random)
  z_list.append(z)
```

```
z_list = np.array(z_list)
z_sum = np.sum(z_list**2)
sp_list = []
for idx, z in enumerate(z_list):
  sp = z/math.sqrt(z_sum)
  sp_list.append(sp)
```

```
plt.title("Network significance profile")
plt.ylabel("Normalized Z score")
plt.plot([1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16], sp_list, 'bo-')
plt.axhline(y=0, color='r', linewidth=0.7)
plt.ylim(-1.0, 1.0)
plt.xticks([1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16])
plt.show()
```



그 결과, 11번째 subgraph는 실제 그래프에서 under-represented 되었고, 16번째 subgraph는 실제 그래프에서 over-represented 되었다는 사실을 알 수 있었고, 실제 그래프의 motif는 16번째 subgraph라는 결론을 도출할 수 있었다.

2.3 Graph role

```
! pip install graphrole
```

```
from pprint import pprint
import seaborn as sns

from graphrole import RecursiveFeatureExtractor, RoleExtractor
```

```
# extract features
feature_extractor = RecursiveFeatureExtractor(g)
features = feature_extractor.extract_features()
```

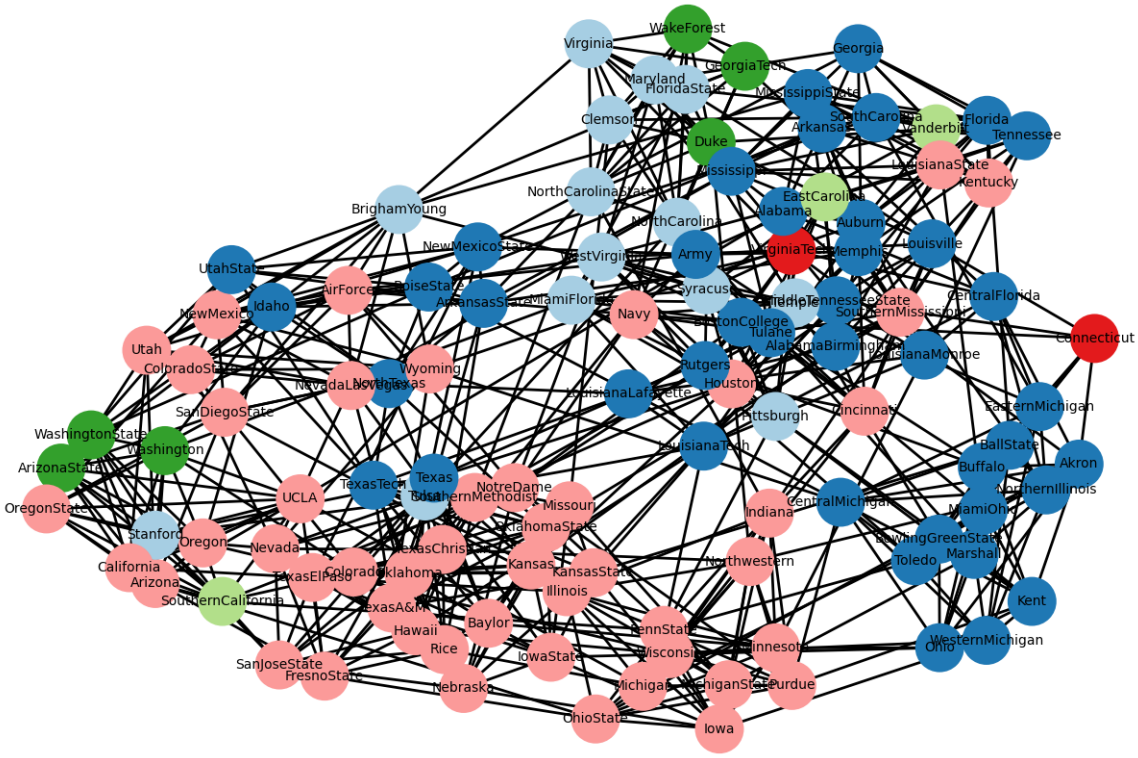
```
# assign node roles
role_extractor = RoleExtractor(n_roles=None)
role_extractor.extract_role_factors(features)
node_roles = role_extractor.roles
```

```
print('\nNode role assignments:')
pprint(node_roles)

print('\nNode role membership by percentage:')
print(role_extractor.role_percentage.round(2))
```

```
# build color palette for plotting
unique_roles = sorted(set(node_roles.values()))
color_map = sns.color_palette('Paired', n_colors=len(unique_roles))
# map roles to colors
role_colors = {role: color_map[i] for i, role in enumerate(unique_roles)}
# build list of colors for all nodes in G
node_colors = [role_colors[node_roles[node]] for node in g.nodes]

# plot graph
plt.figure()
nx.draw(
    g,
    pos=nx.spring_layout(g, seed=42),
    with_labels=True,
    node_color=node_colors,
    font_size = 5
)
plt.show()
plt.savefig('graph_role.png', dpi=200)
```



그래프의 구조적인 role을 찾는 방법 중 하나인 RolX를 통해 각 노드 별로 role을 부여해주었다. RolX를 통해 노드 별 role을 구하기 위해서는 node x node adjacency matrix와 node X feature matrix가 필요하기 때문에 node feature를 추출해주는 과정을 거쳤다. role의 개수를 정하지 않고 role을 추출 결과, 총 5개의 role이 나왔다. 위의 그래프는 추출된 role이 같은 node끼리 같은 색을 할당하여 시각화한 결과이다. 위의 그래프는 node feature인 컨퍼런스가 같은 node끼리 같은 색을 할당하여 시각화한 그래프(혹은 community detection 결과 소속 community가 같은 node끼리 같은 색을 할당하여 시각화한 그래프)와 비교하였을 때 전혀 다른 양상을 띄고 있음을 알 수 있었다. 두 그래프 간의 비교를 통해 role과 community는 다른 개념이며 상호 보완적인 관계임을 배웠다.

2.4 Community detection

```
from networkx.algorithms import community
list(community.asyn_fluidc(g,k=12))
```

networkx에서 제공하는 community detection 알고리즘을 사용하여 해당 그래프의 노드를 총 12개의 community로 분할하였다. 대학교 풋볼팀이 속해 있는 컨퍼런스가 12개가 있고, 대학교 풋볼팀이 각 컨퍼런스를 기준으로 모여 있기 때문에 k를 12로 설정하였다. community detection에 의해 그룹화된 결과와 대학교가 속해 있는 컨퍼러스 기준으로 그룹화된 결과를 비교해보니 전체적으로 유사도가 매우 높았다. 컨퍼러스 기준으로 그룹화된 community 12개와 community detection으로 그룹화된 community 12개

를 일일이 비교해본 결과, 컨퍼런스 기준으로 그룹화된 community 중 5번과 10번은 community detection으로 그룹화했을 때 찢어져 각각 다른 그룹에 속해 있는 경우가 있었다. 예를 들어, 컨퍼러스 기준으로 그롭화된 community에서 10번은 'ArkansasState', 'Idaho', 'LouisianaLafayette', 'LouisianaMonroe', 'MiddleTennesseeState', 'NewMexicoState', 'NorthTexas' 이지만, community detection으로 그룹화된 community에서 10번은 'ArkansasState', 'BoiseState', 'Idaho', 'NewMexicoState', 'NorthTexas', 'UtahState' 이다. 해당 원인은 아래의 그래프가 보여주는 바와 같이 대부분의 대학교 풋볼팀은 conference 기준으로 같이 모여 있는 것에 반해 유독 conference 5번, 10번 소속 대학교 풋볼팀만 흩어져 있기 때문이다.