بسمه تعالی

ارائه دهنده : زهرا منصوری

عنوان : تحلیل شبکه های اجتماعی _ بخش اول

فهرست

شبکه انتخابی

فایل داده شبکه انتخاب شده با نام 'CA-GrQc.txt' در پوشه پروژه وجود دارد. فایل انتخابی مربوط به یک گراف غیر جهت دار می باشد.

نمایش داده انتخاب شده در قالب data frame

با استفاده از متد read_csv از کتابخانه pandas داده انتخابی را در قالب یک data frame لود کردیم. پارامتر های استفاده شده در این متد عبارتند از :

- File_name

ابتدا نام فایل مورد نظر را وارد می کنید.

- Skiprows

به کمک این پارامتر میتوان از خواندن بعضی از خطوط جلوگیری کرد. (خط ۰ و ۱ و ۲ و ۳ فایل، درباره توضیحات گراف داده شده بود و به همین دلیل skip شدند.)

- Delimiter

نحوه جدا کردن اجزای داده را مشخص می کند.

با استفاده از df.columns، نام هر یک از ستون ها را مشخص کردیم.

خروجی به صورت زیر به دست آماده است که ستون اول، اندیس جدول می باشد و ستون های دوم و سوم شماره گره هایی است که با هم اتصال دارند.

```
In [1]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        import random
```

```
In [13]: df = pd.read_csv('CA-GrQc.txt', skiprows=[0,1,2,3], delimiter="\t")
         df.columns =['FromNodeId', 'ToNodeId']
         df
```

Out[13]:

	FromNodeId	ToNodeId
0	3466	5233
1	3466	8579
2	3466	10310
3	3466	15931
4	3466	17038
...	...	...
28974	10154	9224
28975	10154	16830
28976	11113	21723
28977	11113	23836
28978	11113	25050

28979 rows × 2 columns

ماتریس مجاورت گراف

با استفاده از متد crosstab می توان جدول متقاطع از دو ستون dataframe بالا را محاسبه کرد. سپس با استفاده از متد union بین اندیس ها اجتماع گرفته شده و به عنوان اندیس های سطر و ستون ماتریس با استفاده از تابع reindex، اندیس گذاری می شود.

تعداد گره های گراف ۵۲۴۲ است و ماتریس به دست آماده هم یک ماتریس ۵۲۴۲*۵۲۴۲ می باشد.

```
In [14]: df = pd.crosstab(df.FromNodeId, df.ToNodeId)
         idx = df.columns.union(df.index)
         adjacency_matrix = df.reindex(index = idx, columns=idx, fill_value=0)
         adjacency_matrix
```

Out[14]:

	13	14	22	24	25	26	27	28	29	45	...	26173	26176	26178	26180	26181	26190	26191	26193	26194	26196
13	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
22	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
24	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
25	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
26190	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
26191	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
26193	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
26194	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
26196	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

5242 rows × 5242 columns

محاسبه درجه هر گره

برای محاسبه درجه هر گره ابتدا با استفاده از adjacency_matrix.astype(bool).sum(axis=1)، تعداد یک های هر سطر از ماتریس مجاورت را به دست آوردیم. خروجی این ستور مشخص میکند هر گره چه درجه ای دارد.

سپس با استفاده از index. که نمایانگر شماره هر گره است، لیست گره ها را در متغیر NodeId و با استفاده از values . لیست درجات گره ها را در متغیر Degree ذخیره کردیم.

خروجی را نیز در قالب dataframe ایی به نام nodes_degree، شامل ستون های NodeId و Degree به صورت زیر نمایش دادیم.

In [16]:
```
NodeId = adjacency_matrix.astype(bool).sum(axis=1).index
Degree = adjacency_matrix.astype(bool).sum(axis=1).values
nodes_degree = pd.DataFrame({
    'NodeId':NodeId,
    'Degree':Degree
})
nodes_degree
```

Out[16]:

	NodeId	Degree
0	13	4
1	14	1
2	22	6
3	24	4
4	25	1
...	...	...
5237	26190	5
5238	26191	1
5239	26193	5
5240	26194	2
5241	26196	7

5242 rows × 2 columns

توزیع درجات گره

برای محاسبه توزیع درجات گره ها، ابتدا nodes_degree را بر اساس ستون درجه با استفاده از متد sort_values() مرتب کردیم. برای ذخیره اطلاعات از ساختار dictionary استفاده شده است که کلید های آن درجه و مقادیر آن تعداد گره مشاهده شده از آن درجه می باشد.

برای پر کردن این degree_distribution با استفاده از یک حلقه for که روی آیتم های nodes_degree['Degree'] می گردد، چک می شود که اگر آن درجه به عنوان کلید در degree_distribution وجود ندارد، به dictionary اضافه شود و در غیر این صورت یعنی بیشتر از یک گره آن درجه را دارد درنتیجه باید به value آن درجه یک واحد اضافه شود و در نهایت خروجی degree_distribution نشان می دهد که چند گره با درجه d وجود دارد.

```
In [18]: nodes_degree = nodes_degree.sort_values('Degree')
degree_distribution = {}
for degree in nodes_degree['Degree']:
    if degree not in degree_distribution.keys():
        degree_distribution[degree] = 1
    else:
        degree_distribution[degree] += 1
degree_distribution
```

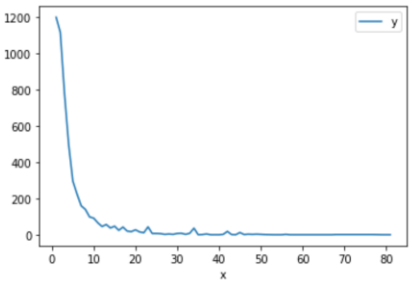
```
Out[18]: {1: 1197,
 2: 1114,
 3: 776,
 4: 498,
 5: 296,
 6: 225,
 7: 160,
 8: 140,
 9: 99,
 10: 92,
 11: 66,
 12: 46,
 13: 57,
 14: 38,
 15: 48,
 16: 25,
 17: 43,
 18: 21,
 19: 18,
```

برای رسم نودار از کتابخانه pyplot استفاده شده است. برای جلوگیری از تکرار در نوشتن کد، برای رسم نمودار تابع plot را نوشتیم که به عنوان ورودی x و y را گرفته و نمودار خطی آن را رسم میکند.

در اینجا x نمایانگر درجات است که از degree_distribution.keys() به دست می آید و y نمایانگر تعداد گره ها با این درجه است که از degree_distribution.values() به دست می آید.

```
In [21]: def plot(x, y):
    data = {'x': x,
            'y': y}
    my_frame = pd.DataFrame(data,columns=['x','y'])
    my_frame.plot(x ='x', y='y', kind = 'line')
    plt.show()
```

```
In [22]: plot(degree_distribution.keys(), degree_distribution.values())
```



متوسط درجه همسایه های هر گره

ابتدا لیستی از همسایه ای هر گره را به دست می آوریم. برای این کار از دو حلقه for که روی سطر و ستون ماتریس مجاورت می گردند، استفاده شده است. سپس چک میشود اگر آن خانه از ماتریس مجاورت برابر یک بود

یعنی آن گره با گره ای که for بیرونی نمایش می دهد، همسایه است و باید به لیست همسایه های آن گره اضافه شود.

برای ذخیره هر گره و همسایه های آن از ساختار dictionary با نام neighbors_of_each_node استفاده شده است که کلید آن گره های گراف و مقادیر آن لیستی از همسایه های آن گره است.

```
In [24]: neighbors_of_each_node={}
for i in adjacency_matrix.columns:
    for j in adjacency_matrix.columns:
        if adjacency_matrix[i][j] == 1:
            if i not in neighbors_of_each_node.keys():
                neighbors_of_each_node[i] = [j]
            else:
                if not isinstance(neighbors_of_each_node[i], list):
                    neighbors_of_each_node[i] = [neighbors_of_each_node[i]]
                neighbors_of_each_node[i].append(j)
neighbors_of_each_node

Out[24]: {13: [13, 7596, 11196, 19170],
 14: [14171],
 22: [106, 11183, 15793, 19440, 22618, 25043],
 24: [3858, 15774, 19517, 23161],
 25: [22891],
 26: [1407, 4550, 11801, 13096, 13142],
 27: [11114, 19081, 24726, 25540],
 28: [7916],
 29: [20243],
 45: [570,
  773,
  1186,
  1653,
  2212,
  2741,
  2952,
  3372,
  4164,
  4180,
```

سپس برای محاسبه متوسط درجه همسایه های هر گره ابتدا با یک حلقه for روی کلید های neighbors_of_each_node که گره های ما هستند، می گردیم و سپس به ازای هر یک از همسایه های آن گره درجه آن گره را به جمع درجات همسایه های دیگر آن گره اضافه می کنیم و در نهایت جمع حاصل را بر طول لیست همسایه های آن گره که همان تعداد همسایه های آن گره می باشد؛ تقسیم می کنیم و جواب را به average_degree_of_neighbors_of_each_node که یک dictionary با کلید هر گره و مقدار میانگین درجه همسایه های آن گره است، اضافه می کنیم.

خروجی average_degree_of_neighbors_of_each_node نشان دهنده میانگین درجه همسایه های هر گره است.

```
In [26]: nodes_degree = adjacency_matrix.astype(bool).sum(axis=1)
average_degree_of_neighbors_of_each_node = {}
for key in neighbors_of_each_node.keys():
    sum = 0
    for v in neighbors_of_each_node[key]:
        sum += int(nodes_degree[v])
    average_degree_of_neighbors_of_each_node[key] = (sum // len(neighbors_of_each_node[key]))
average_degree_of_neighbors_of_each_node

Out[26]: {13: 1,
 14: 1,
 22: 4,
 24: 4,
 25: 1,
 26: 17,
 27: 5,
 28: 3,
 29: 4,
 45: 50,
 46: 53,
 62: 12,
 65: 5,
 70: 4,
 71: 2,
 74: 2,
 75: 6,
 78: 9,
 80: 8,
```

در نهایت نیز برای رسم نمودار ابتدا تعداد گره هایی که دارای میانگین درجه همسایه یکسان هستند را یافته و خروجی را در dic1 ذخیره می کنیم.

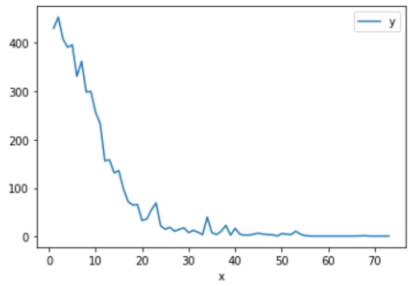
```
In [64]: average_degree_of_neighbors_of_each_node_df = pd.DataFrame({
    'NodeId':average_degree_of_neighbors_of_each_node.keys(),
    'Degree':average_degree_of_neighbors_of_each_node.values()
})
info = average_degree_of_neighbors_of_each_node_df.sort_values('Degree')
dic1 = {}
for item in info['Degree']:
    if item not in dic1.keys():
        dic1[item] = 1
    else:
        dic1[item] += 1
dic1

Out[64]: {1: 429,
 2: 452,
 3: 407,
 4: 390,
 5: 395,
 6: 330,
 7: 361,
 8: 298,
 9: 299,
 10: 256,
 11: 233,
 12: 156,
```

برای رسم نمودار نیز همانطور که در قسمت قبل گفته شد از متد plot استفاده می کنیم.

در اینجا x نمایانگر درجات است که از degree_distribution.keys() به دست می آید و y نمایانگر تعداد گره ها با این درجه است که از degree_distribution.values() به دست می آید.

```
In [24]: plot(dic1.keys(), dic1.values())
```



متوسط طول کوتاهترین مسیرهای هر گره به باقی گره ها

ابتدا ۵ درصد گره ها را به صورت رندوم انتخاب می کنیم. سپس با استفاده از متد BFS_SP که یک گراف و گره هدف و مقصد را می گیرد؛ کوتاه ترین مسیر بین این دو گره را محاسبه کرده و طول آن را به عنوان خروجی بر می گردانیم.

این کار را با استفاده از دو for تو در تو برای لیست five_percent_of_nodes انجام داده و جواب را در shortest_path_avg ذخیره می کنیم.

```
In [16]: def BFS_SP(graph, start, goal):
             explored = []

             queue = [[start]]

             if start == goal:
         #         print("Same Node")
                 return 0

             while queue:
                 path = queue.pop(0)
                 node = path[-1]

                 if node not in explored:
                     neighbours = graph[node]

                     for neighbour in neighbours:
                         new_path = list(path)
                         new_path.append(neighbour)
                         queue.append(new_path)

                         if neighbour == goal:
         #                     print("Shortest path = ", *new_path)
                             return len(new_path) - 1
                     explored.append(node)

         #     print("So sorry, but a connecting"\
         #                   "path doesn't exist :(")
             return 0
```

```
In [31]: five_percent_of_nodes = []
for i in range(int(0.05*len(neighbors_of_each_node))):
    key = random.choice(list(neighbors_of_each_node.keys()))
    if key not in five_percent_of_nodes:
        five_percent_of_nodes.append(key)
```

```
In [47]: shortest_path_avg = {}
for k1 in five_percent_of_nodes:
    sum = 0
    i = 0
    for k2 in five_percent_of_nodes:
        sum += BFS_SP(neighbors_of_each_node,k1,k2)
        i += 1
    shortest_path_avg[k1] = sum // i
shortest_path_avg

Out[47]: {1941: 4,
 24597: 0,
 7585: 0,
 25152: 5,
 12260: 5,
 14972: 5,
```

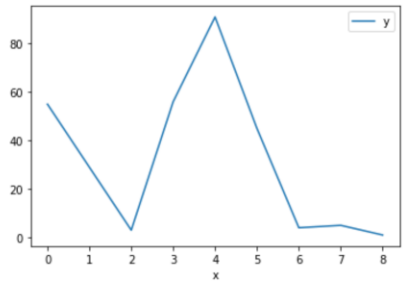
در نهایت نیز برای رسم نمودار ابتدا میانگین کوتاه ترین مسیر هر گره را برحسب تعداد گره هایی که این میانگین را داشتند در dic1 ذخیره کرده و در نهایت با استفاده از متد plot نمودار آن را رسم می کنیم.

در اینجا x نمایانگر می میانگین کوتاه ترین مسیر است که از dic1.keys() به دست می آید و y نمایانگر تعداد گره ها با این میانگین است که از dic1.values() به دست می آید.

```
In [22]: shortest_path_avg_df = pd.DataFrame({
    'NodeId':shortest_path_avg.keys(),
    'avg_path':shortest_path_avg.values()
})
info = shortest_path_avg_df.sort_values('avg_path')
dic1 = {}
for item in info['avg_path']:
    if item not in dic1.keys():
        dic1[item] = 1
    else:
        dic1[item] += 1
dic1

Out[22]: {0: 55, 2: 3, 3: 56, 4: 91, 5: 45, 6: 4, 7: 5, 8: 1}
```

```
In [23]: plot(dic1.keys(), dic1.values())
```



متوسط common neighbor هر گره

برای به دست آوردن همسایه های مشترک هر گره ابتدا روی neighbors_of_each_node.keys() می گردیم و در حلقه داخلی نیز به ازای هر یک از همسایه های neighbors_of_each_node[key]، همسایه مشترک آن گره با گره ای که for خارجی به آن اشاره میکند را با استفاده از متد intersection به دست می آوریم و به common_neighbor اضافه می کنیم.

common_neighbor یک dictionary است که کلید آن به صورت tuple ایی از دو گره ای است که همسایه های مشترک آنها را به دست آورده ایم و مقدار آن نیز همسایه های مشترک آن دو گره می باشد.

```
In [14]: common_neighbor = {}
         for key in neighbors_of_each_node.keys():
             for v in neighbors_of_each_node[key]:
                 common_neighbor[key,v] = set(neighbors_of_each_node[key]).intersection(neighbors_of_each_node[v])
         common_neighbor

Out[14]: {(13, 13): {13, 7596, 11196, 19170},
          (13, 7596): {13},
          (13, 11196): {13},
          (13, 19170): {13},
          (14, 14171): set(),
          (22, 106): {11183, 15793},
          (22, 11183): {106, 15793},
          (22, 15793): {106, 11183, 19440},
          (22, 19440): {15793},
          (22, 22618): {25043},
          (22, 25043): {22618},
          (24, 3858): {19517, 23161},
          (24, 15774): set(),
          (24, 19517): {3858, 23161},
          (24, 23161): {3858, 19517},
          (25, 22891): set(),
          (26, 1407): {4550, 11801, 13096, 13142},
          (26, 4550): {1407, 11801, 13096, 13142},
          (26, 11801): {1407, 4550, 13096, 13142},
```

سپس برای به دست آوردن میانگین تعداد همسایه های مشترک هر گره با استفاده از دو حلقه تو در تو که روی common_neighbor.keys() می چرخند، به ازای کلیدهایی از common_neighbor که عضو اول tuple آنها با هم برابر باشند، طول common_neighbor[k2] را به sum اضافه کرده تا تعداد همسایه های مشترک آن گره را به دست آوریم و در نهایت مجموع را بر تعداد تقسیم کرده تا میانگین تعداد همسایه های مشترک هر گره به دست آید.

خروجی در average_common_neighbor_of_each_node ذخیره می شود. که کلید آن نمایانگر هر گره و مقدار آن نمایانگر میانگین تعداد همسایه های مشترک آن گره است.

```
In [15]: average_common_neighbor_of_each_node = {}
         for k1 in common_neighbor.keys():
             sum = 0
             i = 0
             for k2 in common_neighbor.keys():
                 if k1[0] == k2[0]:
                     sum += len(common_neighbor[k2])
                     i += 1
             average_common_neighbor_of_each_node[k1[0]] = (sum // i)
         average_common_neighbor_of_each_node

Out[15]: {13: 1,
          14: 0,
          22: 1,
          24: 1,
          25: 0,
          26: 4,
          27: 3,
          28: 0,
          29: 0,
          45: 39,
          46: 41,
          62: 1,
          65: 2,
          70: 1,
          71: 0,
          74: 1,
          75: 0,
          78: 1,
          80: 1,
```

در نهایت نیز برای رسم نمودار ابتدا میانگین همسایه های مشترک هر گره را برحسب تعداد گره هایی که این میانگین را داشتند در my_dic ذخیره کرده و در نهایت با استفاده از متد plot نمودار آن را رسم می کنیم.

در اینجا x نمایانگر میانگین همسایه های مشترک است که از my_dic.keys() به دست می آید و y نمایانگر تعداد گره ها با این میانگین است که از my_dic.values() به دست می آید.

```
In [14]: average_common_neighbor_of_each_node = pd.DataFrame({
             'NodeId':average_common_neighbor_of_each_node.keys(),
             'avg_common_neighbor':average_common_neighbor_of_each_node.values()
         })
         info2 = average_common_neighbor_of_each_node.sort_values('avg_common_neighbor')
         my_dic = {}
         for item in info2['avg_common_neighbor']:
             if item not in my_dic.keys():
                 my_dic[item] = 1
             else:
                 my_dic[item] += 1
         my_dic

Out[14]: {0: 1648,
          1: 1599,
          2: 991,
          3: 386,
          4: 139,
          5: 66,
          6: 40,
```

In [36]:
```
plot(my_dic.keys(), my_dic.values())
```