۲۰۱۷

تحلیل روابط بین اعضای شبکه اجتماعی سایت طرفداری

جمشید مظفری

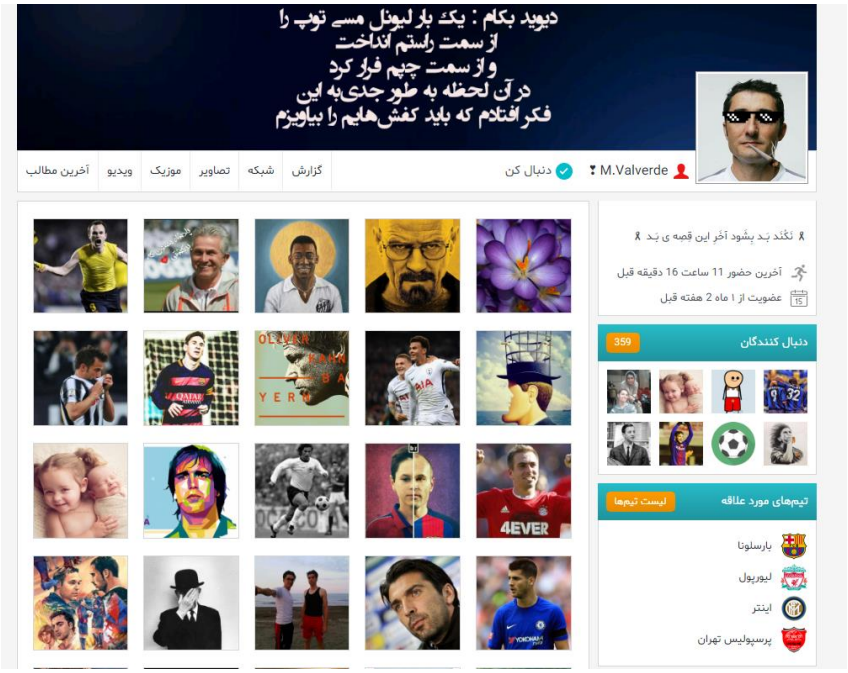
مقدمه:

در این تمرین قصد داریم، در ابتدا شبکه بین اعضای شبکه اجتماعی سایت طرفداری را به کمک کدی به زبان پایتون[1] استخراج کنیم و آن را با قالب مناسبی ذخیره‌سازی کنیم. سپس شبکه استخراج شده را به عنوان ورودی به تمرین اول دهیم و معیارهای اشاره شده در آن تمرین را برای این شبکه محاسبه کنیم. آدرس این شبکه اجتماعی در زیر آورده شده است.

https://www.tarafdari.com

این سایت به صورت یک سایت خبری است که تعدادی نویسنده اصلی، خبرهای اصلی را منتشر می‌کنند. اما در کنار این نویسنده‌ها، اعضای این سایت نیز می‌توانند خبرهای خود را مانند یک پست در سایت منتشر کنند. این سایت اجازه می‌دهد که کاربران مانند شبکه‌های اجتماعی، تعدادی دنبال‌کننده داشته باشند. این ویژگی، این سایت را به یک شبکه اجتماعی تبدیل کرده است. در این تمرین، این شبکه را با کمک کد نوشته شده در فایل **Crawler.py** استخراج می‌کنیم و به صورت قالب لیست پیوندی ذخیره‌سازی می‌کنیم. سپس برای محاسبه معیارها از همان کد تمرین قبل در فایل **Main.py** استفاده می‌کنیم.

در فایل **Crawler.py**، از کتابخانه **BeautifulSoup** برای پردازش صفحات وب استفاده می‌کنیم. منطق کد این فایل، به این شکل است که ابتدا یک دیکشنری[2] برای ذخیره‌سازی ساختار گراف تعریف می‌کنیم. کلید این دیکشنری، شناسه مرتبط با عضو شبکه و مقدار آن لیستی از شناسه‌های اعضایی است که این عضو را دنبال می‌کنند. به صورت کلی، هدف این کد، کامل کردن این دیکشنری بر اساس روابط بین اعضای شبکه اجتماعی است. برای این عمل، از الگوریتم جستجوی اول سطح[3] استفاده می‌کنیم، به این شکل که به ازای هر عضو، ۴۰ عضو دنبال کننده این عضو را به صورت سطحی بررسی می‌کنیم و همین عمل را برای اعضای مرتبط نیز انجام می‌دهیم. این عمل را تا زمانی که ۱۰۰۰ عضو بررسی شوند ادامه خواهیم داد. به صورت پیش‌فرض از کاربری با نام **M.Valverde** ❣ شروع به ساخت شبکه می‌کنیم که در زیر نمایی از صفحه این کاربر نشان داده شده است.



[1] Python
[2] Dictionary
[3] Breadth-first search

این کد، با توجه به دنبال‌کنندگانی که در شکل بالا نیز مشخص شده است(گوشه پایین سمت چپ)، صفحات مرتبط با هر یک را شناسایی کرده و هر یک را جداگانه مورد پردازش قرار می‌دهد تا زمانی که شرط پایان الگوریتم برقرار شود. در خروجی کد نیز مشخص شده است که در هر مرحله چه صفحه‌ای در حال پردازش است. خروجی **Is exist** نشان دهنده اعضایی است که جزء دنبال‌کنندگان صفحه مورد پردازش فعلی است اما قبلا مورد پردازش قرار گرفته است یا در صف قرار دارد تا بعدا مورد بررسی قرار گیرد. در آخر نیز این دیکشنری با قالب مناسب در فایلی با نام **adjList** ذخیره می‌شود. کد این فایل در زیر آورده شده است.

```
import urllib.request as urllib2
from queue import LifoQueue
from bs4 import BeautifulSoup

adjList = dict()

queue = LifoQueue()
queue.put("426202")

while len(adjList) <= 1000:
    userID = queue.get()
    recUrl = "https://www.tarafdari.com/user/" + userID + "/follow/followers"
    request = urllib2.Request(recUrl)
    url = urllib2.urlopen(request)
    soup = BeautifulSoup(str(url.read()), 'html.parser')
    followers = soup.find(id='page-
user').next.next_sibling.next_sibling.next_sibling.find('div').find_all('a')
    print(len(adjList), ": ", recUrl)
    for tag in followers:
        tagStr = str(tag['href'])
        adjID = tagStr[tagStr.rfind('/') + 1:]
        if userID not in adjList.keys():
            adjList[userID] = set()
        if adjID not in adjList.keys():
            queue.put(adjID)
        else:
            print("Is exist: ", adjID)
        adjList[userID].add(adjID)

fout = open("adjList.txt", 'w')
for key in adjList:
    for val in adjList[key]:
        str = key + " " + val
        if key != val:
            fout.write(str + "\n")
fout.close()
```

قسمتی از خروجی کد بالا که برای ساخت شبکه اجرا شده است به شکل زیر است.

```
0 :  https://www.tarafdari.com/user/426202/follow/followers

1 :  https://www.tarafdari.com/user/265518/follow/followers

Is exist:  426202

2 :  https://www.tarafdari.com/user/381396/follow/followers

3 :  https://www.tarafdari.com/user/379761/follow/followers

Is exist:  426202

4 :  https://www.tarafdari.com/user/244319/follow/followers

Is exist:  379761

5 :  https://www.tarafdari.com/user/408170/follow/followers
```

پس از اتمام این پردازش، فایل خروجی به شکل زیر تولید می‌شود.

```
426202 354132
426202 434885
426202 384160
426202 384390
426202 255679
426202 127304
…
```

حال این فایل را به عنوان ورودی به کد تمرین قبل به نام **Main.py** می‌دهیم و معیارهای مختلف را برای این شبکه استخراج شده محاسبه می‌کنیم.

در این گزارش نتایج حاصل از معیارهای زیر را محاسبه و ارائه کنیم.

۱- درجه گراف
۲- توزیع درجه
۳- درجه متوسط گراف
۴- توزیع مشترک درجه
۵- فاصله
۶- قطر
۷- ضریب کلاسترینگ
۸- نزدیکی
۹- همبستگی
۱۰- بینیت

در ابتدا بعضی از ملزومات که برای تعیین بعضی از معیارها نیاز است را آماده می‌کنیم. در کد زیر ابتدا گراف را از فایل ورودی **adjList.txt** خوانده و با کمک توابع آماده **networkX**، ورودی را به ساختار انتزاعی گراف مناسب با این کتابخانه تبدیل می‌کنیم. سپس درجه گره‌ها را به صورت یک دیکشنری[4](**degrees**) ارائه می‌کنیم. بنابر نیاز برای بعضی تحلیلات، یک دیشکنری دیگر تعریف می‌کنیم که عکس دیکشنری مرتبط با درجه گره‌ها است(**swapDegrees**). به بیان بهتر، در دیکشنری جدید درجه هر گره به عنوان کلید و گره‌هایی با درجه کلید به عنوان مقدار این کلیدها در نظر گرفته می‌شود.

```
g = nx.Graph(nx.read_edgelist("adjList.txt","#")).to_undirected()
degrees = dict(nx.degree(g))
numOfNodes = len(degrees)
swapDegrees = dict()
for v in set(degrees.values()):
    for k in degrees.keys():
        if degrees[k] == v:
            if v not in swapDegrees.keys():
                swapDegrees[v] = []
            swapDegrees[v].append(k)
print("Degrees: ", degrees)
print("Swap Degrees: ", swapDegrees)
```

[4] Dictionary

خروجی قطعه کد بالا به صورت زیر است:

Degrees: {'426202': 239, '354132': 15, '434885': 167,...}

Swap Degrees: {1: ['258042', '116662', '387741', '382279',...}

درجه گراف:

برای محاسبه درجه گراف از قطعه کد زیر استفاده می‌شود که میانگین گره‌ها را محاسبه می‌کند.

```
#Compute degree of graph
degreeOfGraph = 0
for v in degrees.values():
    degreeOfGraph += v
degreeOfGraph /= numOfNodes
print("Degree of Graph: ",degreeOfGraph)
```

خروجی قطعه کد بالا به صورت زیر است:

Degree of Graph: 16.217196702002354

توزیع درجه:

برای محاسبه توزیع درجه هر گره از کد زیر استفاده می‌کنیم. خروجی این بخش به این شکل است که این احتمال را برای هر **k** محاسبه کرده و به صورت جداگانه ارائه می‌کند.

```
#Compute degree distribution
Pk = dict()
for k in swapDegrees.keys():
    Pk[k] = len(swapDegrees[k])/numOfNodes
print("P(k): ",Pk)
```

خروجی قطعه کد بالا، دیکشنری به شکل زیر است:

P(k): {1: 0.3083627797408716, 2: 0.13286219081272085, 3: 0.0817432273262662,... }

درجه متوسط گراف:

برای محاسبه درجه متوسط گراف از قطعه کد زیر استفاده می‌کنیم. این قطعه کد از خروجی توزیع درجه استفاده می‌کند و با چند عمل ریاضی پایه‌ای محاسبه می‌شود.

```
#Compute average graph degree
averageGraphDegree = 0
for k in swapDegrees.keys():
    averageGraphDegree += k*Pk[k]
print("<k>: ",averageGraphDegree)
```

خروجی قطعه کد بالا به صورت زیر است که با یک عدد نمایش داده می‌شود.

<k>: 16.21719670200235

توزیع مشترک درجه:

این معیار به صورت کلی نیست و به ازای دو ورودی محاسبه می‌شود. برای این معیار باید درجه گره‌های دو سر یال را در ابتدا مشخص کنیم. سپس قطعه کد زیر توزیع مشترک درجه متناسب با ورودی‌ها را ارائه خواهد داد.

```
#Compute joint degree distribution
k1 = int(input("Enter k1:"))
```

```
k2 = int(input("Enter k2:"))
k1List = swapDegrees[k1]
k2List = swapDegrees[k2]
mk1k2 = 0
for i in k1List:
    for j in k2List:
        if j in g.edge[i]:
            mk1k2 += 1
pk1k2 = mk1k2/len(g.edge)
print("P(k1,k2): ",pk1k2)
```

خروجی قطعه کد بالا به ازای ورودی‌های ۲ و ۲ به شکل زیر است:

Enter k1:2

Enter k2:2

P(k1,k2): 0.0

فاصله:

برای محاسبه این معیار از تابع **eccentricity** استفاده می‌کنیم. این تابع ابتدا فواصل بین یک گره تا بقیه گره‌ها را محاسبه می‌کند و از بین تمام فواصل به دست آمده برای یک گره، بیشترین آن را به عنوان نماینده برای آن‌گره انتخاب می‌کند.

```
#Compute distance of nodes:
print("Distance of nodes: ",nx.eccentricity(g))
```

خروجی قطعه کد بالا به شکل زیر است:

Distance of nodes: {'426202': 4, '354132': 4, '434885': 4, '384160': 4, ...}

قطر:

قطر گراف معادل بیشترین فاصله از بین فواصل موجود است. این معیار به شکل زیر محاسبه می‌شود:

```
#Compute diameter of graph
print("Diameter of graph: ",nx.diameter(g))
```

خروجی قطعه کد بالا عددی به شکل زیر است:

Diameter of graph: 6

ضریب کلاسترینگ:

برای محاسبه ضریب کلاسترینگ یک گراف، از توابع آماده این کتابخانه استفاده می‌کنیم.

```
#Compute clustering coefficient
print("Clustering coefficient: ",nx.average_clustering(g))
```

که خروجی این کد به شکل زیر است:

Clustering coefficient: 0.06794434172935303

نزدیکی:

برای محاسبه معیار نزدیکی گراف نیز از توابع آماده استفاده می‌کنیم.

```
#Compute closeness of nodes
print("Closeness of nodes: ",nx.closeness_centrality(g))
```

خروجی این قطعه کد به شکل دیکشنری ارائه می‌شود که میزان نزدیکی برای هر گره را مشخص کرده است.

Closeness of nodes: {'426202': 0.4168549258422552, '354132': 0.33544103699019917,...}

همبستگی:

برای محاسبه همبستگی هر گره از قطعه کد زیر استفاده می‌کنیم.

```
#Compute coreness of nodes
print("Coreness of nodes: ",nx.core_number(g))
```

خروجی این قطعه کد در زیر آمده است.

Coreness of nodes: {'426202': 27, '354132': 15, '434885': 27, '384160': 27,...}

بینیت:

برای محاسبه معیار بینیت برای هر گره از توابع آماده این کتابخانه استفاده می‌کنیم.

```
#Compute betweenness of nodes
print("Betweenness of nodes: ",nx.betweenness_centrality(g))
```

خروجی این کد به شکل زیر است که یک دیکشنری را ارائه می‌کند.

Betweenness of nodes: {'426202': 0.014778624543798876,...}