

به نام خدا



درس شبکه پیچیده پویا

پروژه پایانی شبکه های پیچیده پویا

مدرس : دکتر رحمانی

سارا سادات یونسی - ۵۳۰۳۳۵۸۹

فهرست

۳	گام اول
۱۱	گام دوم
۱۳	گام سوم
۱۶	گام چهارم
۲۲	گام پنجم

گام اول

الف) گراف جهت دار متناظر با داده ورودی را تشکیل دهید.

```
G = nx.DiGraph()  
G.add_edges_from([
```

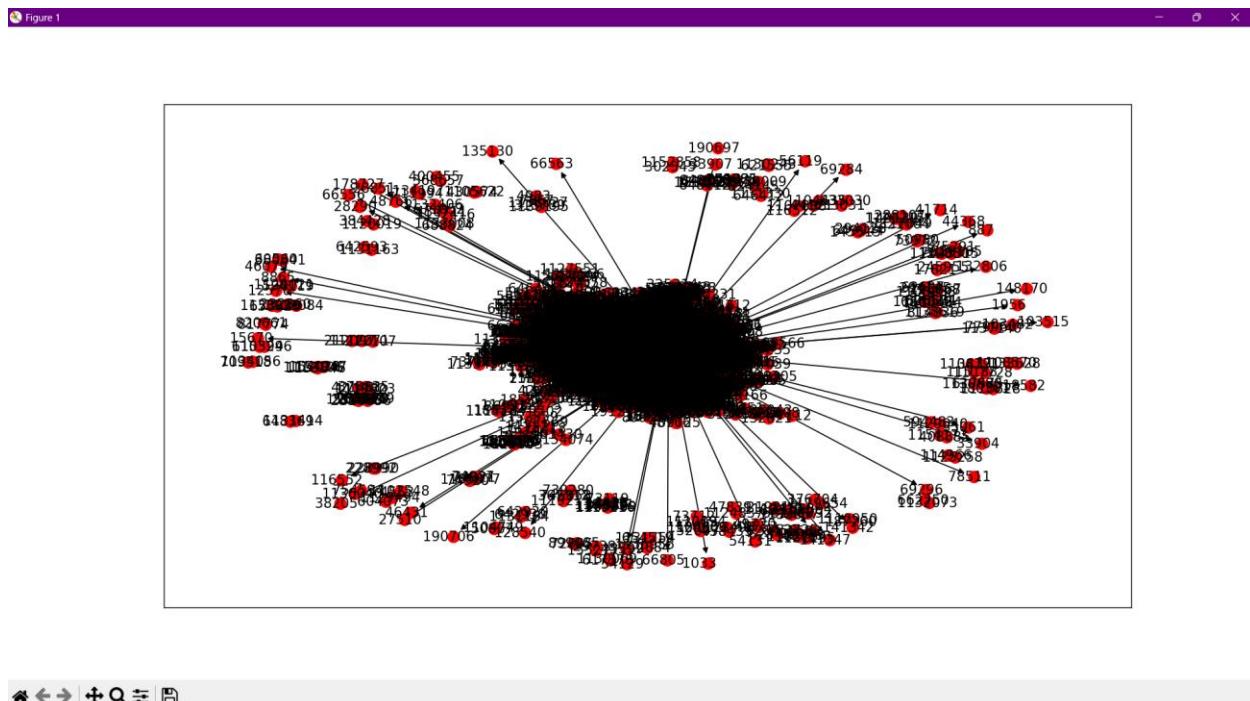
```
plt.figure()
plt.grid(True)

plt.xlim([0, 2*10**2])
plt.close()

pos = nx.spring_layout(G)

nx.draw_networkx_nodes(G, pos, node_size=100, node_color='red')
nx.draw_networkx_edges(G, pos, edgelist=G.edges(), edge_color='black')
nx.draw_networkx_labels(G, pos)

plt.show()
```



ب) اطلاعات آماری مربوط به گراف را ارائه نمایید.

```

print("-----")
print("1.View Of network Information")
print("-----")
print(G)
print('1.1.Number of nodes', len(G.nodes))
print('1.2.Number of edges', len(G.edges))
print('1.3.Average degree', sum(dict(G.degree).values()) / len(G.nodes))
print('1.4.Density of graph', ((len(G.edges)) / (len(G.nodes) * (len(G.nodes) - 1))))
print("-----")

```

```

1.View Of network Information
-----
DiGraph with 2751 nodes and 10556 edges
1.1.Number of nodes 2751
1.2.Number of edges 10556
1.3.Average degree 7.674300254452926
1.4.Density of graph 0.0013953273189914411
-----
```

ج) معیارهای مرکزیت را برای گره ها گزارش و بررسی کنید و ۵ گره با مرکزیت بالا را برای هر کدام از معیارها گزارش دهید.

در شکل ها ۵ تای اول مرکزیت برای هر کدام از معیار ها مشخص است.

```

-----")
print("2.Deegree Centrality")
print("-----")
print('2.1.Deegree Centrality', nx.degree_centrality(G) ,5)
print('2.2.InDeegree Centrality', nx.out_degree_centrality(G))
print('2.3.OutDeegree Centrality', nx.in_degree_centrality(G))
print("-----")
print("3.Clossness Centrality")
print("-----")
print('3.1.Closeness Centrality', nx.closeness_centrality(G) ,5)
print("-----")
print("4.Betweenness Centrality")
print("-----")
print('4.1.Betweenness Centrality', nx.betweenness_centrality(G) )
print("-----")

```

```

print("5.Eigenvector Centrality")
print("-----")
print('5.1.Closeness Centrality', nx.eigenvector_centrality(G) ,5)
print('5.2.Eigenvector Centrality Numpy', nx.eigenvector_centrality_numpy(G))
print("-----")
print("6.Katz Centrality")
print("-----")
print('6.2.Katz Centrality Numpy', nx.katz_centrality_numpy(G))

```

Degree Centrality

35, 1114331,

degree_centrality

`degree_centrality(G)`

[\[source\]](#)

Compute the degree centrality for nodes.

The degree centrality for a node v is the fraction of nodes it is connected to.

Parameters:

`G : graph`

A networkx graph

Returns:

`nodes : dictionary`

Dictionary of nodes with degree centrality as the value.

 See also

[betweenness_centrality](#), [load_centrality](#), [eigenvector_centrality](#)

Notes

The degree centrality values are normalized by dividing by the maximum possible degree in a simple graph $n-1$ where n is the number of nodes in G .

For multigraphs or graphs with self loops the maximum degree might be higher than $n-1$ and values of degree centrality greater than 1 are possible.

```

2.2.InDegree Centrality {'35 ': 0.061090909090909085, '1033 ': 0.0, '103482 ': 0.0, '103515 ': 0.0, '10506
79 ': 0.0014545454545454545, '1103960 ': 0.00181818181818182, '1103985 ': 0.0007272727272727272, '1109199 ': 0
.0010909090909090907, '1112911 ': 0.0014545454545454545, '1113438 ': 0.00181818181818182, '1113831 ': 0.001090
9090909090907, '1114331 ': 0.00181818181818182, '1117476 ': 0.0010909090909090907, '1119505 ': 0.000727272727272
727272, '1119708 ': 0.00181818181818182, '1120431 ': 0.0014545454545454545, '1123756 ': 0.001454545454545454545,
'1125386 ': 0.0014545454545454545, '1127430 ': 0.0007272727272727272, '1127913 ': 0.0014545454545454545, '11282
04 ': 0.0007272727272727272, '1128227 ': 0.0014545454545454545, '1128314 ': 0.0010909090909090907, '1128453 ': 0
.0014545454545454545, '1128945 ': 0.0003636363636363636, '1128959 ': 0.0007272727272727272, '1128985 ': 0.001454
5454545454545, '1129018 ': 0.0007272727272727272, '1129027 ': 0.0007272727272727272, '1129573 ': 0.00109090909090
90907, '1129683 ': 0.00181818181818182, '1129778 ': 0.0014545454545454545, '1130847 ': 0.0010909090909090907,
'1130856 ': 0.0007272727272727272, '1131116 ': 0.0014545454545454545, '1131360 ': 0.0014545454545454545, '11315

```

in_degree_centrality(G) [source]

Compute the in-degree centrality for nodes.

The in-degree centrality for a node v is the fraction of nodes its incoming edges are connected to.

Parameters

`G(graph)` – A NetworkX graph

Returns

`nodes` – Dictionary of nodes with in-degree centrality as values.

Return type

`dictionary`

Raises

`NetworkXNotImplemented`: – If G is undirected.

See also

`degree_centrality()`, `out_degree_centrality()`

Notes

The degree centrality values are normalized by dividing by the maximum possible degree in a simple graph $n-1$ where n is the number of nodes in G .

For multigraphs or graphs with self loops the maximum degree might be higher than $n-1$ and values of degree centrality greater than 1 are possible.

```
5454545454545, '192734': 0.00109090909090907, '20593': 0.00109090909090907, '27230': 0.001454  
2.3. OutDegree Centrality {'35': 0.0610909090909085, '1033': 0.0003636363636363636, '103482': 0.0003636  
3636363636363636, '103515': 0.00036363636363636363, '1050679': 0.0014545454545454545, '1103960': 0.001818181818181  
18182, '1103985': 0.0007272727272727272, '1109199': 0.00109090909090907, '1112911': 0.0014545454545454545,  
'1113438': 0.00181818181818182, '1113831': 0.00109090909090907, '1114331': 0.00181818181818182, '111747  
6': 0.00109090909090907, '1119505': 0.0007272727272727272, '1119708': 0.00181818181818182, '1120431': 0.  
0014545454545454545, '1123756': 0.0014545454545454545, '1125386': 0.0014545454545454545, '1127430': 0.0007272  
727272727272, '1127913': 0.0014545454545454545, '1128204': 0.0007272727272727272, '1128227': 0.0014545454545454545  
54545, '1128314': 0.00109090909090907, '1128453': 0.0014545454545454545, '1128945': 0.0003636363636363636,  
'1128959': 0.0007272727272727272, '1128985': 0.0014545454545454545, '1129018': 0.0007272727272727272, '112902  
7': 0.0007272727272727272, '1129573': 0.00109090909090909, '1129683': 0.00181818181818182, '1129778': 0.  
0014545454545454545, '1130847': 0.00109090909090907, '1130856': 0.0007272727272727272, '1131116': 0.0014545  
45454545454545, '1131360': 0.0014545454545454545, '1131557': 0.0007272727272727272, '1131752': 0.0007272727272727
```

`out_degree_centrality(G)` [source]

Compute the out-degree centrality for nodes.

The out-degree centrality for a node v is the fraction of nodes its outgoing edges are connected to.

Parameters

`G (graph)` – A NetworkX graph

Returns

`nodes` – Dictionary of nodes with out-degree centrality as values.

Return type

`dictionary`

Raises

`NetworkXNotImplemented`: – If G is undirected.

See also

`degree_centrality()`, `in_degree_centrality()`

Notes

The degree centrality values are normalized by dividing by the maximum possible degree in a simple graph $n-1$ where n is the number of nodes in G .

For multigraphs or graphs with self loops the maximum degree might be higher than $n-1$ and values of degree centrality greater than 1 are possible.

2. Degree Centrality

```

2.1. Deegree Centrality {'35' : 0.122181818181817, '1033' : 0.0003636363636363636, '103482' : 0.000363636363636363636363636, '103515' : 0.000363636363636363636, '1050679' : 0.002909090909090909, '1103960' : 0.003636363636363636364, '1103985' : 0.0014545454545454545, '1109199' : 0.00218181818181815, '1112911' : 0.002909090909090909, '1113438' : 0.003636363636363636364, '1113831' : 0.0021818181818181815, '1114331' : 0.0036363636363636364, '1117476' : 0.0021818181818181815, '1119505' : 0.0014545454545454545, '1119708' : 0.0036363636363636364, '1120431' : 0.00290909090909, '1123756' : 0.002909090909090909, '1125386' : 0.002909090909090909, '1127430' : 0.0014545454545454545, '1127913' : 0.002909090909090909, '1128204' : 0.0014545454545454545, '1128227' : 0.002909090909090909, '1128314' : 0.00218181818181815, '1128453' : 0.002909090909090909, '1128945' : 0.0007272727272727272, '1128959' : 0.0014545454545454545, '1128985' : 0.002909090909090909, '1129018' : 0.0014545454545454545, '1129027' : 0.0014545454545454545, '1129573' : 0.00218181818181815, '1129683' : 0.0036363636363636364, '1129778' : 0.002909090909090909, '1130847' : 0.00218181818181815, '1130856' : 0.0014545454545454545, '1131116' : 0.002909090909090909, '

```

Closeness Centrality

networkx.algorithms.centrality.closeness_centrality

```
closeness_centrality(G, u=None, distance=None, wf_improved=True) [source]
```

Compute closeness centrality for nodes

Closeness centrality $\frac{1}{d}$ of a node u is the reciprocal of the average shortest path distance to u over all $n-1$ reachable nodes.

$$C(u) = \frac{n-1}{\sum_{v=1}^{n-1} d(v, u)},$$

where $d(v, u)$ is the shortest-path distance between v and u , and n is the number of nodes that can reach u . Notice that the closeness distance function computes the incoming distance to u for directed graphs. To use outward distance, act on `G.reverse()`.

Notice that higher values of closeness indicate higher centrality.

Wasserman and Faust propose an improved formula for graphs with more than one connected component. The result is "a ratio of the fraction of actors in the group who are reachable, to the average distance" from the reachable actors². You might think this scale factor is inverted but it is not. As is, nodes from small components receive a smaller closeness value. Letting $|N|$ denote the number of nodes in the graph,

$$C_{WF}(u) = \frac{n-1}{N-1} \frac{n-1}{\sum_{v=1}^{n-1} d(v, u)},$$

3.Closeness Centrality

```
3.1.Closeness Centrality {'35 ': 0.21928552135901627, '1033 ': 0.1765775232866528, '103482 ': 0.176577523286  
6528, '103515 ': 0.1765775232866528, '1050679 ': 0.20410529014331433, '1103960 ': 0.19908868274582558, '1103985  
' : 0.17660208221530535, '1109199 ': 0.1920836790125378, '1112911 ': 0.1879642669469259, '1113438 ': 0.181106582  
8190697, '1113831 ': 0.17656039144990987, '1114331 ': 0.1767133539060766, '1117476 ': 0.17669943727716605, '1119  
505 ': 0.17649095056599184, '1119708 ': 0.17686658162899688, '1120431 ': 0.1845324002422448, '1123756 ': 0.17838  
523251275676, '1125386 ': 0.19074466161229742, '1127430 ': 0.17660208221530535, '1127913 ': 0.1767411937412725,  
'1128204 ': 0.17656039144990987, '1128227 ': 0.176546498901995, '1128314 ': 0.1765742861844223, '1128453 ': 0.17  
66994327716605, '1128945 ': 0.17646318950416473, '1128959 ': 0.1770340424921457, '1128985 ': 0.17696422809942,  
'1129018 ': 0.1770340424921457, '1129027 ': 0.17660208221530535, '1129573 ': 0.18064000117103732, '1129683 ': 0.  
.18008904844252785, '1129778 ': 0.17660208221530535, '1130847 ': 0.17650483437267578, '1130856 ': 0.176490950565  
99184, '1131116 ': 0.17656039144990987, '1131360 ': 0.17675511694859417, '1131557 ': 0.17651872036389382, '11317  
52 ': 0.1764770689433266, '1133196 ': 0.17669943727716605, '1133338 ': 0.17660208221530535, '1136814 ': 0.176685  
5228400232, '1137466 ': 0.17646318950416473, '1152421 ': 0.17688052459956283, '1152508 ': 0.17646318950416473, '  
1153065 ': 0.17660208221530535, '1153280 ': 0.17688052459956283, '1153577 ': 0.1767829699452769, '1153853 ': 0.1  
7682476590318025, '1153943 ': 0.18337115516062882, '1154176 ': 0.176546498901995, '1154459 ': 0.1768805245995628  
3, '116552 ': 0.1765775232866528, '12576 ': 0.1765775232866528, '128540 ': 0.1765775232866528, '132806 ': 0  
.1765775232866528, '135130 ': 0.1765775232866528, '141342 ': 0.1765775232866528, '141347 ': 0.1765775232866528  
8, '148170 ': 0.1765775232866528, '15670 ': 0.1765775232866528, '1688 ': 0.1765775232866528, '175291 ': 0  
.1765775232866528, '178727 ': 0.1765775232866528, '18582 ': 0.1765775232866528, '190697 ': 0.1765775232866528,  
'190706 ': 0.1765775232866528, '1956 ': 0.1765775232866528, '197054 ': 0.17715984639127155, '198443 ': 0.
```

Beetweeness Centrality

betweenness_centrality

```
betweenness_centrality(G, k=None, normalized=True, weight=None,  
endpoints=False, seed=None) [source]
```

Compute the shortest-path betweenness centrality for nodes.

Betweenness centrality of a node v is the sum of the fraction of all-pairs shortest paths that pass through v

$$c_B(v) = \sum_{s,t \in V} \frac{\sigma(s,t|v)}{\sigma(s,t)}$$

where V is the set of nodes, $\sigma(s,t)$ is the number of shortest (s,t) -paths, and $\sigma(s,t|v)$ is the number of those paths passing through some node v other than s,t . If $s=t$, $\sigma(s,t)=1$, and if $v \in s,t$, $\sigma(s,t|v)=0$ [2].

Parameters:

G : graph

A NetworkX graph.

k : int, optional (default=None)

If k is not None use k node samples to estimate betweenness. The value of $k \leq n$ where n is the number of nodes in the graph. Higher values give better approximation.

normalized : bool, optional

If True the betweenness values are normalized by $2/((n-1)(n-2))$ for graphs, and $1/((n-1)(n-2))$ for directed graphs where n is the number of nodes in G.

4.Betweenness Centrality

```
4.1.Betweenness Centrality {'35 ': 0.20615757147796815, '1033 ': 0.0, '103482 ': 0.0, '103515 ': 0.0, '1050  
679 ': 0.025526573512184524, '1103960 ': 0.015381676993584881, '1103985 ': 6.544909411059364e-05, '1109199 ': 0.  
01073775303619908, '1112911 ': 0.004007614325474926, '1113438 ': 0.0046334921231266195, '1113831 ': 0.0006623234  
89533384, '1114331 ': 0.0015854819958129253, '1117476 ': 0.00020346253737310943, '1119505 ': 0.00066232348953338  
4, '1119708 ': 0.000662895078110775, '1120431 ': 0.0071161185180811, '1123756 ': 0.004202624299062747, '112538  
6 ': 0.01338903358568888, '1127430 ': 0.0, '1127913 ': 0.00028779121625981563, '1128204 ': 0.000725289645688038  
, '1128227 ': 0.0011581377557971244, '1128314 ': 0.0009131537396810833, '1128453 ': 0.0004752730906130063, '1128  
945 ': 0.0, '1128959 ': 0.0002524665972086802, '1128985 ': 0.0006996056957914307, '1129018 ': 0.0004464588262018  
9466, '1129027 ': 0.0, '1129573 ': 0.0012930118651402033, '1129683 ': 0.004855876380140311, '1129778 ': 0.000808  
9235482726651, '1130847 ': 0.0, '1130856 ': 0.0, '1131116 ': 0.001985383114520983, '1131360 ': 0.002332707583168  
773, '1131557 ': 0.0010826749616528928, '1131752 ': 0.0, '1131752 ': 0.0, '1133196 ': 0.0009371634989421305, '1133338 ': 6.544909  
411059364e-05, '1136814 ': 0.00011677302725498497, '1137466 ': 0.0, '1152421 ': 9.575529157724443e-05, '1152508  
' : 0.0, '1153065 ': 6.544909411059364e-05, '1153280 ': 9.575529157724443e-05, '1153577 ': 0.0006771090967818128,  
'1153853 ': 0.0015822480181856317, '1153943 ': 0.005679716608101696, '1154176 ': 6.96816830433253e-05, '1154459  
' : 9.575529157724443e-05, '116552 ': 0.0, '12576 ': 0.0, '128540 ': 0.0, '132806 ': 0.0, '135130 ': 0.0,  
'141342 ': 0.0, '141347 ': 0.0, '148170 ': 0.0, '15670 ': 0.0, '1688 ': 0.0, '175291 ': 0.0, '178727 '  
': 0.0, '18582 ': 0.0, '190697 ': 0.0, '190706 ': 0.0, '1956 ': 0.0, '197054 ': 0.00376878009916963, '198443 '
```

Eigenvalue Centrality

eigenvector_centrality

```
eigenvector_centrality(G, max_iter=100, tol=1e-06, nstart=None, weight=None)
```

Compute the eigenvector centrality for the graph [G](#).

[\[source\]](#)

Eigenvector centrality computes the centrality for a node based on the centrality of its neighbors.
The eigenvector centrality for node i is the i -th element of the vector x defined by the equation

$$Ax = \lambda x$$

where A is the adjacency matrix of the graph [G](#) with eigenvalue λ . By virtue of the Perron–Frobenius theorem, there is a unique solution x , all of whose entries are positive, if λ is the largest eigenvalue of the adjacency matrix A ([\[2\]](#)).

Parameters:

G : *graph*

A networkx graph

max_iter : *integer, optional (default=100)*

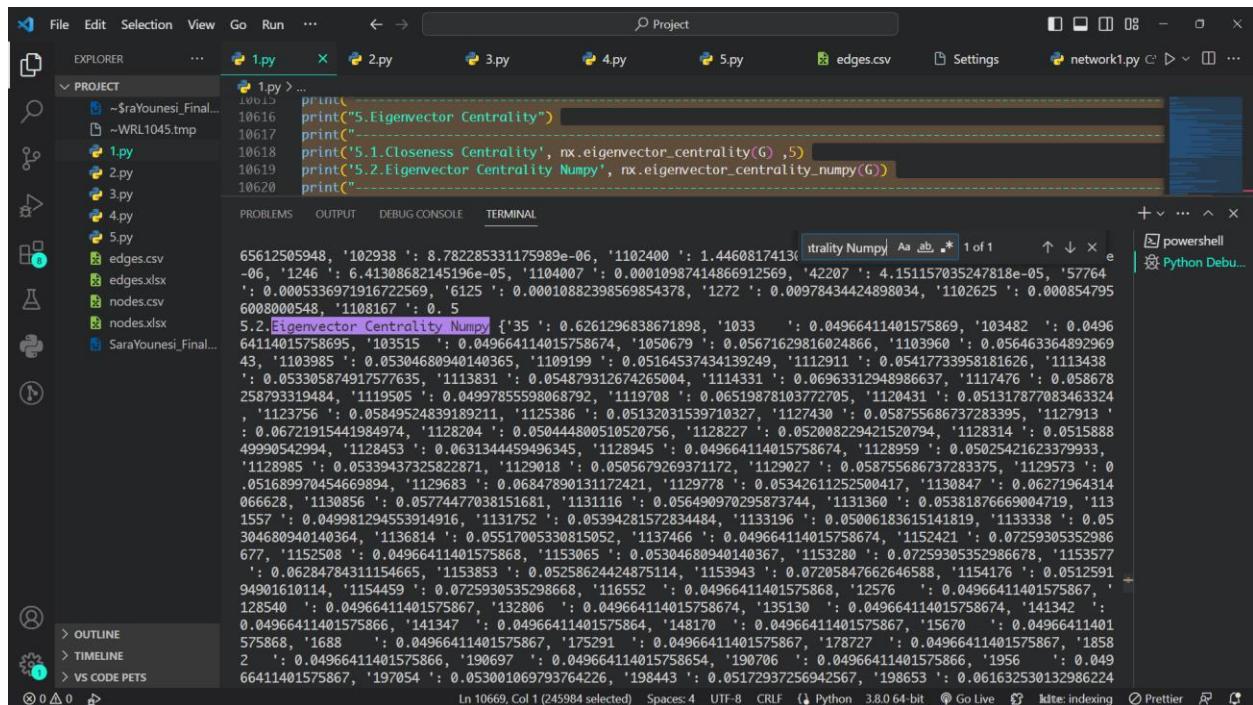
Maximum number of iterations in power method.

tol : *float, optional (default=1.0e-6)*

Error tolerance used to check convergence in power method iteration.

nstart : *dictionary, optional (default=None)*

Starting value of eigenvector iteration for each node.



The screenshot shows the VS Code interface with the following details:

- File Explorer:** Shows a project structure with files 1.py, 2.py, 3.py, 4.py, 5.py, edges.csv, edges.xlsx, nodes.csv, and nodes.xlsx.
- Code Editor:** Displays Python code for calculating eigenvector centrality using NetworkX. The code includes imports for nx and np, and defines a function to calculate closeness centrality and another to calculate eigenvector centrality using Numpy.
- Terminal:** Shows the output of the code execution, displaying a large list of eigenvector centrality values for various nodes. The output starts with:

```
65612505948, '102938' : 7.87228533175989e-06, '1102400' : 1.4460817413e-06, '1246' : 6.41308682145196e-05, '1104007' : 0.0001098741486912569, '42207' : 4.151157035247818e-05, '57764' : 0.0055336971916722569, '6125' : 0.00010882398569854378, '1272' : 0.00978434424898034, '1102625' : 0.000854795600800548, '1108167' : 0.5
```

The terminal also shows a powershell and Python Debug tab.

Katz Centrality

katz_centrality

```
katz_centrality(G, alpha=0.1, beta=1.0, max_iter=1000, tol=1e-06, nstart=None,  
normalized=True, weight=None) [source]
```

Compute the Katz centrality for the nodes of the graph G.

Katz centrality computes the centrality for a node based on the centrality of its neighbors. It is a generalization of the eigenvector centrality. The Katz centrality for node i is

$$x_i = \alpha \sum_j A_{ij} x_j + \beta,$$

where A is the adjacency matrix of graph G with eigenvalues λ .

The parameter β controls the initial centrality and

$$\alpha < \frac{1}{\lambda_{\max}}.$$

Katz centrality computes the relative influence of a node within a network by measuring the number of the immediate neighbors (first degree nodes) and also all other nodes in the network that connect to the node under consideration through these immediate neighbors.

Extra weight can be provided to immediate neighbors through the parameter β . Connections made with distant neighbors are, however, penalized by an attenuation factor α which should be strictly less than the inverse largest eigenvalue of the adjacency matrix in order for the Katz centrality to be computed correctly. More information is provided in [1].

The screenshot shows a Jupyter Notebook interface with the following details:

- File Bar:** File, Edit, Selection, View, Go, Run, ...
- Project Bar:** Project
- Explorer:** PROJECT, ~SraYounes_Final..., ~WRL1045.tmp, 1.py, 2.py, 3.py, 4.py, 5.py, edges.csv, settings, network1.py
- Code Editor:** Python code related to network centrality calculations.
- Output:** katz Centrality, Aa ab, 2 of 2
- TERMINAL:** powershell, Python Debug
- Bottom Status:** Ln 10669, Col 1 (24598 selected), Spaces: 4, UTF-8, CRLF, Python 3.8.0 64-bit, Go Live, Kite indexing, Prettier

• بینابینی (betweenness)

□ بینابینی عبارت است از نسبت تعداد دفعاتی که یک گره یا یک یال بر روی کوتاهترین مسیر میان نودهای مختلف یک گراف قرار می‌گیرد. بینابینی یک نod خاص در شبکه عبارت است از تعداد کوتاهترین مسیرهای میان نودهای شبکه که از یک نod خاص رد میشوند.

در حقیقت این معیار محاسبه می‌کند چه تعداد از نودهای شبکه برای ارتباط سریعتر با هم (با واسطه کمتر) به این نod نیاز دارند. هر چه بینابینی نod زیادتر باشد یعنی اینکه نod در مکان استراتژیکتری قرار گرفته است. همچنین نشان دهنده درصدی از اطلاعات است که از یک گره می‌گذرد و مشخص کننده توانایی یک گره برای تسهیل گسترش ارتباط بین سایر عناصر گرههای گراف است و واقع نمایشی برای میزان قابلیت هر گره برای کمک به دسترسی سایرین به اطلاعات و یا گسترش یک تأثیر در شبکه می‌باشد. این معیار برای یافتن محل افرادی که توانایی مرتبط ساختن با جفت‌ها و گروه‌های دیگر را دارند نشان می‌دهد.

• نزدیکی (closeness)

نزدیکی عبارت است از عکس متوسط فاصله یک نod تا نودهای دیگر گراف. نودی که دارای بیشترین مقدار نزدیکی است سرعت دسترسی بیشتری به نودهای دیگر دارد و میتواند در مدت زمان کمی به همه نودهای اطلاعات ارسال نماید یا از آنها اطلاعات بگیرد.

□ نزدیکی گره X برابر است با عکس متوسط کوتاهترین فاصله گره X تا سایر گره‌های گراف. □ چه مدت زمان می‌برد که اطلاعات از یک گره به دیگر گره‌ها برسد (گره‌هایی که به آن دسترسی دارد). □ جهت پیدا کردن سریع ترین محل انتشار مناسب است.

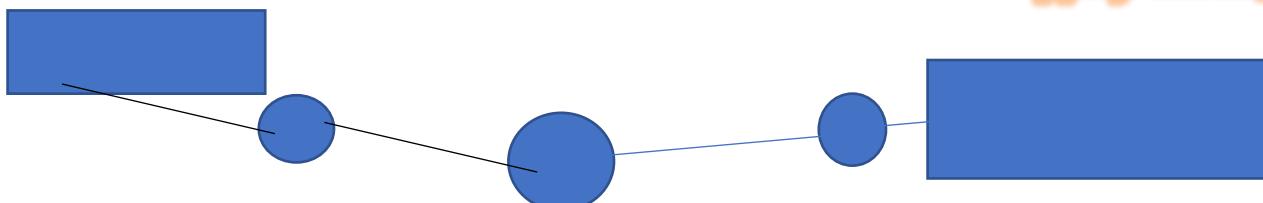
این معیار در صورتی در شبکه قابل محاسبه است که کل شبکه همبند باشد. در صورتی که نودهایی وجود داشته باشند که دسترسی به یکدیگر نداشته باشند \neq مساوی بینهایت شده و این معیار برای آن‌ها صفر خواهد بود و غیر قابل استفاده خواهد گردید. در این حالت معیار کارایی استفاده میشود که فرمولی شبیه نزدیکی دارد اما محدودیت همبند بودن گراف را مرتفع میسازد.

• مرکزیت مقادیر ویژه Eigenvector Centrality

این روش [42] اهمیت گرههای را بر اساس گرههای مجاور محاسبه می‌کند. این محاسبه در گرافهای با اتصال قوی اتفاق می‌افتد. اگر گره‌ای به گرههایی که دارای اهمیت بالایی هستند متصل باشد تحت تأثیر آنها اهمیت او نیز بالا می‌رود. این روش به صورت تکراری برای محاسبه گره اهمیت همسایگان را نیز در نظر می‌گیرد. ابتدا به همه گرههای یک امتیاز اولیه داده می‌شود. در ادامه به صورت زنجیره‌ای تا زمانی که به پایداری برسد این امتیازدهی ادامه می‌یابد. امتیازدهی در این روش بر اساس این مفهوم است که گرههای با اتصالات بالا به گرههای دنبال کننده آنها از نظر امتیاز کمک می‌کنند. روش رتبه‌بندی صفحه از این روش الگوریتمی کرده است.

نتیجه تحلیل و بررسی گام اول

با توجه به ویژگی‌های مختلف مرکزیت گاهای در برخی نودها اهمیت بیشتر و در برخی موارد دیگر کمتر می‌شود برای مثال در یکی از گره‌های ما معیار betweenness بالا بوده است اما معیار نزدیکی و درجه پایین می‌باشد که این به معنی این می‌باشد که این گره در وسط گراف بزرگ ما قرار گرفته است از بقیه نقاط گره در می‌باشد مانند گراف زیر



یا در مورد دیگر می بینیم که درجه کمی دارد اما بیتوینش بالاست.

در مواردی دیگر مشاهده می کنیم که در یک خوشه با درجه بالا قرار گرفته است اما از بقیه نقاط گراف دور می باشد

همچنین چون گراف جهت دار بود درجه گره ها را از جهت گره های وارد شده و خارج شده و مجموع آن ها

بررسی کردیم

گام دوم

با استفاده از معیارهای مختلف شیبکه مانند clustering coefficient، centrality و ... و خلاقیت خود، مهمترین گره ها را در گراف به دست آورید و تحلیل خود را نیز گزارش دهید.

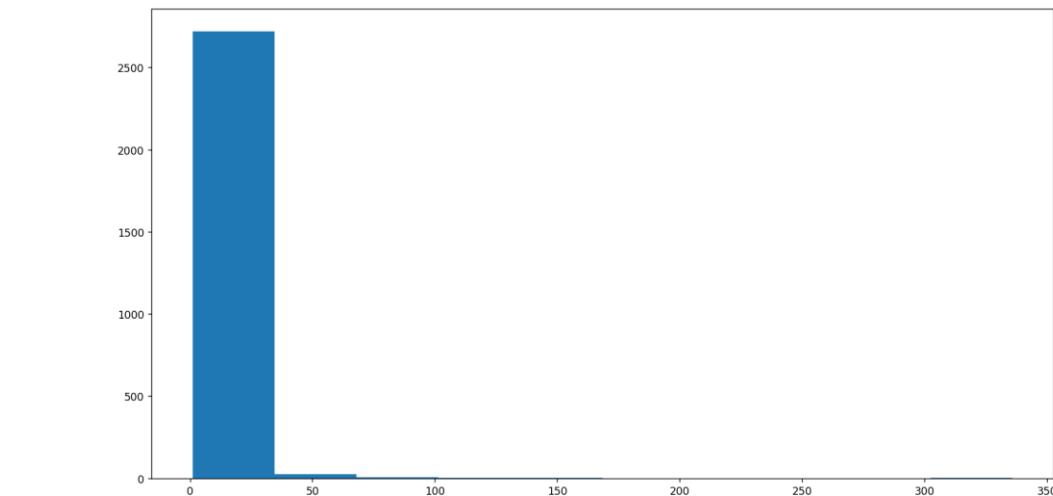
Degree distribution

فرآوانی درجات گره ها (Degree distribute)

درجه یک گره تعداد گره هایی است که با آن گره در همسایگی مستقیم قرار دارد. هر چقدر درجه یک گره بیشتر باشد، اهمیت آن گره بیشتر می شود. مثلا در شبکه وب، اگر درجه ورودی یک دامنه زیاد باشد، یعنی سایت مرتع است، و اگر درجه خروجی آن زیاد باشد، یعنی اینکه سایتی است که از اطلاعات (اخبار) سایت های دیگر استفاده می کند. توزیع درجات، توزیع احتمال این درجات روی کل شبکه است. توزیع درجات شبکه $p(k)=n_k/n$ محاسبه می شود.

به تعداد دوستان (تعداد روابط اجتماعی) یک نو اطلاق میشود. هر چه این یالها بیشتر باشد نشان از روابط اجتماعی کسترده تر فرد دارد. اگر نوع ارتباط اجتماعی جهت دار باشد (مانند اعتماد داشتن) آنگاه درجه خروجی و درجه ورودی نیز مطرح میشود که معانی خاص خود را خواهند داشت. مثلا در مورد شبکه اعتماد، کسی که درجه ورودی زیادی دارد مورد اعتماد افراد بیشتری است و اگر درجه خروجی زیادی داشته باشد یعنی اینکه به افراد زیادی اعتماد دارد.

Figure 1

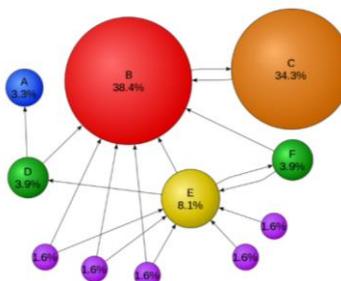


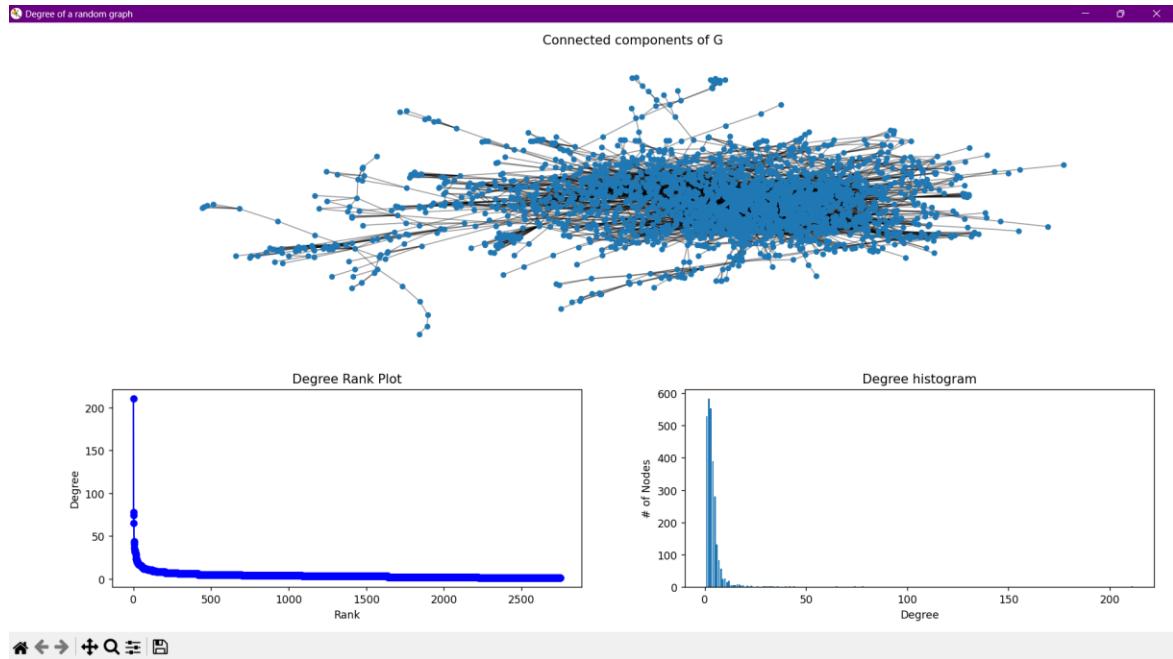
• رتبه صفحه (page rank)

□ این الگوریتم برای رتبهدهی به صفحات وب و تعیین ارزش آنها بر اساس پیوندهای بین آنها میباشد. □ در این الگوریتم، میزان ارزش و اعتبار یک صفحه به ارزش و اعتبار صفحاتی وابسته است که به او لینک داده‌اند و هرچه این صفحات با ارزشتر باشند اعتبار صفحه بالاتر میبود. □ با توجه به اینکه در ابتدا داشتی در مورد ارزش صفحات نداریم ابتدا به تمام صفحات مقدار یکسانی به عنوان ارزش اولیه داده میشود. □ چون با تغییر ارزش هر صفحه، با استفاده از انتشار، ارزش کلیه صفحات دستخوش تغییر قرار می‌گیرد این الگوریتم به صورت تکراری آنقدر اجرا میشود تا در پایان ارزش همه صفحات به عددی همگرا شود. □ تنها گره هایی ارزش میگیرند که دارای همسایه باشند و لینکی از دیگران به آنها وارد شده باشد و در غیر اینصورت ارزش صفر را خواهند گرفت.

این معیار توسط گوگل برای رتبه بندی صفحات و اکنون نیز قسمت زیادی از محاسبات گوگل بر همین مبنای است. ایده این معیار در این است که یک صفحه وب در صورتی مهم است که توسط صفحات مهم زیادی مورد ارجاع قرار گرفته باشد. اهمیت صفحات ارجاع دهنده نیز به همین صورت حساب میشود. آنها نیز در صورتی مهم هستند که توسط صفحات مهمی مورد ارجاع قرار گرفته باشند. این تعریف، مانند تعریف قدرت یک تعریف بازگشتی است که مقدار یک عبارت را به صورت تابعی از خودش تعریف میکند.

برای محاسبه این معیار، فرض میشود همه صفحات در ابتدا دارای ارزش یکسانی هستند. هر نod ارزش خودش را به نسبتی مساوی میان لینکهای خروجی اش تقسیم میکند. ارزش جدید هر نod عبارت میشود از مجموع ارزشی که نودهای ارجاع دهنده به او میدهند. این پروسه تا زمانی که این ارزشها همگرا شوند ادامه میکند. معمولاً پس از چند دور محدود این اعداد همگرا میشوند. نسخه های جدیدتری از رتبه صفحه نیز (مثلاً رتبه صفحه وزن دار) وجود دارند که برخی مشکلات آن را مرتفع میکنند.





Cluster coefficient

معیار های centrality به طور کامل و جامع در سوال قبلی پاسخ داده شده است به بررسی مشخصات جدید و دیگر می پردازیم.

```
PS C:\Users\user\Desktop\Project> & 'C:\Users\user\AppData\Local\Programs\Python\Python38\python.exe' 'c:\Users\user\.vscode\extensions\ms-python.python-2023.10.0\pythonFiles\lib\python\debugpy\adapter/../..\debugpy\launcher' '8172' '--' 'c:\Users\user\Desktop\Project\2.py'

1.Clustering Coefiant
```

The terminal output shows the execution of a Python script named 2.py. The script calculates the clustering coefficient for a network graph. The output is a large dictionary where each key is a node ID and the value is its clustering coefficient. The output is heavily redacted with a large red box.

```
70 ': 0, '238099 ': 0, '86840 ': 0, '3213 ': 0, '1106492 ': 0, '11
1.2.transitivity 0.08092275104256305
```

The terminal output shows the execution of a Python script named 2.py. The script calculates the transitivity of a network graph. The output is a single value: 0.08092275104256305. The output is heavily redacted with a large red box.

The screenshot shows a Python code editor with several tabs open. The active tab is '2.py' which contains the following code:

```
10611 print("-----")
10612 print("1.Clustering Coefiant")
10613 print("-----")
10614 print('1.1.triangles', nx.triangles(G))
10615 print("-----")
10616 print('1.2.transitivity', nx.transitivity(G))
10617 print("-----")
10618 print('1.3.Clustering', nx.clustering(G))
```

Below the code, the terminal window displays the output of the script:

```
'11123239 : 1.0, '1133004 : 0.256106 ': 0.8333333333333334, '469504 ': 0, '5348 ': 0.1333333333333333, '99025 ': 0.2, '2702 ': 0.3333333333333333, '1120777 ': 0.5, '12330 ': 0.1333333333333333, '39
```

The terminal also shows the result of the '1.4.Average Clustering' command:

```
1.4.Average Clustering 0.2369098903778341
```

At the bottom of the terminal, there is a large amount of output for the '1.5.Square Clustering' command, which is partially cut off.

The screenshot shows a Microsoft Visual Studio Code (VS Code) window with the following details:

- File Explorer:** Shows a project structure with files like 1.py, 2.py, network1.py, 3.py, 4.py, edges.csv, and nodes.xlsx.
- Code Editor:** The main editor pane displays Python code for network analysis, specifically calculating clustering coefficients and transitivity. The code includes imports from the networkx library and various print statements.
- Terminal:** The terminal pane shows the output of the code execution, which includes the results of the clustering coefficient calculations.
- Python Debug Console:** A sidebar panel showing the output of the Python debug session.
- Status Bar:** Shows the file path as C:\Users\Raresh\PycharmProjects\NetworkAnalysis\src\main\py\network1.py, the line number as Ln 10608, Col 1 (10 selected), and the status message as 'kitte ready'.

```

File Edit Selection View Go Run ... ← → ⌘ Project
EXPLORER ... 1.py 2.py ✎ network1.p... 3.py 4.py
PROJECT
~$raYounesi_Final...
1.py
2.py
3.py
4.py
5.py
edges.csv
edges.xlsx
nodes - bellman.x...
nodes - lable.xlsx
nodes - path.xlsx
nodes.csv
nodes.xlsx
SaraYounesi_Final...
10611 print("-----")
10612 print("1.Clustering Coefiant")
10613 print("-----")
10614 print('1.1.triangles', nx.triangles(G))
10615 print("-----")
10616 print('1.2.transitivity', nx.transitivity(G))
10617 print("-----")
10618 print('1.3.Clustering', nx.clustering(G))

': Counter({0: 4, 1: 2}), '3828 ': Counter({1: 6, 2: 1, 0: 1}), '38845 ': Counter({2: 3, 1: 2, 0: 1}), '940 ': Counter({1: 3, 3: 1}), '20180 ': Counter({0: 2, 1: 2}), '28265 ': Counter({0: 2, 1: 2}), '941 ': Counter({1: 1})

Network Centralization [{35 ': 0.669902745203269, '1033 ': 0.042817040742268844, '103482 ': 0.042817040742268844, '103515 ': 0.042817040742268844, '103590 ': 0.046934610522080485, '1103960 ': 0.05203929561167253, '1103985 ': 0.04793900855430572, '11040199 ': 0.04647390512192104, '11123311 ': 0.04932930509520834, '1113438 ': 0.0488336074738904, '1113831 ': 0.046232631689589
083, '1114331 ': 0.05538370938171554, '1117476 ': 0.04887772788940267, '1119505 ': 0.042992605763672885, '1119708 ': 0.05794520
7549878246, '1120431 ': 0.04368074717533828, '1123751 ': 0.0481035380019636, '1125386 ': 0.04407036791667651, '1127430 ': 0.04
8407389840232365, '1127913 ': 0.05654794709114895, '1128204 ': 0.043641120822780515, '1128227 ': 0.04702889294905375, '1128314
': 0.04718543181069956, '1128453 ': 0.0511383223437317, '1128945 ': 0.042817040742268844, '1128959 ': 0.0436510730265597, '112
8985 ': 0.045279014626792354, '1129018 ': 0.04332510522580245, '1129027 ': 0.048407389840232365, '1129573 ': 0.0488237949194275
8, '1129683 ': 0.05392697166052226, '1129778 ': 0.0517976208247028, '1130847 ': 0.05096625969682001, '1130856 ': 0.04770997477
583539, '1131116 ': 0.04701341447903587, '1131360 ': 0.0483727785877144, '1131557 ': 0.04299470481813464, '1131752 ': 0.04573
92960395611, '1133196 ': 0.04304110403011398, '1133338 ': 0.04793900855430572, '1136814 ': 0.0519292776546568, '1137466 ': 0.
042817040742268844, '1152421 ': 0.05668292946538046, '1152508 ': 0.042817040742268844, '1153065 ': 0.04793900855430572, '115328
0 ': 0.05668292946538046, '1153577 ': 0.05144644918053101, '1153853 ': 0.048205653208869326, '1153943 ': 0.05604641114966955,
'1154176 ': 0.0438976883598905, '1154459 ': 0.05668292946538046, '116552 ': 0.042817040742268844, '12576 ': 0.042817040742268844
844, '128540 ': 0.042817040742268844, '132806 ': 0.042817040742268844, '135130 ': 0.042817040742268844, '141342 ': 0.042817
040742268844, '141347 ': 0.042817040742268844, '148170 ': 0.042817040742268844, '15670 ': 0.042817040742268844, '1688 ': 0.
042817040742268844, '175291 ': 0.042817040742268844, '178727 ': 0.042817040742268844, '18582 ': 0.042817040742268844, '1
90697 ': 0.042817040742268844, '190706 ': 0.042817040742268844, '198653 ': 0.049804215598261196, '206371 ': 0.04980426263770465,
'210871 ': 0.080862047316653, '2198443 ': 0.04714539397791529, '219853 ': 0.042817040742268844, '197054 ': 0.04865670289337
6553, '229635 ': 0.04818360738466311, '231249 ': 0.04704066716697303, '248425 ': 0.050504620892749895, '249421 ': 0.04361561
856828, '229635 ': 0.04818360738466311, '231249 ': 0.04704066716697303, '248425 ': 0.050504620892749895, '249421 ': 0.04361561

```

In 10608, Col 1 (10 selected) Spaces: 4 UTF-8 CRLF Python 3.8.0 64-bit Go Live Idle: ready Prettier

```

File Edit Selection View Go Run ... ← → ⌘ Project
EXPLORER ... 1.py 2.py ✎ network1.p... 3.py 4.py
PROJECT
~$raYounesi_Final...
1.py
2.py
3.py
4.py
5.py
edges.csv
edges.xlsx
nodes - bellman.x...
nodes - lable.xlsx
nodes - path.xlsx
nodes.csv
nodes.xlsx
SaraYounesi_Final...
10611 print("-----")
10612 print("1.Clustering Coefiant")
10613 print("-----")
10614 print('1.1.triangles', nx.triangles(G))
10615 print("-----")
10616 print('1.2.transitivity', nx.transitivity(G))
10617 print("-----")
10618 print('1.3.Clustering', nx.clustering(G))

42e-05, '1114118 ': 8.598582010694619e-05, '1116569 ': 8.599422132996831e-05, '1118848 ': 0.0001765641490397104, '1120858 ': 9
.198860076873996e-05, '1122460 ': 8.598582010694619e-05, '1126044 ': 0.00020754393339308643, '1129111 ': 9.246658212823507e-05,
'1131537 ': 0.0004900771772283978, '1152194 ': 0.00010453348484045558, '12439 ': 0.0001738103203608267, '12946 ': 0.000159494
17904156693, '131042 ': 0.00010853073046203861, '13136 ': 9.245927552444423e-05, '160705 ': 9.447761150493044e-05, '227286 ': 0
.0001589214661574563, '242637 ': 0.0001308127250725729, '31043 ': 0.00014697161077217985, '340075 ': 0.0004656059313752722, '3
40078 ': 0.00010252082876498841, '35905 ': 0.0001951319132336745044, '42847 ': 0.0002493915196587203, '43679 ': 0.00038114118201
329, '48550 ': 0.000105105190692875508, '5462 ': 0.0001068647547110913, '576257 ': 0.00067205569208999396, '58552 ': 8.638873037
52669e-05, '5869 ': 9.4427284275621e-05, '63651 ': 0.04858159498138e-05, '67292 ': 9.14834407310018e-05, '675649 ': 0.00012
672565773949352, '684372 ': 0.0001953688481091192, '9495 ': 8.563432677339612e-05, '936 ': 0.0036486211944990092, '1107010 ': 0
.000258741324346211, '1111899 ': 0.0031325906560021413, '129558 ': 0.0002499326787488134, '1958 ': 0.04516964789453014, '207
395 ': 0.0005139072078722902, '3084 ': 0.004046199157719317, '3828 ': 0.003473436157691254, '38845 ': 0.0002611629018843023, '94
90 ': 0.0002469595920875269, '20180 ': 6.548251126217145e-05, '28265 ': 6.30948286020867e-05, '941 ': 0.000261692229590035,
'943 ': 0.0004745920578392305, '1152896 ': 0.0037636644781513624, '181852 ': 5.979209594624255e-05, '1026 ': 0.00030696826228874
95, '1034 ': 0.0032440612836543287, '1102550 ': 0.0012925171950659447, '1105231 ': 1.9695365756264445e-05, '1129798 ': 1.961492
093334756e-05, '1153945 ': 0.00022696396277672174, '1033 ': 0.048279011300468085, '1107062 ': 0.0038177857634997124, '1035 ': 0
.001963112573400564, '1110515 ': 0.00433963939823878, '1213 ': 9.43138087087258e-21, '1154525 ': 1.06604481221245ze-20, '4097
25 ': 3.006411386636765e-21, '8766 ': 3.006411386636765e-21, '1237 ': 2.5050364708558424e-05, '102938 ': 1.7269610524336734e-06
, '1102400 ': 4.365000204205858e-06, '143676 ': 1.7269610524336734e-06, '1246 ': 3.716020194319056e-05, '1104007 ': 5.282699549

```

In 10608, Col 1 (10 selected) Spaces: 4 UTF-8 CRLF Python 3.8.0 64-bit Go Live Idle: ready Prettier

نتیجه تحلیل و بررسی گام دوم

ابتدا از نمودار توزیع درجه ها متوجه می شویم تعداد زیادی از نود ها دارای درجه ۴۵ حدودا هستند و بقیه درجات در نود های دیگر پراکنده شده و درجه های بسیار بالا در این گراف کم میباشد.

سپس متوجه می شویم دیامتر و طول گراف بسیار زیاد می باشد و راه کوتاه بسیار کم میباشد سپس متوجه شدیم که مرکزیت گره ها نسبت به کل گراف همچنین قرار گرفتن نود ها در خوش و تشکیل دادن خوش های سه تایی یا **transitivity** مهم ترین گره های گراف اعم از موارد پایین می باشد و در اسکرین ها اندازه و فرمول همه موارد موجود می باشد.

۱۱۰۳۹۸۵ و ۱۱۰۳۹۶۰ می باشند.

گام سوم

گراف ساخته شده را با استفاده از برچسب هر گره رنگ آمیزی کنید به طوری که گرههایی که متعلق به یک برچسب هستند همنگ باشند. گراف رنگی حاصل را تحلیل کنید و نتیجه های جالبی که میتوان از این ۱ تحلیل ها به دست آورد را گزارش کنید. در انتها جوامع را در گراف مورد نظر پیدا کرده و گزارش دهید و ۲ همچنین بگویید که آیا میتوان با استفاده از روش های تشخیص جوامع ، گرهها را بر اساس برچسب های مشخص شده طبقه بندی کرد یا خیر؟

Figure 1

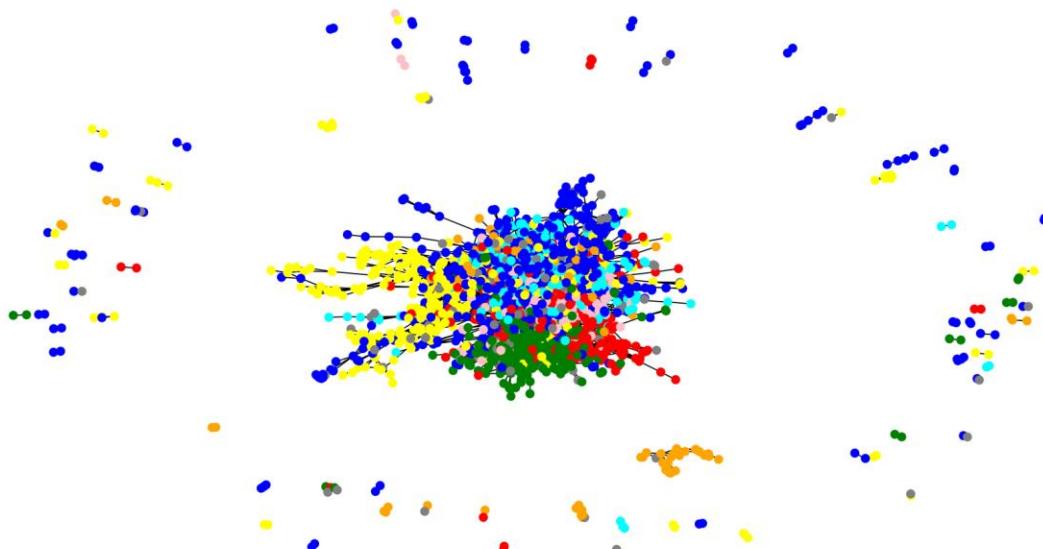


Figure 1

x=1.181 y=0.304

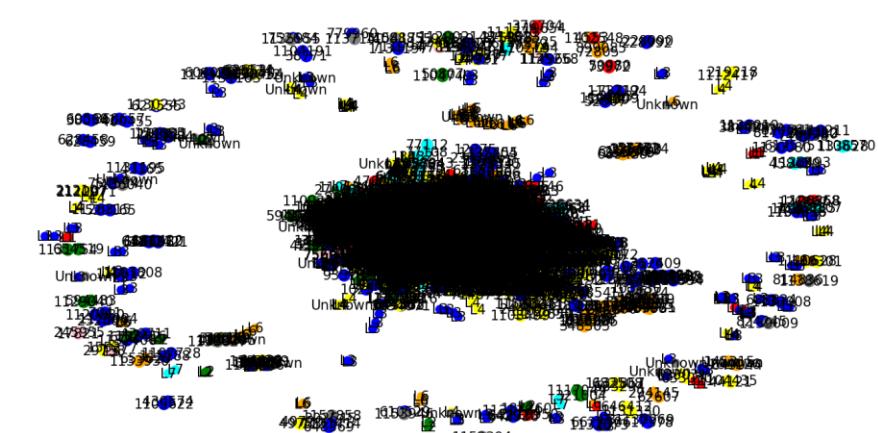


Figure 1

```
print("-----")
print('1. Community detection', sorted(map(sorted, next_level_communities)))
print("-----")
print('2. Modularity 1 ', nx.community.greedy_modularity_communities(G))
```

```

print("-----")
print("-----")
print('3.Modularity 2', nx.community.naive_greedy_modularity_communities(G))
print("-----")
print("-----")
print('4 Measuring partitions', nx.community.modularity(
    G, nx.community.label_propagation_communities(G)))
print("-----")
print("-----")
print('5.girvan_newman', tuple(sorted(c) for c in
next(community_generator)))
print("-----")

```

The screenshot shows a code editor interface with the following details:

- File Explorer:** Shows a project structure with files like 1.py, 2.py, 3.py, 3-2.py, network1.py, 4.py, and 5.py.
- Code Editor:** The current file is 3-2.py, displaying the following code:

```

from networkx.algorithms import community
nodes = pd.read_excel('nodes.xlsx', usecols=['NodeId', 'Labels'])
edges = pd.read_excel('edges.xlsx', usecols=['sourceNodeId', 'targetNodeId'])
G = nx.from_pandas_edgelist(edges, 'sourceNodeId', 'targetNodeId')
Modularity_communities [2] [35, 40, 887, 1026, 1033, 1034, 1035, 1688, 1694, 8865, 8872, 8874, 8875, 12558, 1257
6, 14062, 15670, 18582, 19045, 22229, 22241, 28287, 28290, 28336, 28456, 28851, 33895, 33904, 33907, 35061, 3585
7, 38205, 39474, 41714, 41732, 44368, 44455, 44514, 45599, 45603, 45605, 46079, 48764, 48766, 48768, 48781, 5412
9, 54131, 54132, 56112, 56115, 56119, 56708, 56709, 57119, 58758, 59715, 61069, 61073, 61312, 62329, 62634, 6271
8, 63832, 66556, 66563, 66564, 69284, 69296, 70970, 78511, 81714, 81722, 82920, 83725, 84021, 85352, 87363, 9463
9, 94641, 96845, 97390, 97645, 98698, 100935, 103515, 108047, 110041, 116545, 116552, 116553, 117315, 117
316, 127033, 128540, 132806, 134199, 134219, 135130, 141324, 141342, 144212, 144701, 148170, 148341, 152
483, 164894, 175256, 177993, 177998, 178718, 178727, 182093, 182094, 190697, 190706, 195792, 197054, 198
443, 198653, 206371, 210871, 210872, 220420, 227178, 229635, 231249, 240791, 243483, 246618, 248425, 248431, 251
756, 253971, 254923, 259701, 259702, 263069, 263279, 263498, 265203, 270085, 273152, 286500, 286513, 287787, 292
277, 307015, 307656, 308529, 335733, 375605, 387795, 390693, 395075, 397488, 415693, 416455, 423816, 427606, 447
224, 459206, 459213, 459214, 459216, 467383, 486840, 503871, 503877, 503883, 503893, 509233, 513189, 516238, 568
857, 573964, 573978, 574009, 574264, 574462, 574710, 575077, 575292, 575331, 575402, 575795, 576257, 576362, 576
691, 576725, 576795, 576973, 577086, 577227, 577331, 578645, 578646, 578649, 578650, 578669, 578780, 578845, 578
898, 579008, 579108, 583318, 592973, 592975, 592986, 592993, 592996, 593022, 593060, 593068, 593091, 593104, 593
105, 593155, 593201, 593209, 593210, 593240, 593248, 593260, 593328, 593329, 593544, 593559, 593560, 593813, 593
859, 593921, 593942, 594011, 594039, 594047, 594387, 594511, 594543, 594649, 594900, 595056, 595063, 595157, 595
193, 606479, 606647, 608190, 608191, 608292, 608326, 634902, 634904, 634938, 634975, 640617, 643199, 646809, 646
836, 646837, 646900, 646913, 647315, 647408, 647447, 650807, 650814, 650834, 682815, 694759, 735303, 735
311, 785678, 787016, 815096, 975567, 1050679, 1071981, 1102550, 1103383, 1103960, 1103985, 1104647, 1104999, 110
5221, 1105231, 1107062, 1107418, 1107674, 1109017, 1109199, 1109208, 1110515, 1110546, 1112911, 1113438, 1113742
, 1113831, 1114331, 1114502, 1117476, 1119505, 1119708, 1120643, 1121398, 1123530, 1123756, 1125092, 1125492, 11
25597, 1127430, 1127913, 1128151, 1128198, 1128201, 1128204, 1128227, 1128291, 1128314, 1128319, 112836

```
- Terminal:** Shows the command "Ln 21, Col 15".
- Status Bar:** Shows "File: indexing" and "Prettier" icon.

greedy_modularity_communities

greedy_modularity_communities(*G*, *weight=None*, *resolution=1*, *cutoff=1*, *best_n=N*, *one*) [source]

Find communities in *G* using greedy modularity maximization.

This function uses Clauset-Newman-Moore greedy modularity maximization [2] to find the community partition with the largest modularity.

Greedy modularity maximization begins with each node in its own community and repeatedly joins the pair of communities that lead to the largest modularity until no further increase in modularity is possible (a maximum). Two keyword arguments adjust the stopping condition. *cutoff* is a lower limit on the number of communities so you can stop the process before reaching a maximum (used to save computation time). *best_n* is an upper limit on the number of communities so you can make the process continue until at most *n* communities remain even if the maximum modularity occurs for more. To obtain exactly *n* communities, set both *cutoff* and *best_n* to *n*.

This function maximizes the generalized modularity, where *resolution* is the resolution parameter, often expressed as $\frac{1}{N}$. See [modularity\(\)](#).

Parameters:

***G*NetworkX graph**

***weight*string or None, optional (default=None)**

The name of an edge attribute that holds the numerical value used as a weight. If None, then each edge has weight 1. The degree is the sum of the edge weights adjacent to the node.

***resolution*float, optional (default=1)**

If resolution is less than 1, modularity favors larger communities. Greater than 1 favors smaller communities.

***cutoff*int, optional (default=1)**

A minimum number of communities below which the merging process stops. The process stops at this number of communities even if modularity is not maximized. The goal is to let the user stop the process early. The process stops before the cutoff if it finds a maximum of modularity.

***best_n*int or None, optional (default=None)**

A maximum number of communities above which the merging process will not stop. This forces community merging to continue after modularity starts to decrease until `best_n` communities remain. If `None`, don't force it to continue beyond a maximum.

Returns:

communities: list

A list of frozensets of nodes, one for each community. Sorted by length with largest communities first.

modularity

`modularity(G, communities, weight='weight', resolution=1)`

[\[source\]](#)

Returns the modularity of the given partition of the graph.

Modularity is defined in [1] as

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

where m is the number of edges, A is the adjacency matrix of `G`, k_i is the degree of i , γ is the resolution parameter, and $\delta(c_i, c_j)$ is 1 if i and j are in the same community else 0.

According to [2] (and verified by some algebra) this can be reduced to

$$Q = \sum_{c=1}^n \left[\frac{L_c}{m} - \gamma \left(\frac{k_c}{2m} \right)^2 \right]$$

where the sum iterates over all communities c , m is the number of edges, L_c is the number of intra-community links for community c , k_c is the sum of degrees of the nodes in community c , and γ is the resolution parameter.

The resolution parameter sets an arbitrary tradeoff between intra-group edges and inter-group edges. More complex grouping patterns can be discovered by analyzing the same network with multiple values of gamma and then combining the results [3]. That said, it is very common to simply use `gamma=1`. More on the choice of `gamma` is in [4].

girvan_newman

`girvan_newman(G, most_valuable_edge=None)`

[source]

Finds communities in a graph using the Girvan–Newman method.

Parameters:

G : NetworkX graph

most_valuable_edge : function

Function that takes a graph as input and outputs an edge. The edge returned by this function will be recomputed and removed at each iteration of the algorithm.
If not specified, the edge with the highest `networkx.edge_betweenness_centrality()` will be used.

Returns:

iterator

Iterator over tuples of sets of nodes in `G`. Each set of node is a community, each tuple is a sequence of communities at a particular level of the algorithm.

Notes

The Girvan–Newman algorithm detects communities by progressively removing edges from the original graph. The algorithm removes the “most valuable” edge, traditionally the edge with the highest betweenness centrality, at each step. As the graph breaks down into pieces, the tightly knit community structure is exposed and the result can be depicted as a dendrogram.

modularity

`modularity(G, communities, weight='weight', resolution=1)`

[source]

Returns the modularity of the given partition of the graph.

Modularity is defined in [1] as

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

where m is the number of edges, A is the adjacency matrix of `G`, k_i is the degree of i , γ is the resolution parameter, and $\delta(c_i, c_j)$ is 1 if i and j are in the same community else 0.

According to [2] (and verified by some algebra) this can be reduced to

$$Q = \sum_{c=1}^n \left[\frac{L_c}{m} - \gamma \left(\frac{k_c}{2m} \right)^2 \right]$$

where the sum iterates over all communities c , m is the number of edges, L_c is the number of intra-community links for community c , k_c is the sum of degrees of the nodes in community c , and γ is the resolution parameter.

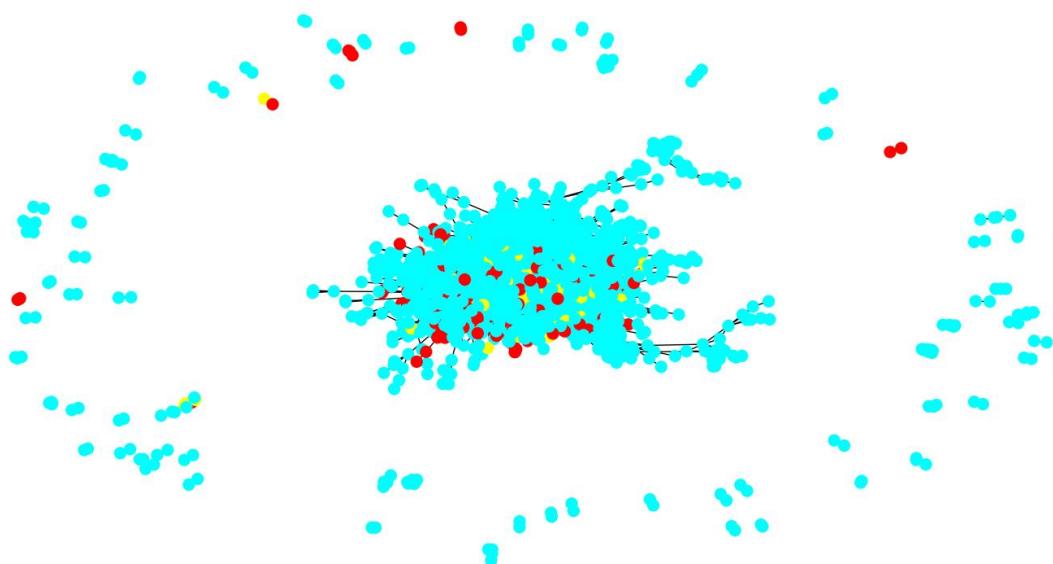
The resolution parameter sets an arbitrary tradeoff between intra-group edges and inter-group edges. More complex grouping patterns can be discovered by analyzing the same network with multiple values of gamma and then combining the results [3]. That said, it is very common to simply use gamma=1. More on the choice of gamma is in [4].

نتیجه تحلیل و بررسی گام سوم

بله با استفاده از مازولریتی ها و جوامع بدست آمده مخصوصا در گیروان نیومن می توان به این نتیجه رسید که خوشه های هم رنگ و دارای یک لیبل تا حد خوبی در یک یا همان خوشه ای که با استفاده از فرمول های شبکه می نویسیم قرار می گیرند

گام چهارم

گره های با برچسبی ۱ را گره های بیمار در جامعه در نظر بگیرید. آدمهای دارای بیش ترین احتمال ابتلا به این بیماری را از بین گره های با برچسب نامشخص تعیین نمایید ۱Community 2Community Detection 3Unknown



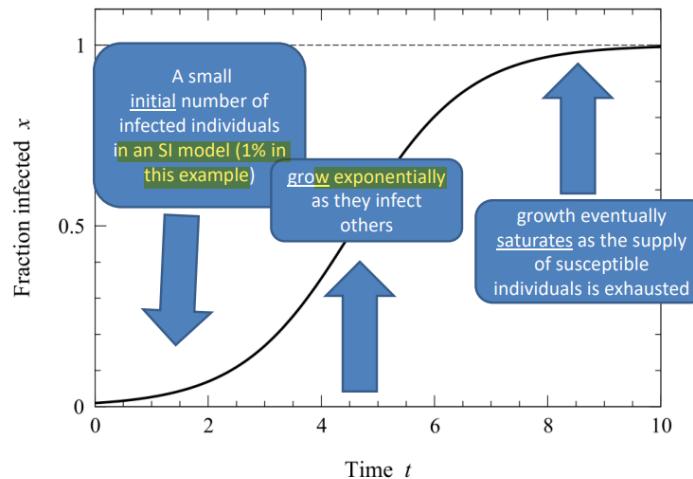
The SI model

- Let $S(t)$ be the number of individuals who are susceptible at time t
- $X(t)$ be the number who are infected
- Define S and X more carefully to be the average or expected numbers of susceptible and infected individuals
- Each individual has, on average, β contacts with randomly chosen others per unit time.

11

The SI model

- If the total population consists of n people, then the average probability of a person you meet at random being susceptible is S/n , and hence an infected person has contact with an average of $\beta S/n$ susceptible people per unit time
- Overall average rate of new infections will be $\beta S X/n$



نتیجه تحلیل و بررسی گام چهارم

همانگونه که بالا می بینیم از فرمول درون اسلاید ها استفاده شده و نرخ ابتلا و یک تری شده برای مبتلایان و مستعدان در کد تری شلد گذاشته شده و بیمار ها با رنگ قرمز و مستعدان با زرد و دیگر افراد با آبی نمایش داده شده اند.

گام پنجم

با اسیتفاذه از ۳ اسیترات ای مختلف گره هایی که برچسب نامشیخج دارند را برچسب گذاری نمایید و پیشینی کنید در کدام دسته قرار میگیرند و در اکسل گره ها وارد نمایید. تحلیل های انجام شیده نیز در گزارش ارسالی شرح داده شود(برچسب گرهها ۱ L و ۲ L و و ۷ L است)

KNN

KNN is a simple, supervised machine learning (ML) algorithm that can be used for classification or regression tasks - and is also frequently used in missing value imputation. It is based on the idea that the observations closest to a given data point are the most "similar" observations in a data set, and we can therefore classify unforeseen points based on the values of the closest existing points. By choosing K, the user can select the number of nearby observations to use in the algorithm.

این الگوریتم برای مسائل طبقه‌بندی k نزدیک ترین همسایه را پیدا و با اکثریت آرا نزدیک‌ترین همسایگان کلاس را پیش‌بینی می‌کند.

برای مسائل رگرسیون k نزدیک‌ترین همسایه را پیدا و با محاسبه میانگین مقدار نزدیک‌ترین همسایه‌ها، مقدار مدنظر را پیش‌بینی می‌کند.

مراحل الگوریتم K نزدیک‌ترین همسایه

بیایید با هم مراحل این الگوریتم را بررسی کنیم:

۱. داده‌ها را بارگذاری می‌کنیم.
۲. مقدار K را تعیین می‌کنیم که همان تعداد نزدیک‌ترین همسایه‌ها هستند.
۳. برای هر نمونه داده:

- فاصله‌ی میان نمونه داده‌ی جدید را با نمونه داده‌های موجود محاسبه می‌کنیم.
 - فاصله و شاخص هر نمونه را به یک فهرست وارد می‌کنیم.
۴. کل لیست را براساس فاصله‌ی نمونه داده‌ها، از کمترین به بیشترین فاصله، مرتب می‌کنیم.
۵. Kتا از اولین نمونه‌های فهرست مرتب شده را به عنوان K نزدیک‌ترین همسایه انتخاب می‌کنیم.
۶. برچسب این K نمونه را بررسی می‌کنیم.
۷. اگر مسئله رگرسیون باشد، میانگین برچسب‌های این K نمونه داده برچسب نمونه داده جدیدمان خواهد بود.
۸. درصورتی‌که مسئله طبقه‌بندی باشد، نمونه‌ی جدید هم همان برچسب K همسایه را خواهد داشت.

K قطعاً بعد از دیدن این الگوریتم، از اولین سوالاتی که برایمان پیش می‌آید این است که اولاً مقدار را چگونه انتخاب کنیم و دوماً چطور این نزدیک‌ترین همسایه‌ها را پیدا کنیم. در ادامه پاسخ این دو سؤال را خواهیم یافت.

را پیدا کنیم؟ K چطور مقدار بهینه

تنظیم یارامتر نامیده می‌شود و برای نتایج بهتر ضروری است. برای انتخاب K انتخاب مقدار مناسب ریشه‌ی مربع یا همان جذر یا رادیکال تعداد کل نقاط داده‌ی موجود در مجموعه‌ی داده را K مقدار حساب می‌کنیم.

همیشه برای جلوگیری از سردرگمی میان دو K این موضوع را در نظر بگیرید که درنهایت مقدار فرد کلاس انتخاب می‌شود.

چطور نزدیک‌ترین همسایه‌ها را پیدا کنیم؟

نزدیک‌ترین همسایه وجود دارد k تکنیک‌های مختلفی برای یافتن

- فاصله‌ی اقلیدسی (Euclidean distance)
- فاصله‌ی منهتن (Manhattan distance)
- فاصله‌ی مینکوفسکی (Minkowski distance)

یکی از تکنیک‌هایی که بسیار استفاده می‌شود فاصله‌ی اقلیدسی است.

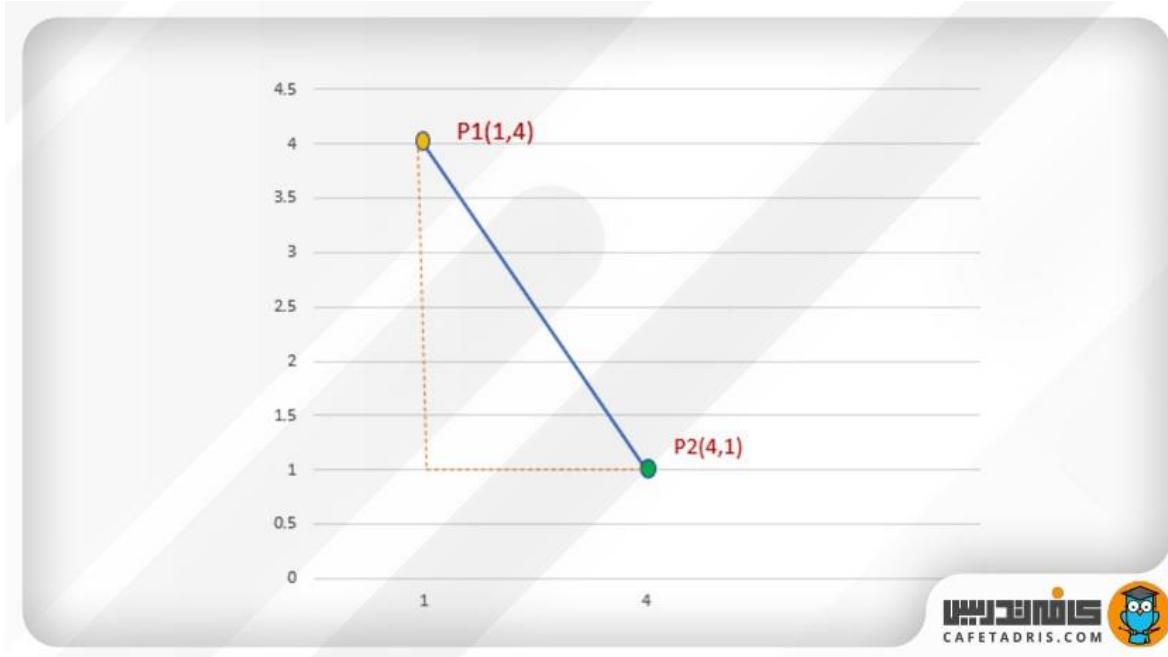
فاصله‌ی اقلیدسی برای محاسبه‌ی فاصله‌ی میان دو نقطه در یک صفحه یا یک فضای سه‌بعدی استفاده می‌شود. فاصله‌ی اقلیدسی براساس قضیه‌ی فیثاغورس است.

را محاسبه کنیم P2 و P1 بیایید فاصله‌ی میان

$$P1(4,1) \text{ و } P2(1,4)$$

قضیه‌ی فیثاغورس

در هر مثلث راست‌گوشه یا قائم‌الزاویه مربع وتر (طولانی‌ترین ضلع مثلث) برابر است با مجموع مربع دو ضلع دیگر مثلث.



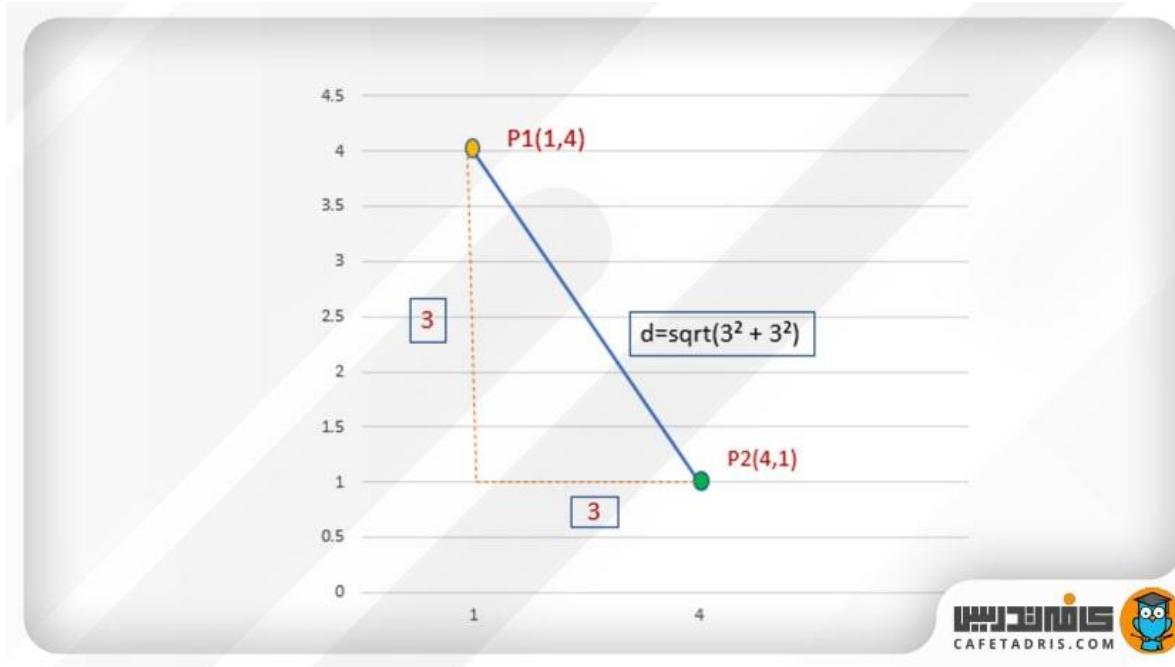
$$a^2 + b^2 = c^2$$

$$\begin{aligned} \rightarrow c^2 &= a^2 + b^2 \\ \rightarrow c &= \sqrt{a^2 + b^2} \end{aligned}$$

فرمول فاصله‌ی اقلیدسی از قضیه‌ی فیثاغورس مشتق شده است:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

را با استفاده از فرمول فاصله‌ی اقلیدسی می‌توانیم فاصله‌ی میان دو نقطه‌ی $P_1(1,4)$ و $P_2(4,1)$ محاسبه کنیم.



$$d = \sqrt{(4 - 1)^2 + (1 - 4)^2}$$

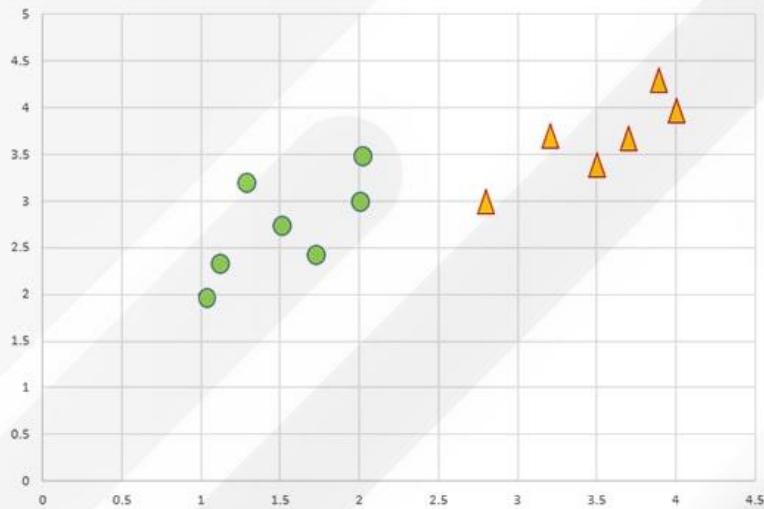
$$d = \sqrt{9 + 9}$$

$$d = 4.24$$

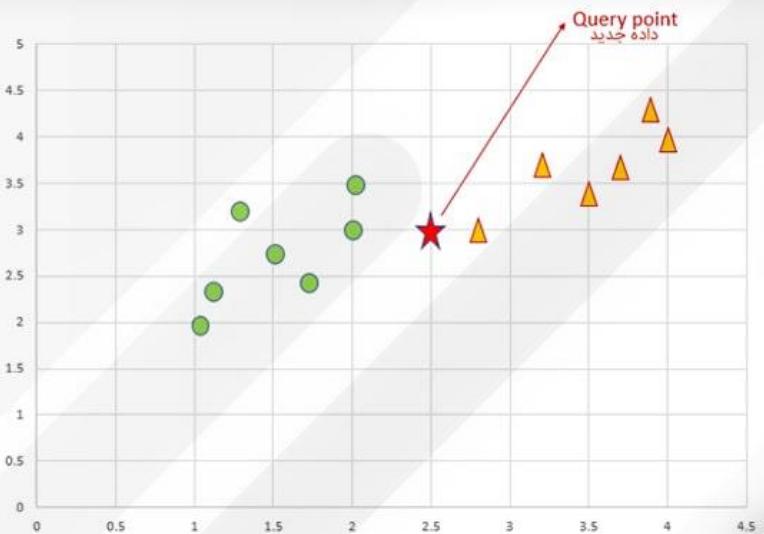
نزدیکترین همسایه K طبقه‌بندی با

را بیاموزیم. فرض کنید دو کلاس KNN ببایید نحوه‌ی طبقه‌بندی داده‌ها با استفاده از الگوریتم داریم دایره و مثلث.

در این شکل نمایش نقاط داده در فضای ویژگی‌ها آمده است:



حال باید کلاس داده‌ی جدید را پیش‌بینی کنیم (شکل ستاره). ما باید پیش‌بینی کنیم که داده‌ی جدید (شکل ستاره) به کلاس دایره تعلق دارد یا کلاس مثلث.



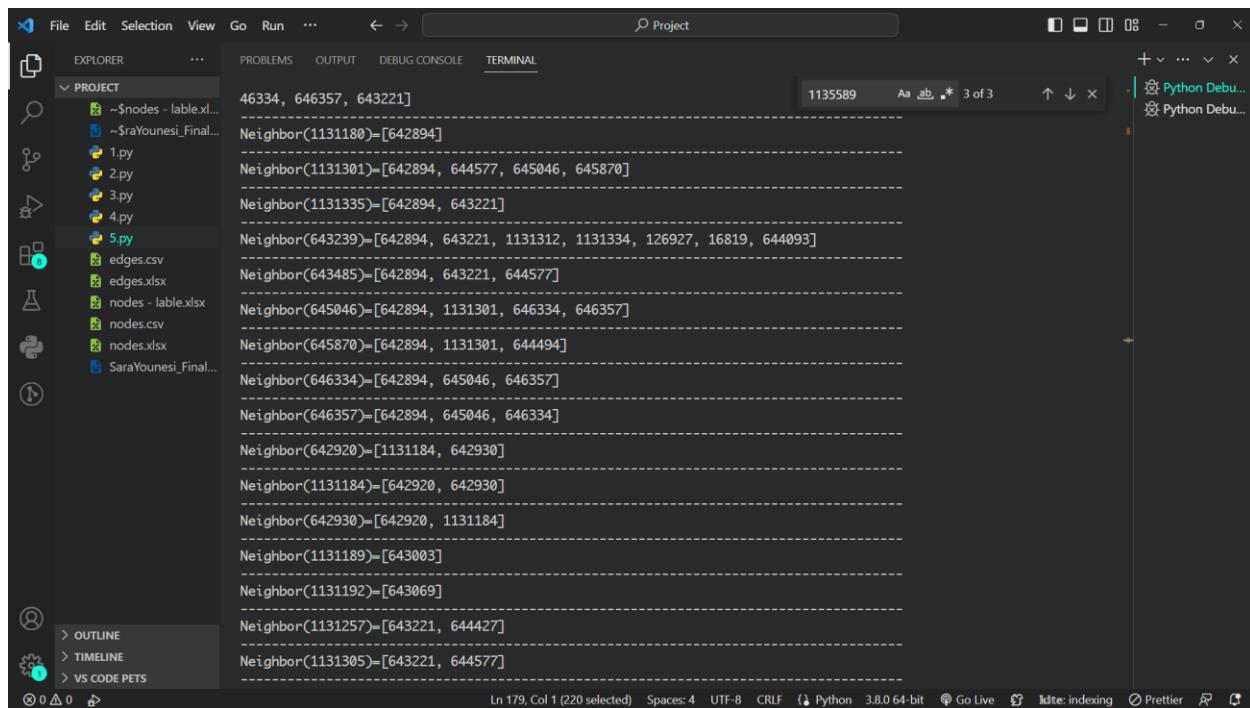
تعداد نزدیک‌ترین همسایه‌ها را نشان می‌دهد k . را تعیین کنیم k ابتدا باید مقدار

نزدیک‌ترین همسایه را براساس فاصله تعیین کنیم K در مرحله‌ی دوم باید

در این مثال، نزدیکترین همسایه‌ها در داخل بیضی آبی نشان داده شده‌اند. همان‌طور که می‌بینیم، اکثریت داده‌ها به کلاس دایره متعلق است؛ بنابراین داده‌ی جدید به کلاس دایره متعلق خواهد بود.

روی یک دیتاست KNN اعمال

```
for node in G.nodes:  
    neighbor_list=[ n for n in G.neighbors(node)]  
    print(f"Neighbor({node})={neighbor_list}")  
    print ("-----")
```



```
46334, 646357, 643221]  
Neighbor(1131180)-[642894]  
Neighbor(1131301)-[642894, 644577, 645046, 645870]  
Neighbor(1131335)-[642894, 643221]  
Neighbor(643239)-[642894, 643221, 1131312, 1131334, 126927, 16819, 644093]  
Neighbor(643485)-[642894, 643221, 644577]  
Neighbor(645046)-[642894, 1131301, 646334, 646357]  
Neighbor(645870)-[642894, 1131301, 644494]  
Neighbor(646334)-[642894, 645046, 646357]  
Neighbor(646357)-[642894, 645046, 646334]  
Neighbor(642920)-[1131184, 642930]  
Neighbor(1131184)-[642920, 642930]  
Neighbor(642930)-[642920, 1131184]  
Neighbor(1131189)-[643003]  
Neighbor(1131192)-[643069]  
Neighbor(1131257)-[643221, 644427]  
Neighbor(1131305)-[643221, 644577]
```

نتیجه تحلیل و بررسی گام پنجم

در این قسمت من از سه راه knn

```
(nx.single_source_shortest_path_length(G))  
print(nx.floyd_marshall_predecessor_and_distance(G))
```

```
print(nx.single_source_bellman_ford_path(G))
```

استفاده کرده ام که تحلیل **knn** به طور کامل در گزارش امده و موارد دیگر هم با استفاده از فرمول های آماده کتابخانه **network** استفاده شده و سپس نتایج به تفکیک در سه فایل جداگانه ذخیره شده است.

سرعت تشخیص **knn** بسیار پایین تر از موارد دیگر بوده ولی از دقت بسیار خوبی برخوردار می باشد.

اما موارد دیگر مانند مسیر کوتاه شاید از دقت مطلوب برخوردار نباشند زیرا که ما به یک نود مسیر کوتاهتری داشته باشیم که ابی باشد اما با خوش از گره های قرمز محاصره شده باشیم و انگاه به اشتباه نود ابی گزارش می شود در صورتی که احتمال داشته باشد به خوش قرمز تعلق داشته باشد بسیار بالاتر است اما در مورد نر=زدیک ترین همسایه خصوصا باب بررسی k های مختلف می توانیم به یک نتیجه همگرا شویم و نتیجه ای با دقت و صحت مطلوب دست یابیم.

روش های مبتنی بر همسایه

- مرکزیت درجه
- امتیازدهی Shell

روش های مبتنی بر مسیر.

- مرکزیت نزدیکی..
- مرکزیت بینایی..
- حفره ساختاری..

روش های تکراری یا مبتنی بر ارزش

- مرکزیت مقادیر ویژه
- PageRank

The screenshot shows a web browser displaying the NetworkX documentation for the `single_source_bellman_ford_path` function. The URL is https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.shortest_paths.weighted.single_source_bellman_ford_path.html. The left sidebar lists various NetworkX algorithms, and the main content area provides detailed documentation for this specific function, including parameters, returns, and raises sections.

The screenshot shows a web browser displaying the NetworkX documentation for the `single_source_shortest_path_length` function. The URL is https://networkx.org/documentation/networkx-1.9/reference/generated/networkx.algorithms.shortest_paths.unweighted.single_source_shortest_path_length.html. The left sidebar shows a Project Homepage and Source Code navigation, and the main content area provides detailed documentation for this function, including a warning about an unmaintained version of NetworkX.

Clustering

Community distance

2558	1136110	L3
2559	1136310	L7
2560	1136342	L3
2561	1136393	L4
2562	1136397	L4
2563	1136422	L4
2564	1136442	L5
2565	1136446	L3
2566	1136447	L3
2567	1136449	L3
2568	1136631	L3
2569	1136634	L7
2570	1136791	L7
2571	1136814	L7
2572	1137140	L3
2573	1137466	L1
2574	1138027	L1
2575	1138043	L1
2576	1138091	L2
2577	1138619	L2
2578	1138755	L2
2579	1138968	L2
2580	1138970	L3
2581	1139009	L3
2582	1139195	L3
2583	1139928	L3
2584	1140040	L4
2585	1140230	L4
2586	1140231	L4
2587	1140289	L4
2588	1140543	L4
2589	1140547	L4
2590	1140548	L4
2591	1152075	L4
2592	1152143	L1
2593	1152150	L1
2594	1152162	L1
2595	1152179	L1
2596	1152194	L1
2597	1152244	L1
2598	1152259	L1
2599	1152272	L1
2600	1152277	L1

2601	1152290	L1
2602	1152307	L1
2603	1152308	L1
2604	1152358	L1
2605	1152379	L2
2606	1152394	L2
2607	1152421	L2
2608	1152436	L1
2609	1152448	L2
2610	1152490	L1
2611	1152508	L5
2612	1152564	L5
2613	1152569	L5
2614	1152633	L5
2615	1152663	L5
2616	1152673	L5
2617	1152676	L5
2618	1152711	L3
2619	1152714	L3
2620	1152740	L3
2621	1152761	L3
2622	1152821	L3
2623	1152858	L3
2624	1152859	L3
2625	1152896	L3
2626	1152904	L5
2627	1152910	L5
2628	1152917	L4
2629	1152944	L4
2630	1152958	L5
2631	1152959	L4
2632	1152975	L4
2633	1152991	L4
2634	1153003	L4
2635	1153014	L7
2636	1153024	L7
2637	1153031	L7
2638	1153056	L7
2639	1153064	L7
2640	1153065	L4
2641	1153091	L4
2642	1153097	L4
2643	1153101	L4
2644	1153106	L7
2645	1153148	L7

2646	1153150	L7
2647	1153160	L7
2648	1153166	L7
2649	1153169	L7
2650	1153183	L7
2651	1153195	L7
2652	1153254	L7
2653	1153262	L7
2654	1153264	L7
2655	1153275	L7
2656	1153280	L7
2657	1153287	L7
2658	1153577	L7
2659	1153703	L7
2660	1153724	L7
2661	1153728	L4
2662	1153736	L4
2663	1153784	L4
2664	1153786	L4
2665	1153811	L4
2666	1153816	L4
2667	1153853	L5
2668	1153860	L5
2669	1153861	L5
2670	1153866	L5
2671	1153877	L5
2672	1153879	L5
2673	1153889	L5
2674	1153891	L5
2675	1153896	L3
2676	1153897	L3
2677	1153899	L3
2678	1153900	L3
2679	1153922	L3
2680	1153933	L3
2681	1153942	L3
2682	1153943	L3
2683	1153945	L3
2684	1153946	L3
2685	1154012	L4
2686	1154042	L4
2687	1154068	L4
2688	1154071	L4
2689	1154074	L4
2690	1154076	L4

2691	1154103	L4
2692	1154123	L4
2693	1154124	L4
2694	1154169	L4
2695	1154173	L4
2696	1154176	L2
2697	1154229	L4
2698	1154230	L4
2699	1154232	L4
2700	1154233	L4
2701	1154251	L4
2702	1154276	L4
2703	1154459	L4
2704	1154500	L4
2705	1154520	L3
2706	1154524	L6
2707	1154525	L6
2708	1155073	L6