



AMRITA
VISHWA VIDYAPEETHAM
DEEMED TO BE UNIVERSITY

19CSE337 Social Networking Security

Lecture 10





Topics to Discuss

- Eigenvector Centrality



Eigenvector Centrality

- A limitation of the degree measure is that it gives the same weight to all the neighbors of a node when computing its importance.
- However, it may make more sense to give a larger weight to nodes that are themselves important.
- In a social network, for example, one node may be important because it has social ties with few but important nodes (instead of just participating in many ties).
- Eigenvector centrality is a measure of influence that takes into account the number of links each node has and the number of links their connections have, and so on throughout the network.

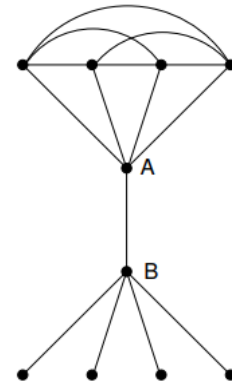


Eigenvector Centrality

- Like degree centrality, EigenCentrality measures a node's influence based on the number of links it has to other nodes in the network.
- EigenCentrality then goes a step further by also taking into account how well connected a node is, and how many links their connections have, and so on through the network.
- By calculating the extended connections of a node, EigenCentrality can identify nodes with influence over the whole network, not just those directly connected to it.
- EigenCentrality is a good 'all-round' SNA score, handy for understanding human social networks, but also for understanding networks like malware propagation.

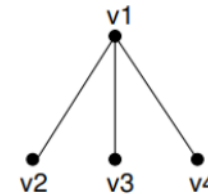
Eigenvector Computation

- Nodes A and B both have degree 5.
- The four nodes (other than A) to which B is adjacent may be unimportant (since they don't have any interactions among themselves).
- So, A seems more central than B.
- Eigenvector centrality was proposed to capture this.



Eigenvector Computation

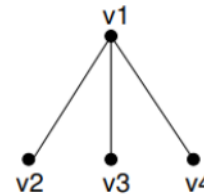
- Consider the following graph and its adjacency matrix.
- We need to find the centrality of each node as function of centrality value of its neighbors.
- The simplest function is the sum of the centrality values.
- The formula to calculate eigenvector centrality of node i is $x_i = 1/\lambda \sum x_j$.



0	1	1	1
1	0	0	0
1	0	0	0
1	0	0	0

Eigenvector Computation

- Let x_i denote the centrality of node v_i , $i=1,2,3,4$.
- To find x_1, x_2, x_3, x_4 solve the following equations
 - $x_1 = 1/\lambda(x_2 + x_3 + x_4)$
 - $x_2 = 1/\lambda(x_1)$
 - $x_3 = 1/\lambda(x_1)$
 - $x_4 = 1/\lambda(x_1)$
- Must avoid trivial solution $x_1 = x_2 = x_3 = x_4 = 0$
- So $x_i > 0$, for at least one $i \in \{1, 2, 3, 4\}$



0	1	1	1
1	0	0	0
1	0	0	0
1	0	0	0

Eigenvector Computation

- Rewriting above equations

- $\lambda x_1 = (x_2 + x_3 + x_4)$

- $\lambda x_2 = (x_1)$

- $\lambda x_3 = (x_1)$

- $\lambda x_4 = (x_1)$

Matrix version:

$$\lambda \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}$$

- ie; $\lambda x = Ax$, λ is the eigen value of matrix A and x is the corresponding eigen vector (centrality of nodes!).

- Perron-Frobenius Theorem

If a matrix A has non-negative entries and is symmetric, then all the values in the eigenvector corresponding to the principal eigen value of A are positive.



Eigenvector Computation

Algorithm: Eigenvector centrality

Input: Adjacency matrix of A representing undirected graph $G=(V,E)$.

Output: The eigenvector centrality of each node of G .

Steps:

- Compute principal eigen value λ of A .
- Compute eigenvector corresponding to λ .
- Each x value gives eigenvector centrality of corresponding node of G .



Eigenvector Computation

- When we solve characteristic equation of matrix A, we get $\lambda = -\sqrt{3}, 0, 0, \sqrt{3}$.
- Principal eigen value is $\sqrt{3}$.
- Eigenvector corresponding to principal eigen value is $[0.707, 0.408, 0.408, 0.408]^T$
- Node v1 has higher centrality.

Eigenvector Computation using NetworkX

Spyder (Python 3.9)

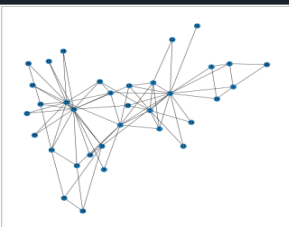
File Edit Search Source Run Debug Consoles Projects Tools View Help

C:\Users\mails\.spyder-py3\untitled0.py

```
1 from pylab import *
2 import networkx as nx
3 from operator import itemgetter
4 # G is the Karate Club Graph, the famous social graph published in 1977
5 G = nx.karate_club_graph()
6 plt.figure(figsize=(15, 15))
7 nx.draw_networkx(G, with_labels=True)
8
9 #calculate and print degree centrality
10 eig_centrality = nx.eigenvector_centrality(G)
11 sorted_degree = sorted(eig_centrality.items(),key=itemgetter(1),
12                        reverse=True)
13 for d in sorted_degree[:5]:
14     print(d)
15
16
17
18
```

C:\Users\mails\.spyder-py3

38 %



Help Variable Explorer Plots Files

Console 1/A X

```
In [3]: runfile('C:/Users/mails/.spyder-py3/untitled0.py', wdir='C:/Users/
mails/.spyder-py3')
(33, 0.373371213013235)
(0, 0.3554834941851943)
(2, 0.31718938996844476)
(32, 0.3086510477336959)
(1, 0.2659538704545025)

In [4]: |
```

IPython console History

LSP Python: ready conda: base (Python 3.9.7) Line 15, Col 1 UTF-8 CRLF RW Mem 99%

15:14
18-01-2022



Thanks.....