

## 5. Link Analysis



# Outline

---

- **Link Analysis Concepts**
- **Metrics for Analyzing Networks**
- **PageRank**
- **HITS**
- **Link Prediction**

# Link Analysis Concepts

---

- **Link**

A relationship between two entities

- **Network or Graph**

A collection of entities and links between them

- **Link Analysis or Mining**

Using links to establish higher-order relationships among entities (such as relative importance in network, isolation from other entities, similarity, etc.)

# Link Analysis Tasks

---

- **Link-based Object Classification (LOC)**
  - Assign class labels to entities based on their link characteristics
  - E.g. Iterative classification, relaxation labeling
- **Link-based Object Ranking (LOR)**
  - Associate a relative quantitative assessment with each entity using link-based measures
  - E.g. PageRank, HITS, SimRank
- **Link prediction**
  - Extrapolating knowledge/pattern of links in a given network to deduce novel links that are plausible, and may occur in the future
  - E.g. Recommendation systems, infrastructure planning

# Outline

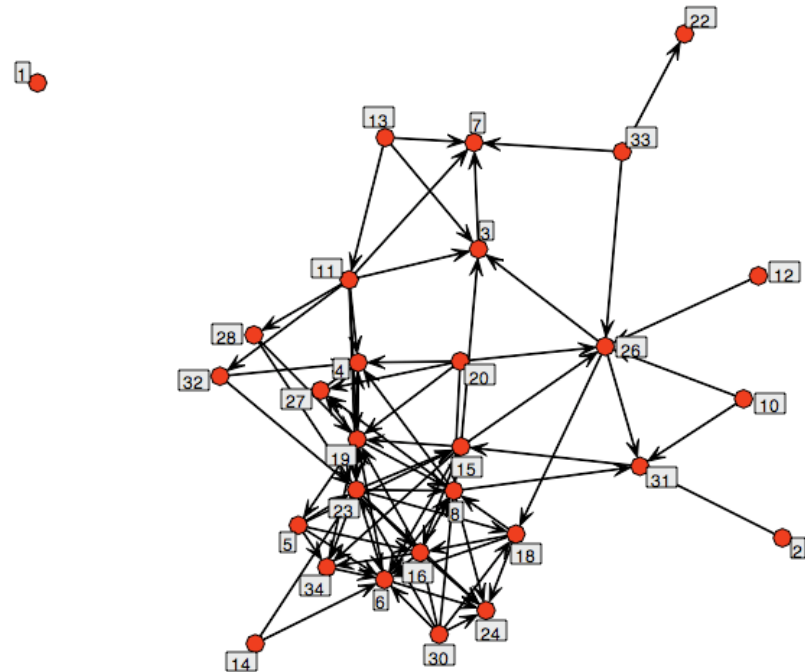
---

- Link Analysis Concepts
- **Metrics for Analyzing Networks**
- PageRank
- HITS
- Link Prediction

# Metrics for Analyzing Networks

---

- Analysis of relationships and information flow between individuals, groups, organizations, servers, and other connected entities
- Social Network Analysis (SNA): Representation of social networks with people as nodes and relationships between them as links in a graph
- SNA is relevant to advertising, national security, medicine, geography, politics, social psychology, etc.



# Network Metrics in R: Setup

- **Setup in R**

- Install and load SNA package in R
- Create a test graph (10 nodes, edges generated randomly)

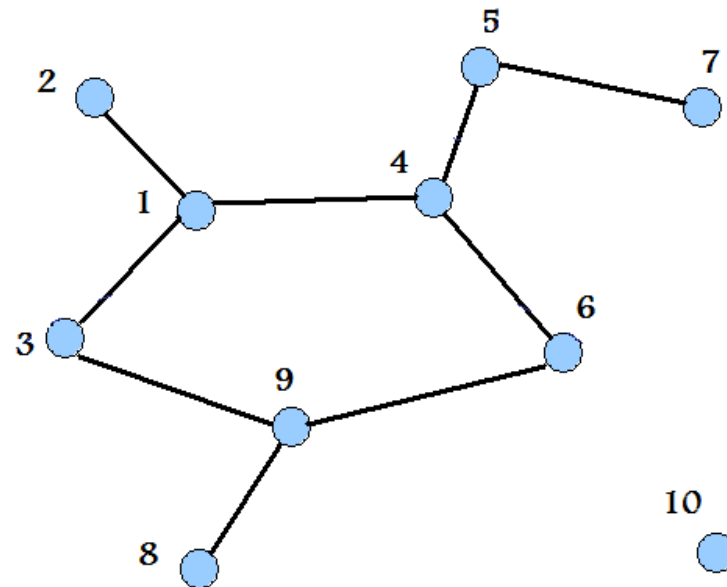
[illegible]

# Network Metrics in R: Overview

---

- **Different Social Network Metrics in R**

- Degree
- Density
- Connectedness
- Betweenness Centrality
- Egocentricity
- Closeness Centrality



A randomly generated 10-node graph  
representing, say, a social network



# Network Metrics in R: Degree

---

- **Degree**

- The degree of a node is the number of edges incident on it
- This measure is the simplest indicator of how connected a node is within a graph
- In a directed graph, *in-degree* is the no. of incoming edges, and *out-degree* the no. of outgoing ones
- For undirected graphs, total degree = in-degree + out-degree

- **Example: degree()**

```
> #vertices 1 and 4 can tolerate more vertex/link failures
> degree(graph)
[1] 6 2 4 6 4 4 2 2 6 0
```

- Here, node 1 is connected to nodes 2, 3 and 5 via undirected edges, hence leading to a total degree of 6
- Node 10 is not connected to any other node, so it has degree 0

# Network Metrics in R: Density

---

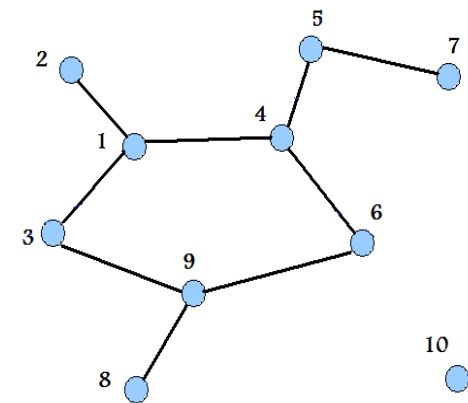
- **Density**

- The density of a graph is the number of existing edges divided by the number of possible ones (assuming no duplicates or loops)
- A graph with higher density is more strongly connected, and in general can better resist link failures

- **Example: density()**

```
> #gden is used to find the density of graph  
> gden(graph,mode="graph")  
[1] 0.4
```

- Total no. of possible edges (for 10 nodes):  
 $[10 * (10 - 1)] / 2 = 90 / 2 = 45$
- But the graph has only 18 edges
- Therefore, the density is  $18 / 45 = 0.4$



# Network Metrics in R: Connectedness

---

- **Connectedness**

- Krackhardt's *connectedness* for a digraph (directed graph)  $G$  is equal to the fraction of all *dyads* (a group of two nodes),  $u$  and  $v$ , such that there exists an undirected path from  $u$  to  $v$  in  $G$
- A graph with higher connectedness is considered to be more resistant to link failures

- **Example: connectedness()**

- The R function `connectedness` takes one or more graphs and returns the Krackhardt connectedness scores

```
> #when connectedness is 1, then the graph given is
  fully connected
> #a connectedness of 1 also gives is.connected value
  to be TRUE
> #a connectedness of 0 indicate completely isolated
  vertices
> #is.connected is used for strict connectedness
> is.connected(graph)
[1] FALSE
> connectedness(graph)
[1] 0.8
```

In our 10-node graph, nodes 1-9 are each connected to 8 other nodes, and node 10 is not connected to any.

So the connectedness of the graph is:

$$\frac{8+8+8+8+8+8+8+8+8+0}{9+9+9+9+9+9+9+9+9+9}=0.8$$

# Network Metrics in R: Betweenness

---

- **Betweenness Centrality**

- A measure of the degree to which a given node lies on the shortest paths (geodesics) between other nodes in the graph
- For node  $v$  in graph  $G$ , betweenness centrality ( $C_b$ ) is defined as:

$$C_b(v) = \sum_{s,t \neq v} \frac{\Omega_v(s,t)}{\Omega(s,t)}$$

where  $\Omega(s,t)$  is the number of distinct geodesics from  $s$  to  $t$  and  $\Omega_v(s,t)$  is the number of geodesics from  $s$  to  $t$  that pass through  $v$ .

- A node has high betweenness if the shortest paths (geodesics) between many pairs of other nodes in the graph pass through it
- Thus, when a node with high betweenness fails, it has a greater influence on the information flow in the network

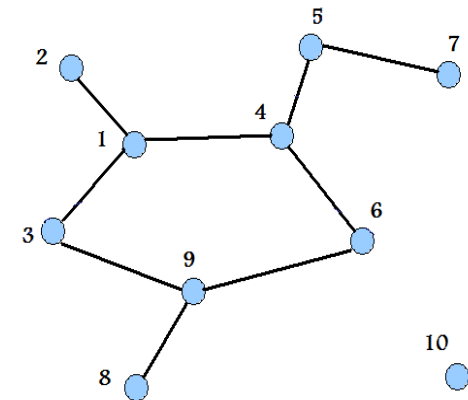
# Network Metrics in R: Betweenness

- **Example: betweenness()**

```
> #Here node 4 has the highest betweenness
> betweenness(graph)
[1] 20 0 8 28 14 12 0 0 16 0
```

- Note that nodes 2, 7, 8 and 10 are not in any of the geodesics
- Path lengths/geodesic distances can be calculated using geodist()

```
> geo=geodist(graph)
> geo$gdist
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,]  0    1    1    1    2    2    3    3    2    Inf
[2,]  1    0    2    2    3    3    4    4    3    Inf
[3,]  1    2    0    2    3    2    4    2    1    Inf
[4,]  1    2    2    0    1    1    2    3    2    Inf
[5,]  2    3    3    1    0    2    1    4    3    Inf
[6,]  2    3    2    1    2    0    3    2    1    Inf
[7,]  3    4    4    2    1    3    0    5    4    Inf
[8,]  3    4    2    3    4    2    5    0    1    Inf
[9,]  2    3    1    2    3    1    4    1    0    Inf
[10,] Inf  Inf  Inf  Inf  Inf  Inf  Inf  Inf  Inf    0
```



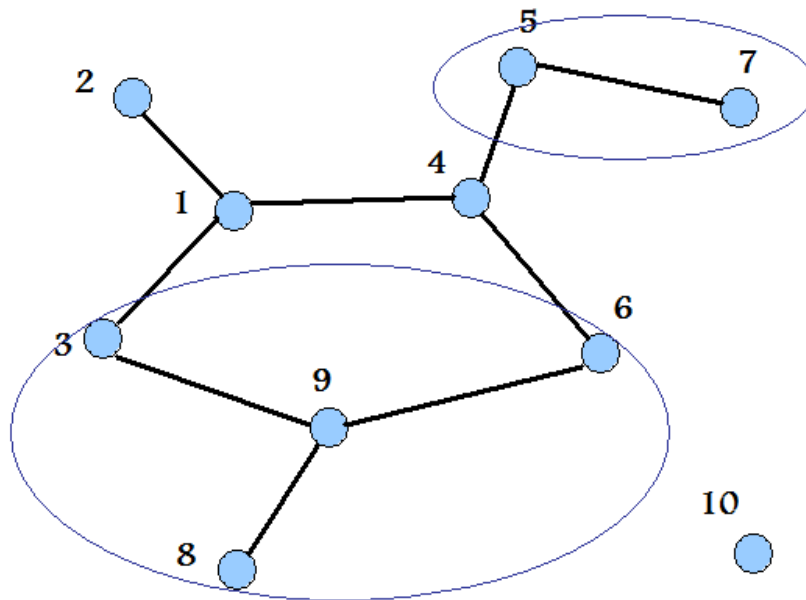
- It could be inferred that node 5 requires two hops to reach node 1 and node 10 is not reachable by any other node

# Network Metrics in R: Egocentricity

---

- **Egocentric Network**

- The egocentric network (or ego net) of vertex  $v$  in graph  $G$  is defined as the subgraph of  $G$  induced by  $v$  and its neighbors
- It can be used to compute metrics over a local neighborhood, especially useful when dealing with large networks



As depicted in this figure, the egocentric network of 9 has nodes 3, 6 and 8 (in addition to 9). Similarly, the ego net of 7 includes node 5.

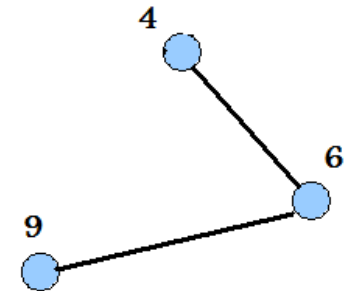
Egocentric networks for nodes 9 and 7

# Network Metrics in R: Egocentricity

---

- **Example: ego.extract()**

```
> #ego.extract takes one or more input graphs and
  returns a list containing the egocentric networks
  centered on vertices named in ego, using adjacency
  rule neighborhood to define inclusion.
> ego.extract(graph, 6)
$`6`
      [,1] [,2] [,3]
[1,]  0    1    1
[2,]  1    0    0
[3,]  1    0    0
```



- The ego-centric network of node 6 has nodes 6, 4 and 9
- Note that the sub-graph extracted in this example has the original nodes 6, 4, 9 renamed to 1, 2, 3, respectively
- Looking at the adjacency matrix, it can be inferred that node 6 is connected to both nodes 4 and 9, whereas nodes 4 and 9 are not directly connected to each other

# Network Metrics in R: Closeness

---

- **Closeness Centrality**

- Closeness Centrality (CLC) is a category of measures that rate the centrality of a node by its closeness (distance) to other nodes
- CLC of a node  $v$  is defined as:

$$CLC(v) = \frac{|V| - 1}{\sum_{i, v \neq v_i} distance(v, v_i)}$$

where  $|V|$  is the number of nodes in the given graph and  $v_i$  is the node  $i$  of the given graph.

- Closeness Centrality decreases if either the number of nodes reachable from the node in question decreases, or the distances between the nodes increases



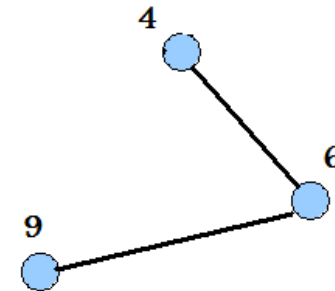
# Network Metrics in R: Closeness

- **Example: closeness()**

- The 10-node graph we have been using has one disconnected node; the resulting infinite distances thus created invalidate any aggregate measure over all nodes such as Closeness Centrality
- So, we choose a sub-graph – the egocentric network of node 6

```
> #closeness centrality measures how many steps are
  required to access every other vertex from a given
  vertex
> closeness(graph)
[1] 0 0 0 0 0 0 0 0 0 0
> #We now consider a sub-graph of the graph
  generated for easy understanding of closeness
> graph1=ego.extract(graph,6)
> graph1
$`6`
      [,1] [,2] [,3]
[1,]     0     1     1
[2,]     1     0     0
[3,]     1     0     0

> closeness(graph1)
      6
[1,] 1.0000000
[2,] 0.6666667
[3,] 0.6666667
```



The closeness centrality of node 6 is:

$$CLC(6) = (3-1) / (1+1) = 1$$

Incidentally, this means node 6 can reach all other nodes in one hop.

Now, considering node 4:

$$CLC(4) = (3-1) / (1+2) = 2 / 3 \\ = 0.667$$

Similarly for node 9:

$$CLC(9) = 0.667$$

# Outline

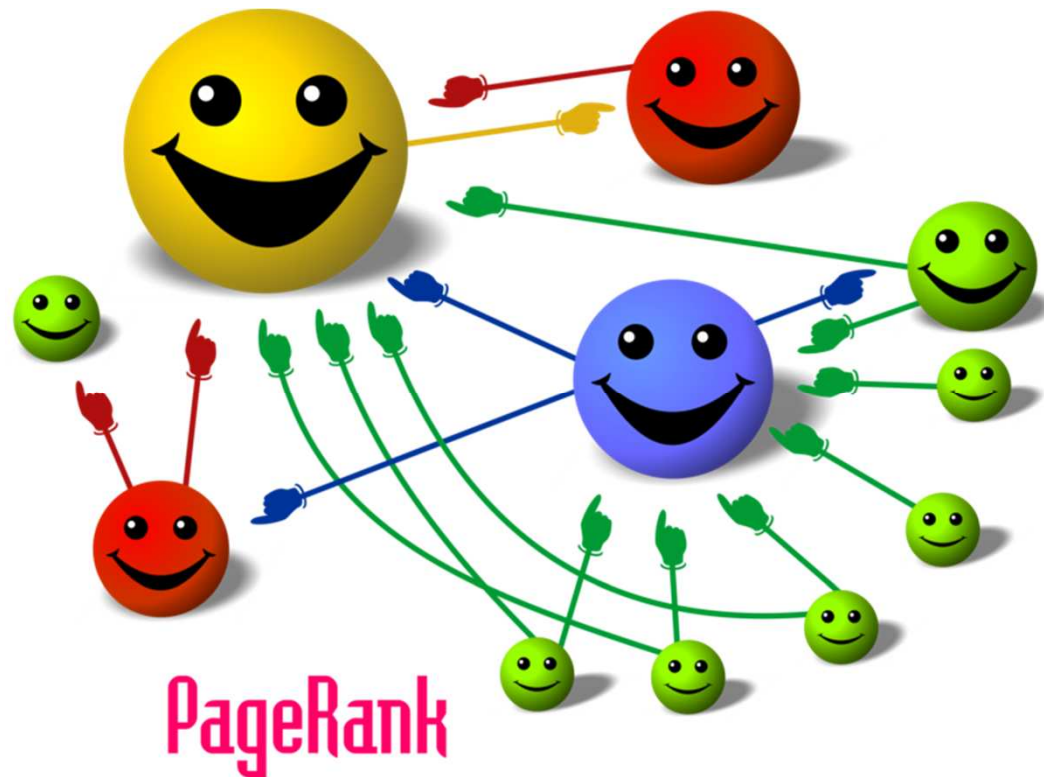
---

- Link Analysis Concepts
- Metrics for Analyzing Networks
- **PageRank**
- HITS
- Link Prediction

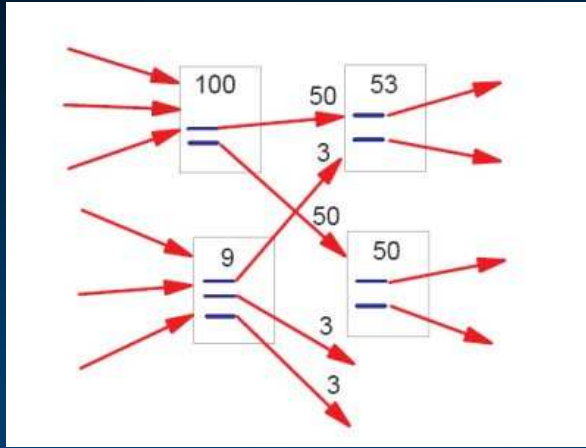
# PageRank

---

- How does Google® rank web pages in order to provide meaningful search results?

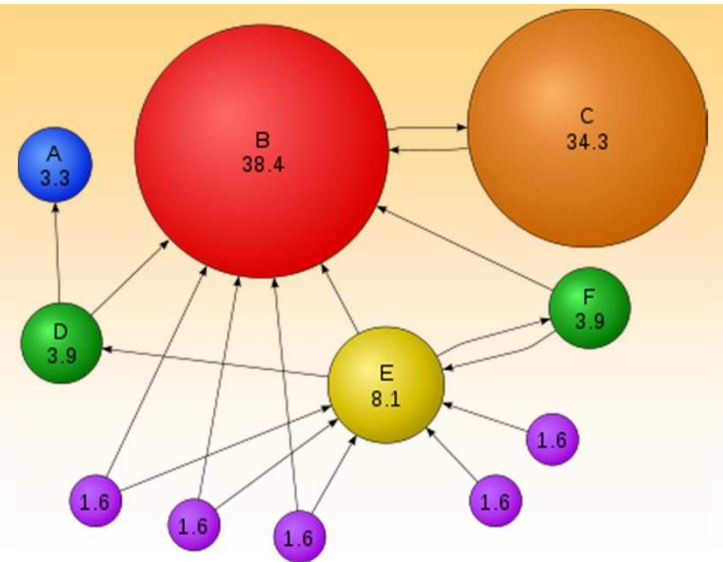


# The PageRank Algorithm



PageRank is an algorithm that addresses the LBR problem (Link-Based Object Ranking). It assigns numerical ranks to pages based on backlink counts and ranks of pages providing those backlinks.

The algorithm considers a model in which a user starts at a webpage and performs a “random walk” by following links from the page he is currently in. To start another such walk, a new webpage may be opened occasionally. PageRank of a webpage is the probability of that webpage being visited on a particular random walk.



# The PageRank Algorithm

$$PR(A) = (1 - d) + d \left( \frac{PR(P_1)}{deg(P_1)^+} + \frac{PR(P_2)}{deg(P_2)^+} + \dots + \frac{PR(P_n)}{deg(P_n)^+} \right)$$

PageRank of a page 'u' is defined as the sum of ratios of PageRank of all webpages ( $v_1, v_2, \dots, v_n$  providing backlinks to u) to the backlink count of all such pages.

- Damping factor 'd', to take into account the probability of a user beginning a new random walk.
- For every page  $P_v$  providing a backlink to  $P_u$ , find the number of outlinks of  $P_v$  [ $deg(P_v)^+$ ] and the PageRank [ $PR(P_v)$ ].
- For each  $P_v$ , find the ratio of the PageRank to the outlink count of  $P_v$ .
- Compute the sum over all such pages providing backlinks to  $P_u$ .

## PageRank Notation

Symbol	Meaning
$P_u$	A webpage 'u'
$d$	Damping factor- The Probability that the user opens a new webpage to begin a new random walk
$PR(P_u)$	PageRank of the page 'u'
$deg(P_u)^-$	The number of links coming in to a page $P_u$ (in-degree of $P_u$ )
$deg(P_u)^+$	The number of links going out of a page $P_u$ (out-degree of $P_u$ )
$N(P_u)^-$	Set of pages that point to $P_u$ (the in-neighborhood of $P_u$ )
$N(P_u)^+$	Set of pages a webpage $P_u$ points to (the out-neighborhood of $P_u$ )
$W$	A hyperlink matrix representing the network, whose entries constitute the fractional PageRank contributions
$x$	Eigen vector containing the ranks for each vertex in the network.

# The Power Method

---

- **Power Method**

- The power method is a recursive method used to compute an eigen vector of eigen value 1 of a square matrix  $W$
- The  $W$  matrix is similar to an adjacency matrix representation of a graph, except that instead of using Boolean values to indicate presence of links, we indicate the fraction of rank contribution for a link connecting two vertices in the graph

- **Calculating PageRank**

- When computing the PageRank of page  $P_u$ , with a backlink from  $P_v$ , the corresponding entry in  $W$  is:

$$W_{u,v} = \left( \frac{1}{deg(P_v)^+} \right)$$

This value denotes the fraction of  $PR(P_v)$  contributed towards  $PR(P_u)$ . Each column in  $W$  must sum to a total PageRank value of 1, since the sum of all fractional PageRank contributions to a page must sum to 1.

# The Power Method

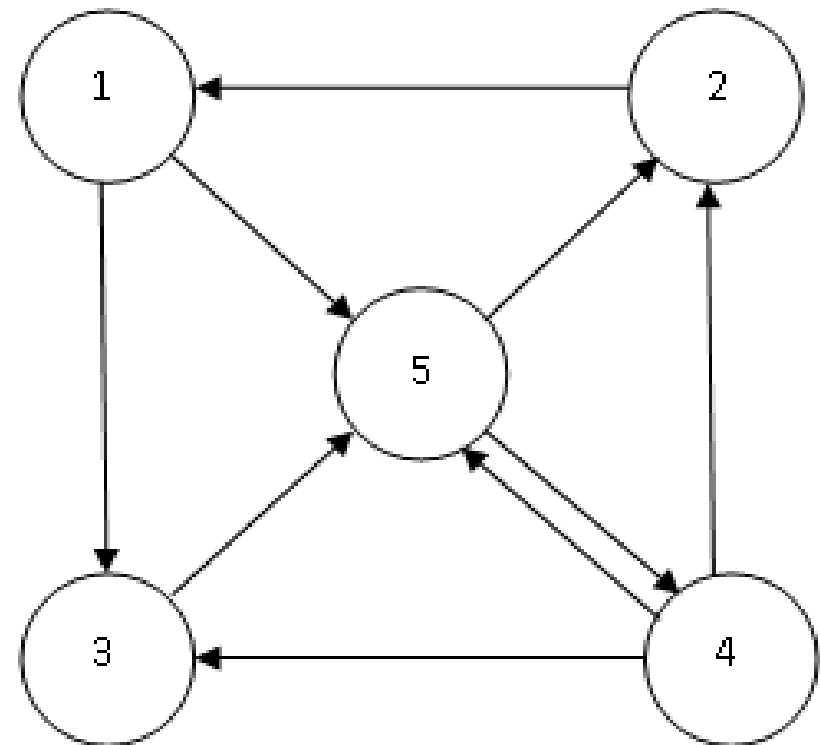
For the graph in the figure below, the matrix 'W' is calculated as follows

$$W = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{3} & \frac{1}{2} \\ \frac{1}{2} & 0 & 0 & \frac{1}{3} & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} \\ \frac{1}{2} & 0 & 1 & \frac{1}{3} & 0 \end{bmatrix}$$

$$Wx = \lambda x$$

- Using the W matrix, we need to solve for  $\lambda$ , where  $\lambda$  is the eigenvalue of the eigenvector  $x$
- $x$  is found using the equation above and here,

$$x = [\text{PR}(1) \text{ PR}(2) \text{ PR}(3) \text{ PR}(4) \text{ PR}(5)]^T$$





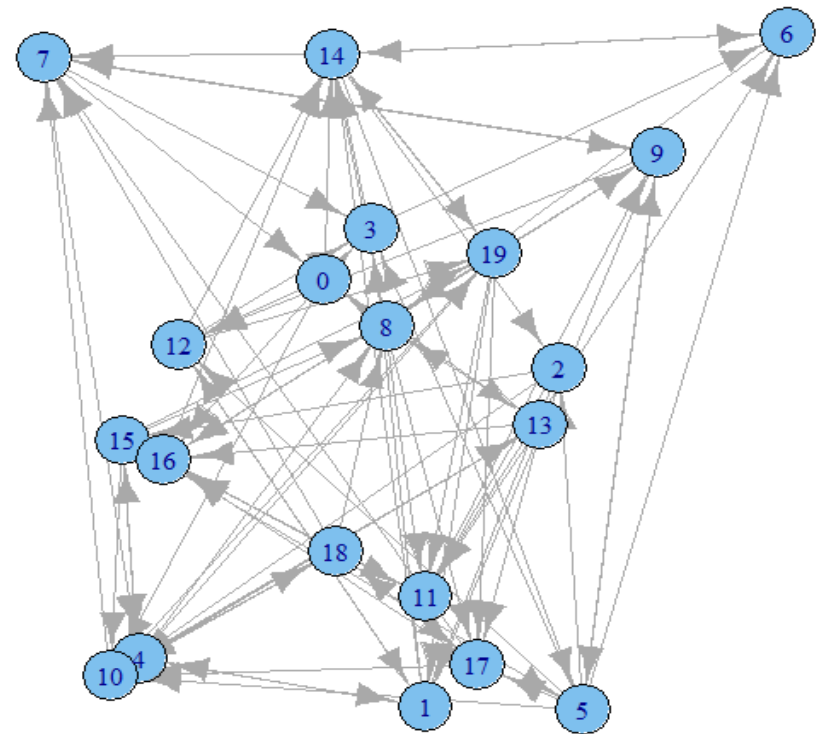
# PageRank in R

```
> library(igraph)
```

The 'igraph' package contains the function 'page.rank' that is capable of taking a graph object as an input and computing the PageRank of the vertices in the graph object.

```
> g <- random.graph.game(20, 5/20, directed=TRUE)
> page.rank(g)$vector
[1] 0.03686596 0.02552204 0.05456715 0.03623592
0.03608888 0.01366011 0.04652149 0.04414025
0.08562624 0.06189504 0.01751444 0.08791547
0.05319910 [14] 0.05706331 0.04290224 0.06482654
0.05538546 0.06322104 0.04124198 0.07560734
```

- The above function call creates a directed random graph with 20 vertices.
- This is stored on the graph object 'g' with an edge between two vertices occurring with probability of 5/20.



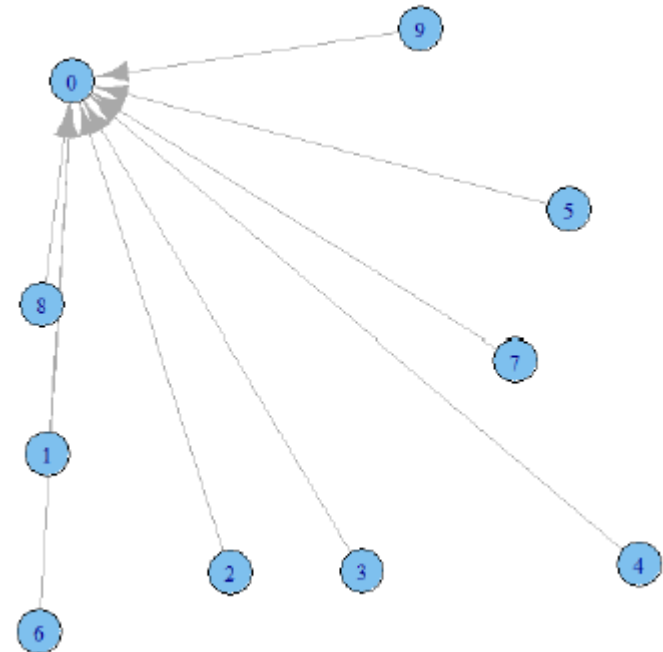


# PageRank in R

Depiction of nodes with their PageRank.

```
> g2 <- graph.star(10)
> page.rank(g2)$vector
[1] 0.68005764 0.03554915 0.03554915 0.03554915
0.03554915 0.03554915 0.03554915 0.03554915
0.03554915 0.03554915
> plot(g)
> plot(g2)
```

- The 'graph.star' function creates a star graph 'g2'.
- In this every single vertex is connected to only the center vertex.
- This is used to depict the vertex that has the highest PageRank in our simulation.



# Outline

---

- Link Analysis Concepts
- Metrics for Analyzing Networks
- PageRank
- **HITS**
- Link Prediction

# HITS: Agenda

---

HITS Introduction and Overview

Authority and Hub

HITS Preprocessor

Adjacency Matrix

Update and Normalize Vectors

Convergence of HITS

Pseudocode and Time Complexity

R Code

Strengths and Weaknesses

# HITS: Introduction

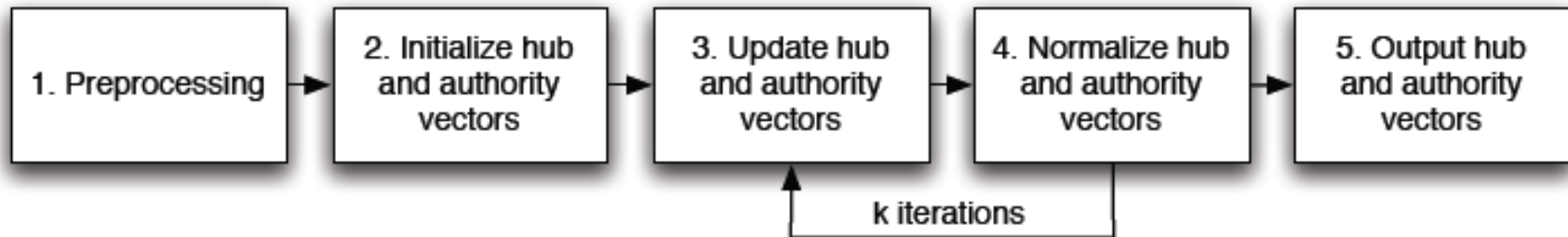
---

- **Hyperlink-Induced Topic Search**
- **Developed by Jon Kleinberg (1999)**
- **“Runtime” algorithm**
  - Applied only when a user submits a query
- **Models linked web pages as a directed graph**



# HITS: Algorithm Overview

---

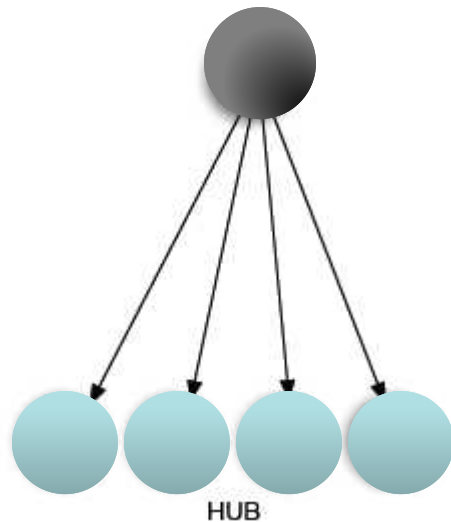
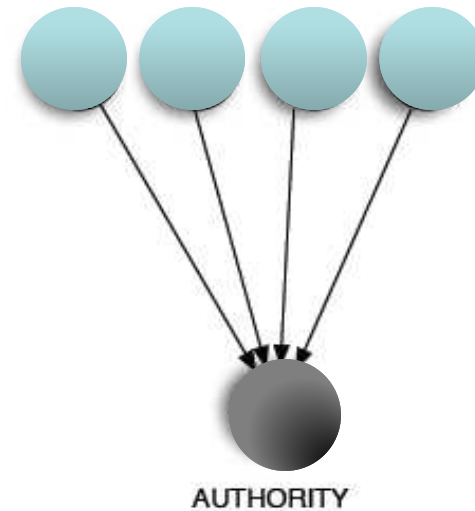


- **Inputs:**
  - An adjacency matrix representing a collection of items
  - A value defining the number of iterations to perform
- **Outputs:**
  - Hub and Authority score vectors

# Authority and Hub

---

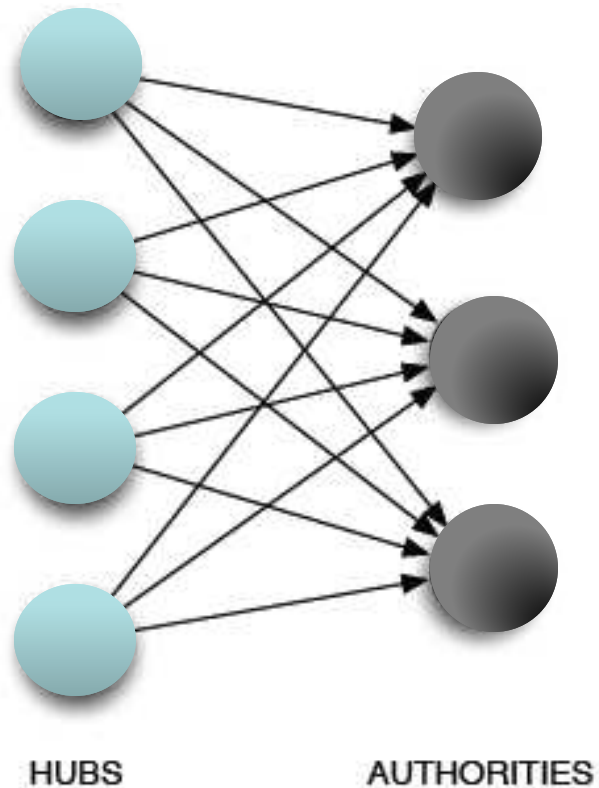
- **Authority** – A vertex is considered an authority if it has many pages linking to it (High Indegree)



- **Hub** – A vertex is considered a hub if it points to many other vertices (High Outdegree)

# Identifying the Most Relevant Pages

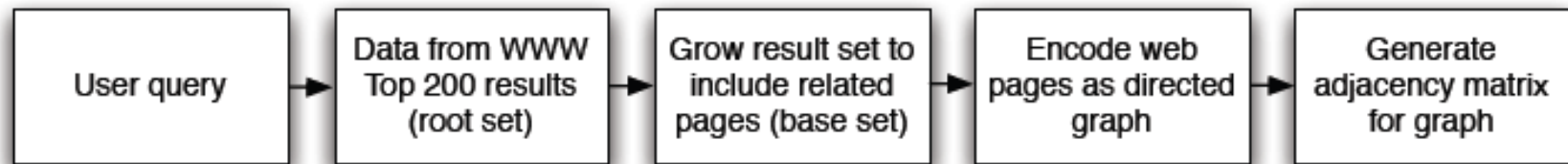
---



- **Generally the pages considered authoritative on the subject are most relevant**
- **Finding the most relevant results is commonly found in dense subgraphs, primarily bipartite graphs**

# HITS Preprocessor

---



- **HITS algorithm must preprocess to limit the set of web pages taken into consideration**
- **Root Set – Set of pages most relevant to user’s query**
- **Base Set – “Grown” set of pages related to query**
- **Encodes the adjacency matrix to be used by the algorithm**



# Constructing the Adjacency Matrix

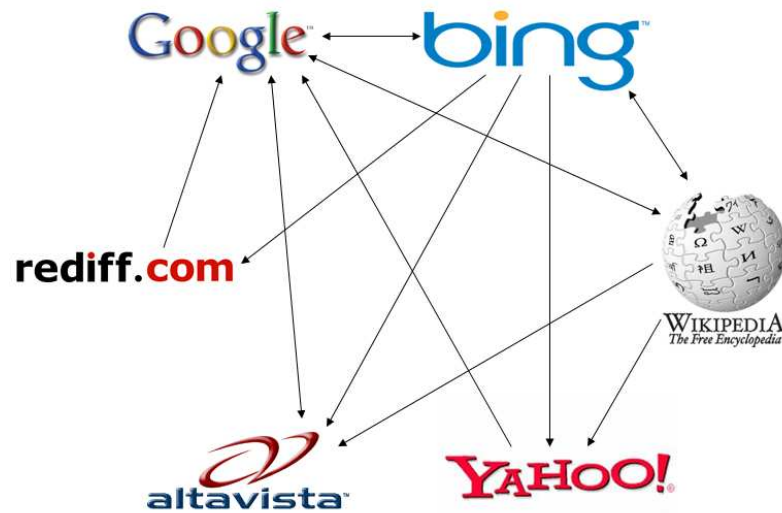
---

An adjacency matrix is defined such that:

$$A_{\{ij\}} = \begin{cases} 1, & \text{if } e_{\{ij\}} \in E \\ 0, & \text{otherwise} \end{cases}$$

- **For each position in the adjacency matrix:**
  - Check if there is a directed edge between the 2 vertexes
  - If there is then place a 1 in that position of the matrix
  - Otherwise place a 0 in that position of the matrix

# Adjacency Matrix (Example)



A graph for a query, “search engine”, is displayed to the left. The adjacency matrix associated with the graph can be found below.

$$A_{\{\text{rediff}, \text{Google}\}} = 1$$

$$A_{\{\text{Google}, \text{rediff}\}} = 0$$

While there is a hyperlink from rediff to Google, there is not one from Google to rediff

	Wiki	Google	Bing	Yahoo	Altavista	Rediff
Wiki	0	1	1	0	0	0
Google	1	0	1	1	1	1
Bing	0	1	0	0	0	0
Yahoo	0	0	1	0	1	0
Altavista	0	1	1	0	0	0
Rediff	0	0	1	0	0	0

# Updating Hub and Authority

---

- For each web page the hub and authority scores are initially set to 1
- For each iteration of the algorithm the hub and authority scores are updated

$$\mathbf{X} = (x_1, x_2, \dots, x_n)^T$$

$$\mathbf{X}_0 = (1, 1, 1, 1, 1, 1)^T$$

Authority Score  
Initialization

$$\mathbf{Y} = (y_1, y_2, \dots, y_n)^T$$

$$\mathbf{Y}_0 = (1, 1, 1, 1, 1, 1)^T$$

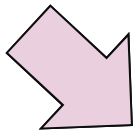
Hub Score Initialization

# Updating Hub and Authority

---

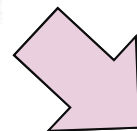
- **Update Authority Score**

- The previous iteration's hub score is used to calculate the current authority score

$$x_j^{(k)} \leftarrow \sum_{(i,j) \in E} y_i^{(k-1)}.$$

$$\mathbf{X}_i \leftarrow \mathbf{A}^T \times \mathbf{Y}_{i-1}.$$

- **Update Hub Score**

- The current iteration's authority score is used to calculate the current hub score

$$y_i^{(k)} \leftarrow \sum_{(i,j) \in E} x_j^{(k)}.$$

$$\mathbf{Y}_i \leftarrow \mathbf{A} \times \mathbf{X}_i.$$

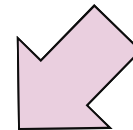
# Normalizing Hub and Authority

---

- The weights are normalized to ensure that the sum of their squares is 1
- The normalization process for Hub and Authority are practically identical

$$\sum_{x \in X} x^2 = 1.$$

$$x' = \frac{x}{\sqrt{\sum_{x \in X} (x^2)}},$$



$$X' = \{x' | x \in X\}.$$

Normalization of Hub Score

# Updating and Normalizing Authority (Example)

---

$$\begin{aligned}
 \mathbf{X}_1 &= \mathbf{A}^T \times \mathbf{Y}_0 \\
 &= \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}^T \times (1, 1, 1, 1, 1, 1)^T \\
 &= \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix} \times \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} \\
 &= \begin{pmatrix} 1 \\ 3 \\ 5 \\ 1 \\ 2 \\ 1 \end{pmatrix}
 \end{aligned}$$

$$\begin{aligned}
 \mathbf{X}'_1 &= \begin{pmatrix} \frac{1}{\sqrt{1^2+3^2+5^2+1^2+2^2+1^2}} \\ \frac{3}{\sqrt{1^2+3^2+5^2+1^2+2^2+1^2}} \\ \frac{5}{\sqrt{1^2+3^2+5^2+1^2+2^2+1^2}} \\ \frac{1}{\sqrt{1^2+3^2+5^2+1^2+2^2+1^2}} \\ \frac{2}{\sqrt{1^2+3^2+5^2+1^2+2^2+1^2}} \\ \frac{1}{\sqrt{1^2+3^2+5^2+1^2+2^2+1^2}} \end{pmatrix} \\
 &= \begin{pmatrix} \frac{1}{\sqrt{41}} \\ \frac{3}{\sqrt{41}} \\ \frac{5}{\sqrt{41}} \\ \frac{1}{\sqrt{41}} \\ \frac{2}{\sqrt{41}} \\ \frac{1}{\sqrt{41}} \end{pmatrix} \\
 &= \begin{pmatrix} 0.15617 \\ 0.46852 \\ 0.78087 \\ 0.15617 \\ 0.312348 \\ 0.15617 \end{pmatrix}
 \end{aligned}$$

# Convergence of HITS

---

- **There is no formal convergence criteria**
- **Generally the upper bound for  $k$  is 20**

Even after just 6 iterations of the “search engine” example the HITS algorithm on Authority Score you can begin to see convergence.

Iteration	Wiki	Google	Bing	Yahoo	Altavista	Rediff
0	1	1	1	1	1	1
1	0.156	0.469	0.781	0.156	0.312	0.156
2	0.204	0.388	0.777	0.204	0.347	0.204
3	0.224	0.350	0.769	0.224	0.369	0.224
4	0.232	0.332	0.765	0.232	0.378	0.232
5	0.236	0.324	0.762	0.236	0.383	0.236
6	0.238	0.320	0.761	0.238	0.385	0.238

# Pseudocode

---

**Input:**  $A$ : an adjacency matrix representing a collection of items (e.g. web pages)

**Input:**  $k$ : a natural number (number of iterations)

**Output:**  $X_k, Y_k$ : vectors of hub and authority scores for each vertex in the graph

1:  $X_0 \leftarrow (1, 1, 1, \dots, 1) \in \mathbb{R}^n$

2:  $Y_0 \leftarrow (1, 1, 1, \dots, 1) \in \mathbb{R}^n$

3: **for**  $i = 1$  **to**  $k$  **do**

4:     Apply the  $\mathcal{J}$  operation to  $(X_{i-1}, Y_{i-1})$ , to obtain new authority scores,  $X'_i$ .

5:     Apply the  $\mathcal{O}$  operation to  $(X'_i, Y_{i-1})$ , to obtain new hub scores,  $Y'_i$ .

6:     Normalize  $X_i$ , obtaining  $X'_i$ .

7:     Normalize  $Y_i$ , obtaining  $Y'_i$ .

8: **end for**

9: **return**  $(X_k, Y_k)$ .



# Time Complexity

---

1:  $\mathbf{X}_0 \leftarrow (1, 1, 1, \dots, 1) \in \mathbb{R}^n$   $O(n)$   
2:  $\mathbf{Y}_0 \leftarrow (1, 1, 1, \dots, 1) \in \mathbb{R}^n$   $O(n)$   
3: **for**  $i = 1$  **to**  $k$  **do**      Each of the following is executed  $k$  times:  
4:     Apply the  $\mathcal{J}$  operation to  $(\mathbf{X}_{i-1}, \mathbf{Y}_{i-1})$   $O(n^2 + n^{2.376})$   
5:     Apply the  $\mathcal{O}$  operation to  $(\mathbf{X}'_i, \mathbf{Y}_{i-1})$   $O(n^{2.376})$   
6:     Normalize  $\mathbf{X}_i$ , obtaining  $\mathbf{X}'_i$ .  $O(n)$   
7:     Normalize  $\mathbf{Y}_i$ , obtaining  $\mathbf{Y}'_i$ .  $O(n)$   
8: **end for**  
9: **return**  $(\mathbf{X}_k, \mathbf{Y}_k)$ .

$$= O( n + k ( n^2 + n^{2.376} + n^{2.376} + n + n ) )$$

**The total time complexity is  $O( k \cdot n^{2.376} )$**

# R Library for HITS

---

- **Library:**
  - ProximityMeasure
- **Function:**
  - HITS(G,k)
- **Inputs:**
  - G is directed adjacency matrix
  - k is the number of iterations
- **Returns:**
  - Two vector columns (hub and authority) bound together

```
1 library(igraph)
2 library(ProximityMeasure)
3 A<-matrix(c(0,1,1,0,0,0,
4             1,0,1,1,1,1,
5             0,1,0,0,0,0,
6             0,0,1,0,1,0,
7             0,1,1,0,0,0,
8             0,0,1,0,0,0),
9           nrow=6,ncol=6,
10          byrow=TRUE);
11 G<-graph.adjacency(A,
12                   mode=c("directed"),weighted=NULL);
13 k<-6;
14 op<-HITS(G,k);
15 op;
```

# Strengths and Weaknesses

---

- **Strengths**

- Two vectors (hub and authority) allow application to decide which vector is most interesting
- Highly efficient

- **Weaknesses**

- “Topic Drift”
- Manipulation of algorithm through “spam”
- Poor performance due to poor selection of  $k$

# Outline

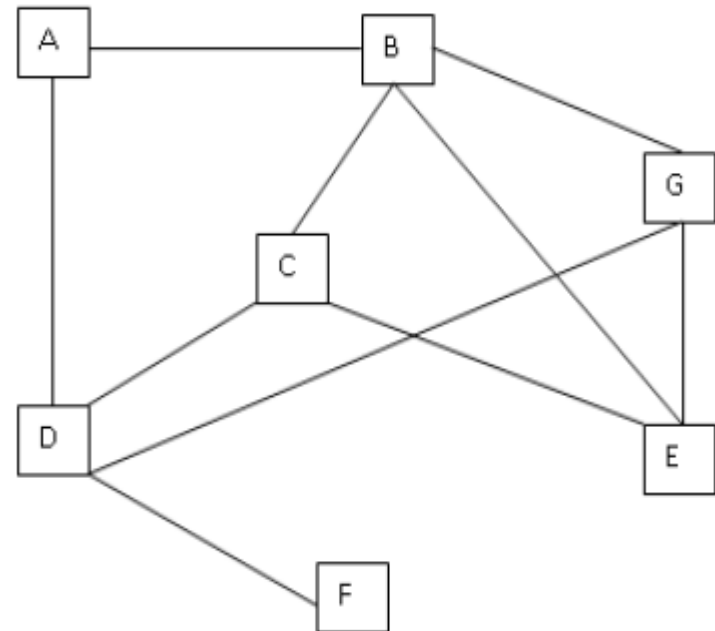
---

- Link Analysis Concepts
- Metrics for Analyzing Networks
- PageRank
- HITS
- **Link Prediction**

# Link Prediction

Given a snapshot of a social network, it is possible to infer new interactions between members who have never interacted before. This is described as the Link Prediction Problem.

- With the advent of social networks and services such as Facebook and Myspace, link analysis and prediction have become prominent terms.
- Primarily used to predict the possibility of new friends, study friend structures and co-authorship networks.



# Link Prediction

‘Core’ is the set containing vertices that are adjacent to 3 or more edges in the graph.

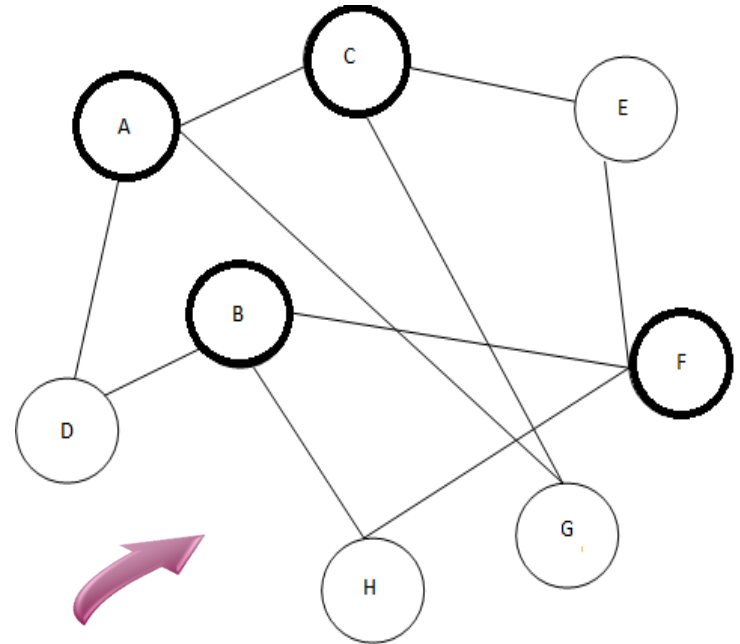
- $k_{\text{training}}$  is the number of edges a vertex in the training set has to be adjacent to in order to enter the core set.

- In the diagram, we have the training set containing vertices A to H in which the vertices A, B, C and F have more than 3 edges adjacent to them, then these edges belong to core.

## Edge list

A→C  
A→G  
A→D  
C→E  
C→G  
B→D  
B→H  
B→F  
E→F  
F→H

Diagram showing the vertices of the core set in bold outlines in the graph.



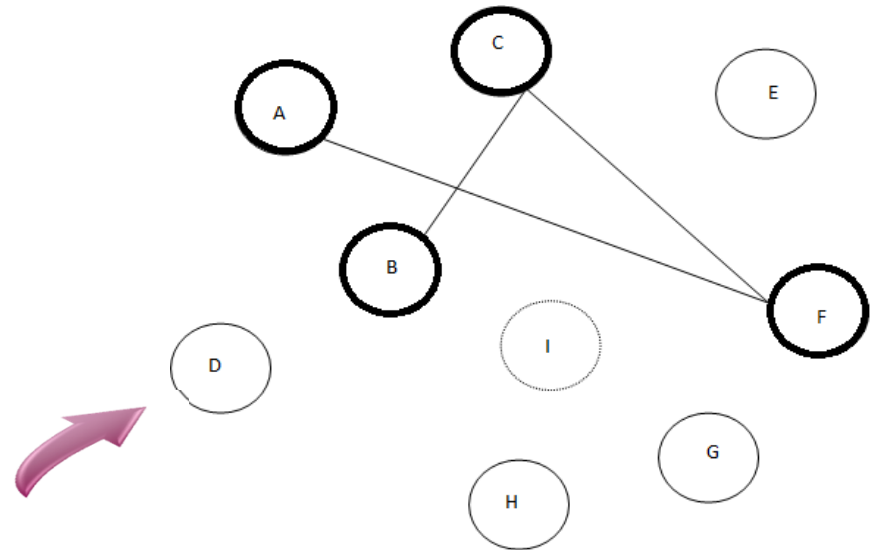
Clearly this is the set of edges connecting the vertices in core.

# Link Prediction Algorithm Description

Given the training set,  $G(V, E_{old})$  as in the figure below, we would like to predict the new edges among the vertices in core, in the test set.

- These new interactions are labeled  $E_{new}$ , given by  $E_{new} = V \times V - E_{old}$
- The test set contains all the vertices including a new vertex 'I'
- Once we have found a ranked list 'L', we pick the first 'n' pairs in the set 'core X core' where n is the count of  $E_{new}$ , given by  $|E_{new}|$
- The size of the intersection of this set with that of  $E_{new}$  is finally determined

Diagram depicting the test set and the newly predicted edges among the vertices A, B, C and F (core vertices).



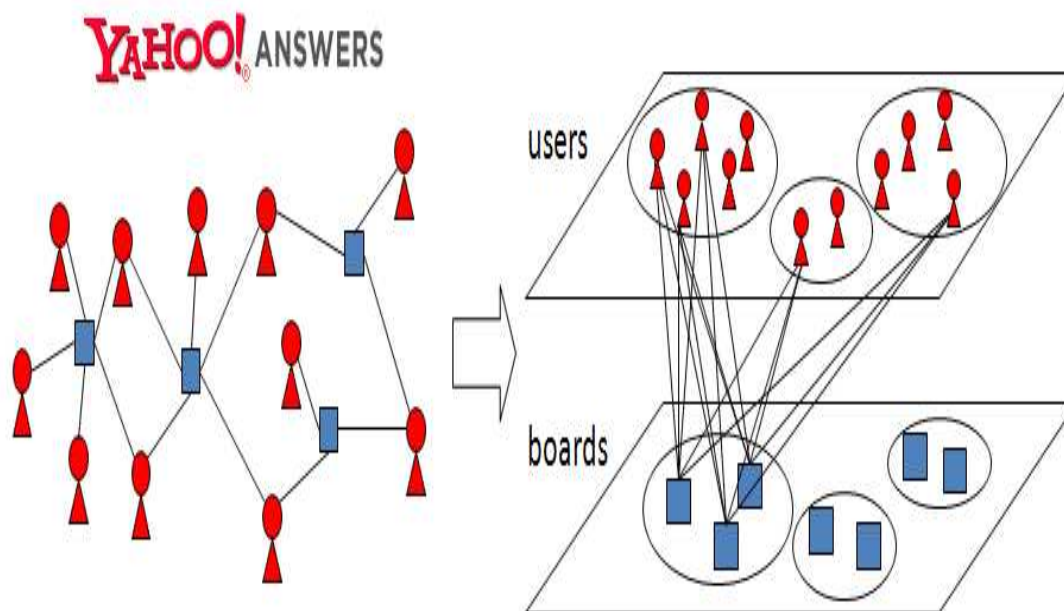
- We do not want to predict edges between vertices other than the ones in core.
- We would not want to predict the edges that are already present in the training set.

# Link Prediction Methods

In order for the proximity measures to make sense while estimating similarity among vertices, we will need to modify these measures.

We will consider such proximity measures under three different categories:

- Node Neighborhood Based Methods
  - Common neighbors
  - Jaccard's coefficient
  - Adamic-Adar
- All Paths Based Methodologies
  - PageRank
  - SimRank
- Higher Level Approaches
  - Unseen bigrams
  - Clustering





# Node Neighborhood Based Methods

1. Common neighbors
  2. Jaccard's coefficient
  3. Adamic-Adar
- 

## 1. Common neighbors

The common neighbors method is a simple measure that takes into account the intersection set of the neighbors of the vertices  $u$  and  $v$ .

This set would contain all the common neighbors of the two vertices. The value of  $\text{score}(u,v)$  will therefore be,

$$\text{score}(u,v) = |N(u) \cap N(v)|$$



- The conclusion is that a future interaction is strongly linked to all the above factors.
- Implementing such a measure can be very simple. We will need to collect the neighbors of  $u$ , the neighbors of  $v$  and compare them for matches.
- All matching vertices as designated as common neighbors.

# Node Neighborhood Based Methods

1. Common neighbors
2. Jaccard's coefficient
3. Adamic-Adar

## 2. Jaccard's coefficient

Jaccard's coefficient is a slightly complex proximity measure which is also based on the node neighborhood principle.

Mathematically the Jaccard coefficient for two sets A and B can be represented as a ratio of the intersection of the two sets to the union of the two sets,

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}$$



This version of the Jaccard coefficient would make sense only in case of multi-dimensional vector data.

To measure dissimilarity we would subtract  $J(A, B)$  from given values,

$A = (1, 0, 0, 0, 0, 0, 0, 0, 0)$  and  $B = (0, 0, 0, 0, 0, 0, 1, 0, 0, 1)$ , the  $J(A, B)$  can be calculated as 0 using:

$$\frac{f_{11}}{f_{01} + f_{10} + f_{11}}$$

where,

$f_{ij}$  is the frequency of simultaneous occurrence



For the vertices  $u$  and  $v$ , we modify the Jaccard coefficient and define it as follows for the link prediction problem,

$$score(x, y) = \frac{|N(u) \cap N(v)|}{|N(u) \cup N(v)|}$$

# Node Neighborhood Based Methods

1. Common neighbors
2. Jaccard's coefficient
3. Adamic-Adar

## 3. Adamic-Adar

Another measure based on common neighbors for measuring proximity is, Adamic-Adar.

This method computes the similarity between any two vertices  $u$  and  $v$  using a common feature of the two, named  $z$ . The similarity measure is then,

$$\sum_z \frac{1}{\log(\text{freq}(z))}$$

\*Where  $\text{freq}(z)$  is the frequency of occurrence of the common feature between  $u$  and  $v$ .

Using this measure we would then estimate the score as follows:

$$\text{score}(u, v) = \sum_{z \in N(u) \cap N(v)} \frac{1}{\log(N(z))}$$

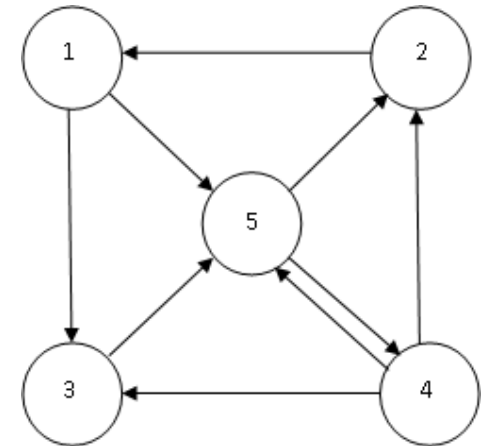
# All Paths Based Methodologies

1. PageRank
  2. SimRank
- 

## 1. PageRank

PageRank is one of the algorithms that aims to perform object ranking. The assumption PageRank makes is that a user starts a random walk by opening a page and then clicking on a link on that page.

[PageRank has been discussed before]



The mathematical formulation of PageRank also takes into account the user getting bored of a browsing session, and hence beginning another random walk on the graph  $G$ .

# All Paths Based Methodologies

1. PageRank

2. SimRank

---

## Challenges and issues involved

It is a challenge to rank web pages in order of their significance, both overall as well as pertaining to a particular query.

There are many aspects of a webpage that make it relevant such as :

- Web page changes and the frequency of this change.
- Keyword changes and keyword count changes.
- Number of new backlinks.
- Data availability and stability.

# All Paths Based Methodologies

1. PageRank

2. SimRank

## 2. SimRank

- Simrank is a link analysis algorithm that works on a graph 'G' to measure the similarity between two vertices u and v in the graph.
- For the nodes u and v, it is denoted by  $s(u,v) \in [0,1]$ . If  $u=v$  then,  $s(u,v)=1$
- The definition iterates on the similarity index of the neighbors of u and v itself.

$$s(u, v) = \frac{C}{|N(u)||N(v)|} \sum_{i=1}^{|N(u)|} \sum_{j=1}^{|N(v)|} s(N(i)u, N(j)v)$$

\*where C is a constant and  $C \in [0,1]$

- We have to calculate the score for this measure using this value of  $s(u,v)$ .
- Using Simrank, the  $\text{score}(u,v)$  is the same as  $s(u,v)$ .

$$\text{score}(u, v) = s(u, v)$$

# Higher level methodologies

## 1. Unseen Bigrams

## 2. Clustering

### 1. Unseen Bigrams

A bigram is any two letter or two word group, and a specific instance on an N-gram.

Some common examples from the English language are TH, AN, IN etc.

If such a bigram is not present in the training set but is found to be present in the test set, it is termed an unseen bigram.

$$score_{unweighted}^*(x, y) = |z : z \in N(y) \cap S_{\delta}^x|$$

where,  $z$  is a vertex similar to  $x$

Weighted score for the same is calculated as follows :

$$score_{weighted}^*(x, y) = \sum_{z \in N(y) \cap S_{\delta}^x} score(x, z)$$

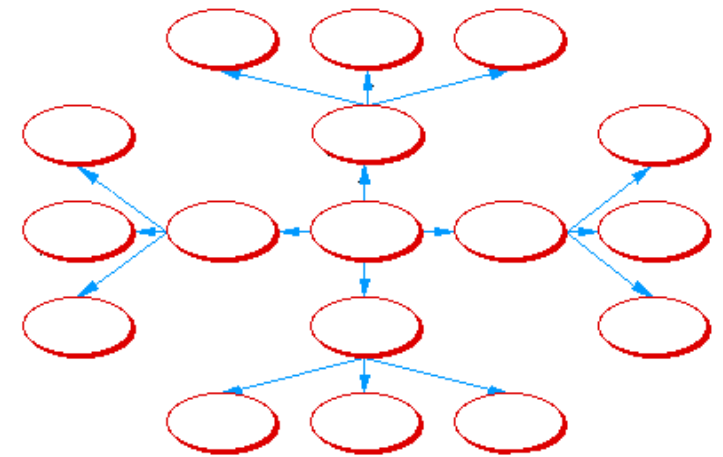
Once we have the  $score(x, y)$  using any of the methods we already detailed, we look at other nodes that are similar to 'x'. Consider 's' to be the set of nodes that are similar to 'x', if we use  $S_{\delta}^x$  to depict ' $\delta$ ' similar nodes to 'x', where  $\delta \in \mathbb{Z}^+$ .

# Higher level methodologies

1. Unseen Bigrams
2. Clustering

## 2. Clustering

- Getting rid of edges that are tentative and vague is one way of making sure prediction accuracy increases.
- If link prediction is attempted on such a graph containing only edges that are appropriate to the prediction process, we can be assured of better results.



Source: [www.sdcoe.k12.ca.us/score/actbank/tcluster.htm](http://www.sdcoe.k12.ca.us/score/actbank/tcluster.htm)

**Jon Kleinberg** et.al. suggest that in order to calculate the  $\text{score}(x,y)$ , we can initially find the  $\text{score}(u,v)$ ,

where ;

$u, v \in E_{\text{old}}$

NOWELL, D. L., AND KLEINBERG, J. The link prediction problem for social networks. In CIKM '03: Proceedings of the twelfth international conference on Information and knowledge management (New York, NY, USA, 2003), ACM, pp. 556–559.

From this list we then remove  $(1-p)$  edges, where the calculated score is found to be low.

This way we arrive at a subgraph lacking edges that are not of much interest to the prediction process.

$\text{Score}(x,y)$  must then be calculated on the new subgraph that we recently formed.

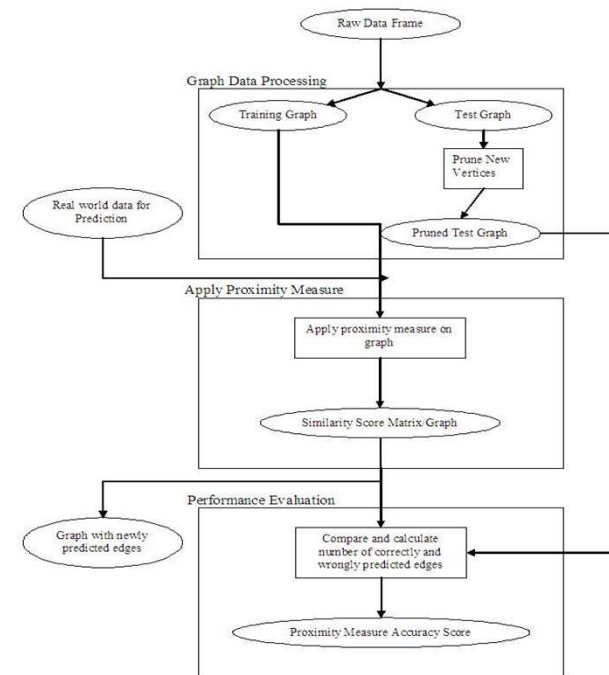


# Link Prediction Algorithm

The diagram gives a high level overview of the link prediction process consisting of three major steps :

- Graph Data Processing
- Apply Proximity Measure
- Performance Evaluation

- Social network analysis [SNA] is the mapping and measuring of relationships between people, groups, organizations, computers, and other connected entities.
- The nodes in the network are the people and groups while the links show relationships or flow between the nodes.
- Also, SNA provides both a visual and a mathematical analysis of human relationships.

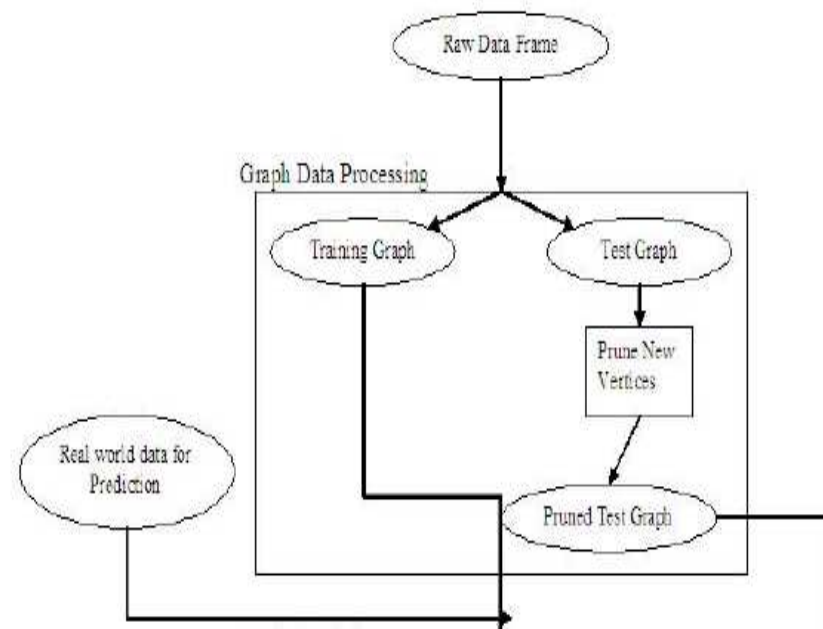


# Link Prediction Algorithm

## Graph Data Processing

The Graph Data Processing step is the first of the three steps in link prediction, in which, the input graph is processed. The raw data in the form of adjacency lists or adjacency matrices are split into training and test set graphs.

- Accept raw data representation of a collaboration or co-authorship network, in the form of an edge list and a year attribute for each edge at the least.
- Split this data into training and test sets.
- For maximum accuracy, the prediction process should depend only on attributes intrinsic to the network. Hence, the newer vertices in test graph not in training graph are pruned.
- The pruned test graph may still contain newer edges not present in the training graph. These are the edges we seek to predict.



# Link Prediction Algorithm

## Graph Data Processing

R code to perform the initial data processing of the graph is detailed below.

- Create data frame from given file
- Get year range
- Based on test duration given ,split data into training and test sets For maximum accuracy, the prediction process should depend only on attributes
- Convert data frames into graphs

```
rawdataframe <- read.table(path, sep = " ",  
> fill = TRUE)
```

```
begin_year <- min(rawdataframe$V3)  
end_year <- max(rawdataframe$V3)
```

```
trainingdataframe <- rawdataframe[!rawdataframe$V3  
%in% c((end_year-testduration+1):end_year),]  
testdataframe <- rawdataframe[rawdataframe$V3  
%in% c((end_year-testduration+1):end_year),]
```

```
rawgraphdata <- graph.data.frame(rawdataframe,  
directed=isDirected, vertices = NULL)  
traininggraphdata <- graph.data.frame(trainingdataframe,
```

# Link Prediction Algorithm

## Graph Data Processing

---

Graph data processing R code continued.

- Convert data frames into graphs
- Remove newly added vertices and edges from test graph
- Return the created graphs

```
directed=isDirected, vertices = NULL)  
testgraphdata <- graph.data.frame(testdataframe,  
directed=isDirected, vertices = NULL)
```

```
testgraphdata_cleaned <- delete.vertices(testgraphdata,  
V(testgraphdata)[!V(testgraphdata)$name  
%in% V(traininggraphdata)$name])
```

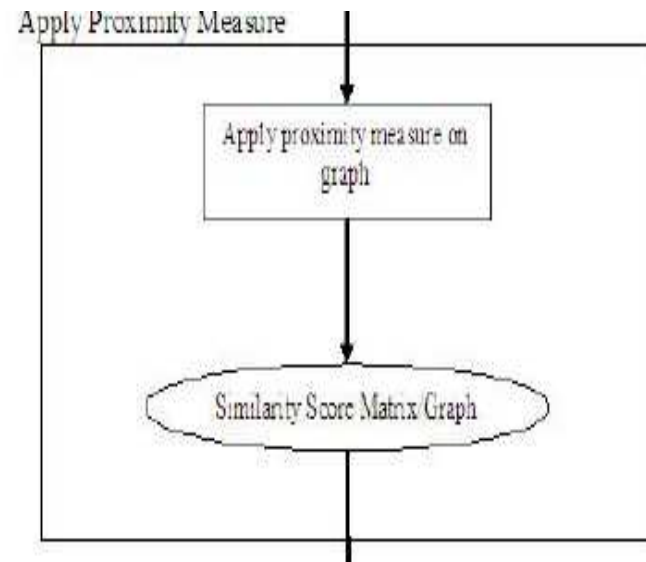
```
return(graphlist)
```

# Link Prediction Algorithm

## Apply Proximity Measures

In this step, the proximity measures are applied on the processed graph data. The proximity measures compute the proximity measures between a pair of vertices and the output of this application is the similarity score matrix.

- Using a graph object as input, compute the score of all possible edges using the proximity measures.
- The input to this section of the algorithm can also be the training graph generated in the graph data processing step.
- Select the proximity values above the threshold and return the edges associated with these values as a graph.



# Link Prediction Algorithm

## Apply Proximity Measures

Proximity measure application on the processed graph data is broken into 5 simple steps and the corresponding R code is explained here.

- Compute pair wise link prediction values
- Select links with predicted value above threshold
- Prevent Self-links
- Convert TRUEs to 1s
- Return predicted edges

```
predval <- measure(g)
```

```
adjmatrix <- (predval >= threshold)
```

```
diag(adjmatrix) <- FALSE
```

```
adjmatrix[adjmatrix == TRUE] <- 1
```

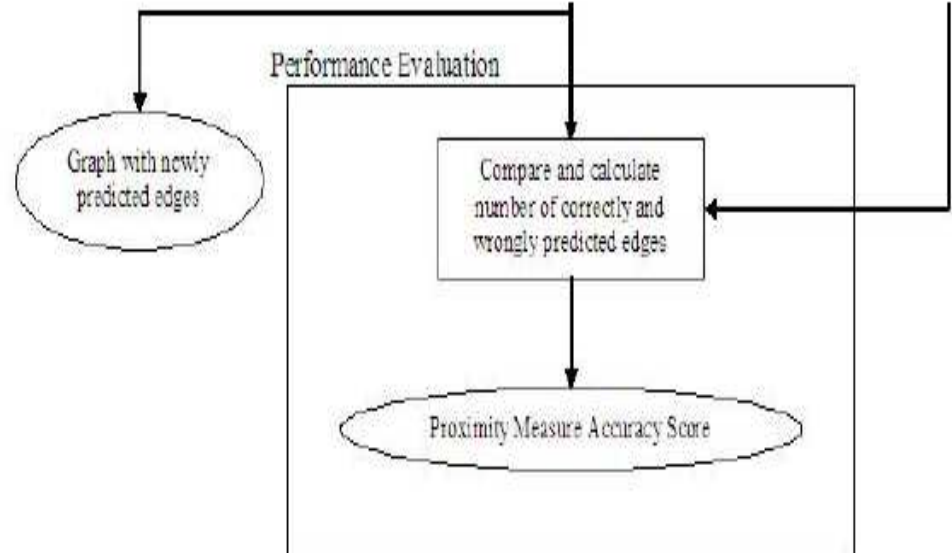
```
return(graph.adjacency(adjmatrix))
```

# Link Prediction Algorithm

## Performance Evaluation

Once proximity measures have been computed, new probable links are predicted. This is then evaluated against the originally predicted links in the test graph and various parameters like True, False positives and True, False negatives are calculated.

- This section is useful only when test data is available.
- Check how many links in the test graph were predicted accurately.
- Compute TP, FP, TN and FN.





# Link Prediction Algorithm

## Performance Evaluation

The code below illustrates the step by step process in R to perform the performance evaluation of the prediction process.

- Compare adjacency matrices row by row
- Compute the values of true and false positives and true and false negatives
- Compute the number of correctly predicted edge

```
testmatrix <- get.adjacency(testgraph)  
predictedmatrix <- get.adjacency(predictedgraph)
```

```
tp <- ((predictedmatrix == 1)  
& (predictedmatrix == testmatrix) )  
tn <- ((predictedmatrix == 0)  
& (predictedmatrix == testmatrix) )  
fp <- ((predictedmatrix == 1)  
& (predictedmatrix != testmatrix) )  
fn <- ((predictedmatrix == 0)  
& (predictedmatrix != testmatrix) )
```

```
Outputlist <- list(truepositive = tp,  
trueneegative = tn, falsepositive = fp, falsenegative = fn)
```