Introduction to Data Science

Dr. Irfan Yousuf
Department of Computer Science (New Campus)
UET, Lahore

(Week 16; May 06 – 10, 2024)

Outline

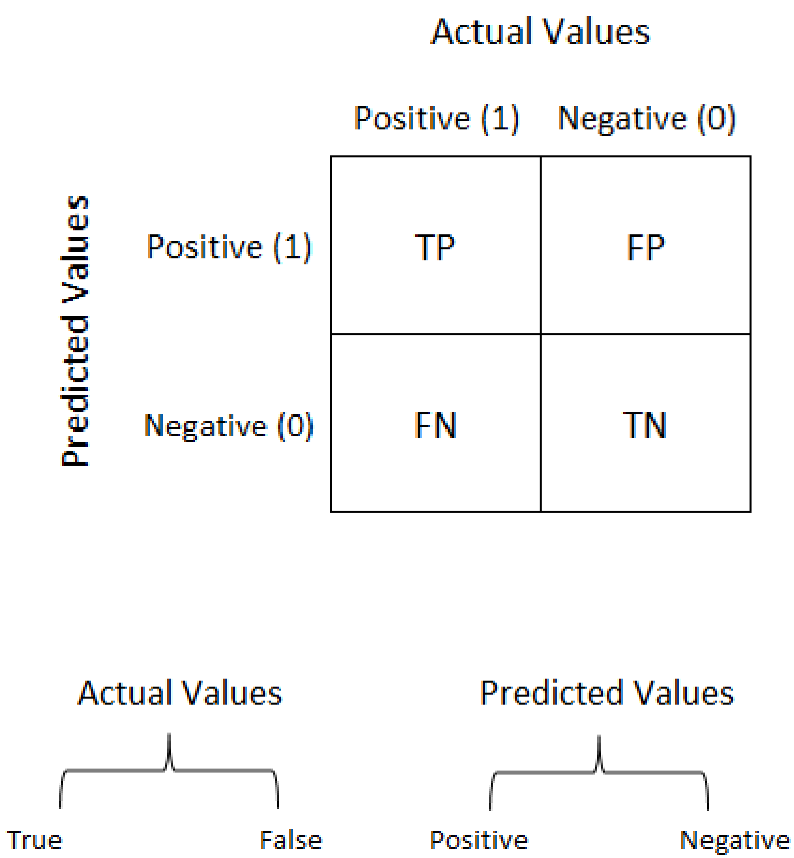
- Confusion Matrix for Multiclassification
- Graph Data Science

Types of Classification

- Binary Classification: It is a process or task of classification, in which a given data is classified into two classes. It's basically a kind of prediction about which of two groups the thing belongs to.

- Multi-class Classification: Multi-class classification is the task of classifying data into more than two classes.

Confusion Matrix



True Positive (TP): You predicted positive and it’s true.

True Negative (TN): You predicted negative and it’s true.

False Positive (FP): You predicted positive and it’s false.

False Negative (FN): You predicted negative and it’s false.

Actual Labels

Predicted Labels	Test Results	Person has Coronavirus: Yes	Person has Coronavirus: No
	Positive	**True Positive (TP):** Person with coronavirus tested positive	**False Positive (FP):** Person without coronavirus tested positive
	Negative	**False Negative (FN):** Person with coronavirus tested negative	**True Negative (TN):** Person without coronavirus tested negative

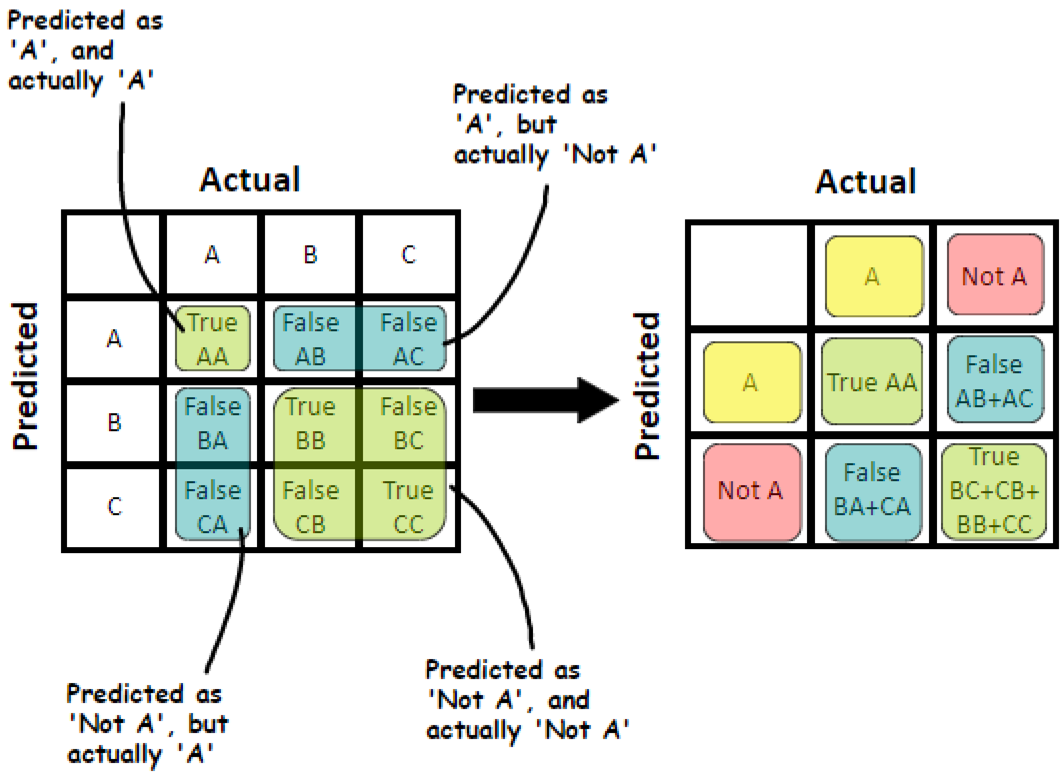
Confusion Matrix for Multi-class classification

		True Class		
		Apple	Orange	Mango
Predicted Class	Apple	7	8	9
	Orange	1	2	3
	Mango	3	2	1

		Actual Classes			
		a	b	c	d
Predicted Classes	a	50	3	0	0
	b	26	8	0	1
	c	20	2	4	0
	d	12	0	0	1

Confusion Matrix for multiclassification

We can separate each class in a single confusion matrix to make calculations and visualizations easier



Confusion Matrix for multiclassification

```python
# Classes
C = "Cat"
D = "Dog"
W = "Wolf"
T = "Tiger"

# True values
Y_true = [C,C,T,D,D,C,W,T,C,D,W,C,C,T,D,C,W,T,D,T,W,D,C,W,T,W,D,C,D,T,D,W,W,T,C,D,T]

# Predicted values
y_pred = [C,C,C,D,W,T,D,T,C,D,D,T,C,C,D,T,W,T,C,D,D,C,W,C,D,D,W,C,D,D,W,T,W,C,C,W,T]
```

```python
# Print the confusion matrix
print('The Confusion matrix is: \n', metrics.confusion_matrix(Y_true, y_pred))

print('    -----------------------------------------------------    ')

# Print the precision and recall, among other metrics
print(metrics.classification_report(Y_true, y_pred, digits=3))
```

Confusion Matrix for multiclassification

```
The Confusion matrix is:
 [[6 0 3 1]
 [2 4 0 4]
 [3 3 3 0]
 [1 4 1 2]]
    -----------------------------------------------------
              precision    recall  f1-score   support

         Cat      0.500     0.600     0.545        10
         Dog      0.364     0.400     0.381        10
       Tiger      0.429     0.333     0.375         9
        Wolf      0.286     0.250     0.267         8

    accuracy                          0.405        37
   macro avg      0.394     0.396     0.392        37
weighted avg      0.399     0.405     0.399        37
```

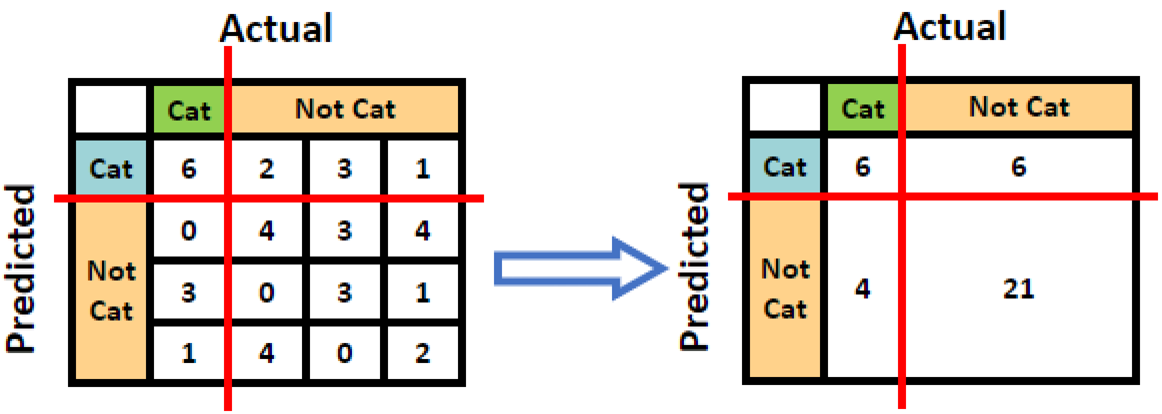
Actual

Predicted	Cat	Dog	Tiger	Wolf
Cat	6	2	3	1
Dog	0	4	3	4
Tiger	3	0	3	1
Wolf	1	4	0	2

Macro-average: Computes metrics independently for each class and then averages them. Each class is treated equally.

Micro-average: Aggregates the contributions of all classes to compute the average metric. It gives equal weight to each instance.

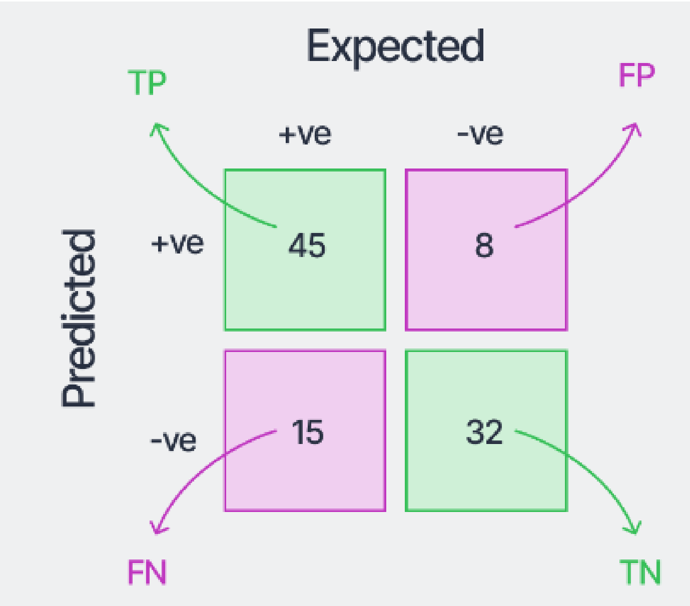
Confusion Matrix for multiclassification



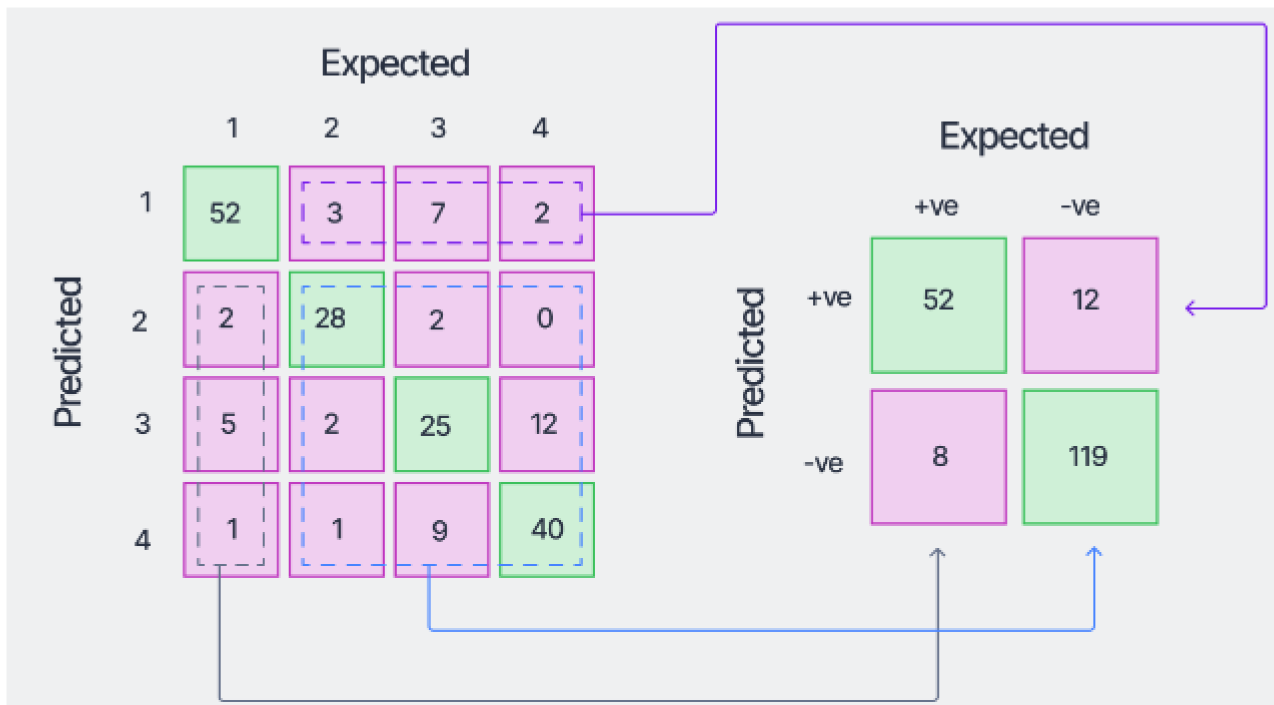
Recall = TP/(TP+FN) = 6/(6+4) = 0.6 → **Out of 10 actual cats, the model captured 6 cats correctly.**

Precision = TP/(TP+FP) = 6/(6+6) = 0.5 → **Out of 12 captured cats, there are 6 actual cats.**

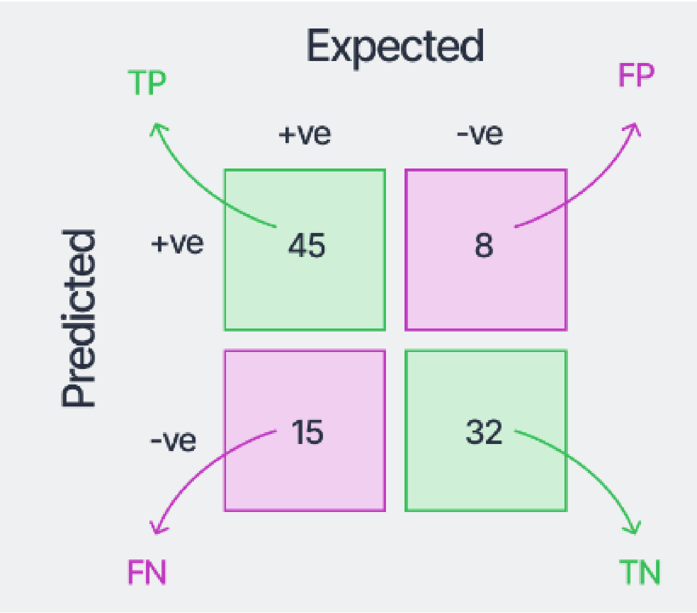
The F1-Score = 2 * (Precision * Recall)/(Precision + Recall) = 0.545.

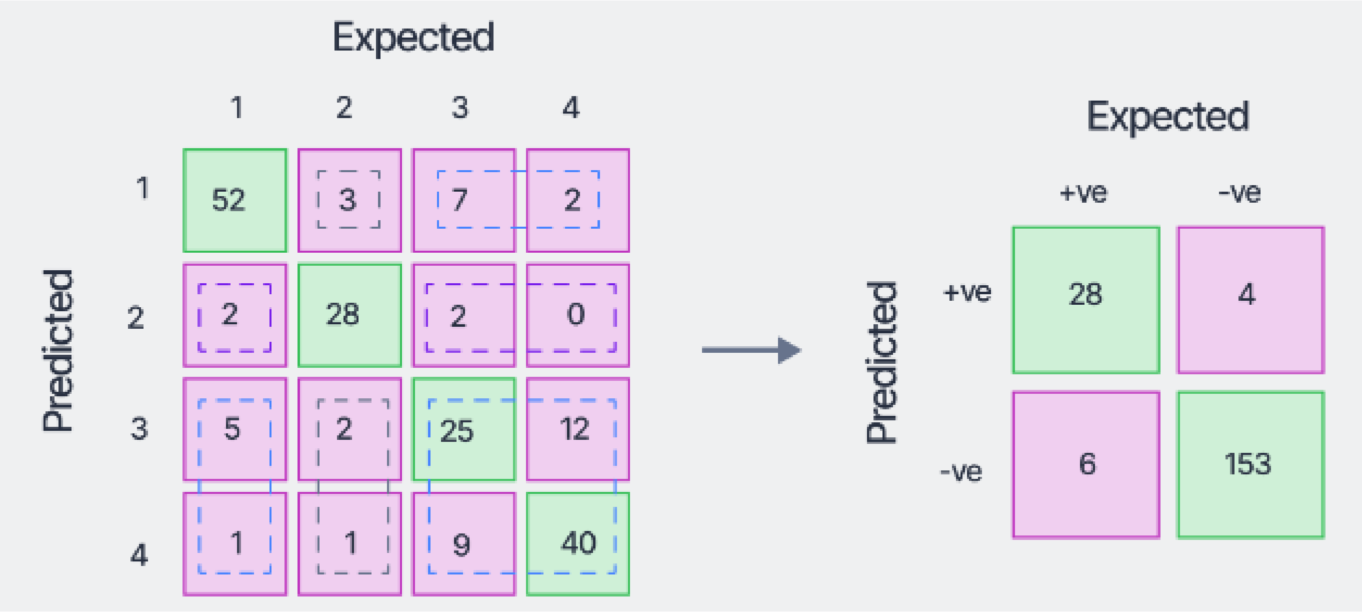


Expected
TP
FP
+ve
-ve
Predicted
+ve
45
8
-ve
15
32
FN
TN



Expected
1
2
3
4
Predicted
1
52
3
7
2
2
2
28
2
0
3
5
2
25
12
4
1
1
9
40
Expected
+ve
-ve
Predicted
+ve
52
12
-ve
8
119


Expected
TP
FP
+ve
-ve
Predicted
+ve
45
8
-ve
15
32
FN
TN


Expected
1
2
3
4
Predicted
1
52
3
7
2
2
2
28
2
0
3
5
2
25
12
4
1
1
9
40
Expected
+ve
-ve
Predicted
+ve
28
4
-ve
6
153

Expected

Predicted	1	2	3	4
1	52	3	7	2
2	2	28	2	0
3	5	2	25	12
4	1	1	9	40

Class	Precision (%)	Recall (%)	F1-Score (%)
1	81. 25	86. 67	83. 87
2	87. 50	82. 35	84. 85
3	56. 82	58. 14	57. 47
4	78. 43	74. 07	76. 19

Graph Data Science

Graph Data Science

- A branch of data science that focuses on the analysis of data presented as graphs.

- Numerous types of data, such as social networks, transportation networks, biological networks, and others, can be represented using graphs.

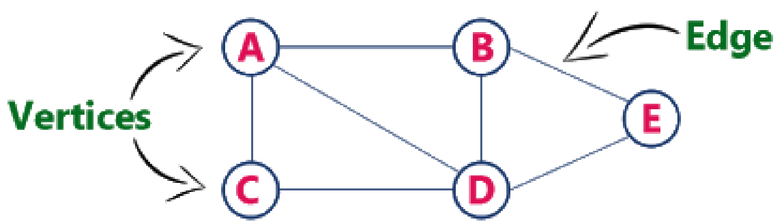
- To glean insights from such data, graph algorithms and machine learning methods are used in graph data science.

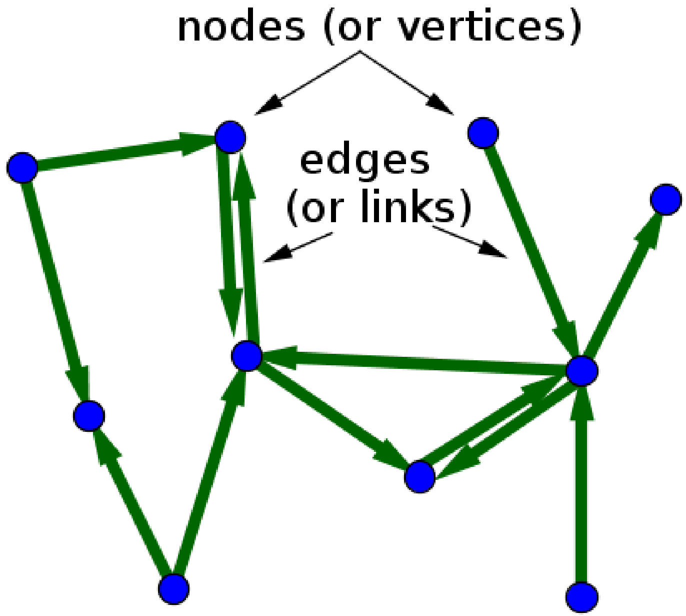
Graph

- A graph is a common data structure that consists of a finite set of nodes (or vertices) and a finite set of edges (or links) connecting them.
- A pair (x,y) is referred to as an edge, which communicates that the x vertex connects to the y vertex.
- Graphs are used to solve real-life problems that involve representation of the problem space as a network.
- For example, a single user in Facebook can be represented as a node (vertex) while their connection with others can be represented as an edge between nodes.

Types of Graphs

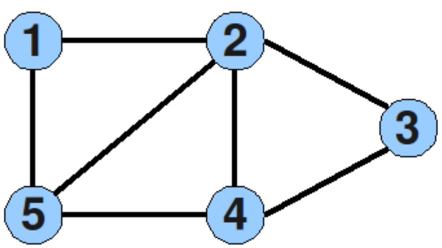
- **Undirected Graphs:** In an undirected graph, nodes are connected by edges that are all bidirectional. For example, if an edge connects node 1 and 2, we can traverse from node 1 to node 2, and from node 2 to 1.
- **Directed Graphs:** In a directed graph, nodes are connected by directed edges – they only go in one direction.

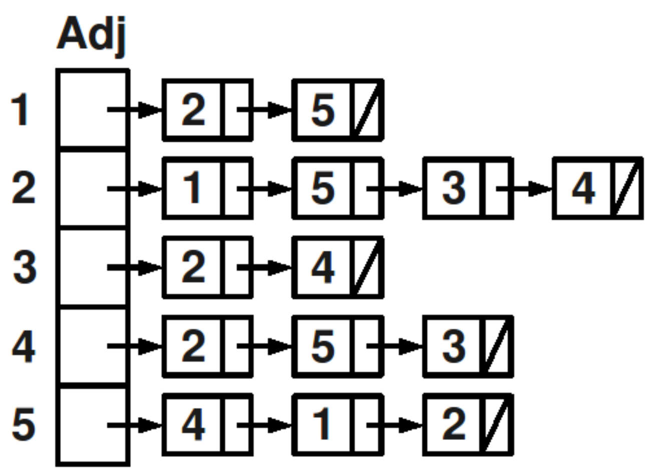




Representation of Graphs

- **Adjacency List:** To create an Adjacency list, an array of lists is used. The size of the array is equal to the number of nodes.
- A single index, array[i] represents the list of nodes adjacent to the i-th node.





1 2 5

2 1 5 3 4

3 2 4

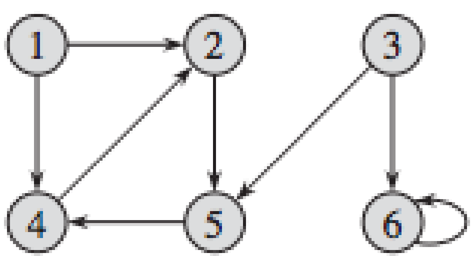
4 2 5 3

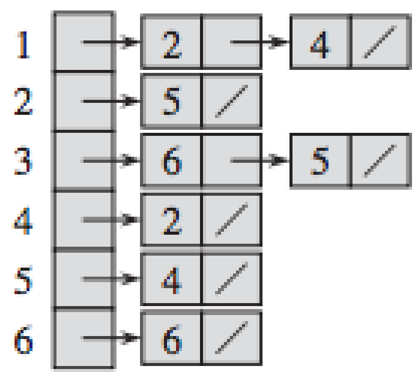
5 4 1 2

Undirected Graph

Representation of Graphs

- **Adjacency List:** To create an Adjacency list, an array of lists is used. The size of the array is equal to the number of nodes.
- A single index, array[i] represents the list of nodes adjacent to the ith node.





1 2 4

2 5

3 5 6

4 2

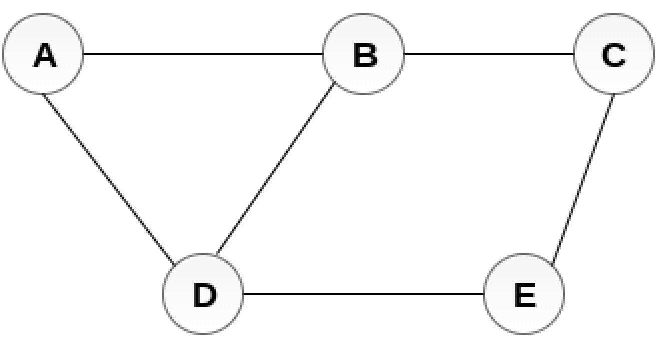
5 4

6 6

Directed Graph

Representation of Graphs

- **Adjacency Matrix:** An Adjacency Matrix is a 2D array of size V x V where V is the number of nodes in a graph.
- A slot matrix[i][j] = 1 indicates that there is an edge from node i to node j.



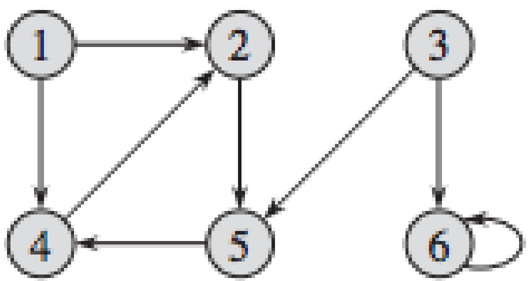
Undirected Graph

$$
\begin{array}{c c}
 & \begin{array}{ccccc} A & B & C & D & E \end{array} \\
\begin{array}{c} A \\ B \\ C \\ D \\ E \end{array} & \begin{bmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix}
\end{array}
$$

Adjacency Matrix

Representation of Graphs

- **Adjacency Matrix:** An Adjacency Matrix is a 2D array of size V x V where V is the number of nodes in a graph.
- A slot matrix[i][j] = 1 indicates that there is an edge from node i to node j.

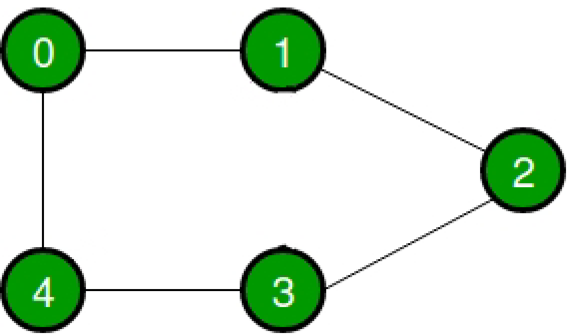


	1	2	3	4	5	6
1	0	1	0	1	0	0
2	0	0	0	0	1	0
3	0	0	0	0	1	1
4	0	1	0	0	0	0
5	0	0	0	1	0	0
6	0	0	0	0	0	1

Directed Graph

Representation of Graphs

- **Edge List:** A graph is represented as a list of edges.



Edge List

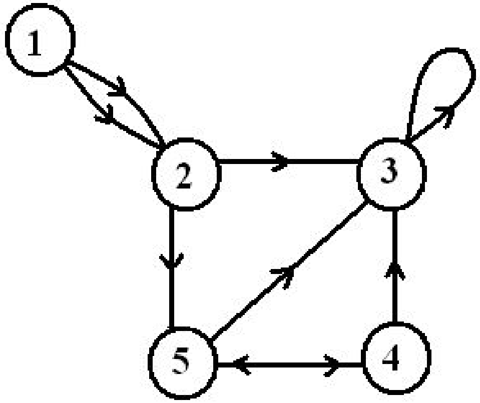
0 1

0 4

1 2

2 3

3 4



Edge List

1 2

1 2

2 3

2 5

3 3

4 3

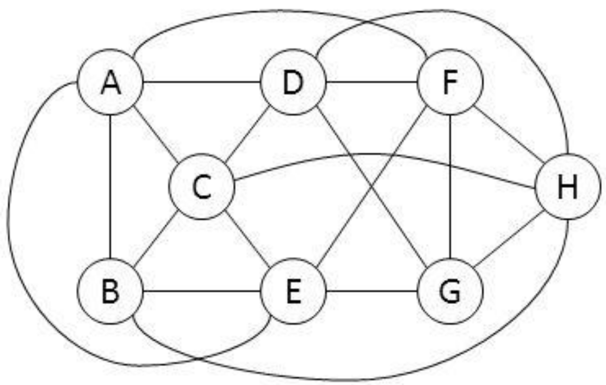
4 5

5 3

5 4

Representation of Graphs

Graph Representations



node list

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

edge list

AB
AC
AD
AE
AF
BA
BC
BE
BH
CA
CB
CD
CE
CH
DA
DC
DF
DG
DH
EA
EB
EC
EF
EG
FA
FD
FE
FG
FH
GD
GE
GF
GH
HB
HC
HD
HF
HG

adjacency matrix

	A	B	C	D	E	F	G	H
A	–	1	1	1	1	1	0	0
B	1	–	1	0	1	0	0	1
C	1	1	–	1	1	0	0	1
D	1	0	1	–	0	1	1	1
E	1	1	1	0	–	1	1	0
F	1	0	0	1	1	–	1	1
G	0	0	0	1	1	1	–	1
H	0	1	1	1	0	1	1	–

node list - lists the nodes connected to each node

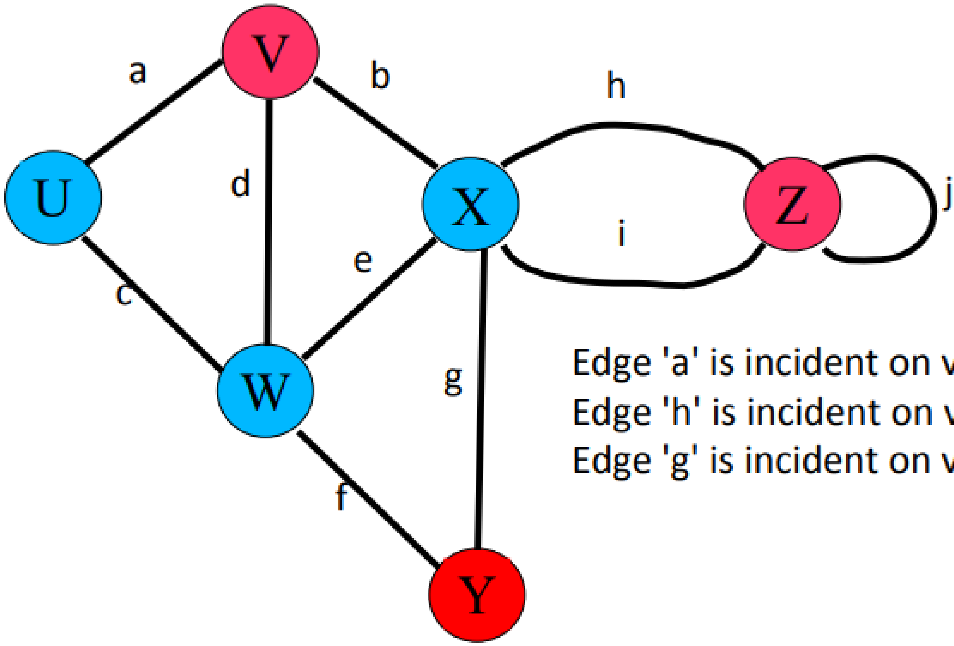
edge list - lists each of the edges as a pair of nodes
undirected edges may be listed twice XY and YX
in order to simplify algorithm implementation

adjacency matrix - for an n-node graph we build an
nxn array with 1's indicating edges and 0's no edge
the main diagonal of the matrix is unused unless a
node has an edge connected to itself. If graph is
weighted, 1's are replaced with edge weight values

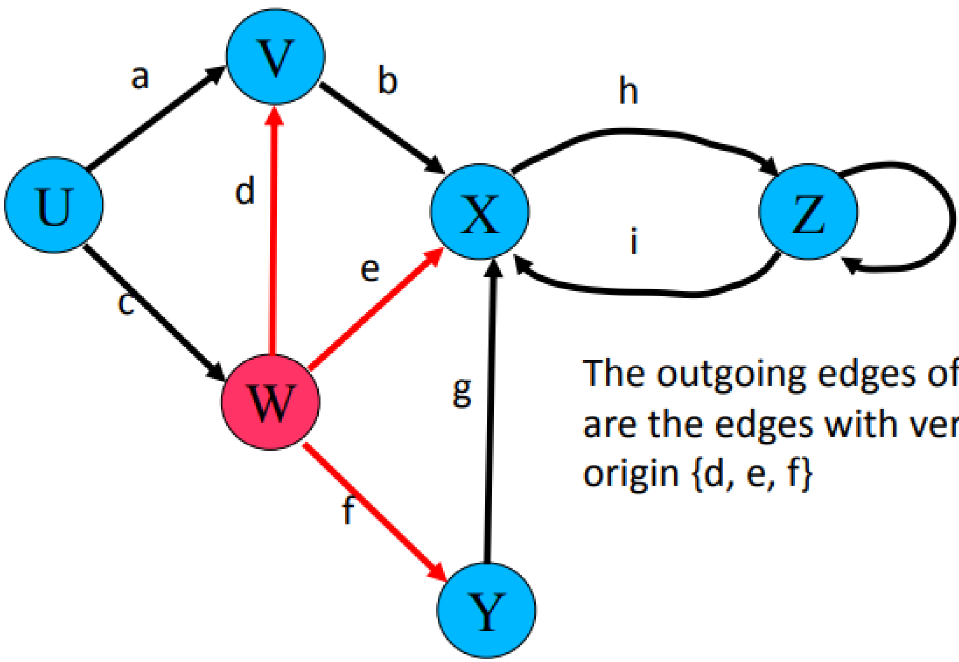
Graph Terminology

- Two vertices joined by an edge are called the end vertices or endpoints of the edge.
- If an edge is directed its first endpoint is called the origin and the other is called the destination.
- Two vertices are said to be adjacent if they are endpoints of the same edge.
- An edge is said to be incident on a vertex if the vertex is one of the edges endpoints.
- The outgoing edges of a vertex are the directed edges whose origin is that vertex.
- The incoming edges of a vertex are the directed edges whose destination is that vertex

Graph Terminology



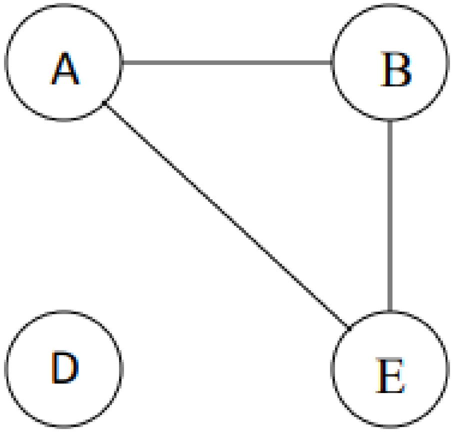
Edge 'a' is incident on vertex V
Edge 'h' is incident on vertex Z
Edge 'g' is incident on vertex Y



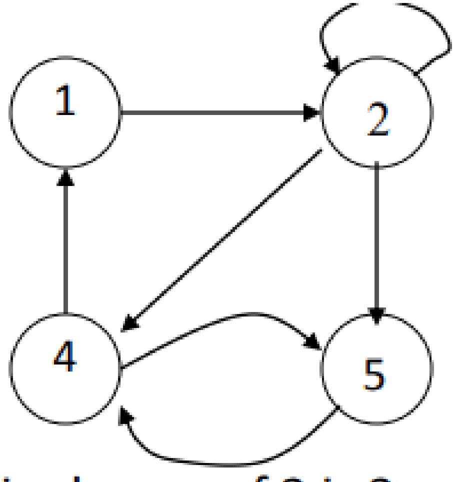
The outgoing edges of vertex W are the edges with vertex W as origin {d, e, f}

Graph Terminology

Degree: Number of edges incident on a node
In-degree: Number of edges entering a node
Out-degree: Number of edges leaving a node
Degree = In-degree + Out-degree



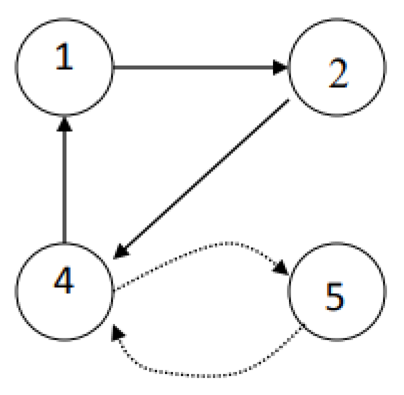
The degree of B is 2.



The in degree of 2 is 2 and
the out degree of 2 is 3.

Graph Terminology

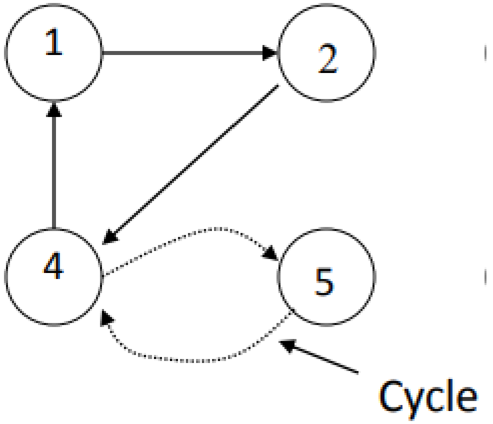
- **Path:** A path is a sequence of vertices such that there is an edge from each vertex to its successor.
- A path is simple if each vertex is distinct.
- A circuit is a path in which the terminal vertex coincides with the initial vertex



Simple path: [1, 2, 4, 5]
Path: [1, 2, 4, 5, 4]
Circuit: [1, 2, 4, 5, 4, 1]

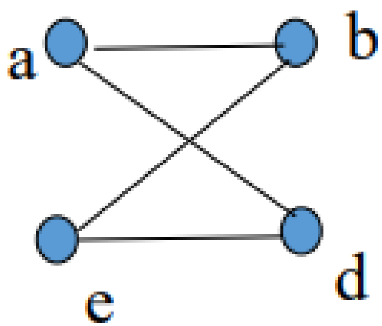
Graph Terminology

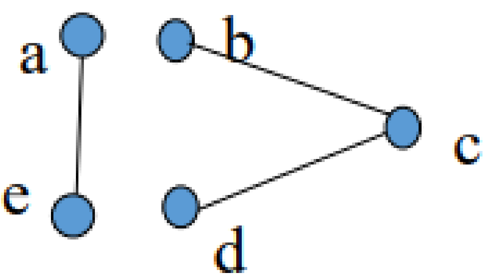
- A path from a vertex to itself is called a cycle.
- A graph is called cyclic if it contains a cycle;
  - otherwise, it is called acyclic



Graph Terminology

- **Connected :** There exists at least one path between any two vertices.
- **Disconnected :** Otherwise





Graph Terminology

- **Clustering coefficient** is a measure of the degree to which nodes in a graph tend to cluster together.

Clustering coefficient C_i for each node i is

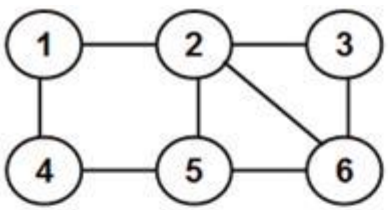
$$C_i = \frac{2n_i}{k_i(k_i - 1)}$$

k_i Degree of node i

n_i is the number of edges among neighbors of i

Average Clustering Coefficient:

$$\bar{C} = \frac{1}{n}\sum_{i=1}^{n} C_i$$



Vertex	Neighbors	# Links connecting the Neighbors	Max. possible # Links connecting the Neighbors	Local Clustering Coefficient
1	2, 4	0	2(1)/2 = 1	0/1 = 0.0
2	1, 3, 5, 6	2	4(3)/2 = 6	2/6 = 0.33
3	2, 6	1	2(1)/2 = 1	1/1 = 1.0
4	1, 5	0	2(1)/2 = 1	0/1 = 0.0
5	2, 4, 6	1	3(2)/2 = 3	1/3 = 0.33
6	2, 3, 5	2	3(2)/2 = 3	2/3 = 0.67

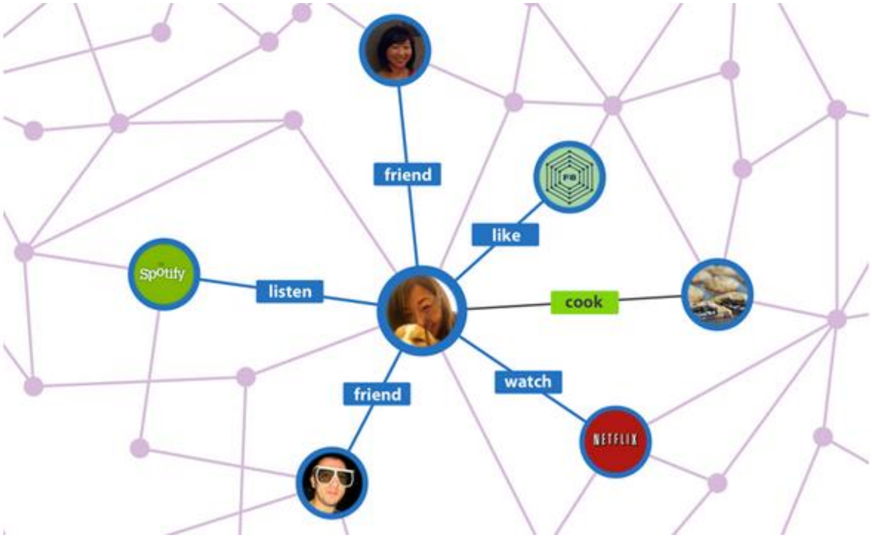
Networks

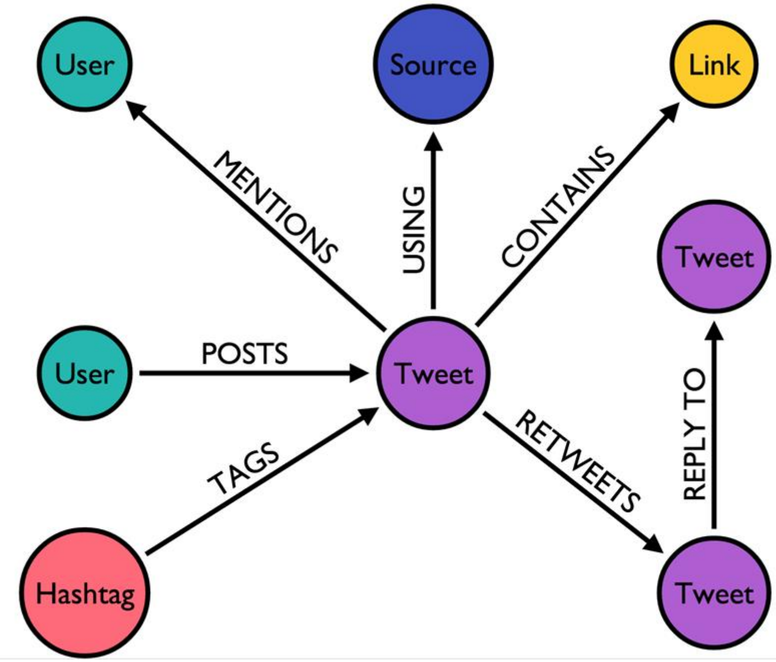
- **Network Analysis (NA)** is a set of integrated techniques to depict relations among actors and to analyze the social structures that emerge from the recurrence of these relations.

- **Network Science** is an academic field which studies complex networks such as telecommunication networks, computer networks, biological networks, cognitive and semantic networks, and social networks.

Network as a Graph

- We can represent **any network** in the form of a graph.
- Nodes become the users and an Edge between two nodes shows their relationship.





Social Networks as Graphs

- A social network graph is a graph where the **nodes represent people** and the lines between nodes, called edges, **represent social connections** between them, such as friendship or working together on a project.

Type	Edge
Friendship graph	Friendship between users
Interaction graph	Visible interaction, such as posting on a wall
Latent graph	Latent interaction, such as browsing profile
Following graph	Subscribe to receive all messages

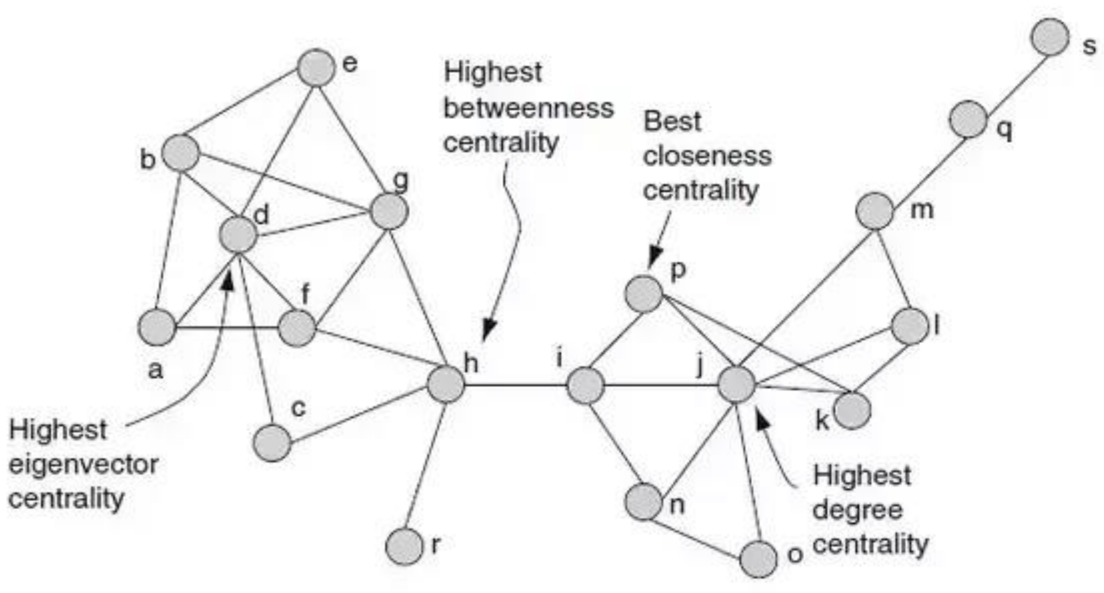
Social Network Analysis

- Social network analysis (SNA) is the process of **investigating social structures** through the use of network analysis and graph theory.

- It characterizes networked structures in terms of **nodes** (individual actors, people, or things within the network) and the ties, **edges**, or links (relationships or interactions) that connect them.

Centrality Measures in Social Network Analysis

- **Centrality Measures:** Centrality is a <u>collection of metrics</u> used to quantify how **important and influential** a specific node is to the network as a whole.
- It is important to remember that centrality measures are used **on specific nodes within the network**, and do not provide information on a network level.



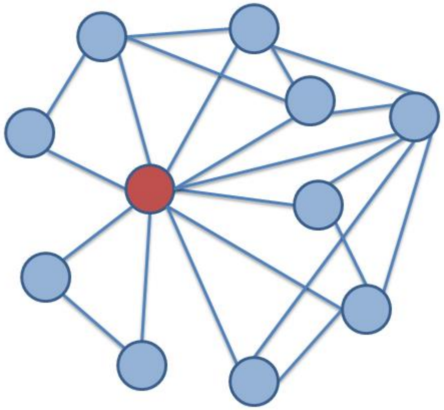
Centrality Measures in Social Network Analysis

- These algorithms / measures use graph theory to calculate the importance of any given node in a network.

- Each measure has its own definition of **‘importance’,** so you need to understand how they work to find the best one for your needs.
  - Degree Centrality
  - Betweenness Centrality
  - Closeness centrality
  - Eigen Centrality
  - PageRank Centrality

Degree Centrality

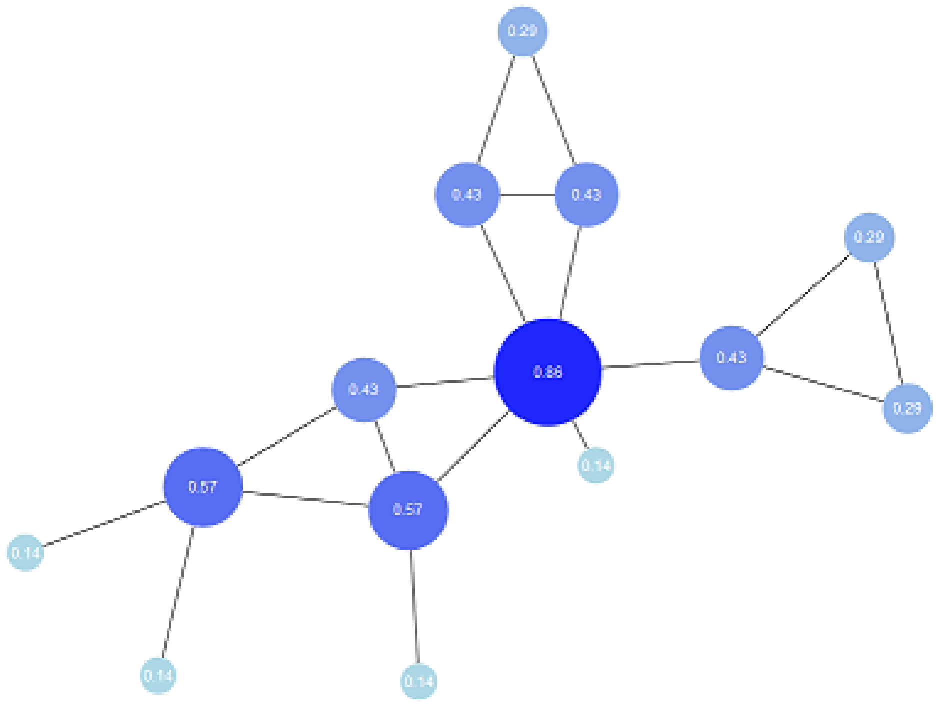
- **Degree centrality** assigns an importance score based simply on the number of links held by each node.
- What it tells us: How many direct, ‘one hop’ connections each node has to other nodes in the network.
- When to use it: For finding very **connected individuals**, **popular individuals**, individuals who are likely to hold most information or individuals who can quickly connect with the wider network.



Degree Centrality: Undirected Graphs

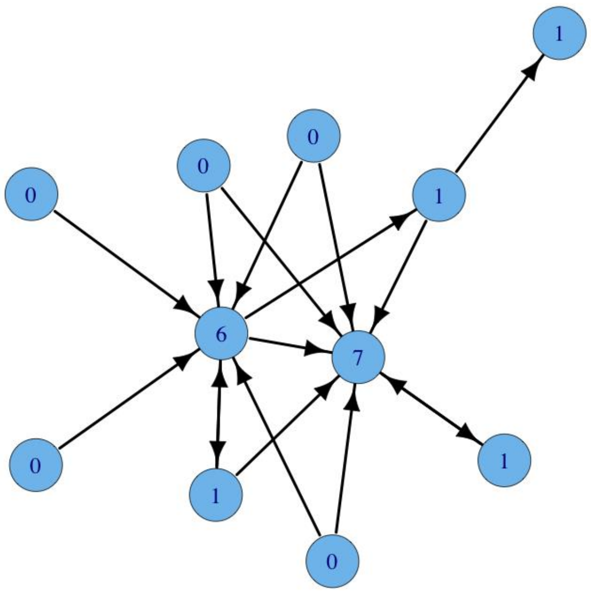
The degree centrality of a vertex v , for a given graph G:=(V,E) with |V| vertices and |E| edges, is defined as

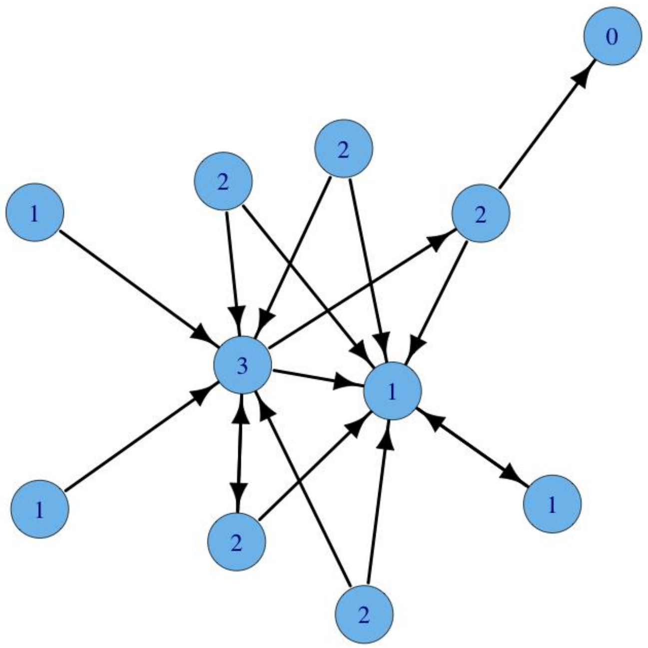
$$\mathbf{C_D(v) = deg(v)}$$



Degree Centrality: Directed Graphs

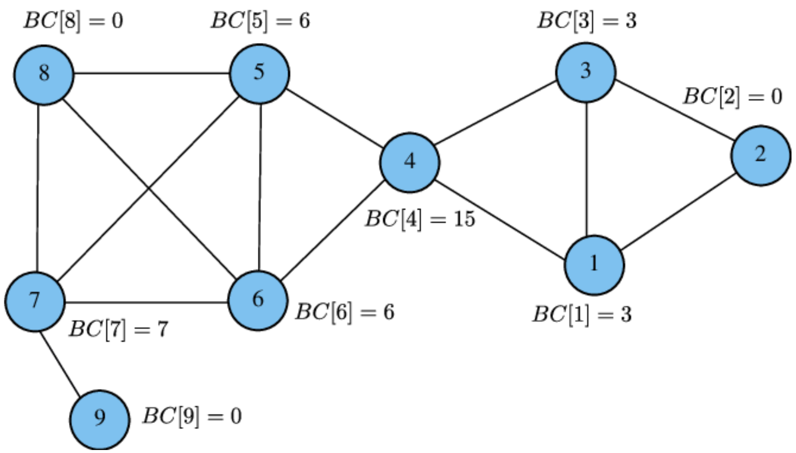
- The nodes with higher outdegree is more central (choices made).
- The nodes with higher indegree is more prestigious (choices received).





Betweenness Centrality

- **Betweenness centrality** measures the number of times a node lies on the shortest path between other nodes.
- What it tells us: This measure shows which nodes are '**bridges**' between nodes in a network. It does this by identifying all the shortest paths and then counting how many times each node falls on one.
- When to use it: For finding the individuals who influence the flow around a system.



Betweenness Centrality

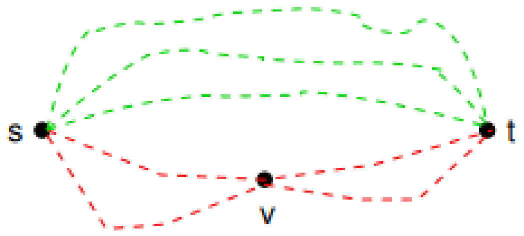
- The betweenness of a vertex v in a graph $G := (V, E)$ with V vertices is computed as follows:
  - For each pair of vertices (s, t), compute the shortest paths between them.
  - For each pair of vertices (s, t), determine the fraction of shortest paths that pass through the vertex in question (here, vertex v).
  - Sum this fraction over all pairs of vertices (s, t).
- More compactly the betweenness can be represented as:

$$Betwenness(v) = \sum_{s \neq v \neq t \in V} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

  - where σ_{st} is total number of shortest paths from node s to node t and $\sigma_{st}(v)$ is the number of those paths that pass through v.

Betweenness Centrality

Consider a node v and two other nodes s and t.

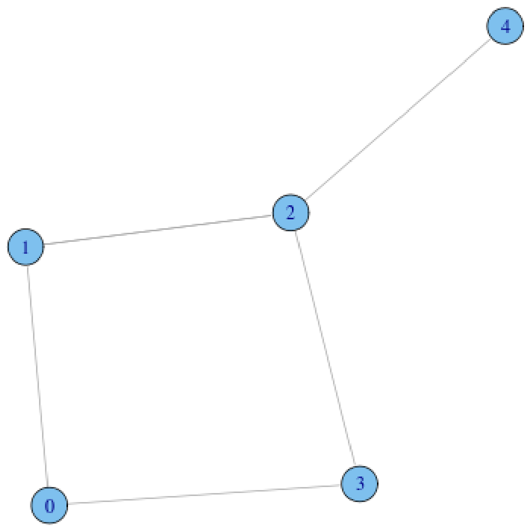


- Each shortest path between s and t shown in **green** **doesn't** pass through node v.
- Each shortest path between s and t shown in **red** passes through node v.

Consider the ratio $\dfrac{\sigma_{st}(v)}{\sigma_{st}}$:

- This gives the fraction of s-t shortest paths passing through v.
- The larger the ratio, the more important v is with respect to the pair of nodes s and t.
- To properly measure the importance of a node v, we need to consider all pairs of nodes (not involving v).

Betweenness Centrality

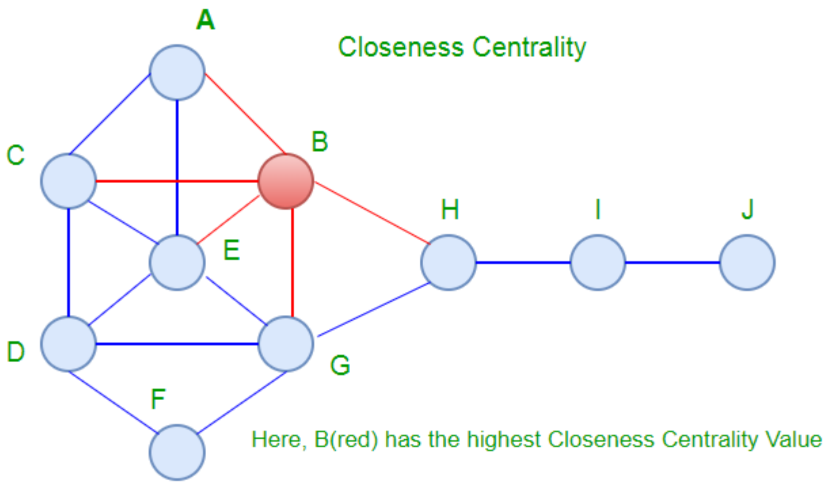


Find the betweenness centrality of node 2.

- The betweenness may be normalized by dividing through the number of pairs of vertices not including v, which for directed graphs is $(n-1)(n-2)$ and for undirected graphs is $(n-1)(n-2)/2$.

Closeness Centrality

- **Closeness centrality** scores each node based on their ‘closeness’ to all other nodes in the network.
- What it tells us: This measure calculates the shortest paths between all nodes, then assigns each node a score based on its sum of shortest paths.
- When to use it: For finding the individuals who are best placed to influence the entire network most quickly.



Closeness Centrality

- Closeness centrality measures **how short** the shortest paths are from node x to all nodes.
- It is usually expressed as the normalised inverse of the sum of the topological distances in the graph.

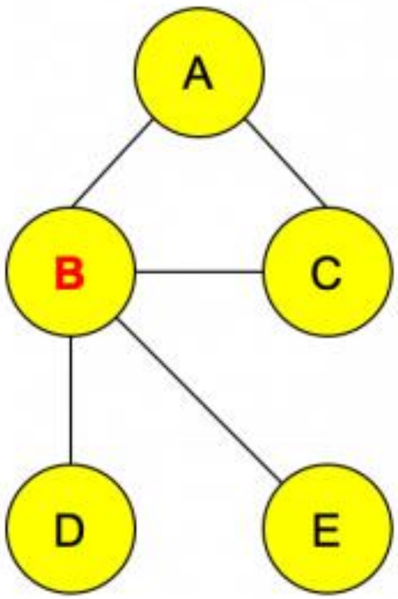
$$\textit{Closeness Centrality Score}(u) = \frac{\textit{number of nodes} - 1}{\sum (\textit{distance from u to all other nodes})}$$

$$CC(i) = \frac{N-1}{\sum_j d(i,j)}$$

Closeness Centrality

$$CC(i) = \frac{N-1}{\sum_j d(i,j)}$$

where
$i \neq j$,
d_{ij} is the length of the shortest path between nodes i and j in the network,
N is the number of nodes.



$$\begin{array}{c} \\ A \\ B \\ C \\ D \\ E \end{array}
\begin{array}{c} \begin{array}{ccccc} A & B & C & D & E \end{array} \\ \begin{pmatrix} 0 & 1 & 1 & 2 & 2 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 2 & 2 \\ 2 & 1 & 2 & 0 & 2 \\ 2 & 1 & 2 & 2 & 0 \end{pmatrix} \end{array}$$

farness $\sum_{j=1}^{n} d(i,j)$	$CC(i) = \frac{N-1}{\sum_j d(i,j)}$
6	(5-1)/6= 0.67
4	1.00
6	0.67
7	0.57
7	0.57

Summary

- Confusion Matrix
- Graph Data Science