Graph/Network Data

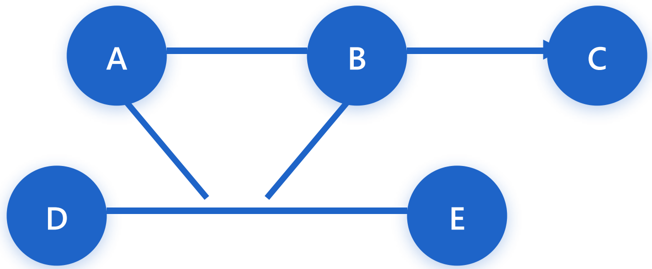
Represents relationships and connections between entities

Nodes (Vertices)
Represent objects or entities

Edges
Relationships between nodes

Directed	Undirected
Weighted	Unweighted

Example Graph Structure



Common Applications

🌐 Social Networks | 🧬 Molecular Structures | 🧠 Knowledge Graphs

⚠️ Non-Euclidean Structure → Requires Specialized Algorithms:

Graph Neural Networks (GNNs)

Graph Feature Extraction Process

📊 Step-by-Step Feature Extraction

1

Graph Representation

Convert graph to mathematical structures (adjacency matrix, edge list)

2

Feature Computation

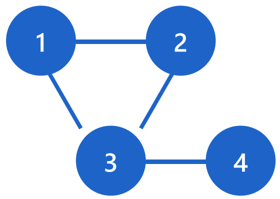
Calculate node-level and graph-level features

3

Feature Vector

Aggregate features into numerical vectors for ML models

Sample Graph



Adjacency Matrix

	1	2	3	4
1	0	1	1	0
2	1	0	0	1
3	1	0	0	1
4	0	1	1	0

Types of Graph Features

Node-Level Features

Degree: Number of connections (노드 1: degree = 2)

Centrality: Importance in the network

Clustering Coefficient: How connected neighbors are

PageRank: Node influence score

Edge-Level Features

Weight: Connection strength

Distance: Shortest path length

Common Neighbors: Shared connections

Graph-Level Features

Number of Nodes: Total vertices (예시: 4개)

Number of Edges: Total connections (예시: 4개)

Density: Connectivity ratio

Diameter: Maximum distance between nodes

Structural Features

Triangles: Number of 3-node cycles

Connected Components: Separate subgraphs

Average Path Length: Mean distance between nodes

💡 **Key Insight:** These features transform non-Euclidean graph data into numerical vectors that machine learning models can process!

Practical Example: Social Network Analysis

🧑‍🤝‍🧑 Friend Network Feature Extraction

```
# Python example using NetworkX import networkx as nx import numpy as np # 1. Create graph G = nx.Graph()
G.add_edges_from([(1,2), (1,3), (2,4), (3,4)]) # 2. Extract node features degrees = dict(G.degree()) #
{1:2, 2:2, 3:2, 4:2} centrality = nx.betweenness_centrality(G) clustering = nx.clustering(G) # 3. Extract
graph features num_nodes = G.number_of_nodes() # 4 num_edges = G.number_of_edges() # 4 density =
nx.density(G) # 0.667 avg_degree = np.mean(list(degrees.values())) # 2.0 # 4. Create feature vector for
Node 1 node_1_features = [ degrees[1], # degree: 2 centrality[1], # centrality: 0.0 clustering[1] #
clustering: 1.0 ] # Result: [2, 0.0, 1.0]
```

🧠 Graph Neural Network (GNN) Process

1

Initial Features

Each node starts with feature vector (e.g., [2, 0.0, 1.0])

2

Message Passing

Nodes aggregate features from neighbors

3

Feature Update

Neural network combines local + neighbor features

```
# GNN-style feature aggregation (simplified) def aggregate_features(node, neighbors, features): # Gather
neighbor features neighbor_features = [features[n] for n in neighbors] # Aggregate (e.g., mean pooling)
aggregated = np.mean(neighbor_features, axis=0) # Combine with node's own features updated =
neural_network([features[node], aggregated]) return updated # For Node 1 with neighbors [2, 3]: # Input:
own features + aggregated neighbor features # Output: enriched representation capturing local graph
structure
```

Feature Extraction Pipeline Summary

Raw Graph
Nodes + Edges

→

Mathematical Representation
Adjacency Matrix / Edge List

→

Feature Extraction
Degree, Centrality, etc.

→

ML-Ready Vectors
[2, 0.0, 1.0, ...]

🎯 Traditional ML Approach

1. Manual Feature Engineering
- Calculate statistical features
- Create handcrafted features
- Limited to known patterns

2. Fixed Representation
- Features don't adapt
- Same for all tasks

🚀 GNN Approach

1. Learned Representations
- Neural networks learn features
- Captures complex patterns
- Discovers hidden structures

2. Task-Specific
- Adapts to specific problems
- End-to-end learning

🎓 Key Takeaway

Graph feature extraction transforms complex relational data into numerical vectors that preserve structural information for machine learning tasks.

GNNs automate and optimize this process through learned representations!